NAG Fortran Library Manual
Mark 19

Volume 4

D04 – E04L

D04 – Numerical Differentiation
D05 – Integral Equations
E01 – Interpolation
E02 – Curve and Surface Fitting
E04L – Minimizing or Maximizing a Function (cont'd in Volume 5)

NAG®

NAG Fortran Library Manual, Mark 19

September 1999 ISBN 1-85206-169-3

NAG Ltd
Wilkinson House
Jordan Hill Road
Oxford
OX2 8DR
United Kingdom

Tel: +44 (0)1865 511245
Fax: +44 (0)1865 310139

Nihon NAG KK
Nagashima Building 2F
2-24-3 Higashi
Shibuya-ku
Tokyo
Japan

Tel: +81 (0)3 5485 2901
Fax: +81 (0)3 5485 2903

NAG GmbH
Schleißheimerstraße 5
85748 Garching
Deutschland

Tel: +49 (0)89 3207395
Fax: +49 (0)89 3207396

NAG Inc
1400 Opus Place, Suite 200
Downers Grove, IL 60515-5702
USA

Tel: +1 630 971 2337
Fax: +1 630 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

[*NP3390/19*]

Chapter D04 – Numerical Differentiation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D04AAF	5	Numerical differentiation, derivatives up to order 14, function of one real variable

Chapter D04

Numerical Differentiation

Contents

1 Scope of the Chapter

This chapter is concerned with calculating approximations to derivatives of a function f, where the user can supply a routine representing f.

2 Background to the Problems

2.1 Description of the Problem

The problem of numerical differentiation does not receive very much attention nowadays. Although the Taylor series plays a key role in much of classical analysis, the poor reputation enjoyed by numerical differentiation has led numerical analysts to construct techniques for most problems which avoid the explicit use of numerical differentiation.

One may briefly and roughly define the term numerical differentiation as any process in which numerical values of derivatives $f^{(s)}(x_0)$ are obtained from evaluations $f(x_i)$ of the function $f(x)$ at several abscissae x_i near x_0. This problem can be stable, well conditioned, and accurate when complex-valued abscissae are used. This was first pointed out by Lyness and Moler [1]. An item of numerical software for this appears in Lyness and Sande [2]. However, in many applications the use of complex-valued abscissae is either prohibitive or prohibited. The main difficulty in using real abscissae is that amplification of round-off error can completely obliterate meaningful results. In the days when one relied on hand calculating machines and tabular data, the frustration caused by this effect led to the abandonment of serious use of the process.

There are several reasons for believing that, in the present-day computing environment, numerical differentiation might find a useful role. The first is that, by present standards, it is rather a small-scale calculation, so its cost may well be negligible compared with any overall saving in cost that might result from its use. Secondly, the assignment of a step length h is now generally open. One does not have to rely on tabular data. Thirdly, although the amplification of round-off error is an integral part of the calculation, its effect can be measured reliably and automatically by the routine at the time of the calculation.

Thus the user does not have to gauge the round-off level (or noise level) of the function values or assess the effect of this on the accuracy of the results. A routine does this automatically, returning with each result an error estimate which has already taken round-off error amplification into account.

We now illustrate, by means of a very simple example, the importance of the round-off error effect. A very simple approximation of $f'(0)$, based on the identity

$$f'(0) = (f(h) - f(-h))/2h + (h^2/3!)f'''(\xi), \tag{1}$$

is

$$(f(h) - f(-h))/2h.$$

If there were no precision problem, this formula would be the only one needed in the theory of first-order numerical differentiation. We could simply take $h = 10^{-40}$ (or $h = 10^{-1000}$) to obtain an excellent approximation based on two function values. It is only when we consider in detail how a finite length machine comes to calculate $(f(h) - f(-h))/2h$ that the necessity for a sophisticated theory becomes apparent.

To simplify the subsequent description we shall assume that the quantities involved are neither so close to zero that the machine underflow characteristics need be considered nor so large that machine overflow occurs. The approximation mentioned above involves the function values $f(h)$ and $f(-h)$. In general no computer has available these numbers exactly. Instead it uses approximations $\hat{f}(h)$ and $\hat{f}(-h)$ whose relative accuracy is less than some tolerance ϵ_f. If the function $f(x)$ is a library function, for example $\sin x$, ϵ_f may coincide with the machine accuracy parameter ϵ_m. More generally the function $f(x)$ is calculated in a user-supplied routine and ϵ_f is larger than ϵ_m by a small factor 5 or 6 if the calculation is short or by some larger factor in an extended calculation. This factor is not usually known by the user.

$\hat{f}(h)$ and $\hat{f}(-h)$ are related to $f(h)$ and $f(-h)$ by:

$$\hat{f}(h) = f(h)(1 + \theta_1\epsilon_f), \qquad |\theta_1| \leq 1$$

$$\hat{f}(-h) = f(-h)(1 + \theta_2\epsilon_f), \quad |\theta_2| \leq 1.$$

Thus even if the rest of the calculation were carried out exactly, it is trivial to show that

$$\frac{\hat{f}(h)-\hat{f}(-h)}{2h}-\frac{f(h)-f(-h)}{2h}\simeq 2\phi\epsilon_f\frac{f(\xi)}{2h}, \qquad |\phi|\leq 1.$$

The difference between the quantity actually calculated and the quantity which one attempts to calculate may be as large as $\epsilon_f f(\xi)/h$; for example using $h = 10^{-40}$ and $\epsilon_m = 10^{-7}$ this gives a 'conditioning error' of $10^{33}f(\xi)$.

In practice much more sophisticated formulae than (1) above are used, and for these and for the corresponding higher-derivative formulae the error analysis is different and more complicated in detail. But invariably the theory contains the same overall feature. In a finite length calculation, the error is composed of two main parts: a discretisation error which increases as h becomes large and is zero for $h = 0$; and a 'conditioning' error due to amplification of round-off error in function evaluation, which increases as h becomes small and is infinite for $h = 0$.

The routine in this chapter has to take into account internally both these sources of error in the results. Thus it returns pairs of results, DER(j) and EREST(j) where DER(j) is an approximation to $f^{(j)}(x_0)$ and EREST(j) is an estimate of the error in the approximation DER(j). If the routine has not been misled, DER(j) and EREST(j) satisfy

$$|\text{DER}(j) - f^{(j)}(x_0)| \leq \text{EREST}(j).$$

In this respect, numerical differentiation routines are fundamentally different from other routines. The user does not specify an error criterion. Instead the routine provides the error estimate and this may be very large.

We mention here a terminological distinction. A fully automatic routine is one in which the user does not need to provide any information other than that required to specify the problem. Such a routine (at a cost in computing time) decides an appropriate internal parameter such as the step length h by itself. On the other hand a routine which requires the user to provide a step length h, but automatically chooses from several different formulae each based on the specified step length, is termed a semi-automatic routine.

The situation described above must have seemed rather depressing when hand machines were in common use. For example in the simple illustration one does not know the values of the quantities $f'''(\xi)$ or ϵ_f involved in the error estimates, and the idea of altering the value of h and starting again must have seemed appalling. However by present-day standards, it is a relatively simple matter to write a program which carries out all the previously considered time-consuming calculations which may seem necessary. None of the routines envisaged for this chapter are particularly revolutionary in concept. They simply utilise the computer for the sort of task for which it was originally designed. It carries out simple tedious calculations which are necessary to estimate the accuracy of the results of other even simpler tedious calculations.

2.2 Examples of Applications for Numerical Differentiation Routines

(a) One immediate use to which a set of derivatives at a point is likely to be put is that of constructing a Taylor series representation:

$$f(x) = f(x_0) + \sum_{j=1}^{n}\frac{f^{(j)}(x_0)}{j!}(x-x_0)^j + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}, \quad |\xi - x_0| \leq x.$$

This infinite series converges so long as $|x - x_0| < R_c$ (the radius of convergence) and it is only for these values of x that such a series is likely to be used. In this case in forming the sum, the required accuracy in $f^{(j)}(x_0)$ diminishes with increasing j.

The series formed from the Taylor series using elementary operations such as integration or differentiation has the same overall characteristic. A technique based on a Taylor series expansion may be quite accurate, even if the individual derivatives are not, so long as the less accurate derivatives are associated with known small coefficients.

The error introduced by using n approximate derivatives DER(j) is bounded by:

$$\sum_{j=1}^{n}\text{EREST}(j)\,|x-x_0|^j/j!$$

Thus, if the user is prepared to base the result on a truncated Taylor series, the additional error introduced by using approximate Taylor coefficients can be readily bounded from the values of EREST(j). However in an automatic code the user must be prepared to introduce some alternative approach in case this error bound turns out to be unduly high.

In this sort of application the accuracy of the result depends on the size of the errors in the numerical differentiation. There are other applications where the effect of large errors EREST(j) is merely to prolong a calculation, but not to impair the final accuracy.

(b) A simple Taylor series approach such as described in (a) is used to find a starting value for a rapidly converging but extremely local iterative process.

(c) The technique known as 'subtracting out the singularity' as a preliminary to numerical quadrature may be extended and may be carried out approximately. Thus suppose we are interested in evaluating

$$\int_0^1 x^{-(1/2)}\phi(x)dx,$$

we have an automatic quadrature routine available, and we have a routine available for $\phi(x)$ which we know to be an analytic function. An integrand function like $x^{-(1/2)}\phi(x)$ is generally accepted to be difficult for an automatic integrator because of the singularity. If $\phi(x)$ and some of its derivatives at the singularity $x = 0$ are known one may effectively 'subtract out' the singularity using the following identity:

$$\int_0^1 x^{-(1/2)}\phi(x)dx = \int_0^1 x^{-(1/2)}(\phi(x) - \phi(0) - Ax - Bx^2/2)dx + 2\phi(0) + 2A/3 + B/5 \qquad (2)$$

with $A = \phi'(0)$ and $B = \phi''(0)$.

The integrand function on the right of (2) has no singularity, but its third derivative does. Thus using numerical quadrature for this integral is much cheaper than using numerical quadrature for the original integral (in the left-hand side of (2)).

However (2) is an identity whatever values of A and B are assigned. Thus the same procedure can be used with A and B being approximations to $\phi'(0)$ and $\phi''(0)$ provided by a numerical differentiation routine. The integrand would now be more difficult to integrate than if A and B were correct but still much less difficult than the original integrand (on the left-hand side of (2)). But, assuming that the automatic quadrature routine is reliable, the overall result would still be correct. The effect of using approximate derivatives rather than exact derivatives does not impair the accuracy of the result. It simply makes the result more expensive to obtain, but not nearly as expensive as if no derivatives were used at all.

(d) The calculation of a definite integral may be based on the Euler–Maclaurin expansion

$$\int_0^1 f(x)dx = \frac{1}{m}{\sum_{j=0}^{m}}''f(j/m) - \sum_{s=1}^{l}\frac{B_{2s}}{2s!}\frac{(f^{(2s-1)}(1) - f^{(2s-1)}(0))}{m^{2s}} + O(m^{(-2l-2)}).$$

Here B_{2s} is a Bernoulli number. If one fixes a value of l then as m is increased the right-hand side (without the remainder term) approaches the true value of the integral. This statement remains true whatever values are used to replace $f^{(2s-1)}(1)$ and $f^{(2s-1)}(0)$. If no derivatives are available, and this formula is used (effectively with the derivatives replaced by zero) the rate of convergence is slow (like m^{-2}) and a large number of function evaluations may be used in calculating the trapezoidal rule sum for large m before a sufficiently accurate result is attained. However if approximate derivatives are used, the initial rate of convergence is enhanced. In fact, in this example any derivative approximation which is closer than the approximation zero is helpful. Thus the use of inaccurate derivatives may have the effect of shortening the overall calculation, since a sufficiently accurate result may be obtained using a smaller value of m, without impairing the accuracy of the result. (The resemblance with Gregory's formula is superficial. Here l is kept fixed and m is increased, ensuring a convergent process.)

The examples given above are only intended to illustrate the sort of use to which approximate derivatives may be put. Very simple illustrations have been used for clarity, and in such simple cases there are usually more efficient approaches to the problem. The same ideas applied in a more complicated or restrictive setting may provide an efficient approach to a problem for which no simple standard approach exists.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

(a) At the present time there is only one numerical differentiation routine available in this chapter, D04AAF. This is a semi-automatic routine for obtaining approximations to the first fourteen derivatives of a real valued function $f(x)$ at a specified point x_0. The user provides a FUNCTION representing $f(x)$, the value of x_0, an upper limit $n \leq 14$ on the order of the derivatives required and a step length h. The routine returns a set of approximations DER(j) and corresponding error estimates EREST(j) which hopefully satisfy

$$|\text{DER}(j) - f^{(j)}(x_0)| \leq \text{EREST}(j), \quad j = 1, 2, \ldots, n \leq 14.$$

We term this routine a semi-automatic routine because the user provides a step length h and this is not needed to specify the problem.

It is important that the error estimate EREST(j) is taken into consideration by the user before any use of DER(j) is made. The actual size of EREST(j) depends on the analytic structure of the function, on the word length of the machine used and on the step size h, and is difficult to predict. Thus the user has to run the routine to find out how accurate the results are. Usually the user will find the higher-order derivatives are successively more inaccurate and that past a certain order, say 10 or 11, the size of EREST(j) actually exceeds DER(j). Clearly when this happens the approximation DER(j) is unusable.

(b) We have investigated a fully automatic routine, which has the same calling sequence with the exception that a step length is not required. This routine finds an appropriate step length h for itself. The cost seems to be greater by a factor of 3 to 5 but the returned values of EREST(j) are usually smaller. It is our intention to develop such a routine only if there is a demand for it in which case the experience of users with the presently available semi-automatic routine will be very helpful.

(c) There is available in the algorithm section of CACM [2] a semi-automatic Fortran routine for numerical differentiation of an analytical function $f(z)$ at a possibly complex abscissa z_0. This is a stable problem. It can be used for any problem for which D04AAF might be used and produces more accurate results, and derivatives of arbitrary high order. However even if z_0 is real and $f(z)$ is real for z, this routine requires a user-supplied FUNCTION which evaluates $f(z)$ for complex values of z and it makes use of complex arithmetic.

(d) Routines are available in Chapter E02 to differentiate functions which are polynomials (in Chebyshev series representation) or cubic splines (in B-spline representation).

4 References

[1] Lyness J N and Moler C B (1967) Numerical differentiation of analytic functions *SIAM J. Numer. Anal.* **4 (2)** 202–210

[2] Lyness J N and Ande G (1971) Algorithm 413, ENTCAF and ENTCRE: Evaluation of normalised Taylor coefficients of an analytic function *Comm. ACM* **14 (10)** 669–675

D04AAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D04AAF calculates a set of derivatives (up to order 14) of a function of one real variable at a point, together with a corresponding set of error estimates, using an extension of the Neville algorithm.

2. Specification

```
SUBROUTINE D04AAF (XVAL, NDER, HBASE, DER, EREST, FUN, IFAIL)
INTEGER          NDER, IFAIL
real             XVAL, HBASE, DER(14), EREST(14), FUN
EXTERNAL         FUN
```

3. Description

This routine provides a set of approximations:

$$\mathrm{DER}(j), \qquad j = 1,2,...,n$$

to the derivatives:

$$f^{(j)}(x_0), \qquad j = 1,2,...,n$$

of a real valued function $f(x)$ at a real abscissa x_0, together with a set of error estimates:

$$\mathrm{EREST}(j), \qquad j = 1,2,...,n$$

which hopefully satisfy:

$$|\mathrm{DER}(j) - f^{(j)}(x_0)| < \mathrm{EREST}(j), \qquad j = 1,2,...,n.$$

The user provides the value of x_0, a value of n (which is reduced to 14 should it exceed 14) a function (sub)program which evaluates $f(x)$ for all real x, and a step length h. The results DER(j) and EREST(j) are based on 21 function values:

$$f(x_0),\ f(x_0 \pm (2i-1)h), \qquad i = 1,2,...,10.$$

Internally the routine calculates the odd order derivatives and the even order derivatives separately. There is a user option for restricting the calculation to only odd (or even) order derivatives. For each derivative the routine employs an extension of the Neville Algorithm (see Lyness and Moler [2]) to obtain a selection of approximations.

For example, for odd derivatives, based on 20 function values, the routine calculates a set of numbers:

$$T_{k,p,s}, \qquad p = s,s+1,...,6,\ k = 0,1,...,9-p$$

each of which is an approximation to $f^{(2s+1)}(x_0)/(2s+1)!$. A specific approximation $T_{k,p,s}$ is of polynomial degree $2p + 2$ and is based on polynomial interpolation using function values $f(x_0 \pm (2i-1)h)$, $i = k,k+1,...,k+p$. In the absence of round-off error, the better approximations would be associated with the larger values of p and of k. However, round-off error in function values has an increasingly contaminating effect for successively larger values of p. This routine proceeds to make a judicious choice between all the approximations in the following way.

For a specified value of s, let:

$$R_p = U_p - L_p, \qquad p = s,s+1,...,6$$

$$\text{where } U_p = \max_k (T_{k,p,s}), \qquad k = 0,1,...,9-p$$

$$L_p = \min_k (T_{k,p,s}), \qquad k = 0,1,...9-p$$

and let $\bar{p}$ be such that $R_{\bar{p}} = \min_p (R_p)$ for $p = s,s+1,...,6$.

The routine returns:

$$\text{DER}(2s+1) = \frac{1}{8-\bar{p}} \times \left\{ \sum_{k=0}^{9-\bar{p}} T_{k,\bar{p},s} - U_{\bar{p}} - L_{\bar{p}} \right\} \times (2s+1)!$$

and

$$\text{EREST}(2s+1) = R_{\bar{p}} \times (2s+1)! \times K_{2s+1}$$

where K_j is a safety factor which has been assigned the values:

$K_j = 1 \quad j \leq 9$
$K_j = 1.5 \quad j = 10,11$
$K_j = 2 \quad j \geq 12$

on the basis of performance statistics.

The even order derivatives are calculated in a precisely analogous manner.

4. References

[1] LYNESS, J.N. and MOLER, C.B.
Van der Monde systems and numerical differentiation.
Num. Math., 8, pp. 458-464. 1966.

[2] LYNESS, J.N. and MOLER, C.B.
Generalised Romberg methods for integrals of derivatives.
Num. Math., 14, pp. 1-14. 1969.

5. Parameters

1: XVAL – ***real***. *Input*

On entry: the point at which the derivatives are required, x_0.

2: NDER – INTEGER. *Input*

On entry: NDER must be set so that its absolute value is the highest order derivative required. If NDER > 0, all derivatives up to order min(NDER,14) are calculated. If NDER < 0 and NDER is even, only even order derivatives up to order min(−NDER,14) are calculated. If NDER < 0 and NDER is odd, only odd order derivatives up to order min(−NDER,13) are calculated.

3: HBASE – ***real***. *Input*

On entry: the initial step length which may be positive or negative.

(If set to zero the routine does not proceed with any calculation, but sets the error flag IFAIL and returns to the (sub)program from which D04AAF is called.) For advice on the choice of HBASE see Section 8.

4: DER(14) – ***real*** array. *Output*

On exit: an approximation to the jth derivative of $f(x)$ at x = XVAL, so long as the jth derivative is one of those requested by the user when specifying NDER. For other values of j, DER(j) is unused.

5: EREST(14) – ***real*** array. *Output*

On exit: an estimate of the absolute error in the corresponding result DER(j) so long as the jth derivative is one of those requested by the user when specifying NDER. The sign of EREST(j) is positive unless the result DER(j) is questionable. It is set negative when |DER(j)| < |EREST(j)| or when for some other reason there is doubt about the validity of the result DER(j) (see Section 6). For other values of j, EREST(j) is unused.

6: FUN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUN must evaluate the function $f(x)$ at a specified point.

Its specification is:

```
real FUNCTION FUN(X)
real          X
```

1: X – ***real***. *Input*

On entry: the value of the argument x.

For users with equally spaced tabular data, the following information may be useful:

(i) in any call of D04AAF the only values of X that will be required are X = XVAL and X = XVAL ± (2j−1)HBASE, for j = 1,2,...,10; and

(ii) FUN(XVAL) is always called, but it is disregarded when only odd order derivatives are required.

FUN must be declared as EXTERNAL in the (sub)program from which D04AAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDER = 0,
or HBASE = 0.

If IFAIL has a value zero on exit then D04AAF has terminated successfully, but before any use is made of a derivative DER(j) the value of EREST(j) must be checked.

7. Accuracy

The accuracy of the results is problem dependent. An estimate of the accuracy of each result DER(j) is returned in EREST(j) (see Sections 3 and 5 8).

A basic feature of any floating-point routine for numerical differentiation based on real function values on the real axis is that successively higher order derivative approximations are successively less accurate. It is expected that in most cases DER(14) will be unusable. As an aid to this process, the sign of EREST(j) is set negative when the estimated absolute error is greater than the approximate derivative itself, i.e. when the approximate derivative may be so inaccurate that it may even have the wrong sign. It is also set negative in some other cases when information available to the routine indicates that the corresponding value of DER(j) is questionable.

The actual values in EREST depend on the accuracy of the function values, the properties of the machine arithmetic, the analytic properties of the function being differentiated and the user-provided step length HBASE (see Section 8). The only hard and fast rule is that for a given FUN(X) and HBASE, the values of EREST(j) increase with increasing j. The limit of 14 is dictated by experience. Only very rarely can one obtain meaningful approximations for higher order derivatives on conventional machines.

8. Further Comments

The time taken by the routine depends on the time spent for function evaluations. Otherwise the time is roughly equivalent to that required to evaluate the function 21 times and calculate a finite difference table having about 200 entries in total.

The results depend very critically on the choice of the user-provided step length HBASE. The overall accuracy is diminished as HBASE becomes small (because of the effect of round-off error) and as HBASE becomes large (because the discretisation error also becomes large). If the routine is used four or five times with different values of HBASE one can find a reasonably good value. A process in which the value of HBASE is successively halved (or doubled) is usually quite effective. Experience has shown that in cases in which the Taylor series for FUN(X) about XVAL has a finite radius of convergence R, the choices of HBASE $> R/21$ are not likely to lead to good results. In this case some function values lie outside the circle of convergence.

9. Example

This example evaluates the odd-order derivatives of the function:

$$f(x) = \tfrac{1}{2}e^{2x}-1$$

up to order 7 at the point $x = \frac{1}{2}$. Several different values of HBASE are used, to illustrate that:

(i) extreme choices of HBASE, either too large or too small, yield poor results;

(ii) the quality of these results is adequately indicated by the values of EREST.

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.

```
*     D04AAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             HBASE, XVAL
      INTEGER          I, IFAIL, J, K, L, NDER
*     .. Local Arrays ..
      real             DER(14), EREST(14)
*     .. External Functions ..
      real             FUN
      EXTERNAL         FUN
*     .. External Subroutines ..
      EXTERNAL         D04AAF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D04AAF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +'Four separate runs to calculate the first four odd order derivati
     +ves of'
      WRITE (NOUT,*) '   FUN(X) = 0.5*exp(2.0*X-1.0) at X = 0.5.'
      WRITE (NOUT,*) 'The exact results are 1, 4, 16 and 64'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Input parameters common to all four runs'
      WRITE (NOUT,*) '   XVAL = 0.5    NDER = -7    IFAIL = 0'
      WRITE (NOUT,*)
      HBASE = 0.5e0
      NDER = -7
      L = ABS(NDER)
      IF (NDER.GE.0) THEN
         J = 1
      ELSE
         J = 2
      END IF
```

```
      XVAL = 0.5e0
      DO 40 K = 1, 4
         IFAIL = 0
*
         CALL D04AAF(XVAL,NDER,HBASE,DER,EREST,FUN,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'with step length', HBASE,
     +     '  the results are'
         WRITE (NOUT,*) 'Order        Derivative       Error estimate'
         DO 20 I = 1, L, J
            WRITE (NOUT,99998) I, DER(I), EREST(I)
   20    CONTINUE
         HBASE = HBASE*0.1e0
   40 CONTINUE
      STOP
*
99999 FORMAT (1X,A,F9.4,A)
99998 FORMAT (1X,I2,2e21.4)
      END
*
      real  FUNCTION FUN(X)
*     .. Scalar Arguments ..
      real                X
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      FUN = 0.5e0*EXP(2.0e0*X-1.0e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D04AAF Example Program Results

 Four separate runs to calculate the first four odd order derivatives of
    FUN(X) = 0.5*exp(2.0*X-1.0) at X = 0.5.
 The exact results are 1, 4, 16 and 64

 Input parameters common to all four runs
    XVAL = 0.5    NDER = -7    IFAIL = 0


 with step length   0.5000  the results are
 Order        Derivative       Error estimate
  1           0.1392E+04          -0.1073E+06
  3          -0.3139E+04          -0.1438E+06
  5           0.8762E+04          -0.2479E+06
  7          -0.2475E+05          -0.4484E+06

 with step length   0.0500  the results are
 Order        Derivative       Error estimate
  1           0.1000E+01           0.1530E-10
  3           0.4000E+01           0.2113E-08
  5           0.1600E+02           0.3815E-06
  7           0.6400E+02           0.7385E-04

 with step length   0.0050  the results are
 Order        Derivative       Error estimate
  1           0.1000E+01           0.1221E-13
  3           0.4000E+01           0.4208E-09
  5           0.1600E+02           0.1450E-04
  7           0.6404E+02           0.2973E+00
```

```
with step length   0.0005  the results are
Order        Derivative       Error estimate
 1           0.1000E+01           0.1422E-12
 3           0.4000E+01           0.3087E-06
 5           0.1599E+02           0.6331E+00
 7           0.3825E+05          -0.1964E+07
```

Chapter D05 – Integral Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D05AAF	5	Linear non-singular Fredholm integral equation, 2nd kind, split kernel
D05ABF	6	Linear non-singular Fredholm integral equation, 2nd kind, smooth kernel
D05BAF	14	Nonlinear Volterra convolution equation, 2nd kind
D05BDF	16	Nonlinear convolution Volterra–Abel equation, 2nd kind, weakly singular
D05BEF	16	Nonlinear convolution Volterra–Abel equation, 1st kind, weakly singular
D05BWF	16	Generate weights for use in solving Volterra equations
D05BYF	16	Generate weights for use in solving weakly singular Abel type equations

Chapter D05

Integral Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of integral equations. Provision will be made for most of the standard types of equation (see below). The following are, however, specifically excluded:

(a) Equations arising in the solution of partial differential equations by integral equation methods. In cases where the prime purpose of an algorithm is the solution of a partial differential equation it will normally be included in Chapter D03.

(b) Calculation of inverse integral transforms. This problem falls within the ambit of Chapter C06.

2 Background to the Problems

2.1 Introduction

Any functional equation in which the unknown function appears under the sign of integration is called an integral equation. Integral equations arise in a great many branches of science; for example, in potential theory, acoustics, elasticity, fluid mechanics, radiative transfer, theory of population, etc. In many instances the integral equation originates from the conversion of a boundary-value problem or an initial-value problem associated with a partial or an ordinary differential equation, but many problems lead directly to integral equations and cannot be formulated in terms of differential equations.

Integral equations are of many types; here we attempt to indicate some of the main distinguishing features with particular regard to the use and construction of algorithms.

2.2 Classification of Integral Equations

In the classical theory of integral equations one distinguishes between *Volterra* equations and *Fredholm* equations. In a Fredholm equation the region of integration is fixed, whereas in a Volterra equation the region is variable. Thus, the equation

$$cy(t) = f(t) + \lambda \int_a^b K(t, s, y(s))\, ds, \quad a \leq t \leq b \tag{1}$$

is an example of Fredholm equation, and the equation

$$cy(t) = f(t) + \lambda \int_a^t K(t, s, y(s))\, ds, \quad a \leq t \tag{2}$$

is an example of a Volterra equation.

Here the *forcing* function $f(t)$ and the *kernel* function $K(t, s, y(s))$ are prescribed, while $y(t)$ is the unknown function to be determined. (More generally the integration and the domain of definition of the functions may extend to more than one dimension.) The parameter λ is often omitted; it is, however, of importance in certain theoretical investigations (e.g. stability) and in the eigenvalue problem discussed below.

If in (1), or (2), $c = 0$, the integral equation is said to be of the *first kind.* If $c = 1$, the equation is said to be of the *second kind.*

Equations (1) and (2) are *linear* if the kernel $K(t, s, y(s)) = k(t, s)y(s)$, otherwise they are *nonlinear.*

Note. in a linear integral equation, $k(t, s)$ is usually referred to as the kernel. We adopt this convention throughout.

These two types of equations are broadly analogous to problems of initial- and boundary-value type for an ordinary differential equation (ODE); thus the Volterra equation, characterised by a variable upper limit of integration, is amenable to solution by methods of marching type whilst most methods for treating Fredholm equations lead ultimately to the solution of an approximating system of simultaneous algebraic equations. For comprehensive discussion of numerical methods see Atkinson [1], Baker [2], Brunner and van der Houwen [3] and Delves and Walsh [5]. In what follows, the term 'integral equation' is used in its general sense, and the type is distinguished when appropriate.

2.3 Structure of Kernel

When considering numerical methods for integral equations, particular attention should be paid to the character of the kernel, which is usually the main factor governing the choice of an appropriate quadrature formula or system of approximating functions. Various commonly occurring types of singularity call for individual treatment.

Likewise provision can be made for cases of symmetry, periodicity or other special structure, where the solution may have special properties and/or economies may be effected in the solution process. We note in particular the following cases to which we shall often have occasion to refer in the description of individual algorithms

(a) A linear integral equation with a kernel $k(t,s) = k(s,t)$ is said to be **symmetric**. This property plays a key role in the theory of Fredholm integral equations.

(b) If $k(t,s) = k(a+b-t, a+b-s)$ in a linear integral equation, the kernel is called **centro-symmetric**.

(c) If in Equations (1) or (2) the kernel has the form $K(t,s,y(s)) = k(t-s)g(s,y(s))$, the equation is called a **convolution** integral equation; in the linear case $g(s,y(s)) = y(s)$.

(d) If the kernel in (1) has the form

$$K(t,s,y(s)) = K_1(t,s,y(s)), \quad a \leq s \leq t,$$
$$K(t,s,y(s)) = K_2(t,s,y(s)), \quad t < s \leq b,$$

where the functions K_1 and K_2 are well-behaved, whilst K or its s-derivative is possibly discontinuous, may be described as discontinuous or of 'split' type; in the linear case $K(t,s,y(s)) = k(t,s)y(s)$ and consequently $K_1 = k_1 y$ and $K_2 = k_2 y$. Examples are the commonly occurring kernels of the type $k(|t-s|)$ and the Green's functions (influence functions) which arise in the conversion of ODE boundary-value problems to integral equations. It is also of interest to note that the Volterra equation (2) may be conceived as a Fredholm equation with kernel of split type, with $K_2(t,s,y(s)) \equiv 0$; consequently methods designed for the solution of Fredholm equations with split kernels are also applicable to Volterra equations.

2.4 Singular and Weakly Singular Equations

An integral equation may be called singular if either

(a) its kernel contains a singularity, or
(b) the range of integration is infinite,

and it is said to be weakly-singular if the kernel becomes infinite at $s = t$.

Sometimes a solution can be effected by a simple adaptation of a method applicable to a non-singular equation: for example, an infinite range may be truncated at a suitably chosen point. In other cases, however, theoretical considerations will dictate the need for special methods and algorithms. Examples are:

(i) Integral equations with singular kernels of Cauchy type;

(ii) Equations of Wiener–Hopf type;

(iii) Various dual integral equations arising in the solution of boundary-value problems of mathematical physics;

(iv) The well-known Abel integral equation, an equation of Volterra type, whose kernel contains an inverse square-root singularity at $s = t$.

Problems of inversion of integral transforms also fall under this heading but, as already remarked, they lie outside the scope of this chapter.

2.5 Fredholm Integral Equations

2.5.1 Eigenvalue problem

Closely connected with the linear Fredholm integral equation of the second kind is the eigenvalue problem represented by the homogeneous equation

$$y(t) - \lambda \int_a^b k(t,s)y(s)\, ds = 0, \quad a \leq t \leq b. \tag{3}$$

If λ is chosen arbitrarily this equation in general possesses only the trivial solution $y(t) = 0$. However, for a certain critical set of values of λ, the **characteristic values** or eigenvalues (the latter term is sometimes reserved for the reciprocals $\mu = 1/\lambda$), there exist non-trivial solutions $y(t)$, termed **characteristic functions** or **eigenfunctions**, which are of fundamental importance in many investigations. The analogy with the eigenproblem of linear algebra is readily apparent, and indeed most methods of solution of equation (3) entail reduction to an approximately equivalent algebraic problem

$$(K - \mu I)y = 0. \tag{4}$$

2.5.2 Equations of the first kind

The Fredholm integral equation of the first kind

$$\int_a^b k(t,s)y(s)\ ds = f(t), \quad a \leq t \leq b, \tag{5}$$

belong to the class of 'ill-posed' problems; even supposing that a solution corresponding to the prescribed $f(t)$ exists, a slight perturbation of $f(t)$ may give rise to an arbitrarily large variation in the solution $y(t)$. Hence the equation may be closely satisfied by a function bearing little resemblance to the 'true' solution. The difficulty associated with this instability is aggravated by the fact that in practice the specification of $f(t)$ is usually inexact.

Nevertheless a great many physical problems (e.g. in radiography, spectroscopy, stereology, chemical analysis) are appropriately formulated in terms of integral equations of the first kind, and useful and meaningful 'solutions' can be obtained with the aid of suitable stabilising procedures. See Chapters 12 and 13 of Delves and Walsh [5] for further discussion and references.

2.5.3 Equations of the second kind

Consider the nonlinear Fredholm equation of the second kind

$$y(t) = f(t) + \int_a^b K(t,s,y(s))\ ds, \quad a \leq t \leq b. \tag{6}$$

The numerical solution of equation (6) is usually accomplished either by simple iteration or by a more sophisticated iterative scheme based on Newton's method; in the latter case it is necessary to solve a sequence of linear integral equations. Convergence may be demonstrated subject to suitable conditions of Lipschitz continuity of the functions K with respect to the argument y.

Examples of Fredholm type (for which the provision of algorithms is contemplated) are:

(a) the Uryson equation

$$u(t) - \int_0^1 F(t,s,u(s))\ ds = 0, \quad 0 \leq t \leq 1, \tag{7}$$

(b) the Hammerstein equation

$$u(t) - \int_0^1 k(t,s)g(s,u(s))\ ds = 0, \quad 0 \leq t \leq 1, \tag{8}$$

where F and g are arbitrary functions.

2.6 Volterra Integral Equations

2.6.1 Equations of the first kind

Consider the Volterra integral equation of the first kind

$$\int_a^t k(t,s)y(s)\ ds = f(t), \quad a \leq t. \tag{9}$$

Clearly it is necessary that $f(a) = 0$; otherwise no solution to (9) can exist.

The following types of Volterra integral equations of the first kind occur in real life problems:

equations with unbounded kernel at $s = t$,

equations with sufficiently smooth kernel.

These types belong also to the class of 'ill-posed' problems. However, the instability is appreciably less severe in the equations with unbounded kernel. In general, a non-singular Volterra equation of the first kind presents less computational difficulty than the Fredholm equation (5) with a smooth kernel.

A Volterra equation of the first kind may, under suitable conditions, be converted by differentiation to one of the second kind or by integration by parts to an equation of the second kind for the integral of the wanted function.

2.6.2 Equations of the second kind

A very general Volterra equation of the second kind is given by

$$y(t) = f(t) + \int_a^t K(t, s, y(s))\, ds, \quad a \leq t. \tag{10}$$

The resemblance of Volterra equations to ODEs suggests that the underlying methods for ODE problems can be applied to Volterra equations. Indeed this turns out to be the case. The main advantages of implementing these methods are their well-developed theoretical background, i.e., convergence and stability, see Brunner and van der Houwen [3], Wolkenfelt [6].

Many Volterra integral equations arising in real life problems have a convolution kernel (cf. Section 2.3(c)), see [3] for references. However, a subclass of these equations which have kernels of the form

$$k(t - s) = \sum_{j=0}^{M} \lambda_j (t - s)^j, \tag{11}$$

where $\{\lambda_j\}$ are real, can be converted into a system of linear or nonlinear ODEs, see [3].

For more information on the theoretical and the numerical treatment of integral equations we refer the user to Atkinson [1], Baker [2], Brunner and van der Houwen [3], Cochran [4] and Delves and Walsh [5].

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

The choice of routine will depend first of all upon the type of integral equation to be solved.

3.1 Fredholm Equations of Second Kind

(a) Linear equations

D05AAF is applicable to an equation with a discontinuous or 'split' kernel as defined in 2.3.(d). Here, however, both the functions k_1 and k_2 are required to be defined (and well-behaved) throughout the square $a \leq s, t \leq b$.

D05ABF is applicable to an equation with a smooth kernel. Note that D05AAF may also be applied to this case, by setting $k_1 = k_2 = k$, but D05ABF is more efficient.

3.2 Volterra Equations of Second Kind

(a) Linear equations

D05AAF may be used to solve a Volterra equation by defining k_2 (or k_1) to be identically zero. (See also (b).)

(b) Nonlinear equations

D05BAF is applicable to a nonlinear convolution Volterra integral equation of the second kind. The kernel function has the form

$$K(t, s, y(s)) = k(t - s)g(s, y(s)).$$

The underlying methods used in the routine are the reducible linear multi-step methods. The user has a choice of variety of these methods. This routine can also be used for linear g.

D05BDF is applicable to a nonlinear convolution equation having a weakly-singular kernel (Abel). The kernel function has the form

$$K(t, s, y(s)) = \frac{k(t - s)}{\sqrt{t - s}}g(s, y(s)).$$

The underlying methods used in the routine are the fractional linear multistep methods based on BDF methods. This routine can also be used for linear g.

3.3 Volterra Equations of First Kind

(a) Linear equations

See (b).

(b) Nonlinear qquations

D05BEF is applicable to a nonlinear equation having a weakly-singular kernel (Abel). The kernel function has the form

$$K(t, s, y(s)) = \frac{k(t - s)}{\sqrt{t - s}}g(s, y(s)).$$

The underlying methods used in the routine are the fractional linear multistep methods based on BDF methods. This routine can also be used for linear g.

3.4 Utility Routines

D05BWF generates the weights associated with Adams and BDF linear multistep methods. These weights can be used for the solution of non-singular Volterra integral and integro-differential equations of general type.

D05BYF generates the weights associated with BDF linear multistep methods. These weights can be used for the solution of weakly-singular Volterra (Abel) integral equations of general type.

3.5 User-supplied Routines

All the routines in this chapter require the user to supply functions or real procedures defining the kernels and other given functions in the equations. It is important to test these independently before using them in conjunction with NAG Library routines.

4 Index

5 References

[1] Atkinson K E (1976) *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind* SIAM, Philadelphia

[2] Baker C T H (1977) *The Numerical Treatment of Integral Equations* Oxford University Press

[3] Brunner H and Van Der Houwen P J (1986) *The Numerical Solution of Volterra Equations* CWI Monographs, North-Holland, Amsterdam

[4] Cochran J A (1972) *The Analysis of Linear Integral Equations* McGraw-Hill

[5] Delves L M and Walsh J (1974) *Numerical Solution of Integral Equations* Clarendon Press, Oxford

[6] Wolkenfelt P H M (1982) The construction of reducible quadrature rules for Volterra integral and integro-differential equations *IMA J. Numer. Anal.* **2** 131–152

D05AAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05AAF solves a linear, non-singular Fredholm equation of the second kind with a split kernel.

2. Specification

```
      SUBROUTINE D05AAF (LAMBDA, A, B, K1, K2, G, F, C, N, IND, W1, W2,
     1                   WD, NMAX, MN, IFAIL)
      INTEGER          N, IND, NMAX, MN, IFAIL
      real             LAMBDA, A, B, K1, K2, G, F(N), C(N), W1(NMAX,MN),
     1                 W2(MN,4), WD(MN)
      EXTERNAL         K1, K2, G
```

3. Description

D05AAF solves an integral equation of the form

$$f(x) - \lambda \int_a^b k(x,s)\, f(s)\, ds = g(x)$$

for $a \leq x \leq b$, when the kernel k is defined in two parts: $k = k_1$ for $a \leq s \leq x$ and $k = k_2$ for $x < s \leq b$. The method used is that of El-gendi [2] for which, it is important to note, each of the functions k_1 and k_2 must be defined, smooth and non-singular, for all x and s in the interval $[a,b]$.

An approximation to the solution $f(x)$ is found in the form of an n term Chebyshev-series $\sum_{i=1}^{n}{}' c_i T_i(x)$, where $'$ indicates that the first term is halved in the sum. The coefficients c_i, for $i = 1,2,..,n$, of this series are determined directly from approximate values f_i, for $i = 1,2,...,n$, of the function $f(x)$ at the first n of a set of $m + 1$ Chebyshev points:

$$x_i = \tfrac{1}{2}(a+b+(b-a)\cos[(i-1)\pi/m]), \qquad i = 1,2,...,m+1.$$

The values f_i are obtained by solving simultaneous linear algebraic equations formed by applying a quadrature formula (equivalent to the scheme of Clenshaw and Curtis [1]) to the integral equation at the above points.

In general $m = n - 1$. However, if the kernel k is centro-symmetric in the interval $[a,b]$, i.e. if $k(x,s) = k(a+b-x,a+b-s)$, then the routine is designed to take advantage of this fact in the formation and solution of the algebraic equations. In this case, symmetry in the function $g(x)$ implies symmetry in the function $f(x)$. In particular, if $g(x)$ is even about the mid-point of the range of integration, then so also is $f(x)$, which may be approximated by an even Chebyshev-series with $m = 2n - 1$. Similarly, if $g(x)$ is odd about the mid-point then $f(x)$ may be approximated by an odd series with $m = 2n$.

4. References

[1] CLENSHAW, C.W. and CURTIS, A.R.
A method for numerical integration on an automatic computer.
Numerische Math., 2, pp. 197-205, 1960.

[2] EL-GENDI, S.E.
Chebyshev solution of differential, integral and integro-differential equations.
Comput. J., 12, pp. 282-287, 1969.

5. Parameters

1: LAMBDA – ***real***. *Input*

On entry: the value of the parameter λ of the integral equation.

2: A – ***real***. *Input*

On entry: the lower limit of integration, a.

3: B – ***real***. *Input*

On entry: the upper limit of integration, b.

Constraint: B > A.

4: K1 – ***real*** FUNCTION, supplied by the user. *External Procedure*

K1 must evaluate the kernel $k(x,s) = k_1(x,s)$ of the integral equation for $a \leq s \leq x$.

Its specification is:

```
real FUNCTION K1(X, S)
real         X, S
```

1: X – ***real***. *Input*

2: S – ***real***. *Input*

On entry: the values of x and s at which $k_1(x,s)$ is to be evaluated.

K1 must be declared as EXTERNAL in the (sub)program from which D05AAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: K2 – ***real*** FUNCTION, supplied by the user. *External Procedure*

K2 must evaluate the kernel $k(x,s) = k_2(x,s)$ of the integral equation for $x < s \leq b$.

Its specification is:

```
real FUNCTION K2(X, S)
real         X, S
```

1: X – ***real***. *Input*

2: S – ***real***. *Input*

On entry: the values of x and s at which $k_2(x,s)$ is to be evaluated.

K2 must be declared as EXTERNAL in the (sub)program from which D05AAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

Note that the functions k_1 and k_2 must be defined, smooth and non-singular for all x and s in the interval $[a,b]$.

6: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x)$ for $a \leq x \leq b$.

Its specification is:

```
real FUNCTION G(X)
real         X
```

1: X – ***real***. *Input*

On entry: the values of x at which $g(x)$ is to be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D05AAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: F(N) – ***real*** array. *Output*

On exit: the approximate values f_i, for i = 1,2,...,N of $f(x)$ evaluated at the first N of M + 1 Chebyshev points x_i, (see Section 3).

If IND is 0 or 3, M = N − 1; if IND is 1, M = 2×N and if IND is 2, M = 2×N − 1.

8: C(N) – ***real*** array. *Output*

On exit: the coefficients c_i, for i = 1,2,...,N of the Chebyshev-series approximation to $f(x)$.

If IND is 1 this series contains polynomials of odd order only and if IND is 2 the series contains even order polynomials only.

9: N – INTEGER. *Input*

On entry: the number of terms in the Chebyshev-series required to approximate $f(x)$.

10: IND – INTEGER. *Input*

On entry: IND must be set to 0,1,2 or 3.

IND = 0

$k(x,s)$ is not centro-symmetric (or no account is to be taken of centro-symmetry).

IND = 1

$k(x,s)$ is centro-symmetric and $g(x)$ is odd.

IND = 2

$k(x,s)$ is centro-symmetric and $g(x)$ is even.

IND = 3

$k(x,s)$ is centro-symmetric but $g(x)$ is neither odd nor even.

11: W1(NMAX,MN) – ***real*** array. *Workspace*
12: W2(MN,4) – ***real*** array. *Workspace*
13: WD(MN) – ***real*** array. *Workspace*

14: NMAX – INTEGER. *Input*

On entry: the first dimension of the array W1 as declared in the (sub)program from which D05AAF is called.

Constraint: NMAX ≥ N.

15: MN – INTEGER. *Input*

On entry: the first dimension of the array W2 as declared in the (sub)program from which D05AAF is called.

Constraint: MN ≥ 2×N + 2.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

A ≥ B.

IFAIL = 2

A failure has occurred (in F04AAF unless N = 1) due to proximity to an eigenvalue. In general, if LAMBDA is near an eigenvalue of the integral equation, the corresponding matrix will be nearly singular.

7. Accuracy

No explicit error estimate is provided by the routine but it is usually possible to obtain a good indication of the accuracy of the solution either

(i) by examining the size of the later Chebyshev coefficients c_i, or

(ii) by comparing the coefficients c_i or the function values f_i for two or more values of N.

8. Further Comments

The time taken by the routine increases with N.

This routine may be used to solve an equation with a continuous kernel by calling the same FUNCTION for K2 as for K1.

This routine may also be used to solve a Volterra equation by defining K2 (or K1) to be identically zero.

9. Example

The example program solves the equation

$$f(x) - \int_0^1 k(x,s)\, f(s)\, ds = (1-\frac{1}{\pi^2})\, \sin(\pi x)$$

where

$$k(x,s) = \begin{cases} s(1-x) & \text{for } 0 \le s < x, \\ x(1-s) & \text{for } x \le s \le 1. \end{cases}$$

Five terms of the Chebyshev-series are sought, taking advantage of the centro-symmetry of the $k(x,s)$ and even nature of $g(x)$ about the mid-point of the range [0,1].

The approximate solution at the point $x = 0.1$ is calculated by calling C06DBF.

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.

```
*     D05AAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, NMAX, MN
      PARAMETER        (N=5,NMAX=N,MN=2*N+2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             R
*     .. Local Scalars ..
      real             A, ANS, B, LAMBDA, X
      INTEGER          I, IFAIL, IND, IS
*     .. Local Arrays ..
      real             C(NMAX), F(NMAX), W1(NMAX,MN), W2(MN,4), WD(MN)
*     .. External Functions ..
      real             C06DBF, G, K1, K2, X01AAF
      EXTERNAL         C06DBF, G, K1, K2, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D05AAF
*     .. Common blocks ..
      COMMON           R
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05AAF Example Program Results'
      WRITE (NOUT,*)
      R = X01AAF(0.0e0)
      LAMBDA = 1.0e0
      A = 0.0e0
      B = 1.0e0
      IND = 2
      IFAIL = 0
      WRITE (NOUT,*)
     +'Kernel is centro-symmetric and G is even so the solution is even'
      WRITE (NOUT,*)
*
      CALL D05AAF(LAMBDA,A,B,K1,K2,G,F,C,N,IND,W1,W2,WD,NMAX,MN,IFAIL)
*
      WRITE (NOUT,*) 'Chebyshev coefficients'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (C(I),I=1,N)
      WRITE (NOUT,*)
      X = 0.1e0
*     Note that X has to be transformed to range [-1,1]
      IS = 1
      IF (IND.EQ.1) THEN
         IS = 3
      ELSE
         IF (IND.EQ.2) IS = 2
      END IF
      ANS = C06DBF(2.0e0/(B-A)*(X-0.5e0*(B+A)),C,N,IS)
      WRITE (NOUT,99999) 'X=', X, '    ANS=', ANS
      STOP
*
99999 FORMAT (1X,A,F5.2,A,1F10.4)
99998 FORMAT (1X,5e14.4)
      END
*
      real  FUNCTION K1(X,S)
*     .. Scalar Arguments ..
      real                S, X
*     .. Executable Statements ..
      K1 = S*(1.0e0-X)
      RETURN
      END
*
      real  FUNCTION K2(X,S)
*     .. Scalar Arguments ..
      real                S, X
*     .. Executable Statements ..
      K2 = X*(1.0e0-S)
      RETURN
      END
*
      real  FUNCTION G(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      real              R
*     .. Intrinsic Functions ..
      INTRINSIC         SIN
*     .. Common blocks ..
      COMMON            R
*     .. Executable Statements ..
      G = SIN(R*X)*(1.0e0-1.0e0/(R*R))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05AAF Example Program Results

Kernel is centro-symmetric and G is even so the solution is even

Chebyshev coefficients

    0.9440E+00   -0.4994E+00    0.2799E-01   -0.5967E-03    0.6658E-05

X= 0.10    ANS=    0.3090
```

D05ABF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05ABF solves any linear non-singular Fredholm integral equation of the second kind with a smooth kernel.

2. Specification

```
      SUBROUTINE D05ABF (K, G, LAMBDA, A, B, ODOREV, EV, N, CM, F1, WK,
     1                   NMAX, NT2P1, F, C, IFAIL)
      INTEGER          N, NMAX, NT2P1, IFAIL
      real             K, G, LAMBDA, A, B, CM(NMAX,NMAX), F1(NMAX,1),
     1                 WK(2,NT2P1), F(N), C(N)
      LOGICAL          ODOREV, EV
      EXTERNAL         K, G
```

3. Description

This routine uses the method of El-gendi [2] to solve an integral equation of the form

$$f(x) - \lambda \int_a^b k(x,s)f(s)ds = g(x)$$

for the function $f(x)$ in the range $a \leq x \leq b$.

An approximation to the solution $f(x)$ is found in the form of an n term Chebyshev-series $\sum_{i=1}^{n}{}' c_i T_i(x)$, where $'$ indicates that the first term is halved in the sum. The coefficients c_i, for $i = 1,2,...,n$, of this series are determined directly from approximate values f_i, for $i = 1,2,...,n$, of the function $f(x)$ at the first n of a set of $m + 1$ Chebyshev points

$$x_i = \tfrac{1}{2}(a+b+(b-a)\times\cos[(i-1)\times\pi/m]), \qquad i = 1,2,...,m+1.$$

The values f_i are obtained by solving a set of simultaneous linear algebraic equations formed by applying a quadrature formula (equivalent to the scheme of Clenshaw and Curtis [1]) to the integral equation at each of the above points.

In general $m = n - 1$. However, advantage may be taken of any prior knowledge of the symmetry of $f(x)$. Thus if $f(x)$ is symmetric (i.e. even) about the mid-point of the range (a,b), it may be approximated by an even Chebyshev-series with $m = 2n - 1$. Similarly, if $f(x)$ is anti-symmetric (i.e. odd) about the mid-point of the range of integration, it may be approximated by an odd Chebyshev-series with $m = 2n$.

4. References

[1] CLENSHAW, C.W. and CURTIS, A.R.
A method for numerical integration on an automatic computer.
Numerische. Math., 2, pp. 197-205, 1960.

[2] EL-GENDI, S.E.
Chebyshev solution of differential, integral and integro-differential equations.
Comput. J., 12, pp. 282-287, 1969.

5. Parameters

1: K – ***real*** FUNCTION, supplied by the user. *External Procedure*

K must compute the value of the kernel $k(x,s)$ of the integral equation over the square $a \leq x \leq b, a \leq s \leq b$.

Its specification is:

```
real FUNCTION K(X, S)
real          X, S
```

1: X – ***real***. *Input*

2: S – ***real***. *Input*

On entry: the values of x and s at which $k(x,s)$ is to be calculated.

K must be declared as EXTERNAL in the (sub)program from which D05ABF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must compute the value of the function $g(x)$ of the integral equation in the interval $a \leq x \leq b$.

Its specification is:

```
real FUNCTION G(X)
real          X
```

1: X – ***real***. *Input*

On entry: the value of x at which $g(x)$ is to be calculated.

G must be declared as EXTERNAL in the (sub)program from which D05ABF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: LAMBDA – ***real***. *Input*

On entry: the value of the parameter λ of the integral equation.

4: A – ***real***. *Input*

On entry: the lower limit of integration, a.

5: B – ***real***. *Input*

On entry: the upper limit of integration, b.

Constraint: B > A.

6: ODOREV – LOGICAL. *Input*

On entry: indicates whether it is known that the solution $f(x)$ is odd or even about the mid-point of the range of integration. If ODOREV is .TRUE. then an odd or even solution is sought depending upon the value of EV.

7: EV – LOGICAL. *Input*

On entry: EV is ignored if ODOREV is .FALSE. Otherwise, if EV is .TRUE., an even solution is sought, whilst if EV is .FALSE., an odd solution is sought.

8: N – INTEGER. *Input*

On entry: the number of terms in the Chebyshev-series which approximates the solution $f(x)$.

9: CM(NMAX,NMAX) – ***real*** array. *Workspace*
10: F1(NMAX,1) – ***real*** array. *Workspace*
11: WK(2,NT2P1) – ***real*** array. *Workspace*

12: NMAX – INTEGER. *Input*

On entry: the first dimension of arrays CM and F1 as declared in the (sub)program from which D05ABF is called.

Constraint: NMAX ≥ N.

13: NT2P1 – INTEGER. *Input*

On entry: the value 2×N+1.

14: F(N) – ***real*** array. *Output*

On exit: the approximate values f_i, for i = 1,2,...,N, of the function $f(x)$ at the first N of M + 1 Chebyshev points (see Section 3).

If ODOREV is .TRUE., then M = 2×N – 1 if EV is .TRUE. and M = 2×N if EV is .FALSE.; otherwise M = N – 1.

15: C(N) – ***real*** array. *Output*

On exit: the coefficients c_i, for i = 1,2,...,N, of the Chebyshev-series approximation to $f(x)$. When ODOREV is .TRUE., this series contains polynomials of even order only or of odd order only, according to EV being .TRUE. or .FALSE. respectively.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

A ≥ B.

IFAIL = 2

A failure has occurred (in F04AAF unless N = 1) due to proximity to an eigenvalue. In general, if LAMBDA is near an eigenvalue of the integral equation, the corresponding matrix will be nearly singular.

7. Accuracy

No explicit error estimate is provided by the routine but it is possible to obtain a good indication of the accuracy of the solution either

(i) by examining the size of the later Chebyshev coefficients c_i, or

(ii) by comparing the coefficients c_i or the function values f_i for two or more values of N.

8. Further Comments

The time taken by the routine depends upon the value of N and upon the complexity of the kernel function $k(x,s)$.

9. Example

Solve Love's equation:

$$f(x) + \frac{1}{\pi}\int_{-1}^{1} \frac{f(s)}{1 + (x-s)^2}\, ds = 1$$

The example program will solve the slightly more general equation:

$$f(x) - \lambda \int_{a}^{b} k(x,s)f(s)\, ds = 1$$

where $k(x,s) = \alpha/(\alpha^2+(x-s)^2)$. The values $\lambda = -1/\pi$, $a = -1$, $b = 1$, $\alpha = 1$ are used below.

It is evident from the symmetry of the given equation that $f(x)$ is an even function. Advantage is taken of this fact both in the application of D05ABF, to obtain the $f_i \simeq f(x_i)$ and the c_i, and in subsequent applications of C06DBF to obtain $f(x)$ at selected points.

The program runs for N = 5 and N = 10.

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.

```
*     D05ABF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, NT2P1
      PARAMETER        (NMAX=10,NT2P1=2*NMAX+1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             ALPHA, W
*     .. Local Scalars ..
      real             A, A1, B, CHEBR, D, E, LAMBDA, S, X
      INTEGER          I, IFAIL, N, SS
      LOGICAL          EV, ODOREV
*     .. Local Arrays ..
      real             C(NMAX), CM(NMAX,NMAX), F(NMAX), F1(NMAX,1),
     +                 WK(2,NT2P1)
*     .. External Functions ..
      real             C06DBF, GE, KE
      EXTERNAL         C06DBF, GE, KE
*     .. External Subroutines ..
      EXTERNAL         D05ABF
*     .. Common blocks ..
      COMMON           /AFRED2/ALPHA, W
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05ABF Example Program Results'
      WRITE (NOUT,*)
      ODOREV = .TRUE.
      EV = .TRUE.
      LAMBDA = -0.3183e0
      A = -1.0e0
      B = 1.0e0
      ALPHA = 1.0e0
      W = ALPHA*ALPHA
      IF (ODOREV .AND. EV) THEN
         WRITE (NOUT,*) 'Solution is even'
      ELSE
         IF (ODOREV) WRITE (NOUT,*) 'Solution is odd'
      END IF
      DO 60 N = 5, NMAX, 5
         IFAIL = 1
*
```

```
      CALL D05ABF(KE,GE,LAMBDA,A,B,ODOREV,EV,N,CM,F1,WK,NMAX,NT2P1,F,
     +            C,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Results for N =', N
         WRITE (NOUT,*)
         WRITE (NOUT,*) '  I          F(I)          C(I)'
         DO 20 I = 1, N
            WRITE (NOUT,99998) I, F(I), C(I)
   20    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) '      X            F(X)'
         IF (ODOREV) THEN
            IF (EV) THEN
               SS = 2
            ELSE
               SS = 3
            END IF
         ELSE
            SS = 1
         END IF
         A1 = 0.5e0*(A+B)
         S = 0.5e0*(B-A)
         X = A1
         IF ( .NOT. ODOREV) THEN
            X = X - 5
         ELSE
            X = A1
         END IF
         D = 1.0e0/S
         S = 0.25e0*S
         E = B + 0.1e0*S
   40    CHEBR = C06DBF((X-A1)*D,C,N,SS)
         WRITE (NOUT,99997) X, CHEBR
         X = X + S
         IF (X.LT.E) GO TO 40
      ELSE
         IF (IFAIL.EQ.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Failure in D05ABF -'
            WRITE (NOUT,*) 'error in integration limits'
         ELSE
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Failure in D05ABF -'
            WRITE (NOUT,*) 'LAMBDA near eigenvalue'
         END IF
      END IF
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,I3,F15.5,e15.5)
99997 FORMAT (1X,F8.4,F15.5)
      END
*
      real  FUNCTION KE(X,S)
*     .. Scalar Arguments ..
      real              S, X
*     .. Scalars in Common ..
      real              ALPHA, W
*     .. Common blocks ..
      COMMON            /AFRED2/ALPHA, W
*     .. Executable Statements ..
      KE = ALPHA/(W+(X-S)*(X-S))
      RETURN
      END
*
```

```
      real   FUNCTION GE(X)
*     .. Scalar Arguments ..
      real                 X
*     .. Executable Statements ..
      GE = 1.0e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05ABF Example Program Results

Solution is even

Results for N =  5

  I          F(I)          C(I)
  1         0.75572     0.14152E+01
  2         0.74534     0.49384E-01
  3         0.71729    -0.10476E-02
  4         0.68319    -0.23282E-03
  5         0.66051     0.20890E-04

     X           F(X)
 0.0000         0.65742
 0.2500         0.66383
 0.5000         0.68319
 0.7500         0.71489
 1.0000         0.75572

Results for N = 10

  I          F(I)          C(I)
  1         0.75572     0.14152E+01
  2         0.75336     0.49384E-01
  3         0.74639    -0.10475E-02
  4         0.73525    -0.23275E-03
  5         0.72081     0.19986E-04
  6         0.70452     0.98675E-06
  7         0.68825    -0.23796E-06
  8         0.67404     0.18581E-08
  9         0.66361     0.24483E-08
 10         0.65812    -0.16527E-09

     X           F(X)
 0.0000         0.65742
 0.2500         0.66384
 0.5000         0.68319
 0.7500         0.71489
 1.0000         0.75572
```

D05BAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05BAF computes the solution of a nonlinear convolution Volterra integral equation of the second kind using a reducible linear multi-step method.

2. Specification

```
      SUBROUTINE D05BAF (CK, CG, CF, METHOD, IORDER, ALIM, TLIM, YN,
     1                   ERREST, NMESH, TOL, THRESH, WORK, LWK, IFAIL)
      INTEGER          IORDER, NMESH, LWK, IFAIL
      real             CK, CG, CF, ALIM, TLIM, YN(NMESH), ERREST(NMESH),
     1                 TOL, THRESH, WORK(LWK)
      CHARACTER*1      METHOD
      EXTERNAL         CK, CG, CF
```

3. Description

D05BAF computes the numerical solution of the nonlinear convolution Volterra integral equation of the second kind

$$y(t) = f(t) + \int_a^t k(t-s)g(s,y(s))ds, \qquad a \le t \le T. \tag{1}$$

It is assumed that the functions involved in (1) are sufficiently smooth. The routine uses a reducible linear multi-step formula selected by the user to generate a family of quadrature rules. The reducible formulae available in D05BAF are the Adams-Moulton formulae of orders 3 to 6, and the backward differentiation formulae (BDF) of orders 2 to 5. For more information about the behaviour and the construction of these rules we refer to Lubich [1] and Wolkenfelt [3].

The algorithm is based on computing the solution in a step-by-step fashion on a mesh of equi-spaced points. The initial stepsize which is given by $(T-a)/N$, N being the number of points at which the solution is sought, is halved and another approximation to the solution is computed. This extrapolation procedure is repeated until successive approximations satisfy a user specified error requirement.

The above methods require some starting values. For the Adams formula of order greater than 3 and the BDF of order greater than 2 we employ an explicit Dormand-Prince-Shampine Runge-Kutta method [2]. The above scheme avoids the calculation of the kernel, $k(t)$, on the negative real line.

4. References

[1] LUBICH, Ch.
On the stability of linear multi-step methods for Volterra convolution equations.
IMA J. Numer. Anal., 3, pp. 439-465, 1983.

[2] SHAMPINE, L.F.
Some practical Runge-Kutta formulas.
Math. Comput. 46(173), pp. 135-150, 1986.

[3] WOLKENFELT, P.H.M.
The construction of reducible quadrature rules for Volterra integral and integro-differential equations.
IMA J. Numer. Anal., 2, pp. 131-152, 1982.

5. Parameters

1: CK – ***real*** FUNCTION, supplied by the user. *External Procedure*

CK must evaluate the kernel $k(t)$ of the integral equation (1).

Its specification is:

```
real FUNCTION CK(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CK must be declared as EXTERNAL in the (sub)program from which D05BAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: CG – ***real*** FUNCTION, supplied by the user. *External Procedure*

CG must evaluate the function $g(s,y(s))$ in (1).

Its specification is:

```
real FUNCTION CG(S, Y)
real         S, Y
```

1: S – ***real***. *Input*

On entry: the value of the independent variable, s.

2: Y – ***real***. *Input*

On entry: the value of the solution y at the point S.

CG must be declared as EXTERNAL in the (sub)program from which D05BAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: CF – ***real*** FUNCTION, supplied by the user. *External Procedure*

CF must evaluate the function $f(t)$ in (1).

Its specification is:

```
real FUNCTION CF(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CF must be declared as EXTERNAL in the (sub)program from which D05BAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: METHOD – CHARACTER*1. *Input*

On entry: the type of method which the user wishes to employ.

For Adams type formulae, METHOD = 'A' or 'a'.

For backward differentiation formulae, METHOD = 'B' or 'b'.

Constraint: METHOD = 'A', 'a', 'B' or 'b'.

5: IORDER – INTEGER. *Input*

On entry: the order of the method to be used.

Constraints: if METHOD = 'A' or 'a', $3 \leq$ IORDER ≤ 6,
if METHOD = 'B' or 'b', $2 \leq$ IORDER ≤ 5.

6: ALIM – ***real***. *Input*

On entry: the lower limit of the integration interval, a.

Constraint: ALIM $\geq$ 0.0.

7: TLIM – ***real***. *Input*

On entry: the final point of the integration interval, T.

Constraint: TLIM > ALIM.

8: YN(NMESH) – ***real*** array. *Output*

On exit: YN(i) contains the approximate value of the true solution $y(t)$ at the specified point t = ALIM + i×H, for i = 1,2,...,NMESH, where H = (TLIM–ALIM)/NMESH.

9: ERREST(NMESH) – ***real*** array. *Output*

On exit: ERREST(i) contains the estimated value of the relative error in the computed solution at the point t = ALIM + i×H, for i = 1,2,...,NMESH, where H = (TLIM–ALIM)/NMESH.

10: NMESH – INTEGER. *Input*

On entry: the number of equi-distant points at which the solution is sought.

Constraints: if METHOD = 'A' or 'a', NMESH $\geq$ IORDER – 1,
if METHOD = 'B' or 'b', NMESH $\geq$ IORDER.

11: TOL – ***real***. *Input*

On entry: the relative accuracy required in the computed values of the solution.

Constraint: $\sqrt{\varepsilon}$ < TOL < 1.0, where ε is the ***machine precision***.

12: THRESH – ***real***. *Input*

On entry: the threshold value for use in the evalution of the estimated relative errors. For two successive meshes the following condition must hold at each point of the coarser mesh

$$\frac{|Y_1 - Y_2|}{\max(|Y_1|,|Y_2|,|\text{THRESH}|)} \leq \text{TOL},$$

where Y_1 is the computed solution on the coarser mesh and Y_2 is the computed solution at the corresponding point in the finer mesh. If this condition is not satisfied then the stepsize is halved and the solution is recomputed.

Note: THRESH can be used to effect a relative, absolute or mixed error test. If THRESH = 0.0 then pure relative error is measured and, if the computed solution is small and THRESH = 1.0, absolute error is measured.

13: WORK(LWK) – ***real*** array. *Workspace*

14: LWK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D05BAF is called.

Constraint: LWK $\geq$ 10×NMESH + 6.

Note: the above value of LWK is sufficient for D05BAF to perform only one extrapolation on the initial mesh as defined by NMESH. In general much more workspace is required and in the case when a large stepsize is supplied (i.e. NMESH is small), the user must provide a considerably larger workspace.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, METHOD ≠ 'A', 'a', 'B' or 'b',
or IORDER < 2 or IORDER > 6,
or METHOD = 'A' or 'a' and IORDER = 2,
or METHOD = 'B' or 'b' and IORDER = 6,
or ALIM < 0,
or TLIM ≤ ALIM,
or TOL < $\sqrt{\varepsilon}$ or TOL > 1.0, where ε is the ***machine precision***.

IFAIL = 2

On entry, NMESH ≤ IORDER – 2, when METHOD = 'A' or 'a',
or NMESH ≤ IORDER – 1, when METHOD = 'B' or 'b'.

IFAIL = 3

On entry, LWK < 10×NMESH + 6.

IFAIL = 4

The solution of the nonlinear equation (2) (see below) could not be computed by C05AVF and C05AZF.

IFAIL = 5

The size of the workspace LWK is too small for the required accuracy. The computation has failed in its initial phase (see below).

IFAIL = 6

The size of the workspace LWK is too small for the required accuracy on the interval [ALIM,TLIM] (see below).

7. Accuracy

The accuracy depends on TOL, the theoretical behaviour of the solution of the integral equation, the interval of integration and on the method being used. It can be controlled by varying TOL and THRESH; the user is recommended to choose a smaller value for TOL, the larger the value of IORDER.

Users are warned not to supply a very small TOL, because the required accuracy may never be achieved. This will usually force an error exit with IFAIL = 5 or IFAIL = 6.

In general, the higher the order of the method, the faster the required accuracy is achieved with less workspace. For non-stiff problems (see below) the users are recommended to use the Adams method (METHOD = 'A') of order greater than 4 (IORDER > 4).

8. Further Comments

When solving (1), the solution of a nonlinear equation of the form

$$Y_n - \alpha g(t_n, Y_n) - \Psi_n = 0, \qquad (2)$$

is required, where Ψ_n and α are constants. D05BAF calls C05AVF to find an interval for the zero of this equation followed by C05AZF to find its zero.

There is an initial phase of the algorithm where the solution is computed only for the first few points of the mesh. The exact number of these points depends on IORDER and METHOD. The stepsize is halved until the accuracy requirements are satisfied on these points and only then the solution on the whole mesh is computed. During this initial phase, if LWK is too small, D05BAF will exit with IFAIL = 5.

In the case IFAIL = 4 or IFAIL = 5, the user may be dealing with a 'stiff' equation; an equation where the Lipschitz constant L of the function $g(t,y)$ in (1) with respect to its second argument is large, viz,

$$|g(t,u)-g(t,v)| \leq L|u-v|. \tag{3}$$

In this case, if a BDF method (METHOD = 'B') has been used, the user is recommended to choose a smaller stepsize by increasing the value of NMESH, or provide a larger workspace. But, if an Adams method (METHOD = 'A') has been selected, the user is recommended to switch to a BDF method instead.

In the case IFAIL = 6, the specified accuracy has not been attained but ERREST and YN contain the most recent approximation to the computed solution and the corresponding error estimate. In this case, the error message informs the user of the number of extrapolations performed and the size of LWK required for the algorithm to proceed further.

On a successful exit, or with IFAIL = 6, the user may wish to examine the contents of the workspace WORK. Specifically, for i = 1,2,...,N, where N = int(int((LWK–6)/5)/2) + 1, WORK(i+N) and WORK(i) contain the computed approximation to the solution and its error estimate respectively at the point t = ALIM + $\frac{(i-1)}{\mathrm{N}}$×(TLIM–ALIM).

9. Example

Consider the following integral equation

$$y(t) = e^{-t} + \int_0^t e^{-(t-s)}[y(s) + e^{-y(s)}]ds, \qquad 0 \leq t \leq 20, \tag{4}$$

with the solution $y(t) = \ln(t+e)$. In this example, the Adams method of order 6 is used to solve this equation with TOL = 1.E–4.

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.

```
*     D05BAF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LWK, NMESH
      PARAMETER        (LWK=1000,NMESH=6)
*     .. Local Scalars ..
      real             ALIM, H, THRESH, TLIM, TOL
      INTEGER          I, IFAIL, IORDER
      CHARACTER        METHOD
*     .. Local Arrays ..
      real             ERRST(NMESH), WORK(LWK), YN(NMESH)
*     .. External Functions ..
      real             CF, CG, CK, SOL, X02AJF
      EXTERNAL         CF, CG, CK, SOL, X02AJF
*     .. External Subroutines ..
      EXTERNAL         D05BAF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05BAF Example Program Results'
      METHOD = 'A'
      IORDER = 6
      ALIM = 0.e0
      TLIM = 20.e0
      H = (TLIM-ALIM)/NMESH
      TOL = 1.e-4
      THRESH = X02AJF()
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Size of workspace =', LWK
      WRITE (NOUT,99998) 'Tolerance         =', TOL
      WRITE (NOUT,*)
      IFAIL = 0
*
      CALL D05BAF(CK,CG,CF,METHOD,IORDER,ALIM,TLIM,YN,ERRST,NMESH,TOL,
     +            THRESH,WORK,LWK,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*)
     +'    T         Approx. Sol.  True Sol.    Est. Error    Actual Error
     +'
         WRITE (NOUT,99997) (ALIM+I*H,YN(I),SOL(I*H),ERRST(I),ABS((YN(I)
     +     -SOL(I*H))/SOL(I*H)),I=1,NMESH)
      END IF
      STOP
*
99999 FORMAT (1X,A,I12)
99998 FORMAT (1X,A,e12.4)
99997 FORMAT (F7.2,2F14.5,2e15.5)
      END
*
      real  FUNCTION SOL(T)
*     .. Scalar Arguments ..
      real                T
*     .. Intrinsic Functions ..
      INTRINSIC           EXP, LOG
*     .. Executable Statements ..
      SOL = LOG(T+EXP(1.e0))
      RETURN
      END
*
      real  FUNCTION CF(T)
*     .. Scalar Arguments ..
      real                T
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      CF = EXP(-T)
      RETURN
      END
*
      real  FUNCTION CK(T)
*     .. Scalar Arguments ..
      real                T
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      CK = EXP(-T)
      RETURN
      END
*
```

```
      real  FUNCTION CG(S,Y)
*     .. Scalar Arguments ..
      real                S, Y
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      CG = Y + EXP(-Y)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05BAF Example Program Results

Size of workspace =        1000
Tolerance         =  0.1000E-03

   T        Approx. Sol.  True Sol.    Est. Error    Actual Error
 3.33        1.80033       1.80033    0.22632E-05    0.47827E-07
 6.67        2.23911       2.23911    0.38345E-05    0.75399E-07
10.00        2.54304       2.54304    0.51505E-05    0.99476E-07
13.33        2.77581       2.77581    0.63970E-05    0.12264E-06
16.67        2.96450       2.96450    0.76404E-05    0.12346E-06
20.00        3.12317       3.12317    0.89901E-05    0.77326E-08
```

D05BDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05BDF computes the solution of a weakly singular nonlinear convolution Volterra-Abel integral equation of the second kind using a fractional Backward Differentiation Formulae (BDF) method.

2. Specification

```
SUBROUTINE D05BDF (CK, CF, CG, INITWT, IORDER, TLIM, TOLNL, NMESH, YN,
1                  WORK, LWK, NCT, IFAIL)
INTEGER          IORDER, NMESH, LWK, NCT(NMESH/32+1), IFAIL
real             CK, CF, CG, TLIM, TOLNL, YN(NMESH), WORK(LWK)
CHARACTER*1      INITWT
EXTERNAL         CK, CF, CG
```

3. Description

D05BDF computes the numerical solution of the weakly singular convolution Volterra-Abel integral equation of the second kind

$$y(t) = f(t) + \frac{1}{\sqrt{\pi}} \int_0^t \frac{k(t-s)}{\sqrt{t-s}} g(s,y(s))ds, \qquad 0 \le t \le T. \tag{1}$$

Note the constant $\frac{1}{\sqrt{\pi}}$ in (1). It is assumed that the functions involved in (1) are sufficiently smooth.

The routine uses a fractional BDF linear multi-step method selected by the user to generate a family of quadrature rules (see D05BYF). The BDF methods available in D05BDF are of orders 4, 5 and 6 (= p say). For a description of theoretical and practical background related to these methods we refer to [3] and [1,2] respectively.

The algorithm is based on computing the solution $y(t)$ in a step-by-step fashion on a mesh of equispaced points. The size of the mesh is given by $T/(N-1)$, N being the number of points at which the solution is sought. These methods require $2p - 1$ (including $y(0)$) starting values which are evaluated internally. The computation of the lag term arising from the discretization of (1) is performed by fast Fourier transform (FFT) techniques when $N > 32 + 2p - 1$, and directly otherwise. The routine does not provide an error estimate and users are advised to check the behaviour of the solution with a different value of N. An option is provided which avoids the re-evaluation of the fractional weights when D05BDF is to be called several times (with the same value of N) within the same program unit with different functions.

4. References

[1] BAKER, C.T.H. and DERAKHSHAN, M.S.
FFT Techniques in the Numerical Solution of Convolution Equations.
J. Comp. Appl. Math. 20, pp. 5-24, 1987.

[2] HAIRER, E., LUBICH, Ch. and SCHLICHTE, M.
Fast Numerical Solution of Weakly Singular Volterra Integral Equations.
J. Comp. Appl. Math. 23, pp. 87-98, 1988.

[3] LUBICH, Ch.
Fractional Linear Multistep Methods for Abel-Volterra Integral Equations of the Second Kind.
Math. Comp. 45, pp. 463-469, 1985.

5. Parameters

1: CK – ***real*** FUNCTION, supplied by the user. *External Procedure*

CK must evaluate the kernel $k(t)$ of the integral equation (1).

Its specification is:

```
real FUNCTION CK(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CK must be declared as EXTERNAL in the (sub)program from which D05BDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: CF – ***real*** FUNCTION, supplied by the user. *External Procedure*

CF must evaluate the function $f(t)$ in (1).

Its specification is:

```
real FUNCTION CF(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CF must be declared as EXTERNAL in the (sub)program from which D05BDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: CG – ***real*** FUNCTION, supplied by the user. *External Procedure*

CG must evaluate the function $g(s,y(s))$ in (1).

Its specification is:

```
real FUNCTION CG(S, Y)
real         S, Y
```

1: S – ***real***. *Input*

On entry: the value of the independent variable, s.

2: Y – ***real***. *Input*

On entry: the value of the solution y at the point s.

CG must be declared as EXTERNAL in the (sub)program from which D05BDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: INITWT – CHARACTER*1. *Input*

On entry: if the fractional weights required by the method need to be calculated by the routine, then set INITWT = 'I' (Initial call).
If INITWT = 'S' (Subsequent call), then the routine assumes the fractional weights have been computed on a previous call and are stored in WORK.

Constraint: INITWT = 'I' or 'S'.

Note: When D05BDF is re-entered with the value of INITWT = 'S', the values of NMESH, IORDER and the contents of WORK **must** not be changed.

5: IORDER – INTEGER. *Input*

On entry: the order of the BDF method to be used, p.

Constraint: $4 \leq$ IORDER ≤ 6.

Suggested value: IORDER = 4.

6: TLIM – ***real***. *Input*

On entry: the final point of the integration interval, T.

Constraint: TLIM > 10×***machine precision***.

7: TOLNL – ***real***. *Input*

On entry: the accuracy required for the computation of the starting value and the solution of the nonlinear equation at each step of the computation (see Section 8).

Constraint: TOLNL > 10×***machine precision***.

Suggested value: TOLNL = $\sqrt{\textbf{\textit{machine precision}}}$.

8: NMESH – INTEGER. *Input*

On entry: the number of equispaced points, N, at which the solution is sought.

Constraint: NMESH = 2^m + 2×IORDER − 1, where $m \geq 1$.

9: YN(NMESH) – ***real*** array. *Output*

On exit: YN(i) contains the approximate value of the true solution $y(t)$ at the point $t = (i-1)\times h$, for i = 1,2,...,NMESH, where h = TLIM/(NMESH−1).

10: WORK(LWK) – ***real*** array. *Workspace*

11: LWK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D05BDF is called.

Constraint: LWK $\geq$ (2×IORDER+6)×NMESH + 8×IORDER^2 − 16×IORDER + 1.

12: NCT(NMESH/32+1) – INTEGER array. *Workspace*

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, IORDER < 4 or IORDER > 6,
or TLIM ≤ 10×***machine precision***,
or INITWT ≠ 'I' or 'S',
or INITWT = 'S' on the first call to D05BDF,
or TOLNL ≤ 10×***machine precision***,
or NMESH ≠ 2^m + 2×IORDER − 1, $m \geq 1$,
or LWK < (2×IORDER+6)×NMESH + 8×IORDER^2 − 15×IORDER + 1.

IFAIL = 2

The routine cannot compute the $2p-1$ starting values due to an error solving the system of nonlinear equations. Relaxing the value of TOLNL and/or increasing the value of NMESH may overcome this problem (see Section 8 for further details).

IFAIL = 3

The routine cannot compute the solution at a specific step due to an error in the solution of single nonlinear equation (2). Relaxing the value of TOLNL and/or increasing the value of NMESH may overcome this problem (see Section 8 for further details).

7. Accuracy

The accuracy depends on NMESH and TOLNL, the theoretical behaviour of the solution of the integral equation and the interval of integration. The value of TOLNL controls the accuracy required for computing the starting values and the solution of (2) at each step of computation. This value can affect the accuracy of the solution. However, for most problems, the value of $\sqrt{\textit{machine precision}}$ should be sufficient.

In general, for the choice of BDF method, the user is recommended to use the fourth order BDF formula (i.e. IORDER = 4).

8. Further Comments

In solving (1), initially, D05BDF computes the solution of a system of nonlinear equations for obtaining the $2p - 1$ starting values. C05NDF is used for this purpose. When a failure with IFAIL = 2 occurs (which corresponds to an error exit from C05NDF), users are advised to either relax the value of TOLNL or choose a smaller step size by increasing the value of NMESH. Once the starting values are computed successfully, the solution of a nonlinear equation of the form

$$Y_n - \alpha g(t_n, Y_n) - \Psi_n = 0, \tag{2}$$

is required at each step of computation, where Ψ_n and α are constants. D05BDF calls C05AXF to find the root of this equation.

If a failure with IFAIL = 3 occurs (which corresponds to an error exit from C05AXF), users are advised to relax the value of the TOLNL or choose a smaller step size by increasing the value of NMESH.

If a failure with IFAIL = 2 or 3 persists even after adjustments to TOLNL and/or NMESH then the user should consider whether there is a more fundamental difficulty. For example, the problem is ill-posed or the functions in (1) are not sufficiently smooth.

9. Example

We solve the following integral equations

$$y(t) = \sqrt{t} + \frac{3}{8}\pi t^2 - \int_0^t \frac{1}{\sqrt{t-s}}[y(s)]^3 ds, \qquad 0 \le t \le 7,$$

with the solution $y(t) = \sqrt{t}$, and

$$y(t) = (3-t)\sqrt{t} - \int_0^t \frac{1}{\sqrt{t-s}}\exp(s(1-s)^2 - [y(s)]^2) ds, \qquad 0 \le t \le 5,$$

with the solution $y(t) = (1-t)\sqrt{t}$. In the above examples, the fourth order BDF is used, and NMESH is set to $2^6 + 7$.

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.

```
*     D05BDF Example Program Text
*     Mark 16 Release. NAG Copyright 1992.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          IORDER, NMESH, LCT, LWK
      PARAMETER        (IORDER=4,NMESH=2**6+2*IORDER-1,LCT=NMESH/32+1,
     +                 LWK=(2*IORDER+6)*NMESH+8*IORDER*IORDER-
     +                 16*IORDER+1)
*     .. Local Scalars ..
      real             ERR, ERRMAX, H, HI1, SOLN, T, TLIM, TOLNL
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             WORK(LWK), YN(NMESH)
      INTEGER          NCT(LCT)
*     .. External Functions ..
      real             CF1, CF2, CG1, CG2, CK1, CK2, X02AJF
      EXTERNAL         CF1, CF2, CG1, CG2, CK1, CK2, X02AJF
*     .. External Subroutines ..
      EXTERNAL         D05BDF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, real, MOD, SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05BDF Example Program Results'
      WRITE (NOUT,*)
      IFAIL = 0
      TLIM = 7.0e0
      TOLNL = SQRT(X02AJF())
      H = TLIM/(NMESH-1)
*
      CALL D05BDF(CK1,CF1,CG1,'Initial',IORDER,TLIM,TOLNL,NMESH,YN,WORK,
     +            LWK,NCT,IFAIL)
*
      WRITE (NOUT,*) 'Example 1'
      WRITE (NOUT,*)
      WRITE (NOUT,99997) H
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     T         Approximate'
      WRITE (NOUT,*) '                 Solution '
      WRITE (NOUT,*)
*
      ERRMAX = 0.0e0
      DO 20 I = 1, NMESH
         HI1 = real(I-1)*H
         ERR = ABS(YN(I)-SQRT(HI1))
         IF (ERR.GT.ERRMAX) THEN
            ERRMAX = ERR
            T = HI1
            SOLN = YN(I)
         END IF
         IF (MOD(I,5).EQ.1) WRITE (NOUT,99998) HI1, YN(I)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ERRMAX, T, SOLN
*
*
      TLIM = 5.0e0
      H = TLIM/(NMESH-1)
*
      CALL D05BDF(CK2,CF2,CG2,'Subsequent',IORDER,TLIM,TOLNL,NMESH,YN,
     +            WORK,LWK,NCT,IFAIL)
*
```

```
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
      WRITE (NOUT,99997) H
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      T         Approximate'
      WRITE (NOUT,*) '                  Solution '
      WRITE (NOUT,*)
*
      ERRMAX = 0.0e0
      DO 40 I = 1, NMESH
         HI1 = real(I-1)*H
         ERR = ABS(YN(I)-(1.0e0-HI1)*SQRT(HI1))
         IF (ERR.GT.ERRMAX) THEN
            ERRMAX = ERR
            T = HI1
            SOLN = YN(I)
         END IF
         IF (MOD(I,7).EQ.1) WRITE (NOUT,99998) HI1, YN(I)
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ERRMAX, T, SOLN
*
      STOP
*
99999 FORMAT (' The maximum absolute error, ',E10.2,', occurred at T =',
     +       F8.4,/' with solution ',F8.4,/)
99998 FORMAT (1X,F8.4,F15.4)
99997 FORMAT (' The stepsize h = ',F8.4)
      END
*
*
      real  FUNCTION CK1(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                PI
*     .. External Functions ..
      real                X01AAF
      EXTERNAL            X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
*     .. Executable Statements ..
      CK1 = -SQRT(X01AAF(PI))
      RETURN
      END
*
*
      real  FUNCTION CF1(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                PI
*     .. External Functions ..
      real                X01AAF
      EXTERNAL            X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
*     .. Executable Statements ..
      CF1 = SQRT(T) + (3.0e0/8.0e0)*T*T*X01AAF(PI)
      RETURN
      END
*
*
      real  FUNCTION CG1(S,Y)
*     .. Scalar Arguments ..
      real                S, Y
*     .. Executable Statements ..
      CG1 = Y*Y*Y
      RETURN
      END
```

```
*
*
      real  FUNCTION CK2(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                PI
*     .. External Functions ..
      real                X01AAF
      EXTERNAL            X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
*     .. Executable Statements ..
      CK2 = -SQRT(X01AAF(PI))
      RETURN
      END
*
*
      real  FUNCTION CF2(T)
*     .. Scalar Arguments ..
      real                T
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
*     .. Executable Statements ..
*
      CF2 = (3.0e0-T)*SQRT(T)
      RETURN
      END
*
*
      real  FUNCTION CG2(S,Y)
*     .. Scalar Arguments ..
      real                S, Y
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      CG2 = EXP(S*(1.0e0-S)*(1.0e0-S)-Y*Y)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D05BDF Example Program Results

 Example 1

 The stepsize h =   0.1000

      T        Approximate
                 Solution

   0.0000          0.0000
   0.5000          0.7071
   1.0000          1.0000
   1.5000          1.2247
   2.0000          1.4142
   2.5000          1.5811
   3.0000          1.7321
   3.5000          1.8708
   4.0000          2.0000
   4.5000          2.1213
   5.0000          2.2361
   5.5000          2.3452
   6.0000          2.4495
   6.5000          2.5495
   7.0000          2.6458
```

```
The maximum absolute error,    0.94E-08, occurred at T =  0.9000
with solution   0.9487

Example 2

The stepsize h =   0.0714

     T         Approximate
                 Solution

  0.0000          0.0000
  0.5000          0.3536
  1.0000          0.0000
  1.5000         -0.6124
  2.0000         -1.4142
  2.5000         -2.3717
  3.0000         -3.4641
  3.5000         -4.6771
  4.0000         -6.0000
  4.5000         -7.4246
  5.0000         -8.9443

The maximum absolute error,    0.42E-07, occurred at T =  4.3571
with solution  -7.0076
```

D05BEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05BEF computes the solution of a weakly singular nonlinear convolution Volterra-Abel integral equation of the first kind using a fractional Backward Differentiation Formulae (BDF) method.

2. Specification

```
SUBROUTINE D05BEF (CK, CF, CG, INITWT, IORDER, TLIM, TOLNL, NMESH, YN,
1                  WORK, LWK, NCT, IFAIL)
INTEGER          IORDER, NMESH, LWK, NCT(NMESH/32+1), IFAIL
real             CK, CF, CG, TLIM, TOLNL, YN(NMESH), WORK(LWK)
CHARACTER*1      INITWT
EXTERNAL         CK, CF, CG
```

3. Description

D05BEF computes the numerical solution of the weakly singular convolution Volterra-Abel integral equation of the first kind

$$f(t) + \frac{1}{\sqrt{\pi}} \int_0^t \frac{k(t-s)}{\sqrt{t-s}} g(s,y(s))ds = 0, \qquad 0 \le t \le T. \tag{1}$$

Note the constant $\frac{1}{\sqrt{\pi}}$ in (1). It is assumed that the functions involved in (1) are sufficiently smooth and if

$$f(t) = t^{\beta} w(t) \text{ with } \beta > -\tfrac{1}{2}, \tag{2}$$

then the solution $y(t)$ is unique and has the form $y(t) = t^{\beta-1/2} z(t)$, (see [4]). It is evident from (1) that $f(0) = 0$. The user is required to provide the value of $y(t)$ at $t = 0$. If $y(0)$ is unknown, Section 8 gives a description of how an approximate value can be obtained.

The routine uses a fractional BDF linear multi-step method selected by the user to generate a family of quadrature rules (see D05BYF). The BDF methods available in D05BEF are of orders 4, 5 and 6 ($= p$ say). For a description of the theoretical and practical background related to these methods we refer to [4] and [1,3] respectively.

The algorithm is based on computing the solution $y(t)$ in a step-by-step fashion on a mesh of equispaced points. The size of the mesh is given by $T/(N-1)$, N being the number of points at which the solution is sought. These methods require $2p - 2$ starting values which are evaluated internally. The computation of the lag term arising from the discretization of (1) is performed by fast Fourier transform (FFT) techniques when $N > 32 + 2p - 1$, and directly otherwise. The routine does not provide an error estimate and users are advised to check the behaviour of the solution with a different value of N. An option is provided which avoids the re-evaluation of the fractional weights when D05BEF is to be called several times (with the same value of N) within the same program with different functions.

4. References

[1] BAKER, C.T.H. and DERAKHSHAN, M.S.
FFT Techniques in the Numerical Solution of Convolution Equations.
J. Comp. Appl. Math. 20, pp. 5-24, 1987.

[2] GORENFLO, R. and PFEIFFER, A.
On Analysis and Discretization of Nonlinear Abel Integral Equations of First Kind.
ACTA Mathematica Vietnamica, 16, pp. 211-262, 1991.

[3] HAIRER, E., LUBICH, Ch. and SCHLICHTE, M.
Fast Numerical Solution of Weakly Singular Volterra Integral Equations.
J. Comp. Appl. Math. 23, pp. 87-98, 1988.

[4] LUBICH, Ch.
Fractional Linear Multistep Methods for Abel-Volterra Integral Equations of the First Kind.
IMA J. Numer. Anal. 7, pp. 97-106, 1987.

5. Parameters

1: CK – ***real*** FUNCTION, supplied by the user. *External Procedure*

CK must evaluate the kernel $k(t)$ of the integral equation (1).

Its specification is:

```
real FUNCTION CK(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CK must be declared as EXTERNAL in the (sub)program from which D05BEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: CF – ***real*** FUNCTION, supplied by the user. *External Procedure*

CF must evaluate the function $f(t)$ in (1).

Its specification is:

```
real FUNCTION CF(T)
real         T
```

1: T – ***real***. *Input*

On entry: the value of the independent variable, t.

CF must be declared as EXTERNAL in the (sub)program from which D05BEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: CG – ***real*** FUNCTION, supplied by the user. *External Procedure*

CG must evaluate the function $g(s,y(s))$ in (1).

Its specification is:

```
real FUNCTION CG(S, Y)
real         S, Y
```

1: S – ***real***. *Input*

On entry: the value of the independent variable, s.

2: Y – ***real***. *Input*

On entry: the value of the solution y at the point s.

CG must be declared as EXTERNAL in the (sub)program from which D05BEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: INITWT – CHARACTER*1. *Input*

On entry: if the fractional weights required by the method need to be calculated by the routine, then set INITWT = 'I' (Initial call).
If INITWT = 'S' (Subsequent call), then the routine assumes the fractional weights have been computed by a previous call and are stored in WORK.

Constraint: INITWT = 'I' or 'S'.

Note: When D05BEF is re-entered with a value of INITWT = 'S', the values of NMESH, IORDER and the contents of WORK **must** not be changed.

5: IORDER – INTEGER. *Input*

On entry: the order of the BDF method to be used, p.

Constraint: $4 \leq$ IORDER ≤ 6.

Suggested value: IORDER = 4.

6: TLIM – ***real***. *Input*

On entry: the final point of the integration interval, T.

Constraint: TLIM > 10×***machine precision***.

7: TOLNL – ***real***. *Input*

On entry: the accuracy required for the computation of the starting value and the solution of the nonlinear equation at each step of the computation (see Section 8).

Constraint: TOLNL > 10×***machine precision***.

Suggested value: TOLNL = $\sqrt{\textbf{machine precision}}$.

8: NMESH – INTEGER. *Input*

On entry: the number of equispaced points, N, at which the solution is sought.

Constraint: NMESH = 2^m + 2×IORDER – 1, where $m \geq 1$.

9: YN(NMESH) – ***real*** array. *Input/Output*

On entry: YN(1) must contain the value of $y(t)$ at $t = 0$ (see Section 8).

On exit: YN(i) contains the approximate value of the true solution $y(t)$ at the point $t = (i-1)\times h$, for i = 1,2,...,NMESH, where h = TLIM/(NMESH–1).

10: WORK(LWK) – ***real*** array. *Workspace*

11: LWK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D05BEF is called.

Constraint: LWK $\geq$ (2×IORDER+6)×NMESH + 8×IORDER^2 – 16×IORDER + 1.

12: NCT(NMESH/32+1) – INTEGER array. *Workspace*

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, IORDER < 4 or IORDER > 6,
or TLIM ≤ 10×***machine precision***,
or INITWT ≠ 'I' or 'S',
or INITWT = 'S' on the first call to D05BEF,
or TOLNL ≤ 10×***machine precision***,
or NMESH ≠ 2^m + 2×IORDER − 1, $m \geq 1$,
or LWK < (2×IORDER+6)×NMESH + 8×IORDER^2 − 16×IORDER + 1.

IFAIL = 2

The routine cannot compute the $2p - 2$ starting values due to an error in solving the system of nonlinear equations. Relaxing the value of TOLNL and/or increasing the value of NMESH may overcome this problem (see Section 8 for further details).

IFAIL = 3

The routine cannot compute the solution at a specific step due to an error in the solution of single nonlinear equation (3). Relaxing the value of TOLNL and/or increasing the value of NMESH may overcome this problem (see Section 8 for further details).

7. Accuracy

The accuracy depends on NMESH and TOLNL, the theoretical behaviour of the solution of the integral equation and the interval of integration. The value of TOLNL controls the accuracy required for computing the starting values and the solution of (3) at each step of computation. This value can affect the accuracy of the solution. However, for most problems, the value of $\sqrt{\textbf{\textit{machine precision}}}$ should be sufficient.

In general, for the choice of BDF method, the user is recommended to use the fourth order BDF formula (i.e. IORDER = 4).

8. Further Comments

Also when solving (1) the initial value $y(0)$ is required. This value may be computed from the limit relation (see [2])

$$\frac{-2}{\sqrt{\pi}}k(0)\ g(0,y(0)) = \lim_{t\to 0}\frac{f(t)}{\sqrt{t}}. \tag{3}$$

If the value of the above limit is known then by solving the nonlinear equation (3) an approximation to $y(0)$ can be computed. If the value of the above limit is not known, an approximation should be provided. Following the analysis presented in [2], the following pth order approximation can be used:

$$\lim_{t\to 0}\frac{f(t)}{\sqrt{t}} \simeq \frac{f(h^P)}{h^{P/2}}. \tag{4}$$

However, it must be emphasized that the approximation in (4) may result in an amplification of the rounding errors and hence users are advised (if possible) to determine $\lim_{t\to 0}\frac{f(t)}{\sqrt{t}}$ by analytical methods.

Also when solving (1), initially, D05BEF computes the solution of a system of nonlinear equation for obtaining the $2p - 2$ starting values. C05NDF is used for this purpose. If a failure with IFAIL = 2 occurs (corresponding to an error exit from C05NDF), users are advised to either relax the value of TOLNL or choose a smaller step size by increasing the value of NMESH. Once the starting values are computed successfully, the solution of a nonlinear equation of the form

$$Y_n - \alpha g(t_n,Y_n) - \Psi_n = 0, \tag{5}$$

is required at each step of computation, where Ψ_n and α are constants. D05BEF calls C05AXF to find the root of this equation.

When a failure with IFAIL = 3 occurs (which corresponds to an error exit from C05AXF), users are advised to either relax the value of the TOLNL or choose a smaller step size by increasing the value of NMESH.

If a failure with IFAIL = 2 or 3 persists even after adjustments to TOLNL and/or NMESH then the user should consider whether there is a more fundamental difficulty. For example, the problem is ill-posed or the functions in (1) are not sufficiently smooth.

9. Example

We solve the following integral equations.

Example 1:

The density of the probability that a Brownian motion crosses a one-sided moving boundary $a(t)$ before time t, satisfies the integral equation (see [3])

$$-\frac{1}{\sqrt{t}}\exp\left(\frac{1}{2}-\{a(t)\}^2/t\right) + \int_0^t \frac{\exp\left(-\frac{1}{2}\{a(t)-a(s)\}^2/(t-s)\right)}{\sqrt{t-s}} y(s)\ ds = 0, \quad 0 \le t \le 7.$$

In the case of a straight line $a(t) = 1 + t$, the exact solution is known to be

$$y(t) = \frac{1}{\sqrt{2\pi t^3}}\exp\{-(1+t)^2/2t\}$$

Example 2:

In this example we consider the equation

$$-\frac{2\log(\sqrt{1+t}+\sqrt{t})}{\sqrt{1+t}} + \int_0^t \frac{y(s)}{\sqrt{t-s}} ds = 0, \quad 0 \le t \le 5.$$

The solution is given by $y(t) = \dfrac{1}{1+t}$.

In the above examples, the fourth order BDF is used, and NMESH is set to $2^6 + 7$.

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.

```
*     D05BEF Example Program Text
*     Mark 16 Release. NAG Copyright 1992.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          IORDER, NMESH, LCT, LWK
      PARAMETER        (IORDER=4,NMESH=2**6+2*IORDER-1,LCT=NMESH/32+1,
     +                 LWK=(2*IORDER+6)*NMESH+8*IORDER*IORDER-
     +                 16*IORDER+1)
*     .. Local Scalars ..
      real             ERR, ERRMAX, H, HI1, SOLN, T, TLIM, TOLNL
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             WORK(LWK), YN(NMESH)
      INTEGER          NCT(LCT)
*     .. External Functions ..
      real             CF1, CF2, CG1, CG2, CK1, CK2, SOL1, SOL2, X02AJF
      EXTERNAL         CF1, CF2, CG1, CG2, CK1, CK2, SOL1, SOL2, X02AJF
*     .. External Subroutines ..
      EXTERNAL         D05BEF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, real, MOD, SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05BEF Example Program Results'
      WRITE (NOUT,*)
      IFAIL = 0
```

```
      TLIM = 7.0e0
      TOLNL = SQRT(X02AJF())
      H = TLIM/(NMESH-1)
*
      YN(1) = 0.0e0
*
      CALL D05BEF(CK1,CF1,CG1,'Initial',IORDER,TLIM,TOLNL,NMESH,YN,WORK,
     +            LWK,NCT,IFAIL)
*
      WRITE (NOUT,*) 'Example 1'
      WRITE (NOUT,*)
      WRITE (NOUT,99997) H
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     T         Approximate'
      WRITE (NOUT,*) '                 Solution '
      WRITE (NOUT,*)
*
      ERRMAX = 0.0e0
      DO 20 I = 2, NMESH
         HI1 = real(I-1)*H
         ERR = ABS(YN(I)-SOL1(HI1))
         IF (ERR.GT.ERRMAX) THEN
            ERRMAX = ERR
            T = HI1
            SOLN = YN(I)
         END IF
         IF (I.GT.5 .AND. MOD(I,5).EQ.1) WRITE (NOUT,99998) HI1, YN(I)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ERRMAX, T, SOLN
*
      WRITE (NOUT,*)
*
      TLIM = 5.0e0
      H = TLIM/(NMESH-1)
      YN(1) = 1.0e0
*
      CALL D05BEF(CK2,CF2,CG2,'Subsequent',IORDER,TLIM,TOLNL,NMESH,YN,
     +            WORK,LWK,NCT,IFAIL)
*
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
      WRITE (NOUT,99997) H
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     T         Approximate'
      WRITE (NOUT,*) '                 Solution '
      WRITE (NOUT,*)
*
      ERRMAX = 0.0e0
      DO 40 I = 1, NMESH
         HI1 = real(I-1)*H
         ERR = ABS(YN(I)-SOL2(HI1))
         IF (ERR.GT.ERRMAX) THEN
            ERRMAX = ERR
            T = HI1
            SOLN = YN(I)
         END IF
         IF (I.GT.7 .AND. MOD(I,7).EQ.1) WRITE (NOUT,99998) HI1, YN(I)
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ERRMAX, T, SOLN
*
      STOP
*
99999 FORMAT (' The maximum absolute error, ',E10.2,', occurred at T =',
     +       F8.4,/' with solution ',F8.4,/)
99998 FORMAT (1X,F8.4,F15.4)
99997 FORMAT (' The stepsize h = ',F8.4)
      END
*
```

```
*
      real  FUNCTION CK1(T)
*     .. Scalar Arguments ..
      real                T
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      CK1 = EXP(-0.5e0*T)
      RETURN
      END
*
*
      real  FUNCTION CF1(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                A, PI, T1
*     .. External Functions ..
      real                X01AAF
      EXTERNAL            X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC           EXP, SQRT
*     .. Executable Statements ..
      T1 = 1.0e0 + T
      A = 1.0e0/SQRT(X01AAF(PI)*T)
      CF1 = -A*EXP(-0.5e0*T1*T1/T)
      RETURN
      END
*
*
      real  FUNCTION CG1(S,Y)
*     .. Scalar Arguments ..
      real                S, Y
*     .. Executable Statements ..
      CG1 = Y
      RETURN
      END
*
*
      real  FUNCTION SOL1(T)
*     .. Scalar Arguments ..
      real                 T
*     .. Local Scalars ..
      real                 C, PI, T1
*     .. External Functions ..
      real                 X01AAF
      EXTERNAL             X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC            EXP, SQRT
*     .. Executable Statements ..
*
      T1 = 1.0e0 + T
      C = 1.0e0/SQRT(2.0e0*X01AAF(PI))
      SOL1 = C*(1.0e0/(T**1.5e0))*EXP(-T1*T1/(2.0e0*T))
      RETURN
      END
*
*
      real  FUNCTION CK2(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                PI
*     .. External Functions ..
      real                X01AAF
      EXTERNAL            X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
```

```
*     .. Executable Statements ..
      CK2 = SQRT(X01AAF(PI))
      RETURN
      END
*
*
      real  FUNCTION CF2(T)
*     .. Scalar Arguments ..
      real                T
*     .. Local Scalars ..
      real                ST1
*     .. Intrinsic Functions ..
      INTRINSIC           LOG, SQRT
*     .. Executable Statements ..
      ST1 = SQRT(1.0e0+T)
      CF2 = -2.0e0*LOG(ST1+SQRT(T))/ST1
      RETURN
      END
*
*
      real  FUNCTION CG2(S,Y)
*     .. Scalar Arguments ..
      real                S, Y
*     .. Executable Statements ..
      CG2 = Y
      RETURN
      END
*
*
      real  FUNCTION SOL2(T)
*     .. Scalar Arguments ..
      real                 T
*     .. Executable Statements ..
      SOL2 = 1.0e0/(1.0e0+T)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05BEF Example Program Results

Example 1

The stepsize h =   0.1000

     T        Approximate
                Solution

  0.5000          0.1191
  1.0000          0.0528
  1.5000          0.0265
  2.0000          0.0146
  2.5000          0.0086
  3.0000          0.0052
  3.5000          0.0033
  4.0000          0.0022
  4.5000          0.0014
  5.0000          0.0010
  5.5000          0.0007
  6.0000          0.0004
  6.5000          0.0003
  7.0000          0.0002

The maximum absolute error,   0.29E-02, occurred at T =  0.1000
with solution   0.0326
```

```
Example 2

The stepsize h =   0.0714

     T         Approximate
                 Solution

  0.5000          0.6667
  1.0000          0.5000
  1.5000          0.4000
  2.0000          0.3333
  2.5000          0.2857
  3.0000          0.2500
  3.5000          0.2222
  4.0000          0.2000
  4.5000          0.1818
  5.0000          0.1667

The maximum absolute error,   0.32E-05, occurred at T =  0.0714
with solution   0.9333
```

D05BWF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05BWF computes the quadrature weights associated with the Adams methods of orders three to six and the Backward Differentiation Formulae (BDF) methods of orders two to five. These rules, which are referred to as reducible quadrature rules, can then be used in the solution of Volterra integral and integro-differential equations.

2. Specification

```
      SUBROUTINE D05BWF (METHOD, IORDER, OMEGA, NOMG, LENSW, SW, LDSW,
     1                   NWT, IFAIL)
      INTEGER           IORDER, NOMG, LENSW, LDSW, NWT, IFAIL
      real              OMEGA(NOMG), SW(LDSW,NWT)
      CHARACTER*1       METHOD
```

3. Description

D05BWF computes the weights $W_{n,j}$ and ω_i for a family of quadrature rules related to the Adams methods of orders three to six and the BDF methods of orders two to five, for approximating the integral:

$$\int_0^t \phi(s)\ ds \simeq h \sum_{j=0}^{p-1} W_{n,j}\phi(jh) + h \sum_{j=p}^{n} \omega_{n-j}\phi(jh), \qquad 0 \leq t \leq T, \tag{1}$$

with $t = nh$, $(n \geq 0)$ for some given constant h.

In (1), h is a uniform mesh, p is related to the order of the method being used and $W_{n,j}$, ω_i are the starting and the convolution weights respectively. A description of how these weights can be used in the solution of a Volterra integral equations of the second kind is given in Section 8. For a general discussion of these methods, see [2] for more details.

4. References

[1] LAMBERT, J.D.
Computational Methods in Ordinary Differential Equations.
John Wiley, London, 1973.

[2] WOLKENFELT, P.H.M.
The construction of reducible quadrature rules for Volterra integral and integro-differential equations.
IMA J. Num. Anal., 2, pp. 131-152, 1982.

5. Parameters

1: METHOD – CHARACTER*1. *Input*

On entry: the type of method to be used.

For Adams type formulae set METHOD = 'A'.

For Backward Differentiation Formulae set METHOD = 'B'.

Constraint: METHOD = 'A' or 'B'.

2: IORDER – INTEGER. *Input*

On entry: the order of the method to be used.

Constraints: if METHOD = 'A', $3 \leq$ IORDER ≤ 6,
if METHOD = 'B', $2 \leq$ IORDER ≤ 5.

3: OMEGA(NOMG) – ***real*** array. *Output*

On exit: contains the first NOMG convolution weights.

4: NOMG – INTEGER. *Input*

On entry: the number of convolution weights.

Constraint: NOMG ≥ 1.

5: LENSW – INTEGER. *Output*

On exit: the number of rows in the weights $W_{i,j}$.

6: SW(LDSW,NWT) – ***real*** array. *Output*

On exit: SW(i,j+1) contains the weights $W_{i,j}$, for i = 1,2,...,LENSW; j = 0,1,...,NWT − 1.

7: LDSW – INTEGER. *Input*

On entry: the first dimension of the array SW as declared in the (sub)program from which D05BWF is called.

Constraints: if METHOD = 'A', LDSW ≥ NOMG + IORDER − 2,
if METHOD = 'B', LDSW ≥ NOMG + IORDER − 1.

8: NWT – INTEGER. *Input*

On entry: the number of columns in the starting weights, p.

Constraints: if METHOD = 'A', NWT = IORDER − 1,
if METHOD = 'B', NWT = IORDER.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, METHOD ≠ 'A' or 'B'.

IFAIL = 2

On entry, IORDER < 2 or IORDER > 6,
or NOMG < 1.

IFAIL = 3

On entry, METHOD = 'A' and IORDER = 2,
or METHOD = 'B' and IORDER = 6.

IFAIL = 4

On entry, METHOD = 'A' and NWT ≠ IORDER − 1,
or METHOD = 'B' and NWT ≠ IORDER.

IFAIL = 5

On entry, METHOD = 'A' and LDSW < NOMG + IORDER − 2,
or METHOD = 'B' and LDSW < NOMG + IORDER − 1.

7. Accuracy

None.

8. Further Comments

Reducible quadrature rules are most appropriate for solving Volterra integral equations (and integro-differential equations). In this section, we propose the following algorithm which the users may find useful in solving a linear Volterra integral equation of the form

$$y(t) = f(t) + \int_0^t K(t,s)y(s)ds, \qquad 0 \le t \le T, \tag{2}$$

using D05BWF. In (2), $K(t,s)$ and $f(t)$ are given and the solution $y(t)$ is sought on a uniform mesh of size h such that $T = Nh$. Discretization of (2) yields

$$y_n = f(nh) + h\sum_{j=0}^{p-1} W_{n,j}K(nh,jh)y_j + h\sum_{j=p}^{n} \omega_{n-j}K(nh,jh)y_j, \tag{3}$$

where $y_n \simeq y(nh)$. We propose the following algorithm for computing y_n from (3) after a call to D05BWF:

(a) Equation (3) requires starting values, y_j, for $j = 1,2,...,\text{NWT} - 1$, with $y_0 = f(0)$. These starting values can be computed by solving the linear system

$$y_n = f(nh) + h\sum_{j=0}^{\text{NWT}-1} \text{SW}(n,j+1)K(nh,jh)y_j, \qquad n = 1,2,...,\text{NWT} - 1.$$

(b) Compute the inhomogenous terms

$$\sigma_n = f(nh) + h\sum_{j=0}^{\text{NWT}-1} \text{SW}(n,j+1)K(nh,jh)y_j, \qquad n = \text{NWT},\text{NWT}+1,...,N.$$

(c) Start the iteration for $n = \text{NWT},\text{NWT}+1,...,N$ to compute y_n from:

$$(1-h\times\text{OMEGA}(1)K(nh,nh))y_n = \sigma_n + h\sum_{j=\text{NWT}}^{n-1} \text{OMEGA}(n-j+1)K(nh,jh)y_j.$$

Note that for a nonlinear integral equation, the solution of a nonlinear algebraic system is required at step (a) and a single nonlinear equation at step (c).

9. Example

The following example generates the first ten convolution and thirteen starting weights generated by the fourth order BDF method.

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.

```
*     D05BWF Example Program Text
*     Mark 16 Release. NAG Copyright 1992.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          IORDER, NOMG, NWT, LDSW
      PARAMETER        (IORDER=4,NOMG=10,NWT=IORDER,LDSW=NOMG+IORDER-1)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, LENSW, N
*     .. Local Arrays ..
      real             OMEGA(NOMG), SW(LDSW,NWT)
```

```
*     .. External Subroutines ..
      EXTERNAL         D05BWF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D05BWF Example Program Results'
      IFAIL = 0
*
      CALL D05BWF('BDF',IORDER,OMEGA,NOMG,LENSW,SW,LDSW,NWT,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The convolution weights'
      WRITE (NOUT,*)
*
      DO 20 N = 1, NOMG
         WRITE (NOUT,99999) N - 1, OMEGA(N)
   20 CONTINUE
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The weights W'
      WRITE (NOUT,*)
*
      DO 40 N = 1, LENSW
         WRITE (NOUT,99999) N, (SW(N,J),J=1,NWT)
   40 CONTINUE
*
      STOP
*
99999 FORMAT (1X,I3,4X,6F10.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05BWF Example Program Results

The convolution weights

  0        0.4800
  1        0.9216
  2        1.0783
  3        1.0504
  4        0.9962
  5        0.9797
  6        0.9894
  7        1.0003
  8        1.0034
  9        1.0017

The weights W

  1        0.3750    0.7917   -0.2083    0.0417
  2        0.3333    1.3333    0.3333    0.0000
  3        0.3750    1.1250    1.1250    0.3750
  4        0.4800    0.7467    1.5467    0.7467
  5        0.5499    0.5719    1.5879    0.8886
  6        0.5647    0.5829    1.5016    0.8709
  7        0.5545    0.6385    1.4514    0.8254
  8        0.5458    0.6629    1.4550    0.8098
  9        0.5449    0.6578    1.4741    0.8170
 10        0.5474    0.6471    1.4837    0.8262
 11        0.5491    0.6428    1.4831    0.8292
 12        0.5492    0.6438    1.4798    0.8279
 13        0.5488    0.6457    1.4783    0.8263
```

D05BYF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D05BYF computes the fractional quadrature weights associated with the Backward Differentiation Formulae (BDF) of orders 4, 5 and 6. These weights can then be used in the solution of weakly singular equations of Abel type.

2. Specification

```
SUBROUTINE D05BYF (IORDER, IQ, LENFW, WT, SW, LDSW, WORK, LWK, IFAIL)
INTEGER          IORDER, IQ, LENFW, LDSW, LWK, IFAIL
real             WT(LENFW), SW(LDSW,2*IORDER-1), WORK(LWK)
```

3. Description

D05BYF computes the weights $W_{n,j}$ and ω_i for a family of quadrature rules related to a BDF method for approximating the integral:

$$\frac{1}{\sqrt{\pi}}\int_0^t \frac{\phi(s)}{\sqrt{t-s}}\,ds \simeq \sqrt{h}\sum_{j=0}^{2p-2} W_{n,j}\phi(jh) + \sqrt{h}\sum_{j=2p-1}^{n}\omega_{n-j}\phi(jh), \qquad 0 \le t \le T, \tag{1}$$

with $t = nh$ $(n \geq 0)$, for some given h. In (1), p is the order of the BDF method used and $W_{n,j}$, ω_i are the fractional starting and the fractional convolution weights respectively. The algorithm for the generation of ω_i is based on Newton's iteration. Fast Fourier transform (FFT) techniques are used for computing these weights and subsequently $W_{n,j}$ (see [1] and [2] for practical details and [3] for theoretical details). Some special functions can be represented as the fractional integrals of simpler functions and fractional quadratures can be employed for their computation (see [3]). A description of how these weights can be used in the solution of weakly singular equations of Abel type is given in Section 8.

4. References

[1] BAKER, C.T.H. and DERAKHSHAN, M.S.
Computational Approximations to Some Power Series.
In: 'Approximation Theory', Eds Meinardus and Nurnberger.
ISNM, Vol. 81, pp. 11-20, 1987.

[2] HENRICI, P.
Fast Fourier Methods in Computational Complex Analysis.
SIAM Review 21, pp. 481-529, 1979.

[3] LUBICH, Ch.
Discretized Fractional Calculus.
SIAM J. Math. Anal. 17, pp. 704-719, 1986.

5. Parameters

1: IORDER – INTEGER. *Input*

On entry: the order of the BDF method to be used, p.

Constraint: $4 \leq$ IORDER ≤ 6.

2: IQ – INTEGER. *Input*

On entry: determines the number of weights to be computed. By setting IQ to a value, 2^{IQ+1} fractional convolution weights are computed.

Constraint: IQ ≥ 0.

3: LENFW – INTEGER. *Input*

On entry: the length of the array WT.

Constraint: LENFW $\geq 2^{IQ+2}$.

4: WT(LENFW) – ***real*** array. *Output*

On exit: the first 2^{IQ+1} elements of WT contains the fractional convolution weights ω_i, for $i = 0,1,...,2^{IQ+1}-1$. The remainder of the array is used as workspace.

5: SW(LDSW,2*IORDER−1) – ***real*** array. *Output*

On exit: SW(n,j+1) contains the fractional starting weights $W_{n-1,j}$, for $n = 1,2,...,(2^{IQ+1}+2\times IORDER-1)$; $j = 0,1,...,2\times IORDER - 2$.

6: LDSW – INTEGER. *Input*

On entry: the first dimension of the array SW as declared in the (sub)program from which D05BYF is called.

Constraint: LDSW $\geq 2^{IQ+1} + 2\times IORDER - 1$.

7: WORK(LWK) – ***real*** array. *Workspace*

8: LWK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D05BYF is called.

Constraint: LWK $\geq 2^{IQ+3}$.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, IORDER < 4 or IORDER > 6,
or IQ < 0,
or LENFW < 2^{IQ+2},
or LDSW < $2^{IQ+1} + 2\times IORDER - 1$,
or LWK < 2^{IQ+3}.

7. Accuracy

None.

8. Further Comments

Fractional quadrature weights can be used for solving weakly singular integral equations of Abel type. In this section, we propose the following algorithm which users may find useful in solving a linear weakly singular integral equation of the form

$$y(t) = f(t) + \frac{1}{\sqrt{\pi}} \int_0^t \frac{K(t,s)y(s)}{\sqrt{t-s}} ds, \qquad 0 \leq t \leq T, \tag{2}$$

using D05BYF. In (2), $K(t,s)$ and $f(t)$ are given and the solution $y(t)$ is sought on a uniform mesh of size h such that $T = Nh$. Discretization of (2) yields

$$y_n = f(nh) + \sqrt{h}\sum_{j=0}^{2p-2} W_{n,j}K(nh,jh)y_j + \sqrt{h}\sum_{j=2p-1}^{n} \omega_{n-j}K(nh,jh)y_j, \quad (3)$$

where $y_n \simeq y(nh)$. We propose the following algorithm for computing y_n from (3) after a call to D05BYF:

(a) Set $N = 2^{IQ+1} + 2\times\text{IORDER} - 2$ and $h = T/N$.

(b) Equation (3) requires $2\times\text{IORDER} - 2$ starting values, y_j, for $j = 1,2,...,2\times\text{IORDER} - 2$, with $y_0 = f(0)$. These starting values can be computed by solving the system

$$y_n = f(nh) + \sqrt{h}\sum_{j=0}^{2\times\text{IORDER}-2} \text{SW}(n+1,j+1)K(nh,jh)y_j,$$
$$n = 1,2,...,2\times\text{IORDER} - 2.$$

(c) Compute the inhomogeneous terms

$$\sigma_n = f(nh) + \sqrt{h}\sum_{j=0}^{2\times\text{IORDER}-2} \text{SW}(n+1,j+1)K(nh,jh)y_j,$$
$$n = 2\times\text{IORDER}-1,2\times\text{IORDER},...,N.$$

(d) Start the iteration for $n = 2\times\text{IORDER}-1,2\times\text{IORDER},...,N$ to compute y_n from:

$$(1-\sqrt{h}\text{WT}(1)K(nh,nh))y_n = \sigma_n + \sqrt{h}\sum_{j=2\times\text{IORDER}-1}^{n-1} \text{WT}(n-j+1)K(nh,jh)y_j.$$

Note that for nonlinear weakly singular equations, the solution of a nonlinear algebraic system is required at step (b) and a single nonlinear equation at step (d).

9. Example

The following example generates the first 16 fractional convolution and 23 fractionl starting weights generated by the fourth order BDF method.

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.

```
*     D05BYF Example Program Text
*     Mark 16 Release. NAG Copyright 1992.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          IORDER, IQ, ITPMT, ITIQ, LENFW, LDSW, LWK
      PARAMETER        (IORDER=4,IQ=3,ITPMT=2*IORDER-1,ITIQ=2**(IQ+1),
     +                 LENFW=2*ITIQ,LDSW=ITIQ+ITPMT,LWK=4*ITIQ)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             SW(LDSW,ITPMT), WORK(LWK), WT(LENFW)
*     .. External Subroutines ..
      EXTERNAL         D05BYF
*     .. Executable Statements ..
*
      WRITE (NOUT,*) 'D05BYF Example Program Results'
      WRITE (NOUT,*)
      IFAIL = 0
*
      CALL D05BYF(IORDER,IQ,LENFW,WT,SW,LDSW,WORK,LWK,IFAIL)
*
      WRITE (NOUT,*) 'Fractional convolution weights'
      WRITE (NOUT,*)
      DO 20 I = 1, ITIQ
         WRITE (NOUT,99999) I - 1, WT(I)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Fractional starting weights'
```

```
      WRITE (NOUT,*)
      DO 40 I = 1, LDSW
         WRITE (NOUT,99999) I - 1, (SW(I,J),J=1,ITPMT)
   40 CONTINUE
*
      STOP
*
99999 FORMAT (1X,I5,7F9.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D05BYF Example Program Results

Fractional convolution weights

    0   0.6928
    1   0.6651
    2   0.4589
    3   0.3175
    4   0.2622
    5   0.2451
    6   0.2323
    7   0.2164
    8   0.2006
    9   0.1878
   10   0.1780
   11   0.1700
   12   0.1629
   13   0.1566
   14   0.1508
   15   0.1457

Fractional starting weights

    0   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
    1   0.0565   2.8928  -6.7497  11.6491 -11.1355   5.5374  -1.1223
    2   0.0371   1.7401  -2.8628   6.5207  -6.4058   3.2249  -0.6583
    3   0.0300   1.3207  -2.4642   6.3612  -5.4478   2.7025  -0.5481
    4   0.0258   1.1217  -2.2620   5.3683  -3.7553   2.2132  -0.4549
    5   0.0230   0.9862  -2.0034   4.5005  -3.2772   2.7262  -0.4320
    6   0.0208   0.9001  -1.8989   4.2847  -3.5881   2.8201   0.2253
    7   0.0190   0.8506  -1.9250   4.4164  -4.0181   2.7932   0.1564
    8   0.0173   0.8177  -1.9697   4.5348  -4.2425   2.7458  -0.0697
    9   0.0160   0.7886  -1.9781   4.5318  -4.2769   2.6997  -0.2127
   10   0.0149   0.7603  -1.9548   4.4545  -4.2332   2.6541  -0.2620
   11   0.0140   0.7338  -1.9198   4.3619  -4.1782   2.6059  -0.2716
   12   0.0132   0.7097  -1.8842   4.2754  -4.1246   2.5544  -0.2767
   13   0.0125   0.6880  -1.8497   4.1933  -4.0662   2.5011  -0.2845
   14   0.0119   0.6681  -1.8153   4.1109  -4.0004   2.4479  -0.2915
   15   0.0114   0.6497  -1.7805   4.0279  -3.9304   2.3962  -0.2951
   16   0.0110   0.6327  -1.7461   3.9463  -3.8598   2.3466  -0.2958
   17   0.0105   0.6168  -1.7126   3.8677  -3.7907   2.2990  -0.2950
   18   0.0102   0.6020  -1.6804   3.7926  -3.7238   2.2536  -0.2935
   19   0.0098   0.5882  -1.6495   3.7209  -3.6589   2.2101  -0.2917
   20   0.0095   0.5752  -1.6199   3.6523  -3.5961   2.1686  -0.2895
   21   0.0093   0.5631  -1.5916   3.5867  -3.5356   2.1291  -0.2871
   22   0.0090   0.5517  -1.5644   3.5240  -3.4774   2.0914  -0.2844
```

Chapter E01 – Interpolation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E01AAF	1	Interpolated values, Aitken's technique, unequally spaced data, one variable
E01ABF	1	Interpolated values, Everett's formula, equally spaced data, one variable
E01AEF	8	Interpolating functions, polynomial interpolant, data may include derivative values, one variable
E01BAF	8	Interpolating functions, cubic spline interpolant, one variable
E01BEF	13	Interpolating functions, monotonicity-preserving, piecewise cubic Hermite, one variable
E01BFF	13	Interpolated values, interpolant computed by E01BEF, function only, one variable
E01BGF	13	Interpolated values, interpolant computed by E01BEF, function and first derivative, one variable
E01BHF	13	Interpolated values, interpolant computed by E01BEF, definite integral, one variable
E01DAF	14	Interpolating functions, fitting bicubic spline, data on rectangular grid
E01RAF	9	Interpolating functions, rational interpolant, one variable
E01RBF	9	Interpolated values, evaluate rational interpolant computed by E01RAF, one variable
E01SAF	13	Interpolating functions, method of Renka and Cline, two variables
E01SBF	13	Interpolated values, evaluate interpolant computed by E01SAF, two variables
E01SEF*	13	Interpolating functions, modified Shepard's method, two variables
E01SFF*	13	Interpolated values, evaluate interpolant computed by E01SEF, two variables
E01SGF	18	Interpolating functions, modified Shepard's method, two variables
E01SHF	18	Interpolated values, evaluate interpolant computed by E01SGF, function and first derivatives, two variables
E01TGF	18	Interpolating functions, modified Shepard's method, three variables
E01THF	18	Interpolated values, evaluate interpolant computed by E01TGF, function and first derivatives, three variables

* This routine is scheduled for withdrawal at Mark 20. See the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines' for details of the recommended replacement routine.

Chapter E01

Interpolation

Contents

1 Scope of the Chapter

This chapter is concerned with the interpolation of a function of one, two or three variables. When provided with the value of the function (and possibly one or more of its lowest-order derivatives) at each of a number of values of the variable(s), the routines provide either an interpolating function or an interpolated value. For some of the interpolating functions, there are supporting routines to evaluate, differentiate or integrate them.

2 Background to the Problems

In motivation and in some of its numerical processes, this chapter has much in common with the E02 Chapter Introduction (Curve and Surface Fitting). For this reason, we shall adopt the same terminology and refer to dependent variable and independent variable(s) instead of function and variable(s). Where there is only one independent variable, we shall denote it by x and the dependent variable by y. Thus, in the basic problem considered in this chapter, we are given a set of distinct values $x_1, x_2, \ldots, x_m$ of x and a corresponding set of values $y_1, y_2, \ldots, y_m$ of y, and we shall describe the problem as being one of interpolating the data points (x_r, y_r), rather than interpolating a function. In modern usage, however, **interpolation** can have either of two rather different meanings, both relevant to routines in this chapter. They are

(a) the determination of a function of x which takes the value y_r at $x = x_r$, for $r = 1, 2, \ldots, m$ (an **interpolating function** or **interpolant**),

(b) the determination of the value (**interpolated value** or **interpolate**) of an interpolating function at any given value, say $\hat{x}$, of x within the range of the x_r (so as to estimate the value at $\hat{x}$ of the function underlying the data).

The latter is the older meaning, associated particularly with the use of mathematical tables. The term 'function underlying the data', like the other terminology described above, is used so as to cover situations additional to those in which the data points have been computed from a known function, as with a mathematical table. In some contexts, the function may be unknown, perhaps representing the dependency of one physical variable on another, say temperature upon time.

Whether the underlying function is known or unknown, the object of interpolation will usually be to approximate it to acceptable accuracy by a function which is easy to evaluate anywhere in some range of interest. Polynomials, rational functions (ratios of two polynomials) and piecewise polynomials, such as cubic splines (see Section 2.2 of the E02 Chapter Introduction for definitions of terms in the latter case), being easy to evaluate and also capable of approximating a wide variety of functions, are the types of function mostly used in this chapter as interpolating functions. An interpolating polynomial is taken to have degree $m-1$ when there are m data points, and so it is unique. It is called the **Lagrange interpolating polynomial**. The rational function, in the special form used, is also unique. An interpolating spline, on the other hand, depends on the choice made for the knots.

One way of achieving the objective in (b) above is, of course, through (a), but there are also methods which do not involve the explicit computation of the interpolating function. Everett's formula and Aitken's successive linear interpolation (see Froberg [2]) provide two such methods. Both are used in this chapter and determine a value of the Lagrange interpolating polynomial.

It is important to appreciate, however, that the Lagrange interpolating polynomial often exhibits unwanted fluctuations between the data points. These tend to occur particularly towards the ends of the data range, and to get larger with increasing number of data points. In severe cases, such as with 30 or 40 equally spaced values of x, the polynomial can take on values several orders of magnitude larger than the data values. (Closer spacing near the ends of the range tends to improve the situation, and wider spacing tends to make it worse.) Clearly, therefore, the Lagrange polynomial often gives a very poor approximation to the function underlying the data. On the other hand, it can be perfectly satisfactory when its use is restricted to providing interpolated values away from the ends of the data range from a reasonably small number of data values.

In contrast, a cubic spline which interpolates a large number of data points can often be used satisfactorily over the whole of the data range. Unwanted fluctuations can still arise but much less frequently and much less severely than with polynomials. Rational functions, when appropriate, would also be used over the whole data range. The main danger with these functions is that their polynomial denominators may take

zero values within that range. Unwanted fluctuations are avoided altogether by a routine using piecewise cubic polynomials having only first derivative continuity. It is designed especially for monotonic data, but for other data still provides an interpolant which increases, or decreases, over the same intervals as the data.

The concept of interpolation can be generalised in a number of ways. Firstly, at each x, the interpolating function may be required to take on not only a given value but also given values for all its derivatives up to some specified order (which can vary with r). This is the Hermite–Birkoff interpolation problem. Secondly, we may be required to estimate the value of the underlying function at a value $\hat{x}$ outside the range of the data. This is the process of **extrapolation**. In general, it is a good deal less accurate than interpolation and is to be avoided whenever possible.

Interpolation can also be extended to the case of two or more independent variables. If the data values are given at the intersections of a regular two-dimensional mesh bicubic splines (see Section 2.3.2 of the E02 Chapter Introduction) are very suitable and usually very effective for the problem. For other cases, perhaps where the data values are quite arbitrarily scattered, polynomials and splines are not at all appropriate and special forms of interpolating function have to be employed. Many such forms have been devised and two of the most successful are in routines in this chapter. They both have continuity in first, but not higher, derivatives.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 General

Before undertaking interpolation, in other than the simplest cases, the user should seriously consider the alternative of using a routine from the E02 Chapter Introduction to approximate the data by a polynomial or spline containing significantly fewer coefficients than the corresponding interpolating function. This approach is much less liable to produce unwanted fluctuations and so can often provide a better approximation to the function underlying the data.

When interpolation is employed to approximate either an underlying function or its values, the user will need to be satisfied that the accuracy of approximation achieved is adequate. There may be a means for doing this which is particular to the application, or the routine used may itself provide a means. In other cases, one possibility is to repeat the interpolation using one or more extra data points, if they are available, or otherwise one or more fewer, and to compare the results. Other possibilities, if it is an interpolating function which is determined, are to examine the function graphically, if that gives sufficient accuracy, or to observe the behaviour of the differences in a finite-difference table, formed from evaluations of the interpolating function at equally-spaced values of x over the range of interest. The spacing should be small enough to cause the typical size of the differences to decrease as the order of difference increases.

3.2 One Independent Variable

3.2.1 Interpolated values: data without derivatives

When the underlying function is well represented by data points on both sides of the value, $\hat{x}$, at which an interpolated value is required, E01ABF should be tried first if the data points are equally spaced, E01AAF if they are not. Both compute a value of the Lagrange interpolating polynomial, the first using Everett's formula, the second Aitken's successive linear interpolation. The first routine requires an equal (or nearly equal) number of data points on each side of $\hat{x}$; such a distribution of points is preferable also for the second routine. If there are many data points, this will be achieved simply by using only an appropriate subset for each value of $\hat{x}$. Ten to twelve data points are the most that would be required for many problems. Both routines provide a means of assessing the accuracy of an interpolated value, with E01ABF by examination of the size of the finite differences supplied, with E01AAF by intercomparison of the set of interpolated values obtained from polynomials of increasing degree.

In other cases, or when the above routines fail to produce a satisfactory result, one of the routines discussed in the next section should be used. The spline and other piecewise polynomial routines are the most generally applicable. They are particularly appropriate when interpolated values towards the

ends of the range are required. They are also likely to be preferable, for reasons of economy, when many interpolated values are required.

E01AAF above, and three of the routines discussed in the next section, can be used to compute extrapolated values. These three are E01AEF, E01BEF and E01RAF based on polynomials, piecewise polynomials and rational functions respectively. Extrapolation is not recommended in general, but can sometimes give acceptable results if it is to a point not far outside the data range, and only the few nearest data points are used in the process. E01RAF is most likely to be successful.

3.2.2 Interpolating function: data without derivatives

E01AEF computes the Lagrange interpolating polynomial by a method (based on **Newton's formula with divided differences** [1]) which has proved numerically very stable. Thus, it can sometimes be used to provide interpolated values in more difficult cases than can E01AAF (see previous section). However, the likelihood of the polynomial having unwanted fluctuations, particularly near the ends of the data range when a moderate or large number of data points are used, should be remembered.

Such fluctuations of the polynomial can be avoided if the user is at liberty to choose the x-values at which to provide data points. In this case, a routine from Chapter E02, namely E02AFF, should be used in the manner and with the x-values discussed in Section 3.2.2 of the E02 Chapter Introduction.

Usually however, when the whole of the data range is of interest, it is preferable to use a cubic spline as the interpolating function. E01BAF computes an interpolating cubic spline, using a particular choice for the set of knots which has proved generally satisfactory in practice. If the user wishes to choose a different set, a cubic spline routine from Chapter E02, namely E02BAF, may be used in its interpolating mode, setting NCAP7 = M + 4 and all elements of the parameter W to unity.

The cubic spline does not always avoid unwanted fluctuations, especially when the data show a steep slope close to a region of small slope, or when the data inadequately represent the underlying curve. In such cases, E01BEF can be very useful. It derives a piecewise cubic polynomial (with first derivative continuity) which, between any adjacent pair of data points, either increases all the way, or decreases all the way (or stays constant). It is especially suited to data which are monotonic over their whole range.

In this routine, the interpolating function is represented simply by its value and first derivative at the data points. Supporting routines compute its value and first derivative elsewhere, as well as its definite integral over an arbitary interval. The other routines above provide the interpolating function either in Chebyshev-series form or in B-spline form (see Section 2.2.1 of the E02 Chapter Introduction and Section 2.2.2 of the E02 Chapter Introduction). Routines for evaluating, differentiating and integrating these forms are discussed in Section 3.7 of the E02 Chapter Introduction. The splines and other piecewise cubics will normally provide better estimates of the derivatives of the underlying function than will interpolating polynomials, at any rate away from the central part of the data range.

E01RAF computes an interpolating rational function. It is intended mainly for those cases where the user knows that this form of function is appropriate. However, it is also worth trying in cases where the other routines have proved unsatisfactory. E01RBF is available to compute values of the function provided by E01RAF.

3.2.3 Data containing derivatives

E01AEF (see previous section) can also compute the polynomial which, at each x_r, has not only a specified value y_r but also a specified value of each derivative up to order p_r.

3.3 Two Independent Variables

3.3.1 Data on a rectangular mesh

Given the value f_{qr} of the dependent variable f at the point (x_q, y_r) in the plane of the independent variables x and y, for each $q = 1, 2, \ldots, m$ and $r = 1, 2, \ldots, n$ (so that the points (x_q, y_r) lie at the $m \times n$ intersections of a rectangular mesh), E01DAF computes an interpolating bicubic spline, using a particular choice for each of the spline's knot-set. This choice, the same as in E01BAF, has proved generally satisfactory in practice. If, instead, the user wishes to specify his own knots, a routine from Chapter E02, namely E02DAF, may be adapted (it is more cumbersome for the purpose, however, and much slower for larger problems). Using m and n in the above sense, the parameter M must be set to

$m \times n$, PX and PY must be set to $m + 4$ and $n + 4$ respectively and all elements of W should be set to unity. The recommended value for EPS is zero.

3.3.2 Arbitrary data

As remarked at the end of Section 2, special types of interpolating are required for this problem, which can often be difficult to solve satisfactorily. Two of the most successful are employed in E01SAF and E01SGF, the two routines which (with their respective evaluation routines E01SBF and E01SHF) are provided for the problem. Definitions can be found in the routine documents. Both interpolants have first derivative continuity and are 'local', in that their value at any point depends only on data in the immediate neighbourhood of the point. This latter feature is necessary for large sets of data to avoid prohibitive computing time. E01SHF allows evaluation of the interpolant and its first partial derivatives.

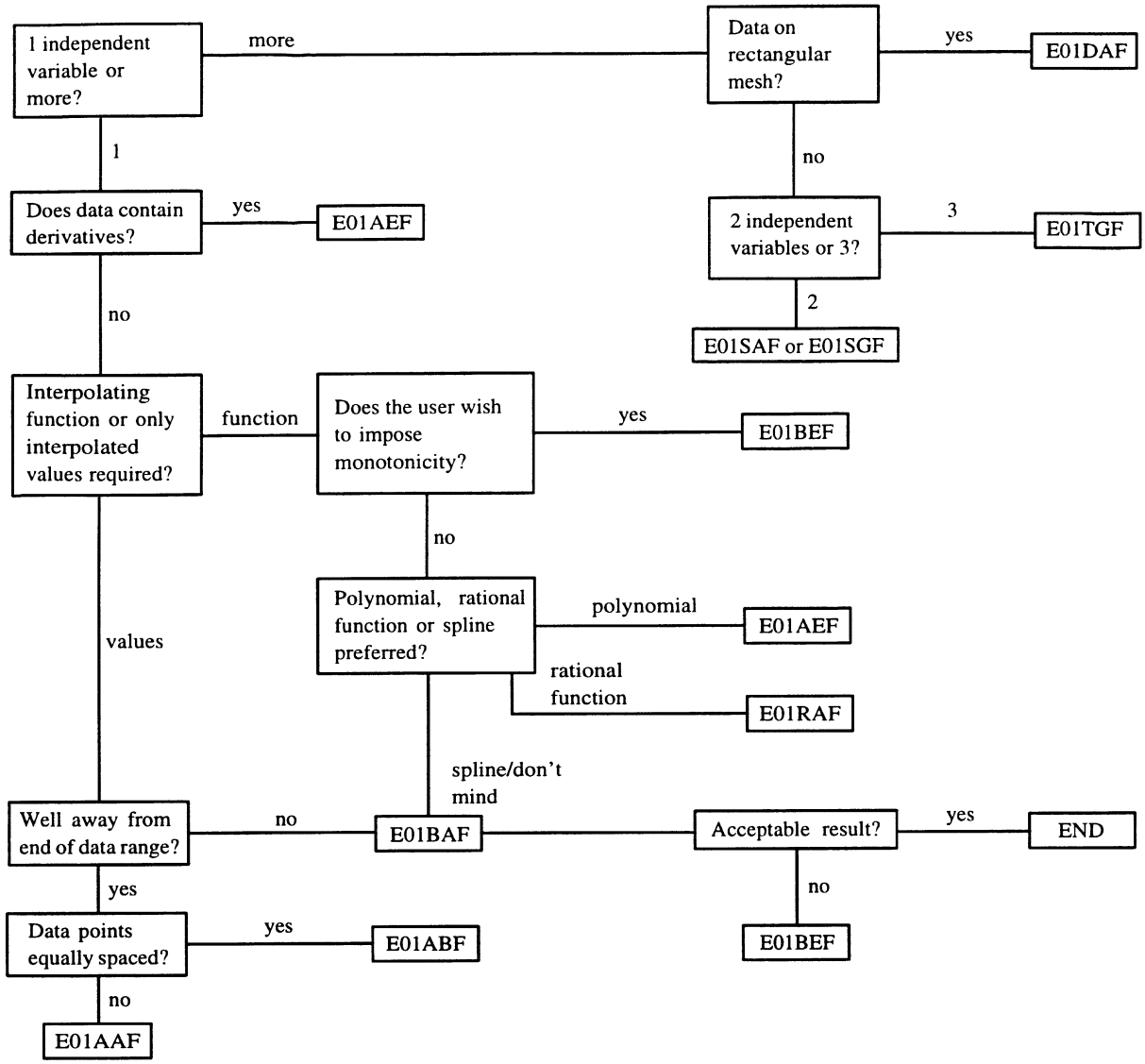
The relative merits of the two methods vary with the data and it is not possible to predict which will be the better in any particular case.

3.4 Three Independent Variables

3.4.1 Arbitrary data

The routine E01TGF and its evaluation routine E01THF are provided for interpolation of three dimensional scattered data. As in the case of two independent variables, the method is local, and produces an interpolant with first derivative continuity. E01THF allows evaluation of the interpolant and its first partial derivatives.

4 Decision Trees



5 Index

from polynomial,	
equally spaced data	**E01ABF**
general data	**E01AAF**
from rational function	**E01RBF**
Interpolated values, two variables,	
from interpolant from E01SAF	**E01SBF**
from interpolant from E01SGF	**E01SHF**
Interpolating function, one variable,	
cubic spline	**E01BAF**
other piecewise polynomial	**E01BEF**
polynomial, data with or without derivatives	**E01AEF**
rational function	**E01RAF**
Interpolating function, two variables	
bicubic spline	**E01DAF**
other piecewise polynomial	**E01SAF**
modified Shepard method	**E01SGF**
Interpolating function, three variables	
modified Shepard method	**E01TGF**

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 19, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E01ACF E01SEF† E01SFF†

7 References

[1] Froberg C E (1970) *Introduction to Numerical Analysis* Addison–Wesley

[2] Dahlquist G and Björck Å(1974) *Numerical Methods* Prentice–Hall

E01AAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01AAF interpolates at a given point x from a table of function values y_i evaluated at equidistant or non-equidistant points x_i, for $i = 1,2,...,n+1$, using Aitken's technique of successive linear interpolations.

2. Specification

```
SUBROUTINE E01AAF (A, B, C, N1, N2, N, X)
INTEGER          N1, N2, N
real             A(N1), B(N1), C(N2), X
```

3. Description

This routine interpolates at a given point x from a table of values x_i and y_i, for $i = 1,2,...,n+1$ using Aitken's method. The intermediate values of linear interpolations are stored to enable an estimate of the accuracy of the results to be made.

4. References

[1] FROBERG, C.E.
Introduction to Numerical Analysis.
Addison-Wesley Publishing Company Inc., pp. 148-151, (2nd Edition) 1970.

5. Parameters

1: A(N1) – ***real*** array. *Input/Output*

On entry: A(i) must contain the x–component of the ith data point, x_i, for $i = 1,2,...,n+1$.

On exit: A(i) contains the value $x_i - x$, for $i = 1,2,...,n+1$.

2: B(N1) – ***real*** array. *Input/Output*

On entry: B(i) must contain the y-component (function value) of the ith data point, y_i, for $i = 1,2,...,n+1$.

On exit: the contents of B are unspecified.

3: C(N2) – ***real*** array. *Output*

On exit:

C(1),...,C(n) contain the first set of linear interpolations,
C(n+1),...,C(2×n−1) contain the second set of linear interpolations
...
C(n×(n+1)/2) contains the interpolated function value at the point x.

4: N1 – INTEGER. *Input*

On entry: the value $n + 1$ where n is the number of intervals; that is, N1 is the number of data points.

5: N2 – INTEGER. *Input*

On entry: the value n×(n+1)/2 where n is the number of intervals.

6: N – INTEGER. *Input*

On entry: the number of intervals which are to be used in interpolating the value at x; that is, there are $n + 1$ data points (x_i,y_i).

7: X – ***real.*** *Input*

On entry: the point x at which the interpolation is required.

6. Error Indicators and Warnings

None.

7. Accuracy

An estimate of the accuracy of the result can be made from a comparison of the final result and the previous interpolates, given in the array C. In particular, the first interpolate in the ith set, for $i = 1,2,...,n$, is the value at x of the polynomial interpolating the first $i + 1$ data points. It is given in position $1 + \frac{1}{2}(i-1)(2n-i+2)$ of the array C. Ideally, providing n is large enough, this set of n interpolates should exhibit convergence to the final value, the difference between one interpolate and the next settling down to a roughly constant magnitude (but with varying sign). This magnitude indicates the size of the error (any subsequent increase meaning that the value of n is too high). Better convergence will be obtained if the data points are supplied, not in their natural order, but ordered so that the first i data points give good coverage of the neighbourhood of x, for all i. To this end, the following ordering is recommended as widely suitable: first the point nearest to x, then the nearest point on the opposite side of x, followed by the remaining points in increasing order of their distance from x, that is of $|x_r - x|$. With this modification the Aitken method will generally perform better than the related method of Neville, which is often given in the literature as superior to that of Aitken.

8. Further Comments

The computation time for interpolation at any point x is proportional to $n \times (n+1)/2$.

9. Example

To interpolate at $x = 0.28$ the function value of a curve defined by the points

$$\begin{pmatrix} x_i & -1.00 & -0.50 & 0.00 & 0.50 & 1.00 & 1.50 \\ y_i & 0.00 & -0.53 & -1.00 & -0.46 & 2.00 & 11.09 \end{pmatrix}.$$

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.

```
*     E01AAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, N1MAX, N2MAX
      PARAMETER        (NMAX=9,N1MAX=NMAX+1,N2MAX=NMAX*N1MAX/2)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             X
      INTEGER          I, J, K, N
*     .. Local Arrays ..
      real             A(N1MAX), B(N1MAX), C(N2MAX)
*     .. External Subroutines ..
      EXTERNAL         E01AAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01AAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N, X
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         READ (NIN,*) (A(I),I=1,N+1)
         READ (NIN,*) (B(I),I=1,N+1)
*
```

```
         CALL E01AAF(A,B,C,N+1,N*(N+1)/2,N,X)
*
         K = 1
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Interpolated values'
         DO 20 I = 1, N - 1
            WRITE (NOUT,99999) (C(J),J=K,K+N-I)
            K = K + N - I + 1
   20    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Interpolation point = ', X
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Function value at interpolation point = ',
     +     C(N*(N+1)/2)
       END IF
       STOP
*
99999 FORMAT (1X,6F12.5)
99998 FORMAT (1X,A,F12.5)
      END
```

9.2. Program Data

```
E01AAF Example Program Data
  5     0.28
    -1.00   -0.50    0.00    0.50    1.00    1.50
     0.00   -0.53   -1.00   -0.46    2.00   11.09
```

9.3. Program Results

```
 E01AAF Example Program Results

 Interpolated values
     -1.35680    -1.28000    -0.39253     1.28000     5.67808
     -1.23699    -0.60467     0.01434     1.38680
     -0.88289    -0.88662    -0.74722
     -0.88125    -0.91274

 Interpolation point =      0.28000

 Function value at interpolation point =     -0.83591
```

E01ABF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01ABF interpolates at a given point x from a table of function values evaluated at equidistant points, by Everett's formula.

2. Specification

```
SUBROUTINE E01ABF (N, P, A, G, N1, N2, IFAIL)
INTEGER          N, N1, N2, IFAIL
real             P, A(N1), G(N2)
```

3. Description

This routine interpolates at a given point

$$x = x_0 + ph, \quad \text{where } -1 < p < 1$$

from a table of values (x_0+mh) and y_m where $m = -(n-1),-(n-2),...,-1,0,1,...,n$. The formula used is that of Everett [1], neglecting the remainder term:

$$y_p = \sum_{r=0}^{n-1} \left(\frac{1-p+r}{2r+1}\right)\delta^{2r}y_0 + \sum_{r=0}^{n-1} \left(\frac{p+r}{2r+1}\right)\delta^{2r}y_1$$

The values of $\delta^{2r}y_0$ and $\delta^{2r}y_1$ are stored on exit from the routine in addition to the interpolated function value y_p.

4. References

[1] FROBERG, C.E.
Introduction to Numerical Analysis.
Addison-Wesley Publishing Company Inc., pp. 174-182, 1969.

5. Parameters

1: N – INTEGER. *Input*

On entry: n, half the number of points to be used in the interpolation.

2: P – ***real***. *Input*

On entry: the point p at which the interpolated function value is required i.e. $p = (x-x_0)/h$ with $-1.0 < p < 1.0$.

Constraint: $-1.0 < \text{P} < 1.0$.

3: A(N1) – ***real*** array. *Input/Output*

On entry: A(i) must be set to the function value y_{i-n} for $i = 1,2,...,2n$.

On exit: the contents of A are unspecified.

4: G(N2) – ***real*** array. *Output*

On exit: the array contains

y_0 in G(1)
y_1 in G(2)
$\delta^{2r}y_0$ in G($2r$+1)
$\delta^{2r}y_1$ in G($2r$+2) for $r = 1,2,...,n-1$.

The interpolated function value y_p is stored in G($2n$+1).

5: N1 – INTEGER. *Input*

On entry: the value $2n$, that is, N1 is equal to the number of data points.

6: N2 – INTEGER. *Input*

On entry: the value $2n + 1$, that is, N2 is one more than the number of data points.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, P ≤ –1.0,
or P ≥ 1.0.

7. Accuracy

In general, increasing n improves the accuracy of the result until full attainable accuracy is reached, after which it might deteriorate. If x lies in the central interval of the data (i.e. $0.0 \leq p \leq 1.0$), as is desirable, an upper bound on the contribution of the highest order differences (which is usually an upper bound on the error of the result) is given approximately in terms of the elements of the array G by $a \times (|G(2n-1)| + |G(2n)|)$, where a = 0.1, 0.02, 0.005, 0.001, 0.0002 for n = 1,2,3,4,5 respectively, thereafter decreasing roughly by a factor of 4 each time.

8. Further Comments

The computation time increases as the order of n increases.

9. Example

To interpolate at the point $x = 0.28$ from the function values

$$\begin{pmatrix} x_i & -1.00 & -0.50 & 0.00 & 0.50 & 1.00 & 1.50 \\ y_i & 0.00 & -0.53 & -1.00 & -0.46 & 2.00 & 11.09 \end{pmatrix}.$$

We take $n = 3$ and $p = 0.56$.

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.

```
*     E01ABF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, N1MAX, N2MAX
      PARAMETER        (NMAX=10,N1MAX=2*NMAX,N2MAX=2*NMAX+1)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             P
      INTEGER          I, IFAIL, N, R
*     .. Local Arrays ..
      real             A(N1MAX), G(N2MAX)
*     .. External Subroutines ..
      EXTERNAL         E01ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01ABF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N, P
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         READ (NIN,*) (A(I),I=1,2*N)
         IFAIL = 0
*
         CALL E01ABF(N,P,A,G,2*N,2*N+1,IFAIL)
*
         WRITE (NOUT,*)
         DO 20 R = 0, N - 1
            WRITE (NOUT,99999) 'Central differences order ', R,
     +        ' of Y0 =', G(2*R+1)
            WRITE (NOUT,99998) '                                  Y1 =',
     +        G(2*R+2)
   20    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Function value at interpolation point =',
     +     G(2*N+1)
       END IF
       STOP
*
99999 FORMAT (1X,A,I1,A,F12.5)
99998 FORMAT (1X,A,F12.5)
      END
```

9.2. Program Data

```
E01ABF Example Program Data
  3     0.56
    0.00   -0.53   -1.00   -0.46    2.00   11.09
```

9.3. Program Results

```
 E01ABF Example Program Results

 Central differences order 0 of Y0 =    -1.00000
                                 Y1 =    -0.46000
 Central differences order 1 of Y0 =     1.01000
                                 Y1 =     1.92000
 Central differences order 2 of Y0 =    -0.04000
                                 Y1 =     3.80000

 Function value at interpolation point =    -0.83591
```

E01AEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01AEF constructs the Chebyshev-series representation of a polynomial interpolant to a set of data which may contain derivative values.

2. Specification

```
      SUBROUTINE E01AEF (M, XMIN, XMAX, X, Y, IP, N, ITMIN, ITMAX, A, WRK,
     1                   LWRK, IWRK, LIWRK, IFAIL)
      INTEGER          M, IP(M), N, ITMIN, ITMAX, LWRK, IWRK(LIWRK),
     1                 LIWRK, IFAIL
      real             XMIN, XMAX, X(M), Y(N), A(N), WRK(LWRK)
```

3. Description

Let m distinct values x_i of an independent variable x be given, with $x_{\min} \leq x_i \leq x_{\max}$, for $i = 1,2,...,m$. For each value x_i, suppose that the value y_i of the dependent variable y together with the first p_i derivatives of y with respect to x are given. Each p_i must therefore be a non-negative integer, with the total number of interpolating conditions, n, equal to $m + \sum_{i=1}^{m} p_i$.

E01AEF calculates the unique polynomial $q(x)$ of degree $n - 1$ (or less) which is such that $q^{(k)}(x_i) = y_i^{(k)}$ for $i = 1,2,...,m$; $k = 0,1,...,p_i$. Here $q^{(0)}(x_i)$ means $q(x_i)$. This polynomial is represented in Chebyshev-series form in the normalised variable $\bar{x}$, as follows:

$$q(x) = \tfrac{1}{2}a_0T_0(\bar{x}) + a_1T_1(\bar{x}) + \ldots + a_{n-1}T_{n-1}(\bar{x}),$$

where

$$\bar{x} = \frac{2x - x_{\min} - x_{\max}}{x_{\max} - x_{\min}}$$

so that $-1 \leq \bar{x} \leq 1$ for x in the interval $x_{\min}$ to $x_{\max}$, and where $T_i(\bar{x})$ is the Chebyshev polynomial of the first kind of degree i with argument $\bar{x}$.

(The polynomial interpolant can subsequently be evaluated for any value of x in the given range by using E02AKF. Chebyshev-series representations of the derivative(s) and integral(s) of $q(x)$ may be obtained by (repeated) use of E02AHF and E02AJF.)

The method used consists first of constructing a divided-difference table from the normalised $\bar{x}$ values and the given values of y and its derivatives with respect to $\bar{x}$. The Newton form of $q(x)$ is then obtained from this table, as described in Huddleston [1] and Krogh [2], with the modification described in Section 8.2. The Newton form of the polynomial is then converted to Chebyshev-series form as described in Section 8.3.

Since the errors incurred by these stages can be considerable, a form of iterative refinement is used to improve the solution. This refinement is particularly useful when derivatives of rather high order are given in the data. In reasonable examples, the refinement will usually terminate with a certain accuracy criterion satisfied by the polynomial (see Section 7). In more difficult examples, the criterion may not be satisfied and refinement will continue until the maximum number of iterations (as specified by the input parameter ITMAX) is reached.

In extreme examples, the iterative process may diverge (even though the accuracy criterion is satisfied): if a certain divergence criterion is satisfied, the process terminates at once. In all cases the routine returns the 'best' polynomial achieved before termination. For the definition of 'best' and details of iterative refinement and termination criteria, see Section 8.4.

4. References

[1] HUDDLESTON, R.E.
CDC 6600 routines for the interpolation of data and of data with derivatives.
Sandia Laboratories. Reprint SLL-74-0214, 1974.

[2] KROGH, F.T.
Efficient algorithms for polynomial interpolation and numerical differentiation.
Math. Comp., 24, pp. 185-190, 1970.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of given values of the independent variable x.

Constraint: M $\geq$ 1.

2: XMIN – ***real***. *Input*

3: XMAX – ***real***. *Input*

On entry: the lower and upper endpoints, respectively, of the interval $[x_{\min},x_{\max}]$. If they are not determined by the user's problem, it is recommended that they be set respectively to the smallest and largest values among the x_i.

Constraint: XMIN < XMAX.

4: X(M) – ***real*** array. *Input*

On entry: the value of x_i, for $i = 1,2,...,m$. The X(i) need not be ordered.

Constraint: XMIN $\leq$ X(i) $\leq$ XMAX, and the X(i) must be distinct.

5: Y(N) – ***real*** array. *Input*

On entry: the given values of the dependent variable, and derivatives, as follows:

The first $p_1 + 1$ elements contain $y_1,y_1^{(1)},...,y_1^{(p_1)}$ in that order.

The next $p_2 + 1$ elements contain $y_2,y_2^{(1)},...,y_2^{(p_2)}$ in that order.

...

The last $p_m + 1$ elements contain $y_m,y_m^{(1)},...,y_m^{(p_m)}$ in that order.

6: IP(M) – INTEGER array. *Input*

On entry: p_i, the order of the highest-order derivative whose value is given at x_i, for $i = 1,2,...,m$. If the value of y only is given for some x_i then the corresponding value of IP(i) must be zero.

Constraint: IP(i) $\geq$ 0, for i = 1,2,...,M.

7: N – INTEGER. *Input*

On entry: the total number of interpolating conditions, n.

Constraint: N = M + IP(1) + IP(2) + ... + IP(M).

8: ITMIN – INTEGER. *Input*

9: ITMAX – INTEGER. *Input*

On entry: respectively the minimum and maximum number of iterations to be performed by the routine (for full details see Section 8.4, second paragraph). Setting ITMIN and/or ITMAX negative or zero invokes default value(s) of 2 and/or 10, respectively.

The default values will be satisfactory for most problems, but occasionally significant improvement will result from using higher values.

Suggested value: ITMIN = 0 and ITMAX = 0.

10: A(N) – ***real*** array. *Output*

On exit: A(i) contains the coefficient a_{i-1} in the Chebyshev-series representation of $q(x)$, for i = 1,2,...,n.

11: WRK(LWRK) – ***real*** array. *Workspace*

Used as workspace, but see also Section 8.5.

12: LWRK – INTEGER. *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E01AEF is called.

Constraint: LWRK ≥ 7×N + 5×IPMAX + M + 7, where IPMAX is the largest value of IP(i), for i = 1,2,...,M.

13: IWRK(LIWRK) – INTEGER array. *Workspace*

Used as workspace, but see also Section 8.5.

14: LIWRK – INTEGER. *Input*

On entry: the dimension of the array IWRK as declared in the (sub)program from which E01AEF is called.

Constraint: LIWRK ≥ 2×M + 2.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, M < 1,
or N ≠ M + IP(1) + IP(2) + ... + IP(M),
or LWRK < 7×N + 5×IPMAX + M + 7,
or LIWRK < 2×M + 2
(IPMAX is defined under LWRK).

IFAIL = 2

On entry, IP(i) < 0 for some i.

IFAIL = 3

On entry, XMIN ≥ XMAX,
or X(i) < XMIN for some i,
or X(i) > XMAX,
or X(i) = X(j) for some $i \neq j$.

IFAIL = 4

Not all the performance indices are less than eight times the ***machine precision***, although ITMAX iterations have been performed. Parameters A, WRK and IWRK relate to the best polynomial determined. A more accurate solution may possibly be obtained by increasing ITMAX and re-calling the routine. See also Sections 7 and 8.4-8.5.

IFAIL = 5

The computation has been terminated because the iterative process appears to be diverging. (Parameters A, WRK and IWRK relate to the best polynomial determined.) Thus the problem specified by the user's data is probably too ill-conditioned for the solution to be satisfactory. This may result from some of the X(i) being very close together, or from the number of interpolating conditions, N, being large. If in such cases the conditions do not involve derivatives, the user is likely to obtain a much more satisfactory solution to his problem either by cubic spline interpolation (see E01BAF) or by curve fitting with a polynomial or spline in which the number of coefficients is less than N, preferably much less if N is large (see Chapter E02). But see Sections 7 and 8.4-8.5.

7. Accuracy

A complete error analysis is not currently available, but the method gives good results for reasonable problems.

It is important to realise that for some sets of data, the polynomial interpolation problem is **ill-conditioned**. That is, a **small** perturbation in the data may induce **large** changes in the polynomial, **even in exact arithmetic**. Though by no means the worst example, interpolation by a single polynomial to a large number of function values given at points equally spaced across the range is notoriously ill-conditioned and the polynomial interpolating such a data set is prone to exhibit enormous oscillations between the data points, especially near the ends of the range. These will be reflected in the Chebyshev coefficients being large compared with the given function values. A more familiar example of ill-conditioning occurs in the solution of certain systems of linear algebraic equations, in which a small change in the elements of the matrix and/or in the components of the right-hand side vector induces a relatively large change in the solution vector. The best that can be achieved in these cases is to make the residual vector small in some sense. If this is possible, the computed solution is exact for a slightly perturbed set of data. Similar considerations apply to the interpolation problem.

The residuals $y_i^{(k)} - q^{(k)}(x_i)$ are available for inspection (see Section 8.5). To assess whether these are reasonable, however, it is necessary to relate them to the largest function and derivative values taken by $q(x)$ over the interval $[x_{\min}, x_{\max}]$. The following performance indices aim to do this. Let the kth derivative of q with respect to the normalised variable $\bar{x}$ be given by the Chebyshev-series

$$\tfrac{1}{2}a_0^k T_0(\bar{x}) + a_1^k T_1(\bar{x}) + \ldots + a_{n-1-k}^k T_{n-1-k}(\bar{x}).$$

Let A_k denote the sum of the moduli of these coefficients (this is an upper bound on the kth derivative in the interval and is taken as a measure of the maximum size of this derivative), and define

$$S_k = \max_{i \leq k} A_i.$$

Then if the root-mean-square value of the residuals of $q^{(k)}$, scaled so as to relate to the normalised variable $\bar{x}$, is denoted by r_k, the performance indices are defined by

$$P_k = r_k / S_k, \text{ for } k = 0,1,\ldots,\max_i (p_i).$$

It is expected that, in reasonable cases, they will all be less than (say) 8 times the ***machine precision*** (this is the accuracy criterion mentioned in Section 3), and in many cases will be of the order of ***machine precision*** or less.

8. Further Comments

8.1. Timing

Computation time is approximately proportional to IT$\times n^3$, where IT is the number of iterations actually used. (See Section 8.5).

8.2. Divided-difference Strategy

In constructing each new coefficient in the Newton form of the polynomial, a new x_i must be brought into the computation. The x_i chosen is that which yields the smallest new coefficient. This strategy increases the stability of the divided-difference technique, sometimes quite markedly, by reducing errors due to cancellation.

8.3. Conversion to Chebyshev Form

Conversion from the Newton form to Chebyshev-series form is effected by evaluating the former at the n values of $\bar{x}$ at which $T_{n-1}(\bar{x})$ takes the value ± 1, and then interpolating these n function values by a call of E02AFF, which provides the Chebyshev-series representation of the polynomial with very small additional relative error.

8.4. Iterative Refinement

The iterative refinement process is performed as follows. First, an initial approximation, $q_1(x)$ say, is found by the technique described above. The rth step of the refinement process then consists of evaluating the residuals of the rth approximation $q_r(x)$, and constructing an interpolant, $dq_r(x)$, to these residuals. The next approximation $q_{r+1}(x)$ to the interpolating polynomial is then obtained as

$$q_{r+1}(x) = q_r(x) + dq_r(x).$$

This completes the description of the rth step.

The iterative process is terminated according to the following criteria. When a polynomial is found whose performance indices (as defined in Section 7) are all less than 8 times the ***machine precision***, the process terminates after ITMIN further iterations (or after a total of ITMAX iterations if that occurs earlier). This will occur in most reasonable problems. The extra iterations are to allow for the possibility of further improvement. If no such polynomial is found, the process terminates after a total of ITMAX iterations. Both these criteria are over-ridden, however, in two special cases. Firstly, if for some value of r the sum of the moduli of the Chebyshev coefficients of $dq_r(x)$ is greater than that of $q_r(x)$, it is concluded that the process is diverging and the process is terminated at once ($q_{r+1}(x)$ is not computed). Secondly, if at any stage, the performance indices are all computed as zero, again the process is terminated at once.

As the iterations proceed, a record is kept of the best polynomial. Subsequently, at the end of each iteration, the new polynomial replaces the current best polynomial if it satisfies two conditions (otherwise the best polynomial remains unchanged). The first condition is that at least one of its root-mean-square residual values, r_k (see Section 7) is smaller than the corresponding value for the current best polynomial. The second condition takes two different forms according to whether or not the performance indices (see Section 7) of the current best polynomial are all less than 8 times the ***machine precision***. If they are, then the largest performance index of the new polynomial is required to be less than that of the current best polynomial. If they are not, the number of indices which are less than 8 times ***machine precision*** must not be smaller than for the current best polynomial. When the iterative process is terminated, it is the polynomial then recorded as best, which is returned to the user as $q(x)$.

8.5. Workspace Information

On successful exit, and also if IFAIL = 4 or 5 on exit, the following information is contained in the workspace arrays WRK and IWRK:

WRK(k+1), for k = 0,1,...,IPMAX where IPMAX = $\max_i p_i$, contains the ratio of p_k, the performance index relating to the kth derivative of the $q(x)$ finally provided, to 8 times the ***machine precision***.

WRK(IPMAX+1+j), for j = 1,2,...,n, contains the jth residual, i.e. the value of $y_i^{(k)} - q^{(k)}(x_i)$, where i and k are the appropriate values corresponding to the jth element in the array Y (see description of Y in Section 5).

IWRK(1) contains the number of iterations actually performed in deriving $q(x)$.

If, on exit, IFAIL = 4 or 5, the $q(x)$ finally provided may still be adequate for the user's requirements. To assess this the user should examine the residuals contained in WRK(IPMAX+1+j), for j = 1,2,...,n, to see whether they are acceptably small.

9. Example

To construct an interpolant $q(x)$ to the following data:

$m = 4,\ x_{\min} = 2,\ x_{\max} = 6,$
$x_1 = 2,\ p_1 = 0,\ y_1 = 1,$
$x_2 = 4,\ p_2 = 1,\ y_2 = 2,\ y_2^{(1)} = -1,$
$x_3 = 5,\ p_3 = 0,\ y_3 = 1,$
$x_4 = 6,\ p_4 = 2,\ y_4 = 2,\ y_4^{(1)} = 4,\ y_4^{(2)} = -2.$

The coefficients in the Chebyshev-series representation of $q(x)$ are printed, and also the residuals corresponding to each of the given function and derivative values.

This program is written in a generalised form which can read any number of data-sets.

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.

```
*     E01AEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX, IPMX, LWRK, LIWRK
      PARAMETER        (MMAX=4,NMAX=8,IPMX=2,LWRK=7*NMAX+5*IPMX+MMAX+7,
     +                 LIWRK=2*MMAX+2)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             XMAX, XMIN
      INTEGER          I, IFAIL, IP1, IPMAX, IRES, IY, J, M, N
*     .. Local Arrays ..
      real             A(NMAX), WRK(LWRK), X(MMAX), Y(NMAX)
      INTEGER          IP(MMAX), IWRK(LIWRK)
*     .. External Subroutines ..
      EXTERNAL         E01AEF
*     .. Intrinsic Functions ..
      INTRINSIC        MAX
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01AEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, XMIN, XMAX
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         N = 0
         IPMAX = 0
         DO 40 I = 1, M
            READ (NIN,*) IP(I), X(I), (Y(J),J=N+1,N+IP(I)+1)
            IPMAX = MAX(IPMAX,IP(I))
            N = N + IP(I) + 1
   40    CONTINUE
         IF (N.LE.NMAX .AND. IPMAX.LE.IPMX) THEN
            IFAIL = 1
*
            CALL E01AEF(M,XMIN,XMAX,X,Y,IP,N,-1,-1,A,WRK,LWRK,IWRK,
     +                  LIWRK,IFAIL)
*
            WRITE (NOUT,*)
            IF (IFAIL.EQ.0 .OR. IFAIL.GE.4) THEN
               WRITE (NOUT,99999)
     +           'Total number of interpolating conditions =', N
               WRITE (NOUT,*)
               WRITE (NOUT,*) 'Interpolating polynomial'
```

```
                  WRITE (NOUT,*)
                  WRITE (NOUT,*) '    I     Chebyshev Coefficient A(I+1)'
                  DO 60 I = 1, N
                     WRITE (NOUT,99998) I - 1, A(I)
   60             CONTINUE
                  WRITE (NOUT,*)
                  WRITE (NOUT,*) '  X     R    Rth derivative     Residual'
                  IY = 0
                  IRES = IPMAX + 1
                  DO 100 I = 1, M
                     IP1 = IP(I) + 1
                     DO 80 J = 1, IP1
                        IY = IY + 1
                        IRES = IRES + 1
                        IF (J-1.NE.0) THEN
                           WRITE (NOUT,99997) J - 1, Y(IY), WRK(IRES)
                        ELSE
                           WRITE (NOUT,99996) X(I), '    0', Y(IY),
     +                       WRK(IRES)
                        END IF
   80                CONTINUE
  100             CONTINUE
               ELSE
                  WRITE (NOUT,99995) 'E01AEF exits with IFAIL =', IFAIL
               END IF
            END IF
            GO TO 20
         END IF
  120    STOP
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,I4,F20.3)
99997 FORMAT (5X,I4,F12.1,F17.6)
99996 FORMAT (1X,F4.1,A,F12.1,F17.6)
99995 FORMAT (1X,A,I2,A)
      END
```

9.2. Program Data

```
E01AEF Example Program Data
    4       2.0       6.0
    0       2.0       1.0
    1       4.0       2.0      -1.0
    0       5.0       1.0
    2       6.0       2.0       4.0      -2.0
```

9.3. Program Results

```
 E01AEF Example Program Results

 Total number of interpolating conditions =   7

 Interpolating polynomial

    I    Chebyshev Coefficient A(I+1)
    0              9.125
    1             -4.578
    2              0.461
    3              2.852
    4             -2.813
    5              2.227
    6             -0.711
```

X	R	Rth derivative	Residual
2.0	0	1.0	0.000000
4.0	0	2.0	0.000000
	1	-1.0	0.000000
5.0	0	1.0	0.000000
6.0	0	2.0	0.000000
	1	4.0	0.000000
	2	-2.0	0.000000

E01BAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01BAF determines a cubic-spline interpolant to a given set of data.

2. Specification

```
SUBROUTINE E01BAF (M, X, Y, LAMDA, C, LCK, WRK, LWRK, IFAIL)
INTEGER          M, LCK, LWRK, IFAIL
real             X(M), Y(M), LAMDA(LCK), C(LCK), WRK(LWRK)
```

3. Description

This routine determines a cubic spline $s(x)$, defined in the range $x_1 \leq x \leq x_m$, which interpolates (passes exactly through) the set of data points (x_i,y_i), for $i = 1,2,...,m$, where $m \geq 4$ and $x_1 < x_2 < ... < x_m$. Unlike some other spline interpolation algorithms, derivative end conditions are not imposed. The spline interpolant chosen has $m-4$ interior knots $\lambda_5,\lambda_6,...,\lambda_m$, which are set to the values of $x_3,x_4,...,x_{m-2}$ respectively. This spline is represented in its B-spline form (see Cox [1]):

$$s(x) = \sum_{i=1}^{m} c_i N_i(x),$$

where $N_i(x)$ denotes the normalised B-Spline of degree 3, defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$, and c_i denotes its coefficient, whose value is to be determined by the routine.

The use of B-splines requires eight additional knots λ_1, λ_2, λ_3, λ_4, λ_{m+1}, λ_{m+2}, λ_{m+3} and λ_{m+4} to be specified; the routine sets the first four of these to x_1 and the last four to x_m.

The algorithm for determining the coefficients is as described in [1] except that *QR* factorization is used instead of *LU* decomposition. The implementation of the algorithm involves setting up appropriate information for the related routine E02BAF followed by a call of that routine. (For further details of E02BAF, see the routine document.)

Values of the spline interpolant, or of its derivatives or definite integral, can subsequently be computed as detailed in Section 8.

4. References

[1] COX, M.G.
An algorithm for spline interpolation.
J. Inst. Math. Appl., 15, pp. 95-108, 1975.

[2] COX, M.G.
'A survey of numerical methods for data and function approximation'.
In: The State of the Art in Numerical Analysis, D.A.H. Jacobs, (ed).
Academic Press, London, pp. 627-668, 1977.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of data points.

Constraint: M $\geq$ 4.

2: X(M) – ***real*** array. *Input*

On entry: X(i) must be set to x_i, the ith data value of the independent variable x, for $i = 1,2,...,m$.

Constraint: X(i) < X(i+1), for $i = 1,2,...,$M−1.

3: Y(M) – ***real*** array. *Input*

On entry: Y(i) must be set to y_i, the ith data value of the dependent variable y, for $i = 1,2,...,m$.

4: LAMDA(LCK) – ***real*** array. *Output*

On exit: the value of λ_i, the ith knot, for $i = 1,2,...,m+4$.

5: C(LCK) – ***real*** array. *Output*

On exit: the coefficient c_i of the B-spline $N_i(x)$, for $i = 1,2,...,m$. The remaining elements of the array are not used.

6: LCK – INTEGER. *Input*

On entry: the dimension of the arrays LAMDA and C as declared in the (sub)program from which E01BAF is called.

Constraint: LCK ≥ M + 4.

7: WRK(LWRK) – ***real*** array. *Workspace*

8: LWRK – INTEGER. *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E01BAF is called.

Constraint: LWRK ≥ 6×M + 16.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, M < 4,
or LCK < M + 4,
or LWRK < 6×M + 16.

IFAIL = 2

The X-values fail to satisfy the condition

X(1) < X(2) < X(3) < ... < X(M).

7. Accuracy

The rounding errors incurred are such that the computed spline is an exact interpolant for a slightly perturbed set of ordinates $y_i + \delta y_i$. The ratio of the root-mean-square value of the δy_i to that of the y_i is no greater than a small multiple of the relative ***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to m.

All the x_i are used as knot positions except x_2 and x_{m-1}. This choice of knots (see Cox [2]) means that $s(x)$ is composed of $m - 3$ cubic arcs as follows. If $m = 4$, there is just a single arc space spanning the whole interval x_1 to x_4. If $m \geq 5$, the first and last arcs span the intervals x_1 to x_3 and x_{m-2} to x_m respectively. Additionally if $m \geq 6$, the ith arc, for $i = 2,3,...,m-4$ spans the interval x_{i+1} to x_{i+2}.

After the call

```
CALL E01BAF (M, X, Y, LAMDA, C, LCK, WRK, LWRK, IFAIL)
```

the following operations may be carried out on the interpolant $s(x)$.

The value of $s(x)$ at x = XVAL can be provided in the ***real*** variable SVAL by the call

```
CALL E02BBF (M+4, LAMDA, C, XVAL, SVAL, IFAIL)
```

The values of $s(x)$ and its first three derivatives at x = XVAL can be provided in the ***real*** array SDIF of dimension 4, by the call

```
CALL E02BCF (M+4, LAMDA, C, XVAL, LEFT, SDIF, IFAIL)
```

Here LEFT must specify whether the left- or right-hand value of the third derivative is required (see E02BCF for details).

The value of the integral of $s(x)$ over the range x_1 to x_m can be provided in the ***real*** variable SINT by

```
CALL E02BDF (M+4, LAMDA, C, SINT, IFAIL)
```

9. Example

The following example program sets up data from 7 values of the exponential function in the interval 0 to 1. E01BAF is then called to compute a spline interpolant to these data.

The spline is evaluated by E02BBF, at the data points and at points halfway between each adjacent pair of data points, and the spline values and the values of e^x are printed out.

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.

```
*     E01BAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          M, LCK, LWRK
      PARAMETER        (M=7,LCK=M+4,LWRK=6*M+16)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             FIT, XARG
      INTEGER          I, IFAIL, J, R
*     .. Local Arrays ..
      real             C(LCK), LAMDA(LCK), WRK(LWRK), X(M), Y(M)
*     .. External Subroutines ..
      EXTERNAL         E01BAF, E02BBF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Data statements ..
      DATA             (X(I),I=1,M)/0.0e0, 0.2e0, 0.4e0, 0.6e0, 0.75e0,
     +                 0.9e0, 1.0e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01BAF Example Program Results'
      DO 20 I = 1, M
         Y(I) = EXP(X(I))
   20 CONTINUE
      IFAIL = 0
*
      CALL E01BAF(M,X,Y,LAMDA,C,LCK,WRK,LWRK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   J    Knot LAMDA(J+2)   B-spline coeff C(J)'
      WRITE (NOUT,*)
      J = 1
      WRITE (NOUT,99998) J, C(1)
      DO 40 J = 2, M - 1
         WRITE (NOUT,99999) J, LAMDA(J+2), C(J)
```

```
   40 CONTINUE
      WRITE (NOUT,99998) M, C(M)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '    R        Abscissa            Ordinate              Spline'
      WRITE (NOUT,*)
      DO 60 R = 1, M
         IFAIL = 0
*
         CALL E02BBF(M+4,LAMDA,C,X(R),FIT,IFAIL)
*
         WRITE (NOUT,99999) R, X(R), Y(R), FIT
         IF (R.LT.M) THEN
            XARG = 0.5e0*(X(R)+X(R+1))
*
            CALL E02BBF(M+4,LAMDA,C,XARG,FIT,IFAIL)
*
            WRITE (NOUT,99997) XARG, FIT
         END IF
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,I4,F15.4,2F20.4)
99998 FORMAT (1X,I4,F35.4)
99997 FORMAT (1X,F19.4,F40.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E01BAF Example Program Results

    J    Knot LAMDA(J+2)   B-spline coeff C(J)

    1                           1.0000
    2         0.0000            1.1336
    3         0.4000            1.3726
    4         0.6000            1.7827
    5         0.7500            2.1744
    6         1.0000            2.4918
    7                           2.7183

    R        Abscissa           Ordinate            Spline

    1         0.0000            1.0000              1.0000
              0.1000                                1.1052
    2         0.2000            1.2214              1.2214
              0.3000                                1.3498
    3         0.4000            1.4918              1.4918
              0.5000                                1.6487
    4         0.6000            1.8221              1.8221
              0.6750                                1.9640
    5         0.7500            2.1170              2.1170
              0.8250                                2.2819
    6         0.9000            2.4596              2.4596
              0.9500                                2.5857
    7         1.0000            2.7183              2.7183
```

E01BEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01BEF computes a monotonicity-preserving piecewise cubic Hermite interpolant to a set of data points.

2. Specification

```
SUBROUTINE E01BEF (N, X, F, D, IFAIL)
INTEGER          N, IFAIL
real             X(N), F(N), D(N)
```

3. Description

This routine estimates first derivatives at the set of data points (x_r,f_r), for $r = 1,2,...,n$, which determine a piecewise cubic Hermite interpolant to the data, that preserves monotonicity over ranges where the data points are monotonic. If the data points are only piecewise monotonic, the interpolant will have an extremum at each point where monotonicity switches direction. The estimates of the derivatives are computed by a formula due to Brodlie, which is described in Fritsch and Butland [1], with suitable changes at the boundary points.

The routine is derived from routine PCHIM in Fritsch [2].

Values of the computed interpolant, and of its first derivative and definite integral, can subsequently be computed by calling E01BFF, E01BGF and E01BHF, as described in Section 8.

4. References

[1] FRITSCH, F.N. and BUTLAND, J.
A method for constructing local monotone piecewise cubic interpolants.
SIAM J. Sci. Stat. Comput., 5, pp. 300-304, 1984.

[2] FRITSCH, F.N.
PCHIP Final Specifications.
Lawrence Livermore National Laboratory report UCID-30194, August 1982.

5. Parameters

1: N – INTEGER. *Input*

On entry: n, the number of data points.

Constraint: N ≥ 2.

2: X(N) – ***real*** array. *Input*

On entry: X(r) must be set to x_r, the rth value of the independent variable (abscissa), for $r = 1,2,...,n$.

Constraint: X(r) < X(r+1).

3: F(N) – ***real*** array. *Input*

On entry: F(r) must be set to f_r, the rth value of the dependent variable (ordinate), for $r = 1,2,...,n$.

4: D(N) – ***real*** array. *Output*

On exit: estimates of derivatives at the data points. D(r) contains the derivative at X(r).

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, N < 2.

IFAIL = 2

The values of X(r), for r = 1,2,...,N, are not in strictly increasing order.

7. Accuracy

The computational errors in the array D should be negligible in most practical situations.

8. Further Comments

The time taken by the routine is approximately proportional to n.

The values of the computed interpolant at the points PX(i), for i = 1,2,...,M, may be obtained in the ***real*** array PF, of length at least M, by the call:

```
CALL E01BFF(N,X,F,D,M,PX,PF,IFAIL)
```

where N, X and F are the input parameters to E01BEF and D is the output parameter from E01BEF.

The values of the computed interpolant at the points PX(i), for i = 1,2,...,M, together with its first derivatives, may be obtained in the ***real*** arrays PF and PD, both of length at least M, by the call:

```
CALL E01BGF(N,X,F,D,M,PX,PF,PD,IFAIL)
```

where N, X, F and D are as described above.

The value of the definite integral of the interpolant over the interval A to B can be obtained in the ***real*** variable PINT by the call:

```
CALL E01BHF(N,X,F,D,A,B,PINT,IFAIL)
```

where N, X, F and D are as described above.

9. Example

This example program reads in a set of data points, calls E01BEF to compute a piecewise monotonic interpolant, and then calls E01BFF to evaluate the interpolant at equally spaced points.

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.

```
*     E01BEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=50,NMAX=50)
```

```
*     .. Local Scalars ..
      real               STEP
      INTEGER            I, IFAIL, M, N, R
*     .. Local Arrays ..
      real               D(NMAX), F(NMAX), PF(MMAX), PX(MMAX), X(NMAX)
*     .. External Subroutines ..
      EXTERNAL           E01BEF, E01BFF
*     .. Intrinsic Functions ..
      INTRINSIC          MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01BEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         DO 20 R = 1, N
            READ (NIN,*) X(R), F(R)
   20    CONTINUE
         IFAIL = 0
*
         CALL E01BEF(N,X,F,D,IFAIL)
*
         READ (NIN,*) M
         IF (M.GT.0 .AND. M.LE.MMAX) THEN
*           Compute M equally spaced points from X(1) to X(N).
            STEP = (X(N)-X(1))/(M-1)
            DO 40 I = 1, M
               PX(I) = MIN(X(1)+(I-1)*STEP,X(N))
   40       CONTINUE
            IFAIL = 0
*
            CALL E01BFF(N,X,F,D,M,PX,PF,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*) '                  Interpolated'
            WRITE (NOUT,*) '     Abscissa           Value'
            DO 60 I = 1, M
               WRITE (NOUT,99999) PX(I), PF(I)
   60       CONTINUE
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,F13.4,2X,F13.4)
      END
```

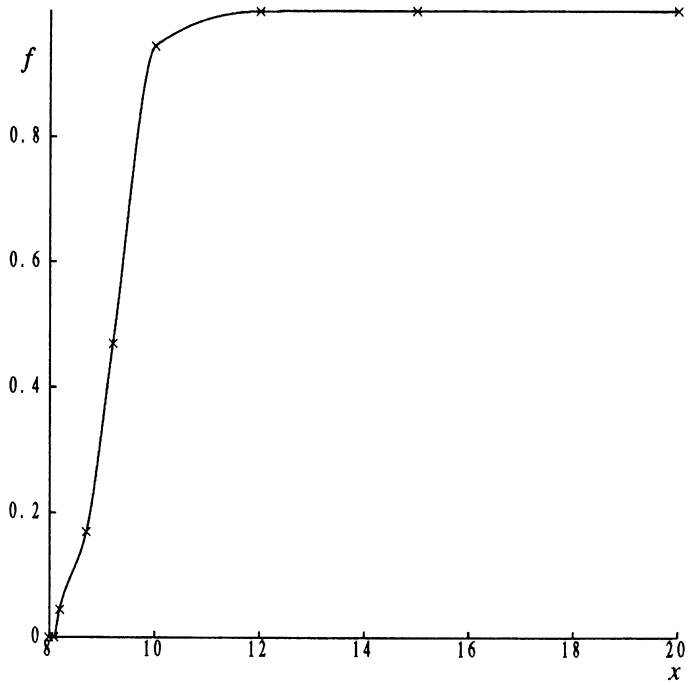
9.2. Program Data

```
E01BEF Example Program Data
   9                N, the number of data points
  7.99   0.00000E+0  X(R), F(R), independent and dependent variable
  8.09   0.27643E-4
  8.19   0.43750E-1
  8.70   0.16918E+0
  9.20   0.46943E+0
 10.00   0.94374E+0
 12.00   0.99864E+0
 15.00   0.99992E+0
 20.00   0.99999E+0  End of data points
  11                M, the number of evaluation points
```

9.3. Program Results

```
E01BEF Example Program Results

                Interpolated
     Abscissa           Value
       7.9900          0.0000
       9.1910          0.4640
      10.3920          0.9645
      11.5930          0.9965
      12.7940          0.9992
      13.9950          0.9998
      15.1960          0.9999
      16.3970          1.0000
      17.5980          1.0000
      18.7990          1.0000
      20.0000          1.0000
```



E01BFF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01BFF evaluates a piecewise cubic Hermite interpolant at a set of points.

2. Specification

```
SUBROUTINE E01BFF (N, X, F, D, M, PX, PF, IFAIL)
INTEGER          N, M, IFAIL
real             X(N), F(N), D(N), PX(M), PF(M)
```

3. Description

This routine evaluates a piecewise cubic Hermite interpolant, as computed by E01BEF, at the points PX(i), for i = 1,2,...,m. If any point lies outside the interval from X(1) to X(N), a value is extrapolated from the nearest extreme cubic, and a warning is returned.

The routine is derived from routine PCHFE in Fritsch [1].

4. References

[1] FRITSCH, F.N.
PCHIP Final Specifications.
Lawrence Livermore National Laboratory report UCID-30194, August 1982.

5. Parameters

1: N – INTEGER. *Input*

2: X(N) – ***real*** array. *Input*

3: F(N) – ***real*** array. *Input*

4: D(N) – ***real*** array. *Input*

On entry: N, X, F and D must be unchanged from the previous call of E01BEF.

5: M – INTEGER. *Input*

On entry: m, the number of points at which the interpolant is to be evaluated.

Constraint: M ≥ 1.

6: PX(M) – ***real*** array. *Input*

On entry: the m values of x at which the interpolant is to be evaluated.

7: PF(M) – ***real*** array. *Output*

On exit: PF(i) contains the value of the interpolant evaluated at the point PX(i), for i = 1,2,...,m.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, N < 2.

IFAIL = 2

The values of X(r), for r = 1,2,...,N, are not in strictly increasing order.

IFAIL = 3

On entry, M < 1.

IFAIL = 4

At least one of the points PX(i), for i = 1,2,...,M, lies outside the interval [X(1),X(N)], and extrapolation was performed at all such points. Values computed at such points may be very unreliable.

7. Accuracy

The computational errors in the array PF should be negligible in most practical situations.

8. Further Comments

The time taken by the routine is approximately proportional to the number of evaluation points, m. The evaluation will be most efficient if the elements of PX are in non-decreasing order (or, more generally, if they are grouped in increasing order of the intervals [X(r−1),X(r)]). A single call of E01BFF with m > 1 is more efficient than several calls with m = 1.

9. Example

This example program reads in values of N, X, F and D, and then calls E01BFF to evaluate the interpolant at equally spaced points.

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.

```
*     E01BFF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=21,NMAX=50)
*     .. Local Scalars ..
      real             STEP
      INTEGER          I, IFAIL, M, N, R
*     .. Local Arrays ..
      real             D(NMAX), F(NMAX), PF(MMAX), PX(MMAX), X(NMAX)
*     .. External Subroutines ..
      EXTERNAL         E01BFF
*     .. Intrinsic Functions ..
      INTRINSIC        MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01BFF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         DO 20 R = 1, N
            READ (NIN,*) X(R), F(R), D(R)
```

```
   20    CONTINUE
         READ (NIN,*) M
         IF (M.GT.0 .AND. M.LE.MMAX) THEN
*           Compute M equally spaced points from X(1) to X(N).
            STEP = (X(N)-X(1))/(M-1)
            DO 40 I = 1, M
               PX(I) = MIN(X(1)+(I-1)*STEP,X(N))
   40       CONTINUE
            IFAIL = 0
*
            CALL E01BFF(N,X,F,D,M,PX,PF,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*) '                    Interpolated'
            WRITE (NOUT,*) '       Abscissa          Value'
            DO 60 I = 1, M
               WRITE (NOUT,99999) PX(I), PF(I)
   60       CONTINUE
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,3F15.4)
      END
```

9.2. Program Data

```
E01BFF Example Program Data
   9                                N, the number of data points
  7.990  0.00000E+0  0.00000E+0  X(R), F(R), D(R)
  8.090  0.27643E-4  5.52510E-4
  8.190  0.43749E-1  0.33587E+0
  8.700  0.16918E+0  0.34944E+0
  9.200  0.46943E+0  0.59696E+0
  10.00  0.94374E+0  6.03260E-2
  12.00  0.99864E+0  8.98335E-4
  15.00  0.99992E+0  2.93954E-5
  20.00  0.99999E+0  0.00000E+0  End of data points
   11                             M, the number of evaluation points
```

9.3. Program Results

```
 E01BFF Example Program Results

                  Interpolated
       Abscissa          Value
         7.9900         0.0000
         9.1910         0.4640
        10.3920         0.9645
        11.5930         0.9965
        12.7940         0.9992
        13.9950         0.9998
        15.1960         0.9999
        16.3970         1.0000
        17.5980         1.0000
        18.7990         1.0000
        20.0000         1.0000
```

E01BGF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01BGF evaluates a piecewise cubic Hermite interpolant and its first derivative at a set of points.

2. Specification

```
SUBROUTINE E01BGF (N, X, F, D, M, PX, PF, PD, IFAIL)
INTEGER          N, M, IFAIL
real             X(N), F(N), D(N), PX(M), PF(M), PD(M)
```

3. Description

This routine evaluates a piecewise cubic Hermite interpolant, as computed by E01BEF, at the points PX(i), for $i = 1,2,...,m$. The first derivatives at the points are also computed. If any point lies outside the interval from X(1) to X(N), values of the interpolant and its derivative are extrapolated from the nearest extreme cubic, and a warning is returned.

If values of the interpolant only, and not of its derivative, are required, E01BFF should be used.

The routine is derived from routine PCHFD in Fritsch [1].

4. References

[1] FRITSCH, F.N.
PCHIP Final Specifications.
Lawrence Livermore National Laboratory report UCID-30194, August 1982.

5. Parameters

1: N – INTEGER. *Input*

2: X(N) – ***real*** array. *Input*

3: F(N) – ***real*** array. *Input*

4: D(N) – ***real*** array. *Input*

On entry: N, X, F and D must be unchanged from the previous call of E01BEF.

5: M – INTEGER. *Input*

On entry: m, the number of points at which the interpolant is to be evaluated.

Constraint: M ≥ 1.

6: PX(M) – ***real*** array. *Input*

On entry: the m values of x at which the interpolant is to be evaluated.

7: PF(M) – ***real*** array. *Output*

On exit: PF(i) contains the value of the interpolant evaluated at the point PX(i), for $i = 1,2,...,m$.

8: PD(M) – ***real*** array. *Output*

On exit: PD(i) contains the first derivative of the interpolant evaluated at the point PX(i), for $i = 1,2,...,m$.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, N < 2.

IFAIL = 2

The values of X(r), for r = 1,2,...,N, are not in strictly increasing order.

IFAIL = 3

On entry, M < 1.

IFAIL = 4

At least one of the points PX(i), for i = 1,2,...,M, lies outside the interval [X(1),X(N)], and extrapolation was performed at all such points. Values computed at these points may be very unreliable.

7. Accuracy

The computational errors in the arrays PF and PD should be negligible in most practical situations.

8. Further Comments

The time taken by the routine is approximately proportional to the number of evaluation points, m. The evaluation will be most efficient if the elements of PX are in non-decreasing order (or, more generally, if they are grouped in increasing order of the intervals [X(r–1),X(r)]). A single call of E01BGF with m > 1 is more efficient than several calls with m = 1.

9. Example

This example program reads in values of N, X, F and D, and calls E01BGF to compute the values of the interpolant and its derivative at equally spaced points.

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.

```
*     E01BGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=21,NMAX=50)
*     .. Local Scalars ..
      real             STEP
      INTEGER          I, IFAIL, M, N, R
*     .. Local Arrays ..
      real             D(NMAX), F(NMAX), PD(MMAX), PF(MMAX), PX(MMAX),
     +                 X(NMAX)
*     .. External Subroutines ..
      EXTERNAL         E01BGF
*     .. Intrinsic Functions ..
      INTRINSIC        MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01BGF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         DO 20 R = 1, N
            READ (NIN,*) X(R), F(R), D(R)
   20    CONTINUE
         READ (NIN,*) M
         IF (M.GT.0 .AND. M.LE.MMAX) THEN
*           Compute M equally spaced points from X(1) to X(N).
            STEP = (X(N)-X(1))/(M-1)
            DO 40 I = 1, M
               PX(I) = MIN(X(1)+(I-1)*STEP,X(N))
   40       CONTINUE
            IFAIL = 0
*
            CALL E01BGF(N,X,F,D,M,PX,PF,PD,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '                        Interpolated   Interpolated'
            WRITE (NOUT,*)
     +        '       Abscissa           Value     Derivative'
            DO 60 I = 1, M
               WRITE (NOUT,99999) PX(I), PF(I), PD(I)
   60       CONTINUE
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,2F15.4,1P,e15.3)
      END
```

9.2. Program Data

```
E01BGF Example Program Data
   9                                N, the number of data points
 7.990   0.00000E+0   0.00000E+0   X(R), F(R), D(R)
 8.090   0.27643E-4   5.52510E-4
 8.190   0.43749E-1   0.33587E+0
 8.700   0.16918E+0   0.34944E+0
 9.200   0.46943E+0   0.59696E+0
 10.00   0.94374E+0   6.03260E-2
 12.00   0.99864E+0   8.98335E-4
 15.00   0.99992E+0   2.93954E-5
 20.00   0.99999E+0   0.00000E+0   End of data points
  11                               M, the number of evaluation points
```

9.3. Program Results

```
 E01BGF Example Program Results

                  Interpolated   Interpolated
       Abscissa          Value     Derivative
         7.9900         0.0000      0.000E+00
         9.1910         0.4640      6.060E-01
        10.3920         0.9645      4.569E-02
        11.5930         0.9965      9.917E-03
        12.7940         0.9992      6.249E-04
        13.9950         0.9998      2.708E-04
        15.1960         0.9999      2.809E-05
        16.3970         1.0000      2.034E-05
        17.5980         1.0000      1.308E-05
        18.7990         1.0000      6.297E-06
        20.0000         1.0000     -9.529E-22
```

E01BHF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01BHF evaluates the definite integral of a piecewise cubic Hermite interpolant over the interval $[a,b]$.

2. Specification

```
SUBROUTINE E01BHF (N, X, F, D, A, B, PINT, IFAIL)
INTEGER          N, IFAIL
real             X(N), F(N), D(N), A, B, PINT
```

3. Description

This routine evaluates the definite integral of a piecewise cubic Hermite interpolant, as computed by E01BEF, over the interval $[a,b]$.

If either a or b lies outside the interval from X(1) to X(N) computation of the integral involves extrapolation and a warning is returned.

The routine is derived from routine PCHIA in Fritsch [1].

4. References

[1] FRITSCH, F.N.
PCHIP Final Specifications.
Lawrence Livermore National Laboratory report UCID-30194, August 1982.

5. Parameters

1: N – INTEGER. *Input*

2: X(N) – ***real*** array. *Input*

3: F(N) – ***real*** array. *Input*

4: D(N) – ***real*** array. *Input*

On entry: N, X, F and D must be unchanged from the previous call of E01BEF.

5: A – ***real***. *Input*

6: B – ***real***. *Input*

On entry: the interval $[a,b]$ over which integration is to be performed.

7: PINT – ***real***. *Output*

On exit: the value of the definite integral of the interpolant over the interval $[a,b]$.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, N < 2.

IFAIL = 2

The values of X(r), for r = 1,2,...,N, are not in strictly increasing order.

IFAIL = 3

On entry, at least one of A or B lies outside the interval [X(1),X(N)], and extrapolation was performed to compute the integral. The value returned is therefore unreliable.

7. Accuracy

The computational error in the value returned for PINT should be negligible in most practical situations.

8. Further Comments

The time taken by the routine is approximately proportional to the number of data points included within the interval [a,b].

9. Example

This example program reads in values of N, X, F and D. It then reads in pairs of values for A and B, and evaluates the definite integral of the interpolant over the interval [A,B] until end-of-file is reached.

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.

```
*     E01BHF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NMAX
      PARAMETER        (NMAX=50)
*     .. Local Scalars ..
      real             A, B, PINT
      INTEGER          IFAIL, N, R
*     .. Local Arrays ..
      real             D(NMAX), F(NMAX), X(NMAX)
*     .. External Subroutines ..
      EXTERNAL         E01BHF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01BHF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         DO 20 R = 1, N
            READ (NIN,*) X(R), F(R), D(R)
   20    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) '                                    Integral'
         WRITE (NOUT,*) '          A            B      over (A,B)'
*        Read A, B pairs until end of file and compute
*        definite integrals
   40    READ (NIN,*,END=60) A, B
         IFAIL = 0
*
```

```
         CALL E01BHF(N,X,F,D,A,B,PINT,IFAIL)
*
         WRITE (NOUT,99999) A, B, PINT
         GO TO 40
      END IF
   60 STOP
*
99999 FORMAT (1X,3F13.4)
      END
```

9.2. Program Data

```
E01BHF Example Program Data
 9                                          N, the number of data points
   7.990  0.00000E+0  0.00000E+0  X(R), F(R), D(R)
   8.090  0.27643E-4  5.52510E-4
   8.190  0.43749E-1  0.33587E+0
   8.700  0.16918E+0  0.34944E+0
   9.200  0.46943E+0  0.59696E+0
   10.00  0.94374E+0  6.03260E-2
   12.00  0.99864E+0  8.98335E-4
   15.00  0.99992E+0  2.93954E-5
   20.00  0.99999E+0  0.00000E+0
   7.99       20.0                 A, B pairs until end of file
   10.0       12.0
   12.0       10.0
   15.0       15.0
```

9.3. Program Results

```
 E01BHF Example Program Results

                                 Integral
         A            B        over (A,B)
      7.9900      20.0000         10.7648
     10.0000      12.0000          1.9622
     12.0000      10.0000         -1.9622
     15.0000      15.0000          0.0000
```

E01DAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01DAF computes a bicubic spline interpolating surface through a set of data values, given on a rectangular grid in the x-y plane.

2. Specification

```
SUBROUTINE E01DAF (MX, MY, X, Y, F, PX, PY, LAMDA, MU, C, WRK, IFAIL)
INTEGER          MX, MY, PX, PY, IFAIL
real             X(MX), Y(MY), F(MX*MY), LAMDA(MX+4), MU(MX+4), C(MX*MY),
1                WRK((MX+6)*(MY+6))
```

3. Description

This routine determines a bicubic spline interpolant to the set of data points $(x_q,y_r,f_{q,r})$, for $q = 1,2,...,m_x$; $r = 1,2,...,m_y$. The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{m_x}\sum_{j=1}^{m_y} c_{ij}M_i(x)N_j(y),$$

such that

$$s(x_q,y_r) = f_{q,r} ,$$

where $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}, and the c_{ij} are the spline coefficients. These knots, as well as the coefficients, are determined by the routine, which is derived from the routine B2IRE in Anthony *et al.* [1]. The method used is described in Section 8.2.

For further information on splines, see Hayes and Halliday [4] for bicubic splines and de Boor [3] for normalised B-splines.

Values of the computed spline can subsequently be obtained by calling E02DEF or E02DFF as described in Section 8.3.

4. References

[1] ANTHONY, G.T., COX, M.G. and HAYES, J.G.
DASL – Data Approximation Subroutine Library, National Physical Laboratory, 1982.

[2] COX, M.G.
An algorithm for spline interpolation.
J. Inst. Maths. Applics., 15, pp. 95–108, 1975.

[3] DE BOOR, C.
On Calculating with B-splines.
J. Approx. Theory, 6, pp. 50-62, 1972.

[4] HAYES, J.G. and HALLIDAY, J.
The Least-squares Fitting of Cubic Spline Surfaces to General Data Sets.
J. Inst. Maths. Applics., 14, pp. 89-103, 1974.

5. Parameters

1: MX – INTEGER. *Input*

2: MY – INTEGER. *Input*

On entry: MX and MY must specify m_x and m_y respectively, the number of points along the x and y axis that define the rectangular grid.

Constraint: MX $\geq$ 4 and MY $\geq$ 4.

3: X(MX) – ***real*** array. *Input*
4: Y(MY) – ***real*** array. *Input*

On entry: X(q) and Y(r) must contain x_q, for $q = 1,2,...,m_x$, and y_r, for $r = 1,2,...,m_y$, respectively.

Constraints: X(q) < X(q+1), for $q = 1,2,...,m_x-1$,
Y(r) < Y(r+1), for $r = 1,2,...,m_y-1$.

5: F(MX*MY) – ***real*** array. *Input*

On entry: F($m_y\times(q-1)+r$) must contain $f_{q,r}$, for $q = 1,2,...,m_x$; $r = 1,2,...,m_y$.

6: PX – INTEGER. *Output*
7: PY – INTEGER. *Output*

On exit: PX and PY contain $m_x + 4$ and $m_y + 4$, the total number of knots of the computed spline with respect to the x and y variables, respectively.

8: LAMDA(MX+4) – ***real*** array. *Output*
9: MU(MY+4) – ***real*** array. *Output*

On exit: LAMDA contains the complete set of knots λ_i associated with the x variable, i.e. the interior knots LAMDA(5), LAMDA(6),...,LAMDA(PX−4), as well as the additional knots LAMDA(1) = LAMDA(2) = LAMDA(3) = LAMDA(4) = X(1) and LAMDA(PX−3) = LAMDA(PX−2) = LAMDA(PX−1) = LAMDA(PX) = X(MX) needed for the B-spline representation. MU contains the corresponding complete set of knots μ_i associated with the y variable.

10: C(MX*MY) – ***real*** array. *Output*

On exit: the coefficients of the spline interpolant. C($m_y\times(i-1)+j$) contains the coefficient c_{ij} described in Section 3.

11: WRK((MX+6)*(MY+6)) – ***real*** array. *Workspace*

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, MX < 4,
or MY < 4.

IFAIL = 2

On entry, either the values in the X array or the values in the Y array are not in increasing order.

IFAIL = 3

A system of linear equations defining the B-spline coefficients was singular; the problem is too ill-conditioned to permit solution.

7. Accuracy

The main sources of rounding errors are in steps (2), (3), (6) and (7) of the algorithm described in Section 8.2. It can be shown (Cox [2]) that the matrix A_x formed in step (2) has elements differing relatively from their true values by at most a small multiple of 3ϵ, where ϵ is the ***machine precision***. A_x is 'totally positive', and a linear system with such a coefficient matrix can be solved quite safely by elimination without pivoting. Similar comments apply to steps (6) and (7). Thus the complete process is numerically stable.

8. Further Comments

8.1. Timing

The time taken by this routine is approximately proportional to $m_x m_y$.

8.2. Outline of method used

The process of computing the spline consists of the following steps:

(1) choice of the interior x-knots $\lambda_5, \lambda_6, \ldots, \lambda_{m_x}$ as $\lambda_i = x_{i-2}$, for $i = 5,6,\ldots,m_x$,

(2) formation of the system

$$A_x E = F,$$

where A_x is a band matrix of order m_x and bandwidth 4, containing in its qth row the values at x_q of the B-splines in x, F is the m_x by m_y rectangular matrix of values $f_{q,r}$, and E denotes an m_x by m_y rectangular matrix of intermediate coefficients,

(3) use of Gaussian elimination to reduce this system to band triangular form,

(4) solution of this triangular system for E,

(5) choice of the interior y knots $\mu_5, \mu_6, \ldots, \mu_{m_y}$ as $\mu_i = y_{i-2}$, for $i = 5,6,\ldots,m_y$,

(6) formation of the system

$$A_y C^T = E^T,$$

where A_y is the counterpart of A_x for the y variable, and C denotes the m_x by m_y rectangular matrix of values of c_{ij},

(7) use of Gaussian elimination to reduce this system to band triangular form,

(8) solution of this triangular system for C^T and hence C.

For computational convenience, steps (2) and (3), and likewise steps (6) and (7), are combined so that the formation of A_x and A_y and the reductions to triangular form are carried out one row at a time.

8.3. Evaluation of Computed Spline

The values of the computed spline at the points (TX(r),TY(r)), for $r = 1,2,\ldots,$N, may be obtained in the ***real*** array FF, of length at least N, by the following call:

```
IFAIL = 0
CALL E02DEF(N,PX,PY,TX,TY,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
```

where PX, PY, LAMDA, MU and C are the output parameters of E01DAF, WRK is a ***real*** workspace array of length at least PY−4, and IWRK is an integer workspace array of length at least PY−4.

To evaluate the computed spline on an NX by NY rectangular grid of points in the x-y plane, which is defined by the x co-ordinates stored in TX(q), for $q = 1,2,\ldots,$NX, and the y co-ordinates stored in TY(r), for $r = 1,2,\ldots,$NY, returning the results in the ***real*** array FG which is of length at least NX×NY, the following call may be used:

```
IFAIL = 0
CALL E02DFF(NX,NY,PX,PY,TX,TY,LAMDA,MU,C,FG,WRK,LWRK,
*            IWRK,LIWRK,IFAIL)
```

where PX, PY, LAMDA, MU and C are the output parameters of E01DAF, WRK is a ***real***

workspace array of length at least LWRK = min(NWRK1,NWRK2), NWRK1 = NX×4 + PX, NWRK2 = NY×4 + PY, and IWRK is an integer workspace array of length at least LIWRK = NY + PY − 4 if NWRK1 > NWRK2, or NX + PX − 4 otherwise. The result of the spline evaluated at grid point (q,r) is returned in element $(NY\times(q-1)+r)$ of the array FG.

9. Example

This program reads in values of m_x, x_q for $q = 1,2,...,m_x$, m_y and y_r for $r = 1,2,...,m_y$, followed by values of the ordinates $f_{q,r}$ defined at the grid points (x_q,y_r). It then calls E01DAF to compute a bicubic spline interpolant of the data values, and prints the values of the knots and B-spline coefficients. Finally it evaluates the spline at a small sample of points on a rectangular grid.

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.

```
*     E01DAF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MXMAX, MYMAX
      PARAMETER        (MXMAX=20,MYMAX=MXMAX)
      INTEGER          LIWRK, LWRK
      PARAMETER        (LIWRK=MXMAX+2*(MXMAX-3)*(MYMAX-3),LWRK=(MXMAX+6)
     +                 *(MYMAX+6))
*     .. Local Scalars ..
      real             STEP, XHI, XLO, YHI, YLO
      INTEGER          I, IFAIL, J, MX, MY, NX, NY, PX, PY
*     .. Local Arrays ..
      real             C(MXMAX*MYMAX), F(MXMAX*MYMAX), FG(MXMAX*MYMAX),
     +                 LAMDA(MXMAX+4), MU(MYMAX+4), TX(MXMAX),
     +                 TY(MYMAX), WRK(LWRK), X(MXMAX), Y(MYMAX)
      INTEGER          IWRK(LIWRK)
      CHARACTER*10     CLABS(MYMAX), RLABS(MXMAX)
*     .. External Subroutines ..
      EXTERNAL         E01DAF, E02DFF, X04CBF
*     .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01DAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Read the number of X points, MX, and the values of the
*     X co-ordinates.
      READ (NIN,*) MX
      READ (NIN,*) (X(I),I=1,MX)
*     Read the number of Y points, MY, and the values of the
*     Y co-ordinates.
      READ (NIN,*) MY
      READ (NIN,*) (Y(I),I=1,MY)
*     Read the function values at the grid points.
      DO 20 J = 1, MY
         READ (NIN,*) (F(MY*(I-1)+J),I=1,MX)
   20 CONTINUE
      IFAIL = 0
*
*     Generate the (X,Y,F) interpolating bicubic B-spline.
      CALL E01DAF(MX,MY,X,Y,F,PX,PY,LAMDA,MU,C,WRK,IFAIL)
*
```

```
*     Print the knot sets, LAMDA and MU.
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '                I   Knot LAMDA(I)       J     Knot MU(J)'
      DO 40 J = 4, MAX(PX,PY) - 3
         IF (J.LE.PX-3 .AND. J.LE.PY-3) THEN
            WRITE (NOUT,99997) J, LAMDA(J), J, MU(J)
         ELSE IF (J.LE.PX-3) THEN
            WRITE (NOUT,99997) J, LAMDA(J)
         ELSE IF (J.LE.PY-3) THEN
            WRITE (NOUT,99996) J, MU(J)
         END IF
   40 CONTINUE
*     Print the spline coefficients.
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The B-Spline coefficients:'
      WRITE (NOUT,99999) (C(I),I=1,MX*MY)
      WRITE (NOUT,*)
*     Evaluate the spline on a regular rectangular grid at NX*NY
*     points over the domain (XLO to XHI) x (YLO to YHI).
      READ (NIN,*) NX, XLO, XHI
      READ (NIN,*) NY, YLO, YHI
      IF (NX.LE.MXMAX .AND. NY.LE.MYMAX) THEN
         STEP = (XHI-XLO)/(NX-1)
         DO 60 I = 1, NX
*           Generate NX equispaced X co-ordinates.
            TX(I) = MIN(XLO+(I-1)*STEP,XHI)
*           Generate X axis labels for printing results.
            WRITE (CLABS(I),99998) TX(I)
   60    CONTINUE
         STEP = (YHI-YLO)/(NY-1)
         DO 80 I = 1, NY
            TY(I) = MIN(YLO+(I-1)*STEP,YHI)
            WRITE (RLABS(I),99998) TY(I)
   80    CONTINUE
*
*        Evaluate the spline.
         CALL E02DFF(NX,NY,PX,PY,TX,TY,LAMDA,MU,C,FG,WRK,LWRK,IWRK,
     +               LIWRK,IFAIL)
*
*        Print the results.
         CALL X04CBF('General','X',NY,NX,FG,NY,'F8.3',
     +          'Spline evaluated on a regular mesh (X across, Y down):'
     +               ,'Character',RLABS,'Character',CLABS,80,0,IFAIL)
*
      END IF
      STOP
*
99999 FORMAT (1X,8F9.4)
99998 FORMAT (F5.2)
99997 FORMAT (1X,I16,F12.4,I11,F12.4)
99996 FORMAT (1X,I39,F12.4)
      END
```

9.2. Program Data

```
E01DAF Example Program Data
  7                                                    MX
  1.00  1.10  1.30  1.50  1.60  1.80  2.00             X(1) .. X(MX)
  6                                                    MY
  0.00  0.10  0.40  0.70  0.90  1.00                   Y(1) .. Y(MY)
  1.00  1.21  1.69  2.25  2.56  3.24  4.00             (F(MY*(I-1)+J),I=1..MX),J=1..MY
  1.10  1.31  1.79  2.35  2.66  3.34  4.10
  1.40  1.61  2.09  2.65  2.96  3.64  4.40
  1.70  1.91  2.39  2.95  3.26  3.94  4.70
  1.90  2.11  2.59  3.15  3.46  4.14  4.90
  2.00  2.21  2.69  3.25  3.56  4.24  5.00
  6  1.0  2.0                                         NX  XLO  XHI
  6  0.0  1.0                                         NY  YLO  YHI
```

9.3. Program Results

```
E01DAF Example Program Results

                I    Knot LAMDA(I)       J      Knot MU(J)
                4       1.0000           4       0.0000
                5       1.3000           5       0.4000
                6       1.5000           6       0.7000
                7       1.6000           7       1.0000
                8       2.0000

The B-Spline coefficients:
   1.0000   1.1333   1.3667   1.7000   1.9000   2.0000   1.2000   1.3333
   1.5667   1.9000   2.1000   2.2000   1.5833   1.7167   1.9500   2.2833
   2.4833   2.5833   2.1433   2.2767   2.5100   2.8433   3.0433   3.1433
   2.8667   3.0000   3.2333   3.5667   3.7667   3.8667   3.4667   3.6000
   3.8333   4.1667   4.3667   4.4667   4.0000   4.1333   4.3667   4.7000
   4.9000   5.0000

Spline evaluated on a regular mesh (X across, Y down):
          1.00     1.20     1.40     1.60     1.80     2.00
0.00     1.000    1.440    1.960    2.560    3.240    4.000
0.20     1.200    1.640    2.160    2.760    3.440    4.200
0.40     1.400    1.840    2.360    2.960    3.640    4.400
0.60     1.600    2.040    2.560    3.160    3.840    4.600
0.80     1.800    2.240    2.760    3.360    4.040    4.800
1.00     2.000    2.440    2.960    3.560    4.240    5.000
```

E01RAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01RAF produces, from a set of function values and corresponding abscissae, the coefficients of an interpolating rational function expressed in continued fraction form.

2. Specification

```
SUBROUTINE E01RAF (N, X, F, M, A, U, IW, IFAIL)
INTEGER          N, M, IW(N), IFAIL
real             X(N), F(N), A(N), U(N)
```

3. Description

E01RAF produces the parameters of a rational function $R(x)$ which assumes prescribed values f_i at prescribed values x_i of the independent variable x, for $i = 1,2,...,n$. More specifically, E01RAF determines the parameters a_j, for $j = 1,2,...,m$ and u_j, $j = 1,2,...,m-1$, in the continued fraction

$$R(x) = a_1 + R_m(x) \qquad (1)$$

where

$$R_i(x) = \frac{a_{m-i+2}(x-u_{m-i+1})}{1 + R_{i-1}(x)}, \qquad \text{for } i = m,m-1,...,2,$$

and

$$R_1(x) = 0,$$

such that $R(x_i) = f_i$, for $i = 1,2,...,n$. The value of m in (1) is determined by the routine; normally $m = n$. The values of u_j form a re-ordered subset of the values of x_i and their ordering is designed to ensure that a representation of the form (1) is determined whenever one exists.

The subsequent evaluation of (1) for given values of x can be carried out using E01RBF.

The computational method employed in E01RAF is the modification of the Thacher-Tukey algorithm described in Graves-Morris and Hopkins [1].

4. References

[1] GRAVES-MORRIS, P.R. and HOPKINS, T.R.
Reliable Rational Interpolation.
Numer. Math., 36, pp. 111-128, 1981.

5. Parameters

1: N – INTEGER. *Input*

On entry: n, the number of data points.

Constraint: N > 0.

2: X(N) – ***real*** array. *Input*

On entry: X(i) must be set to the value of the ith data abscissa, x_i, for $i = 1,2,...,n$.

Constraint: the X(i) must be distinct.

3: F(N) – ***real*** array. *Input*

On entry: F(i) must be set to the value of the data ordinate, f_i, corresponding to x_i, for $i = 1,2,...,n$.

4: M – INTEGER. *Output*

On exit: m, the number of terms in the continued fraction representation of $R(x)$.

5: A(N) – ***real*** array. *Output*

On exit: the value of the parameter a_j in $R(x)$, for $j = 1,2,...,m$. The remaining elements of A, if any, are set to zero.

6: U(N) – ***real*** array. *Output*

On exit: the value of the parameter u_j in $R(x)$, for $j = 1,2,...,m-1$. The u_j are a permuted subset of the elements of X. The remaining $n - m + 1$ locations contain a permutation of the remaining x_i, which can be ignored.

7: IW(N) – INTEGER array. *Workspace*

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N ≤ 0.

IFAIL = 2

At least one pair of the values X(i) are equal (or so nearly so that a subsequent division will inevitably cause overflow).

IFAIL = 3

A continued fraction of the required form does not exist.

7. Accuracy

Usually, it is not the accuracy of the coefficients produced by this routine which is of prime interest, but rather the accuracy of the value of $R(x)$ that is produced by the associated routine E01RBF when subsequently it evaluates the continued fraction (1) for a given value of x. This final accuracy will depend mainly on the nature of the interpolation being performed. If interpolation of a 'well-behaved smooth' function is attempted (and provided the data adequately represents the function), high accuracy will normally ensue, but, if the function is not so 'smooth' or extrapolation is being attempted, high accuracy is much less likely. Indeed, in extreme cases, results can be highly inaccurate.

There is no built-in test of accuracy but several courses are open to the user to prevent the production or the acceptance of inaccurate results.

(1) If the origin of a variable is well outside the range of its data values, the origin should be shifted to correct this; and, if the new data values are still excessively large or small, scaling to make the largest value of the order of unity is recommended. Thus, normalisation to the range −1.0 to +1.0 is ideal. This applies particularly to the independent variable; for the dependent variable, the removal of leading figures which are common to all the data values will usually suffice.

(2) To check the effect of rounding errors engendered in the routines themselves, E01RAF should be re-entered with x_1 interchanged with x_i and f_1 with f_i, ($i \neq 1$). This will produce a completely different vector a and a re-ordered vector u, but any change in the value of $R(x)$ subsequently produced by E01RBF will be due solely to rounding error.

(3) Even if the data consist of calculated values of a formal mathematical function, it is only in exceptional circumstances that bounds for the interpolation error (the difference between the true value of the function underlying the data and the value which would be produced by the two routines if exact arithmetic were used) can be derived that are sufficiently precise to be of practical use. Consequently, the user is recommended to rely on comparison checks: if extra data points are available, the calculation may be repeated with one or more data pairs added or exchanged, or alternatively, one of the original data pairs may be omitted. If the algorithms are being used for extrapolation, the calculations should be performed repeatedly with the 2,3,... nearest points until, hopefully, successive values of $R(x)$ for the given x agree to the required accuracy.

8. Further Comments

The time taken by the routine is approximately proportional to n^2.

The continued fraction (1) when expanded produces a rational function in x, the degree of whose numerator is either equal to or exceeds by unity that of the denominator. Only if this rather special form of interpolatory rational function is needed explicitly, would this routine be used without subsequent entry (or entries) to E01RBF.

9. Example

This example program reads in the abscissae and ordinates of 5 data points and prints the parameters a_j and u_j of a rational function which interpolates them.

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.

```
*     E01RAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=5)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, M
*     .. Local Arrays ..
      real             A(N), F(N), U(N), X(N)
      INTEGER          IW(N)
*     .. External Subroutines ..
      EXTERNAL         E01RAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01RAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) (X(I),I=1,N)
      READ (NIN,*) (F(I),I=1,N)
      IFAIL = 0
*
      CALL E01RAF(N,X,F,M,A,U,IW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The values of U(J) are'
      WRITE (NOUT,99999) (U(I),I=1,M-1)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The Thiele coefficients A(J) are'
      WRITE (NOUT,99999) (A(I),I=1,M)
      STOP
*
99999 FORMAT (1X,1P,4e12.4,/)
      END
```

9.2. Program Data

```
E01RAF Example Program Data
   0.0      1.0      2.0      3.0      4.0
   4.0      2.0      4.0      7.0     10.4
```

9.3. Program Results

```
 E01RAF Example Program Results

 The values of U(J) are
   0.0000E+00  3.0000E+00  1.0000E+00

 The Thiele coefficients A(J) are
   4.0000E+00  1.0000E+00  7.5000E-01 -1.0000E+00
```

E01RBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01RBF evaluates continued fractions of the form produced by E01RAF.

2. Specification

```
SUBROUTINE E01RBF (M, A, U, X, F, IFAIL)
INTEGER          M, IFAIL
real             A(M), U(M), X, F
```

3. Description

E01RBF evaluates the continued fraction

$$R(x) = a_1 + R_m(x)$$

where

$$R_i(x) = \frac{a_{m-i+2}(x-u_{m-i+1})}{1 + R_{i-1}(x)}, \qquad \text{for } i = m,m-1,...,2,$$

and

$$R_1(x) = 0$$

for a prescribed value of x. E01RBF is intended to be used to evaluate the continued fraction representation (of an interpolatory rational function) produced by E01RAF.

4. References

[1] GRAVES-MORRIS, P.R. and HOPKINS, T.R.
Reliable Rational Interpolation.
Numer. Math., 36, pp. 111-128, 1981.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of terms in the continued fraction.

Constraint: M $\geq$ 1.

2: A(M) – ***real*** array. *Input*

On entry: A(j) must be set to the value of the parameter a_j in the continued fraction, for $j = 1,2,...,m$.

3: U(M) – ***real*** array. *Input*

On entry: U(j) must be set to the value of the parameter u_j in the continued fraction, for $j = 1,2,...,m-1$. (The element U(m) is not used).

4: X – ***real***. *Input*

On entry: the value of x at which the continued fraction is to be evaluated.

5: F – ***real***. *Output*

On exit: the value of the continued fraction corresponding to the value of x.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The value of X corresponds to a pole of $R(x)$ or is so close that an overflow is likely to ensue.

7. Accuracy

See Section 7 of the routine document for E01RAF.

8. Further Comments

The time taken by the routine is approximately proportional to m.

9. Example

This example program reads in the parameters a_j and u_j of a continued fraction (as determined by the example for E01RAF) and evaluates the continued fraction at a point x.

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.

```
*     E01RBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          M
      PARAMETER        (M=4)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             F, X
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             A(M), U(M)
*     .. External Subroutines ..
      EXTERNAL         E01RBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01RBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) (A(I),I=1,M)
      READ (NIN,*) (U(I),I=1,M-1)
      READ (NIN,*) X
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'X =', X
      IFAIL = 0
*
      CALL E01RBF(M,A,U,X,F,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'The value of R(X) is ', F
      STOP
*
99999 FORMAT (1X,A,1P,e12.4)
      END
```

9.2. Program Data

```
E01RBF Example Program Data
   4.000    1.000    0.750   -1.000
   0.000    3.000    1.000
   6.000
```

9.3. Program Results

```
 E01RBF Example Program Results

 X =  6.0000E+00

 The value of R(X) is   1.7714E+01
```

E01SAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01SAF generates a two-dimensional surface interpolating a set of scattered data points, using the method of Renka and Cline.

2. Specification

```
SUBROUTINE E01SAF (M, X, Y, F, TRIANG, GRADS, IFAIL)
INTEGER          M, TRIANG(7*M), IFAIL
real             X(M), Y(M), F(M), GRADS(2,M)
```

3. Description

This routine constructs an interpolating surface $F(x,y)$ through a set of m scattered data points (x_r,y_r,f_r), for $r = 1,2,...,m$, using a method due to Renka and Cline. In the (x,y) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

The method involves firstly creating a triangulation with all the (x,y) data points as nodes, the triangulation being as nearly equiangular as possible (see Cline and Renka [1]). Then gradients in the x– and y–directions are estimated at node r, for $r = 1,2,...,m$, as the partial derivatives of a quadratic function of x and y which interpolates the data value f_r, and which fits the data values at nearby nodes (those within a certain distance chosen by the algorithm) in a weighted least-squares sense. The weights are chosen such that closer nodes have more influence than more distant nodes on derivative estimates at node r. The computed partial derivatives, with the f_r values, at the three nodes of each triangle define a piecewise polynomial surface of a certain form which is the interpolant on that triangle. See Renka and Cline [4] for more detailed information on the algorithm, a development of that by Lawson [2]. The code is derived from Renka [3].

The interpolant $F(x,y)$ can subsequently be evaluated at any point (x,y) inside or outside the domain of the data by a call to E01SBF. Points outside the domain are evaluated by extrapolation.

4. References

[1] CLINE, A.K. and RENKA, R.L.
A Storage-efficient Method for Construction of a Thiessen Triangulation.
Rocky Mountain J. Math., 14, pp. 119-139, 1984.

[2] LAWSON, C.L.
Software for C^1 Surface Interpolation.
In, 'Mathematical Software III', Rice, J.R. (ed).
Academic Press, New York, pp. 161-194, 1977.

[3] RENKA, R.L.
Algorithm 624: Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane.
ACM Trans. Math. Software, 10, pp. 440-442, 1984.

[4] RENKA, R.L. and CLINE, A.K.
A Triangle-based C^1 Interpolation Method.
Rocky Mountain J. Math., 14, pp. 223-237, 1984.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of data points.

Constraint: M $\geq$ 3.

2: X(M) – ***real*** array. *Input*

3: Y(M) – ***real*** array. *Input*

4: F(M) – ***real*** array. *Input*

On entry: the co-ordinates of the rth data point, for $r = 1,2,...,m$. The data points are accepted in any order, but see Section 8.

Constraint: The (x,y) nodes must not all be collinear, and each node must be unique.

5: TRIANG(7*M) – INTEGER array. *Output*

On exit: a data structure defining the computed triangulation, in a form suitable for passing to E01SBF.

6: GRADS(2,M) – ***real*** array. *Output*

On exit: the estimated partial derivatives at the nodes, in a form suitable for passing to E01SBF. The derivatives at node r with respect to x and y are contained in GRADS(1,r) and GRADS(2,r) respectively, for $r = 1,2,...,m$.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, M < 3.

IFAIL = 2

On entry, all the (X,Y) pairs are collinear.

IFAIL = 3

On entry, (X(i),Y(i)) = (X(j),Y(j)) for some $i \neq j$.

7. Accuracy

On successful exit, the computational errors should be negligible in most situations but the user should always check the computed surface for acceptability, by drawing contours for instance. The surface always interpolates the input data exactly.

8. Further Comments

The time taken for a call of E01SAF is approximately proportional to the number of data points, m. The routine is more efficient if, before entry, the values in X, Y, F are arranged so that the X array is in ascending order.

9. Example

This program reads in a set of 30 data points and calls E01SAF to construct an interpolating surface. It then calls E01SBF to evaluate the interpolant at a sample of points on a rectangular grid.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger, and the interpolant would need to be evaluated on a finer grid to obtain an accurate plot, say.

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.

```
*     E01SAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=100,NMAX=25)
*     .. Local Scalars ..
      real             XHI, XLO, YHI, YLO
      INTEGER          I, IFAIL, J, M, NX, NY
*     .. Local Arrays ..
      real             F(MMAX), GRADS(2,MMAX), PF(NMAX), PX(NMAX),
     +                 PY(NMAX), X(MMAX), Y(MMAX)
      INTEGER          TRIANG(7*MMAX)
*     .. External Subroutines ..
      EXTERNAL         E01SAF, E01SBF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01SAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of nodes.
      READ (NIN,*) M
      IF (M.GE.1 .AND. M.LE.MMAX) THEN
*        Input the nodes (X,Y) and heights, F.
         DO 20 I = 1, M
            READ (NIN,*) X(I), Y(I), F(I)
   20    CONTINUE
*        Generate the triangulation and gradients.
         IFAIL = 0
*
         CALL E01SAF(M,X,Y,F,TRIANG,GRADS,IFAIL)
*
*        Evaluate the interpolant on a rectangular grid at NX*NY points
*        over the domain (XLO to XHI) x (YLO to YHI).
         READ (NIN,*) NX, XLO, XHI
         READ (NIN,*) NY, YLO, YHI
         IF (NX.LE.NMAX .AND. NY.LE.NMAX) THEN
            DO 40 I = 1, NX
               PX(I) = (real(NX-I)/(NX-1))*XLO + (real(I-1)/(NX-1))*XHI
   40       CONTINUE
            DO 60 I = 1, NY
               PY(I) = (real(NY-I)/(NY-1))*YLO + (real(I-1)/(NY-1))*YHI
   60       CONTINUE
            WRITE (NOUT,*)
            WRITE (NOUT,99999) '          X', (PX(I),I=1,NX)
            WRITE (NOUT,*) '     Y'
            DO 100 I = NY, 1, -1
               DO 80 J = 1, NX
                  IFAIL = 0
*
```

```
                  CALL E01SBF(M,X,Y,F,TRIANG,GRADS,PX(J),PY(I),PF(J),
     +                        IFAIL)
*
   80           CONTINUE
                WRITE (NOUT,99998) PY(I), (PF(J),J=1,NX)
  100        CONTINUE
          END IF
        END IF
        STOP
*
99999 FORMAT (1X,A,7F8.2)
99998 FORMAT (1X,F8.2,3X,7F8.2)
      END
```

9.2. Program Data

```
E01SAF Example Program Data
     30                          M, the number of data points
   11.16    1.24   22.15         X, Y, F data point definition
   12.85    3.06   22.11
   19.85   10.72    7.97
   19.72    1.39   16.83
   15.91    7.74   15.30
    0.00   20.00   34.60
   20.87   20.00    5.74
    3.45   12.78   41.24
   14.26   17.87   10.74
   17.43    3.46   18.60
   22.80   12.39    5.47
    7.58    1.98   29.87
   25.00   11.87    4.40
    0.00    0.00   58.20
    9.66   20.00    4.73
    5.22   14.66   40.36
   17.25   19.57    6.43
   25.00    3.87    8.74
   12.13   10.79   13.71
   22.23    6.21   10.25
   11.52    8.53   15.74
   15.20    0.00   21.60
    7.54   10.69   19.31
   17.32   13.78   12.11
    2.14   15.03   53.10
    0.51    8.37   49.43
   22.69   19.63    3.25
    5.47   17.13   28.63
   21.67   14.36    5.52
    3.31    0.33   44.08         End of the data points
 7   3.0   21.0                  Grid definition, X axis
 6   2.0   17.0                  Grid definition, Y axis
```

9.3. Program Results

```
E01SAF Example Program Results

        X    3.00    6.00    9.00   12.00   15.00   18.00   21.00
    Y
17.00        41.25   27.62   18.03   12.29   11.68    9.09    5.37
14.00        47.61   36.66   22.87   14.02   13.44   11.20    6.46
11.00        38.55   25.25   16.72   13.83   13.08   10.71    6.88
 8.00        37.90   23.97   16.79   16.43   15.46   13.02    9.30
 5.00        40.49   29.26   22.51   20.72   19.30   16.72   12.87
 2.00        43.52   33.91   26.59   22.23   21.15   18.67   14.88
```

E01SBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01SBF evaluates at a given point the two-dimensional interpolant function computed by E01SAF.

2. Specification

```
SUBROUTINE E01SBF (M, X, Y, F, TRIANG, GRADS, PX, PY, PF, IFAIL)
INTEGER          M, TRIANG(7*M), IFAIL
real             X(M), Y(M), F(M), GRADS(2,M), PX, PY, PF
```

3. Description

This routine takes as input the parameters defining the interpolant $F(x,y)$ of a set of scattered data points (x_r,y_r,f_r), for $r = 1,2,...,m$, as computed by E01SAF, and evaluates the interpolant at the point (px,py).

If (px,py) is equal to (x_r,y_r) for some value of r, the returned value will be equal to f_r.

If (px,py) is not equal to (x_r,y_r) for any r, the derivatives in GRADS will be used to compute the interpolant. A triangle is sought which contains the point (px,py), and the vertices of the triangle along with the partial derivatives and f_r values at the vertices are used to compute the value $F(px,py)$. If the point (px,py) lies outside the triangulation defined by the input parameters, the returned value is obtained by extrapolation. In this case, the interpolating function F is extended linearly beyond the triangulation boundary. The method is described in more detail in Renka and Cline [2] and the code is derived from Renka [1].

E01SBF must only be called after a call to E01SAF.

4. References

[1] RENKA, R.L.
Algorithm 624: Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane.
ACM Trans. Math. Software, 10, pp. 440-442, 1984.

[2] RENKA, R.L. and CLINE, A.K.
A Triangle-based C^1 Interpolation Method.
Rocky Mountain J. Math., 14, pp. 223-237, 1984.

5. Parameters

1: M – INTEGER. *Input*
2: X(M) – ***real*** array. *Input*
3: Y(M) – ***real*** array. *Input*
4: F(M) – ***real*** array. *Input*
5: TRIANG(7*M) – INTEGER array. *Input*
6: GRADS(2,M) – ***real*** array. *Input*

On entry: M, X, Y, F, TRIANG and GRADS must be unchanged from the previous call of E01SAF.

7: PX – ***real***. *Input*
8: PY – ***real***. *Input*

On entry: the point (px,py) at which the interpolant is to be evaluated.

9: PF – ***real.*** *Output*

On exit: the value of the interpolant evaluated at the point (*px*,*py*).

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, M < 3.

IFAIL = 2

On entry the triangulation information held in the array TRIANG does not specify a valid triangulation of the data points. TRIANG may have been corrupted since the call to E01SAF.

IFAIL = 3

The evaluation point (PX,PY) lies outside the nodal triangulation, and the value returned in PF is computed by extrapolation.

7. Accuracy

Computational errors should be negligible in most practical situations.

8. Further Comments

The time taken for a call of E01SBF is approximately proportional to the number of data points, *m*.

The results returned by this routine are particularly suitable for applications such as graph plotting, producing a smooth surface from a number of scattered points.

9. Example

See the example for E01SAF.

E01SEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01SEF generates a two-dimensional surface interpolating a set of scattered data points, using a modified Shepard method.

2. Specification

```
SUBROUTINE E01SEF (M, X, Y, F, RNW, RNQ, NW, NQ, FNODES, MINNQ, WRK,
1                  IFAIL)
INTEGER          M, NW, NQ, MINNQ, IFAIL
real             X(M), Y(M), F(M), RNW, RNQ, FNODES(5*M), WRK(6*M)
```

3. Description

This routine constructs an interpolating surface $F(x,y)$ through a set of m scattered data points (x_r,y_r,f_r), for $r = 1,2,...,m$, using a modification of Shepard's method. The surface is continuous and has continuous first derivatives.

The basic Shepard method, described in [2], interpolates the input data with the weighted mean

$$F(x,y) = \frac{\sum_{r=1}^{m} w_r(x,y) f_r}{\sum_{r=1}^{m} w_r(x,y)}, \text{ where } w_r(x,y) = \frac{1}{d_r^2} \text{ and } d_r^2 = (x-x_r)^2 + (y-y_r)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this routine uses a modification due to Franke and Nielson described in [1], whereby the method becomes local by adjusting each $w_r(x,y)$ to be zero outside a circle with centre (x_r,y_r) and some radius R_w. Also, to improve the performance of the basic method, each f_r above is replaced by a function $f_r(x,y)$, which is a quadratic fitted by weighted least-squares to data local to (x_r,y_r) and forced to interpolate (x_r,y_r,f_r). In this context, a point (x,y) is defined to be local to another point if it lies within some distance R_q of it. Computation of these quadratics constitutes the main work done by this routine. If there are less than 5 other points within distance R_q from (x_r,y_r), the quadratic is replaced by a linear function. In cases of rank-deficiency, the minimum norm solution is computed.

The user may specify values for R_w and R_q, but it is usually easier to choose instead two integers N_w and N_q, from which the routine will compute R_w and R_q. These integers can be thought of as the average numbers of data points lying within distances R_w and R_q respectively from each node. Default values are provided, and advice on alternatives is given in Section 8.2.

The interpolant $F(x,y)$ generated by this routine can subsequently be evaluated for any point (x,y) in the domain of the data by a call to E01SFF.

4. References

[1] FRANKE, R. and NIELSON, G.
Smooth Interpolation of Large Sets of Scattered Data.
Internat. J. Num. Methods Engrg., 15, pp. 1691-1704, 1980.

[2] SHEPARD, D.
A Two-dimensional Interpolation Function for Irregularly Spaced Data.
Proc. 23rd Nat. Conf. ACM, Brandon/Systems Press Inc., Princeton, pp. 517-523, 1968.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of data points.

Constraint: M $\geq$ 3.

2: X(M) – ***real*** array. *Input*

3: Y(M) – ***real*** array. *Input*

4: F(M) – ***real*** array. *Input*

On entry: the co-ordinates of the rth data point, for r = 1,2,...,m. The order of the data points is immaterial.

Constraint: each of the (X(r),Y(r)) pairs must be unique.

5: RNW – ***real***. *Input/Output*

6: RNQ – ***real***. *Input/Output*

On entry: suitable values for the radii R_w and R_q, described in Section 3.

Constraint: RNQ $\leq$ 0 or 0 < RNW $\leq$ RNQ.

On exit: if RNQ is set less than or equal to zero on entry, then default values for both of them will be computed from the parameters NW and NQ, and RNW and RNQ will contain these values on exit.

7: NW – INTEGER. *Input*

8: NQ – INTEGER. *Input*

On entry: if RNQ > 0.0 and RNW > 0.0, then NW and NQ are not referenced by the routine. Otherwise, NW and NQ must specify suitable values for the integers N_w and N_q described in Section 3.

If NQ is less than or equal to zero on entry, then default values for both of them, namely NW = 9 and NQ = 18, will be used.

Constraint: NQ $\leq$ 0 or 0 < NW $\leq$ NQ.

9: FNODES(5*M) – ***real*** array. *Output*

On exit: the coefficients of the constructed quadratic nodal functions. These are in a form suitable for passing to E01SFF.

10: MINNQ – INTEGER. *Output*

On exit: the minimum number of data points that lie within radius RNQ of any node, and thus define a nodal function. If MINNQ is very small (say, less than 5), then the interpolant may be unsatisfactory in regions where the data points are sparse.

11: WRK(6*M) – ***real*** array. *Workspace*

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, M < 3.

IFAIL = 2

On entry, RNQ > 0 and either RNW > RNQ or RNW ≤ 0.

IFAIL = 3

On entry, NQ > 0 and either NW > NQ or NW ≤ 0.

IFAIL = 4

On entry, (X(i),Y(i)) is equal to (X(j),Y(j)) for some $i \neq j$.

7. Accuracy

On successful exit, the computational errors should be negligible in most situations but the user should always check the computed surface for acceptability, by drawing contours for instance. The surface always interpolates the input data exactly.

8. Further Comments

8.1. Timing

The time taken for a call of E01SEF is approximately proportional to the number of data points, m, provided that N_q is of the same order as its default value (18). However if N_q is increased so that the method becomes more global, the time taken becomes approximately proportional to m^2.

8.2. Choice of N_w and N_q

Note first that the radii R_w and R_q, described in Section 3, are computed as $\frac{D}{2}\sqrt{\frac{N_w}{m}}$ and $\frac{D}{2}\sqrt{\frac{N_q}{m}}$ respectively, where D is the maximum distance between any pair of data points.

Default values $N_w = 9$ and $N_q = 18$ work quite well when the data points are fairly uniformly distributed. However, for data having some regions with relatively few points or for small data sets ($m < 25$), a larger value of N_w may be needed. This is to ensure a reasonable number of data points within a distance R_w of each node, and to avoid some regions in the data area being left outside all the discs of radius R_w on which the weights $w_r(x,y)$ are non-zero. Maintaining N_q approximately equal to $2N_w$ is usually an advantage.

Note however that increasing N_w and N_q does not improve the quality of the interpolant in all cases. It does increase the computational cost and makes the method less local.

9. Example

This program reads in a set of 30 data points and calls E01SEF to construct an interpolating surface. It then calls E01SFF to evaluate the interpolant at a sample of points on a rectangular grid.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger, and the interpolant would need to be evaluated on a finer grid to obtain an accurate plot, say.

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.

```
*     E01SEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=100,NMAX=25)
*     .. Local Scalars ..
      real             RNQ, RNW, XHI, XLO, YHI, YLO
      INTEGER          I, IFAIL, J, M, MINNQ, NQ, NW, NX, NY
*     .. Local Arrays ..
      real             F(MMAX), FNODES(5*MMAX), PF(NMAX), PX(NMAX),
     +                 PY(NMAX), WRK(6*MMAX), X(MMAX), Y(MMAX)
*     .. External Subroutines ..
      EXTERNAL         E01SEF, E01SFF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01SEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of nodes.
      READ (NIN,*) M
      IF (M.GE.1 .AND. M.LE.MMAX) THEN
*        Input the nodes (X,Y) and heights, F.
         DO 20 I = 1, M
            READ (NIN,*) X(I), Y(I), F(I)
   20    CONTINUE
*        Compute the nodal function coefficients.
         RNQ = 0.0e0
         NQ = 0
         IFAIL = 0
*
         CALL E01SEF(M,X,Y,F,RNW,RNQ,NW,NQ,FNODES,MINNQ,WRK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) '     RNW =', RNW, '  RNQ =', RNQ,
     +     '  MINNQ =', MINNQ
         WRITE (NOUT,*)
*        Evaluate the interpolant on a rectangular grid at NX*NY points
*        over the domain (XLO to XHI) x (YLO to YHI).
         READ (NIN,*) NX, XLO, XHI
         READ (NIN,*) NY, YLO, YHI
         IF (NX.LE.NMAX .AND. NY.LE.NMAX) THEN
            DO 40 I = 1, NX
               PX(I) = (real(NX-I)/(NX-1))*XLO + (real(I-1)/(NX-1))*XHI
   40       CONTINUE
            DO 60 I = 1, NY
               PY(I) = (real(NY-I)/(NY-1))*YLO + (real(I-1)/(NY-1))*YHI
   60       CONTINUE
            WRITE (NOUT,99999) '           X', (PX(I),I=1,NX)
            WRITE (NOUT,*) '     Y'
            DO 100 I = NY, 1, -1
               DO 80 J = 1, NX
                  IFAIL = 0
*
                  CALL E01SFF(M,X,Y,F,RNW,FNODES,PX(J),PY(I),PF(J),
     +                        IFAIL)
*
   80          CONTINUE
               WRITE (NOUT,99998) PY(I), (PF(J),J=1,NX)
```

```
  100       CONTINUE
          END IF
        END IF
        STOP
*
99999 FORMAT (1X,A,7F8.2)
99998 FORMAT (1X,F8.2,3X,7F8.2)
99997 FORMAT (1X,A,F8.2,A,F8.2,A,I3)
        END
```

9.2. Program Data

```
E01SEF Example Program Data
     30                          M, the number of data points
     11.16    1.24   22.15       X, Y, F data point definition
     12.85    3.06   22.11
     19.85   10.72    7.97
     19.72    1.39   16.83
     15.91    7.74   15.30
      0.00   20.00   34.60
     20.87   20.00    5.74
      3.45   12.78   41.24
     14.26   17.87   10.74
     17.43    3.46   18.60
     22.80   12.39    5.47
      7.58    1.98   29.87
     25.00   11.87    4.40
      0.00    0.00   58.20
      9.66   20.00    4.73
      5.22   14.66   40.36
     17.25   19.57    6.43
     25.00    3.87    8.74
     12.13   10.79   13.71
     22.23    6.21   10.25
     11.52    8.53   15.74
     15.20    0.00   21.60
      7.54   10.69   19.31
     17.32   13.78   12.11
      2.14   15.03   53.10
      0.51    8.37   49.43
     22.69   19.63    3.25
      5.47   17.13   28.63
     21.67   14.36    5.52
      3.31    0.33   44.08       End of data points
   7    3.0   21.0               Grid definition, X axis
   6    2.0   17.0               Grid definition, Y axis
```

9.3. Program Results

```
 E01SEF Example Program Results

     RNW =    8.22  RNQ =   11.62  MINNQ =  4

          X    3.00    6.00    9.00   12.00   15.00   18.00   21.00
      Y
    17.00      39.27   27.87   21.90   14.43   12.06    9.48    5.26
    14.00      47.21   37.79   25.17   14.43   13.29   11.34    6.23
    11.00      37.41   24.65   16.31   13.76   12.75   10.40    6.98
     8.00      35.43   20.22   18.25   15.72   15.56   13.02    9.63
     5.00      40.83   32.47   25.15   21.36   18.98   16.81   12.60
     2.00      44.56   34.46   26.48   22.30   21.09   18.70   15.07
```

E01SFF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E01SFF evaluates at a given point the two-dimensional interpolating function computed by E01SEF.

2. Specification

```
SUBROUTINE E01SFF (M, X, Y, F, RNW, FNODES, PX, PY, PF, IFAIL)
INTEGER          . M, IFAIL
real             X(M), Y(M), F(M), RNW, FNODES(5*M), PX, PY, PF
```

3. Description

This routine takes as input the interpolant $F(x,y)$ of a set of scattered data points (x_r,y_r,f_r), for $r = 1,2,...,m$, as computed by E01SEF, and evaluates the interpolant at the point (px,py).

If (px,py) is equal to (x_r,y_r) for some value of r, the returned value will be equal to f_r.

If (px,py) is not equal to (x_r,y_r) for any r, all points that are within distance RNW of (px,py), along with the corresponding nodal functions given by FNODES, will be used to compute a value of the interpolant.

E01SFF must only be called after a call to E01SEF.

4. References

[1] FRANKE, R. and NIELSON, G.
Smooth Interpolation of Large Sets of Scattered Data.
Internat. J. Numer. Methods Engrg., 15, pp. 1691-1704, 1980.

[2] SHEPARD, D.
A Two-dimensional Interpolation Function for Irregularly Spaced Data.
Proc. 23rd Nat. Conf. ACM, Brandon/Systems Press Inc., Princeton, pp. 517-523, 1968.

5. Parameters

1: M – INTEGER. *Input*

2: X(M) – ***real*** array. *Input*

3: Y(M) – ***real*** array. *Input*

4: F(M) – ***real*** array. *Input*

5: RNW – ***real***. *Input*

6: FNODES(5*M) – ***real*** array. *Input*

On entry: M, X, Y, F, RNW and FNODES must be unchanged from the previous call of E01SEF.

7: PX – ***real***. *Input*

8: PY – ***real***. *Input*

On entry: the point (px,py) at which the interpolant is to be evaluated.

9: PF – ***real***. *Output*

On exit: the value of the interpolant evaluated at the point (px,py).

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, M < 3.

IFAIL = 2

The interpolant cannot be evaluated because the evaluation point (PX,PY) lies outside the support region of the data supplied in X, Y and F. This error exit will occur if (PX,PY) lies at a distance greater than or equal to RNW from every point given by arrays X and Y.

The value 0.0 is returned in PF. This value will not provide continuity with values obtained at other points (PX,PY), i.e. values obtained when IFAIL = 0 on exit.

7. Accuracy

Computational errors should be negligible in most practical situations.

8. Further Comments

The time taken for a call of E01SFF is approximately proportional to the number of data points, *m*.

The results returned by this routine are particularly suitable for applications such as graph plotting, producing a smooth surface from a number of scattered points.

9. Example

See the example for E01SEF.

E01SGF – NAG Fortran Library Routine Document

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

E01SGF generates a two-dimensional interpolant to a set of scattered data points, using a modified Shepard method.

2 Specification

```
SUBROUTINE E01SGF(M, X, Y, F, NW, NQ, IQ, LIQ, RQ, LRQ, IFAIL)
real              X(M), Y(M), F(M), RQ(LRQ)
INTEGER           M, NW, NQ, IQ(LIQ), LIQ, LRQ, IFAIL
```

3 Description

This routine constructs a smooth function $Q(x,y)$ which interpolates a set of m scattered data points (x_r, y_r, f_r), for $r = 1, 2, \ldots, m$, using a modification of Shepard's method. The surface is continuous and has continuous first partial derivatives.

The basic Shepard method, described in [5], interpolates the input data with the weighted mean

$$Q(x,y) = \frac{\sum_{r=1}^{m} w_r(x,y) q_r}{\sum_{r=1}^{m} w_r(x,y)},$$

where

$$q_r = f_r \text{ and } w_r(x,y) = \frac{1}{d_r^2} \text{ and } d_r^2 = (x - x_r)^2 + (y - y_r)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this routine uses a modification (see [2], [3]), whereby the method becomes local by adjusting each $w_r(x,y)$ to be zero outside a circle with centre (x_r, y_r) and some radius R_w. Also, to improve the performance of the basic method, each q_r above is replaced by a function $q_r(x,y)$, which is a quadratic fitted by weighted least-squares to data local to (x_r, y_r) and forced to interpolate (x_r, y_r, f_r). In this context, a point (x,y) is defined to be local to another point if it lies within some distance R_q of it. Computation of these quadratics constitutes the main work done by this routine.

The efficiency of the routine is further enhanced by using a cell method for nearest neighbour searching due to Bentley and Friedman [1].

The radii R_w and R_q are chosen to be just large enough to include N_w and N_q data points, respectively, for user-supplied constants N_w and N_q. Default values of these parameters are provided by the routine, and advice on alternatives is given in Section 8.2.

This routine is derived from the routine QSHEP2 described by Renka [4].

Values of the interpolant $Q(x,y)$ generated by this routine, and its first partial derivatives, can subsequently be evaluated for points in the domain of the data by a call to E01SHF.

4 References

[1] Bentley J L and Friedman J H (1979) Data structures for range searching *ACM Comput. Surv.* **11** 397–409

[2] Franke R and Nielson G (1980) Smooth interpolation of large sets of scattered data *Internat. J. Num. Methods Engrg.* **15** 1691–1704

[3] Renka R J (1988) Multivariate interpolation of large sets of scattered data *ACM Trans. Math. Software* **14** 139–148

[4] Renka R J (1988) Algorithm 660: QSHEP2D: Quadratic Shepard method for bivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 149–150

[5] Shepard D (1968) A two-dimensional interpolation function for irregularly spaced data *Proc. 23rd Nat. Conf. ACM* Brandon/Systems Press Inc., Princeton 517–523

5 Parameters

1: M — INTEGER *Input*

On entry: m, the number of data points.

Constraint: M ≥ 6.

2: X(M) — ***real*** array *Input*

3: Y(M) — ***real*** array *Input*

On entry: the Cartesian coordinates of the data points (x_r, y_r), for $r = 1, 2, \ldots, m$.

Constraint: these coordinates must be distinct, and must not all be collinear.

4: F(M) — ***real*** array *Input*

On entry: the data values f_r, for $r = 1, 2, \ldots, m$.

5: NW — INTEGER *Input*

On entry: the number N_w of data points that determines each radius of influence R_w, appearing in the definition of each of the weights w_r, for $r = 1, 2, \ldots, m$ (see Section 3). Note that R_w is different for each weight. If NW ≤ 0 the default value NW = min(19,M−1) is used instead.

Constraint: NW $\leq$ min(40,M−1).

6: NQ — INTEGER *Input*

On entry: the number N_q of data points to be used in the least-squares fit for coefficients defining the nodal functions $q_r(x, y)$ (see Section 3). If NQ ≤ 0 the default value NQ = min(13,M−1) is used instead.

Constraint: NQ ≤ 0 or $5 \leq$ NQ $\leq$ min(40,M−1).

7: IQ(LIQ) — INTEGER array *Output*

On exit: integer data defining the interpolant $Q(x, y)$.

8: LIQ — INTEGER *Input*

On entry: the dimension of the array IQ as declared in the (sub)program from which E01SGF is called.

Constraint: LIQ $\geq 2 \times$ M + 1.

9: RQ(LRQ) — ***real*** array *Output*

On exit: real data defining the interpolant $Q(x, y)$.

10: LRQ — INTEGER *Input*

On entry: the dimension of the array RQ as declared in the (sub)program from which E01SGF is called.

Constraint: LRQ $\geq 6 \times$ M + 5.

11: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, M < 6,
or 0 < NQ < 5,
or NQ > min(40,M−1),
or NW > min(40,M−1),
or LIQ < 2 × M + 1,
or LRQ < 6 × M + 5.

IFAIL = 2

On entry, (X(i),Y(i)) = (X(j),Y(j)) for some $i \neq j$.

IFAIL = 3

On entry, all the data points are collinear. No unique solution exists.

7 Accuracy

On successful exit, the function generated interpolates the input data exactly and has quadratic accuracy.

8 Further Comments

8.1 Timing

The time taken for a call to E01SGF will depend in general on the distribution of the data points. If X and Y are uniformly randomly distributed, then the time taken should be O(M). At worst O(M^2) time will be required.

8.2 Choice of N_w and N_q

Default values of the parameters N_w and N_q may be selected by calling E01SGF with NW $\leq$ 0 and NQ $\leq$ 0. These default values may well be satisfactory for many applications.

If non-default values are required they must be supplied to E01SGF through positive values of NW and NQ. Increasing these parameters makes the method less local. This may increase the accuracy of the resulting interpolant at the expense of increased computational cost. The default values NW = min(19,M−1) and NQ = min(13,M−1) have been chosen on the basis of experimental results reported in [3]. In these experiments the error norm was found to vary smoothly with N_w and N_q, generally increasing monotonically and slowly with distance from the optimal pair. The method is not therefore thought to be particularly sensitive to the parameter values. For further advice on the choice of these parameters see [3].

9 Example

This program reads in a set of 30 data points and calls E01SGF to construct an interpolating function $Q(x, y)$. It then calls E01SHF to evaluate the interpolant at a set of points.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger.

9.1 Program Text

```
*     E01SGF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX, LIQ, LRQ
      PARAMETER        (MMAX=100,NMAX=100,LIQ=2*MMAX+1,LRQ=6*MMAX+5)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, M, N, NQ, NW
*     .. Local Arrays ..
      real             F(MMAX), Q(NMAX), QX(NMAX), QY(NMAX), RQ(LRQ),
     +                 U(NMAX), V(NMAX), X(MMAX), Y(MMAX)
      INTEGER          IQ(LIQ)
*     .. External Subroutines ..
      EXTERNAL         E01SGF, E01SHF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01SGF Example Program Results'
      WRITE (NOUT,*)
*     Skip heading in data file
      READ (NIN,*)
*
*     Input the number of nodes.
*
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
*
*        Input the data points X,Y and F.
*
         DO 20 I = 1, M
            READ (NIN,*) X(I), Y(I), F(I)
   20    CONTINUE
*
*        Generate the interpolant.
*
         NQ = 0
         NW = 0
         IFAIL = 0
         CALL E01SGF(M,X,Y,F,NW,NQ,IQ,LIQ,RQ,LRQ,IFAIL)
*
*        Input the number of evaluation points.
*
         READ (NIN,*) N
*
*        Input the evaluation points.
*
         DO 40 I = 1, N
            READ (NIN,*) U(I), V(I)
   40    CONTINUE
*
*        Evaluate the interpolant using E01SHF.
*
         IFAIL = -1
         CALL E01SHF(M,X,Y,F,IQ,LIQ,RQ,LRQ,N,U,V,Q,QX,QY,IFAIL)
*
         WRITE (NOUT,*) '     I      U(I)      V(I)      Q(I)'
         DO 60 I = 1, N
```

```
            WRITE (NOUT,99999) I, U(I), V(I), Q(I)
   60    CONTINUE
*
      END IF
      STOP
*
99999 FORMAT (1X,I6,3F10.2)
      END
```

9.2 Program Data

```
E01SGF Example Program Data
 30                        M, the number of data points
 11.16    1.24   22.15     X, Y, F data point definition
 12.85    3.06   22.11
 19.85   10.72    7.97
 19.72    1.39   16.83
 15.91    7.74   15.30
  0.00   20.00   34.60
 20.87   20.00    5.74
  3.45   12.78   41.24
 14.26   17.87   10.74
 17.43    3.46   18.60
 22.80   12.39    5.47
  7.58    1.98   29.87
 25.00   11.87    4.40
  0.00    0.00   58.20
  9.66   20.00    4.73
  5.22   14.66   40.36
 17.25   19.57    6.43
 25.00    3.87    8.74
 12.13   10.79   13.71
 22.23    6.21   10.25
 11.52    8.53   15.74
 15.20    0.00   21.60
  7.54   10.69   19.31
 17.32   13.78   12.11
  2.14   15.03   53.10
  0.51    8.37   49.43
 22.69   19.63    3.25
  5.47   17.13   28.63
 21.67   14.36    5.52
  3.31    0.33   44.08     End of data points
 5                         N, the number of evaluation points
 20.00    3.14            U, V evaluation point definition
  6.41   15.44
  7.54   10.69
  9.91   18.27
 12.30    9.22            End of evaluation points
```

9.3 Program Results

```
E01SGF Example Program Results

      I       U(I)       V(I)       Q(I)
      1      20.00       3.14      15.89
      2       6.41      15.44      34.05
      3       7.54      10.69      19.31
      4       9.91      18.27      13.68
      5      12.30       9.22      14.56
```

E01SHF – NAG Fortran Library Routine Document

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

E01SHF evaluates the two-dimensional interpolating function generated by E01SGF and its first partial derivatives.

2 Specification

```
      SUBROUTINE E01SHF(M, X, Y, F, IQ, LIQ, RQ, LRQ, N, U, V, Q, QX,
     1                  QY, IFAIL)
      real              X(M), Y(M), F(M), RQ(LRQ), U(N), V(N), Q(N),
     1                  QX(N), QY(N)
      INTEGER           M, IQ(LIQ), LIQ, LRQ, N, IFAIL
```

3 Description

This routine takes as input the interpolant $Q(x, y)$ of a set of scattered data points (x_r, y_r, f_r), for $r = 1, 2, \ldots, m$, as computed by E01SGF, and evaluates the interpolant and its first partial derivatives at the set of points (u_i, v_i), for $i = 1, 2, \ldots, n$.

E01SHF must only be called after a call to E01SGF.

This routine is derived from the routine QS2GRD described by Renka [1].

4 References

[1] Renka R J (1988) Algorithm 660: QSHEP2D: Quadratic Shepard method for bivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 149–150

5 Parameters

1: M — INTEGER *Input*

2: X(M) — ***real*** array *Input*

3: Y(M) — ***real*** array *Input*

4: F(M) — ***real*** array *Input*

On entry: M, X, Y and F must be the same values as were supplied in the preceeding call to E01SGF.

5: IQ(LIQ) — INTEGER array *Input*

On entry: IQ must be unchanged from the value returned from a previous call to E01SGF.

6: LIQ — INTEGER *Input*

On entry: the dimension of the array IQ as declared in the (sub)program from which E01SHF is called.

Constraint: LIQ $\geq 2 \times$ M $+ 1$.

7: RQ(LRQ) — ***real*** array *Input*

On entry: RQ must be unchanged from the value returned from a previous call to E01SGF.

8: LRQ — INTEGER *Input*

On entry: the dimension of the array RQ as declared in the (sub)program from which E01SHF is called.

Constraint: LRQ $\geq 6 \times$ M $+ 5$.

9: N — INTEGER *Input*

On entry: n, the number of evaluation points.

Constraint: N ≥ 1.

10: U(N) — ***real*** array *Input*

11: V(N) — ***real*** array *Input*

On entry: the evaluation points (u_i, v_i), for $i = 1, 2, \ldots, n$.

12: Q(N) — ***real*** array *Output*

On exit: the values of the interpolant at (u_i, v_i), for $i = 1, 2, \ldots, n$. If any of these evaluation points lie outside the region of definition of the interpolant the corresponding entries in Q are set to the largest machine representable number (see X02ALF), and E01SHF returns with IFAIL = 3.

13: QX(N) — ***real*** array *Output*

14: QY(N) — ***real*** array *Output*

On exit: the values of the partial derivatives of the interpolant $Q(x, y)$ at (u_i, v_i), for $i = 1, 2, \ldots, n$. If any of these evaluation points lie outside the region of definition of the interpolant, the corresponding entries in QX and QY are set to the largest machine representable number (see X02ALF), and E01SHF returns with IFAIL = 3.

15: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, M < 6,
or LIQ $< 2 \times$ M $+ 1$,
or LRQ $< 6 \times$ M $+ 5$,
or N < 1.

IFAIL = 2

Values supplied in IQ or RQ appear to be invalid. Check that these arrays have not been corrupted between the calls to E01SGF and E01SHF.

IFAIL = 3

At least one evaluation point lies outside the region of definition of the interpolant. At all such points the corresponding values in Q, QX and QY have been set to the largest machine representable number (see X02ALF).

7 Accuracy

Computational errors should be negligible in most practical situations.

8 Further Comments

The time taken for a call to E01SHF will depend in general on the distribution of the data points. If X and Y are approximately uniformly distributed, then the time taken should be only O(N). At worst O(MN) time will be required.

9 Example

See Section 9 of the document for E01SGF.

E01TGF – NAG Fortran Library Routine Document

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

E01TGF generates a three-dimensional interpolant to a set of scattered data points, using a modified Shepard method.

2 Specification

```
SUBROUTINE E01TGF(M, X, Y, Z, F, NW, NQ, IQ, LIQ, RQ, LRQ, IFAIL)
real              X(M), Y(M), Z(M), F(M), RQ(LRQ)
INTEGER           M, NW, NQ, IQ(LIQ), LIQ, LRQ, IFAIL
```

3 Description

This routine constructs a smooth function $Q(x, y, z)$ which interpolates a set of m scattered data points (x_r, y_r, z_r, f_r), for $r = 1, 2, \ldots, m$, using a modification of Shepard's method. The surface is continuous and has continuous first partial derivatives.

The basic Shepard method, which is a generalization of the two-dimensional method described in [5], interpolates the input data with the weighted mean

$$Q(x, y, z) = \frac{\sum_{r=1}^{m} w_r(x, y, z) q_r}{\sum_{r=1}^{m} w_r(x, y, z)},$$

where

$$q_r = f_r \text{ and } w_r(x, y, z) = \frac{1}{d_r^2} \text{ and } d_r^2 = (x - x_r)^2 + (y - y_r)^2 + (z - z_r)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this routine uses a modification (see [2], [3]), whereby the method becomes local by adjusting each $w_r(x, y, z)$ to be zero outside a sphere with centre (x_r, y_r, z_r) and some radius R_w. Also, to improve the performance of the basic method, each q_r above is replaced by a function $q_r(x, y, z)$, which is a quadratic fitted by weighted least-squares to data local to (x_r, y_r, z_r) and forced to interpolate (x_r, y_r, z_r, f_r). In this context, a point (x, y, z) is defined to be local to another point if it lies within some distance R_q of it. Computation of these quadratics constitutes the main work done by this routine.

The efficiency of the routine is further enhanced by using a cell method for nearest neighbour searching due to Bentley and Friedman [1].

The radii R_w and R_q are chosen to be just large enough to include N_w and N_q data points, respectively, for user-supplied constants N_w and N_q. Default values of these parameters are provided by the routine, and advice on alternatives is given in Section 8.2.

This routine is derived from the routine QSHEP3 described by Renka [4].

Values of the interpolant $Q(x, y, z)$ generated by this routine, and its first partial derivatives, can subsequently be evaluated for points in the domain of the data by a call to E01THF.

4 References

[1] Bentley J L and Friedman J H (1979) Data structures for range searching *ACM Comput. Surv.* **11** 397–409

[2] Franke R and Nielson G (1980) Smooth interpolation of large sets of scattered data *Internat. J. Num. Methods Engrg.* **15** 1691–1704

[3] Renka R J (1988) Multivariate interpolation of large sets of scattered data *ACM Trans. Math. Software* **14** 139–148

[4] Renka R J (1988) Algorithm 661: QSHEP3D: Quadratic Shepard method for trivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 151–152

[5] Shepard D (1968) A two-dimensional interpolation function for irregularly spaced data *Proc. 23rd Nat. Conf. ACM* Brandon/Systems Press Inc., Princeton 517–523

5 Parameters

1: M — INTEGER *Input*

On entry: m, the number of data points.

Constraint: M ≥ 10.

2: X(M) — ***real*** array *Input*

3: Y(M) — ***real*** array *Input*

4: Z(M) — ***real*** array *Input*

On entry: the Cartesian coordinates of the data points (x_r, y_r, z_r), for $r = 1, 2, \ldots, m$.

Constraint: these coordinates must be distinct, and must not all be coplanar.

5: F(M) — ***real*** array *Input*

On entry: the data values f_r, for $r = 1, 2, \ldots, m$.

6: NW — INTEGER *Input*

On entry: the number N_w of data points that determines each radius of influence R_w, appearing in the definition of each of the weights w_r, for $r = 1, 2, \ldots, m$ (see Section 3). Note that R_w is different for each weight. If NW ≤ 0 the default value NW = min(32,M−1) is used instead.

Constraint: NW $\leq$ min(40,M−1).

7: NQ — INTEGER *Input*

On entry: the number N_q of data points to be used in the least-squares fit for coefficients defining the nodal functions $q_r(x, y, z)$ (see Section 3). If NQ ≤ 0 the default value NQ = min(17,M−1) is used instead.

Constraint: NQ ≤ 0 or $9 \leq$ NQ $\leq$ min(40,M−1).

8: IQ(LIQ) — INTEGER array *Output*

On exit: integer data defining the interpolant $Q(x, y, z)$.

9: LIQ — INTEGER *Input*

On entry: the dimension of the array IQ as declared in the (sub)program from which E01TGF is called.

Constraint: LIQ $\geq 2 \times$ M $+ 1$.

10: RQ(LRQ) — ***real*** array *Output*

On exit: real data defining the interpolant $Q(x, y, z)$.

11: LRQ — INTEGER *Input*

On entry: the dimension of the array RQ as declared in the (sub)program from which E01TGF is called.

Constraint: LRQ $\geq 10 \times$ M $+ 7$.

12: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, M < 10,
or 0 < NQ < 9,
or NQ > min(40,M−1),
or NW > min(40,M−1),
or LIQ < 2 × M + 1,
or LRQ < 10 × M + 7.

IFAIL = 2

On entry, (X(i),Y(i),Z(i)) = (X(j),Y(j),Z(j)) for some $i \neq j$.

IFAIL = 3

On entry, all the data points are coplanar. No unique solution exists.

7 Accuracy

On successful exit, the function generated interpolates the input data exactly and has quadratic accuracy.

8 Further Comments

8.1 Timing

The time taken for a call to E01TGF will depend in general on the distribution of the data points. If X, Y and Z are uniformly randomly distributed, then the time taken should be O(M). At worst O(M^2) time will be required.

8.2 Choice of N_w and N_q

Default values of the parameters N_w and N_q may be selected by calling E01TGF with NW $\leq$ 0 and NQ $\leq$ 0. These default values may well be satisfactory for many applications.

If non-default values are required they must be supplied to E01TGF through positive values of NW and NQ. Increasing these parameters makes the method less local. This may increase the accuracy of the resulting interpolant at the expense of increased computational cost. The default values NW = min(32,M−1) and NQ = min(17,M−1) have been chosen on the basis of experimental results reported in [3]. In these experiments the error norm was found to vary smoothly with N_w and N_q, generally increasing monotonically and slowly with distance from the optimal pair. The method is not therefore thought to be particularly sensitive to the parameter values. For further advice on the choice of these parameters see [3].

9 Example

This program reads in a set of 30 data points and calls E01TGF to construct an interpolating function $Q(x, y, z)$. It then calls E01THF to evaluate the interpolant at a set of points.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger.

9.1 Program Text

```
*     E01TGF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX, LIQ, LRQ
      PARAMETER        (MMAX=100,NMAX=100,LIQ=2*MMAX+1,LRQ=10*MMAX+7)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, M, N, NQ, NW
*     .. Local Arrays ..
      real             F(MMAX), Q(NMAX), QX(NMAX), QY(NMAX), QZ(NMAX),
     +                 RQ(LRQ), U(NMAX), V(NMAX), W(NMAX), X(MMAX),
     +                 Y(MMAX), Z(MMAX)
      INTEGER          IQ(LIQ)
*     .. External Subroutines ..
      EXTERNAL         E01TGF, E01THF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E01TGF Example Program Results'
      WRITE (NOUT,*)
*     Skip heading in data file
      READ (NIN,*)
*
*     Input the number of nodes.
*
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
*
*        Input the data points X,Y,Z and F.
*
         DO 20 I = 1, M
            READ (NIN,*) X(I), Y(I), Z(I), F(I)
   20    CONTINUE
*
*        Generate the interpolant.
*
         NQ = 0
         NW = 0
         IFAIL = 0
         CALL E01TGF(M,X,Y,Z,F,NW,NQ,IQ,LIQ,RQ,LRQ,IFAIL)
*
*        Input the number of evaluation points.
*
         READ (NIN,*) N
*
*        Input the evaluation points.
*
         DO 40 I = 1, N
            READ (NIN,*) U(I), V(I), W(I)
   40    CONTINUE
```

```
*
*        Evaluate the interpolant using E01THF.
*
         IFAIL = -1
         CALL E01THF(M,X,Y,Z,F,IQ,LIQ,RQ,LRQ,N,U,V,W,Q,QX,QY,QZ,IFAIL)
*
         WRITE (NOUT,*) '     I     U(I)      V(I)      W(I)      Q(I)'
         DO 60 I = 1, N
            WRITE (NOUT,99999) I, U(I), V(I), W(I), Q(I)
   60    CONTINUE
*
      END IF
      STOP
*
99999 FORMAT (1X,I6,4F10.4)
      END
```

9.2 Program Data

```
E01TGF Example Program Data
 30                          M, the number of data points
 0.80  0.23  0.37  0.51      X, Y, Z, F data point definition
 0.23  0.88  0.05  1.80
 0.18  0.43  0.04  0.11
 0.58  0.95  0.62  2.65
 0.64  0.69  0.20  0.93
 0.88  0.35  0.49  0.72
 0.30  0.10  0.78 -0.11
 0.87  0.09  0.05  0.67
 0.04  0.02  0.40  0.00
 0.62  0.90  0.43  2.20
 0.87  0.96  0.24  3.17
 0.62  0.64  0.45  0.74
 0.86  0.13  0.47  0.64
 0.87  0.60  0.46  1.07
 0.49  0.43  0.13  0.22
 0.12  0.61  0.00  0.41
 0.02  0.71  0.82  0.58
 0.62  0.93  0.44  2.48
 0.49  0.54  0.04  0.37
 0.36  0.56  0.39  0.35
 0.62  0.42  0.97 -0.20
 0.01  0.72  0.45  0.78
 0.41  0.36  0.52  0.11
 0.17  0.99  0.65  2.82
 0.51  0.29  0.59  0.14
 0.85  0.05  0.04  0.61
 0.20  0.20  0.87 -0.25
 0.04  0.67  0.04  0.59
 0.31  0.63  0.18  0.50
 0.88  0.27  0.07  0.71      End of data points
 6                           N, the number of evaluation points
 0.10  0.10  0.10            U, V, W evaluation point definition
 0.20  0.20  0.20
 0.30  0.30  0.30
 0.40  0.40  0.40
 0.50  0.50  0.50
 0.60  0.60  0.60            End of evaluation points
```

9.3 Program Results

```
E01TGF Example Program Results

   I      U(I)       V(I)       W(I)       Q(I)
   1    0.1000     0.1000     0.1000     0.2630
   2    0.2000     0.2000     0.2000     0.1182
   3    0.3000     0.3000     0.3000     0.0811
   4    0.4000     0.4000     0.4000     0.1552
   5    0.5000     0.5000     0.5000     0.3019
   6    0.6000     0.6000     0.6000     0.5712
```

E01THF – NAG Fortran Library Routine Document

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

E01THF evaluates the three-dimensional interpolating function generated by E01TGF and its first partial derivatives.

2 Specification

```
      SUBROUTINE E01THF(M, X, Y, Z, F, IQ, LIQ, RQ, LRQ, N, U, V, W, Q,
     1                  QX, QY, QZ, IFAIL)
      real              X(M), Y(M), Z(M), F(M), RQ(LRQ), U(N), V(N),
     1                  W(N), Q(N), QX(N), QY(N), QZ(N)
      INTEGER           M, IQ(LIQ), LIQ, LRQ, N, IFAIL
```

3 Description

This routine takes as input the interpolant $Q(x, y, z)$ of a set of scattered data points (x_r, y_r, z_r, f_r), for $r = 1, 2, \ldots, m$, as computed by E01TGF, and evaluates the interpolant and its first partial derivatives at the set of points (u_i, v_i, w_i), for $i = 1, 2, \ldots, n$.

E01THF must only be called after a call to E01TGF.

This routine is derived from the routine QS3GRD described by Renka [1].

4 References

[1] Renka R J (1988) Algorithm 661: QSHEP3D: Quadratic Shepard method for trivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 151–152

5 Parameters

1: M — INTEGER *Input*

2: X(M) — ***real*** array *Input*

3: Y(M) — ***real*** array *Input*

4: Z(M) — ***real*** array *Input*

5: F(M) — ***real*** array *Input*

On entry: M, X, Y, Z and F must be the same values as were supplied in the preceeding call to E01TGF.

6: IQ(LIQ) — INTEGER array *Input*

On entry: IQ must be unchanged from the value returned from a previous call to E01TGF.

7: LIQ — INTEGER *Input*

On entry: the dimension of the array IQ as declared in the (sub)program from which E01THF is called.

Constraint: LIQ $\geq 2 \times$ M $+ 1$.

8: RQ(LRQ) — ***real*** array *Input*

On entry: RQ must be unchanged from the value returned from a previous call to E01TGF.

Chapter E02 – Curve and Surface Fitting

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E02ACF	1	Minimax curve fit by polynomials
E02ADF	5	Least-squares curve fit, by polynomials, arbitrary data points
E02AEF	5	Evaluation of fitted polynomial in one variable from Chebyshev series form (simplified parameter list)
E02AFF	5	Least-squares polynomial fit, special data points (including interpolation)
E02AGF	8	Least-squares polynomial fit, values and derivatives may be constrained, arbitrary data points,
E02AHF	8	Derivative of fitted polynomial in Chebyshev series form
E02AJF	8	Integral of fitted polynomial in Chebyshev series form
E02AKF	8	Evaluation of fitted polynomial in one variable, from Chebyshev series form
E02BAF	5	Least-squares curve cubic spline fit (including interpolation)
E02BBF	5	Evaluation of fitted cubic spline, function only
E02BCF	7	Evaluation of fitted cubic spline, function and derivatives
E02BDF	7	Evaluation of fitted cubic spline, definite integral
E02BEF	13	Least-squares cubic spline curve fit, automatic knot placement
E02CAF	7	Least-squares surface fit by polynomials, data on lines
E02CBF	7	Evaluation of fitted polynomial in two variables
E02DAF	6	Least-squares surface fit, bicubic splines
E02DCF	13	Least-squares surface fit by bicubic splines with automatic knot placement, data on rectangular grid
E02DDF	13	Least-squares surface fit by bicubic splines with automatic knot placement, scattered data
E02DEF	14	Evaluation of a fitted bicubic spline at a vector of points
E02DFF	14	Evaluation of a fitted bicubic spline at a mesh of points
E02GAF	7	L_1-approximation by general linear function
E02GBF	7	L_1-approximation by general linear function subject to linear inequality constraints
E02GCF	8	L_∞-approximation by general linear function
E02RAF	7	Padé-approximants
E02RBF	7	Evaluation of fitted rational function as computed by E02RAF
E02ZAF	6	Sort 2-D data into panels for fitting bicubic splines

Chapter E02

Curve and Surface Fitting

Contents

1 Scope of the Chapter

The main aim of this chapter is to assist the user in finding a function which approximates a set of data points. Typically the data contain random errors, as of experimental measurement, which need to be smoothed out. To seek an approximation to the data, it is first necessary to specify for the approximating function a mathematical form (a polynomial, for example) which contains a number of unspecified coefficients: the appropriate fitting routine then derives for the coefficients the values which provide the best fit of that particular form. The chapter deals mainly with curve and surface fitting (i.e., fitting with functions of one and of two variables) when a polynomial or a cubic spline is used as the fitting function, since these cover the most common needs. However, fitting with other functions and/or more variables can be undertaken by means of general linear or nonlinear routines (some of which are contained in other chapters) depending on whether the coefficients in the function occur linearly or nonlinearly. Cases where a graph rather than a set of data points is given can be treated simply by first reading a suitable set of points from the graph.

The chapter also contains routines for evaluating, differentiating and integrating polynomial and spline curves and surfaces, once the numerical values of their coefficients have been determined.

There is, too, a routine for computing a Padé approximant of a mathematical function (see Section 2.6 and Section 3.8).

2 Background to the Problems

2.1 Preliminary Considerations

In the curve-fitting problems considered in this chapter, we have a dependent variable y and an independent variable x, and we are given a set of data points (x_r, y_r), for $r = 1, 2, \ldots, m$. The preliminary matters to be considered in this section will, for simplicity, be discussed in this context of curve-fitting problems. In fact, however, these considerations apply equally well to surface and higher-dimensional problems. Indeed, the discussion presented carries over essentially as it stands if, for these cases, we interpret x as a vector of several independent variables and correspondingly each x_r as a vector containing the rth data value of each independent variable.

We wish, then, to approximate the set of data points as closely as possible with a specified function, $f(x)$ say, which is as smooth as possible: $f(x)$ may, for example, be a polynomial. The requirements of smoothness and closeness conflict, however, and a balance has to be struck between them. Most often, the smoothness requirement is met simply by limiting the number of coefficients allowed in the fitting function – for example, by restricting the degree in the case of a polynomial. Given a particular number of coefficients in the function in question, the fitting routines of this chapter determine the values of the coefficients such that the 'distance' of the function from the data points is as small as possible. The necessary balance is struck by the user comparing a selection of such fits having different numbers of coefficients. If the number of coefficients is too low, the approximation to the data will be poor. If the number is too high, the fit will be too close to the data, essentially following the random errors and tending to have unwanted fluctuations between the data points. Between these extremes, there is often a group of fits all similarly close to the data points and then, particularly when least-squares polynomials are used, the choice is clear: it is the fit from this group having the smallest number of coefficients.

The above process can be seen as the user minimizing the smoothness measure (i.e., the number of coefficients) subject to the distance from the data points being acceptably small. Some of the routines, however, do this task themselves. They use a different measure of smoothness (in each case one that is continuous) and minimize it subject to the distance being less than a threshold specified by the user. This is a much more automatic process, requiring only some experimentation with the threshold.

2.1.1 Fitting criteria: norms

A measure of the above 'distance' between the set of data points and the function $f(x)$ is needed. The distance from a single data point (x_r, y_r) to the function can simply be taken as

$$\epsilon_r = y_r - f(x_r), \tag{1}$$

and is called the **residual** of the point. (With this definition, the residual is regarded as a function of the coefficients contained in $f(x)$; however, the term is also used to mean the particular value of ϵ_r which

corresponds to the fitted values of the coefficients.) However, we need a measure of distance for the set of data points as a whole. Three different measures are used in the different routines (which measure to select, according to circumstances, is discussed later in this sub-section). With ϵ_r defined in (1), these measures, or **norms**, are

$$\sum_{r=1}^{m} |\epsilon_r|, \tag{2}$$

$$\sqrt{\sum_{r=1}^{m} \epsilon_r^2}, \tag{3}$$

and

$$\max_r |\epsilon_r|, \tag{4}$$

respectively the l_1 norm, the l_2 norm and the l_∞ norm.

Minimization of one or other of these norms usually provides the fitting criterion, the minimization being carried out with respect to the coefficients in the mathematical form used for $f(x)$: with respect to the b_i for example if the mathematical form is the power series in (8) below. The fit which results from minimizing (2) is known as the l_1 fit, or the fit in the l_1 norm: that which results from minimizing (3) is the l_2 fit, the well-known least-squares fit (minimizing (3) is equivalent to minimizing the square of (3), i.e., the sum of squares of residuals, and it is the latter which is used in practice), and that from minimizing (4) is the l_∞, or minimax, fit.

Strictly speaking, implicit in the use of the above norms are the statistical assumptions that the random errors in the y_r are independent of one another and that any errors in the x_r are negligible by comparison. From this point of view, the use of the l_2 norm is appropriate when the random errors in the y_r have a normal distribution, and the l_∞ norm is appropriate when they have a rectangular distribution, as when fitting a table of values rounded to a fixed number of decimal places. The l_1 norm is appropriate when the error distribution has its frequency function proportional to the negative exponential of the modulus of the normalised error – not a common situation.

However, the user is often indifferent to these statistical considerations, and simply seeks a fit which can be assessed by inspection, perhaps visually from a graph of the results. In this event, the l_1 norm is particularly appropriate when the data are thought to contain some 'wild' points (since fitting in this norm tends to be unaffected by the presence of a small number of such points), though of course in simple situations the user may prefer to identify and reject these points. The l_∞ norm should be used only when the maximum residual is of particular concern, as may be the case for example when the data values have been obtained by accurate computation, as of a mathematical function. Generally, however, a routine based on least-squares should be preferred, as being computationally faster and usually providing more information on which to assess the results. In many problems the three fits will not differ significantly for practical purposes.

Some of the routines based on the l_2 norm do not minimize the norm itself but instead minimize some (intuitively acceptable) measure of smoothness subject to the norm being less than a user-specified threshold. These routines fit with cubic or bicubic splines (see (10) and (14) below) and the smoothing measures relate to the size of the discontinuities in their third derivatives. A much more automatic fitting procedure follows from this approach.

2.1.2 Weighting of data points

The use of the above norms also assumes that the data values y_r are of equal (absolute) accuracy. Some of the routines enable an allowance to be made to take account of differing accuracies. The allowance takes the form of 'weights' applied to the y-values so that those values known to be more accurate have a greater influence on the fit than others. These weights, to be supplied by the user, should be calculated from estimates of the absolute accuracies of the y-values, these estimates being expressed as standard deviations, probable errors or some other measure which has the same dimensions as y. Specifically, for each y_r the corresponding weight w_r should be inversely proportional to the accuracy estimate of y_r. For example, if the percentage accuracy is the same for all y_r, then the absolute accuracy of y_r is proportional to y_r (assuming y_r to be positive, as it usually is in such cases) and so $w_r = K/y_r$, for $r = 1, 2, \ldots, m$, for

an arbitrary positive constant K. (This definition of weight is stressed because often weight is defined as the square of that used here.) The norms (2), (3) and (4) above are then replaced respectively by

$$\sum_{r=1}^{m} |w_r \epsilon_r|, \tag{5}$$

$$\sqrt{\sum_{r=1}^{m} w_r^2 \epsilon_r^2}, \tag{6}$$

and

$$\max_r |w_r \epsilon_r|. \tag{7}$$

Again it is the square of (6) which is used in practice rather than (6) itself.

2.2 Curve Fitting

When, as is commonly the case, the mathematical form of the fitting function is immaterial to the problem, polynomials and cubic splines are to be preferred because their simplicity and ease of handling confer substantial benefits. The **cubic spline** is the more versatile of the two. It consists of a number of cubic polynomial segments joined end to end with continuity in first and second derivatives at the joins. The third derivative at the joins is in general discontinuous. The x-values of the joins are called **knots**, or, more precisely, interior knots. Their number determines the number of coefficients in the spline, just as the degree determines the number of coefficients in a polynomial.

2.2.1 Representation of polynomials

Two different forms for representing a polynomial are used in different routines. One is the usual power-series form

$$f(x) \equiv b_0 + b_1 x + b_2 x^2 + \ldots + b_k x^k. \tag{8}$$

The other is the Chebyshev series form

$$f(x) \equiv \tfrac{1}{2} a_0 T_0(x) + a_1 T_1(x) + a_2 T_2(x) + \ldots + a_k T_k(x), \tag{9}$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind of degree i (see Cox and Hayes [1], page 9), and where the range of x has been normalised to run from -1 to $+1$. The use of either form leads theoretically to the same fitted polynomial, but in practice results may differ substantially because of the effects of rounding error. The Chebyshev form is to be preferred, since it leads to much better accuracy in general, both in the computation of the coefficients and in the subsequent evaluation of the fitted polynomial at specified points. This form also has other advantages: for example, since the later terms in (9) generally decrease much more rapidly from left to right than do those in (8), the situation is more often encountered where the last terms are negligible and it is obvious that the degree of the polynomial can be reduced (note that on the interval $-1 \leq x \leq 1$ for all i, $T_i(x)$ attains the value unity but never exceeds it, so that the coefficient a_i gives directly the maximum value of the term containing it). If the power-series form is used it is most advisable to work with the variable x normalised to the range -1 to $+1$, carrying out the normalisation before entering the relevant routine. This will often substantially improve computational accuracy.

2.2.2 Representation of cubic splines

A cubic spline is represented in the form

$$f(x) \equiv c_1 N_1(x) + c_2 N_2(x) + \ldots + c_p N_p(x), \tag{10}$$

where $N_i(x)$, for $i = 1, 2, \ldots, p$, is a normalised cubic B-spline (see Hayes [2]). This form, also, has advantages of computational speed and accuracy over alternative representations.

2.3 Surface Fitting

There are now two independent variables, and we shall denote these by x and y. The dependent variable, which was denoted by y in the curve-fitting case, will now be denoted by f. (This is a rather different notation from that indicated for the general-dimensional problem in the first paragraph of Section 2.1, but it has some advantages in presentation.)

Again, in the absence of contrary indications in the particular application being considered, polynomials and splines are the approximating functions most commonly used.

2.3.1 Representation of bivariate polynomials

The type of bivariate polynomial currently considered in the chapter can be represented in either of the two forms

$$f(x,y) \equiv \sum_{i=0}^{k} \sum_{j=0}^{l} b_{ij} x^i y^j, \tag{11}$$

and

$$f(x,y) \equiv \sum_{i=0}^{k'} \sum_{j=0}^{l'} a_{ij} T_i(x) T_j(y), \tag{12}$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind of degree i in the argument x (see Cox and Hayes [1] page 9), and correspondingly for $T_j(y)$. The prime on the two summation signs, following standard convention, indicates that the first term in each sum is halved, as shown for one variable in equation (9). The two forms (11) and (12) are mathematically equivalent, but again the Chebyshev form is to be preferred on numerical grounds, as discussed in Section 2.2.1.

2.3.2 Bicubic splines: definition and representation

The bicubic spline is defined over a rectangle R in the (x,y) plane, the sides of R being parallel to the x- and y-axes. R is divided into rectangular panels, again by lines parallel to the axes. Over each panel the bicubic spline is a bicubic polynomial, that is it takes the form

$$\sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j. \tag{13}$$

Each of these polynomials joins the polynomials in adjacent panels with continuity up to the second derivative. The constant x-values of the dividing lines parallel to the y-axis form the set of interior knots for the variable x, corresponding precisely to the set of interior knots of a cubic spline. Similarly, the constant y-values of dividing lines parallel to the x-axis form the set of interior knots for the variable y. Instead of representing the bicubic spline in terms of the above set of bicubic polynomials, however, it is represented, for the sake of computational speed and accuracy, in the form

$$f(x,y) = \sum_{i=1}^{p} \sum_{j=1}^{q} c_{ij} M_i(x) N_j(y), \tag{14}$$

where $M_i(x)$, for $i = 1,2,\ldots,p$, and $N_j(y)$, for $j = 1,2,\ldots,q$, are normalised B-splines (see Hayes and Halliday [4] for further details of bicubic splines and Hayes [2] for normalised B-splines).

2.4 General Linear and Nonlinear Fitting Functions

We have indicated earlier that, unless the data-fitting application under consideration specifically requires some other type of fitting function, a polynomial or a spline is usually to be preferred. Special routines for these functions, in one and in two variables, are provided in this chapter. When the application does specify some other fitting function, however, it may be treated by a routine which deals with a general linear function, or by one for a general nonlinear function, depending on whether the coefficients in the given function occur linearly or nonlinearly.

The general linear fitting function can be written in the form

$$f(x) \equiv c_1\phi_1(x) + c_2\phi_2(x) + \ldots + c_p\phi_p(x), \tag{15}$$

where x is a vector of one or more independent variables, and the ϕ_i are any given functions of these variables (though they must be linearly independent of one another if there is to be the possibility of a unique solution to the fitting problem). This is not intended to imply that each ϕ_i is necessarily a function of all the variables: we may have, for example, that each ϕ_i is a function of a different single variable, and even that one of the ϕ_i is a constant. All that is required is that a value of each $\phi_i(x)$ can be computed when a value of each independent variable is given.

When the fitting function $f(x)$ is not linear in its coefficients, no more specific representation is available in general than $f(x)$ itself. However, we shall find it helpful later on to indicate the fact that $f(x)$ contains a number of coefficients (to be determined by the fitting process) by using instead the notation $f(x;c)$, where c denotes the vector of coefficients. An example of a nonlinear fitting function is

$$f(x;c) \equiv c_1 + c_2 \exp(-c_4 x) + c_3 \exp(-c_5 x), \tag{16}$$

which is in one variable and contains five coefficients. Note that here, as elsewhere in this Chapter Introduction, we use the term 'coefficients' to include all the quantities whose values are to be determined by the fitting process, not just those which occur linearly. We may observe that it is only the presence of the coefficients c_4 and c_5 which makes the form (16) nonlinear. If the values of these two coefficients were known beforehand, (16) would instead be a linear function which, in terms of the general linear form (15), has $p = 3$ and

$$\phi_1(x) \equiv 1, \ \phi_2(x) \equiv \exp(-c_4 x), \text{ and } \phi_3(x) \equiv \exp(-c_5 x).$$

We may note also that polynomials and splines, such as (9) and (14), are themselves linear in their coefficients. Thus if, when fitting with these functions, a suitable special routine is not available (as when more than two independent variables are involved or when fitting in the l_1 norm), it is appropriate to use a routine designed for a general linear function.

2.5 Constrained Problems

So far, we have considered only fitting processes in which the values of the coefficients in the fitting function are determined by an unconstrained minimization of a particular norm. Some fitting problems, however, require that further restrictions be placed on the determination of the coefficient values. Sometimes these restrictions are contained explicitly in the formulation of the problem in the form of equalities or inequalities which the coefficients, or some function of them, must satisfy. For example, if the fitting function contains a term $A\exp(-kx)$, it may be required that $k \geq 0$. Often, however, the equality or inequality constraints relate to the value of the fitting function or its derivatives at specified values of the independent variable(s), but these too can be expressed in terms of the coefficients of the fitting function, and it is appropriate to do this if a general linear or nonlinear routine is being used. For example, if the fitting function is that given in (10), the requirement that the first derivative of the function at $x = x_0$ be non-negative can be expressed as

$$c_1 N_1'(x_0) + c_2 N_2'(x_0) + \ldots + c_p N_p'(x_0) \geq 0, \tag{17}$$

where the prime denotes differentiation with respect to x and each derivative is evaluated at $x = x_0$. On the other hand, if the requirement had been that the derivative at $x = x_0$ be exactly zero, the inequality sign in (17) would be replaced by an equality.

Routines which provide a facility for minimizing the appropriate norm subject to such constraints are discussed in Section 3.6.

2.6 Padé Approximants

A Padé approximant to a function $f(x)$ is a rational function (ratio of two polynomials) whose Maclaurin-series expansion is the same as that of $f(x)$ up to and including the term in x^k, where k is the sum of the degrees of the numerator and denominator of the approximant. Padé approximation can be a useful technique when values of a function are to be obtained from its Maclaurin series but convergence of the series is unacceptably slow or even non-existent.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 General

The choice of a routine to treat a particular fitting problem will depend first of all on the fitting function and the norm to be used. Unless there is good reason to the contrary, the fitting function should be a polynomial or a cubic spline (in the appropriate number of variables) and the norm should be the l_2 norm (leading to the least-squares fit). If some other function is to be used, the choice of routine will depend on whether the function is nonlinear (in which case see Section 3.5.2) or linear in its coefficients (see Section 3.5.1), and, in the latter case, on whether the l_1, l_2 or l_∞ norm is to be used. The latter section is appropriate for polynomials and splines, too, if the l_1 or l_∞ norm is preferred, with one exception: there is a special routine for fitting polynomial curves in the unweighted l_∞ norm (see Section 3.2.3).

In the case of a polynomial or cubic spline, if there is only one independent variable, the user should choose a spline (Section 3.3) when the curve represented by the data is of complicated form, perhaps with several peaks and troughs. When the curve is of simple form, first try a polynomial (see Section 3.2) of low degree, say up to degree 5 or 6, and then a spline if the polynomial fails to provide a satisfactory fit. (Of course, if third-derivative discontinuities are unacceptable to the user, a polynomial is the only choice.) If the problem is one of surface fitting, the polynomial routine (Section 3.4.1) should be tried first if the data arrangement happens to be appropriate, otherwise one of the spline routines (Section 3.4.2). If the problem has more than two independent variables, it may be treated by the general linear routine in Section 3.5.1, again using a polynomial in the first instance.

Another factor which affects the choice of routine is the presence of constraints, as previously discussed in Section 2.5. Indeed this factor is likely to be overriding at present, because of the limited number of routines which have the necessary facility. Consequently those routines have been grouped together for discussion in Section 3.6.

3.1.1 Data considerations

A satisfactory fit cannot be expected by any means if the number and arrangement of the data points do not adequately represent the character of the underlying relationship: sharp changes in behaviour, in particular, such as sharp peaks, should be well covered. Data points should extend over the whole range of interest of the independent variable(s): extrapolation outside the data ranges is most unwise. Then, with polynomials, it is advantageous to have additional points near the ends of the ranges, to counteract the tendency of polynomials to develop fluctuations in these regions. When, with polynomial curves, the user can precisely choose the x-values of the data, the special points defined in Section 3.2.2 should be selected. With polynomial surfaces, each of these same x-values should, where possible, be combined with each of a corresponding set of y-values (not necessarily with the same value of n), thus forming a rectangular grid of (x, y)-values. With splines the choice is less critical as long as the character of the relationship is adequately represented. All fits should be tested graphically before accepting them as satisfactory.

For this purpose it should be noted that it is not sufficient to plot the values of the fitted function only at the data values of the independent variable(s); at the least, its values at a similar number of intermediate points should also be plotted, as unwanted fluctuations may otherwise go undetected. Such fluctuations are the less likely to occur the lower the number of coefficients chosen in the fitting function. No firm guide can be given, but as a rough rule, at least initially, the number of coefficients should not exceed half the number of data points (points with equal or nearly equal values of the independent variable, or both independent variables in surface fitting, counting as a single point for this purpose). However, the situation may be such, particularly with a small number of data points, that a satisfactorily close fit to the data cannot be achieved without unwanted fluctuations occurring. In such cases, it is often possible to improve the situation by a transformation of one or more of the variables, as discussed in the next section: otherwise it will be necessary to provide extra data points. Further advice on curve fitting is given in Cox and Hayes [1] and, for polynomials only, in Hayes [3]. Much of the advice applies also to surface fitting; see also the routine documents.

3.1.2 Transformation of variables

Before starting the fitting, consideration should be given to the choice of a good form in which to deal with each of the variables: often it will be satisfactory to use the variables as they stand, but sometimes the use of the logarithm, square root, or some other function of a variable will lead to a better-behaved relationship. This question is customarily taken into account in preparing graphs and tables of a relationship and the same considerations apply when curve or surface fitting. The practical context will often give a guide. In general, it is best to avoid having to deal with a relationship whose behaviour in one region is radically different from that in another. A steep rise at the left-hand end of a curve, for example, can often best be treated by curve fitting in terms of $\log(x + c)$ with some suitable value of the constant c. A case when such a transformation gave substantial benefit is discussed in page 60 of Hayes [3]. According to the features exhibited in any particular case, transformation of either dependent variable or independent variable(s) or both may be beneficial. When there is a choice it is usually better to transform the independent variable(s): if the dependent variable is transformed, the weights attached to the data points must be adjusted. Thus (denoting the dependent variable by y, as in the notation for curves) if the y_r to be fitted have been obtained by a transformation $y = g(Y)$ from original data values Y_r, with weights W_r, for $r = 1, 2, \ldots, m$, we must take

$$w_r = W_r/(dy/dY), \tag{18}$$

where the derivative is evaluated at Y_r. Strictly, the transformation of Y and the adjustment of weights are valid only when the data errors in the Y_r are small compared with the range spanned by the Y_r, but this is usually the case.

3.2 Polynomial Curves

3.2.1 Least-squares polynomials: arbitrary data points

E02ADF fits to arbitrary data points, with arbitrary weights, polynomials of all degrees up to a maximum degree k, which is a choice. If the user is seeking only a low-degree polynomial, up to degree 5 or 6 say, $k = 10$ is an appropriate value, providing there are about 20 data points or more. To assist in deciding the degree of polynomial which satisfactorily fits the data, the routine provides the root-mean-square residual s_i for all degrees $i = 1, 2, \ldots, k$. In a satisfactory case, these s_i will decrease steadily as i increases and then settle down to a fairly constant value, as shown in the example

i	s_i
0	3.5215
1	0.7708
2	0.1861
3	0.0820
4	0.0554
5	0.0251
6	0.0264
7	0.0280
8	0.0277
9	0.0297
10	0.0271

If the s_i values settle down in this way, it indicates that the closest polynomial approximation justified by the data has been achieved. The degree which first gives the approximately constant value of s_i (degree 5 in the example) is the appropriate degree to select. (Users who are prepared to accept a fit higher than sixth degree should simply find a high enough value of k to enable the type of behaviour indicated by the example to be detected: thus they should seek values of k for which at least 4 or 5 consecutive values of s_i are approximately the same.) If the degree were allowed to go high enough, s_i would, in most cases, eventually start to decrease again, indicating that the data points are being fitted too closely and that undesirable fluctuations are developing between the points. In some cases, particularly with a small number of data points, this final decrease is not distinguishable from the initial decrease in s_i. In such cases, users may seek an acceptable fit by examining the graphs of several of the polynomials obtained. Failing this, they may (a) seek a transformation of variables which improves the behaviour, (b) try fitting a spline, or (c) provide more data points. If data can be provided simply by drawing an approximating curve by hand and reading points from it, use the points discussed in Section 3.2.2.

3.2.2 Least-squares polynomials: selected data points

When users are at liberty to choose the x-values of data points, such as when the points are taken from a graph, it is most advantageous when fitting with polynomials to use the values $x_r = \cos(\pi r/n)$, for $r = 0, 1, \ldots, n$ for some value of n, a suitable value for which is discussed at the end of this section. Note that these x_r relate to the variable x after it has been normalised so that its range of interest is -1 to $+1$. E02ADF may then be used as in Section 3.2.1 to seek a satisfactory fit. However, if the ordinate values are of equal weight, as would often be the case when they are read from a graph, E02AFF is to be preferred, as being simpler to use and faster. This latter algorithm provides the coefficients a_j, for $j = 0, 1, \ldots, n$, in the Chebyshev series form of the polynomial of degree n which interpolates the data. In a satisfactory case, the later coefficients in this series, after some initial significant ones, will exhibit a random behaviour, some positive and some negative, with a size about that of the errors in the data or less. All these 'random' coefficients should be discarded, and the remaining (initial) terms of the series be taken as the approximating polynomial. This truncated polynomial is a least-squares fit to the data, though with the point at each end of the range given half the weight of each of the other points. The following example illustrates a case in which degree 5 or perhaps 6 would be chosen for the approximating polynomial.

j	a_j
0	9.315
1	−8.030
2	0.303
3	−1.483
4	0.256
5	−0.386
6	0.076
7	0.022
8	0.014
9	0.005
10	0.011
11	−0.040
12	0.017
13	−0.054
14	0.010
15	−0.034
16	−0.001

Basically, the value of n used needs to be large enough to exhibit the type of behaviour illustrated in the above example. A value of 16 is suggested as being satisfactory for very many practical problems, the required cosine values for this value of n being given in Cox and Hayes [1], page 11. If a satisfactory fit is not obtained, a spline fit should be tried, or, if the user is prepared to accept a higher degree of polynomial, n should be increased: doubling n is an advantageous strategy, since the set of values $\cos(\pi r/n)$, for $r = 0, 1, \ldots, n$, contains all the values of $\cos(\pi r/2n)$, for $r = 0, 1, \ldots, 2n$, so that the old data set will then be re-used in the new one. Thus, for example, increasing n from 16 to 32 will require only 16 new data points, a smaller number than for any other increase of n. If data points are particularly expensive to obtain, a smaller initial value than 16 may be tried, provided the user is satisfied that the number is adequate to reflect the character of the underlying relationship. Again, the number should be doubled if a satisfactory fit is not obtained.

3.2.3 Minimax space polynomials

E02ACF determines the polynomial of given degree which is a minimax space fit to arbitrary data points with equal weights. (If unequal weights are required, the polynomial must be treated as a general linear function and fitted using E02GCF.) To arrive at a satisfactory degree it will be necessary to try several different degrees and examine the results graphically. Initial guidance can be obtained from the value of the maximum residual: this will vary with the degree of the polynomial in very much the same way as does s_i in least-squares fitting, but it is much more expensive to investigate this behaviour in the same detail.

The algorithm uses the power-series form of the polynomial so for numerical accuracy it is advisable to normalise the data range of x to $[-1, 1]$.

3.3 Cubic Spline Curves

3.3.1 Least-squares cubic splines

E02BAF fits to arbitrary data points, with arbitrary weights, a cubic spline with interior knots specified by the user. The choice of these knots so as to give an acceptable fit must largely be a matter of trial and error, though with a little experience a satisfactory choice can often be made after one or two trials. It is usually best to start with a small number of knots (too many will result in unwanted fluctuations in the fit, or even in there being no unique solution) and, examining the fit graphically at each stage, to add a few knots at a time at places where the fit is particularly poor. Moving the existing knots towards these places will also often improve the fit. In regions where the behaviour of the curve underlying the data is changing rapidly, closer knots will be needed than elsewhere. Otherwise, positioning is not usually very critical and equally-spaced knots are often satisfactory. See also the next section, however.

A useful feature of the routine is that it can be used in applications which require the continuity to be less than the normal continuity of the cubic spline. For example, the fit may be required to have a discontinuous slope at some point in the range. This can be achieved by placing three coincident knots at the given point. Similarly a discontinuity in the second derivative at a point can be achieved by placing two knots there. Analogy with these discontinuous cases can provide guidance in more usual cases: for example, just as three coincident knots can produce a discontinuity in slope, so three close knots can produce a rapid change in slope. The closer the knots are, the more rapid can the change be.

An example set of data is given in Figure 1. It is a rather tricky set, because of the scarcity of data on the right, but it will serve to illustrate some of the above points and to show some of the dangers to be avoided. Three interior knots (indicated by the vertical lines at the top of the diagram) are chosen as a start. We see that the resulting curve is not steep enough in the middle and fluctuates at both ends, severely on the right. The spline is unable to cope with the shape and more knots are needed.

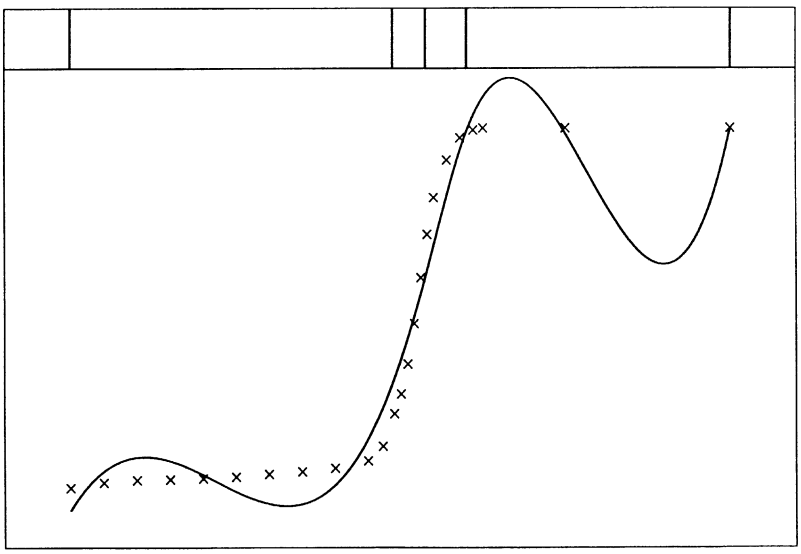


Figure 1

In Figure 2, three knots have been added in the centre, where the data shows a rapid change in behaviour, and one further out at each end, where the fit is poor. The fit is still poor, so a further knot is added in this region and, in Figure 3, disaster ensues in rather spectacular fashion.

The reason is that, at the right-hand end, the fits in Figure 1 and 2 have been interpreted as poor simply because of the fluctuations about the curve underlying the data (or what it is naturally assumed to be). But the fitting process knows only about the data and nothing else about the underlying curve, so it is important to consider only closeness to the data when deciding goodness of fit.

Thus, in Figure 1, the curve fits the last two data points quite well compared with the fit elsewhere, so no knot should have been added in this region. In Figure 2, the curve goes exactly through the last two points, so a further knot is certainly not needed here.

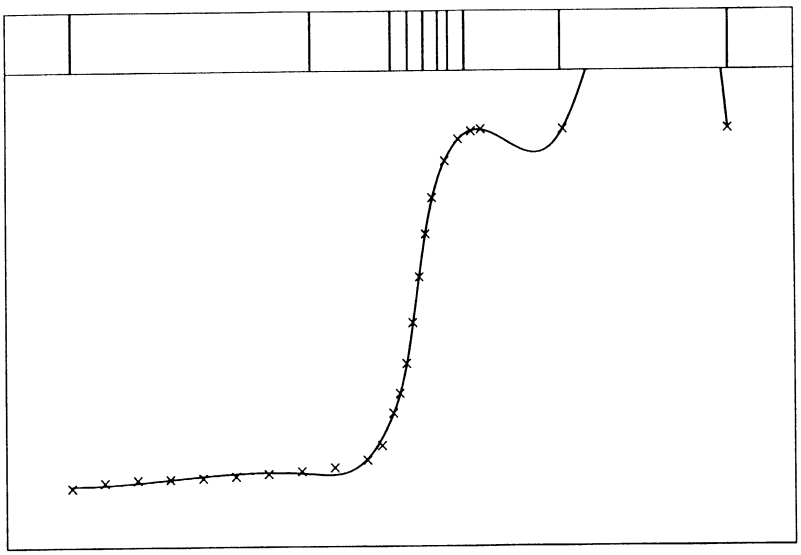


Figure 2

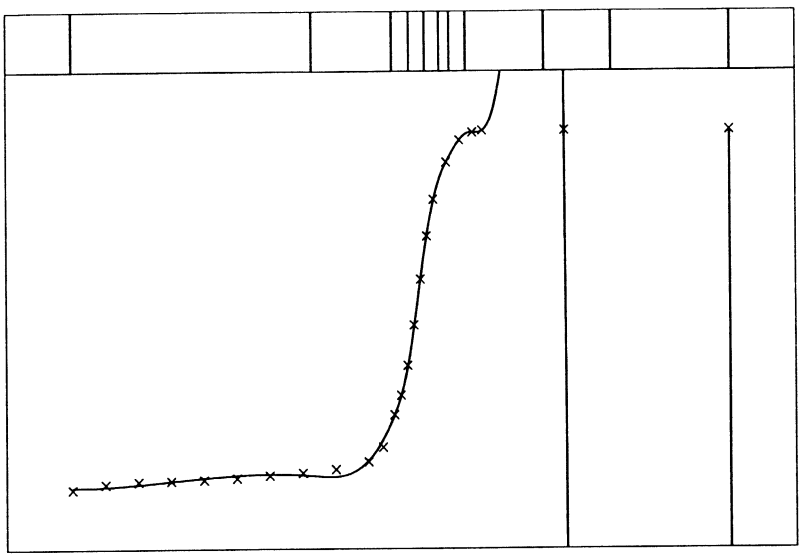


Figure 3

Figure 4 shows what can be achieved without the extra knot on each of the flat regions. Remembering that within each knot interval the spline is a cubic polynomial, there is really no need to have more than one knot interval covering each flat region.

What we have, in fact, in Figures 2 and 3 is a case of too many knots (so too many coefficients in the spline equation) for the number of data points. The warning in the second paragraph of Section 2.1 was that the fit will then be too close to the data, tending to have unwanted fluctuations between the data points. The warning applies locally for splines, in the sense that, in localities where there are plenty of data points, there can be a lot of knots, as long as there are few knots where there are few points, especially near the ends of the interval. In the present example, with so few data points on the right, just the one extra knot in Figure 2 is too many! The signs are clearly present, with the last two points fitted exactly (at least to the graphical accuracy and actually much closer than that) and fluctuations **within** the last two knot-intervals (cf. Figure 1, where only the final point is fitted exactly and one of the wobbles spans several data points).

The situation in Figure 3 is different. The fit, if computed exactly, **would** still pass through the last two data points, with even more violent fluctuations. However, the problem has become so ill-conditioned that all accuracy has been lost. Indeed, if the last interior knot were moved a tiny amount to the right,

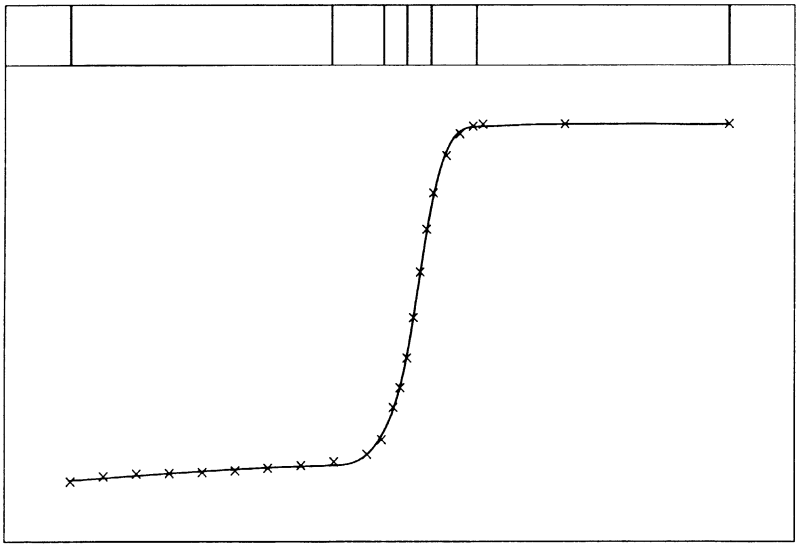


Figure 4

there would be no unique solution and an error message would have been caused. **Near-singularity** is, sadly, not picked up by the routine, but can be spotted readily in a graph, as Figure 3. B-spline coefficients becoming large, with alternating signs, is another indication. However, it is better to avoid such situations, firstly by providing, whenever possible, data adequately covering the range of interest, and secondly by placing knots only where there is a reasonable amount of data.

The example here could, in fact, have utilised from the start the observation made in the second paragraph of this section, that three close knots can produce a rapid change in slope. The example has two such rapid changes and so requires two sets of three close knots (in fact, the two sets can be so close that one knot can serve in both sets, so only five knots prove sufficient in Figure 4). It should be noted, however, that the rapid turn occurs within the range spanned by the three knots. This is the reason that the six knots in Figure 2 are not satisfactory as they do not quite span the two turns.

Some more examples to illustrate the choice of knots are given in Cox and Hayes [1].

3.3.2 Automatic fitting with cubic splines

E02BEF also fits cubic splines to arbitrary data points with arbitrary weights but itself chooses the number and positions of the knots. The user has to supply only a threshold for the sum of squares of residuals. The routine first builds up a knot set by a series of trial fits in the l_2 norm. Then, with the knot set decided, the final spline is computed to minimize a certain smoothing measure subject to satisfaction of the chosen threshold. Thus it is easier to use than E02BAF (see previous section), requiring only some experimentation with this threshold. It should therefore be first choice unless the user has a preference for the ordinary least-squares fit or, for example, wishes to experiment with knot positions, trying to keep their number down (E02BEF aims only to be reasonably frugal with knots).

3.4 Polynomial and Spline Surfaces

3.4.1 Least-squares polynomials

E02CAF fits bivariate polynomials of the form (12), with k and l specified by the user, to data points in a particular, but commonly occurring, arrangement. This is such that, when the data points are plotted in the plane of the independent variables x and y, they lie on lines parallel to the x-axis. Arbitrary weights are allowed. The matter of choosing satisfactory values for k and l is discussed in Section 8 of the routine document.

3.4.2 Least-squares bicubic splines

E02DAF fits to arbitrary data points, with arbitrary weights, a bicubic spline with its two sets of interior knots specified by the user. For choosing these knots, the advice given for cubic splines, in Section 3.3.1

above, applies here too. (See also the next section, however.) If changes in the behaviour of the surface underlying the data are more marked in the direction of one variable than of the other, more knots will be needed for the former variable than the latter. Note also that, in the surface case, the reduction in continuity caused by coincident knots will extend across the whole spline surface: for example, if three knots associated with the variable x are chosen to coincide at a value L, the spline surface will have a discontinuous slope across the whole extent of the line $x = L$.

With some sets of data and some choices of knots, the least-squares bicubic spline will not be unique. This will not occur, with a reasonable choice of knots, if the rectangle R is well covered with data points: here R is defined as the smallest rectangle in the (x, y) plane, with sides parallel to the axes, which contains all the data points. Where the least-squares solution is not unique, the minimal least-squares solution is computed, namely that least-squares solution which has the smallest value of the sum of squares of the B-spline coefficients c_{ij} (see the end of Section 2.3.2 above). This choice of least-squares solution tends to minimize the risk of unwanted fluctuations in the fit. The fit will not be reliable, however, in regions where there are few or no data points.

3.4.3 Automatic fitting with bicubic splines

E02DDF also fits bicubic splines to arbitrary data points with arbitrary weights but chooses the knot sets itself. The user has to supply only a threshold for the sum of squares of residuals. Just like the automatic curve E02BEF (Section 3.3.2), E02DDF then builds up the knot sets and finally fits a spline minimizing a smoothing measure subject to satisfaction of the threshold. Again, this easier to use routine is normally to be preferred, at least in the first instance.

E02DCF is a very similar routine to E02DDF but deals with data points of equal weight which lie on a rectangular mesh in the (x, y) plane. This kind of data allows a very much faster computation and so is to be preferred when applicable. Substantial departures from equal weighting can be ignored if the user is not concerned with statistical questions, though the quality of the fit will suffer if this is taken too far. In such cases, the user should revert to E02DDF.

3.5 General Linear and Nonlinear Fitting Functions

3.5.1 General linear functions

For the general linear function (15), routines are available for fitting in all three norms. The least-squares routines (which are to be preferred unless there is good reason to use another norm – see Section 2.1.1) are in Chapter F04. The l_∞ routine is E02GCF. Two routines for the l_1 norm are provided, E02GAF and E02GBF. Of these two, the former should be tried in the first instance, since it will be satisfactory in most cases, has a much shorter code and is faster. E02GBF, however, uses a more stable computational algorithm and therefore may provide a solution when E02GAF fails to do so. It also provides a facility for imposing linear inequality constraints on the solution (see Section 3.6).

All the above routines are essentially linear algebra routines, and in considering their use we need to view the fitting process in a slightly different way from hitherto. Taking y to be the dependent variable and x the vector of independent variables, we have, as for equation (1) but with each x_r now a vector,

$$\epsilon_r = y_r - f(x_r), \quad r = 1, 2, \ldots, m.$$

Substituting for $f(x)$ the general linear form (15), we can write this as

$$c_1\phi_1(x_r) + c_2\phi_2(x_r) + \ldots + c_p\phi_p(x_r) = y_r - \epsilon_r, \quad r = 1, 2, \ldots, m. \tag{19}$$

Thus we have a system of linear equations in the coefficients c_j. Usually, in writing these equations, the ϵ_r are omitted and simply taken as implied. The system of equations is then described as an overdetermined system (since we must have $m \geq p$ if there is to be the possibility of a unique solution to our fitting problem), and the fitting process of computing the c_j to minimize one or other of the norms (2), (3) and (4) can be described, in relation to the system of equations, as solving the overdetermined system in that particular norm. In matrix notation, the system can be written as

$$\Phi c = y, \tag{20}$$

where Φ is the m by p matrix whose element in row r and column j is $\phi_j(x_r)$, for $r = 1, 2, \ldots, m$; $j = 1, 2, \ldots, p$. The vectors c and y respectively contain the coefficients c_j and the data values y_r.

All four routines, however, use the standard notation of linear algebra, the overdetermined system of equations being denoted by

$$Ax = b. \tag{21}$$

(In fact, F04AMF can deal with several right-hand sides simultaneously, and thus is concerned with a matrix of right-hand sides, denoted by B, instead of the single vector b, and correspondingly with a matrix X of solutions instead of the single vector x.) The correspondence between this notation and that which we have used for the data-fitting problem (20) is therefore given by

$$\begin{aligned} A &\equiv \Phi, \\ x &\equiv c, \\ b &\equiv y. \end{aligned} \tag{22}$$

Note that the norms used by these routines are the unweighted norms (2), (3) and (4). If the user wishes to apply weights to the data points, that is to use the norms (5), (6) or (7), the equivalences (22) should be replaced by

$$\begin{aligned} A &\equiv D\Phi, \\ x &\equiv c, \\ b &\equiv Dy, \end{aligned}$$

where D is a diagonal matrix with w_r as the rth diagonal element. Here w_r, for $r = 1, 2, \ldots, m$, is the weight of the rth data point as defined in Section 2.1.2.

3.5.2 Nonlinear functions

Routines for fitting with a nonlinear function in the l_2 norm are provided in Chapter E04. The the E04 Chapter Introduction should be consulted for the appropriate choice of routine. Again, however, the notation adopted is different from that we have used for data fitting. In the latter, we denote the fitting function by $f(x; c)$, where x is the vector of independent variables and c is the vector of coefficients, whose values are to be determined. The squared l_2 norm, to be minimized with respect to the elements of c, is then

$$\sum_{r=1}^{m} w_r^2 [y_r - f(x_r; c)]^2 \tag{23}$$

where y_r is the rth data value of the dependent variable, x_r is the vector containing the rth values of the independent variables, and w_r is the corresponding weight as defined in Section 2.1.2.

On the other hand, in the nonlinear least-squares routines of Chapter E04, the function to be minimized is denoted by

$$\sum_{i=1}^{m} f_i^2(x), \tag{24}$$

the minimization being carried out with respect to the elements of the vector x. The correspondence between the two notations is given by

$$x \equiv c \text{ and } f_i(x) \equiv w_r[y_r - f(x_r; c)], \quad i = r = 1, 2, \ldots, m.$$

Note especially that the vector x of variables of the nonlinear least-squares routines is the vector c of coefficients of the data-fitting problem, and in particular that, if the selected routine requires derivatives of the $f_i(x)$ to be provided, these are derivatives of $w_r[y_r - f(x_r; c)]$ with respect to the coefficients of the data-fitting problem.

3.6 Constraints

At present, there are only a limited number of routines which fit subject to constraints. E02GBF allows the imposition of linear inequality constraints (the inequality (17) for example) when fitting with the general linear function in the l_1 norm. In addition, Chapter E04 contains a routine, E04UNF, which can be used for fitting with a nonlinear function in the l_2 norm subject to general equality or inequality constraints.

The remaining two constraint routines relate to fitting with polynomials in the l_2 norm. E02AGF deals with polynomial curves and allows precise values of the fitting function and (if required) all its derivatives

up to a given order to be prescribed at one or more values of the independent variable. The related surface-fitting E02CAF, designed for data on lines as discussed in Section 3.4.1, has a feature which permits precise values of the function and its derivatives to be imposed all along one or more lines parallel to the x- or y-axes (see the routine document for the relationship between these normalised variables and the user's original variables). In this case, however, the prescribed values cannot be supplied directly to the routine: instead, the user must provide modified data ordinates $F_{r,s}$ and polynomial factors $\gamma_1(x)$ and $\gamma_2(x)$, as defined on page 95 of Hayes [5].

3.7 Evaluation, Differentiation and Integration

Routines are available to evaluate, differentiate and integrate polynomials in Chebyshev-series form and cubic or bicubic splines in B-spline form. These polynomials and splines may have been produced by the various fitting routines or, in the case of polynomials, from prior calls of the differentiation and integration routines themselves.

E02AEF and E02AKF evaluate polynomial curves: the latter has a longer parameter list but does not require the user to normalise the values of the independent variable and can accept coefficients which are not stored in contiguous locations. E02CBF evaluates polynomial surfaces, E02BBF cubic spline curves, and E02DEF and E02DFF bicubic spline surfaces.

Differentiation and integration of polynomial curves are carried out by E02AHF and E02AJF respectively. The results are provided in Chebyshev-series form and so repeated differentiation and integration are catered for. Values of the derivative or integral can then be computed using the appropriate evaluation routine. Polynomial surfaces can be treated by a sequence of calls of one or other of the same two routines, differentiating or integrating the form (12) piece by piece. For example, if, for some given value of j, the coefficients a_{ij}, for $i = 0, 1, \ldots, k$, are supplied to E02AHF, we obtain coefficients $\bar{a}_{ij}$ say, for $i = 0, 1, \ldots, k-1$, which are the coefficients in the derivative with respect to x of the polynomial

$$\sum_{i=0}^{k}{}' a_{ij} T_i(x).$$

If this is repeated for all values of j, we obtain all the coefficients in the derivative of the surface with respect to x, namely

$$\sum_{i=0}^{k-1}{}' \sum_{j=0}^{l}{}' \bar{a}_{ij} T_j(y). \tag{25}$$

The derivative of (12), or of (25), with respect to y can be obtained in a corresponding manner. In the latter case, for example, for each value of i in turn we supply the coefficients $\bar{a}_{i0}, \bar{a}_{i1}, \bar{a}_{i2}, \ldots$, to the routine. Values of the resulting polynomials, such as (25), can subsequently be computed using E02CBF. It is important, however, to note one exception: the process described will not give valid results for differentiating or integrating a surface with respect to y if the normalisation of x was made dependent upon y, an option which is available in the fitting routine E02CAF.

For splines the differentiation and integration routines provided are of a different nature from those for polynomials. E02BCF provides values of a cubic spline curve and its first three derivatives (the rest, of course, are zero) at a given value of x. E02BDF computes the value of the definite integral of a cubic spline over its whole range. Again the routines can be applied to surfaces, this time of the form (14). For example, if, for each value of j in turn, the coefficients c_{ij}, for $i = 1, 2, \ldots, p$, are supplied to E02BCF with $x = x_0$ and on each occasion we select from the output the value of the second derivative, d_j say, and if the whole set of d_j are then supplied to the same routine with $x = y_0$, the output will contain all the values at (x_0, y_0) of

$$\frac{\partial^2 f}{\partial x^2} \text{ and } \frac{\partial^{r+2} f}{\partial x^2 \partial y^r}, \quad r = 1, 2, 3.$$

Equally, if after each of the first p calls of E02BCF we had selected the function value (E02BBF would also provide this) instead of the second derivative and we had supplied these values to E02BDF, the result obtained would have been the value of

$$\int_A^B f(x_0, y)\, dy,$$

where A and B are the end-points of the y interval over which the spline was defined.

3.8 Padé Approximants

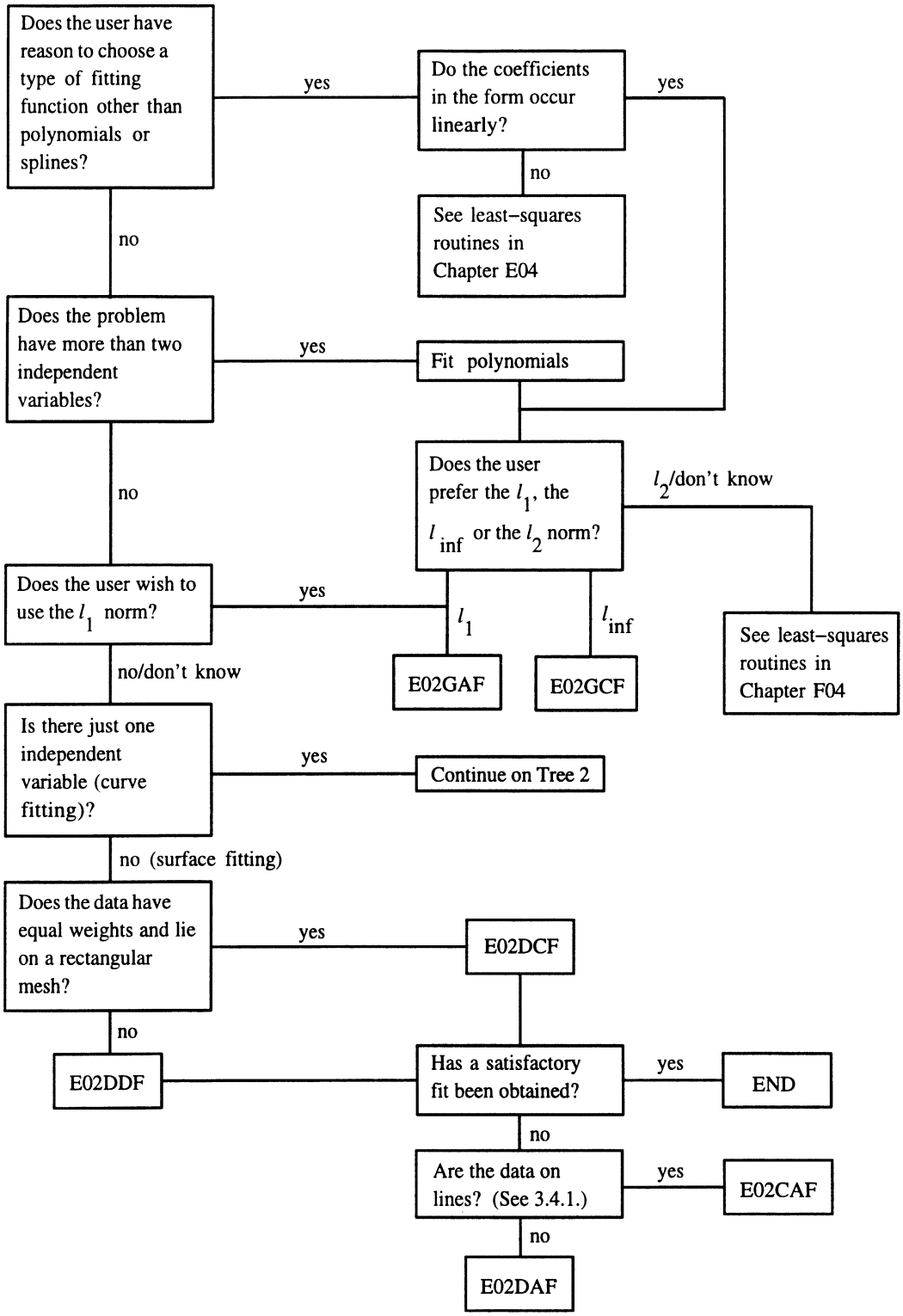
Given two non-negative integers l and m, and the coefficients in the Maclaurin expansion of a function up to degree $l+m$, E02RAF calculates the Padé approximant of degree l in the numerator and degree m in the denominator. For advice on the use of this routine, see the routine document, Sections 3 and 10.

E02RBF is provided to compute values of the Padé approximant, once it has been obtained.

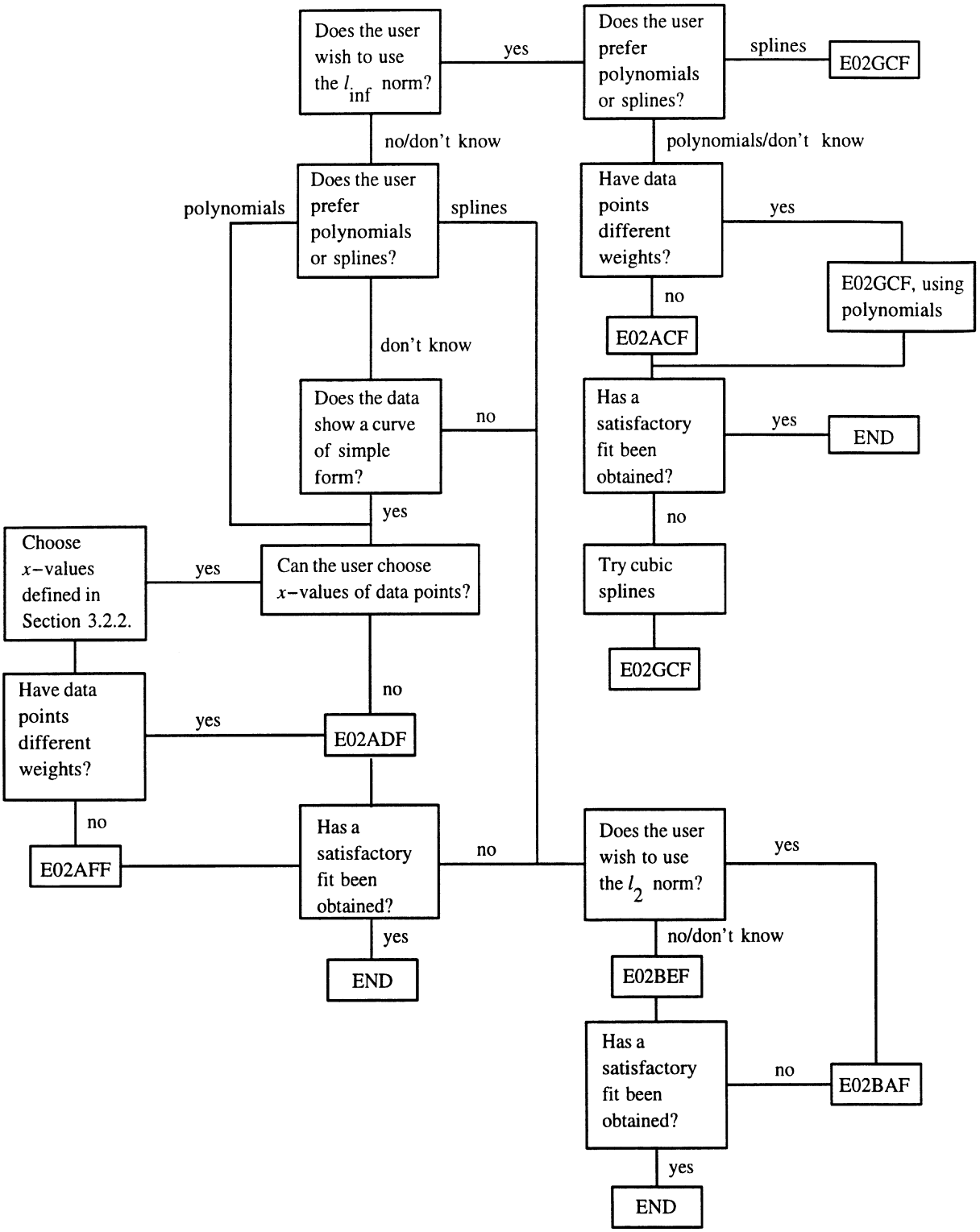
4 Decision Trees

Note. These Decision Trees are concerned with unconstrained fitting: for constrained fitting, consult Section 3.6.

Tree 1



Tree 2


Does the user wish to use the l_{inf} norm?
yes
Does the user prefer polynomials or splines?
splines
E02GCF
no/don't know
polynomials/don't know
Does the user prefer polynomials or splines?
polynomials
splines
Have data points different weights?
yes
E02GCF, using polynomials
no
E02ACF
don't know
Does the data show a curve of simple form?
no
Has a satisfactory fit been obtained?
yes
END
yes
no
Choose x–values defined in Section 3.2.2.
yes
Can the user choose x–values of data points?
Try cubic splines
E02GCF
Have data points different weights?
yes
E02ADF
no
no
E02AFF
Has a satisfactory fit been obtained?
no
Does the user wish to use the l_2 norm?
yes
yes
END
no/don't know
E02BEF
Has a satisfactory fit been obtained?
no
E02BAF
yes
END

5 Index

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 18, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E02DBF

7 References

[1] Cox M G and Hayes J G (1973) Curve fitting: A guide and suite of algorithms for the non-specialist user *NPL Report NAC 26* National Physical Laboratory

[2] Hayes J G (1974) Numerical methods for curve and surface fitting *Bull. Inst. Math. Appl.* **10** 144–152

[3] Hayes J G (ed.) (1970) Curve fitting by polynomials in one variable *Numerical Approximation to Functions and Data* Athlone Press, London

[4] Hayes J G and Halliday J (1974) The least-squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

[5] Hayes J G (ed.) (1970) Fitting data in more than one variable *Numerical Approximation to Functions and Data* Athlone Press, London

[6] Baker G A (1975) *Essentials of Padé Approximants* Academic Press, New York

E02ACF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02ACF calculates a minimax polynomial fit to a set of data points.

2. Specification

```
SUBROUTINE E02ACF (X, Y, N, A, M1, REF)
INTEGER          N, M1
real             X(N), Y(N), A(M1), REF
```

3. Description

Given a set of data points (x_i,y_i), for $i = 1,2,...,n$, this routine uses the exchange algorithm to compute an mth order polynomial

$$P(x) = a_1 + a_2x + a_3x^2 + ... + a_{m+1}x^m$$

such that $\max_i 2|P(x_i)-y_i|$ is a minimum.

The routine also returns a number whose absolute value is the final reference deviation (see Section 6). The routine is an adaptation of Boothroyd [2].

4. References

[1] STIEFFEL, E.
Numerical Methods of Tchebycheff Approximation.
On Numerical Approximation, Langer R.E. (ed).
University of Wisconsin Press, pp. 217-232, 1959.

[2] BOOTHROYD, J.B.
Algorithm 318.
Comm. ACM, 10, pp. 801, 1967.

5. Parameters

1: X(N) – ***real*** array. *Input*

On entry: the values of the x co-ordinates, x_i, for $i = 1,2,...,n$.

Constraint: $x_i < x_2 < ... < x_n$.

2: Y(N) – ***real*** array. *Input*

On entry: the values of the y co-ordinates, y_i, for $i = 1,2,...,n$.

3: N – INTEGER. *Input*

On entry: the number n of data points.

4: A(M1) – ***real*** array. *Output*

On exit: the coefficients a_i of the final polynomial, for $i = 1,2,...,m+1$.

5: M1 – INTEGER. *Input*

On entry: $m + 1$, where m is the order of the polynomial to be found.

Constraint: M1 < min(N,100).

6: REF – ***real***. *Output*

On exit: the final reference deviation (see Section 6).

6. Error Indicators and Warnings

This routine calls P01ABF internally using the hard fail option to detect the following errors.

IFAIL = 1

M1 ≥ min(N,100).

IFAIL = 2

The constraint $x_1 < x_2 < ... < x_n$ is violated.

With exact arithmetic the algorithm should terminate after a finite number of steps. This need not necessarily be the case using computer arithmetic. Should the routine start cycling then an exit is made with REF given a negative value. This is by no means an indicator that a catastrophic error has occurred and does not preclude useful results being obtained.

The absolute value of REF is the final reference deviation. See Stieffel [1] of Section 4 for an explanation of this term.

7. Accuracy

This is wholly dependent on the given data points.

8. Further Comments

The time taken by this routine increases with m.

9. Example

The example program calculates a minimax fit with a polynomial of degree 5 to the exponential function evaluated at 21 points over the interval [0,1]. It then prints values of the function and the fitted polynomial.

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.

```
*     E02ACF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, M1
      PARAMETER        (N=21,M1=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      real             P1
      PARAMETER        (P1=0.1e0)
*     .. Local Scalars ..
      real             REF, S, T, Z
      INTEGER          I, J
*     .. Local Arrays ..
      real             A(M1), X(N), Y(N)
*     .. External Subroutines ..
      EXTERNAL         E02ACF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP, real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02ACF Example Program Results'
      DO 20 I = 1, N
         X(I) = real(I-1)/real(N-1)
         Y(I) = EXP(X(I))
   20 CONTINUE
*
```

```
      CALL E02ACF(X,Y,N,A,M1,REF)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   Polynomial coefficients'
      WRITE (NOUT,99998) (A(I),I=1,M1)
      WRITE (NOUT,*)
      WRITE (NOUT,99997) '   Reference deviation = ', REF
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  X     exp(X)    Fit     Residual'
      DO 60 J = 1, 11
         Z = real(J-1)*P1
         S = A(M1)
         DO 40 I = M1 - 1, 1, -1
            S = S*Z + A(I)
   40    CONTINUE
         T = EXP(Z)
         WRITE (NOUT,99999) Z, S, T, S - T
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,F5.2,2F9.4,e11.2)
99998 FORMAT (6X,e12.4)
99997 FORMAT (1X,A,e10.2)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E02ACF Example Program Results

    Polynomial coefficients
        0.1000E+01
        0.1000E+01
        0.4991E+00
        0.1704E+00
        0.3478E-01
        0.1391E-01

    Reference deviation =   0.11E-05

  X     exp(X)    Fit     Residual
 0.00   1.0000   1.0000  -0.11E-05
 0.10   1.1052   1.1052   0.97E-06
 0.20   1.2214   1.2214  -0.74E-06
 0.30   1.3499   1.3499  -0.92E-06
 0.40   1.4918   1.4918   0.30E-06
 0.50   1.6487   1.6487   0.11E-05
 0.60   1.8221   1.8221   0.46E-06
 0.70   2.0138   2.0138  -0.82E-06
 0.80   2.2255   2.2255  -0.84E-06
 0.90   2.4596   2.4596   0.88E-06
 1.00   2.7183   2.7183  -0.11E-05
```

E02ADF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02ADF computes weighted least-squares polynomial approximations to an arbitrary set of data points.

2. Specification

```
SUBROUTINE E02ADF (M, KPLUS1, NROWS, X, Y, W, WORK1, WORK2, A, S,
1                  IFAIL)
INTEGER          M, KPLUS1, NROWS, IFAIL
real             X(M), Y(M), W(M), WORK1(3*M), WORK2(2*KPLUS1),
1                A(NROWS,KPLUS1), S(KPLUS1)
```

3. Description

This routine determines least-squares polynomial approximations of degrees 0,1,...,k to the set of data points (x_r,y_r) with weights w_r, for $r = 1,2,...,m$.

The approximation of degree i has the property that it minimizes σ_i the sum of squares of the weighted residuals ε_r, where

$$\varepsilon_r = w_r(y_r - f_r)$$

and f_r is the value of the polynomial of degree i at the rth data point.

Each polynomial is represented in Chebyshev-series form with normalised argument $\bar{x}$. This argument lies in the range -1 to $+1$ and is related to the original variable x by the linear transformation

$$\bar{x} = \frac{(2x - x_{\max} - x_{\min})}{(x_{\max} - x_{\min})}.$$

Here $x_{\max}$ and $x_{\min}$ are respectively the largest and smallest values of x_r. The polynomial approximation of degree i is represented as

$$\tfrac{1}{2}a_{i+1,1}T_0(\bar{x}) + a_{i+1,2}T_1(\bar{x}) + a_{i+1,3}T_2(\bar{x}) + \ldots + a_{i+1,i+1}T_i(\bar{x}),$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with argument $(\bar{x})$.

For $i = 0,1,...,k$, the routine produces the values of $a_{i+1,j+1}$, for $j = 0,1,...,i$, together with the value of the root mean square residual $s_i = \sqrt{\frac{\sigma_i}{m-i-1}}$. In the case $m = i + 1$ the routine sets the value of s_i to zero.

The method employed is due to Forsythe [4] and is based upon the generation of a set of polynomials orthogonal with respect to summation over the normalised data set. The extensions due to Clenshaw [1] to represent these polynomials as well as the approximating polynomials in their Chebyshev-series forms are incorporated. The modifications suggested by Reinsch and Gentleman (see [5]) to the method originally employed by Clenshaw for evaluating the orthogonal polynomials from their Chebyshev-series representations are used to give greater numerical stability.

For further details of the algorithm and its use see Cox [2] and [3].

Subsequent evaluation of the Chebyshev-series representations of the polynomial approximations should be carried out using E02AEF.

4. References

[1] CLENSHAW, C.W.
Curve fitting with a digital computer.
Comput. J., 2, pp. 170-173, 1960.

[2] COX, M.G.
A data-fitting package for the non-specialist user.
Software for Numerical Mathematics, D.J. Evans, (ed).
Academic Press, London, 1974.

[3] COX, M.G. and HAYES, J.G.
Curve fitting: a guide and suite of algorithms for the non-specialist user.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[4] FORSYTHE, G.E.
Generation and use of orthogonal polynomials for data fitting with a digital computer.
J. Soc. Ind. Appl. Math., 5, pp. 74-88, 1957.

[5] GENTLEMEN, W.M.
An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients.
Comput. J., 12, pp. 160-165, 1969.

[6] HAYES, J.G.
Curve fitting by polynomials in one variable.
Numerical Approximation to Functions and Data, J.G. Hayes, (ed).
Athlone Press, London, 1970.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number m of data points.

Constraint: M $\geq$ MDIST $\geq$ 2, where MDIST is the number of distinct x values in the data.

2: KPLUS1 – INTEGER. *Input*

On entry: $k + 1$, where k is the maximum degree required.

Constraint: 0 < KPLUS1 $\leq$ MDIST, where MDIST is the number of distinct x values in the data.

3: NROWS – INTEGER. *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which E02ADF is called.

Constraint: NROWS $\geq$ KPLUS1.

4: X(M) – ***real*** array. *Input*

On entry: the values x_r of the independent variable, for $r = 1,2,...,m$.

Constraint: the values must be supplied in non-decreasing order with X(M) > X(1).

5: Y(M) – ***real*** array. *Input*

On entry: the values y_r of the dependent variable, for $r = 1,2,...,m$.

6: W(M) – ***real*** array. *Input*

On entry: the set of weights, w_r, for $r = 1,2,...,m$. For advice on the choice of weights, see Section 2.1.2 of the Chapter Introduction.

Constraint: W(r) > 0.0, for $r = 1,2,...m$.

7: WORK1(3*M) – ***real*** array. *Workspace*

8: WORK2(2*KPLUS1) – ***real*** array. *Workspace*

9: A(NROWS,KPLUS1) – ***real*** array. *Output*

On exit: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree i. A(i+1,j+1) contains the coefficient $a_{i+1,j+1}$, for $i = 0,1,...,k$; $j = 0,1,...,i$.

10: S(KPLUS1) – ***real*** array. *Output*

On exit: S(i+1) contains the root mean square residual s_i, for $i = 0,1,...,k$, as described in Section 3. For the interpretation of the values of the s_i and their use in selecting an appropriate degree, see Section 3.1 of the Chapter Introduction.

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The weights are not all strictly positive.

IFAIL = 2

The values of X(r), for $r = 1,2,...,$M are not in non-decreasing order.

IFAIL = 3

All X(r) have the same value: thus the normalisation of X is not possible.

IFAIL = 4

On entry, KPLUS1 < 1 (so the maximum degree required is negative)
or KPLUS1 > MDIST, where MDIST is the number of distinct x values in the data (so there cannot be a unique solution for degree k = KPLUS1 – 1).

IFAIL = 5

NROWS < KPLUS1.

7. Accuracy

No error analysis for the method has been published. Practical experience with the method, however, is generally extremely satisfactory.

8. Further Comments

The time taken by the routine is approximately proportional to $m(k+1)(k+11)$.

The approximating polynomials may exhibit undesirable oscillations (particularly near the ends of the range) if the maximum degree k exceeds a critical value which depends on the number of data points m and their relative positions. As a rough guide, for equally-spaced data, this critical value is about $2\times\sqrt{m}$. For further details see Hayes [6] page 60.

9. Example

Determine weighted least-squares polynomial approximations of degrees 0, 1, 2 and 3 to a set of 11 prescribed data points. For the approximation of degree 3, tabulate the data and the corresponding values of the approximating polynomial, together with the residual errors, and also the values of the approximating polynomial at points half-way between each pair of adjacent data points.

The example program supplied is written in a general form that will enable polynomial approximations of degrees 0,1,...,k to be obtained to m data points, with arbitrary positive weights, and the approximation of degree k to be tabulated. E02AEF is used to evaluate the approximating polynomial. The program is self-starting in that any number of data sets can be supplied.

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.

```
*     E02ADF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, KP1MAX, NROWS
      PARAMETER        (MMAX=200,KP1MAX=50,NROWS=KP1MAX)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FIT, X1, XARG, XCAPR, XM
      INTEGER          I, IFAIL, IWGHT, J, K, M, R
*     .. Local Arrays ..
      real             A(NROWS,KP1MAX), AK(KP1MAX), S(KP1MAX), W(MMAX),
     +                 WORK1(3*MMAX), WORK2(2*KP1MAX), X(MMAX), Y(MMAX)
*     .. External Subroutines ..
      EXTERNAL         E02ADF, E02AEF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02ADF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         READ (NIN,*) K, IWGHT
         IF (K+1.LE.KP1MAX) THEN
            DO 40 R = 1, M
               IF (IWGHT.NE.1) THEN
                  READ (NIN,*) X(R), Y(R), W(R)
               ELSE
                  READ (NIN,*) X(R), Y(R)
                  W(R) = 1.0e0
               END IF
   40       CONTINUE
            IFAIL = 0
*
            CALL E02ADF(M,K+1,NROWS,X,Y,W,WORK1,WORK2,A,S,IFAIL)
*
            DO 60 I = 0, K
               WRITE (NOUT,*)
               WRITE (NOUT,99998) 'Degree', I, '   R.M.S. residual =',
     +           S(I+1)
               WRITE (NOUT,*)
               WRITE (NOUT,*) '  J  Chebyshev coeff A(J)'
               WRITE (NOUT,99997) (J,A(I+1,J),J=1,I+1)
   60       CONTINUE
            DO 80 J = 1, K + 1
               AK(J) = A(K+1,J)
   80       CONTINUE
            X1 = X(1)
            XM = X(M)
            WRITE (NOUT,*)
            WRITE (NOUT,99996)
     +        'Polynomial approximation and residuals for degree', K
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +      '  R   Abscissa     Weight   Ordinate Polynomial  Residual'
            DO 100 R = 1, M
               XCAPR = ((X(R)-X1)-(XM-X(R)))/(XM-X1)
               IFAIL = 0
```

```
*
                CALL E02AEF(K+1,AK,XCAPR,FIT,IFAIL)
*
                WRITE (NOUT,99999) R, X(R), W(R), Y(R), FIT, FIT - Y(R)
                IF (R.LT.M) THEN
                   XARG = 0.5e0*(X(R)+X(R+1))
                   XCAPR = ((XARG-X1)-(XM-XARG))/(XM-X1)
*
                   CALL E02AEF(K+1,AK,XCAPR,FIT,IFAIL)
*
                   WRITE (NOUT,99995) XARG, FIT
                END IF
  100        CONTINUE
             GO TO 20
          END IF
       END IF
  120  STOP
*
99999 FORMAT (1X,I3,4F11.4,e11.2)
99998 FORMAT (1X,A,I4,A,e12.2)
99997 FORMAT (1X,I3,F15.4)
99996 FORMAT (1X,A,I4)
99995 FORMAT (4X,F11.4,22X,F11.4)
      END
```

9.2. Program Data

```
E02ADF Example Program Data
  11
   3   2
     1.00     10.40      1.00
     2.10      7.90      1.00
     3.10      4.70      1.00
     3.90      2.50      1.00
     4.90      1.20      1.00
     5.80      2.20      0.80
     6.50      5.10      0.80
     7.10      9.20      0.70
     7.80     16.10      0.50
     8.40     24.50      0.30
     9.00     35.30      0.20
```

9.3. Program Results

```
 E02ADF Example Program Results

 Degree   0   R.M.S. residual =    0.41E+01

   J  Chebyshev coeff A(J)
   1        12.1740

 Degree   1   R.M.S. residual =    0.43E+01

   J  Chebyshev coeff A(J)
   1        12.2954
   2         0.2740

 Degree   2   R.M.S. residual =    0.17E+01

   J  Chebyshev coeff A(J)
   1        20.7345
   2         6.2016
   3         8.1876
```

```
Degree   3   R.M.S. residual =     0.68E-01

  J  Chebyshev coeff A(J)
  1        24.1429
  2         9.4065
  3        10.8400
  4         3.0589

Polynomial approximation and residuals for degree   3

  R    Abscissa      Weight    Ordinate  Polynomial  Residual
  1      1.0000      1.0000     10.4000     10.4461   0.46E-01
         1.5500                             9.3106
  2      2.1000      1.0000      7.9000      7.7977  -0.10E+00
         2.6000                             6.2555
  3      3.1000      1.0000      4.7000      4.7025   0.25E-02
         3.5000                             3.5488
  4      3.9000      1.0000      2.5000      2.5533   0.53E-01
         4.4000                             1.6435
  5      4.9000      1.0000      1.2000      1.2390   0.39E-01
         5.3500                             1.4257
  6      5.8000      0.8000      2.2000      2.2425   0.42E-01
         6.1500                             3.3803
  7      6.5000      0.8000      5.1000      5.0116  -0.88E-01
         6.8000                             6.8400
  8      7.1000      0.7000      9.2000      9.0982  -0.10E+00
         7.4500                            12.3171
  9      7.8000      0.5000     16.1000     16.2123   0.11E+00
         8.1000                            20.1266
 10      8.4000      0.3000     24.5000     24.6048   0.10E+00
         8.7000                            29.6779
 11      9.0000      0.2000     35.3000     35.3769   0.77E-01
```

E02AEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AEF evaluates a polynomial from its Chebyshev-series representation.

2. Specification

```
SUBROUTINE E02AEF (NPLUS1, A, XCAP, P, IFAIL)
INTEGER          NPLUS1, IFAIL
real             A(NPLUS1), XCAP, P
```

3. Description

This routine evaluates the polynomial

$$\tfrac{1}{2}a_1T_0(\bar{x}) + a_2T_1(\bar{x}) + a_3T_2(\bar{x}) + \ldots + a_{n+1}T_n(\bar{x})$$

for any value of $\bar{x}$ satisfying $-1 \leq \bar{x} \leq 1$. Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. The value of n is prescribed by the user.

In practice, the variable $\bar{x}$ will usually have been obtained from an original variable x, where $x_{\min} \leq x \leq x_{\max}$ and

$$\bar{x} = \frac{((x-x_{\min})-(x_{\max}-x))}{(x_{\max}-x_{\min})}$$

Note that this form of the transformation should be used computationally rather than the mathematical equivalent

$$\bar{x} = \frac{(2x-x_{\min}-x_{\max})}{(x_{\max}-x_{\min})}$$

since the former guarantees that the computed value of $\bar{x}$ differs from its true value by at most 4ε, where ε is the ***machine precision***, whereas the latter has no such guarantee.

The method employed is based upon the three-term recurrence relation due to Clenshaw [1], with modifications to give greater numerical stability due to Reinsch and Gentleman (see [4]).

For further details of the algorithm and its use see Cox [2] and [3].

4. References

[1] CLENSHAW, C.W.
A note on the summation of Chebyshev series.
Maths. Tables Aids Comp., 9, pp. 118-120, 1955.

[2] COX, M.G.
A data-fitting package for the non-specialist user.
In: 'Software for Numerical Mathematics', D.J. Evans, (ed.).
Academic Press, London, 1974.

[3] COX, M.G. and HAYES, J.G.
Curve fitting: a guide and suite of algorithms for the non-specialist user.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[4] GENTLEMAN, W.M.
An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients.
Comput. J., 12, pp. 160-165, 1969.

5. Parameters

1: NPLUS1 – INTEGER. *Input*

On entry: the number $n + 1$ of terms in the series (i.e. one greater than the degree of the polynomial).

Constraint: NPLUS1 ≥ 1.

2: A(NPLUS1) – ***real*** array. *Input*

On entry: A(i) must be set to the value of the ith coefficient in the series, for $i = 1,2,...,n+1$.

3: XCAP – ***real***. *Input*

On entry: $\bar{x}$, the argument at which the polynomial is to be evaluated. It should lie in the range -1 to $+1$, but a value just outside this range is permitted (see Section 6) to allow for possible rounding errors committed in the transformation from x to $\bar{x}$ discussed in Section 3. Provided the recommended form of the transformation is used, a successful exit is thus assured whenever the value of x lies in the range $x_{\min}$ to $x_{\max}$.

4: P – ***real***. *Output*

On exit: the value of the polynomial.

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

ABS(XCAP) $> 1.0 + 4\varepsilon$, where ε is the ***machine precision***. In this case the value of P is set arbitrarily to zero.

IFAIL = 2

On entry, NPLUS1 < 1.

7. Accuracy

The rounding errors committed are such that the computed value of the polynomial is exact for a slightly perturbed set of coefficients $a_i + \delta a_i$. The ratio of the sum of the absolute values of the δa_i to the sum of the absolute values of the a_i is less than a small multiple of $(n+1)$ times ***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to $n + 1$.

It is expected that a common use of E02AEF will be the evaluation of the polynomial approximations produced by E02ADF and E02AFF.

9. Example

Evaluate at 11 equally-spaced points in the interval $-1 \leq \bar{x} \leq 1$ the polynomial of degree 4 with Chebyshev coefficients, 2.0, 0.5, 0.25, 0.125, 0.0625.

The example program is written in a general form that will enable a polynomial of degree n in its Chebyshev-series form to be evaluated at m equally-spaced points in the interval $-1 \leq \bar{x} \leq 1$. The program is self-starting in that any number of data sets can be supplied.

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.

```
*     E02AEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, NP1MAX
      PARAMETER        (NMAX=199,NP1MAX=NMAX+1)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             P, XCAP
      INTEGER          I, IFAIL, M, N, R
*     .. Local Arrays ..
      real             A(NP1MAX)
*     .. External Subroutines ..
      EXTERNAL         E02AEF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=60) M
      IF (M.GT.0) THEN
         READ (NIN,*) N
         IF (N.GE.0 .AND. N.LE.NMAX) THEN
            READ (NIN,*) (A(I),I=1,N+1)
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '  R       Argument       Value of polynomial'
            DO 40 R = 1, M
               XCAP = real(2*R-M-1)/real(M-1)
               IFAIL = 0
*
               CALL E02AEF(N+1,A,XCAP,P,IFAIL)
*
               WRITE (NOUT,99999) R, XCAP, P
   40       CONTINUE
            GO TO 20
         END IF
      END IF
   60 STOP
*
99999 FORMAT (1X,I3,F14.4,4X,F14.4)
      END
```

9.2. Program Data

```
E02AEF Example Program Data
  11
   4
     2.0000
     0.5000
     0.2500
     0.1250
     0.0625
```

9.3. Program Results

```
E02AEF Example Program Results

  R       Argument        Value of polynomial
  1       -1.0000              0.6875
  2       -0.8000              0.6613
  3       -0.6000              0.6943
  4       -0.4000              0.7433
  5       -0.2000              0.7843
  6        0.0000              0.8125
  7        0.2000              0.8423
  8        0.4000              0.9073
  9        0.6000              1.0603
 10        0.8000              1.3733
 11        1.0000              1.9375
```

E02AFF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AFF computes the coefficients of a polynomial, in its Chebyshev-series form, which interpolates (passes exactly through) data at a special set of points. Least-squares polynomial approximations can also be obtained.

2. Specification

```
SUBROUTINE E02AFF (NPLUS1, F, A, IFAIL)
INTEGER          NPLUS1, IFAIL
real             F(NPLUS1), A(NPLUS1)
```

3. Description

This routine computes the coefficients a_j, for $j = 1,2,...,n+1$, in the Chebyshev-series

$$\tfrac{1}{2}a_1T_0(\bar{x}) + a_2T_1(\bar{x}) + a_3T_2(\bar{x}) + ... + a_{n+1}T_n(\bar{x}),$$

which interpolates the data f_r at the points

$$\bar{x}_r = \cos\left(\frac{(r-1)\pi}{n}\right), \qquad r = 1,2,...,n+1$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. The use of these points minimizes the risk of unwanted fluctuations in the polynomial and is recommended when the data abscissae can be chosen by the user, e.g. when the data is given as a graph. For further advantages of this choice of points, see Clenshaw [2].

In terms of the user's original variables, x say, the values of x at which the data f_r are to be provided are

$$x_r = \tfrac{1}{2}(x_{\max}-x_{\min})\cos\left(\frac{\pi(r-1)}{n}\right) + \tfrac{1}{2}(x_{\max}+x_{\min}), \qquad r = 1,2,...,n+1$$

where $x_{\max}$ and $x_{\min}$ are respectively the upper and lower ends of the range of x over which the user wishes to interpolate.

Truncation of the resulting series after the term involving a_{i+1}, say, yields a least-squares approximation to the data. This approximation, $p(\bar{x})$, say, is the polynomial of degree i which minimizes

$$\tfrac{1}{2}\varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 + ... + \varepsilon_n^2 + \tfrac{1}{2}\varepsilon_{n+1}^2,$$

where the residual $\varepsilon_r = p(\bar{x}_r) - f_r$, for $r = 1,2,...n+1$.

The method employed is based on the application of the three-term recurrence relation due to Clenshaw [1] for the evaluation of the defining expression for the Chebyshev coefficients (see, for example, Clenshaw [2]). The modifications to this recurrence relation suggested by Reinsch and Gentleman (see [5]) are used to give greater numerical stability.

For further details of the algorithm and its use see Cox [3] and [4].

Subsequent evaluation of the computed polynomial, perhaps truncated after an appropriate number of terms, should be carried out using E02AEF.

4. References

[1] CLENSHAW, C.W.
A note on the summation of Chebyshev-series.
Math. Tables Aids Comp. 9, pp. 118-120, 1955.

[2] CLENSHAW, C.W.
Mathematical Tables, Vol. 5.
Chebyshev-series for mathematical functions.
HMSO, London, 1962.

[3] COX, M.G.
A Data-fitting Package for the Non-specialist User, In Software for Numerical Mathematics, D.J. Evans, (ed).
Academic Press, London, 1974.

[4] COX, M.G. and HAYES, J.G.
Curve fitting: a guide and suite of algorithms for the non-specialist user.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[5] GENTLEMAN, W.M.
An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients.
Comput. J., 12, pp. 160-165, 1969.

5. Parameters

1: NPLUS1 – INTEGER. *Input*

On entry: the number $n + 1$ of data points (one greater than the degree n of the interpolating polynomial).

Constraint: NPLUS1 ≥ 2.

2: F(NPLUS1) – ***real*** array. *Input*

On entry: for $r = 1,2,...,n+1$, F(r) must contain f_r the value of the dependent variable (ordinate) corresponding to the value

$$\bar{x}_r = \cos\left(\frac{\pi(r-1)}{n}\right)$$

of the independent variable (abscissa) $\bar{x}$, or equivalently to the value

$$x(r) = \tfrac{1}{2}(x_{\max}-x_{\min})\cos\left(\frac{\pi(r-1)}{n}\right) + \tfrac{1}{2}(x_{\max}+x_{\min})$$

of the user's original variable x. Here $x_{\max}$ and $x_{\min}$ are respectively the upper and lower ends of the range over which the user wishes to interpolate.

3: A(NPLUS1) – ***real*** array. *Output*

On exit: A(j) is the coefficient a_j in the interpolating polynomial, for $j = 1,2,...,n+1$.

4: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NPLUS1 < 2.

7. Accuracy

The rounding errors committed are such that the computed coefficients are exact for a slightly perturbed set of ordinates $f_r + \delta f_r$. The ratio of the sum of the absolute values of the δf_r to the sum of the absolute values of the f_r is less than a small multiple of $(n+1)\varepsilon$, where ε is the ***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to $(n+1)^2 + 30$.

For choice of degree when using the routine for least-squares approximation, see Section 3.2 of the Chapter Introduction.

9. Example

Determine the Chebyshev coefficients of the polynomial which interpolates the data $\bar{x}_r$, f_r, for $r = 1,2,...,11$, where $\bar{x}_r = \cos(\pi\times(r-1)/10)$ and $f_r = e^{\bar{x}_r}$. Evaluate, for comparison with the values of f_r, the resulting Chebyshev series at $\bar{x}_r$, for $r = 1,2,...,11$.

The example program supplied is written in a general form that will enable polynomial interpolations of arbitrary data at the cosine points $\cos(\pi\times(r-1)/n)$, for $r = 1,2,...,n+1$ to be obtained for any n (= NPLUS1−1). Note that E02AEF is used to evaluate the interpolating polynomial. The program is self-starting in that any number of data sets can be supplied.

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.

```
*     E02AFF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, NP1MAX
      PARAMETER        (NMAX=199,NP1MAX=NMAX+1)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FIT, PI, PIBY2N
      INTEGER          I, IFAIL, J, N, R
*     .. Local Arrays ..
      real             AN(NP1MAX), F(NP1MAX), XCAP(NP1MAX)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         E02AEF, E02AFF
*     .. Intrinsic Functions ..
      INTRINSIC        real, SIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AFF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      PI = X01AAF(PI)
   20 READ (NIN,*,END=80) N
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         PIBY2N = 0.5e0*PI/real(N)
         READ (NIN,*) (F(R),R=1,N+1)
         DO 40 R = 1, N + 1
            I = R - 1
*
*           The following method of evaluating  XCAP = cos(PI*I/N)
*           ensures that the computed value has a small relative error
*           and, moreover, is bounded in modulus by unity for all
*           I = 0, 1, ..., N.  (It is assumed that the sine routine
*           produces a result with a small relative error for values
*           of the argument between  -PI/4  and  PI/4).
*
            IF (4*I.LE.N) THEN
               XCAP(I+1) = 1.0e0 - 2.0e0*SIN(PIBY2N*I)**2
            ELSE IF (4*I.GT.3*N) THEN
               XCAP(I+1) = 2.0e0*SIN(PIBY2N*(N-I))**2 - 1.0e0
            ELSE
               XCAP(I+1) = SIN(PIBY2N*(N-2*I))
            END IF
```

```
   40     CONTINUE
          IFAIL = 0
*
          CALL E02AFF(N+1,F,AN,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,*) '          Chebyshev'
          WRITE (NOUT,*) '  J   coefficient A(J)'
          WRITE (NOUT,99998) (J,AN(J),J=1,N+1)
          WRITE (NOUT,*)
          WRITE (NOUT,*) '  R    Abscissa   Ordinate      Fit'
          DO 60 R = 1, N + 1
             IFAIL = 0
*
             CALL E02AEF(N+1,AN,XCAP(R),FIT,IFAIL)
*
             WRITE (NOUT,99999) R, XCAP(R), F(R), FIT
   60     CONTINUE
          GO TO 20
       END IF
   80  STOP
*
99999 FORMAT (1X,I3,3F11.4)
99998 FORMAT (1X,I3,F14.7)
      END
```

9.2. Program Data

```
E02AFF Example Program Data
  10
     2.7182
     2.5884
     2.2456
     1.7999
     1.3620
     1.0000
     0.7341
     0.5555
     0.4452
     0.3863
     0.3678
```

9.3. Program Results

```
 E02AFF Example Program Results

          Chebyshev
   J   coefficient A(J)
   1     2.5320000
   2     1.1303095
   3     0.2714893
   4     0.0443462
   5     0.0055004
   6     0.0005400
   7     0.0000307
   8    -0.0000006
   9    -0.0000004
  10     0.0000049
  11    -0.0000200
```

R	Abscissa	Ordinate	Fit
1	1.0000	2.7182	2.7182
2	0.9511	2.5884	2.5884
3	0.8090	2.2456	2.2456
4	0.5878	1.7999	1.7999
5	0.3090	1.3620	1.3620
6	0.0000	1.0000	1.0000
7	-0.3090	0.7341	0.7341
8	-0.5878	0.5555	0.5555
9	-0.8090	0.4452	0.4452
10	-0.9511	0.3863	0.3863
11	-1.0000	0.3678	0.3678

E02AGF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AGF computes constrained weighted least-squares polynomial approximations in Chebyshev-series form to an arbitrary set of data points. The values of the approximations and any number of their derivatives can be specified at selected points.

2. Specification

```
      SUBROUTINE E02AGF (M, KPLUS1, NROWS, XMIN, XMAX, X, Y, W, MF, XF,
     1                   YF, LYF, IP, A, S, NP1, WRK, LWRK, IWRK, LIWRK,
     2                   IFAIL)
      INTEGER          M, KPLUS1, NROWS, MF, LYF, IP(MF), NP1, LWRK,
     1                 IWRK(LIWRK), LIWRK, IFAIL
      real             XMIN, XMAX, X(M), Y(M), W(M), XF(MF), YF(LYF),
     1                 A(NROWS,KPLUS1), S(KPLUS1), WRK(LWRK)
```

3. Description

This routine determines least-squares polynomial approximations of degrees up to k to the set of data points (x_r,y_r) with weights w_r, for $r = 1,2,...,m$. The value of k, the maximum degree required, is prescribed by the user. At each of the values XF_r, for $r = 1,2,...,MF$, of the independent variable x, the approximations and their derivatives up to order p_r are constrained to have one of the user-specified values YF_s, for $s = 1,2,...,n$, where $n = MF + \sum_{r=1}^{MF} p_r$.

The approximation of degree i has the property that, subject to the imposed contraints, it minimizes Σ_i, the sum of the squares of the weighted residuals ε_r for $r = 1,2,...,m$ where

$$\varepsilon_r = w_r(y_r - f_i(x_r))$$

and $f_i(x_r)$ is the value of the polynomial approximation of degree i at the rth data point.

Each polynomial is represented in Chebyshev-series form with normalised argument $\bar{x}$. This argument lies in the range -1 to $+1$ and is related to the original variable x by the linear transformation

$$\bar{x} = \frac{2x - (x_{max}+x_{min})}{(x_{max}-x_{min})}$$

where x_{min} and x_{max}, specified by the user, are respectively the lower and upper endpoints of the interval of x over which the polynomials are to be defined.

The polynomial approximation of degree i can be written as

$$\tfrac{1}{2}a_{i,0} + a_{i,1}T_1(\bar{x}) + ... + a_{ij}T_j(\bar{x}) + ... + a_{ii}T_i(\bar{x})$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. For $i = n,n+1,...,k$, the routine produces the values of the coefficients a_{ij}, for $j = 0,1,...,i$, together with the value of the root mean square residual, S_i, defined as $\sqrt{\frac{\Sigma_i}{(m'+n-i-1)}}$, where m' is the number of data points with non-zero weight.

Values of the approximations may subsequently be computed using E02AEF or E02AKF.

First E02AGF determines a polynomial $\mu(\bar{x})$, of degree $n - 1$, which satisfies the given constraints, and a polynomial $\nu(\bar{x})$, of degree n, which has value (or derivative) zero wherever a constrained value (or derivative) is specified. It then fits $y_r - \mu(x_r)$, for $r = 1,2,...,m$ with polynomials of the required degree in $\bar{x}$ each with factor $\nu(\bar{x})$. Finally the coefficients of $\mu(\bar{x})$ are added to the coefficients of these fits to give the coefficients of the constrained polynomial

approximations to the data points (x_r,y_r), for $r = 1,2,...,m$. The method employed is given in Hayes [3]: it is an extension of Forsythe's orthogonal polynomials method [2] as modified by Clenshaw [1].

4. References

[1] CLENSHAW, C.W.
Curve fitting with a digital computer.
Comput. J., 2, pp. 170-173, 1960.

[2] FORSYTHE, G.E.
Generation and use of orthogonal polynomials for data fitting with a digital computer.
J. Soc. Ind. Appl. Maths., 5, pp. 74-88, 1957.

[3] HAYES, J.G.
Curve Fitting by Polynomials in One Variable.
Ch. 5 in Numerical Approximation to Functions and Data, Hayes J.G. (ed).
Univ. of London, The Athlone Press, 1970.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number m of data points to be fitted.

Constraint: M $\geq$ 1.

2: KPLUS1 – INTEGER. *Input*

On entry: $k + 1$, where k is the maximum degree required.

Constraint: $n + 1 \leq$ KPLUS1 $\leq m'' + n$, where n is the total number of constraints and m'' is the number of data points with non-zero weights and distinct abscissae which do not coincide with any of the XF(r).

3: NROWS – INTEGER. *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which E02AGF is called.

Constraint: NROWS $\geq$ KPLUS1.

4: XMIN – ***real***. *Input*

5: XMAX – ***real***. *Input*

On entry: the lower and upper endpoints, respectively, of the interval $[x_{\min},x_{\max}]$. Unless there are specific reasons to the contrary, it is recommended that XMIN and XMAX be set respectively to the lowest and highest value among the x_r and XF(r). This avoids the danger of extrapolation provided there is a constraint point or data point with non-zero weight at each endpoint.

Constraint: XMAX > XMIN.

6: X(M) – ***real*** array. *Input*

On entry: the value x_r of the independent variable at the rth data point, for $r = 1,2,...,m$.

Constraint: the X(r) must be in non-decreasing order and satisfy XMIN $\leq$ X(r) $\leq$ XMAX.

7: Y(M) – ***real*** array. *Input*

On entry: Y(r) must contain y_r, the value of the dependent variable at the rth data point, for $r = 1,2,...,m$.

8: W(M) – ***real*** array. *Input*

On entry: the weights w_r to be applied to the data points x_r, for $r = 1,2...,m$. For advice on the choice of weights see the Chapter Introduction. Negative weights are treated as positive. A zero weight causes the corresponding data point to be ignored. Zero weight should be given to any data point whose x and y values both coincide with those of a constraint (otherwise the denominators involved in the root-mean-square residuals s_i will be slightly in error).

9: MF – INTEGER. *Input*

On entry: the number of values of the independent variable at which a constraint is specified.

Constraint: MF $\geq$ 1.

10: XF(MF) – ***real*** array. *Input*

On entry: the rth value of the independent variable at which a constraint is specified, for $r = 1,2,...,$MF.

Constraint: these values need not be ordered but must be distinct and satisfy XMIN $\leq$ XF(r) $\leq$ XMAX.

11: YF(LYF) – ***real*** array. *Input*

On entry: the values which the approximating polynomials and their derivatives are required to take at the points specified in XF. For each value of XF(r), YF contains in successive elements the required value of the approximation, its first derivative, second derivative,..., p_rth derivative, for $r = 1,2,...,$MF. Thus the value which the kth derivative of each approximation ($k = 0$ referring to the approximation itself) is required to take at the point XF(r) must be contained in YF(s), where

$$s = r + k + p_1 + p_2 + \ldots + p_{r-1},$$

for $k = 0,1,...,p_r$ and $r = 1,2,...,$MF. The derivatives are with respect to the user's variable x.

12: LYF – INTEGER. *Input*

On entry: the dimension of the array YF as declared in the (sub)program from which E02AGF is called.

Constraint: LYF $\geq n$, where n = MF + $p_1 + p_2 + \ldots + p_{\mathrm{MF}}$.

13: IP(MF) – INTEGER array. *Input*

On entry: IP(r) must contain p_r, the order of the highest-order derivative specified at XF(r), for $r = 1,2,...,$MF. $p_r = 0$ implies that the value of the approximation at XF(r) is specified, but not that of any derivative.

Constraint: IP(r) $\geq$ 0, for $r = 1,2,...,$MF.

14: A(NROWS,KPLUS1) – ***real*** array. *Output*

On exit: A(i+1,j+1) contains the coefficient a_{ij} in the approximating polynomial of degree i, for $i = n,n+1,...,k$; $j = 0,1,...,i$.

15: S(KPLUS1) – ***real*** array. *Output*

On exit: S(i+1) contains s_i, for $i = n,n+1,...,k$, the root-mean-square residual corresponding to the approximating polynomial of degree i. In the case where the number of data points with non-zero weight is equal to $k + 1 - n$, s_i is indeterminate: the routine sets it to zero. For the interpretation of the values of s_i and their use in selecting an appropriate degree, see Section 3.1 of the Chapter Introduction.

16: NP1 – INTEGER. *Output*

On exit: $n + 1$, where n is the total number of constraint conditions imposed: $n = \text{MF} + p_1 + p_2 + \ldots + p_{\text{MF}}$.

17: WRK(LWRK) – ***real*** array. *Output*

On exit: WRK contains weighted residuals of the highest degree of fit determined (k). The residual at x_r is in element $2(n+1) + 3(m+k+1) + r$, for $r = 1,2,\ldots,m$. The rest of the array is used as workspace.

18: LWRK – INTEGER. *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E02AGF is called.

Constraint: LWRK $\geq$ max(4×M+3×KPLUS1, 8×n+5×IPMAX+MF+10) + 2×n + 2, where IPMAX = max(IP(R)).

19: IWRK(LIWRK) – INTEGER array. *Workspace*

20: LIWRK – INTEGER. *Input*

On entry: the dimension of the array IWRK as declared in the (sub)program from which E02AGF is called.

Constraint: LIWRK $\geq$ 2×MF + 2.

21: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, M < 1,
or KPLUS1 < n + 1,
or NROWS < KPLUS1,
or MF < 1,
or LYF < n,
or LWRK is too small (see Section 5),
or LIWRK < 2×MF + 2.

(Here n is the total number of constraint conditions.)

IFAIL = 2

IP(r) < 0 for some r = 1,2,...,MF.

IFAIL = 3

XMIN $\geq$ XMAX, or XF(r) is not in the interval XMIN to XMAX for some r = 1,2,...,MF, or the XF(r) are not distinct.

IFAIL = 4

X(r) is not in the interval XMIN to XMAX for some r = 1,2,...,M.

IFAIL = 5

X(r) < X(r−1) for some r = 2,3,...,M.

IFAIL = 6

KPLUS1 > m'' + n, where m'' is the number of data points with non-zero weight and distinct abscissae which do not coincide with any XF(r). Thus there is no unique solution.

IFAIL = 7

The polynomials $\mu(x)$ and/or $\nu(x)$ cannot be determined. The problem supplied is too ill-conditioned. This may occur when the constraint points are very close together, or large in number, or when an attempt is made to constrain high-order derivatives.

7. Accuracy

No complete error analysis exists for either the interpolating algorithm or the approximating algorithm. However, considerable experience with the approximating algorithm shows that it is generally extremely satisfactory. Also the moderate number of constraints, of low order, which are typical of data fitting applications, are unlikely to cause difficulty with the interpolating routine.

8. Further Comments

The time taken by the routine to form the interpolating polynomial is approximately proportional to n^3, and that to form the approximating polynomials is very approximately proportional to $m(k+1)(k+1-n)$.

To carry out a least-squares polynomial fit without constraints, use E02ADF. To carry out polynomial interpolation only, use E01AEF.

9. Example

The example program reads data in the following order, using the notation of the parameter list above:

MF
IP(i), XF(i), Y-value and derivative values (if any) at XF(i), for i = 1,2,...,MF
M
X(i), Y(i), W(i), for i = 1,2,...,M
k, XMIN, XMAX

The output is:

the root-mean-square residual for each degree from n to k;
the Chebyshev coefficients for the fit of degree k;
the data points, and the fitted values and residuals for the fit of degree k.

The program is written in a generalized form which will read any number of data sets.

The data set supplied specifies 5 data points in the interval [0.0,4.0] with unit weights, to which are to be fitted polynomials, p, of degrees up to 4, subject to the 3 constraints:

$p(0.0) = 1.0$, $p'(0.0) = -2.0$, $p(4.0) = 9.0$.

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.

```
*     E02AGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MFMAX, MMAX, KP1MAX, NROWS, LA, LIWRK, LYF, LWRK
      PARAMETER        (MFMAX=5,MMAX=20,KP1MAX=6,NROWS=KP1MAX,
     +                 LA=NROWS*KP1MAX,LIWRK=2*MFMAX+2,LYF=15,LWRK=200)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FITI, XMAX, XMIN
      INTEGER          I, IFAIL, IY, J, K, M, MF, NP1
```

```
*     .. Local Arrays ..
      real              A(NROWS,KP1MAX), S(KP1MAX), W(MMAX), WRK(LWRK),
     +                  X(MMAX), XF(MFMAX), Y(MMAX), YF(LYF)
      INTEGER           IP(MFMAX), IWRK(LIWRK)
*     .. External Subroutines ..
      EXTERNAL          E02AGF, E02AKF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AGF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) MF
      IF (MF.GT.0 .AND. MF.LE.MFMAX) THEN
         IY = 1
         DO 40 I = 1, MF
            READ (NIN,*) IP(I), XF(I), (YF(J),J=IY,IY+IP(I))
            IY = IY + IP(I) + 1
   40    CONTINUE
         READ (NIN,*) M
         IF (M.GT.0 .AND. M.LE.MMAX) THEN
            READ (NIN,*) (X(I),Y(I),W(I),I=1,M)
            READ (NIN,*) K, XMIN, XMAX
            IFAIL = 0
*
            CALL E02AGF(M,K+1,NROWS,XMIN,XMAX,X,Y,W,MF,XF,YF,LYF,IP,A,S,
     +                  NP1,WRK,LWRK,IWRK,LIWRK,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Degree  RMS residual'
            WRITE (NOUT,99999) (I,S(I+1),I=NP1-1,K)
            WRITE (NOUT,*)
            WRITE (NOUT,99996) 'Details of the fit of degree ', K
            WRITE (NOUT,*)
            WRITE (NOUT,*) '  Index   Coefficient'
            DO 60 I = 1, K + 1
               WRITE (NOUT,99997) I - 1, A(K+1,I)
   60       CONTINUE
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '     I       X(I)        Y(I)        Fit      Residual'
            DO 80 I = 1, M
*
               CALL E02AKF(K+1,XMIN,XMAX,A(K+1,1),NROWS,LA-K,X(I),FITI,
     +                     IFAIL)
*
               WRITE (NOUT,99998) I, X(I), Y(I), FITI, FITI - Y(I)
   80       CONTINUE
            GO TO 20
         END IF
      END IF
  100 STOP
*
99999 FORMAT (1X,I4,1P,e15.2)
99998 FORMAT (1X,I6,3F11.4,e11.2)
99997 FORMAT (1X,I6,F11.4)
99996 FORMAT (1X,A,I2)
      END
```

9.2. Program Data

```
E02AGF Example Program Data
    2
    1       0.0       1.0      -2.0
    0       4.0       9.0
    5
       0.5      0.03       1.0
       1.0     -0.75       1.0
       2.0     -1.0        1.0
       2.5     -0.1        1.0
       3.0      1.75       1.0
    4       0.0       4.0
```

9.3. Program Results

```
E02AGF Example Program Results

Degree  RMS residual
   3       2.55E-03
   4       2.94E-03

Details of the fit of degree  4

  Index   Coefficient
     0      3.9980
     1      3.4995
     2      3.0010
     3      0.5005
     4      0.0000

     I       X(I)        Y(I)        Fit      Residual
     1      0.5000      0.0300      0.0310    0.10E-02
     2      1.0000     -0.7500     -0.7508   -0.78E-03
     3      2.0000     -1.0000     -1.0020   -0.20E-02
     4      2.5000     -0.1000     -0.0961    0.39E-02
     5      3.0000      1.7500      1.7478   -0.22E-02
```

E02AHF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AHF determines the coefficients in the Chebyshev-series representation of the derivative of a polynomial given in Chebyshev-series form.

2. Specification

```
SUBROUTINE E02AHF (NP1, XMIN, XMAX, A, IA1, LA, PATM1, ADIF, IADIF1,
1                  LADIF, IFAIL)
INTEGER          NP1, IA1, LA, IADIF1, LADIF, IFAIL
real             XMIN, XMAX, A(LA), PATM1, ADIF(LADIF)
```

3. Description

This routine forms the polynomial which is the derivative of a given polynomial. Both the original polynomial and its derivative are represented in Chebyshev-series form. Given the coefficients a_i, for $i = 0,1,...,n$, of a polynomial $p(x)$ of degree n, where

$$p(x) = \tfrac{1}{2}a_0 + a_1T_1(\bar{x}) + ... + a_nT_n(\bar{x})$$

the routine returns the coefficients $\bar{a}_i$, for $i = 0,1,...,n-1$, of the polynomial $q(x)$ of degree $n-1$, where

$$q(x) = \frac{dp(x)}{dx} = \tfrac{1}{2}\bar{a}_0 + \bar{a}_1T_1(\bar{x}) + ... + \bar{a}_{n-1}T_{n-1}(\bar{x}).$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. It is assumed that the normalised variable $\bar{x}$ in the interval $[-1,+1]$ was obtained from the user's original variable x in the interval $[x_{\min},x_{\max}]$ by the linear transformation

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{x_{\max} - x_{\min}}$$

and that the user requires the derivative to be with respect to the variable x. If the derivative with respect to $\bar{x}$ is required, set $x_{\max} = 1$ and $x_{\min} = -1$.

Values of the derivative can subsequently be computed, from the coefficients obtained, by using E02AKF.

The method employed is that of [1] modified to obtain the derivative with respect to x. Initially setting $\bar{a}_{n+1} = \bar{a}_n = 0$, the routine forms successively

$$\bar{a}_{i-1} = \bar{a}_{i+1} + \frac{2}{x_{\max} - x_{\min}}2ia_i, \qquad i = n,n-1,...,1.$$

4. References

[1] Chebyshev-series.
Modern Computing Methods, Chapter 8.
NPL Notes on Applied Science, 16 (2nd Edition) HMSO, 1961.

5. Parameters

1: NP1 – INTEGER. *Input*

On entry: $n + 1$, where n is the degree of the given polynomial $p(x)$. Thus NP1 is the number of coefficients in this polynomial.

Constraint: NP1 ≥ 1.

2: XMIN – ***real***. *Input*

3: XMAX – ***real***. *Input*

On entry: the lower and upper endpoints respectively of the interval $[x_{\min},x_{\max}]$. The Chebyshev-series representation is in terms of the normalised variable $\bar{x}$, where

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{x_{\max} - x_{\min}}.$$

Constraint: XMAX > XMIN.

4: A(LA) – ***real*** array. *Input*

On entry: the Chebyshev coefficients of the polynomial $p(x)$. Specifically, element 1 + i×IA1 of A must contain the coefficient a_i, for i = 0,1,...,n. Only these n + 1 elements will be accessed.

Unchanged on exit, but see ADIF, below.

5: IA1 – INTEGER. *Input*

On entry: the index increment of A. Most frequently the Chebyshev coefficients are stored in adjacent elements of A, and IA1 must be set to 1. However, if, for example, they are stored in A(1),A(4),A(7),..., then the value of IA1 must be 3. See also Section 8.

Constraint: IA1 ≥ 1.

6: LA – INTEGER. *Input*

On entry: the dimension of the array A as declared in the (sub)program from which E02AHF is called.

Constraint: LA ≥ 1 + (NP1−1)×IA1.

7: PATM1 – ***real***. *Output*

On exit: the value of $p(x_{\min})$. If this value is passed to the integration routine E02AJF with the coefficients of $q(x)$, then the original polynomial $p(x)$ is recovered, including its constant coefficient.

8: ADIF(LADIF) – ***real*** array. *Output*

On exit: the Chebyshev coefficients of the derived polynomial $q(x)$. (The differentiation is with respect to the variable x). Specifically, element 1 + i×IADIF1 of ADIF contains the coefficient $\bar{a}_i$, i = 0,1,...n−1. Additionally element 1 + n×IADIF1 is set to zero. A call of the routine may have the array name ADIF the same as A, provided that note is taken of the order in which elements are overwritten, when choosing the starting elements and increments IA1 and IADIF1: i.e. the coefficients $a_0,a_1,...,a_{i-1}$ must be intact after coefficient $\bar{a}_i$ is stored. In particular, it is possible to overwrite the a_i completely by having IA1 = IADIF1, and the actual arrays for A and ADIF identical.

9: IADIF1 – INTEGER. *Input*

On entry: the index increment of ADIF. Most frequently the Chebyshev coefficients are required in adjacent elements of ADIF, and IADIF1 must be set to 1. However, if, for example, they are to be stored in ADIF(1),ADIF(4),ADIF(7),..., then the value of IADIF1 must be 3. See Section 8.

Constraint: IADIF1 ≥ 1.

10: LADIF – INTEGER. *Input*

On entry: the dimension of the array ADIF as declared in the (sub)program from which E02AHF is called.

Constraint: LADIF ≥ 1 + (NP1−1)×IADIF1.

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NP1 < 1,
or XMAX ≤ XMIN,
or IA1 < 1,
or LA ≤ (NP1−1)×IA1,
or IADIF1 < 1,
or LADIF ≤ (NP1−1)×IADIF1.

7. Accuracy

There is always a loss of precision in numerical differentiation, in this case associated with the multiplication by $2i$ in the formula quoted in Section 3.

8. Further Comments

The time taken by the routine is approximately proportional to $n + 1$.

The increments IA1, IADIF1 are included as parameters to give a degree of flexibility which, for example, allows a polynomial in two variables to be differentiated with respect to either variable without rearranging the coefficients.

9. Example

Suppose a polynomial has been computed in Chebyshev-series form to fit data over the interval [–0.5,2.5]. The following program evaluates the 1st and 2nd derivatives of this polynomial at 4 equally spaced points over the interval. (For the purposes of this example, XMIN, XMAX and the Chebyshev coefficients are simply supplied in DATA statements. Normally a program would first read in or generate data and compute the fitted polynomial.)

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.

```
*     E02AHF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NP1, LA, LADIF
      PARAMETER        (NP1=7,LA=NP1,LADIF=NP1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             DERIV, DERIV2, PATM1, X, XMAX, XMIN
      INTEGER          I, IFAIL, M
*     .. Local Arrays ..
      real             A(LA), ADIF(LADIF), ADIF2(LADIF)
*     .. External Subroutines ..
      EXTERNAL         E02AHF, E02AKF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Data statements ..
      DATA             XMIN, XMAX/-0.5e0, 2.5e0/
      DATA             (A(I),I=1,NP1)/2.53213e0, 1.13032e0, 0.27150e0,
     +                 0.04434e0, 0.00547e0, 0.00054e0, 0.00004e0/
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AHF Example Program Results'
      IFAIL = 0
*
      CALL E02AHF(NP1,XMIN,XMAX,A,1,LA,PATM1,ADIF,1,LADIF,IFAIL)
      CALL E02AHF(NP1-1,XMIN,XMAX,ADIF,1,LADIF,PATM1,ADIF2,1,LADIF,
     +            IFAIL)
*
      M = 4
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    I  Argument    1st deriv    2nd deriv'
      DO 20 I = 1, M
         X = (XMIN*real(M-I)+XMAX*real(I-1))/real(M-1)
*
         CALL E02AKF(NP1-1,XMIN,XMAX,ADIF,1,LADIF,X,DERIV,IFAIL)
         CALL E02AKF(NP1-2,XMIN,XMAX,ADIF2,1,LADIF,X,DERIV2,IFAIL)
*
         WRITE (NOUT,99999) I, X, DERIV, DERIV2
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,I4,F9.4,2(4X,F9.4))
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E02AHF Example Program Results

    I  Argument    1st deriv    2nd deriv
    1  -0.5000       0.2453       0.1637
    2   0.5000       0.4777       0.3185
    3   1.5000       0.9304       0.6203
    4   2.5000       1.8119       1.2056
```

E02AJF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AJF determines the coefficients in the Chebyshev-series representation of the indefinite integral of a polynomial given in Chebyshev-series form.

2. Specification

```
SUBROUTINE E02AJF (NP1, XMIN, XMAX, A, IA1, LA, QATM1, AINT, IAINT1,
1                  LAINT, IFAIL)
INTEGER          NP1, IA1, LA, IAINT1, LAINT, IFAIL
real             XMIN, XMAX, A(LA), QATM1, AINT(LAINT)
```

3. Description

This routine forms the polynomial which is the indefinite integral of a given polynomial. Both the original polynomial and its integral are represented in Chebyshev-series form. If supplied with the coefficients a_i, for $i = 0,1,...,n$, of a polynomial $p(x)$ of degree n, where

$$p(x) = \tfrac{1}{2}a_0 + a_1T_1(\bar{x}) + ... + a_nT_n(\bar{x}),$$

the routine returns the coefficients a'_i, for $i = 0,1,...,n+1$, of the polynomial $q(x)$ of degree $n + 1$, where

$$q(x) = \tfrac{1}{2}a'_0 + a'_1T_1(\bar{x}) + ... + a'_{n+1}T_{n+1}(\bar{x}),$$

and

$$q(x) = \int p(x)dx.$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. It is assumed that the normalised variable $\bar{x}$ in the interval $[-1,+1]$ was obtained from the user's original variable x in the interval $[x_{\min},x_{\max}]$ by the linear transformation

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{x_{\max} - x_{\min}}$$

and that the user requires the integral to be with respect to the variable x. If the integral with respect to $\bar{x}$ is required, set $x_{\max} = 1$ and $x_{\min} = -1$.

Values of the integral can subsequently be computed, from the coefficients obtained, by using E02AKF.

The method employed is that of Chebyshev-series [1] modified for integrating with respect to x. Initially taking $a_{n+1} = a_{n+2} = 0$, the routine forms successively

$$a'_i = \frac{a_{i-1} - a_{i+1}}{2i} \times \frac{x_{\max} - x_{\min}}{2}, \qquad i = n+1,n,...,1.$$

The constant coefficient a'_0 is chosen so that $q(x)$ is equal to a specified value, QATM1, at the lower endpoint of the interval on which it is defined, i.e. $\bar{x} = -1$, which corresponds to $x = x_{\min}$.

4. References

[1] Chebyshev-series.
Modern Computing Methods, Ch. 8.
NPL Notes on Applied Science, No. 16, (2nd Edition), HMSO, 1961.

5. Parameters

1: NP1 – INTEGER. *Input*

On entry: $n + 1$, where n is the degree of the given polynomial $p(x)$. Thus NP1 is the number of coefficients in this polynomial.

Constraint: NP1 ≥ 1.

2: XMIN – ***real***. *Input*

3: XMAX – ***real***. *Input*

On entry: the lower and upper endpoints respectively of the interval $[x_{\min},x_{\max}]$. The Chebyshev-series representation is in terms of the normalised variable $\bar{x}$, where

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{x_{\max} - x_{\min}}.$$

Constraint: XMAX > XMIN.

4: A(LA) – ***real*** array. *Input*

On entry: the Chebyshev coefficients of the polynomial $p(x)$. Specifically, element $1 + i\times$IA1 of A must contain the coefficient a_i, for $i = 0,1,...,n$. Only these $n + 1$ elements will be accessed.

Unchanged on exit, but see AINT, below.

5: IA1 – INTEGER. *Input*

On entry: the index increment of A. Most frequently the Chebyshev coefficients are stored in adjacent elements of A, and IA1 must be set to 1. However, if for example, they are stored in A(1),A(4),A(7),..., then the value of IA1 must be 3. See also Section 8.

Constraint: IA1 ≥ 1.

6: LA – INTEGER. *Input*

On entry: the dimension of the array A as declared in the (sub)program from which E02AJF is called.

Constraint: LA $\geq 1 + (\text{NP1}-1)\times\text{IA1}$.

7: QATM1 – ***real***. *Input*

On entry: the value that the integrated polynomial is required to have at the lower endpoint of its interval of definition, i.e. at $\bar{x} = -1$ which corresponds to $x = x_{\min}$. Thus, QATM1 is a constant of integration and will normally be set to zero by the user.

8: AINT(LAINT) – ***real*** array. *Output*

On exit: the Chebyshev coefficients of the integral $q(x)$. (The integration is with respect to the variable x, and the constant coefficient is chosen so that $q(x_{\min})$ equals QATM1). Specifically, element $1 + i\times$IAINT1 of AINT contains the coefficient a'_i, for $i = 0,1,...,n+1$. A call of the routine may have the array name AINT the same as A, provided that note is taken of the order in which elements are overwritten when choosing starting elements and increments IA1 and IAINT1: i.e. the coefficients, $a_0,a_1,...,a_{i-2}$ must be intact after coefficient a'_i is stored. In particular it is possible to overwrite the a_i entirely by having IA1 = IAINT1, and the actual array for A and AINT identical.

9: IAINT1 – INTEGER. *Input*

On entry: the index increment of AINT. Most frequently the Chebyshev coefficients are required in adjacent elements of AINT, and IAINT1 must be set to 1. However, if, for example, they are to be stored in AINT(1),AINT(4),AINT(7),..., then the value of IAINT1 must be 3. See also Section 8.

Constraint: IAINT1 ≥ 1.

10: LAINT – INTEGER. *Input*

On entry: the dimension of the array AINT as declared in the (sub)program from which E02AJF is called.

Constraint: LAINT ≥ 1 + NP1×IAINT1.

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NP1 < 1,
or XMAX ≤ XMIN,
or IA1 < 1,
or LA ≤ (NP1−1)×IA1,
or IAINT1 < 1,
or LAINT ≤ NP1×IAINT1.

7. Accuracy

In general there is a gain in precision in numerical integration, in this case associated with the division by $2i$ in the formula quoted in Section 3.

8. Further Comments

The time taken by the routine is approximately proportional to $n + 1$.

The increments IA1, IAINT1 are included as parameters to give a degree of flexibility which, for example, allows a polynomial in two variables to be integrated with respect to either variable without rearranging the coefficients.

9. Example

Suppose a polynomial has been computed in Chebyshev-series form to fit data over the interval [–0.5,2.5]. The following program evaluates the integral of the polynomial from 0.0 to 2.0. (For the purpose of this example, XMIN, XMAX and the Chebyshev coefficients are simply supplied in DATA statements. Normally a program would read in or generate data and compute the fitted polynomial).

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.

```
*     E02AJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NP1, LA, LAINT
      PARAMETER        (NP1=7,LA=NP1,LAINT=NP1+1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             RA, RB, RESULT, XA, XB, XMAX, XMIN
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             A(LA), AINT(LAINT)
*     .. External Subroutines ..
      EXTERNAL         E02AJF, E02AKF
```

```
*     .. Data statements ..
      DATA              XMIN, XMAX/-0.5e0, 2.5e0/
      DATA              (A(I),I=1,NP1)/2.53213e0, 1.13032e0, 0.27150e0,
     +                  0.04434e0, 0.00547e0, 0.00054e0, 0.00004e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AJF Example Program Results'
      IFAIL = 0
*
      CALL E02AJF(NP1,XMIN,XMAX,A,1,LA,0.0e0,AINT,1,LAINT,IFAIL)
*
      XA = 0.0e0
      XB = 2.0e0
*
      CALL E02AKF(NP1+1,XMIN,XMAX,AINT,1,LAINT,XA,RA,IFAIL)
      CALL E02AKF(NP1+1,XMIN,XMAX,AINT,1,LAINT,XB,RB,IFAIL)
*
      RESULT = RB - RA
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Value of definite integral is ', RESULT
      STOP
*
99999 FORMAT (1X,A,F10.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E02AJF Example Program Results

Value of definite integral is     2.1515
```

E02AKF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02AKF evaluates a polynomial from its Chebyshev-series representation, allowing an arbitrary index increment for accessing the array of coefficients.

2. Specification

```
SUBROUTINE E02AKF (NP1, XMIN, XMAX, A, IA1, LA, X, RESULT, IFAIL)
INTEGER          NP1, IA1, LA, IFAIL
real             XMIN, XMAX, A(LA), X, RESULT
```

3. Description

If supplied with the coefficients a_i, for $i = 0,1,...,n$, of a polynomial $p(\bar{x})$ of degree n, where

$$p(\bar{x}) = \tfrac{1}{2}a_0 + a_1T_1(\bar{x}) + ... + a_nT_n(\bar{x}),$$

this routine returns the value of $p(\bar{x})$ at a user-specified value of the variable x. Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. It is assumed that the independent variable $\bar{x}$ in the interval $[-1,+1]$ was obtained from the user's original variable x in the interval $[x_{min},x_{max}]$ by the linear transformation

$$\bar{x} = \frac{2x - (x_{max}+x_{min})}{x_{max} - x_{min}}.$$

The coefficients a_i may be supplied in the array A, with any increment between the indices of array elements which contain successive coefficients. This enables the routine to be used in surface fitting and other applications, in which the array might have two or more dimensions.

The method employed is based upon the three-term recurrence relation due to Clenshaw [1], with modifications due to Reinsch and Gentleman (see [4]). For further details of the algorithm and its use see Cox [2] and Cox and Hayes [3].

4. References

[1] CLENSHAW, C.W.
A note on the summation of Chebyshev-series.
Maths. Tables Aids Comp., 9, pp. 118-120, 1955.

[2] COX, M.G.
A data-fitting package for the non-specialist user.
Report NAC40, National Physical Laboratory, Teddington, Middlesex, 1973.

[3] COX, M.G. and HAYES, J.G.
Curve Fitting: a guide and suite of algorithms for the non-specialist user.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[4] GENTLEMAN, W.M.
An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients.
Comput. J., 12, pp. 160-165, 1969.

5. Parameters

1: NP1 – INTEGER. *Input*

On entry: $n + 1$, where n is the degree of the given polynomial $p(\bar{x})$.

Constraint: NP1 $\geq$ 1.

2: XMIN – ***real.*** *Input*

3: XMAX – ***real.*** *Input*

On entry: the lower and upper endpoints respectively of the interval $[x_{\min},x_{\max}]$. The Chebyshev-series representation is in terms of the normalised variable $\bar{x}$, where

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{x_{\max} - x_{\min}}.$$

Constraint: XMIN < XMAX.

4: A(LA) – ***real*** array. *Input*

On entry: the Chebyshev coefficients of the polynomial $p(\bar{x})$. Specifically, element $1 + i\times$IA1 must contain the coefficient a_i, for $i = 0,1,...,n$. Only these $n + 1$ elements will be accessed.

5: IA1 – INTEGER. *Input*

On entry: the index increment of A. Most frequently, the Chebyshev coefficients are stored in adjacent elements of A, and IA1 must be set to 1. However, if, for example, they are stored in A(1),A(4),A(7),..., then the value of IA1 must be 3.

Constraint: IA1 ≥ 1.

6: LA – INTEGER. *Input*

On entry: the dimension of the array A as declared in the (sub)program from which E02AKF is called.

Constraint: LA ≥ (NP1−1)×IA1 + 1.

7: X – ***real.*** *Input*

On entry: the argument x at which the polynomial is to be evaluated.

Constraint: XMIN ≤ X ≤ XMAX.

8: RESULT – ***real.*** *Output*

On exit: the value of the polynomial $p(\bar{x})$.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NP1 < 1,
or IA1 < 1,
or LA ≤ (NP1−1)×IA1,
or XMIN ≥ XMAX.

IFAIL = 2

X does not satisfy the restriction XMIN ≤ X ≤ XMAX.

7. Accuracy

The rounding errors are such that the computed value of the polynomial is exact for a slightly perturbed set of coefficients $a_i + \delta a_i$. The ratio of the sum of the absolute values of the δa_i to the sum of the absolute values of the a_i is less than a small multiple of $(n+1)\times$***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to $n + 1$.

9. Example

Suppose a polynomial has been computed in Chebyshev-series form to fit data over the interval [–0.5,2.5]. The following program evaluates the polynomial at 4 equally spaced points over the interval. (For the purposes of this example, XMIN, XMAX and the Chebyshev coefficients are supplied in DATA statements. Normally a program would first read in or generate data and compute the fitted polynomial.)

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.

```
*     E02AKF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NP1, LA
      PARAMETER        (NP1=7,LA=NP1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             P, X, XMAX, XMIN
      INTEGER          I, IFAIL, M
*     .. Local Arrays ..
      real             A(LA)
*     .. External Subroutines ..
      EXTERNAL         E02AKF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Data statements ..
      DATA             XMIN, XMAX/-0.5e0, 2.5e0/
      DATA             (A(I),I=1,NP1)/2.53213e0, 1.13032e0, 0.27150e0,
     +                 0.04434e0, 0.00547e0, 0.00054e0, 0.00004e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02AKF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   I   Argument  Value of polynomial'
      M = 4
      DO 20 I = 1, M
         X = (XMIN*real(M-I)+XMAX*real(I-1))/real(M-1)
         IFAIL = 0
*
         CALL E02AKF(NP1,XMIN,XMAX,A,1,LA,X,P,IFAIL)
*
         WRITE (NOUT,99999) I, X, P
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,I4,F10.4,4X,F9.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E02AKF Example Program Results

   I    Argument  Value of polynomial
   1    -0.5000        0.3679
   2     0.5000        0.7165
   3     1.5000        1.3956
   4     2.5000        2.7183
```

E02BAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02BAF computes a weighted least-squares approximation to an arbitrary set of data points by a cubic spline with knots prescribed by the user. Cubic spline interpolation can also be carried out.

2. Specification

```
SUBROUTINE E02BAF (M, NCAP7, X, Y, W, LAMDA, WORK1, WORK2, C, SS, IFAIL)
INTEGER          M, NCAP7, IFAIL
real             X(M), Y(M), W(M), LAMDA(NCAP7), WORK1(M),
1                WORK2(4*NCAP7), C(NCAP7), SS
```

3. Description

This routine determines a least-squares cubic spline approximation $s(x)$ to the set of data points (x_r,y_r) with weights w_r, for $r = 1,2,...,m$. The value of NCAP7 $= \bar{n}+7$, where $\bar{n}$ is the number of intervals of the spline (one greater than the number of interior knots), and the values of the knots $\lambda_5,\lambda_6,...,\lambda_{\bar{n}+3}$, interior to the data interval, are prescribed by the user.

$s(x)$ has the property that it minimizes θ, the sum of squares of the weighted residuals ϵ_r, for $r = 1,2,...,m$, where

$$\epsilon_r = w_r(y_r - s(x_r)).$$

The routine produces this minimizing value of θ and the coefficients $c_1,c_2,...,c_q$, where $q = \bar{n}+3$, in the B-spline representation

$$s(x) = \sum_{i=1}^{q} c_i N_i(x).$$

Here $N_i(x)$ denotes the normalised B-spline of degree 3 defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$.

In order to define the full set of B-splines required, eight additional knots $\lambda_1,\lambda_2,\lambda_3,\lambda_4$ and $\lambda_{\bar{n}+4},\lambda_{\bar{n}+5},\lambda_{\bar{n}+6},\lambda_{\bar{n}+7}$ are inserted automatically by the routine. The first four of these are set equal to the smallest x_r and the last four to the largest x_r.

The representation of $s(x)$ in terms of B-splines is the most compact form possible in that only $\bar{n}+3$ coefficients, in addition to the $\bar{n}+7$ knots, fully define $s(x)$.

The method employed involves forming and then computing the least-squares solution of a set of m linear equations in the coefficients c_i $(i = 1,2,...,\bar{n}+3)$. The equations are formed using a recurrence relation for B-splines that is unconditionally stable (Cox [1], de Boor [5]), even for multiple (coincident) knots. The least-squares solution is also obtained in a stable manner by using orthogonal transformations, viz. a variant of Givens rotations (Gentleman [6] and [7]). This requires only one equation to be stored at a time. Full advantage is taken of the structure of the equations, there being at most four non-zero values of $N_i(x)$ for any value of x and hence at most four coefficients in each equation.

For further details of the algorithm and its use see Cox [2], [3] and [4].

Subsequent evaluation of $s(x)$ from its B-spline representation may be carried out using E02BBF. If derivatives of $s(x)$ are also required, E02BCF may be used. E02BDF can be used to compute the definite integral of $s(x)$.

4. References

[1] COX, M.G.
The Numerical Evaluation of B-splines.
J. Inst. Math. Appl., 10, pp. 134-149, 1972.

[2] COX, M.G.
A Data-fitting Package for the Non-specialist User.
In: Software for Numerical Mathematics, D.J. Evans, (Ed.).
Academic Press, London, 1974.

[3] COX, M.G.
Numerical methods for the interpolation and approximation of data by spline functions.
PhD Thesis, City University, London, 1975.

[4] COX, M.G. and HAYES, J.G.
Curve Fitting: A Guide and Suite of Algorithms for the Non-specialist User.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[5] DE BOOR, C.
On Calculating with B-splines.
J. Approx. Theory, 6, pp. 50-62, 1972.

[6] GENTLEMAN, W.M.
Algorithm AS 75.
Basic Procedures for Large Sparse or Weighted Linear Least-squares Problems.
Appl. Statist., 23, pp. 448-454, 1974.

[7] GENTLEMAN, W.M.
Least-squares Computations by Givens Transformations without Square Roots.
J. Inst. Math. Applic., 12, pp. 329-336, 1973.

[8] SCHOENBERG, I.J. and WHITNEY, A.
On Polya Frequency Functions III.
Trans. Amer. Math. Soc., 74, pp. 246-259, 1953.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number m of data points.

Constraint: M $\geq$ MDIST $\geq$ 4, where MDIST is the number of distinct x values in the data.

2: NCAP7 – INTEGER. *Input*

On entry: $\bar{n}+7$, where $\bar{n}$ is the number of intervals of the spline (which is one greater than the number of interior knots, i.e. the knots strictly within the range x_1 to x_m) over which the spline is defined.

Constraint: 8 $\leq$ NCAP7 $\leq$ MDIST + 4, where MDIST is the number of distinct x values in the data.

3: X(M) – ***real*** array. *Input*

On entry: the values x_r of the independent variable (abscissa), for $r = 1,2,...,m$.

Constraint: $x_1 \leq x_2 \leq ... \leq x_m$.

4: Y(M) – ***real*** array. *Input*

On entry: the values y_r of the of the dependent variable (ordinate), for $r = 1,2,...,m$.

5: W(M) – ***real*** array. *Input*

On entry: the values w_r of the weights, for $r = 1,2,...,m$. For advice on the choice of weights, see the Chapter Introduction.

Constraint: W(r) > 0, for $r = 1,2,...,m$.

6: LAMDA(NCAP7) – ***real*** array. *Input/Output*

On entry: LAMDA(i) must be set to the (i–4)th (interior) knot, λ_i, for $i = 5,6,...,\bar{n}+3$.

Constraint:
X(1) < LAMDA(5) ≤ LAMDA(6) ≤ ... ≤ LAMDA(NCAP7–4) < X(M).

On exit: the input values are unchanged, and LAMDA(i), for $i = 1,2,3,4$, NCAP7–3,NCAP7–2,NCAP7–1,NCAP7 contains the additional (exterior) knots introduced by the routine. For advice on the choice of knots, see Section 3.3 of the Chapter Introduction.

7: WORK1(M) – ***real*** array. *Workspace*

8: WORK2(4*NCAP7) – ***real*** array. *Workspace*

9: C(NCAP7) – ***real*** array. *Output*

On exit: the coefficient c_i of the B-spline $N_i(x)$, for $i = 1,2,...,\bar{n}+3$. The remaining elements of the array are not used.

10: SS – ***real.*** *Output*

On exit: the residual sum of squares, θ.

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The knots fail to satisfy the condition

X(1) < LAMDA(5) ≤ LAMDA(6) ≤ ... ≤ LAMDA(NCAP7–4) < X(M).

Thus the knots are not in correct order or are not interior to the data interval.

IFAIL = 2

The weights are not all strictly positive.

IFAIL = 3

The values of X(r), for $r = 1,2,...,$M are not in non-decreasing order.

IFAIL = 4

NCAP7 < 8 (so the number of interior knots is negative) or NCAP7 > MDIST + 4, where MDIST is the number of distinct x values in the data (so there cannot be a unique solution).

IFAIL = 5

The conditions specified by Schoenberg and Whitney [8] fail to hold for at least one subset of the distinct data abscissae. That is, there is no subset of NCAP7–4 strictly increasing values, X(R(1)),X(R(2)),...,X(R(NCAP7–4)), among the abscissae such that

X(R(1)) < LAMDA(1) < X(R(5)),

X(R(2)) < LAMDA(2) < X(R(6)),

...

X(R(NCAP7–8)) < LAMDA(NCAP7–8) < X(R(NCAP7–4)).

This means that there is no unique solution: there are regions containing too many knots compared with the number of data points.

7. Accuracy

The rounding errors committed are such that the computed coefficients are exact for a slightly perturbed set of ordinates $y_r + \delta y_r$. The ratio of the root-mean-square value for the δy_r to the root-mean-square value of the y_r can be expected to be less than a small multiple of $\kappa \times m \times$ ***machine precision***, where κ is a condition number for the problem. Values of κ for 20-30 practical data sets all proved to lie between 4.5 and 7.8 (see Cox [3]). (Note that for these data sets, replacing the coincident end knots at the end-points x_1 and x_m used in the routine by various choices of non-coincident exterior knots gave values of κ between 16 and 180. Again see Cox [3] for further details.) In general we would not expect κ to be large unless the choice of knots results in near-violation of the Schoenberg-Whitney conditions.

A cubic spline which adequately fits the data and is free from spurious oscillations is more likely to be obtained if the knots are chosen to be grouped more closely in regions where the function (underlying the data) or its derivatives change more rapidly than elsewhere.

8. Further Comments

The time taken by the routine is approximately $C \times (2m+\bar{n}+7)$ seconds, where C is a machine-dependent constant.

Multiple knots are permitted as long as their multiplicity does not exceed 4, i.e. the complete set of knots must satisfy $\lambda_i < \lambda_{i+4}$, for $i = 1,2,...,\bar{n}+3$, (cf. Section 6). At a knot of multiplicity one (the usual case), $s(x)$ and its first two derivatives are continuous. At a knot of multiplicity two, $s(x)$ and its first derivative are continuous. At a knot of multiplicity three, $s(x)$ is continuous, and at a knot of multiplicity four, $s(x)$ is generally discontinous.

The routine can be used efficiently for cubic spline interpolation, i.e. if $m = \bar{n}+3$. The abscissae must then of course satisfy $x_i < x_2 < ... < x_m$. Recommended values for the knots in this case are $\lambda_i = x_{i-2}$, for $i = 5,6,...,\bar{n}+3$.

9. Example

Determine a weighted least-squares cubic spline approximation with five intervals (four interior knots) to a set of 14 given data points. Tabulate the data and the corresponding values of the approximating spline, together with the residual errors, and also the values of the approximating spline at points half-way between each pair of adjacent data points.

The example program is written in a general form that will enable a cubic spline approximation with $\bar{n}$ intervals ($\bar{n}-1$ interior knots) to be obtained to m data points, with arbitrary positive weights, and the approximation to be tabulated. Note that E02BBF is used to evaluate the approximating spline. The program is self-starting in that any number of data sets can be supplied.

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.

```
*     E02BAF Example Program Text
*     Mark 15 Revised.  NAG Copyright 1991.
*     .. Parameters ..
      INTEGER          MMAX, NC7MAX
      PARAMETER        (MMAX=200,NC7MAX=50)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FIT, SS, XARG
      INTEGER          IFAIL, IWGHT, J, M, NCAP, R
*     .. Local Arrays ..
      real             C(NC7MAX), LAMDA(NC7MAX), W(MMAX), WORK1(MMAX),
     +                 WORK2(4*NC7MAX), X(MMAX), Y(MMAX)
```

```
*     .. External Subroutines ..
      EXTERNAL         E02BAF, E02BBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02BAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         READ (NIN,*) NCAP, IWGHT
         IF (NCAP+7.LE.NC7MAX) THEN
            IF (NCAP.GT.1) READ (NIN,*) (LAMDA(J),J=5,NCAP+3)
            DO 40 R = 1, M
               IF (IWGHT.EQ.1) THEN
                  READ (NIN,*) X(R), Y(R)
                  W(R) = 1.0e0
               ELSE
                  READ (NIN,*) X(R), Y(R), W(R)
               END IF
   40       CONTINUE
            IFAIL = 0
*
            CALL E02BAF(M,NCAP+7,X,Y,W,LAMDA,WORK1,WORK2,C,SS,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '  J       LAMDA(J+2)     B-spline coeff C(J)'
            WRITE (NOUT,*)
            J = 1
            WRITE (NOUT,99998) J, C(1)
            DO 60 J = 2, NCAP + 2
               WRITE (NOUT,99999) J, LAMDA(J+2), C(J)
   60       CONTINUE
            WRITE (NOUT,99998) NCAP + 3, C(NCAP+3)
            WRITE (NOUT,*)
            WRITE (NOUT,99997) 'Residual sum of squares = ', SS
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Cubic spline approximation and residuals'
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '  R   Abscissa     Weight   Ordinate     Spline  Residual'
            WRITE (NOUT,*)
            DO 80 R = 1, M
               IFAIL = 0
*
               CALL E02BBF(NCAP+7,LAMDA,C,X(R),FIT,IFAIL)
*
               WRITE (NOUT,99995) R, X(R), W(R), Y(R), FIT, FIT - Y(R)
               IF (R.LT.M) THEN
                  XARG = 0.5e0*(X(R)+X(R+1))
*
                  CALL E02BBF(NCAP+7,LAMDA,C,XARG,FIT,IFAIL)
*
                  WRITE (NOUT,99996) XARG, FIT
               END IF
   80       CONTINUE
            GO TO 20
         END IF
      END IF
  100 STOP
*
99999 FORMAT (1X,I3,F15.4,F20.4)
99998 FORMAT (1X,I3,F35.4)
99997 FORMAT (1X,A,e12.2)
99996 FORMAT (1X,F14.4,F33.4)
99995 FORMAT (1X,I3,4F11.4,e10.2)
      END
```

9.2. Program Data

```
E02BAF Example Program Data
  14
   5   2
     1.50
     2.60
     4.00
     8.00
     0.20      0.00      0.20
     0.47      2.00      0.20
     0.74      4.00      0.30
     1.09      6.00      0.70
     1.60      8.00      0.90
     1.90      8.62      1.00
     2.60      9.10      1.00
     3.10      8.90      1.00
     4.00      8.15      0.80
     5.15      7.00      0.50
     6.17      6.00      0.70
     8.00      4.54      1.00
    10.00      3.39      1.00
    12.00      2.56      1.00
```

9.3. Program Results

```
E02BAF Example Program Results

  J       LAMDA(J+2)     B-spline coeff C(J)

  1                          -0.0465
  2          0.2000           3.6150
  3          1.5000           8.5724
  4          2.6000           9.4261
  5          4.0000           7.2716
  6          8.0000           4.1207
  7         12.0000           3.0822
  8                           2.5597

Residual sum of squares =     0.18E-02

Cubic spline approximation and residuals

  R   Abscissa     Weight   Ordinate     Spline  Residual

  1     0.2000     0.2000     0.0000    -0.0465 -0.47E-01
        0.3350                          1.0622
  2     0.4700     0.2000     2.0000     2.1057  0.11E+00
        0.6050                          3.0817
  3     0.7400     0.3000     4.0000     3.9880 -0.12E-01
        0.9150                          5.0558
  4     1.0900     0.7000     6.0000     5.9983 -0.17E-02
        1.3450                          7.1376
  5     1.6000     0.9000     8.0000     7.9872 -0.13E-01
        1.7500                          8.3544
  6     1.9000     1.0000     8.6200     8.6348  0.15E-01
        2.2500                          9.0076
  7     2.6000     1.0000     9.1000     9.0896 -0.10E-01
        2.8500                          9.0353
  8     3.1000     1.0000     8.9000     8.9125  0.12E-01
        3.5500                          8.5660
  9     4.0000     0.8000     8.1500     8.1321 -0.18E-01
        4.5750                          7.5592
 10     5.1500     0.5000     7.0000     6.9925 -0.75E-02
        5.6600                          6.5010
 11     6.1700     0.7000     6.0000     6.0255  0.26E-01
        7.0850                          5.2292
 12     8.0000     1.0000     4.5400     4.5315 -0.85E-02
        9.0000                          3.9045
 13    10.0000     1.0000     3.3900     3.3928  0.28E-02
```

```
        11.0000                                 2.9574
  14    12.0000      1.0000      2.5600         2.5597 -0.35E-03
```

E02BBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02BBF evaluates a cubic spline from its B-spline representation.

2. Specification

```
SUBROUTINE E02BBF (NCAP7, LAMDA, C, X, S, IFAIL)
INTEGER          NCAP7, IFAIL
real             LAMDA(NCAP7), C(NCAP7), X, S
```

3. Description

This routine evaluates the cubic spline $s(x)$ at a prescribed argument x from its augmented knot set λ_i, for $i = 1,2,...,\bar{n}+7$, (see E02BAF) and from the coefficients c_i, for $i = 1,2,...,q$ in its B-spline representation

$$s(x) = \sum_{i=1}^{q} c_i N_i(x)$$

Here $q = \bar{n}+3$, where $\bar{n}$ is the number of intervals of the spline, and $N_i(x)$ denotes the normalised B-spline of degree 3 defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$. The prescribed argument x must satisfy $\lambda_4 \leq x \leq \lambda_{\bar{n}+4}$.

It is assumed that $\lambda_j \geq \lambda_{j-1}$, for $j = 2,3,...,\bar{n}+7$, and $\lambda_{\bar{n}+4} > \lambda_4$.

If x is a point at which 4 knots coincide, $s(x)$ is discontinuous at x; in this case, S contains the value defined as x is approached from the right.

The method employed is that of evaluation by taking convex combinations due to de Boor [4]. For further details of the algorithm and its use see Cox [1] and [3].

It is expected that a common use of E02BBF will be the evaluation of the cubic spline approximations produced by E02BAF. A generalization of E02BBF which also forms the derivative of $s(x)$ is E02BCF. E02BCF takes about 50% longer than E02BBF.

4. References

[1] COX, M.G.
The Numerical Evaluation of B-splines.
J. Inst. Math. Appl., 10, pp. 134-149, 1972.

[2] COX, M.G.
The Numerical Evaluation of a Spline from its B-spline Representation.
J. Inst. Math. Appl., 21, pp. 135-143, 1978.

[3] COX, M.G. and HAYES, J.G.
Curve Fitting: A Guide and Suite of Algorithms for the Non-specialist User.
Report NAC26, National Physical Laboratory, Teddington, Middlesex, 1973.

[4] DE BOOR, C.
On Ealculating with B-splines.
J. Approx. Theory, 6, pp. 50-62, 1972.

5. Parameters

1: NCAP7 – INTEGER. *Input*

On entry: $\bar{n} + 7$, where $\bar{n}$ is the number of intervals (one greater than the number of interior knots, i.e. the knots strictly within the range λ_4 to $\lambda_{\bar{n}+4}$) over which the spline is defined.

Constraint: NCAP7 ≥ 8.

2: LAMDA(NCAP7) – ***real*** array. *Input*

On entry: LAMDA(j) must be set to the value of the jth member of the complete set of knots, λ_j for $j = 1,2,...,\bar{n}+7$.

Constraint: the LAMDA(j) must be in non-decreasing order with LAMDA(NCAP7−3) > LAMDA(4).

3: C(NCAP7) – ***real*** array. *Input*

On entry: the coefficient c_i of the B-spline $N_i(x)$, for $i = 1,2,...,\bar{n}+3$. The remaining elements of the array are not used.

4: X – ***real.*** *Input*

On entry: the argument x at which the cubic spline is to be evaluated.

Constraint: LAMDA(4) ≤ X ≤ LAMDA(NCAP7−3).

5: S – ***real.*** *Output*

On exit: the value of the spline, $s(x)$.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The argument X does not satisfy LAMDA(4) ≤ X ≤ LAMDA(NCAP7−3).
In this case the value of S is set arbitrarily to zero.

IFAIL = 2

NCAP7 < 8, i.e. the number of interior knots is negative.

7. Accuracy

The computed value of $s(x)$ has negligible error in most practical situations. Specifically, this value has an **absolute** error bounded in modulus by $18 \times c_{max} \times$***machine precision***, where c_{max} is the largest in modulus of c_j, c_{j+1}, c_{j+2} and c_{j+3}, and j is an integer such that $\lambda_{j+3} \leq x \leq \lambda_{j+4}$. If c_j, c_{j+1}, c_{j+2} and c_{j+3} are all of the same sign, then the computed value of $s(x)$ has a **relative** error not exceeding $20 \times$***machine precision*** in modulus. For further details see Cox [2].

8. Further Comments

The time taken by the routine is approximately $C \times (1+0.1 \times \log(\bar{n}+7))$ seconds, where C is a machine-dependent constant.

Note: the routine does not test all the conditions on the knots given in the description of LAMDA in Section 5, since to do this would result in a computation time approximately linear in $\bar{n}+7$ instead of $\log(\bar{n}+7)$. All the conditions are tested in E02BAF, however.

9. Example

Evaluate at 9 equally-spaced points in the interval $1.0 \leq x \leq 9.0$ the cubic spline with (augmented) knots 1.0, 1.0, 1.0, 1.0, 3.0, 6.0, 8.0, 9.0, 9.0, 9.0, 9.0 and normalised cubic B-spline coefficients 1.0, 2.0, 4.0, 7.0, 6.0, 4.0, 3.0.

The example program is written in a general form that will enable a cubic spline with $\bar{n}$ intervals, in its normalised cubic B-spline form, to be evaluated at m equally-spaced points in the interval LAMDA(4) ≤ x ≤ LAMDA($\bar{n}$+4). The program is self-starting in that any number of data sets may be supplied.

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.

```
*     E02BBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NC7MAX
      PARAMETER        (NC7MAX=200)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             A, B, S, X
      INTEGER          IFAIL, J, M, NCAP, R
*     .. Local Arrays ..
      real             C(NC7MAX), LAMDA(NC7MAX)
*     .. External Subroutines ..
      EXTERNAL         E02BBF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02BBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=80) M
      IF (M.GT.0) THEN
         READ (NIN,*) NCAP
         IF (NCAP+7.LE.NC7MAX) THEN
            READ (NIN,*) (LAMDA(J),J=1,NCAP+7)
            READ (NIN,*) (C(J),J=1,NCAP+3)
            A = LAMDA(4)
            B = LAMDA(NCAP+4)
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '  J       LAMDA(J)    B-spline coefficient (J-2)'
            WRITE (NOUT,*)
            DO 40 J = 1, NCAP + 7
               IF (J.LT.3 .OR. J.GT.NCAP+5) THEN
                  WRITE (NOUT,99999) J, LAMDA(J)
               ELSE
                  WRITE (NOUT,99999) J, LAMDA(J), C(J-2)
               END IF
   40       CONTINUE
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        '  R       Argument      Value of cubic spline'
            WRITE (NOUT,*)
            DO 60 R = 1, M
               X = (real(M-R)*A+real(R-1)*B)/real(M-1)
               IFAIL = 0
*
               CALL E02BBF(NCAP+7,LAMDA,C,X,S,IFAIL)
*
               WRITE (NOUT,99999) R, X, S
   60       CONTINUE
            GO TO 20
         END IF
      END IF
   80 STOP
*
99999 FORMAT (1X,I3,F14.4,F21.4)
      END
```

9.2. Program Data

```
E02BBF Example Program Data
  9
   4
     1.00
     1.00
     1.00
     1.00
     3.00
     6.00
     8.00
     9.00
     9.00
     9.00
     9.00
     1.00
     2.00
     4.00
     7.00
     6.00
     4.00
     3.00
```

9.3. Program Results

```
 E02BBF Example Program Results

   J       LAMDA(J)    B-spline coefficient (J-2)

   1        1.0000
   2        1.0000
   3        1.0000              1.0000
   4        1.0000              2.0000
   5        3.0000              4.0000
   6        6.0000              7.0000
   7        8.0000              6.0000
   8        9.0000              4.0000
   9        9.0000              3.0000
  10        9.0000
  11        9.0000

   R       Argument      Value of cubic spline

   1        1.0000              1.0000
   2        2.0000              2.3779
   3        3.0000              3.6229
   4        4.0000              4.8327
   5        5.0000              5.8273
   6        6.0000              6.3571
   7        7.0000              6.1905
   8        8.0000              5.1667
   9        9.0000              3.0000
```

E02BCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02BCF evaluates a cubic spline and its first three derivatives from its B-spline representation.

2. Specification

```
SUBROUTINE E02BCF (NCAP7, LAMDA, C, X, LEFT, S, IFAIL)
INTEGER          NCAP7, LEFT, IFAIL
real             LAMDA(NCAP7), C(NCAP7), X, S(4)
```

3. Description

This routine evaluates the cubic spline $s(x)$ and its first three derivatives at a prescribed argument x. It is assumed that $s(x)$ is represented in terms of its B-spline coefficients c_i, for $i = 1,2,...,\bar{n}+3$ and (augmented) ordered knot set λ_i, for $i = 1,2,...,\bar{n}+7$, (see E02BAF), i.e.

$$s(x) = \sum_{i=1}^{q} c_i N_i(x)$$

Here $q = \bar{n}+3$, $\bar{n}$ is the number of intervals of the spline and $N_i(x)$ denotes the normalised B-spline of degree 3 (order 4) defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$. The prescribed argument x must satisfy

$$\lambda_4 \leq x \leq \lambda_{\bar{n}+4}$$

At a simple knot λ_i (i.e. one satisfying $\lambda_{i-1} < \lambda_i < \lambda_{i+1}$), the third derivative of the spline is in general discontinuous. At a multiple knot (i.e. two or more knots with the same value), lower derivatives, and even the spline itself, may be discontinuous. Specifically, at a point $x = u$ where (exactly) r knots coincide (such a point is termed a knot of multiplicity r), the values of the derivatives of order $4 - j$, for $j = 1,2,...,r$, are in general discontinuous. (Here $1 \leq r \leq 4$; $r > 4$ is not meaningful.) The user must specify whether the value at such a point is required to be the left- or right-hand derivative.

The method employed is based upon:

(i) carrying out a binary search for the knot interval containing the argument x (see Cox [3]),

(ii) evaluating the non-zero B-splines of orders 1,2,3 and 4 by recurrence (see Cox [2] and [3]),

(iii) computing all derivatives of the B-splines of order 4 by applying a second recurrence to these computed B-spline values (see Cox [1]),

(iv) multiplying the 4th-order B-spline values and their derivative by the appropriate B-spline coefficients, and summing, to yield the values of $s(x)$ and its derivatives.

E02BCF can be used to compute the values and derivatives of cubic spline fits and interpolants produced by E02BAF.

If only values and not derivatives are required, E02BBF may be used instead of E02BCF, which takes about 50% longer than E02BBF.

4. References

[1] DE BOOR, C.
On calculating with B-splines.
J. Approx. Theory, 6, pp. 50-62, 1972.

[2] COX, M.G.
The numerical evaluation of B-splines.
J. Inst. Maths. Applics., 10, pp. 134-149, 1972.

[3] COX, M.G.
The numerical evaluation of a spline from its B-spline representation.
J. Inst. Maths. Applics., 21, pp. 135-143, 1978.

5. Parameters

1: NCAP7 – INTEGER. *Input*

On entry: $\bar{n} + 7$, where $\bar{n}$ is the number of intervals of the spline (which is one greater than the number of interior knots, i.e. the knots strictly within the range λ_4 to $\lambda_{\bar{n}+4}$ over which the spline is defined).

Constraint: NCAP7 ≥ 8.

2: LAMDA(NCAP7) – ***real*** array. *Input*

On entry: LAMDA(j) must be set to the value of the jth member of the complete set of knots, λ_j, for $j = 1,2,...,\bar{n}+7$.

Constraint: the LAMDA(j) must be in non-decreasing order with LAMDA(NCAP7−3) > LAMDA(4).

3: C(NCAP7) – ***real*** array. *Input*

On entry: the coefficient c_i of the B-spline $N_i(x)$, for $i = 1,2,...,\bar{n}+3$. The remaining elements of the array are not used.

4: X – ***real***. *Input*

On entry: the argument x at which the cubic spline and its derivatives are to be evaluated.

Constraint: LAMDA(4) ≤ X ≤ LAMDA(NCAP7−3).

5: LEFT – INTEGER. *Input*

On entry: specifies whether left- or right-hand values of the spline and its derivatives are to be computed (see Section 3). Left- or right-hand values are formed according to whether LEFT is equal or not equal to 1. If x does not coincide with a knot, the value of LEFT is immaterial. If x = LAMDA(4), right-hand values are computed, and if x = LAMDA(NCAP7−3), left-hand values are formed, regardless of the value of LEFT.

6: S(4) – ***real*** array. *Output*

On exit: S(j) contains the value of the (j−1)th derivative of the spline at the argument x, for $j = 1,2,3,4$. Note that S(1) contains the value of the spline.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

NCAP7 < 8, i.e. the number of intervals is not positive.

IFAIL = 2

Either LAMDA(4) ≥ LAMDA(NCAP7−3), i.e. the range over which $s(x)$ is defined is null or negative in length, or X is an invalid argument, i.e. X < LAMDA(4) or X > LAMDA(NCAP7−3).

7. Accuracy

The computed value of $s(x)$ has negligible error in most practical situations. Specifically, this value has an **absolute** error bounded in modulus by $18 \times c_{max} \times$***machine precision***, where c_{max} is the largest in modulus of c_j, c_{j+1}, c_{j+2} and c_{j+3}, and j is an integer such that $\lambda_{j+3} \leq x \leq \lambda_{j+4}$. If c_j, c_{j+1}, c_{j+2} and c_{j+3} are all of the same sign, then the computed value of $s(x)$ has **relative** error bounded by $18 \times$***machine precision***. For full details see Cox [3].

No complete error analysis is available for the computation of the derivatives of $s(x)$. However, for most practical purposes the absolute errors in the computed derivatives should be small.

8. Further Comments

The time taken by this routine is approximately linear in $\log(\bar{n}+7)$.

Note: the routine does not test all the conditions on the knots given in the description of LAMDA in Section 5, since to do this would result in a computation time approximately linear in $\bar{n}+7$ instead of $\log(\bar{n}+7)$. All the conditions are tested in E02BAF, however.

9. Example

Compute, at the 7 arguments $x = 0, 1, 2, 3, 4, 5, 6$, the left- and right-hand values and first 3 derivatives of the cubic spline defined over the interval $0 \leq x \leq 6$ having the 6 interior knots $x = 1, 3, 3, 3, 4, 4$, the 8 additional knots 0, 0, 0, 0, 6, 6, 6, 6, and the 10 B-spline coefficients 10, 12, 13, 15, 22, 26, 24, 18, 14, 12.

The input data items (using the notation of Section 5) comprise the following values in the order indicated:

$\bar{n}$	m
LAMDA(j),	for j = 1,2,...,NCAP7
C(j),	for j = 1,2,...,NCAP7−4
X(i),	for i = 1,2,...,m

The example program is written in a general form that will enable the values and derivatives of a cubic spline having an arbitrary number of knots to be evaluated at a set of arbitrary points. Any number of data sets may be supplied. The only changes required to the program relate to the dimensions of the arrays LAMDA, C and X.

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.

```
*     E02BCF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NC7MAX
      PARAMETER        (NC7MAX=100)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             X
      INTEGER          I, IFAIL, J, L, LEFT, M, NCAP
*     .. Local Arrays ..
      real             C(NC7MAX), LAMDA(NC7MAX), S(4)
*     .. External Subroutines ..
      EXTERNAL         E02BCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02BCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=80) NCAP, M
      IF (NCAP.GT.0 .AND. NCAP+7.LE.NC7MAX) THEN
         READ (NIN,*) (LAMDA(J),J=1,NCAP+7)
         READ (NIN,*) (C(J),J=1,NCAP+3)
```

```
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +   '      X              Spline    1st deriv  2nd deriv  3rd deriv'
         DO 60 I = 1, M
            READ (NIN,*) X
            DO 40 LEFT = 1, 2
               IFAIL = 0
*
               CALL E02BCF(NCAP+7,LAMDA,C,X,LEFT,S,IFAIL)
*
               IF (LEFT.EQ.1) THEN
                  WRITE (NOUT,*)
                  WRITE (NOUT,99999) X, '  LEFT', (S(L),L=1,4)
               ELSE
                  WRITE (NOUT,99999) X, ' RIGHT', (S(L),L=1,4)
               END IF
   40       CONTINUE
   60    CONTINUE
         GO TO 20
      END IF
   80 STOP
*
99999 FORMAT (1X,e11.3,A,4e11.3)
      END
```

9.2. Program Data

```
E02BCF Example Program Data
   7   7
  0.0      0.0      0.0      0.0      1.0      3.0      3.0      3.0
  4.0      4.0      6.0      6.0      6.0      6.0
 10.0     12.0     13.0     15.0     22.0     26.0     24.0     18.0
 14.0     12.0
  0.0
  1.0
  2.0
  3.0
  4.0
  5.0
  6.0
```

9.3. Program Results

```
 E02BCF Example Program Results

      X              Spline    1st deriv  2nd deriv  3rd deriv

  0.000E+00  LEFT  0.100E+02  0.600E+01 -0.100E+02  0.107E+02
  0.000E+00 RIGHT  0.100E+02  0.600E+01 -0.100E+02  0.107E+02

  0.100E+01  LEFT  0.128E+02  0.133E+01  0.667E+00  0.107E+02
  0.100E+01 RIGHT  0.128E+02  0.133E+01  0.667E+00  0.392E+01

  0.200E+01  LEFT  0.151E+02  0.396E+01  0.458E+01  0.392E+01
  0.200E+01 RIGHT  0.151E+02  0.396E+01  0.458E+01  0.392E+01

  0.300E+01  LEFT  0.220E+02  0.105E+02  0.850E+01  0.392E+01
  0.300E+01 RIGHT  0.220E+02  0.120E+02 -0.360E+02  0.360E+02

  0.400E+01  LEFT  0.220E+02 -0.600E+01  0.000E+00  0.360E+02
  0.400E+01 RIGHT  0.220E+02 -0.600E+01  0.000E+00  0.150E+01

  0.500E+01  LEFT  0.163E+02 -0.525E+01  0.150E+01  0.150E+01
  0.500E+01 RIGHT  0.163E+02 -0.525E+01  0.150E+01  0.150E+01

  0.600E+01  LEFT  0.120E+02 -0.300E+01  0.300E+01  0.150E+01
  0.600E+01 RIGHT  0.120E+02 -0.300E+01  0.300E+01  0.150E+01
```

E02BDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02BDF computes the definite integral of a cubic spline from its B-spline representation.

2. Specification

```
SUBROUTINE E02BDF (NCAP7, LAMDA, C, DEFINT, IFAIL)
INTEGER          NCAP7, IFAIL
real             LAMDA(NCAP7), C(NCAP7), DEFINT
```

3. Description

This routine computes the definite integral of the cubic spline $s(x)$ between the limits $x = a$ and $x = b$, where a and b are respectively the lower and upper limits of the range over which $s(x)$ is defined. It is assumed that $s(x)$ is represented in terms of its B-spline coefficients c_i, for $i = 1,2,...,\bar{n}+3$ and (augmented) ordered knot set λ_i, for $i = 1,2,...,\bar{n}+7$, with $\lambda_i = a$, for $i = 1,2,3,4$ and $\lambda_i = b$, for $i = \bar{n}+4,\bar{n}+5,\bar{n}+6,\bar{n}+7$, (see E02BAF), i.e.

$$s(x) = \sum_{i=1}^{q} c_i N_i(x).$$

Here $q = \bar{n} + 3$, $\bar{n}$ is the number of intervals of the spline and $N_i(x)$ denotes the normalised B-spline of degree 3 (order 4) defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$.

The method employed uses the formula given in Section 3 of Cox [1].

E02BDF can be used to determine the definite integrals of cubic spline fits and interpolants produced by E02BAF.

4. References

[1] COX, M.G.
An algorithm for spline interpolation.
J. Inst. Maths. Applics., 15, pp. 95-108, 1975.

5. Parameters

1: NCAP7 – INTEGER. *Input*

On entry: $\bar{n} + 7$, where $\bar{n}$ is the number of intervals of the spline (which is one greater than the number of interior knots, i.e. the knots strictly within the range a to b) over which the spline is defined.

Constraint: NCAP7 ≥ 8.

2: LAMDA(NCAP7) – ***real*** array. *Input*

On entry: LAMDA(j) must be set to the value of the jth member of the complete set of knots, λ_j for $j = 1,2,...,\bar{n}+7$.

Constraint: the LAMDA(j) must be in non-decreasing order with LAMDA(NCAP7−3) > LAMDA(4) and satisfy

LAMDA(1) = LAMDA(2) = LAMDA(3) = LAMDA(4)

and

LAMDA(NCAP7−3) = LAMDA(NCAP7−2) = LAMDA(NCAP7−1) = LAMDA(NCAP7).

3: C(NCAP7) – ***real*** array. *Input*

On entry: the coefficient c_i of the B-spline $N_i(x)$, for $i = 1,2,...,\bar{n}+3$. The remaining elements of the array are not used.

4: DEFINT – ***real.*** *Output*

On exit: the value of the definite integral of $s(x)$ between the limits $x = a$ and $x = b$, where $a = \lambda_4$ and $b = \lambda_{\bar{n}+4}$.

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

NCAP7 < 8, i.e. the number of intervals is not positive.

IFAIL = 2

At least one of the following restrictions on the knots is violated:

LAMDA(NCAP7–3) > LAMDA(4), LAMDA(j) ≥ LAMDA(j–1),
for j = 2,3,...,NCAP7, with equality in the cases j = 2,3,4,NCAP7–2,NCAP7–1, and NCAP7.

7. Accuracy

The rounding errors are such that the computed value of the integral is exact for a slightly perturbed set of B-spline coefficients c_i differing in a relative sense from those supplied by no more than $2.2\times(\bar{n}+3)\times$***machine precision.***

8. Further Comments

The time taken by the routine is approximately proportional to $\bar{n} + 7$.

9. Example

Determine the definite integral over the interval $0 \le x \le 6$ of a cubic spline having 6 interior knots at the positions λ = 1, 3, 3, 3, 4, 4, the 8 additional knots 0, 0, 0, 0, 6, 6, 6, 6, and the 10 B-spline coefficients 10, 12, 13, 15, 22, 26, 24, 18, 14, 12.

The input data items (using the notation of Section 5) comprise the following values in the order indicated:

$\bar{n}$
LAMDA(j), for j = 1,2,...,NCAP7
C(j), for j = 1,2,...,NCAP7–3

The example program is written in a general form that will enable the definite integral of a cubic spline having an arbitrary number of knots to be computed. Any number of data sets may be supplied. The only changes required to the program relate to the dimensions of the arrays LAMDA and C.

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.

```
*     E02BDF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NC7MAX
      PARAMETER        (NC7MAX=100)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             DEFINT
      INTEGER          IFAIL, J, NCAP
*     .. Local Arrays ..
      real             C(NC7MAX), K(NC7MAX)
*     .. External Subroutines ..
      EXTERNAL         E02BDF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02BDF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*) NCAP
      IF (NCAP.GT.0 .AND. NCAP+7.LE.NC7MAX) THEN
         READ (NIN,*) (K(J),J=1,NCAP+7)
         READ (NIN,*) (C(J),J=1,NCAP+3)
         IFAIL = 0
*
         CALL E02BDF(NCAP+7,K,C,DEFINT,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Definite integral = ', DEFINT
         GO TO 20
      END IF
      STOP
*
99999 FORMAT (1X,A,e11.3)
      END
```

9.2. Program Data

```
E02BDF Example Program Data
   7
 0.0      0.0      0.0      0.0      1.0      3.0      3.0      3.0
 4.0      4.0      6.0      6.0      6.0      6.0
10.0     12.0     13.0     15.0     22.0     26.0     24.0     18.0
14.0     12.0
   0
```

9.3. Program Results

```
 E02BDF Example Program Results

 Definite integral =   0.100E+03
```

E02BEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02BEF computes a cubic spline approximation to an arbitrary set of data points. The knots of the spline are located automatically, but a single parameter must be specified to control the trade-off between closeness of fit and smoothness of fit.

2. Specification

```
      SUBROUTINE E02BEF (START, M, X, Y, W, S, NEST, N, LAMDA, C, FP, WRK,
     1                   LWRK, IWRK, IFAIL)
      INTEGER           M, NEST, N, LWRK, IWRK(NEST), IFAIL
      real              X(M), Y(M), W(M), S, LAMDA(NEST), C(NEST), FP,
     1                  WRK(LWRK)
      CHARACTER*1       START
```

3. Description

This routine determines a smooth cubic spline approximation $s(x)$ to the set of data points (x_r,y_r), with weights w_r, for $r = 1,2,...,m$.

The spline is given in the B-spline representation

$$s(x) = \sum_{i=1}^{n-4} c_i N_i(x), \tag{1}$$

where $N_i(x)$ denotes the normalised cubic B-spline defined upon the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$.

The total number n of these knots and their values $\lambda_1,...,\lambda_n$ are chosen automatically by the routine. The knots $\lambda_5,...,\lambda_{n+4}$ are the interior knots; they divide the approximation interval $[x_1,x_m]$ into $n-7$ subintervals. The coefficients $c_1,c_2,...,c_{n-4}$ are then determined as the solution of the following constrained minimization problem:

minimize

$$\eta = \sum_{i=5}^{n-4} \delta_i^2 \tag{2}$$

subject to the constraint

$$\theta = \sum_{r=1}^{m} \epsilon_r^2 \leq S, \tag{3}$$

where: δ_i stands for the discontinuity jump in the third order derivative of $s(x)$ at the interior knot λ_i,

ϵ_r denotes the weighted residual $w_r(y_r-s(x_r))$,

and S is a non-negative number to be specified by the user.

The quantity η can be seen as a measure of the (lack of) smoothness of $s(x)$, while closeness of fit is measured through θ. By means of the parameter S, 'the smoothing factor', the user will then control the balance between these two (usually conflicting) properties. If S is too large, the spline will be too smooth and signal will be lost (underfit); if S is too small, the spline will pick up too much noise (overfit). In the extreme cases the routine will return an interpolating spline ($\theta = 0$) if S is set to zero, and the weighted least-squares cubic polynomial ($\eta = 0$) if S is set very large. Experimenting with S values between these two extremes should result in a good compromise. (See Section 8.2 for advice on choice of S.)

The method employed is outlined in Section 8.3 and fully described in Dierckx [1], [2] and [3]. It involves an adaptive strategy for locating the knots of the cubic spline (depending on the function underlying the data and on the value of S), and an iterative method for solving the constrained minimization problem once the knots have been determined.

Values of the computed spline, or of its derivatives or definite integral, can subsequently be computed by calling E02BBF, E02BCF or E02BDF, as described in Section 8.4.

4. References

[1] DIERCKX, P.
An Algorithm for Smoothing, Differentiating and Integration of Experimental Data Using Spline Functions.
J. Comp. Appl. Maths., 1, pp. 165-184, 1975.

[2] DIERCKX, P.
A Fast Algorithm for Smoothing Data on a Rectangular Grid while using Spline Functions.
SIAM J. Numer. Anal., 19, pp. 1286-1304, 1982.

[3] DIERCKX, P.
An Improved Algorithm for Curve Fitting with Spline Functions.
Dept. Computer Science, K.U.Leuven, Report TW54, 1981.

[4] REINSCH, C.H.
Smoothing by Spline Functions.
Num. Math., 10, pp. 177-183, 1967.

5. Parameters

1: START – CHARACTER*1. *Input*

On entry: START must be set to 'C', 'c', 'W' or 'w' (or to the corresponding lower-case letters).

If START = 'C' or 'c' (Cold start), the routine will build up the knot set starting with no interior knots. No values need be assigned to the parameters N, LAMDA, WRK or IWRK.

If START = 'W' or 'w' (Warm start), the routine will restart the knot-placing strategy using the knots found in a previous call of the routine. In this case, the parameters N, LAMDA, WRK, and IWRK must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of S.

Constraint: START = 'C', 'c', 'W' or 'w'.

2: M – INTEGER. *Input*

On entry: m, the number of data points.

Constraint: M $\geq$ 4.

3: X(M) – ***real*** array. *Input*

On entry: the values x_r of the independent variable (abscissa) x, for $r = 1,2,...,m$.

Constraint: $x_1 < x_2 < ... < x_m$

4: Y(M) – ***real*** array. *Input*

On entry: the values y_r of the dependent variable (ordinate) y, for $r = 1,2,...,m$.

5: W(M) – ***real*** array. *Input*

On entry: the values w_r of the weights, for $r = 1,2,...,m$. For advice on the choice of weights, see the Chapter Introduction, Section 2.1.2.

Constraint: W(r) > 0, for $r = 1,2,...,m$.

6: S – ***real***. *Input*

On entry: the smoothing factor, S.

If $S = 0.0$, the routine returns an interpolating spline.

If S is smaller than ***machine precision***, it is assumed equal to zero.

For advice on the choice of S, see Sections 3 and 8.2.

Constraint: S ≥ 0.0.

7: NEST – INTEGER. *Input*

On entry: an over-estimate for the number, n, of knots required.

Constraint: NEST ≥ 8. In most practical situations, NEST = M/2 is sufficient. NEST never needs to be larger than M + 4, the number of knots needed for interpolation (S = 0.0).

8: N – INTEGER. *Input/Output*

On entry: if the warm start option is used, the value of N must be left unchanged from the previous call.

On exit: the total number, n, of knots of the computed spline.

9: LAMDA(NEST) – ***real*** array. *Input/Output*

On entry: if the warm start option is used, the values LAMDA(1), LAMDA(2),...,LAMDA(N) must be left unchanged from the previous call.

On exit: the knots of the spline i.e. the positions of the interior knots LAMDA(5), LAMDA(6),...,LAMDA(N–4) as well as the positions of the additional knots LAMDA(1) = LAMDA(2) = LAMDA(3) = LAMDA(4) = x_1 and LAMDA(N–3) = LAMDA(N–2) = LAMDA(N–1) = LAMDA(N) = x_m needed for the B-spline representation.

10: C(NEST) – ***real*** array. *Output*

On exit: the coefficient c_i of the B-spline $N_i(x)$ in the spline approximation $s(x)$, for $i = 1,2,...,n-4$.

11: FP – ***real***. *Output*

On exit: the weighted sum of squared residuals, θ, of the computed spline approximation. If FP = 0.0, this is an interpolating spline. FP should equal S within a relative tolerance of 0.001 unless $n = 8$ when the spline has no interior knots and so is simply a cubic polynomial. For knots to be inserted, S must be set to a value below the value of FP produced in this case.

12: WRK(LWRK) – ***real*** array. *Workspace*

On entry: if the warm start option is used, the values WRK(1),...,WRK(n) must be left unchanged from the previous call.

13: LWRK – INTEGER. *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E02BEF is called.

Constraint: LWRK ≥ 4×M + 16×NEST + 41.

14: IWRK(NEST) – INTEGER array. *Workspace*

On entry: if the warm start option is used, the values IWRK(1),...,IWRK(n) must be left unchanged from the previous call.

This array is used as workspace.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, START ≠ 'C', 'c', 'W' or 'w',
or M < 4,
or S < 0.0,
or S = 0.0 and NEST < M + 4,
or NEST < 8,
or LWRK < 4×M + 16×NEST + 41.

IFAIL = 2

The weights are not all strictly positive.

IFAIL = 3

The values of X(r), for r = 1,2,...,M, are not in strictly increasing order.

IFAIL = 4

The number of knots required is greater than NEST. Try increasing NEST and, if necessary, supplying larger arrays for the parameters LAMDA, C, WRK and IWRK. However, if NEST is already large, say NEST > M/2, then this error exit may indicate that S is too small.

IFAIL = 5

The iterative process used to compute the coefficients of the approximating spline has failed to converge. This error exit may occur if S has been set very small. If the error persists with increased S, consult NAG.

If IFAIL = 4 or 5, a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3) in Section 3) – perhaps by only a small amount, however.

7. Accuracy

On successful exit, the approximation returned is such that its weighted sum of squared residuals FP is equal to the smoothing factor S, up to a specified relative tolerance of 0.001 – except that if n = 8, FP may be significantly less than S: in this case the computed spline is simply a weighted least-squares polynomial approximation of degree 3, i.e. a spline with no interior knots.

8. Further Comments

8.1. Timing

The time taken for a call of E02BEF depends on the complexity of the shape of the data, the value of the smoothing factor S, and the number of data points. If E02BEF is to be called for different values of S, much time can be saved by setting START = 'W' or 'w' after the first call.

8.2. Choice of S

If the weights have been correctly chosen (see Section 2.1.2 of the Chapter Introduction), the standard deviation of $w_r y_r$ would be the same for all r, equal to σ, say. In this case, choosing the smoothing factor S in the range $\sigma^2(m \pm \sqrt{2m})$, as suggested by Reinsch [4], is likely to give a good start in the search for a satisfactory value. Otherwise, experimenting with different values of S will be required from the start, taking account of the remarks in Section 3.

In that case, in view of computation time and memory requirements, it is recommended to start with a very large value for S and so determine the least-squares cubic polynomial; the value returned for FP, call it FP_0, gives an upper bound for S. Then progressively decrease the value of S to obtain closer fits – say by a factor of 10 in the beginning, i.e. $S = FP_0/10$, $S = FP_0/100$, and so on, and more carefully as the approximation shows more details.

The number of knots of the spline returned, and their location, generally depend on the value of S and on the behaviour of the function underlying the data. However, if E02BEF is called with START = 'W' or 'w', the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of S and START = 'W' or 'w', a fit can finally be accepted as satisfactory, it may be worthwhile to call E02BEF once more with the selected value for S but now using START = 'C' or 'c'. Often, E02BEF then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

8.3. Outline of Method Used

If $S = 0$, the requisite number of knots is known in advance, i.e. $n = m + 4$; the interior knots are located immediately as $\lambda_i = x_{i-2}$, for $i = 5,6,...,n-4$. The corresponding least-squares spline (see E02BAF) is then an interpolating spline and therefore a solution of the problem.

If $S > 0$, a suitable knot set is built up in stages (starting with no interior knots in the case of a cold start but with the knot set found in a previous call if a warm start is chosen). At each stage, a spline is fitted to the data by least-squares (see E02BAF) and θ, the weighted sum of squares of residuals, is computed. If $\theta > S$, new knots are added to the knot set to reduce θ at the next stage. The new knots are located in intervals where the fit is particularly poor, their number depending on the value of S and on the progress made so far in reducing θ. Sooner or later, we find that $\theta \leq S$ and at that point the knot set is accepted. The routine then goes on to compute the (unique) spline which has this knot set and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = S$. The routine computes the spline by an iterative scheme which is ended when $\theta = S$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in E02BAF.

An exception occurs when the routine finds at the start that, even with no interior knots ($n = 8$), the least-squares spline already has its weighted sum of squares of residuals $\leq S$. In this case, since this spline (which is simply a cubic polynomial) also has an optimal value for the smoothness measure η, namely zero, it is returned at once as the (trivial) solution. It will usually mean that S has been chosen too large.

For further details of the algorithm and its use, see Dierckx [3].

8.4. Evaluation of Computed Spline

The value of the computed spline at a given value X may be obtained in the ***real*** variable S by the call:

```
CALL E02BBF(N,LAMDA,C,X,S,IFAIL)
```

where N, LAMDA and C are the output parameters of E02BEF.

The values of the spline and its first three derivatives at a given value X may be obtained in the ***real*** array SDIF of dimension at least 4 by the call:

```
CALL E02BCF(N,LAMDA,C,X,LEFT,SDIF,IFAIL)
```

where if LEFT = 1, left-hand derivatives are computed and if LEFT $\neq$ 1, right-hand derivatives are calculated. The value of LEFT is only relevant if X is an interior knot.

The value of the definite integral of the spline over the interval X(1) to X(M) can be obtained in the ***real*** variable SINT by the call:

```
CALL E02BDF(N,LAMDA,C,SINT,IFAIL)
```

9. Example

This example program reads in a set of data values, followed by a set of values of S. For each value of S it calls E02BEF to compute a spline approximation, and prints the values of the knots and the B-spline coefficients c_i.

The program includes code to evaluate the computed splines, by calls to E02BBF, at the points x_r and at points mid-way between them. These values are not printed out, however; instead the results are illustrated by plots of the computed splines, together with the data points (indicated by

×) and the positions of the knots (indicated by vertical lines): the effect of decreasing S can be clearly seen. (The plots were obtained by calling NAG Graphical Supplement routine J06FAF.)

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.

```
*     E02BEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NEST, LWRK
      PARAMETER        (MMAX=50,NEST=MMAX+4,LWRK=4*MMAX+16*NEST+41)
*     .. Local Scalars ..
      real             FP, S, TXR
      INTEGER          IFAIL, J, M, N, R
      CHARACTER*1      START
*     .. Local Arrays ..
      real             C(NEST), K(NEST), SP(2*MMAX-1), W(MMAX),
     +                 WRK(LWRK), X(MMAX), Y(MMAX)
      INTEGER          IWRK(NEST)
*     .. External Subroutines ..
      EXTERNAL         E02BBF, E02BEF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02BEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of data points, followed by the data points (X),
*     the function values (Y) and the weights (W).
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         DO 20 R = 1, M
            READ (NIN,*) X(R), Y(R), W(R)
   20    CONTINUE
         START = 'Cold Start'
*        Read in successive values of S until end of data file.
   40    READ (NIN,*,END=120) S
*        Determine the spline approximation.
         IFAIL = 0
*
         CALL E02BEF(START,M,X,Y,W,S,NEST,N,K,C,FP,WRK,LWRK,IWRK,IFAIL)
*
*        Evaluate the spline at each X point and midway between
*        X points, saving the results in SP.
         DO 60 R = 1, M
            IFAIL = 0
*
            CALL E02BBF(N,K,C,X(R),SP((R-1)*2+1),IFAIL)
*
   60    CONTINUE
         DO 80 R = 1, M - 1
            IFAIL = 0
            TXR = (X(R)+X(R+1))/2
*
            CALL E02BBF(N,K,C,TXR,SP(R*2),IFAIL)
*
   80    CONTINUE
*        Output the results.
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calling with smoothing factor S =', S
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     '                                                B-Spline'
         WRITE (NOUT,*)
     +     '              J       Knot K(J+2)    Coefficient C(J)'
         WRITE (NOUT,99998) 1, C(1)
         DO 100 J = 2, N - 5
```

```
            WRITE (NOUT,99997) J, K(J+2), C(J)
  100     CONTINUE
          WRITE (NOUT,99998) N - 4, C(N-4)
          WRITE (NOUT,*)
          WRITE (NOUT,99999) 'Weighted sum of squared residuals FP =', FP
          IF (FP.EQ.0.0e0) THEN
             WRITE (NOUT,*) '(The spline is an interpolating spline)'
          ELSE IF (N.EQ.8) THEN
             WRITE (NOUT,*)
     +       '(The spline is the weighted least-squares cubic polynomial)'
          END IF
          WRITE (NOUT,*)
          START = 'Warm Start'
          GO TO 40
        END IF
  120 STOP
*
99999 FORMAT (1X,A,1P,e12.3)
99998 FORMAT (11X,I4,16X,F16.4)
99997 FORMAT (11X,I4,2F16.4)
      END
```

9.2. Program Data

```
E02BEF Example Program Data
  15                                  M, the number of data points
  0.0000E+00  -1.1000E+00   1.00    X, Y, W, abscissa, ordinate and weight
  5.0000E-01  -3.7200E-01   2.00
  1.0000E+00   4.3100E-01   1.50
  1.5000E+00   1.6900E+00   1.00
  2.0000E+00   2.1100E+00   3.00
  2.5000E+00   3.1000E+00   1.00
  3.0000E+00   4.2300E+00   0.50
  4.0000E+00   4.3500E+00   1.00
  4.5000E+00   4.8100E+00   2.00
  5.0000E+00   4.6100E+00   2.50
  5.5000E+00   4.7900E+00   1.00
  6.0000E+00   5.2300E+00   3.00
  7.0000E+00   6.3500E+00   1.00
  7.5000E+00   7.1900E+00   2.00
  8.0000E+00   7.9700E+00   1.00    End of data points
  1.0                               S, smoothing factor
  0.5                               S, smoothing factor
  0.1                               S, smoothing factor
```

9.3. Program Results

```
 E02BEF Example Program Results

 Calling with smoothing factor S =   1.000E+00

                                        B-Spline
              J       Knot K(J+2)   Coefficient C(J)
              1                         -1.3201
              2          0.0000          1.3542
              3          4.0000          5.5510
              4          8.0000          4.7031
              5                          8.2277

 Weighted sum of squared residuals FP =   1.000E+00
```

Calling with smoothing factor S = 5.000E-01

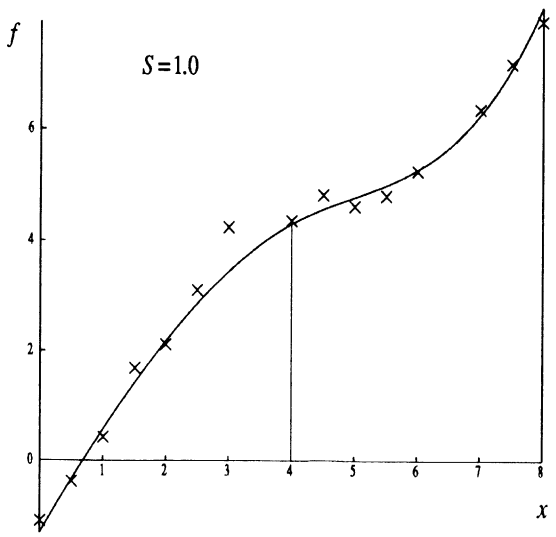
J	Knot K(J+2)	B-Spline Coefficient C(J)
1		-1.1072
2	0.0000	-0.6571
3	1.0000	0.4350
4	2.0000	2.8061
5	4.0000	4.6824
6	5.0000	4.6416
7	6.0000	5.1976
8	8.0000	6.9008
9		7.9979

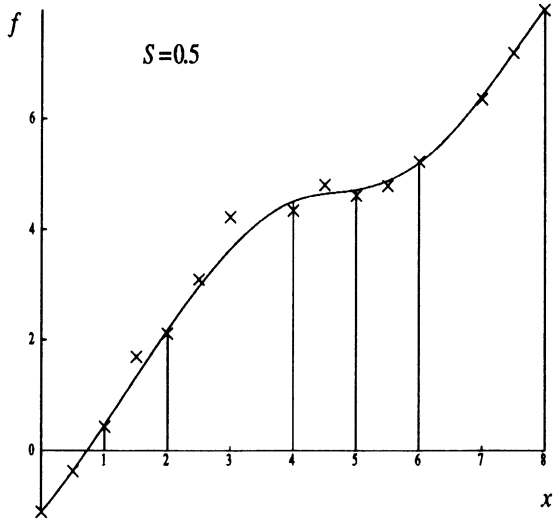
Weighted sum of squared residuals FP = 5.001E-01

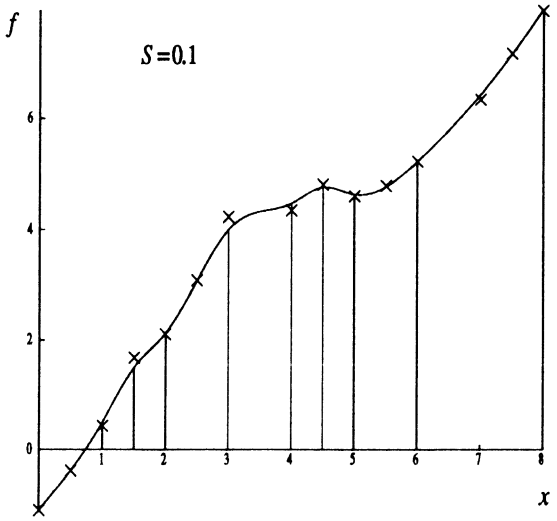
Calling with smoothing factor S = 1.000E-01

J	Knot K(J+2)	B-Spline Coefficient C(J)
1		-1.0900
2	0.0000	-0.6422
3	1.0000	0.0369
4	1.5000	1.6353
5	2.0000	2.1274
6	3.0000	4.5526
7	4.0000	4.2225
8	4.5000	4.9108
9	5.0000	4.4159
10	6.0000	5.4794
11	8.0000	6.8308
12		7.9935

Weighted sum of squared residuals FP = 1.000E-01




f
S=0.5
6
4
2
0
1
2
3
4
5
6
7
8
x


f
S=0.1
6
4
2
0
1
2
3
4
5
6
7
8
x

E02CAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02CAF forms an approximation to the weighted, least-squares Chebyshev-series surface fit to data arbitrarily distributed on lines parallel to one independent co-ordinate axis.

2. Specification

```
      SUBROUTINE E02CAF (M, N, K, L, X, Y, F, W, MTOT, A, NA, XMIN, XMAX,
     1                   NUX, INUXP1, NUY, INUYP1, WORK, NWORK, IFAIL)
      INTEGER          M(N), N, K, L, MTOT, NA, INUXP1, INUYP1, NWORK,
     1                 IFAIL
      real             X(MTOT), Y(N), F(MTOT), W(MTOT), A(NA), XMIN(N),
     1                 XMAX(N), NUX(INUXP1), NUY(INUYP1), WORK(NWORK)
```

3. Description

This subroutine determines a bivariate polynomial approximation of degree k in x and l in y to the set of data points $(x_{r,s},y_s f_{r,s})$, with weights $w_{r,s}$, for $s = 1,2,...,n$ and $r = 1,2,...,m_s$. That is, the data points are on lines $y = y_s$, but the x values may be different on each line. The values of k and l are prescribed by the user (for guidance on their choice, see Section 8). The subroutine is based on the method described in Clenshaw and Hayes [1], Sections 5 and 6.

The polynomial is represented in double Chebyshev-series form with arguments $\bar{x}$ and $\bar{y}$. The arguments lie in the range -1 to $+1$ and are related to the original variables x and y by the transformations

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{(x_{\max}-x_{\min})} \quad \text{and} \quad \bar{y} = \frac{2y - (y_{\max}+y_{\min})}{(y_{\max}-y_{\min})}.$$

Here $y_{\max}$ and $y_{\min}$ are set by the subroutine to, respectively, the largest and smallest value of y_s, but $x_{\max}$ and $x_{\min}$ are functions of y prescribed by the user (see Section 8). For this subroutine, only their values $x_{\max}^{(s)}$ and $x_{\min}^{(s)}$ at each $y = y_s$ are required. For each $s = 1,2,...,n$, $x_{\max}^{(s)}$ must not be less than the largest $x_{r,s}$ on the line $y = y_s$, and, similarly, $x_{\min}^{(s)}$ must not be greater than the smallest $x_{r,s}$.

The double Chebyshev-series can be written as

$$\sum_{i=0}^{k}\sum_{j=0}^{l} a_{ij}T_i(\bar{x})T_j(\bar{y})$$

where $T_i(\bar{x})$ is the Chebyshev polynomial of the first kind of degree i with argument $\bar{x}$, and $T_j(\bar{y})$ is similarly defined. However, the standard convention, followed in this subroutine, is that coefficients in the above expression which have either i or j zero are written as $\frac{1}{2}a_{ij}$, instead of simply a_{ij}, and the coefficient with both i and j = zero is written as $\frac{1}{4}a_{0,0}$. The series with coefficients output by the subroutine should be summed using this convention. E02CBF is available to compute values of the fitted function from these coefficients.

The subroutine first obtains Chebyshev-series coefficients $c_{s,i}$, for $i = 0,1,...,k$, of the weighted least-squares polynomial curve fit of degree k in $\bar{x}$ to the data on each line $y = y_s$, for $s = 1,2,..,n$ in turn, using an auxiliary subroutine. The same subroutine is then called $k + 1$ times to fit $c_{s,i}$, for $s = 1,2,...,n$ by a polynomial of degree l in $\bar{y}$, for each $i = 0,1,...,k$. The resulting coefficients are the required a_{ij}.

The fit can, at the option of the user, be forced to contain a given polynomial factor. This allows for the surface fit to be constrained to have specified values and derivatives along the boundaries $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$ and $y = y_{\max}$ or indeed along any lines $\bar{x}$ = constant or $\bar{y}$ = constant (see Clenshaw and Hayes [1], Section 8).

4. References

[1] CLENSHAW, C.W. and HAYES, J.G.
Curve and Surface Fitting.
J. Inst. Maths. Applics., 1, pp. 164-183, 1965.

[2] HAYES, J.G.
Curve fitting by polynomials in one variable.
In: Numerical Approximation to Functions and Data, J.G. Hayes (ed).
The Athlone Press, Ch. 5, 1970.

5. Parameters

1: M(N) – INTEGER array. *Input*

On entry: M(s) must be set to m_s, the number of data x values on the line $y = y_s$, for $s = 1,2,...,n$.

Constraint: M(s) > 0, for $s = 1,2,...,n$.

2: N – INTEGER. *Input*

On entry: the number of lines y = constant on which data points are given.

Constraint: N > 0.

3: K – INTEGER. *Input*

On entry: k, the required degree of x in the fit.

Constraint: for $s = 1,2,...,n$, INUXP1 − 1 ≤ K < MDIST(s) + INUXP1 − 1, where MDIST(s) is the number of distinct x values with non-zero weight on the line $y = y_s$. See Section 8, paragraph 3.

4: L – INTEGER. *Input*

On entry: l, the required degree of y in the fit.

Constraint: INUYP1 − 1 ≤ L < N + INUYP1 − 1.

5: X(MTOT) – ***real*** array. *Input*

On entry: the x values of the data points. The sequence must be

all points on $y = y_1$, followed by
all points on $y = y_2$, followed by
.
.
.
all points on $y = y_n$.

Constraint: for each y_s, the x values must be in non-decreasing order.

6: Y(N) – ***real*** array. *Input*

On entry: the y values of lines $y = y_s$, for $s = 1,2,...,n$ on which data is given.

Constraint: the y_s values must be in strictly increasing order.

7: F(MTOT) – ***real*** array. *Input*

On entry: the data values of the dependent variable, f, in the same sequence as the x values.

8: W(MTOT) – ***real*** array. *Input*

On entry: the weights to be assigned to the data points, in the same sequence as the x values. These weights should be calculated from estimates of the absolute accuracies of the f_r, expressed as standard deviations, probable errors or some other measure which is of the same dimensions as f_r. Specifically, each w_r should be inversely proportional to the

accuracy estimate of f_r. Often weights all equal to unity will be satisfactory. If a particular weight is zero, the corresponding data point is omitted from the fit.

9: MTOT – INTEGER. *Input*

On entry: the dimension of the arrays X, F and W as declared in the (sub)program from which E02CAF is called.

Constraint: MTOT $\geq \sum_{s=1}^{N} M(s)$.

10: A(NA) – ***real*** array. *Output*

On exit: A contains the Chebyshev coefficients of the fit. A($i\times$(L+1)+(j+1)) is the coefficient a_{ij} of Section 3 defined according to the standard convention. These coefficients are used by E02CBF to calculate values of the fitted function.

11: NA – INTEGER. *Input*

On entry: the dimension of the array A as declared in the (sub)program from which E02CAF is called.

Constraint: NA $\geq$ (K+1)$\times$(L+1), the total number of coefficients in the fit.

12: XMIN(N) – ***real*** array. *Input*

On entry: $x_{\min}^{(s)}$, the lower end of the range of x on the line $y = y_s$, for s = 1,2,...,n. It must not be greater than the lowest data value of x on the line. Each $x_{\min}^{(s)}$ is scaled to −1.0 in the fit. (See also Section 8.)

13: XMAX(N) – ***real*** array. *Input*

On entry: $x_{\max}^{(s)}$, the upper end of the range of x on the line $y = y_s$, for s = 1,2,...,n. It must not be less than the highest data value of x on the line. Each $x_{\max}^{(s)}$ is scaled to +1.0 in the fit. (See also Section 8.)

Constraint: XMAX(s) > XMIN(s).

14: NUX(INUXP1) – ***real*** array. *Input*

On entry: NUX(i) must contain the coefficient of the Chebyshev polynomial of degree (i−1) in $\bar{x}$, in the Chebyshev-series representation of the polynomial factor in $\bar{x}$ which the user requires the fit to contain, for i = 1,2,...,INUXP1. These coefficients are defined according to the standard convention of Section 3.

Constraint: NUX(INUXP1) must be non-zero, unless INUXP1 = 1, in which case NUX is ignored.

15: INUXP1 – INTEGER. *Input*

On entry: INUX + 1, where INUX is the degree of a polynomial factor in $\bar{x}$ which the user requires the fit to contain. (See Section 3, last paragraph.) If this option is not required, INUXP1 should be set equal to 1.

Constraint: 1 $\leq$ INUXP1 $\leq$ K + 1.

16: NUY(INUYP1) – ***real*** array. *Input*

On entry: NUY(i) must contain the coefficient of the Chebyshev polynomial of degree (i−1) in $\bar{y}$, in the Chebyshev-series representation of the polynomial factor which the user requires the fit to contain, for i = 1,2,...,INUYP1. These coefficients are defined according to the standard convention of Section 3.

Constraint: NUY(INUYP1) must be non-zero, unless INUYP1 = 1, in which case NUY is ignored.

17: INUYP1 – INTEGER. *Input*

On entry: INUY + 1, where INUY is the degree of a polynomial factor in $\bar{y}$ which the user requires the fit to contain. (See Section 3, last paragraph.) If this option is not required, INUYP1 should be set equal to 1.

18: WORK(NWORK) – ***real*** array. *Workspace*

19: NWORK – INTEGER. *Input*

On entry: the dimension of WORK as declared in the (sub)program from which E02CAF is called.

Constraint:
NWORK ≥ MTOT + 2×max(N,M(S)) + 2×N×(K+2) + 5×(1+max(K,L)).

20: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, K or L < 0,
or INUXP1 or INUYP1 < 1,
or INUXP1 > K + 1,
or INUYP1 > L + 1,
or M(i) < K − INUXP1 + 2 for some i = 1,2,...,N,
or N < L − INUYP1 + 2,
or NA is too small,
or NWORK is too small,
or MTOT is too small.

IFAIL = 2

XMIN(i) and XMAX(i) do not span the data X values on Y = Y(i) for some i = 1,2,...,N, possibly because XMIN(i) ≥ XMAX(i).

IFAIL = 3

The data X values on Y = Y(i) are not non-decreasing for some i = 1,2,...,N, or the Y(i) themselves are not strictly increasing.

IFAIL = 4

The number of distinct X values with non-zero weight on Y = Y(i) is less than K − INUXP1 + 2 for some i = 1,2,...,N.

IFAIL = 5

On entry, NUX(INUXP1) = 0 and INUXP1 ≠ 1,
or NUY(INUYP1) = 0 and INUYP1 ≠ 1.

7. Accuracy

No error analysis for this method has been published. Practical experience with the method, however, is generally extremely satisfactory.

8. Further Comments

The time taken by this routine is approximately proportional to $k\times(k\times\text{MTOT}+n\times l^2)$.

The reason for allowing $x_{\max}$ and $x_{\min}$ (which are used to normalise the range of x) to vary with y is that unsatisfactory fits can result if the highest (or lowest) data values of the normalised x on each line $y = y_s$ are not approximately the same. (For an explanation of this phenomenon, see Clenshaw and Hayes [1], page 176.) Commonly in practice, the lowest (for example) data values $x_{1,s}$, while not being approximately constant, do lie close to some smooth curve in the (x,y) plane. Using values from this curve as the values of $x_{\min}$, different in general on each line, causes the lowest transformed data values $\bar{x}_{1,s}$ to be approximately constant. Sometimes, appropriate curves for $x_{\max}$ and $x_{\min}$ will be clear from the context of the problem (they need not be polynomials). If this is not the case, suitable curves can often be obtained by fitting to the lowest data values $x_{1,s}$ and to the corresponding highest data values of x, low degree polynomials in y, using routine E02ADF, and then shifting the two curves outwards by a small amount so that they just contain all the data between them. The complete curves are not in fact supplied to the present subroutine, only their values at each y_s; and the values simply need to lie on smooth curves. More values on the complete curves will be required subsequently, when computing values of the fitted surface at arbitrary y values.

Naturally, a satisfactory approximation to the surface underlying the data cannot be expected if the character of the surface is not adequately represented by the data. Also, as always with polynomials, the approximating function may exhibit unwanted oscillations (particularly near the ends of the ranges) if the degrees k and l are taken greater than certain values, generally unknown but depending on the total number of coefficients $(k+1)\times(l+1)$ should be significantly smaller than, say not more than half, the total number of data points. Similarly, $k + 1$ should be significantly smaller than most (preferably all) the m_s, and $l + 1$ significantly smaller than n. Closer spacing of the data near the ends of the x and y ranges is an advantage. In particular, if $\bar{y}_s = -\cos(\pi(s-1)/(n-1))$, for $s = 1,2,...,n$ and $\bar{x}_{r,s} = -\cos(\pi(r-1)/(m-1))$, for $r = 1,2,...,m$, (thus $m_s = m$ for all s), then the values $k = m - 1$ and $l = n - 1$ (so that the polynomial passes exactly through all the data points) should not give unwanted oscillations. Other data sets should be similarly satisfactory if they are everywhere at least as closely spaced as the above cosine values with m replaced by $k + 1$ and n by $l + 1$ (more precisely, if for every s the largest interval between consecutive values of arc cos $\bar{x}_{r,s}$, for $r = 1,2,...,m$, is not greater than π/k, and similarly for the $\bar{y}_s$). The polynomial obtained should always be examined graphically before acceptance. Note that, for this purpose it is not sufficient to plot the polynomial only at the data values of x and y: intermediate values should also be plotted, preferably via a graphics facility.

Provided the data are adequate, and the surface underlying the data is of a form that can be represented by a polynomial of the chosen degrees, the subroutine should produce a good approximation to this surface. It is not, however, the true least-squares surface fit nor even a polynomial in x and y, the original variables (see Clenshaw and Hayes [1], Section 6), except in certain special cases. The most important of these is where the data values of x are the same on each line $y = y_s$, (i.e. the data points lie on a rectangular mesh in the (x,y) plane), the weights of the data points are all equal, and $x_{\max}$ and $x_{\min}$ are both constants (in this case they should be set to the largest and smallest data values of x, respectively).

If the data set is such that it can be satisfactorily approximated by a polynomial of degrees k' and l', say, then if higher values are used for k and l in the subroutine, all the coefficients a_{ij} for $i > k'$ or $j > l'$ will take apparently random values within a range bounded by the size of the data errors, or rather less. (This behaviour of the Chebyshev coefficients, most readily observed if they are set out in a rectangular array, closely parallels that in curve fitting, examples of which are given in Hayes [2], Section 8.) In practice, therefore, to establish suitable values of k' and l', the user should first be seeking (within the limitations discussed above) values for k and l which are large enough to exhibit the behaviour described. Values for k' and l' should then be chosen as the smallest which do not exclude any coefficients significantly larger than the random ones. A polynomial of degrees k' and l' should then be fitted to the data.

If the option to force the fit to contain a given polynomial factor in x is used and if zeros of the chosen factor coincide with data x values on any line, then the effective number of data points on

that line is reduced by the number of such coincidences. A similar consideration applies when forcing the y direction. No account is taken of this by the subroutine when testing that the degrees k and l have not been chosen too large.

9. Example

The example program reads data in the following order, using the notation of the parameter list for E02CAF above:

N K L
Y(i) M(i) XMIN(i) XMAX(i) for i = 1,2,...,N.
X(i) F(i) W(i) for i = 1,2,...,MTOT.

The data points are fitted using E02CAF, and then the fitting polynomial is evaluated at the data points using E02CBF.

The output is:

the data points and their fitted values
the Chebyshev coefficients of the fit.

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.

```
*     E02CAF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, MTMAX, NA, NWORK
      PARAMETER        (NMAX=20,MTMAX=400,NA=100,NWORK=1500)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             YMAX
      INTEGER          I, IFAIL, J, K, L, MI, MJ, N, R, T
*     .. Local Arrays ..
      real             A(NA), F(MTMAX), FF(MTMAX), W(MTMAX),
     +                 WORK(NWORK), X(MTMAX), XMAX(NMAX), XMIN(NMAX),
     +                 Y(NMAX)
      INTEGER          M(20)
*     .. External Subroutines ..
      EXTERNAL         E02CAF, E02CBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02CAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of lines Y = Y(I) on which data is given,
*     and the required degree of fit in the X and Y directions
   20 READ (NIN,*,END=120) N, K, L
      WRITE (NOUT,*)
      IF (N.GT.0 .AND. N.LE.NMAX) THEN
         MJ = 0
*        Input Y(I), the number of data points on Y = Y(I) and the
*        range of X-values on this line, for I = 1,2,...N
         DO 40 I = 1, N
            READ (NIN,*) Y(I), MI, XMIN(I), XMAX(I)
            M(I) = MI
            MJ = MJ + MI
   40    CONTINUE
*        Terminate program if the arrays have not been declared
*        large enough to contain the data
         IF (MTMAX.LT.MJ) THEN
            WRITE (NOUT,99999)
     +        'MTOT is too small. It should be at least ', MJ
            STOP
         END IF
```

```
*        Input the X-values and function values, F, together with
*        their weights, W.
         READ (NIN,*) (X(I),F(I),W(I),I=1,MJ)
*        Evaluate the coefficients, A, of the fit to this set of data
         IFAIL = 0
*
         CALL E02CAF(M,N,K,L,X,Y,F,W,MTMAX,A,NA,XMIN,XMAX,Y,1,Y,1,WORK,
     +               NWORK,IFAIL)
*
         MI = 0
         WRITE (NOUT,*)
     +     '     Data Y     Data X     Data F   Fitted F   Residual'
         WRITE (NOUT,*)
         DO 80 R = 1, N
            T = MI + 1
            MI = MI + M(R)
            YMAX = Y(N)
            IF (N.EQ.1) YMAX = YMAX + 1.0e0
*           Evaluate the fitted polynomial at each of the data points
*           on the line Y = Y(R)
            IFAIL = 0
*
            CALL E02CBF(T,MI,K,L,X,XMIN(R),XMAX(R),Y(R),Y(1),YMAX,FF,A,
     +                  NA,WORK,NWORK,IFAIL)
*
*           Output the data and fitted values on the line Y = Y(R)
            DO 60 I = T, MI
               WRITE (NOUT,99998) Y(R), X(I), F(I), FF(I), FF(I) - F(I)
   60       CONTINUE
            WRITE (NOUT,*)
   80    CONTINUE
*        Output the Chebyshev coefficients of the fit
         WRITE (NOUT,*) 'Chebyshev coefficients of the fit'
         WRITE (NOUT,*)
         DO 100 J = 1, K + 1
            WRITE (NOUT,99997) (A(I),I=1+(J-1)*(L+1),J*(L+1))
  100    CONTINUE
         GO TO 20
      END IF
  120 STOP
*
99999 FORMAT (1X,A,I5)
99998 FORMAT (1X,4F11.4,e11.2)
99997 FORMAT (1X,6F11.4)
      END
```

9.2. Program Data

```
E02CAF Example Program Data
    4    3    2
         0.0      8          0.0            5.0
         1.0      7          0.1            4.5
         2.0      7          0.4            4.0
         4.0      6          1.6            3.5
         0.1        1.01005          1.0
         1.0        1.10517          1.0
         1.6        1.17351          1.0
         2.1        1.23368          1.0
         3.3        1.39097          1.0
         3.9        1.47698          1.0
         4.2        1.52196          1.0
         4.9        1.63232          1.0
         0.1        2.02010          1.0
         1.1        2.23256          1.0
         1.9        2.41850          1.0
         2.7        2.61993          1.0
         3.2        2.75426          1.0
         4.1        3.01364          1.0
         4.5        3.13662          1.0
         0.5        3.15381          1.0
```

```
1.1         3.34883         1.0
1.3         3.41649         1.0
2.2         3.73823         1.0
2.9         4.00928         1.0
3.5         4.25720         1.0
3.9         4.43094         1.0
1.7         5.92652         1.0
2.0         6.10701         1.0
2.4         6.35625         1.0
2.7         6.54982         1.0
3.1         6.81713         1.0
3.5         7.09534         1.0
```

9.3. Program Results

```
E02CAF Example Program Results

    Data Y      Data X      Data F    Fitted F    Residual

    0.0000      0.1000      1.0100      1.0175    0.74E-02
    0.0000      1.0000      1.1052      1.1126    0.74E-02
    0.0000      1.6000      1.1735      1.1809    0.74E-02
    0.0000      2.1000      1.2337      1.2412    0.75E-02
    0.0000      3.3000      1.3910      1.3992    0.82E-02
    0.0000      3.9000      1.4770      1.4857    0.87E-02
    0.0000      4.2000      1.5220      1.5310    0.90E-02
    0.0000      4.9000      1.6323      1.6422    0.98E-02

    1.0000      0.1000      2.0201      1.9987   -0.21E-01
    1.0000      1.1000      2.2326      2.2110   -0.22E-01
    1.0000      1.9000      2.4185      2.3962   -0.22E-01
    1.0000      2.7000      2.6199      2.5966   -0.23E-01
    1.0000      3.2000      2.7543      2.7299   -0.24E-01
    1.0000      4.1000      3.0136      2.9869   -0.27E-01
    1.0000      4.5000      3.1366      3.1084   -0.28E-01

    2.0000      0.5000      3.1538      3.1700    0.16E-01
    2.0000      1.1000      3.3488      3.3648    0.16E-01
    2.0000      1.3000      3.4165      3.4325    0.16E-01
    2.0000      2.2000      3.7382      3.7549    0.17E-01
    2.0000      2.9000      4.0093      4.0272    0.18E-01
    2.0000      3.5000      4.2572      4.2769    0.20E-01
    2.0000      3.9000      4.4309      4.4521    0.21E-01

    4.0000      1.7000      5.9265      5.9231   -0.34E-02
    4.0000      2.0000      6.1070      6.1036   -0.34E-02
    4.0000      2.4000      6.3563      6.3527   -0.35E-02
    4.0000      2.7000      6.5498      6.5462   -0.36E-02
    4.0000      3.1000      6.8171      6.8132   -0.40E-02
    4.0000      3.5000      7.0953      7.0909   -0.45E-02

Chebyshev coefficients of the fit

   15.3482      5.1507      0.1014
    1.1472      0.1442     -0.1046
    0.0490     -0.0031     -0.0070
    0.0015     -0.0003     -0.0002
```

E02CBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02CBF evaluates a bivariate polynomial from the rectangular array of coefficients in its double Chebyshev-series representation.

2. Specification

```
      SUBROUTINE E02CBF (MFIRST, MLAST, K, L, X, XMIN, XMAX, Y, YMIN,
     1                   YMAX, FF, A, NA, WORK, NWORK, IFAIL)
      INTEGER          MFIRST, MLAST, K, L, NA, NWORK, IFAIL
      real             X(MLAST), XMIN, XMAX, Y, YMIN, YMAX, FF(MLAST),
     1                 A(NA), WORK(NWORK)
```

3. Description

This subroutine evaluates a bivariate polynomial (represented in double Chebyshev form) of degree k in one variable, $\bar{x}$, and degree l in the other, $\bar{y}$. The range of both variables is -1 to $+1$. However, these normalised variables will usually have been derived (as when the polynomial has been computed by E02CAF, for example) from the user's original variables x and y by the transformations

$$\bar{x} = \frac{2x - (x_{\max}+x_{\min})}{(x_{\max}-x_{\min})} \quad \text{and} \quad \bar{y} = \frac{2y - (y_{\max}+y_{\min})}{(y_{\max}-y_{\min})}.$$

(Here $x_{\min}$ and $x_{\max}$ are the ends of the range of x which has been transformed to the range -1 to $+1$ of $\bar{x}$. $y_{\min}$ and $y_{\max}$ are correspondingly for y. See Section 8.) For this reason, the subroutine has been designed to accept values of x and y rather than $\bar{x}$ and $\bar{y}$, and so requires values of $x_{\min}$, etc. to be supplied by the user. In fact, for the sake of efficiency in appropriate cases, the routine evaluates the polynomial for a sequence of values of x, all associated with the same value of y.

The double Chebyshev-series can be written as

$$\sum_{i=0}^{k}\sum_{j=0}^{l} a_{ij}T_i(\bar{x})T_j(\bar{y}),$$

where $T_i(\bar{x})$ is the Chebyshev polynomial of the first kind of degree i and argument $\bar{x}$, and $T_j(\bar{y})$ is similarly defined. However the standard convention, followed in this subroutine, is that coefficients in the above expression which have either i or j zero are written $\frac{1}{2}a_{ij}$, instead of simply a_{ij}, and the coefficient with both i and j zero is written $\frac{1}{4}a_{0,0}$.

The subroutine first forms $c_i = \sum_{j=0}^{l} a_{ij}T_j(\bar{y})$, with $a_{i,0}$ replaced by $\frac{1}{2}a_{i,0}$, for each of $i = 0,1,...,k$. The value of the double series is then obtained for each value of x, by summing $c_i \times T_i(\bar{x})$, with c_0 replaced by $\frac{1}{2}c_0$, over $i = 0,1,...,k$. The Clenshaw three term recurrence [1] with modifications due to Reinsch and Gentleman [2] is used to form the sums.

4. References

[1] CLENSHAW, C.W.
A Note on the Summation of Chebyshev-series.
Math. Tables Aids Comput., 9, pp. 118-120, 1955.

[2] GENTLEMAN, W.M.
An Error Analysis of Goertzel's (Watt's) Method for Computing Fourier Coefficients.
Comput. J., 12, pp. 160-165, 1969.

5. Parameters

1: MFIRST – INTEGER. *Input*
2: MLAST – INTEGER. *Input*

On entry: the index of the first and last x value in the array x at which the evaluation is required respectively (see Section 8).

Constraint: MLAST ≥ MFIRST.

3: K – INTEGER. *Input*
4: L – INTEGER. *Input*

On entry: the degree k of x and l of y, respectively, in the polynomial.

Constraint: K and L ≥ 0.

5: X(MLAST) – ***real*** array. *Input*

On entry: X(i), for i = MFIRST,MFIRST+1,...,MLAST, must contain the x values at which the evaluation is required.

Constraint: XMIN ≤ X(i) ≤ XMAX, for all i.

6: XMIN – ***real***. *Input*
7: XMAX – ***real***. *Input*

On entry: the lower and upper ends, $x_{\min}$ and $x_{\max}$, of the range of the variable x (see Section 3).

The values of XMIN and XMAX may depend on the value of y (e.g. when the polynomial has been derived using E02CAF).

Constraint: XMAX > XMIN.

8: Y – ***real***. *Input*

On entry: the value of the y co-ordinate of all the points at which the evaluation is required.

Constraint: YMIN ≤ Y ≤ YMAX.

9: YMIN – ***real***. *Input*
10: YMAX – ***real***. *Input*

On entry: the lower and upper ends, $y_{\min}$ and $y_{\max}$, of the range of the variable y (see Section 3).

Constraint: YMAX > YMIN.

11: FF(MLAST) – ***real*** array. *Output*

On exit: FF(i) gives the value of the polynomial at the point (x_i,y), for i = MFIRST,MFIRST+1,...,MLAST.

12: A(NA) – ***real*** array. *Input*

On entry: the Chebyshev coefficients of the polynomial. The coefficient a_{ij} defined according to the standard convention (see Section 3) must be in A($i\times(l+1)+j+1$).

13: NA – INTEGER. *Input*

On entry: the dimension of the array A as declared in the (sub)program from which E02CBF is called.

Constraint: NA ≥ (K+1)×(L+1), the number of coefficients in a polynomial of the specified degree.

14: WORK(NWORK) – ***real*** array. *Workspace*

15: NWORK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which E02CBF is called.

Constraint: NWORK ≥ K + 1.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, MFIRST > MLAST,
or K < 0,
or L < 0,
or NA < (K+1)×(L+1),
or NWORK < K + 1.

IFAIL = 2

On entry, YMIN ≥ YMAX,
or Y < YMIN,
or Y > YMAX.

IFAIL = 3

On entry, XMIN ≥ XMAX,
or X(i) < XMIN, or X(i) > XMAX, for some i = MFIRST,MFIRST+1,..., MLAST.

7. Accuracy

The method is numerically stable in the sense that the computed values of the polynomial are exact for a set of coefficients which differ from those supplied by only a modest multiple of ***machine precision***.

8. Further Comments

The time taken by this routine is approximately proportional to $(k+1)\times(m+l+1)$, where m = MLAST – MFIRST + 1, the number of points at which the evaluation is required.

This subroutine is suitable for evaluating the polynomial surface fits produced by the subroutine E02CAF, which provides the ***real*** array A in the required form. For this use, the values of $y_{\min}$ and $y_{\max}$ supplied to the present subroutine must be the same as those supplied to E02CAF. The same applies to $x_{\min}$ and $x_{\max}$ if they are independent of y. If they vary with y, their values must be consistent with those supplied to E02CAF (see Section 8 of document E02CAF).

The parameters MFIRST and MLAST are intended to permit the selection of a segment of the array X which is to be associated with a particular value of y, when, for example, other segments of X are associated with other values of y. Such a case arises when, after using E02CAF to fit a set of data, the user wishes to evaluate the resulting polynomial at all the data values. In this case, if the parameters X, Y, MFIRST and MLAST of the present routine are set respectively (in terms of parameters of E02CAF) to X, Y(S), $1 + \sum_{i=1}^{S-1} \mathrm{M}(i)$ and $\sum_{i=1}^{S} \mathrm{M}(i)$, the routine will compute values of the polynomial surface at all data points which have Y(S) as their y co-ordinate (from which values the residuals of the fit may be derived).

9. Example

The example program reads data in the following order, using the notation of the parameter list above:

N K L
A(i), for i = 1,2,...,(K+1)×(L+1)
YMIN YMAX
Y(i) M(i) XMIN(i) XMAX(i) X1(i) XM(i), for i = 1,2,...,N.

For each line Y = Y(i) the polynomial is evaluated at M(i) equispaced points between X1(i) and XM(i) inclusive.

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.

```
*     E02CBF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, KMAX, NWORK, LMAX, NA
      PARAMETER        (MMAX=100,KMAX=9,NWORK=KMAX+1,LMAX=9,NA=(KMAX+1)
     +                 *(LMAX+1))
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             X1, XM, XMAX, XMIN, Y, YMAX, YMIN
      INTEGER          I, IFAIL, J, K, L, M, N, NCOEF
*     .. Local Arrays ..
      real             A(NA), FF(MMAX), WORK(NWORK), X(MMAX)
*     .. External Subroutines ..
      EXTERNAL         E02CBF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02CBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) N, K, L
      IF (K.LE.KMAX .AND. L.LE.LMAX) THEN
         NCOEF = (K+1)*(L+1)
         READ (NIN,*) (A(I),I=1,NCOEF)
         READ (NIN,*) YMIN, YMAX
         DO 80 I = 1, N
            READ (NIN,*) Y, M, XMIN, XMAX, X1, XM
            IF (M.LE.MMAX) THEN
               DO 40 J = 1, M
                  X(J) = X1 + ((XM-X1)*real(J-1))/real(M-1)
   40          CONTINUE
               IFAIL = 0
*
               CALL E02CBF(1,M,K,L,X,XMIN,XMAX,Y,YMIN,YMAX,FF,A,NA,WORK,
     +                     NWORK,IFAIL)
*
               WRITE (NOUT,*)
               WRITE (NOUT,99999) 'Y = ', Y
               WRITE (NOUT,*)
               WRITE (NOUT,*) '   I      X(I)       Poly(X(I),Y)'
               DO 60 J = 1, M
                  WRITE (NOUT,99998) J, X(J), FF(J)
   60          CONTINUE
            END IF
   80    CONTINUE
```

```
            GO TO 20
         END IF
  100 STOP
*
99999 FORMAT (1X,A,e13.4)
99998 FORMAT (1X,I3,1P,2e13.4)
      END
```

9.2. Program Data

```
E02CBF Example Program Data
   3   3   2
15.34820
 5.15073
 0.10140
 1.14719
 0.14419
-0.10464
 0.04901
-0.00314
-0.00699
 0.00153
-0.00033
-0.00022
   0.0          4.0
   1.0          9   0.1          4.5          0.5          4.5
   1.5          8   0.225        4.25         0.5          4.0
   2.0          8   0.4          4.0          0.5          4.0
```

9.3. Program Results

```
 E02CBF Example Program Results

 Y =    0.1000E+01

   I      X(I)       Poly(X(I),Y)
   1   5.0000E-01     2.0812E+00
   2   1.0000E+00     2.1888E+00
   3   1.5000E+00     2.3018E+00
   4   2.0000E+00     2.4204E+00
   5   2.5000E+00     2.5450E+00
   6   3.0000E+00     2.6758E+00
   7   3.5000E+00     2.8131E+00
   8   4.0000E+00     2.9572E+00
   9   4.5000E+00     3.1084E+00

 Y =    0.1500E+01

   I      X(I)       Poly(X(I),Y)
   1   5.0000E-01     2.6211E+00
   2   1.0000E+00     2.7553E+00
   3   1.5000E+00     2.8963E+00
   4   2.0000E+00     3.0444E+00
   5   2.5000E+00     3.2002E+00
   6   3.0000E+00     3.3639E+00
   7   3.5000E+00     3.5359E+00
   8   4.0000E+00     3.7166E+00
```

```
Y =    0.2000E+01

  I      X(I)       Poly(X(I),Y)
  1   5.0000E-01     3.1700E+00
  2   1.0000E+00     3.3315E+00
  3   1.5000E+00     3.5015E+00
  4   2.0000E+00     3.6806E+00
  5   2.5000E+00     3.8692E+00
  6   3.0000E+00     4.0678E+00
  7   3.5000E+00     4.2769E+00
  8   4.0000E+00     4.4971E+00
```

E02DAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02DAF forms a minimal, weighted least-squares bicubic spline surface fit with prescribed knots to a given set of data points.

2. Specification

```
      SUBROUTINE E02DAF (M, PX, PY, X, Y, F, W, LAMDA, MU, POINT, NPOINT,
     1                   DL, C, NC, WS, NWS, EPS, SIGMA, RANK, IFAIL)
      INTEGER          M, PX, PY, POINT(NPOINT), NPOINT, NC, NWS, RANK,
     1                 IFAIL
      real             X(M), Y(M), F(M), W(M), LAMDA(PX), MU(PY), DL(NC),
     1                 C(NC), WS(NWS), EPS, SIGMA
```

3. Description

This routine determines a bicubic spline fit $s(x,y)$ to the set of data points (x_r,y_r,f_r) with weights w_r, for $r = 1,2,...,m$. The two sets of internal knots of the spline, $\{\lambda\}$ and $\{\mu\}$, associated with the variables x and y respectively, are prescribed by the user. These knots can be thought of as dividing the data region of the (x,y) plane into panels (see diagram in Section 5). A bicubic spline consists of a separate bicubic polynomial in each panel, the polynomials joining together with continuity up to the second derivative across the panel boundaries.

$s(x,y)$ has the property that Σ, the sum of squares of its weighted residuals ρ_r, for $r = 1,2,...,m$, where

$$\rho_r = w_r(s(x_r,y_r)-f_r), \tag{1}$$

is as small as possible for a bicubic spline with the given knot sets. The routine produces this minimized value of Σ and the coefficients c_{ij} in the B-spline representation of $s(x,y)$ - see Section 8. E02DEF and E02DFF are available to compute values of the fitted spline from the coefficients c_{ij}.

The least-squares criterion is not always sufficient to determine the bicubic spline uniquely: there may be a whole family of splines which have the same minimum sum of squares. In these cases, the routine selects from this family the spline for which the sum of squares of the coefficients c_{ij} is smallest: in other words, the minimal least-squares solution. This choice, although arbitrary, reduces the risk of unwanted fluctuations in the spline fit. The method employed involves forming a system of m linear equations in the coefficients c_{ij} and then computing its least-squares solution, which will be the minimal least-squares solution when appropriate. The basis of the method is described in Hayes and Halliday [4]. The matrix of the equation is formed using a recurrence relation for B-splines which is numerically stable (see Cox [1] and de Boor [2] – the former contains the more elementary derivation but, unlike [2], does not cover the case of coincident knots). The least-squares solution is also obtained in a stable manner by using orthogonal transformations, viz. a variant of Givens rotation (see Gentleman [3]). This requires only one row of the matrix to be stored at a time. Advantage is taken of the stepped-band structure which the matrix possesses when the data points are suitably ordered, there being at most sixteen non-zero elements in any row because of the definition of B-splines. First the matrix is reduced to upper triangular form and then the diagonal elements of this triangle are examined in turn. When an element is encountered whose square, divided by the mean squared weight, is less than a threshold ε, it is replaced by zero and the rest of the elements in its row are reduced to zero by rotations with the remaining rows. The rank of the system is taken to be the number of non-zero diagonal elements in the final triangle, and the non-zero rows of this triangle are used to compute the minimal least-squares solution. If all the diagonal elements are non-zero, the rank is equal to the number of coefficients c_{ij} and the solution obtained is the ordinary least-squares solution, which is unique in this case.

4. References

[1] COX, M.G.
The numerical evaluation of B-splines.
J. Inst. Maths. Applics., 10, pp. 134-149, 1972.

[2] DE BOOR, C.
On calculating with B-splines.
J. Approx. Theory, 6, pp. 50-62, 1972.

[3] GENTLEMAN, W.M.
Least squares computations by Givens transformations without square roots.
J. Inst. Maths. Applics., 12, pp. 329-336, 1973.

[4] HAYES, J.G. and HALLIDAY, J.
The least-squares fitting of cubic spline surfaces to general data sets.
J. Inst. Maths. Applics., 14, pp. 89-103, 1974.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of data points, m.

Constraint: M > 1.

2: PX – INTEGER. *Input*

3: PY – INTEGER. *Input*

On entry: the total number of knots λ and μ associated with the variables x and y, respectively.

Constraint: PX ≥ 8 and PY ≥ 8.

(They are such that PX−8 and PY−8 are the corresponding numbers of interior knots.) The running time and storage required by the routine are both minimized if the axes are labelled so that PY is the smaller of PX and PY.

4: X(M) – ***real*** array. *Input*

5: Y(M) – ***real*** array. *Input*

6: F(M) – ***real*** array. *Input*

On entry: the co-ordinates of the data point (x_r,y_r,f_r), for $r = 1,2,...,m$. The order of the data points is immaterial, but see the array POINT, below.

7: W(M) – ***real*** array. *Input*

On entry: the weight w_r of the rth data point. It is important to note the definition of weight implied by the equation (1) in Section 3, since it is also common usage to define weight as the square of this weight. In this routine, each w_r should be chosen inversely proportional to the (absolute) accuracy of the corresponding f_r, as expressed, for example, by the standard deviation or probable error of the f_r. When the f_r are all of the same accuracy, all the w_r may be set equal to 1.0.

8: LAMDA(PX) – ***real*** array. *Input/Output*

On entry: LAMDA(i+4) must contain the ith interior knot λ_{i+4} associated with the variable x, for $i = 1,2,...,$PX−8. The knots must be in non-decreasing order and lie strictly within the range covered by the data values of x. A knot is a value of x at which the spline is allowed to be discontinuous in the third derivative with respect to x, though continuous up to the second derivative. This degree of continuity can be reduced, if the user requires, by the use of coincident knots, provided that no more than four knots are chosen to coincide at any point. Two, or three, coincident knots allow loss of continuity in, respectively, the second and first derivative with respect to x at the value of x at which they coincide. Four coincident knots split the spline surface into two independent parts. For choice of knots see Section 8.

On exit: the interior knots LAMDA(5) to LAMDA(PX–4) are unchanged, and the segments LAMDA(1:4) and LAMDA(PX–3:PX) contain additional (exterior) knots introduced by the routine in order to define the full set of B-splines required. The four knots in the first segment are all set equal to the lowest data value of x and the other four additional knots are all set equal to the highest value: there is experimental evidence that coincident end-knots are best for numerical accuracy. The complete array must be left undisturbed if E02DEF or E02DFF is to be used subsequently.

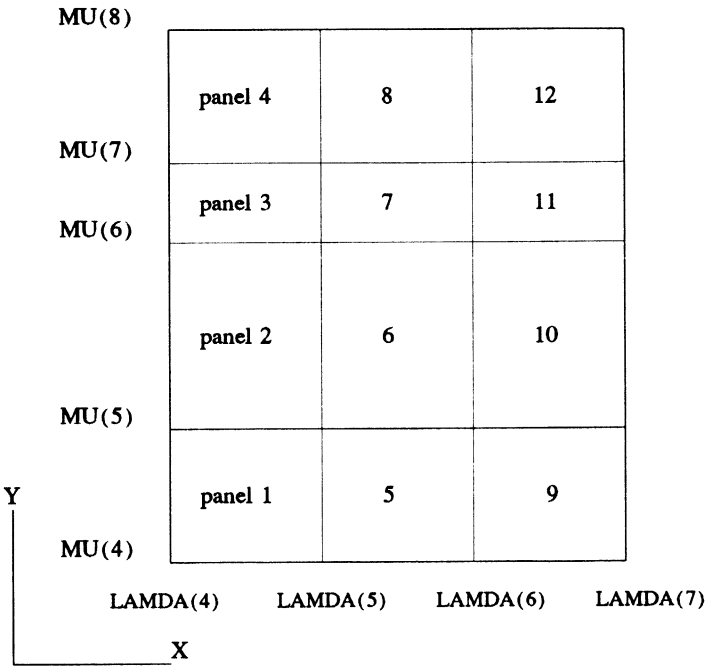
9: MU(PY) – ***real*** array. *Input/Output*

On entry: MU(i+4) must contain the ith interior knot μ_{i+4} associated with the variable y, i = 1,2,...,PY–8.

On exit: the same remarks apply to MU as to LAMDA above, with Y replacing X, and y replacing x.

10: POINT(NPOINT) – INTEGER array. *Input*

On entry: indexing information usually provided by E02ZAF which enables the data points to be accessed in the order which produces the advantageous matrix structure mentioned in Section 3. This order is such that, if the (x,y) plane is thought of as being divided into rectangular panels by the two sets of knots, all data in a panel occur before data in succeeding panels, where the panels are numbered from bottom to top and then left to right with the usual arrangement of axes, as indicated in the diagram.



A data point lying exactly on one or more panel sides is considered to be in the highest numbered panel adjacent to the point. E02ZAF should be called to obtain the array POINT, unless it is provided by other means.

11: NPOINT – INTEGER. *Input*

On entry: the dimension of the array POINT as declared in the (sub)program from which E02DAF is called.

Constraint: NPOINT $\geq$ M + (PX–7)$\times$(PY–7).

12: DL(NC) – ***real*** array. *Output*

On exit: DL gives the squares of the diagonal elements of the reduced triangular matrix, divided by the mean squared weight. It includes those elements, less than ϵ, which are treated as zero (see Section 3).

13: C(NC) – ***real*** array. *Output*

On exit: C gives the coefficients of the fit. C((PY−4)×(i−1)+j) is the coefficient c_{ij} of Sections 3 and 8 for i = 1,2,...,PX−4 and j = 1,2,...,PY−4. These coefficients are used by E02DEF or E02DFF to calculate values of the fitted function.

14: NC – INTEGER. *Input*

On entry: the value (PX−4)×(PY−4).

15: WS(NWS) – ***real*** array. *Workspace*

16: NWS – INTEGER. *Input*

On entry: the dimension of the array WS as declared in the (sub)program from which E02DAF is called.

Constraint: NWS ≥ (2×NC+1)×(3×PY−6)−2.

17: EPS – ***real***. *Input*

On entry: a threshold ε for determining the effective rank of the system of linear equations. The rank is determined as the number of elements of the array DL (see below) which are non-zero. An element of DL is regarded as zero if it is less than ε. ***Machine precision*** is a suitable value for ε in most practical applications which have only 2 or 3 decimals accurate in data. If some coefficients of the fit prove to be very large compared with the data ordinates, this suggests that ε should be increased so as to decrease the rank. The array DL will give a guide to appropriate values of ε to achieve this, as well as to the choice of ε in other cases where some experimentation may be needed to determine a value which leads to a satisfactory fit.

18: SIGMA – ***real***. *Output*

On exit: Σ, the weighted sum of squares of residuals. This is not computed from the individual residuals but from the right-hand sides of the orthogonally-transformed linear equations. For further details see Hayes and Halliday [4] page 97. The two methods of computation are theoretically equivalent, but the results may differ because of rounding error.

19: RANK – INTEGER. *Output*

On exit: the rank of the system as determined by the value of the threshold ε. When RANK = NC, the least-squares solution is unique: in other cases the minimal least-squares solution is computed.

20: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one set of knots is not in non-decreasing order, or an interior knot is outside the range of the data values.

IFAIL = 2

More than four knots coincide at a single point, possibly because all data points have the same value of x (or y) or because an interior knot coincides with an extreme data value.

IFAIL = 3

Array POINT does not indicate the data points in panel order. Call E02ZAF to obtain a correct array.

IFAIL = 4

On entry, M ≤ 1,
or PX < 8,
or PY < 8,
or NC ≠ (PX−4)×(PY−4),
or NWS is too small,
or NPOINT is too small.

IFAIL = 5

All the weights w_r are zero or rank determined as zero.

7. Accuracy

The computation of the B-splines and reduction of the observation matrix to triangular form are both numerically stable.

8. Further Comments

The time taken by this routine is approximately proportional to the number of data points, m, and to $(3\times(\text{PY}-4)+4)^2$.

The B-spline representation of the bicubic spline is

$$s(x,y) = \sum_{i,j} c_{ij} M_i(x) N_j(y)$$

summed over $i = 1,2,...,\text{PX}-4$ and over $j = 1,2,...,\text{PY}-4$. Here $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots $\lambda_i,\lambda_{i+1},...,\lambda_{i+4}$ and the latter on the knots $\mu_j,\mu_{j+1},...,\mu_{j+4}$. For further details, see Hayes and Halliday [4] for bicubic splines and de Boor [2] for normalised B-splines.

The choice of the interior knots, which help to determine the spline's shape, must largely be a matter of trial and error. It is usually best to start with a small number of knots and, examining the fit at each stage, add a few knots at a time at places where the fit is particularly poor. In intervals of x or y where the surface represented by the data changes rapidly, in function value or derivatives, more knots will be needed than elsewhere. In some cases guidance can be obtained by analogy with the case of coincident knots: for example, just as three coincident knots can produce a discontinuity in slope, three close knots can produce rapid change in slope. Of course, such rapid changes in behaviour must be adequately represented by the data points, as indeed must the behaviour of the surface generally, if a satisfactory fit is to be achieved. When there is no rapid change in behaviour, equally-spaced knots will often suffice.

In all cases the fit should be examined graphically before it is accepted as satisfactory.

The fit obtained is not defined outside the rectangle

$$\lambda_4 \le x \le \lambda_{\text{PX}-3},\ \mu_4 \le y \le \mu_{\text{PY}-3}$$

The reason for taking the extreme data values of x and y for these four knots is that, as is usual in data fitting, the fit cannot be expected to give satisfactory values outside the data region. If, nevertheless, the user requires values over a larger rectangle, this can be achieved by augmenting the data with two artificial data points $(a,c,0)$ and $(b,d,0)$ with zero weight, where $a \le x \le b$, $c \le y \le d$ defines the enlarged rectangle. In the case when the data are adequate to make the least-squares solution unique (RANK = NC), this enlargement will not affect the fit over the original rectangle, except for possibly enlarged rounding errors, and will simply continue the bicubic polynomials in the panels bordering the rectangle out to the new boundaries: in other cases the fit will be affected. Even using the original rectangle there may be regions within it, particularly at its corners, which lie outside the data region and where, therefore, the fit will be unreliable. For example, if there is no data point in panel 1 of the diagram in Section 5, the

least-squares criterion leaves the spline indeterminate in this panel: the minimal spline determined by the subroutine in this case passes through the value zero at the point (λ_4,μ_4).

9. Example

This example program reads a value for ϵ, and a set of data points, weights and knot positions. If there are more y knots than x knots, it interchanges the x and y axes. It calls E02ZAF to sort the data points into panel order, E02DAF to fit a bicubic spline to them, and E02DEF to evaluate the spline at the data points.

Finally it prints:

the weighted sum of squares of residuals computed from the linear equations;
the rank determined by E02DAF;
data points, fitted values and residuals in panel order;
the weighted sum of squares of the residuals;
the coefficients of the spline fit.

The program is written to handle any number of data sets.

Note: the data supplied in this example is **not typical** of a realistic problem: the number of data points would normally be much larger (in which case the array dimensions and the value of NWS in the program would have to be increased); and the value of ϵ would normally be much smaller on most machines (see Section 5; the relatively large value of 10^{-6} has been chosen in order to illustrate a minimal least-squares solution when RANK < NC; in this example NC = 24).

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.

```
*     E02DAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, MAXPX, MAXPY, NCMAX, IP, NIWS, NWS, NADRES,
     +                 NPTMAX
      PARAMETER        (MMAX=40,MAXPX=10,MAXPY=10,NCMAX=(MAXPX-4)
     +                 *(MAXPY-4),IP=3*(MAXPY-4)+4,NIWS=MAXPY-4,
     +                 NWS=2*NCMAX*(IP+2)+IP,NADRES=(MAXPX-7)*(MAXPY-7),
     +                 NPTMAX=MMAX+NADRES)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             EPS, SIGMA, SUM, TEMP
      INTEGER          I, IADRES, IFAIL, ITEMP, J, M, NC, NP, PX, PY,
     +                 RANK
*     .. Local Arrays ..
      real             C(NCMAX), DL(NCMAX), F(MMAX), FF(MMAX),
     +                 LAMDA(MAXPX), MU(MAXPY), W(MMAX), WS(NWS),
     +                 X(MMAX), Y(MMAX)
      INTEGER          ADRES(NADRES), IWS(NIWS), POINT(NPTMAX)
      CHARACTER*1      LABEL(2)
*     .. External Subroutines ..
      EXTERNAL         E02DAF, E02DEF, E02ZAF
*     .. Data statements ..
      DATA             LABEL/'X', 'Y'/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02DAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=140) EPS
*     Read data, interchanging X and Y axes if PX.LT.PY
      READ (NIN,*) M
      IF (M.LE.MMAX .AND. M.GT.0) THEN
         READ (NIN,*) PX, PY
         IF (PX.GE.8 .AND. PX.LE.MAXPX .AND. PY.GE.8 .AND. PY.LE.MAXPY)
     +       THEN
```

```
      IF (PX.LT.PY) THEN
         ITEMP = PX
         PX = PY
         PY = ITEMP
         ITEMP = 1
         READ (NIN,*) (Y(I),X(I),F(I),W(I),I=1,M)
         IF (PY.GT.8) READ (NIN,*) (MU(J),J=5,PY-4)
         IF (PX.GT.8) READ (NIN,*) (LAMDA(J),J=5,PX-4)
      ELSE
         ITEMP = 0
         READ (NIN,*) (X(I),Y(I),F(I),W(I),I=1,M)
         IF (PX.GT.8) READ (NIN,*) (LAMDA(J),J=5,PX-4)
         IF (PY.GT.8) READ (NIN,*) (MU(J),J=5,PY-4)
      END IF
      NC = (PX-4)*(PY-4)
      NP = (PX-7)*(PY-7)
      WRITE (NOUT,*)
      WRITE (NOUT,99995) 'Interior ', LABEL(ITEMP+1), '-knots'
      DO 40 J = 5, PX - 4
         WRITE (NOUT,99996) LAMDA(J)
   40 CONTINUE
      IF (PX.EQ.8) WRITE (NOUT,*) 'None'
      WRITE (NOUT,*)
      WRITE (NOUT,99995) 'Interior ', LABEL(2-ITEMP), '-knots'
      DO 60 J = 5, PY - 4
         WRITE (NOUT,99996) MU(J)
   60 CONTINUE
      IF (PY.EQ.8) WRITE (NOUT,*) 'None'
*     Sort points into panel order
      IFAIL = 0
*
      CALL E02ZAF(PX,PY,LAMDA,MU,M,X,Y,POINT,NPTMAX,ADRES,NP,
     +            IFAIL)
*
*     Fit bicubic spline to data points
      IFAIL = 0
*
      CALL E02DAF(M,PX,PY,X,Y,F,W,LAMDA,MU,POINT,NPTMAX,DL,C,NC,
     +            WS,NWS,EPS,SIGMA,RANK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Sum of squares of residual RHS', SIGMA
      WRITE (NOUT,*)
      WRITE (NOUT,99998) 'Rank', RANK
*     Evaluate spline at the data points
      IFAIL = 0
*
      CALL E02DEF(M,PX,PY,X,Y,LAMDA,MU,C,FF,WS,IWS,IFAIL)
*
      SUM = 0
      IF (ITEMP.EQ.1) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'X and Y have been interchanged'
      END IF
*     Output data points, fitted values and residuals
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +   '      X           Y          Data       Fit      Residual'
      DO 100 I = 1, NP
         IADRES = I + M
   80    IADRES = POINT(IADRES)
         IF (IADRES.LE.0) GO TO 100
         TEMP = FF(IADRES) - F(IADRES)
         WRITE (NOUT,99997) X(IADRES), Y(IADRES), F(IADRES),
     +     FF(IADRES), TEMP
         SUM = SUM + (TEMP*W(IADRES))**2
         GO TO 80
  100 CONTINUE
```

```
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Sum of squared residuals', SUM
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Spline coefficients'
            DO 120 I = 1, PX - 4
               WRITE (NOUT,99996) (C((I-1)*(PY-4)+J),J=1,PY-4)
  120       CONTINUE
            GO TO 20
         END IF
      END IF
  140 STOP
*
99999 FORMAT (1X,A,1P,e16.2)
99998 FORMAT (1X,A,I5)
99997 FORMAT (1X,4F11.4,e11.2)
99996 FORMAT (1X,6F11.4)
99995 FORMAT (1X,A,A1,A)
      END
```

9.2. Program Data

```
E02DAF Example Program Data
0.000001
  30
   8
  10
   -0.52    0.60    0.93   10.
   -0.61   -0.95   -1.79   10.
    0.93    0.87    0.36   10.
    0.09    0.84    0.52   10.
    0.88    0.17    0.49   10.
   -0.70   -0.87   -1.76   10.
    1.00    1.00    0.33    1.
    1.00    0.10    0.48    1.
    0.30    0.24    0.65    1.
   -0.77   -0.77   -1.82    1.
   -0.23    0.32    0.92    1.
   -1.00    1.00    1.00    1.
   -0.26   -0.63    8.88    1.
   -0.83   -0.66   -2.01    1.
    0.22    0.93    0.47    1.
    0.89    0.15    0.49    1.
   -0.80    0.99    0.84    1.
   -0.88   -0.54   -2.42    1.
    0.68    0.44    0.47    1.
   -0.14   -0.72    7.15    1.
    0.67    0.63    0.44    1.
   -0.90   -0.40   -3.34    1.
   -0.84    0.20    2.78    1.
    0.84    0.43    0.44    1.
    0.15    0.28    0.70    1.
   -0.91   -0.24   -6.52    1.
   -0.35    0.86    0.66    1.
   -0.16   -0.41    2.32    1.
   -0.35   -0.05    1.66    1.
   -1.00   -1.00   -1.00    1.
  -0.5
   0.0
```

9.3. Program Results

```
E02DAF Example Program Results

Interior Y-knots
    -0.5000
     0.0000

Interior X-knots
None
```

```
Sum of squares of residual RHS        1.47E+01

Rank   22

X and Y have been interchanged

      X          Y          Data        Fit       Residual
   -0.9500    -0.6100    -1.7900    -1.7931   -0.31E-02
   -0.8700    -0.7000    -1.7600    -1.7521    0.79E-02
   -0.7700    -0.7700    -1.8200    -2.4301   -0.61E+00
   -0.6300    -0.2600     8.8800     7.6346   -0.12E+01
   -0.6600    -0.8300    -2.0100    -1.5815    0.43E+00
   -0.5400    -0.8800    -2.4200    -2.6795   -0.26E+00
   -0.7200    -0.1400     7.1500     7.5708    0.42E+00
   -1.0000    -1.0000    -1.0000    -1.0228   -0.23E-01
   -0.4000    -0.9000    -3.3400    -4.6955   -0.14E+01
   -0.2400    -0.9100    -6.5200    -4.7072    0.18E+01
   -0.4100    -0.1600     2.3200     2.7039    0.38E+00
   -0.0500    -0.3500     1.6600     2.2865    0.63E+00
    0.6000    -0.5200     0.9300     0.9441    0.14E-01
    0.8700     0.9300     0.3600     0.3529   -0.71E-02
    0.8400     0.0900     0.5200     0.5024   -0.18E-01
    0.1700     0.8800     0.4900     0.4705   -0.20E-01
    1.0000     1.0000     0.3300     0.6315    0.30E+00
    0.1000     1.0000     0.4800     1.4910    0.10E+01
    0.2400     0.3000     0.6500     0.9241    0.27E+00
    0.3200    -0.2300     0.9200    -0.3692   -0.13E+01
    1.0000    -1.0000     1.0000     1.0835    0.84E-01
    0.9300     0.2200     0.4700     1.4912    0.10E+01
    0.1500     0.8900     0.4900     0.4414   -0.49E-01
    0.9900    -0.8000     0.8400     0.5495   -0.29E+00
    0.4400     0.6800     0.4700     1.5862    0.11E+01
    0.6300     0.6700     0.4400     0.6288    0.19E+00
    0.2000    -0.8400     2.7800     1.7123   -0.11E+01
    0.4300     0.8400     0.4400     0.6888    0.25E+00
    0.2800     0.1500     0.7000     0.7713    0.71E-01
    0.8600    -0.3500     0.6600     0.9347    0.27E+00

Sum of squared residuals        1.47E+01

Spline coefficients
   -1.0228   115.4668  -433.5558   -68.1973
   24.8426  -140.1485   258.5042    15.6756
  -29.4878   132.2933  -173.5103    20.0983
    9.9575   -51.6200    67.6666    -5.8765
   10.0577     4.7543   -15.3533    -0.3260
    1.0835    -2.7932     7.7708     0.6315
```

E02DCF – NAG Fortran Library Routine Document

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

E02DCF computes a bicubic spline approximation to a set of data values, given on a rectangular grid in the x-y plane. The knots of the spline are located automatically, but a single parameter must be specified to control the trade-off between closeness of fit and smoothness of fit.

2 Specification

```
      SUBROUTINE E02DCF(START, MX, X, MY, Y, F, S, NXEST, NYEST, NX,
     1                  LAMDA, NY, MU, C, FP, WRK, LWRK, IWRK, LIWRK,
     2                  IFAIL)
      INTEGER           MX, MY, NXEST, NYEST, NX, NY, LWRK, IWRK(LIWRK),
     1                  LIWRK, IFAIL
      real              X(MX), Y(MY), F(MX*MY), S, LAMDA(NXEST),
     1                  MU(NYEST), C((NXEST-4)*(NYEST-4)), FP, WRK(LWRK)
      CHARACTER*1       START
```

3 Description

This routine determines a smooth bicubic spline approximation $s(x,y)$ to the set of data points $(x_q, y_r, f_{q,r})$, for $q = 1,2,\ldots,m_x$ and $r = 1,2,\ldots,m_y$.

The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{n_x-4}\sum_{j=1}^{n_y-4} c_{ij} M_i(x) N_j(y), \tag{1}$$

where $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}. For further details, see Hayes and Halliday [4] for bicubic splines and de Boor [1] for normalised B-splines.

The total numbers n_x and n_y of these knots and their values $\lambda_1,\ldots,\lambda_{n_x}$ and $\mu_1,\ldots,\mu_{n_y}$ are chosen automatically by the routine. The knots $\lambda_5,\ldots,\lambda_{n_x-4}$ and $\mu_5,\ldots,\mu_{n_y-4}$ are the interior knots; they divide the approximation domain $[x_1, x_{m_x}]\times[y_1, y_{m_y}]$ into $(n_x-7)\times(n_y-7)$ subpanels $[\lambda_i, \lambda_{i+1}]\times[\mu_j, \mu_{j+1}]$, for $i = 4,5,\ldots,n_x-4$, $j = 4,5,\ldots,n_y-4$. Then, much as in the curve case (see E02BEF), the coefficients c_{ij} are determined as the solution of the following constrained minimization problem:

minimize

$$\eta, \tag{2}$$

subject to the constraint

$$\theta = \sum_{q=1}^{m_x}\sum_{r=1}^{m_y} \epsilon_{q,r}^2 \leq S, \tag{3}$$

where η is a measure of the (lack of) smoothness of $s(x,y)$. Its value depends on the discontinuity jumps in $s(x,y)$ across the boundaries of the subpanels. It is zero only when there are no discontinuities and is positive otherwise, increasing with the size of the jumps (see Dierckx [2] for details).

$\epsilon_{q,r}$ denotes the residual $f_{q,r} - s(x_q, y_r)$,

and S is a non-negative number to be specified by the user.

By means of the parameter S, 'the smoothing factor', the user will then control the balance between smoothness and closeness of fit, as measured by the sum of squares of residuals in (3). If S is too large, the spline will be too smooth and signal will be lost (underfit); if S is too small, the spline will pick up too much noise (overfit). In the extreme cases the routine will return an interpolating spline ($\theta = 0$) if S is set to zero, and the least-squares bicubic polynomial ($\eta = 0$) if S is set very large. Experimenting with S-values between these two extremes should result in a good compromise. (See Section 8.3 for advice on choice of S.)

The method employed is outlined in Section 8.5 and fully described in Dierckx [2] and [3]. It involves an adaptive strategy for locating the knots of the bicubic spline (depending on the function underlying the data and on the value of S), and an iterative method for solving the constrained minimization problem once the knots have been determined.

Values of the computed spline can subsequently be computed by calling E02DEF or E02DFF as described in Section 8.6.

4 References

[1] De Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

[2] Dierckx P (1982) A fast algorithm for smoothing data on a rectangular grid while using spline functions *SIAM J. Numer. Anal.* **19** 1286–1304

[3] Dierckx P (1981) An improved algorithm for curve fitting with spline functions *Report TW54* Department of Computer Science, Katholieke Universiteit Leuven

[4] Hayes J G and Halliday J (1974) The least-squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

[5] Reinsch C H (1967) Smoothing by spline functions *Numer. Math.* **10** 177–183

5 Parameters

1: START — CHARACTER*1 *Input*

On entry: START must be set to 'C' or 'W'.

If START = 'C' (Cold start), the routine will build up the knot set starting with no interior knots. No values need be assigned to the parameters NX, NY, LAMDA, MU, WRK or IWRK.

If START = 'W' (Warm start), the routine will restart the knot-placing strategy using the knots found in a previous call of the routine. In this case, the parameters NX, NY, LAMDA, MU, WRK and IWRK must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of S.

Constraint: START = 'C' or 'W'.

2: MX — INTEGER *Input*

On entry: m_x, the number of grid points along the x axis.

Constraint: MX ≥ 4.

3: X(MX) — ***real*** array *Input*

On entry: X(q) must be set to x_q, the x co-ordinate of the qth grid point along the x axis, for $q = 1, 2, \ldots, m_x$.

Constraint: $x_1 < x_2 < \ldots < x_{m_x}$.

4: MY — INTEGER *Input*

On entry: m_y, the number of grid points along the y axis.

Constraint: MY ≥ 4.

5: Y(MY) — ***real*** array *Input*

On entry: Y(r) must be set to y_r, the y co-ordinate of the rth grid point along the y axis, for $r = 1, 2, \ldots, m_y$.

Constraint: $y_1 < y_2 < \ldots < y_{m_y}$.

6: F(MX*MY) — ***real*** array *Input*

On entry: F($m_y \times (q-1) + r$) must contain the data value $f_{q,r}$, for $q = 1, 2, \ldots, m_x$ and $r = 1, 2, \ldots, m_y$.

7: S — ***real*** *Input*

On entry: the smoothing factor, S.

If S = 0.0, the routine returns an interpolating spline.

If S is smaller than ***machine precision***, it is assumed equal to zero.

For advice on the choice of S, see Section 3 and Section 8.3.

Constraint: S $\geq$ 0.0.

8: NXEST — INTEGER *Input*

9: NYEST — INTEGER *Input*

On entry: an upper bound for the number of knots n_x and n_y required in the x- and y-directions respectively.

In most practical situations, NXEST = $m_x/2$ and NYEST = $m_y/2$ is sufficient. NXEST and NYEST never need to be larger than $m_x + 4$ and $m_y + 4$ respectively, the numbers of knots needed for interpolation (S = 0.0). See also Section 8.4.

Constraint: NXEST $\geq$ 8 and NYEST $\geq$ 8.

10: NX — INTEGER *Input/Output*

On entry: if the warm start option is used, the value of NX must be left unchanged from the previous call.

On exit: the total number of knots, n_x, of the computed spline with respect to the x variable.

11: LAMDA(NXEST) — ***real*** array *Input/Output*

On entry: if the warm start option is used, the values LAMDA(1), LAMDA(2),...,LAMDA(NX) must be left unchanged from the previous call.

On exit: LAMDA contains the complete set of knots λ_i associated with the x variable, i.e., the interior knots LAMDA(5), LAMDA(6),...,LAMDA(NX− 4) as well as the additional knots

$$\text{LAMDA(1)} = \text{LAMDA(2)} = \text{LAMDA(3)} = \text{LAMDA(4)} = \text{X(1)}$$

and

$$\text{LAMDA(NX} - 3) = \text{LAMDA(NX} - 2) = \text{LAMDA(NX} - 1) = \text{LAMDA(NX)} = \text{X(MX)}$$

needed for the B-spline representation.

12: NY — INTEGER *Input/Output*

On entry: if the warm start option is used, the value of NY must be left unchanged from the previous call.

On exit: the total number of knots, n_y, of the computed spline with respect to the y variable.

13: MU(NYEST) — ***real*** array *Input/Output*

On entry: if the warm start option is used, the values MU(1), MU(2),...,MU(NY) must be left unchanged from the previous call.

On exit: MU contains the complete set of knots μ_i associated with the y variable, i.e., the interior knots MU(5), MU(6),...,MU(NY − 4) as well as the additional knots

$$\text{MU}(1) = \text{MU}(2) = \text{MU}(3) = \text{MU}(4) = \text{Y}(1)$$

and

$$\text{MU}(\text{NY} - 3) = \text{MU}(\text{NY} - 2) = \text{MU}(\text{NY} - 1) = \text{MU}(\text{NY}) = \text{Y}(\text{MY})$$

needed for the B-spline representation.

14: C((NXEST−4)*(NYEST−4)) — ***real*** array *Output*

On exit: the coefficients of the spline approximation. C($(n_y - 4) \times (i - 1) + j$) is the coefficient c_{ij} defined in Section 3.

15: FP — ***real*** *Output*

On exit: the sum of squared residuals, θ, of the computed spline approximation. If FP = 0.0, this is an interpolating spline. FP should equal S within a relative tolerance of 0.001 unless NX = NY = 8, when the spline has no interior knots and so is simply a bicubic polynomial. For knots to be inserted, S must be set to a value below the value of FP produced in this case.

16: WRK(LWRK) — ***real*** array *Workspace*

On entry: if the warm start option is used, the values WRK(1),...,WRK(4) must be left unchanged from the previous call.

This array is used as workspace.

17: LWRK — INTEGER *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E02DCF is called.

Constraint: LWRK $\geq$ 4 × (MX+MY) +11 × (NXEST+NYEST) + NXEST × MY + max(MY,NXEST) + 54.

18: IWRK(LIWRK) — INTEGER array *Workspace*

On entry: if the warm start option is used, the values IWRK(1),...,IWRK(3) must be left unchanged from the previous call.

This array is used as workspace.

19: LIWRK — INTEGER *Input*

On entry: the dimension of the array IWRK as declared in the (sub)program from which E02DCF is called.

Constraint: LIWRK $\geq$ 3 + MX + MY + NXEST + NYEST.

20: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (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).

Errors detected by the routine:

IFAIL = 1

On entry, START ≠ 'C' or 'W',
or MX < 4,
or MY < 4,
or S < 0.0,
or S = 0.0 and NXEST < MX + 4,
or S = 0.0 and NYEST < MY + 4,
or NXEST < 8,
or NYEST < 8,
or LWRK < 4 × (MX+MY) + 11 × (NXEST+NYEST) + NXEST × MY + max(MY,NXEST) + 54,
or LIWRK < 3 + MX + MY + NXEST + NYEST.

IFAIL = 2

The values of X(q), for $q = 1, 2, \ldots$,MX, are not in strictly increasing order.

IFAIL = 3

The values of Y(r), for $r = 1, 2, \ldots$,MY, are not in strictly increasing order.

IFAIL = 4

The number of knots required is greater than allowed by NXEST and NYEST. Try increasing NXEST and/or NYEST and, if necessary, supplying larger arrays for the parameters LAMDA, MU, C, WRK and IWRK. However, if NXEST and NYEST are already large, say NXEST > MX/2 and NYEST > MY/2, then this error exit may indicate that S is too small.

IFAIL = 5

The iterative process used to compute the coefficients of the approximating spline has failed to converge. This error exit may occur if S has been set very small. If the error persists with increased S, consult NAG.

If IFAIL = 4 or 5, a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3) in Section 3) – perhaps by only a small amount, however.

7 Accuracy

On successful exit, the approximation returned is such that its sum of squared residuals FP is equal to the smoothing factor S, up to a specified relative tolerance of 0.001 – except that if $n_x = 8$ and $n_y = 8$, FP may be significantly less than S: in this case the computed spline is simply the least-squares bicubic polynomial approximation of degree 3, i.e., a spline with no interior knots.

8 Further Comments

8.1 Timing

The time taken for a call of E02DCF depends on the complexity of the shape of the data, the value of the smoothing factor S, and the number of data points. If E02DCF is to be called for different values of S, much time can be saved by setting START = 'W' after the first call.

8.2 Weighting of Data Points

E02DCF does not allow individual weighting of the data values. If these were determined to widely differing accuracies, it may be better to use E02DDF. The computation time would be very much longer, however.

8.3 Choice of S

If the standard deviation of $f_{q,r}$ is the same for all q and r (the case for which this routine is designed – see Section 8.2.) and known to be equal, at least approximately, to σ, say, then following Reinsch [5] and choosing the smoothing factor S in the range $\sigma^2(m \pm \sqrt{2m})$, where $m = m_x m_y$, is likely to give a good start in the search for a satisfactory value. If the standard deviations vary, the sum of their squares over all the data points could be used. Otherwise experimenting with different values of S will be required from the start, taking account of the remarks in Section 3.

In that case, in view of computation time and memory requirements, it is recommended to start with a very large value for S and so determine the least-squares bicubic polynomial; the value returned for FP, call it FP_0, gives an upper bound for S. Then progressively decrease the value of S to obtain closer fits – say by a factor of 10 in the beginning, i.e., $S = \mathrm{FP}_0/10$, $S = \mathrm{FP}_0/100$, and so on, and more carefully as the approximation shows more details.

The number of knots of the spline returned, and their location, generally depend on the value of S and on the behaviour of the function underlying the data. However, if E02DCF is called with START = 'W', the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of S and START = 'W', a fit can finally be accepted as satisfactory, it may be worthwhile to call E02DCF once more with the selected value for S but now using START = 'C'. Often, E02DCF then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

8.4 Choice of NXEST and NYEST

The number of knots may also depend on the upper bounds NXEST and NYEST. Indeed, if at a certain stage in E02DCF the number of knots in one direction (say n_x) has reached the value of its upper bound (NXEST), then from that moment on all subsequent knots are added in the other (y) direction. Therefore the user has the option of limiting the number of knots the routine locates in any direction. For example, by setting NXEST = 8 (the lowest allowable value for NXEST), the user can indicate that he wants an approximation which is a simple cubic polynomial in the variable x.

8.5 Outline of Method Used

If $S = 0$, the requisite number of knots is known in advance, i.e., $n_x = m_x + 4$ and $n_y = m_y + 4$; the interior knots are located immediately as $\lambda_i = x_{i-2}$ and $\mu_j = y_{j-2}$, for $i = 5, 6, \ldots, n_x - 4$ and $j = 5, 6, \ldots, n_y - 4$. The corresponding least-squares spline is then an interpolating spline and therefore a solution of the problem.

If $S > 0$, suitable knot sets are built up in stages (starting with no interior knots in the case of a cold start but with the knot set found in a previous call if a warm start is chosen). At each stage, a bicubic spline is fitted to the data by least-squares, and θ, the sum of squares of residuals, is computed. If $\theta > S$, new knots are added to one knot set or the other so as to reduce θ at the next stage. The new knots are located in intervals where the fit is particularly poor, their number depending on the value of S and on the progress made so far in reducing θ. Sooner or later, we find that $\theta \leq S$ and at that point the knot sets are accepted. The routine then goes on to compute the (unique) spline which has these knot sets and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = S$. The routine computes the spline by an iterative scheme which is ended when $\theta = S$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in E02BAF for least-squares curve fitting.

An exception occurs when the routine finds at the start that, even with no interior knots ($n_x = n_y = 8$), the least-squares spline already has its sum of residuals $\leq S$. In this case, since this spline (which is simply a bicubic polynomial) also has an optimal value for the smoothness measure η, namely zero, it is returned at once as the (trivial) solution. It will usually mean that S has been chosen too large.

For further details of the algorithm and its use see Dierckx [2].

8.6 Evaluation of Computed Spline

The values of the computed spline at the points (TX(r),TY(r)), for $r = 1, 2, \ldots$,N, may be obtained in the ***real*** array FF, of length at least N, by the following code:

```
IFAIL = 0
CALL E02DEF(N,NX,NY,TX,TY,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
```

where NX, NY, LAMDA, MU and C are the output parameters of E02DCF, WRK is a ***real*** workspace array of length at least NY -4, and IWRK is an integer workspace array of length at least NY -4.

To evaluate the computed spline on a KX by KY rectangular grid of points in the x-y plane, which is defined by the x co-ordinates stored in TX(q), for $q = 1, 2, \ldots$,KX, and the y co-ordinates stored in TY(r), for $r = 1, 2, \ldots$,KY, returning the results in the ***real*** array FG which is of length at least KX $\times$ KY, the following call may be used:

```
IFAIL = 0
CALL E02DFF(KX,KY,NX,NY,TX,TY,LAMDA,MU,C,FG,WRK,LWRK,
*            IWRK,LIWRK,IFAIL)
```

where NX, NY, LAMDA, MU and C are the output parameters of E02DCF, WRK is a ***real*** workspace array of length at least LWRK = min(NWRK1,NWRK2), NWRK1 = KX $\times$ 4 + NX, NWRK2 = KY $\times$ 4 + NY, and IWRK is an integer workspace array of length at least LIWRK = KY + NY $-$ 4 if NWRK1 $\geq$ NWRK2, or KX + NX $-$ 4 otherwise. The result of the spline evaluated at grid point (q, r) is returned in element (KY$\times(q-1)+r$) of the array FG.

9 Example

This example program reads in values of MX, MY, x_q, for $q = 1, 2, \ldots$,MX, and y_r, for $r = 1, 2, \ldots$,MY, followed by values of the ordinates $f_{q,r}$ defined at the grid points (x_q,y_r). It then calls E02DCF to compute a bicubic spline approximation for one specified value of S, and prints the values of the computed knots and B-spline coefficients. Finally it evaluates the spline at a small sample of points on a rectangular grid.

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.

```
*     E02DCF Example Program Text
*     Mark 18 Revised.  NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MXMAX, MYMAX, NXEST, NYEST, NMAX, LWRK, LIWRK
      PARAMETER        (MXMAX=11,MYMAX=9,NXEST=MXMAX+4,NYEST=MYMAX+4,
     +                 NMAX=7,LWRK=4*(MXMAX+MYMAX)+11*(NXEST+NYEST)
     +                 +NXEST*MYMAX+NXEST+54,LIWRK=3+MXMAX+MYMAX+NXEST+
     +                 NYEST)
*     .. Local Scalars ..
      real             DELTA, FP, S, XHI, XLO, YHI, YLO
      INTEGER          I, IFAIL, J, MX, MY, NPX, NPY, NX, NY
      CHARACTER        START
*     .. Local Arrays ..
      real             C((NXEST-4)*(NYEST-4)), F(MXMAX*MYMAX),
     +                 FG(NMAX*NMAX), LAMDA(NXEST), MU(NYEST), PX(NMAX),
     +                 PY(NMAX), WRK(LWRK), X(MXMAX), Y(MYMAX)
      INTEGER          IWRK(LIWRK)
```

```
*     .. External Subroutines ..
      EXTERNAL         CPRINT, E02DCF, E02DFF
*     .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02DCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of X, Y co-ordinates MX, MY.
      READ (NIN,*) MX, MY
      IF (MX.GT.0 .AND. MX.LE.MXMAX .AND. MY.GT.0 .AND. MY.LE.MYMAX)
     +    THEN
*        Input the X co-ordinates followed by the Y co-ordinates.
         READ (NIN,*) (X(I),I=1,MX)
         READ (NIN,*) (Y(I),I=1,MY)
*        Input the MX*MY function values F at the grid points.
         READ (NIN,*) (F(I),I=1,MX*MY)
         START = 'Cold Start'
         READ (NIN,*,END=100) S
*        Determine the spline approximation.
         IFAIL = 0
*
         CALL E02DCF(START,MX,X,MY,Y,F,S,NXEST,NYEST,NX,LAMDA,NY,MU,C,
     +              FP,WRK,LWRK,IWRK,LIWRK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calling with smoothing factor S =', S,
     +     ': NX =', NX, ', NY =', NY, '.'
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     '                I   Knot LAMDA(I)      J     Knot MU(J)'
         WRITE (NOUT,*)
         DO 20 J = 4, MAX(NX,NY) - 3
            IF (J.LE.NX-3 .AND. J.LE.NY-3) THEN
               WRITE (NOUT,99997) J, LAMDA(J), J, MU(J)
            ELSE IF (J.LE.NX-3) THEN
               WRITE (NOUT,99997) J, LAMDA(J)
            ELSE IF (J.LE.NY-3) THEN
               WRITE (NOUT,99996) J, MU(J)
            END IF
   20    CONTINUE
         CALL CPRINT(C,NY,NX,NOUT)
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Sum of squared residuals FP =', FP
         IF (FP.EQ.0.0e+0) THEN
            WRITE (NOUT,*) '(The spline is an interpolating spline)'
         ELSE IF (NX.EQ.8 .AND. NY.EQ.8) THEN
            WRITE (NOUT,*)
     +        '(The spline is the least-squares bi-cubic polynomial)'
         END IF
*        Evaluate the spline on a rectangular grid at NPX*NPY points
*        over the domain (XLO to XHI) x (YLO to YHI).
         READ (NIN,*) NPX, XLO, XHI
         READ (NIN,*) NPY, YLO, YHI
         IF (NPX.LE.NMAX .AND. NPY.LE.NMAX) THEN
            DELTA = (XHI-XLO)/(NPX-1)
            DO 40 I = 1, NPX
```

```
               PX(I) = MIN(XLO+(I-1)*DELTA,XHI)
   40       CONTINUE
            DO 60 I = 1, NPY
               PY(I) = MIN(YLO+(I-1)*DELTA,YHI)
   60       CONTINUE
*
            CALL E02DFF(NPX,NPY,NX,NY,PX,PY,LAMDA,MU,C,FG,WRK,LWRK,IWRK,
     +                  LIWRK,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Values of computed spline:'
            WRITE (NOUT,*)
            WRITE (NOUT,99995) '          X', (PX(I),I=1,NPX)
            WRITE (NOUT,*) '     Y'
            DO 80 I = NPY, 1, -1
               WRITE (NOUT,99994) PY(I), (FG(NPY*(J-1)+I),J=1,NPX)
   80       CONTINUE
         END IF
      END IF
  100 CONTINUE
      STOP
*
99999 FORMAT (1X,A,1P,e13.4,A,I5,A,I5,A)
99998 FORMAT (1X,A,1P,e13.4)
99997 FORMAT (1X,I16,F12.4,I11,F12.4)
99996 FORMAT (1X,I39,F12.4)
99995 FORMAT (1X,A,7F8.2)
99994 FORMAT (1X,F8.2,3X,7F8.2)
      END
*
      SUBROUTINE CPRINT(C,NY,NX,NOUT)
*     .. Scalar Arguments ..
      INTEGER           NOUT, NX, NY
*     .. Array Arguments ..
      real              C(NY-4,NX-4)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The B-spline coefficients:'
      WRITE (NOUT,*)
      DO 20 I = 1, NY - 4
         WRITE (NOUT,99999) (C(I,J),J=1,NX-4)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,8F9.4)
      END
```

9.2 Program Data

```
E02DCF Example Program Data
 11    9      MX, MY, number of grid points on the X and Y axes
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00  4.5000E+00
 5.0000E+00   End of MX grid points
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00   End of MY grid points
```

```
 1.0000E+00  8.8758E-01  5.4030E-01  7.0737E-02 -4.1515E-01
-8.0114E-01 -9.7999E-01 -9.3446E-01 -6.5664E-01  1.5000E+00
 1.3564E+00  8.2045E-01  1.0611E-01 -6.2422E-01 -1.2317E+00
-1.4850E+00 -1.3047E+00 -9.8547E-01  2.0600E+00  1.7552E+00
 1.0806E+00  1.5147E-01 -8.3229E-01 -1.6023E+00 -1.9700E+00
-1.8729E+00 -1.4073E+00  2.5700E+00  2.1240E+00  1.3508E+00
 1.7684E-01 -1.0404E+00 -2.0029E+00 -2.4750E+00 -2.3511E+00
-1.6741E+00  3.0000E+00  2.6427E+00  1.6309E+00  2.1221E-01
-1.2484E+00 -2.2034E+00 -2.9700E+00 -2.8094E+00 -1.9809E+00
 3.5000E+00  3.1715E+00  1.8611E+00  2.4458E-01 -1.4565E+00
-2.8640E+00 -3.2650E+00 -3.2776E+00 -2.2878E+00  4.0400E+00
 3.5103E+00  2.0612E+00  2.8595E-01 -1.6946E+00 -3.2046E+00
-3.9600E+00 -3.7958E+00 -2.6146E+00  4.5000E+00  3.9391E+00
 2.4314E+00  3.1632E-01 -1.8627E+00 -3.6351E+00 -4.4550E+00
-4.2141E+00 -2.9314E+00  5.0400E+00  4.3879E+00  2.7515E+00
 3.5369E-01 -2.0707E+00 -4.0057E+00 -4.9700E+00 -4.6823E+00
-3.2382E+00  5.5050E+00  4.8367E+00  2.9717E+00  3.8505E-01
-2.2888E+00 -4.4033E+00 -5.4450E+00 -5.1405E+00 -3.5950E+00
 6.0000E+00  5.2755E+00  3.2418E+00  4.2442E-01 -2.4769E+00
-4.8169E+00 -5.9300E+00 -5.6387E+00 -3.9319E+00   End of data values
 0.1         S, smoothing factor
 6    0.0   5.0
 5    0.0   4.0
```

9.3 Program Results

```
E02DCF Example Program Results

Calling with smoothing factor S =   1.0000E-01: NX =   10, NY =   13.

               I   Knot LAMDA(I)      J     Knot MU(J)

               4      0.0000          4      0.0000
               5      1.5000          5      1.0000
               6      2.5000          6      2.0000
               7      5.0000          7      2.5000
                                      8      3.0000
                                      9      3.5000
                                     10      4.0000

The B-spline coefficients:

   0.9918   1.5381   2.3913   3.9845   5.2138   5.9965
   1.0546   1.5270   2.2441   4.2217   5.0860   6.0821
   0.6098   0.9557   1.5587   2.3458   3.3860   3.7716
  -0.2915  -0.4199  -0.7399  -1.1763  -1.5527  -1.7775
  -0.8476  -1.3296  -1.8521  -3.3468  -4.3628  -5.0085
  -1.0168  -1.5952  -2.4022  -3.9390  -5.4680  -6.1656
  -0.9529  -1.3381  -2.2844  -3.9559  -5.0032  -5.8709
  -0.7711  -1.0914  -1.8488  -3.2549  -3.9444  -4.7297
  -0.6476  -1.0373  -1.5936  -2.5887  -3.3485  -3.9330

Sum of squared residuals FP =   1.0004E-01
```

Values of computed spline:

X	0.00	1.00	2.00	3.00	4.00	5.00
Y						
4.00	-0.65	-1.36	-1.99	-2.61	-3.25	-3.93
3.00	-0.98	-1.97	-2.91	-3.91	-4.97	-5.92
2.00	-0.42	-0.83	-1.24	-1.66	-2.08	-2.48
1.00	0.54	1.09	1.61	2.14	2.71	3.24
0.00	0.99	2.04	3.03	4.01	5.02	6.00

E02DDF – NAG Fortran Library Routine Document

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

E02DDF computes a bicubic spline approximation to a set of scattered data. The knots of the spline are located automatically, but a single parameter must be specified to control the trade-off between closeness of fit and smoothness of fit.

2 Specification

```
 SUBROUTINE E02DDF(START, M, X, Y, F, W, S, NXEST, NYEST, NX,
1                  LAMDA, NY, MU, C, FP, RANK, WRK, LWRK, IWRK,
2                  LIWRK, IFAIL)
 INTEGER           M, NXEST, NYEST, NX, NY, RANK, LWRK,
1                  IWRK(LIWRK), LIWRK, IFAIL
 real              X(M), Y(M), F(M), W(M), S, LAMDA(NXEST),
1                  MU(NYEST), C((NXEST-4)*(NYEST-4)), FP, WRK(LWRK)
 CHARACTER*1       START
```

3 Description

This routine determines a smooth bicubic spline approximation $s(x,y)$ to the set of data points (x_r, y_r, f_r) with weights w_r, for $r = 1, 2, \ldots, m$.

The approximation domain is considered to be the rectangle $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, where $x_{\min}(y_{\min})$ and $x_{\max}(y_{\max})$ denote the lowest and highest data values of x (y).

The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{n_x-4} \sum_{j=1}^{n_y-4} c_{ij} M_i(x) N_j(y), \tag{1}$$

where $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}. For further details, see Hayes and Halliday [4] for bicubic splines and de Boor [1] for normalised B-splines.

The total numbers n_x and n_y of these knots and their values $\lambda_1, \ldots, \lambda_{n_x}$ and $\mu_1, \ldots, \mu_{n_y}$ are chosen automatically by the routine. The knots $\lambda_5, \ldots, \lambda_{n_x-4}$ and $\mu_5, \ldots, \mu_{n_y-4}$ are the interior knots; they divide the approximation domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ into $(n_x - 7) \times (n_y - 7)$ subpanels $[\lambda_i, \lambda_{i+1}] \times [\mu_j, \mu_{j+1}]$, for $i = 4, 5, \ldots, n_x - 4$; $j = 4, 5, \ldots, n_y - 4$. Then, much as in the curve case (see E02BEF), the coefficients c_{ij} are determined as the solution of the following constrained minimization problem:

minimize

$$\eta, \tag{2}$$

subject to the constraint

$$\theta = \sum_{r=1}^{m} \epsilon_r^2 \leq S \tag{3}$$

where: η is a measure of the (lack of) smoothness of $s(x,y)$. Its value depends on the discontinuity jumps in $s(x,y)$ across the boundaries of the subpanels. It is zero only when there are no discontinuities and is positive otherwise, increasing with the size of the jumps (see Dierckx [2] for details).

ϵ_r denotes the weighted residual $w_r(f_r - s(x_r, y_r))$,

and S is a non-negative number to be specified by the user.

By means of the parameter S, 'the smoothing factor', the user will then control the balance between smoothness and closeness of fit, as measured by the sum of squares of residuals in (3). If S is too large, the spline will be too smooth and signal will be lost (underfit); if S is too small, the spline will pick up too much noise (overfit). In the extreme cases the method would return an interpolating spline ($\theta = 0$) if S were set to zero, and returns the least-squares bicubic polynomial ($\eta = 0$) if S is set very large. Experimenting with S-values between these two extremes should result in a good compromise. (See Section 8.2 for advice on choice of S.) Note however, that this routine, unlike E02BEF and E02DCF, does not allow S to be set exactly to zero: to compute an interpolant to scattered data, E01SAF or E01SEF should be used.

The method employed is outlined in Section 8.5 and fully described in Dierckx [2] and [3]. It involves an adaptive strategy for locating the knots of the bicubic spline (depending on the function underlying the data and on the value of S), and an iterative method for solving the constrained minimization problem once the knots have been determined.

Values of the computed spline can subsequently be computed by calling E02DEF or E02DFF as described in Section 8.6.

4 References

[1] De Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

[2] Dierckx P (1981) An algorithm for surface fitting with spline functions *IMA J. Numer. Anal.* **1** 267–283

[3] Dierckx P (1981) An improved algorithm for curve fitting with spline functions *Report TW54* Department of Computer Science, Katholieke Universiteit Leuven

[4] Hayes J G and Halliday J (1974) The least-squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

[5] Peters G and Wilkinson J H (1970) The least-squares problem and pseudo-inverses *Comput. J.* **13** 309–316

[6] Reinsch C H (1967) Smoothing by spline functions *Numer. Math.* **10** 177–183

5 Parameters

1: START — CHARACTER*1 *Input*

On entry: START must be set to 'C' or 'W'.

If START = 'C' (Cold start), the routine will build up the knot set starting with no interior knots. No values need be assigned to the parameters NX, NY, LAMDA, MU or WRK.

If START = 'W' (Warm start), the routine will restart the knot-placing strategy using the knots found in a previous call of the routine. In this case, the parameters NX, NY, LAMDA, MU and WRK must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of S.

Constraint: START = 'C' or 'W'.

2: M — INTEGER *Input*

On entry: m, the number of data points.

The number of data points with non-zero weight (see W below) must be at least 16.

3: X(M) — ***real*** array *Input*

4: Y(M) — ***real*** array *Input*

5: F(M) — ***real*** array *Input*

On entry: X(r), Y(r), F(r) must be set to the co-ordinates of (x_r, y_r, f_r), the rth data point, for $r = 1, 2, \ldots, m$. The order of the data points is immaterial.

6: W(M) — ***real*** array *Input*

On entry: W(r) must be set to w_r, the rth value in the set of weights, for $r = 1, 2, \ldots, m$. Zero weights are permitted and the corresponding points are ignored, except when determining $x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$ (see Section 8.4). For advice on the choice of weights, see Section 2.1.2 of the Chapter Introduction.

Constraint: the number of data points with non-zero weight must be at least 16.

7: S — ***real*** *Input*

On entry: the smoothing factor, S.

For advice on the choice of S, see Section 3 and Section 8.2.

Constraint: S > 0.0.

8: NXEST — INTEGER *Input*

9: NYEST — INTEGER *Input*

On entry: an upper bound for the number of knots n_x and n_y required in the x- and y-directions respectively.

In most practical situations, NXEST = NYEST = $4 + \sqrt{m/2}$ is sufficient. See also Section 8.3.

Constraint: NXEST ≥ 8 and NYEST ≥ 8.

10: NX — INTEGER *Input/Output*

On entry: if the warm start option is used, the value of NX must be left unchanged from the previous call.

On exit: the total number of knots, n_x, of the computed spline with respect to the x variable.

11: LAMDA(NXEST) — ***real*** array *Input/Output*

On entry: if the warm start option is used, the values LAMDA(1), LAMDA(2),...,LAMDA(NX) must be left unchanged from the previous call.

On exit: LAMDA contains the complete set of knots λ_i associated with the x variable, i.e., the interior knots LAMDA(5), LAMDA(6),...,LAMDA(NX− 4) as well as the additional knots

$$\text{LAMDA}(1) = \text{LAMDA}(2) = \text{LAMDA}(3) = \text{LAMDA}(4) = x_{\min}$$

and

$$\text{LAMDA}(\text{NX} - 3) = \text{LAMDA}(\text{NX} - 2) = \text{LAMDA}(\text{NX} - 1) = \text{LAMDA}(\text{NX}) = x_{\max}$$

needed for the B-spline representation (where $x_{\min}$ and $x_{\max}$ are as described in Section 3).

12: NY — INTEGER *Input/Output*

On entry: if the warm start option is used, the value of NY must be left unchanged from the previous call.

On exit: the total number of knots, n_y, of the computed spline with respect to the y variable.

13: MU(NYEST) — ***real*** array *Input/Output*

On entry: if the warm start option is used, the values MU(1), MU(2),...,MU(NY) must be left unchanged from the previous call.

On exit: MU contains the complete set of knots μ_i associated with the y variable, i.e., the interior knots MU(5), MU(6),...,MU(NY − 4) as well as the additional knots

$$\text{MU}(1) = \text{MU}(2) = \text{MU}(3) = \text{MU}(4) = y_{\min}$$

and

$$\text{MU}(\text{NY} - 3) = \text{MU}(\text{NY} - 2) = \text{MU}(\text{NY} - 1) = \text{MU}(\text{NY}) = y_{\max}$$

needed for the B-spline representation (where $y_{\min}$ and $y_{\max}$ are as described in Section 3).

14: C((NXEST−4)*(NYEST−4)) — ***real*** array *Output*

On exit: the coefficients of the spline approximation. C($(n_y - 4) \times (i - 1) + j$) is the coefficient c_{ij} defined in Section 3.

15: FP — ***real*** *Output*

On exit: the weighted sum of squared residuals, θ, of the computed spline approximation. FP should equal S within a relative tolerance of 0.001 unless NX = NY = 8, when the spline has no interior knots and so is simply a bicubic polynomial. For knots to be inserted, S must be set to a value below the value of FP produced in this case.

16: RANK — INTEGER *Output*

On exit: RANK gives the rank of the system of equations used to compute the final spline (as determined by a suitable machine-dependent threshold). When RANK = (NX−4) × (NY−4), the solution is unique; otherwise the system is rank-deficient and the minimum-norm solution is computed. The latter case may be caused by too small a value of S.

17: WRK(LWRK) — ***real*** array *Workspace*

On entry: if the warm start option is used, the value of WRK(1) must be left unchanged from the previous call.

This array is used as workspace.

18: LWRK — INTEGER *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E02DDF is called.

Constraint: LWRK $\geq (7 \times u \times v + 25 \times w) \times (w + 1) + 2 \times (u + v + 4 \times \mathrm{M}) + 23 \times w + 56$,

where u = NXEST − 4, v = NYEST − 4, and $w = \max(u, v)$.

For some problems, the routine may need to compute the minimal least-squares solution of a rank-deficient system of linear equations (see Section 3). The amount of workspace required to solve such problems will be larger than specified by the value given above, which must be increased by an amount, LWRK2 say. An upper bound for LWRK2 is given by $4 \times u \times v \times w + 2 \times u \times v + 4 \times w$, where u, v and w are as above. However, if there are enough data points, scattered uniformly over the approximation domain, and if the smoothing factor S is not too small, there is a good chance that this extra workspace is not needed. A lot of memory might therefore be saved by assuming LWRK2 = 0.

19: IWRK(LIWRK) — INTEGER array *Workspace*

20: LIWRK — INTEGER *Input*

On entry: the dimension of the array IWRK as declared in the (sub)program from which E02DDF is called.

Constraint: LIWRK $\geq$ M + 2 × (NXEST−7) × (NYEST−7).

21: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (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).

Errors detected by the routine:

IFAIL = 1

On entry, START ≠ 'C' or 'W',

or the number of data points with non-zero weight < 16,

or S ≤ 0.0,

or NXEST < 8,

or NYEST < 8,

or LWRK < $(7 \times u \times v + 25 \times w) \times (w + 1) + 2 \times (u + v + 4 \times \mathrm{M}) + 23 \times w + 56$, where u = NXEST−4, v = NYEST−4 and $w = \max(u, v)$,

or LIWRK < M + 2 × (NXEST−7) × (NYEST−7).

IFAIL = 2

On entry, either all the X(r), for $r = 1, 2, \ldots$,M, are equal, or all the Y(r), for $r = 1, 2, \ldots$,M, are equal.

IFAIL = 3

The number of knots required is greater than allowed by NXEST and NYEST. Try increasing NXEST and/or NYEST and, if necessary, supplying larger arrays for the parameters LAMDA, MU, C, WRK and IWRK. However, if NXEST and NYEST are already large, say NXEST, NYEST > $4 + \sqrt{\mathrm{M}/2}$, then this error exit may indicate that S is too small.

IFAIL = 4

No more knots can be added because the number of B-spline coefficients (NX − 4) × (NY − 4) already exceeds the number of data points M. This error exit may occur if either of S or M is too small.

IFAIL = 5

No more knots can be added because the additional knot would (quasi) coincide with an old one. This error exit may occur if too large a weight has been given to an inaccurate data point, or if S is too small.

IFAIL = 6

The iterative process used to compute the coefficients of the approximating spline has failed to converge. This error exit may occur if S has been set very small. If the error persists with increased S, consult NAG.

IFAIL = 7

LWRK is too small; the routine needs to compute the minimal least-squares solution of a rank-deficient system of linear equations, but there is not enough workspace. There is no approximation returned but, having saved the information contained in NX, LAMDA, NY, MU and WRK, and having adjusted the value of LWRK and the dimension of array WRK accordingly, the user can continue at the point the program was left by calling E02DDF with START = 'W'. Note that the requested value for LWRK is only large enough for the current phase of the algorithm. If the routine is restarted with LWRK set to the minimum value requested, a larger request may be made at a later stage of the computation. See Section 5 for the upper bound on LWRK. On soft failure, the minimum requested value for LWRK is returned in IWRK(1) and the safe value for LWRK is returned in IWRK(2).

If IFAIL = 3,4,5 or 6, a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3) in Section 3 – perhaps only by a small amount, however.

7 Accuracy

On successful exit, the approximation returned is such that its weighted sum of squared residuals FP is equal to the smoothing factor S, up to a specified relative tolerance of 0.001 - except that if $n_x = 8$ and $n_y = 8$, FP may be significantly less than S: in this case the computed spline is simply the least-squares bicubic polynomial approximation of degree 3, i.e., a spline with no interior knots.

8 Further Comments

8.1 Timing

The time taken for a call of E02DDF depends on the complexity of the shape of the data, the value of the smoothing factor S, and the number of data points. If E02DDF is to be called for different values of S, much time can be saved by setting START = 'W' after the first call.

It should be noted that choosing S very small considerably increases computation time.

8.2 Choice of S

If the weights have been correctly chosen (see Section 2.1.2 of the Chapter Introduction), the standard deviation of $w_r f_r$ would be the same for all r, equal to σ, say. In this case, choosing the smoothing factor S in the range $\sigma^2(m \pm \sqrt{2m})$, as suggested by Reinsch [6], is likely to give a good start in the search for a satisfactory value. Otherwise, experimenting with different values of S will be required from the start.

In that case, in view of computation time and memory requirements, it is recommended to start with a very large value for S and so determine the least-squares bicubic polynomial; the value returned for FP, call it FP_0, gives an upper bound for S. Then progressively decrease the value of S to obtain closer fits - say by a factor of 10 in the beginning, i.e., S = $\mathrm{FP}_0/10$, S = $\mathrm{FP}_0/100$, and so on, and more carefully as the approximation shows more details.

To choose S very small is strongly discouraged. This considerably increases computation time and memory requirements. It may also cause rank-deficiency (as indicated by the parameter RANK) and endanger numerical stability.

The number of knots of the spline returned, and their location, generally depend on the value of S and on the behaviour of the function underlying the data. However, if E02DDF is called with START = 'W', the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of S and START = 'W', a fit can finally be accepted as satisfactory, it may be worthwhile to call E02DDF once more with the selected value for S but now using START = 'C'. Often, E02DDF then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

8.3 Choice of NXEST and NYEST

The number of knots may also depend on the upper bounds NXEST and NYEST. Indeed, if at a certain stage in E02DDF the number of knots in one direction (say n_x) has reached the value of its upper bound (NXEST), then from that moment on all subsequent knots are added in the other (y) direction. This may indicate that the value of NXEST is too small. On the other hand, it gives the user the option of limiting the number of knots the routine locates in any direction. For example, by setting NXEST = 8 (the lowest allowable value for NXEST), the user can indicate that he wants an approximation which is a simple cubic polynomial in the variable x.

8.4 Restriction of the approximation domain

The fit obtained is not defined outside the rectangle $[\lambda_4, \lambda_{n_x-3}] \times [\mu_4, \mu_{n_y-3}]$. The reason for taking the extreme data values of x and y for these four knots is that, as is usual in data fitting, the fit cannot be expected to give satisfactory values outside the data region. If, nevertheless, the user requires values over a larger rectangle, this can be achieved by augmenting the data with two artificial data points $(a, c, 0)$ and $(b, d, 0)$ with zero weight, where $[a, b] \times [c, d]$ denotes the enlarged rectangle.

8.5 Outline of method used

First suitable knot sets are built up in stages (starting with no interior knots in the case of a cold start but with the knot set found in a previous call if a warm start is chosen). At each stage, a bicubic spline is fitted to the data by least-squares and θ, the sum of squares of residuals, is computed. If θ >S, a new knot is added to one knot set or the other so as to reduce θ at the next stage. The new knot is located in an interval where the fit is particularly poor. Sooner or later, we find that $\theta \leq$ S and at that point the knot sets are accepted. The routine then goes on to compute a spline which has these knot sets and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = S$. The routine computes the spline by an iterative scheme which is ended when $\theta = S$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in E02DAF. As there also, the minimal least-squares solution is computed wherever the linear system is found to be rank-deficient.

An exception occurs when the routine finds at the start that, even with no interior knots (N = 8), the least-squares spline already has its sum of squares of residuals $\leq$ S. In this case, since this spline (which is simply a bicubic polynomial) also has an optimal value for the smoothness measure η, namely zero, it is returned at once as the (trivial) solution. It will usually mean that S has been chosen too large.

For further details of the algorithm and its use see Dierckx [2].

8.6 Evaluation of computed spline

The values of the computed spline at the points (TX(r),TY(r)), for $r = 1, 2, \ldots$,N, may be obtained in the ***real*** array FF, of length at least N, by the following code:

```
IFAIL = 0
CALL E02DEF(N,NX,NY,TX,TY,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
```

where NX, NY, LAMDA, MU and C are the output parameters of E02DDF, WRK is a ***real*** workspace array of length at least NY −4, and IWRK is an integer workspace array of length at least NY −4.

To evaluate the computed spline on a KX by KY rectangular grid of points in the x-y plane, which is defined by the x co-ordinates stored in TX(q), for $q = 1, 2, \ldots$,KX, and the y co-ordinates stored in TY(r), for $r = 1, 2, \ldots$,KY, returning the results in the ***real*** array FG which is of length at least KX × KY, the following call may be used:

```
IFAIL = 0
CALL E02DFF(KX,KY,NX,NY,TX,TY,LAMDA,MU,C,FG,WRK,LWRK,
*            IWRK,LIWRK,IFAIL)
```

where NX, NY, LAMDA, MU and C are the output parameters of E02DDF, WRK is a ***real*** workspace array of length at least LWRK = min(NWRK1,NWRK2), NWRK1 = KX × 4 + NX, NWRK2 = KY × 4 + NY, and IWRK is an integer workspace array of length at least LIWRK = KY + NY − 4 if NWRK1 $\geq$ NWRK2, or KX + NX − 4 otherwise. The result of the spline evaluated at grid point (q, r) is returned in element (KY×(q − 1)+r) of the array FG.

9 Example

This example program reads in a value of M, followed by a set of M data points (x_r, y_r, f_r) and their weights w_r. It then calls E02DDF to compute a bicubic spline approximation for one specified value of S, and prints the values of the computed knots and B-spline coefficients. Finally it evaluates the spline at a small sample of points on a rectangular grid.

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.

```
*     E02DDF Example Program Text
*     Mark 18 Revised.  NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NXEST, NYEST, NMAX, LWRK, LIWRK
      PARAMETER        (MMAX=30,NXEST=14,NYEST=14,NMAX=7,
     +                 LWRK=(7*(NXEST-4)*(NYEST-4)+25*(NXEST-4))
     +                 *(NXEST-3)+2*((NXEST-4)+(NYEST-4)+4*MMAX)
     +                 +23*(NXEST-4)+56,LIWRK=MMAX+2*(NXEST-7)*(NYEST-7)
     +                 )
*     .. Local Scalars ..
      real             DELTA, FP, S, XHI, XLO, YHI, YLO
      INTEGER          I, IFAIL, J, M, NPX, NPY, NX, NY, RANK
      CHARACTER        START
*     .. Local Arrays ..
      real             C((NXEST-4)*(NYEST-4)), F(MMAX), FG(NMAX*NMAX),
     +                 LAMDA(NXEST), MU(NYEST), PX(NMAX), PY(NMAX),
     +                 W(MMAX), WRK(LWRK), X(MMAX), Y(MMAX)
      INTEGER          IWRK(LIWRK)
*     .. External Subroutines ..
      EXTERNAL         CPRINT, E02DDF, E02DFF
*     .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02DDF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Input the number of data-points M.
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
*        Input the data-points and the weights.
         DO 20 I = 1, M
            READ (NIN,*) X(I), Y(I), F(I), W(I)
   20    CONTINUE
         START = 'Cold Start'
         READ (NIN,*,END=120) S
*        Determine the spline approximation.
         IFAIL = 0
*
         CALL E02DDF(START,M,X,Y,F,W,S,NXEST,NYEST,NX,LAMDA,NY,MU,C,FP,
     +              RANK,WRK,LWRK,IWRK,LIWRK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calling with smoothing factor S =', S,
     +     ': NX =', NX, ', NY =', NY, ','
         WRITE (NOUT,99998) 'rank deficiency =', (NX-4)*(NY-4) - RANK
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     '               I   Knot LAMDA(I)      J     Knot MU(J)'
         WRITE (NOUT,*)
         DO 40 J = 4, MAX(NX,NY) - 3
            IF (J.LE.NX-3 .AND. J.LE.NY-3) THEN
               WRITE (NOUT,99996) J, LAMDA(J), J, MU(J)
```

```
            ELSE IF (J.LE.NX-3) THEN
               WRITE (NOUT,99996) J, LAMDA(J)
            ELSE IF (J.LE.NY-3) THEN
               WRITE (NOUT,99995) J, MU(J)
            END IF
   40     CONTINUE
          CALL CPRINT(C,NY,NX,NOUT)
          WRITE (NOUT,*)
          WRITE (NOUT,99997) ' Sum of squared residuals FP =', FP
          IF (NX.EQ.8 .AND. NY.EQ.8) THEN
             WRITE (NOUT,*)
     +         ' ( The spline is the least-squares bi-cubic polynomial )'
          END IF
*         Evaluate the spline on a rectangular grid at NPX*NPY points
*         over the domain (XLO to XHI) x (YLO to YHI).
          READ (NIN,*) NPX, XLO, XHI
          READ (NIN,*) NPY, YLO, YHI
          IF (NPX.LE.NMAX .AND. NPY.LE.NMAX) THEN
             DELTA = (XHI-XLO)/(NPX-1)
             DO 60 I = 1, NPX
                PX(I) = MIN(XLO+(I-1)*DELTA,XHI)
   60        CONTINUE
             DO 80 I = 1, NPY
                PY(I) = MIN(YLO+(I-1)*DELTA,YHI)
   80        CONTINUE
*
             CALL E02DFF(NPX,NPY,NX,NY,PX,PY,LAMDA,MU,C,FG,WRK,LWRK,IWRK,
     +                  LIWRK,IFAIL)
*
             WRITE (NOUT,*)
             WRITE (NOUT,*) 'Values of computed spline:'
             WRITE (NOUT,*)
             WRITE (NOUT,99994) '          X', (PX(I),I=1,NPX)
             WRITE (NOUT,*) '     Y'
             DO 100 I = NPY, 1, -1
                WRITE (NOUT,99993) PY(I), (FG(NPY*(J-1)+I),J=1,NPX)
  100        CONTINUE
          END IF
       END IF
  120  CONTINUE
       STOP
*
99999  FORMAT (1X,A,1P,e13.4,A,I5,A,I5,A)
99998  FORMAT (1X,A,I5)
99997  FORMAT (1X,A,1P,e13.4,A)
99996  FORMAT (1X,I16,F12.4,I11,F12.4)
99995  FORMAT (1X,I39,F12.4)
99994  FORMAT (1X,A,7F8.2)
99993  FORMAT (1X,F8.2,3X,7F8.2)
       END
*
       SUBROUTINE CPRINT(C,NY,NX,NOUT)
*      .. Scalar Arguments ..
       INTEGER           NOUT, NX, NY
*      .. Array Arguments ..
       real              C(NY-4,NX-4)
*      .. Local Scalars ..
       INTEGER           I, J
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The B-spline coefficients:'
      WRITE (NOUT,*)
      DO 20 I = 1, NY - 4
         WRITE (NOUT,99999) (C(I,J),J=1,NX-4)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,7F9.2)
      END
```

9.2 Program Data

```
E02DDF Example Program Data
 30                                  M, number of data points
 11.16   1.24  22.15  1.00  X,Y,F,W data point coordinates and weight
 12.85   3.06  22.11  1.00
 19.85  10.72   7.97  1.00
 19.72   1.39  16.83  1.00
 15.91   7.74  15.30  1.00
  0.00  20.00  34.60  1.00
 20.87  20.00   5.74  1.00
  3.45  12.78  41.24  1.00
 14.26  17.87  10.74  1.00
 17.43   3.46  18.60  1.00
 22.80  12.39   5.47  1.00
  7.58   1.98  29.87  1.00
 25.00  11.87   4.40  1.00
  0.00   0.00  58.20  1.00
  9.66  20.00   4.73  1.00
  5.22  14.66  40.36  1.00
 17.25  19.57   6.43  1.00
 25.00   3.87   8.74  1.00
 12.13  10.79  13.71  1.00
 22.23   6.21  10.25  1.00
 11.52   8.53  15.74  1.00
 15.20   0.00  21.60  1.00
  7.54  10.69  19.31  1.00
 17.32  13.78  12.11  1.00
  2.14  15.03  53.10  1.00
  0.51   8.37  49.43  1.00
 22.69  19.63   3.25  1.00
  5.47  17.13  28.63  1.00
 21.67  14.36   5.52  1.00
  3.31   0.33  44.08  1.00  End of data points
 10.0                        S, smoothing factor
   7    3.0   21.0
   6    2.0   17.0
```

9.3 Program Results

```
E02DDF Example Program Results

Calling with smoothing factor S =   1.0000E+01: NX =   10, NY =    9,
rank deficiency =    0

              I   Knot LAMDA(I)      J     Knot MU(J)

              4      0.0000          4      0.0000
              5      9.7575          5      9.0008
              6     18.2582          6     20.0000
              7     25.0000

The B-spline coefficients:

    58.16    46.31     6.01    32.00     5.86   -23.78
    63.78    46.74    33.37    18.30    14.36    15.95
    40.84   -33.79     5.17    13.10    -4.13    19.37
    75.44   111.92     6.94    17.33     7.09   -13.24
    34.61   -42.61    25.20    -1.96    10.37    -9.09

 Sum of squared residuals FP =   1.0002E+01

Values of computed spline:

          X    3.00    6.00    9.00   12.00   15.00   18.00   21.00
    Y
  17.00       40.74   28.62   19.84   14.29   11.21    9.46    7.09
  14.00       48.34   33.97   21.56   14.71   12.32   10.82    7.15
  11.00       37.26   24.46   17.21   14.14   13.02   11.23    7.29
   8.00       30.25   19.66   16.90   16.28   15.21   12.71    8.99
   5.00       36.64   26.75   23.07   21.13   18.97   15.90   11.98
   2.00       45.04   33.70   26.25   22.88   21.62   19.39   13.40
```

E02DEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02DEF calculates values of a bicubic spline from its B-spline representation.

2. Specification

```
SUBROUTINE E02DEF (M, PX, PY, X, Y, LAMDA, MU, C, FF, WRK, IWRK, IFAIL)
INTEGER          M, PX, PY, IWRK(PY-4), IFAIL
real             X(M), Y(M), LAMDA(PX), MU(PY), C((PX-4)*(PY-4)), FF(M),
1                WRK(PY-4)
```

3. Description

This routine calculates values of the bicubic spline $s(x,y)$ at prescribed points (x_r,y_r), for $r = 1,2,...,m$, from its augmented knot sets $\{\lambda\}$ and $\{\mu\}$ and from the coefficients c_{ij}, for $i = 1,2,...,PX-4$; $j = 1,2,...,PY-4$, in its B-spline representation

$$s(x,y) = \sum_{i,j} c_{ij} M_i(x) N_j(y) .$$

Here $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}.

This routine may be used to calculate values of a bicubic spline given in the form produced by E01DAF, E02DAF, E02DCF and E02DDF. It is derived from the routine B2VRE in Anthony *et al.* [1].

4. References

[1] ANTHONY, G.T., COX, M.G. and HAYES, J.G.
DASL – Data Approximation Subroutine Library, National Physical Laboratory, 1982.

[2] COX, M.G.
The numerical evaluation of a spline from its B-spline representation.
J. Inst. Maths. Applics., 21, pp. 135–143, 1978.

5. Parameters

1: M – INTEGER. *Input*

On entry: m, the number of points at which values of the spline are required.

Constraint: M ≥ 1.

2: PX – INTEGER. *Input*

3: PY – INTEGER. *Input*

On entry: PX and PY must specify the total number of knots associated with the variables x and y respectively. They are such that PX−8 and PY−8 are the corresponding numbers of interior knots.

Constraint: PX ≥ 8 and PY ≥ 8.

4: X(M) – ***real*** array. *Input*

5: Y(M) – ***real*** array. *Input*

On entry: X and Y must contain x_r and y_r, for $r = 1,2,...,m$, respectively. These are the co-ordinates of the points at which values of the spline are required. The order of the points is immaterial.

Constraint: X and Y must satisfy

$$\text{LAMDA}(4) \leq \text{X}(r) \leq \text{LAMDA}(\text{PX}-3)$$

and

$$\text{MU}(4) \leq \text{Y}(r) \leq \text{MU}(\text{PY}-3), \quad \text{for } r = 1,2,\ldots,m.$$

The spline representation is not valid outside these intervals.

6: LAMDA(PX) – ***real*** array. *Input*

7: MU(PY) – ***real*** array. *Input*

On entry: LAMDA and MU must contain the complete sets of knots $\{\lambda\}$ and $\{\mu\}$ associated with the x and y variables respectively.

Constraint: the knots in each set must be in non-decreasing order, with LAMDA(PX–3) > LAMDA(4) and MU(PY–3) > MU(4).

8: C((*PX*–4)*(PY–4)) – ***real*** array. *Input*

On entry: C((PY–4)×(*i*–1)+*j*) must contain the coefficient c_{ij} described in Section 3, for $i = 1,2,\ldots,\text{PX}-4$; $j = 1,2,\ldots,\text{PY}-4$.

9: FF(M) – ***real*** array. *Output*

On exit: FF(r) contains the value of the spline at the point (x_r,y_r), for $r = 1,2,\ldots,m$.

10: WRK(PY–4) – ***real*** array. *Workspace*

11: IWRK(PY–4) – INTEGER array. *Workspace*

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, M < 1,
or PY < 8,
or PX < 8.

IFAIL = 2

On entry, the knots in array LAMDA, or those in array MU, are not in non-decreasing order, or LAMDA(PX–3) ≤ LAMDA(4), or MU(PY–3) ≤ MU(4).

IFAIL = 3

On entry, at least one of the prescribed points (x_r,y_r) lies outside the rectangle defined by LAMDA(4), LAMDA(PX–3) and MU(4), MU(PY–3).

7. Accuracy

The method used to evaluate the B-splines is numerically stable, in the sense that each computed value of $s(x_r,y_r)$ can be regarded as the value that would have been obtained in exact arithmetic from slightly perturbed B-spline coefficients. See Cox [2] for details.

8. Further Comments

Computation time is approximately proportional to the number of points, m, at which the evaluation is required.

9. Example

This program reads in knot sets LAMDA(1),...,LAMDA(PX) and MU(1),...,MU(PY), and a set of bicubic spline coefficients c_{ij}. Following these are a value for m and the co-ordinates (x_r,y_r), for $r = 1,2,...,m$, at which the spline is to be evaluated.

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.

```
*     E02DEF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, PXMAX, PYMAX
      PARAMETER        (MMAX=20,PXMAX=MMAX,PYMAX=PXMAX)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, M, PX, PY
*     .. Local Arrays ..
      real             C((PXMAX-4)*(PYMAX-4)), FF(MMAX), LAMDA(PXMAX),
     +                 MU(PYMAX), WRK(PYMAX-7), X(MMAX), Y(MMAX)
      INTEGER          IWRK(PYMAX-7)
*     .. External Subroutines ..
      EXTERNAL         E02DEF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02DEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Read PX and PY, the number of knots in the X and Y directions.
      READ (NIN,*) PX, PY
      IF (PX.LE.PXMAX .AND. PY.LE.PYMAX) THEN
*        Read the knots LAMDA(1) .. LAMDA(PX) and MU(1) .. MU(PY).
         READ (NIN,*) (LAMDA(I),I=1,PX)
         READ (NIN,*) (MU(I),I=1,PY)
*        Read C, the bicubic spline coefficients.
         READ (NIN,*) (C(I),I=1,(PX-4)*(PY-4))
*        Read M, the number of spline evaluation points.
         READ (NIN,*) M
         IF (M.LE.MMAX) THEN
*           Read the X and Y co-ordinates of the evaluation points.
            DO 20 I = 1, M
               READ (NIN,*) X(I), Y(I)
   20       CONTINUE
            IFAIL = 0
*
*           Evaluate the spline at the M points.
            CALL E02DEF(M,PX,PY,X,Y,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
*
*           Print the results.
            WRITE (NOUT,*)
            WRITE (NOUT,*) '      I       X(I)       Y(I)      FF(I)'
            WRITE (NOUT,99999) (I,X(I),Y(I),FF(I),I=1,M)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,I7,3F11.3)
      END
```

9.2. Program Data

```
E02DEF Example Program Data
11  10                                                        PX  PY
1.0  1.0  1.0  1.0  1.3  1.5  1.6  2.0  2.0  2.0  2.0
LAMDA(1) .. LAMDA(PX)
0.0  0.0  0.0  0.0  0.4  0.7  1.0  1.0  1.0  1.0            MU(1) .. MU(PY)
1.0000   1.1333   1.3667   1.7000   1.9000   2.0000
1.2000   1.3333   1.5667   1.9000   2.1000   2.2000
1.5833   1.7167   1.9500   2.2833   2.4833   2.5833
2.1433   2.2767   2.5100   2.8433   3.0433   3.1433
2.8667   3.0000   3.2333   3.5667   3.7667   3.8667
3.4667   3.6000   3.8333   4.1667   4.3667   4.4667
4.0000   4.1333   4.3667   4.7000   4.9000   5.0000
Spline coefficients, C
7                                                           M
1.0  0.0                                                    X(1), Y(1)
1.1  0.1
1.5  0.7
1.6  0.4
1.9  0.3
1.9  0.8
2.0  1.0                                                    X(M), Y(M)
```

9.3. Program Results

```
 E02DEF Example Program Results

      I        X(I)        Y(I)       FF(I)
      1       1.000       0.000       1.000
      2       1.100       0.100       1.310
      3       1.500       0.700       2.950
      4       1.600       0.400       2.960
      5       1.900       0.300       3.910
      6       1.900       0.800       4.410
      7       2.000       1.000       5.000
```

E02DFF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02DFF calculates values of a bicubic spline from its B-spline representation. The spline is evaluated at all points on a rectangular grid.

2. Specification

```
      SUBROUTINE E02DFF (MX, MY, PX, PY, X, Y, LAMDA, MU, C, FF, WRK, LWRK,
     1                   IWRK, LIWRK, IFAIL)
      INTEGER          MX, MY, PX, PY, LWRK, IWRK(LIWRK), LIWRK, IFAIL
      real             X(MX), Y(MY), LAMDA(PX), MU(PY), C((PX-4)*(PY-4)),
     1                 FF(MX*MY), WRK(LWRK)
```

3. Description

This routine calculates values of the bicubic spline $s(x,y)$ on a rectangular grid of points in the x-y plane, from its augmented knot sets $\{\lambda\}$ and $\{\mu\}$ and from the coefficients c_{ij}, for $i = 1,2,...,PX-4$; $j = 1,2,...,PY-4$, in its B-spline representation

$$s(x,y) = \sum_{i,j} c_{ij} M_i(x) N_j(y) .$$

Here $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}.
The points in the grid are defined by co-ordinates x_q, for $q = 1,2,...,m_x$, along the x axis, and co-ordinates y_r, for $r = 1,2,...,m_y$ along the y axis.

This routine may be used to calculate values of a bicubic spline given in the form produced by E01DAF, E02DAF, E02DCF and E02DDF. It is derived from the routine B2VRE in Anthony *et al.* [1].

4. References

[1] ANTHONY, G.T., COX, M.G. and HAYES, J.G.
DASL – Data Approximation Subroutine Library, National Physical Laboratory, 1982.

[2] COX, M.G.
The numerical evaluation of a spline from its B-spline representation.
J. Inst. Maths. Applics., 21, pp. 135–143, 1978.

5. Parameters

1: MX – INTEGER. *Input*

2: MY – INTEGER. *Input*

On entry: MX and MY must specify m_x and m_y respectively, the number of points along the x and y axis that define the rectangular grid.

Constraint: MX $\geq$ 1 and MY $\geq$ 1.

3: PX – INTEGER. *Input*

4: PY – INTEGER. *Input*

On entry: PX and PY must specify the total number of knots associated with the variables x and y respectively. They are such that PX–8 and PY–8 are the corresponding numbers of interior knots.

Constraint: PX $\geq$ 8 and PY $\geq$ 8.

5: X(MX) – ***real*** array. *Input*

6: Y(MY) – ***real*** array. *Input*

On entry: X and Y must contain x_q, for $q = 1,2,...,m_x$, and y_r, for $r = 1,2,...,m_y$, respectively. These are the x and y co-ordinates that define the rectangular grid of points at which values of the spline are required.

Constraint: X and Y must satisfy

$$\text{LAMDA}(4) \leq \text{X}(q) < \text{X}(q+1) \leq \text{LAMDA}(\text{PX}-3), \quad \text{for } q = 1,2,...,m_x-1,$$

and

$$\text{MU}(4) \leq \text{Y}(r) < \text{Y}(r+1) \leq \text{MU}(\text{PY}-3), \quad \text{for } r = 1,2,...,m_y-1.$$

The spline representation is not valid outside these intervals.

7: LAMDA(PX) – ***real*** array. *Input*

8: MU(PY) – ***real*** array. *Input*

On entry: LAMDA and MU must contain the complete sets of knots $\{\lambda\}$ and $\{\mu\}$ associated with the x and y variables respectively.

Constraint: the knots in each set must be in non-decreasing order, with LAMDA(PX–3) > LAMDA(4) and MU(PY–3) > MU(4).

9: C((PX–4)*(PY–4)) – ***real*** array. *Input*

On entry: C((PY–4)×(*i*–1)+*j*) must contain the coefficient c_{ij} described in Section 3, for $i = 1,2,...,\text{PX}-4$; $j = 1,2,...,\text{PY}-4$.

10: FF(MX*MY) – ***real*** array. *Output*

On exit: FF(MY×(*q*–1)+*r*) contains the value of the spline at the point (x_q,y_r), for $q = 1,2,...,m_x$; $r = 1,2,...,m_y$.

11: WRK(LWRK) – ***real*** array. *Workspace*

12: LWRK – INTEGER. *Input*

On entry: the dimension of the array WRK as declared in the (sub)program from which E02DFF is called.

Constraint: LWRK ≥ min(NWRK1,NWRK2), where NWRK1 = 4×MX + PX, NWRK2 = 4×MY + PY.

13: IWRK(LIWRK) – INTEGER array. *Workspace*

14: LIWRK – INTEGER. *Input*

On entry: the dimension of the array IWRK as declared in the (sub)program from which E02DFF is called.

Constraint: LIWRK ≥ MY + PY – 4 if NWRK1 > NWRK2, or MX + PX – 4 otherwise, where NWRK1 and NWRK2 are as defined in the description of argument LWRK.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, MX < 1,
or MY < 1,
or PY < 8,
or PX < 8.

IFAIL = 2

On entry, LWRK is too small,
or LIWRK is too small.

IFAIL = 3

On entry, the knots in array LAMDA, or those in array MU, are not in non-decreasing order, or LAMDA(PX−3) ≤ LAMDA(4), or MU(PY−3) ≤ MU(4).

IFAIL = 4

On entry, the restriction LAMDA(4) ≤ X(1) < ... < X(MX) ≤ LAMDA(PX−3), or the restriction MU(4) ≤ Y(1) < ... < Y(MY) ≤ MU(PY−3), is violated.

7. Accuracy

The method used to evaluate the B-splines is numerically stable, in the sense that each computed value of $s(x_r,y_r)$ can be regarded as the value that would have been obtained in exact arithmetic from slightly perturbed B-spline coefficients. See Cox [2] for details.

8. Further Comments

Computation time is approximately proportional to $m_x m_y + 4(m_x+m_y)$.

9. Example

This program reads in knot sets LAMDA(1),...,LAMDA(PX) and MU(1),...,MU(PY), and a set of bicubic spline coefficients c_{ij}. Following these are values for m_x and the x co-ordinates x_q, for $q = 1,2,...m_x$, and values for m_y and the y co-ordinates y_r, for $r = 1,2,...m_y$, defining the grid of points on which the spline is to be evaluated.

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.

```
*     E02DFF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MXMAX, MYMAX, PXMAX, PYMAX
      PARAMETER        (MXMAX=20,MYMAX=MXMAX,PXMAX=MXMAX,PYMAX=PXMAX)
      INTEGER          LIWRK, LWRK
      PARAMETER        (LIWRK=MXMAX+PXMAX-4,LWRK=4*MXMAX+PXMAX+8)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, MX, MY, PX, PY
*     .. Local Arrays ..
      real             C((PXMAX-4)*(PYMAX-4)), FF(MXMAX*MYMAX),
     +                 LAMDA(PXMAX), MU(PYMAX), WRK(LWRK), X(MXMAX),
     +                 Y(MYMAX)
      INTEGER          IWRK(LIWRK)
      CHARACTER*10     CLABS(MYMAX), RLABS(MXMAX)
*     .. External Subroutines ..
      EXTERNAL         E02DFF, X04CBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02DFF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
*     Read PX and PY, the number of knots in the X and Y directions.
      READ (NIN,*) PX, PY
      IF (PX.LE.PXMAX .AND. PY.LE.PYMAX) THEN
*        Read the knots LAMDA(1) .. LAMDA(PX) and MU(1) .. MU(PY).
         READ (NIN,*) (LAMDA(I),I=1,PX)
         READ (NIN,*) (MU(I),I=1,PY)
*        Read C, the bicubic spline coefficients.
         READ (NIN,*) (C(I),I=1,(PX-4)*(PY-4))
*        Read MX and MY, the number of grid points in the X and Y
*        directions respectively.
         READ (NIN,*) MX, MY
         IF (MX.LE.MXMAX .AND. MY.LE.MYMAX) THEN
*           Read the X and Y co-ordinates defining the evaluation grid.
            READ (NIN,*) (X(I),I=1,MX)
            READ (NIN,*) (Y(I),I=1,MY)
            IFAIL = 0
*
*           Evaluate the spline at the MX by MY points.
            CALL E02DFF(MX,MY,PX,PY,X,Y,LAMDA,MU,C,FF,WRK,LWRK,IWRK,
     +                  LIWRK,IFAIL)
*
*           Generate column and row labels to print the results with.
            DO 20 I = 1, MX
               WRITE (CLABS(I),99999) X(I)
   20       CONTINUE
            DO 40 I = 1, MY
               WRITE (RLABS(I),99999) Y(I)
   40       CONTINUE
*
*           Print the result array.
            CALL X04CBF('G','X',MY,MX,FF,MY,'F8.3',
     +                'Spline evaluated on X-Y grid (X across, Y down):'
     +                  ,'Character',RLABS,'Character',CLABS,80,0,IFAIL)
*
         END IF
      END IF
      STOP
*
99999 FORMAT (F5.1)
      END
```

9.2. Program Data

```
E02DFF Example Program Data
11  10                                                      PX  PY
1.0  1.0  1.0  1.0  1.3  1.5  1.6  2.0  2.0  2.0  2.0      LAMDA(1) .. LAMDA(PX)
0.0  0.0  0.0  0.0  0.4  0.7  1.0  1.0  1.0  1.0           MU(1) .. MU(PY)
1.0000   1.1333   1.3667   1.7000   1.9000   2.0000
1.2000   1.3333   1.5667   1.9000   2.1000   2.2000
1.5833   1.7167   1.9500   2.2833   2.4833   2.5833
2.1433   2.2767   2.5100   2.8433   3.0433   3.1433
2.8667   3.0000   3.2333   3.5667   3.7667   3.8667
3.4667   3.6000   3.8333   4.1667   4.3667   4.4667
4.0000   4.1333   4.3667   4.7000   4.9000   5.0000
Spline coefficients, C
7  6                                                       MX  MY
1.0  1.1  1.3  1.4  1.5  1.7  2.0                         X(1) .. X(MX)
0.0  0.2  0.4  0.6  0.8  1.0                             Y(1) .. Y(MY)
```

9.3. Program Results

```
E02DFF Example Program Results

Spline evaluated on X-Y grid (X across, Y down):
          1.0     1.1     1.3     1.4     1.5     1.7     2.0
0.0     1.000   1.210   1.690   1.960   2.250   2.890   4.000
0.2     1.200   1.410   1.890   2.160   2.450   3.090   4.200
0.4     1.400   1.610   2.090   2.360   2.650   3.290   4.400
0.6     1.600   1.810   2.290   2.560   2.850   3.490   4.600
0.8     1.800   2.010   2.490   2.760   3.050   3.690   4.800
1.0     2.000   2.210   2.690   2.960   3.250   3.890   5.000
```

E02GAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02GAF calculates an l_1 solution to an over-determined system of linear equations.

2. Specification

```
      SUBROUTINE E02GAF (M, A, LA, B, NPLUS2, TOLER, X, RESID, IRANK, ITER,
     1                   IWORK, IFAIL)
      INTEGER          M, LA, NPLUS2, IRANK, ITER, IWORK(M), IFAIL
      real             A(LA,NPLUS2), B(M), TOLER, X(NPLUS2), RESID
```

3. Description

Given a matrix A with m rows and n columns ($m \geq n$) and a vector b with m elements, the routine calculates an l_1 solution to the over-determined system of equations

$$Ax = b.$$

That is to say, it calculates a vector x, with n elements, which minimizes the l_1 norm (the sum of the absolute values) of the residuals

$$r(x) = \sum_{i=1}^{m} |r_i|,$$

where the residuals r_i are given by

$$r_i = b_i - \sum_{j=1}^{n} a_{ij}x_j, \qquad i = 1,2,...,m.$$

Here a_{ij} is the element in row i and column j of A, b_i is the ith element of b and x_j the jth element of x. The matrix A need not be of full rank.

Typically in applications to data fitting, data consisting of m points with co-ordinates (t_i,y_i) are to be approximated in the l_1 norm by a linear combination of known functions $\phi_j(t)$,

$$\alpha_1\phi_1(t) + \alpha_2\phi_2(t) + ... + \alpha_n\phi_n(t).$$

This is equivalent to fitting an l_1 solution to the over-determined system of equations

$$\sum_{j=1}^{n} \phi_j(t_i)\ \alpha_j = y_i, \qquad i = 1,2,...,m.$$

Thus if, for each value of i and j, the element a_{ij} of the matrix A in the previous paragraph is set equal to the value of $\phi_j(t_i)$ and b_i is set equal to y_i, the solution vector x will contain the required values of the α_j. Note that the independent variable t above can, instead, be a vector of several independent variables (this includes the case where each ϕ_i is a function of a different variable, or set of variables).

The algorithm is a modification of the simplex method of linear programming applied to the primal formulation of the l_1 problem (see Barrodale and Roberts [1] and [2]). The modification allows several neighbouring simplex vertices to be passed through in a single iteration, providing a substantial improvement in efficiency.

4. References

[1] BARRODALE, I. and ROBERTS, F.D.K.
An improved algorithm for discrete l_1 linear approximation.
SIAM J. Numer. Anal., 10, pp. 839-848, 1973.

[2] BARRODALE, I. and ROBERTS, F.D.K.
Solution of an overdetermined system of equations in the l_1-norm.
ACM Comm., 17, No. 6, Algorithm 478, pp. 319-320, 1974.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of equations, m (the number of rows of the matrix A).

Constraint: $M \geq n \geq 1$.

2: A(LA,NPLUS2) – ***real*** array. *Input/Output*

On entry: A(i,j) must contain a_{ij}, the element in the ith row and jth column of the matrix A, for $i = 1,2,...,m$ and $j = 1,2,...,n$. The remaining elements need not be set.

On exit: A contains the last simplex tableau generated by the simplex method.

3: LA – INTEGER. *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which E02GAF is called.

Constraint: LA $\geq$ M + 2.

4: B(M) – ***real*** array. *Input/Output*

On entry: b_i, the ith element of the vector b, for $i = 1,2,...,m$.

On exit: the ith residual r_i corresponding to the solution vector x, for $i = 1,2,...,m$.

5: NPLUS2 – INTEGER. *Input*

On entry: $n + 2$, where n is the number of unknowns (the number of columns of the matrix A).

Constraint: 3 $\leq$ NPLUS2 $\leq$ M + 2.

6: TOLER – ***real***. *Input*

On entry: a non-negative value. In general TOLER specifies a threshold below which numbers are regarded as zero. The recommended threshold value is $\epsilon^{2/3}$ where ϵ is the ***machine precision***. The recommended value can be computed within the routine by setting TOLER to zero. If premature termination occurs a larger value for TOLER may result in a valid solution.

Suggested value: 0.0.

7: X(NPLUS2) – ***real*** array. *Output*

On exit: X(j) contains the jth element of the solution vector x, for $j = 1,2,...,n$. The elements X(n+1) and X(n+2) are unused.

8: RESID – ***real***. *Output*

On exit: the sum of the absolute values of the residuals for the solution vector x.

9: IRANK – INTEGER. *Output*

On exit: the computed rank of the matrix A.

10: ITER – INTEGER. *Output*

On exit: the number of iterations taken by the simplex method.

11: IWORK(M) – INTEGER array. *Workspace*

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

An optimal solution has been obtained but this may not be unique.

IFAIL = 2

The calculations have terminated prematurely due to rounding errors. Experiment with larger values of TOLER or try scaling the columns of the matrix (see Section 8).

IFAIL = 3

On entry, NPLUS2 < 3,
or NPLUS2 > M + 2,
or LA < M + 2.

7. Accuracy

Experience suggests that the computational accuracy of the solution x is comparable with the accuracy that could be obtained by applying Gaussian elimination with partial pivoting to the n equations satisfied by this algorithm (i.e. those equations with zero residuals). The accuracy therefore varies with the conditioning of the problem, but has been found generally very satisfactory in practice.

8. Further Comments

The effects of m and n on the time and on the number of iterations in the Simplex Method vary from problem to problem, but typically the number of iterations is a small multiple of n and the total time taken by the routine is approximately proportional to mn^2.

It is recommended that, before the routine is entered, the columns of the matrix A are scaled so that the largest element in each column is of the order of unity. This should improve the conditioning of the matrix, and also enable the parameter TOLER to perform its correct function. The solution x obtained will then, of course, relate to the scaled form of the matrix. Thus if the scaling is such that, for each $j = 1,2,...,n$, the elements of the jth column are multiplied by the constant k_j, the element x_j of the solution vector x must be multiplied by k_j if it is desired to recover the solution corresponding to the original matrix A.

9. Example

Suppose we wish to approximate a set of data by a curve of the form

$$y = Ke^t + Le^{-t} + M$$

where K, L and M are unknown. Given values y_i at 5 points t_i we may form the over-determined set of equations for K, L and M

$$e^{x_i} K + e^{-x_i} L + M = y_i, \qquad i = 1,2,...,5.$$

E02GAF is used to solve these in the l_1 sense.

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.

```
*     E02GAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, LA, NPLUS2
      PARAMETER        (MMAX=5,LA=MMAX+2,NPLUS2=5)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             RESID, T, TOL
      INTEGER          I, IFAIL, ITER, M, RANK
*     .. Local Arrays ..
      real             A(LA,NPLUS2), B(MMAX), X(NPLUS2)
      INTEGER          IWORK(MMAX)
*     .. External Subroutines ..
      EXTERNAL         E02GAF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02GAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         DO 20 I = 1, M
            READ (NIN,*) T, B(I)
            A(I,1) = EXP(T)
            A(I,2) = EXP(-T)
            A(I,3) = 1.0e0
   20    CONTINUE
         TOL = 0.0e0
         IFAIL = 1
*
         CALL E02GAF(M,A,LA,B,NPLUS2,TOL,X,RESID,RANK,ITER,IWORK,IFAIL)
*
         IF (IFAIL.LE.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Resid = ', RESID, '  Rank = ', RANK,
     +        '  Iterations = ', ITER, '  IFAIL =', IFAIL
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Solution'
            WRITE (NOUT,99998) (X(I),I=1,NPLUS2-2)
         ELSE
            WRITE (NOUT,99997) 'E02GAF fails with error', IFAIL
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,e10.2,A,I5,A,I5,A,I5)
99998 FORMAT (1X,6F10.4)
99997 FORMAT (1X,A,I2)
      END
```

9.2. Program Data

```
E02GAF Example Program Data
  5
   0.0 4.501
   0.2 4.360
   0.4 4.333
   0.6 4.418
   0.8 4.625
```

9.3. Program Results

```
E02GAF Example Program Results

Resid =   0.28E-02  Rank =     3  Iterations =     5  IFAIL =    0

Solution
    1.0014    2.0035    1.4960
```

E02GBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02GBF calculates an l_1 solution to an over-determined system of linear equations, possibly subject to linear inequality constraints.

2. Specification

```
      SUBROUTINE E02GBF (M, N, MPL, E, IE, F, X, MXS, MONIT, IPRINT, K,
     1                   EL1N, INDX, W, IW, IFAIL)
      INTEGER           M, N, MPL, IE, MXS, IPRINT, K, INDX(MPL), IW,
     1                  IFAIL
      real              E(IE,MPL), F(MPL), X(N), EL1N, W(IW)
      EXTERNAL          MONIT
```

3. Description

Given a matrix A with m rows and n columns ($m \geq n$) and a vector b with m elements, the routine calculates an l_1 solution to the over-determined system of equations

$$Ax = b.$$

That is to say, it calculates a vector x, with n elements, which minimises the l_1-norm (the sum of the absolute values) of the residuals

$$r(x) = \sum_{i=1}^{m} |r_i|,$$

where the residuals r_i are given by

$$r_i = b_i - \sum_{j=1}^{n} a_{ij}x_j, \qquad i = 1,2,...,m.$$

Here a_{ij} is the element in row i and column j of A, b_i is the ith element of b and x_j the jth element of x.

If, in addition, a matrix C with l rows and n columns and a vector d with l elements, are given, the vector x computed by the routine is such as to minimize the l_1-norm $r(x)$ subject to the set of inequality constraints $Cx \geq d$.

The matrices A and C need not be of full rank.

Typically in applications to data fitting, data consisting of m points with co-ordinates (t_i,y_i) is to be approximated by a linear combination of known functions $\phi_i(t)$,

$$\alpha_1\phi_1(t) + \alpha_2\phi_2(t) + ... + \alpha_n\phi_n(t),$$

in the l_1-norm, possibly subject to linear inequality constraints on the coefficients α_j of the form $C\alpha \geq d$ where α is the vector of the α_j and C and d are as in the previous paragraph. This is equivalent to finding an l_1 solution to the over-determined system of equations

$$\sum_{j=1}^{n} \phi_j(t_i)\ \alpha_j = y_i, \qquad i = 1,2,...,m,$$

subject to $C\alpha \geq d$.

Thus if, for each value of i and j, the element a_{ij} of the matrix A above is set equal to the value of $\phi_j(t_i)$ and b_i is equal to y_i and C and d are also supplied to the routine, the solution vector x will contain the required values of the α_j. Note that the independent variable t above can, instead, be a vector of several independent variables (this includes the case where each of ϕ_i is a function of a different variable, or set of variables).

The algorithm follows the Conn-Pietrzykowski approach (see Bartels *et al.* [1] and Conn and Pietrzykowski [2]), which is via an exact penalty function

$$g(x) = \gamma r(x) - \sum_{i=1}^{l} \min(0, c_i^T x - d_i),$$

where γ is a penalty parameter, c_i^T is the ith row of the matrix C, and d_i is the ith element of the vector d. It proceeds in a step-by-step manner much like the simplex method for linear programming but does not move from vertex to vertex and does not require the problem to be cast in a form containing only non-negative unknowns. It uses stable procedures to update an orthogonal factorization of the current set of active equations and constraints.

4. References

[1] BARTELS, R.H., CONN, A.R. and SINCLAIR, J.W.
Minimisation Techniques for Piecewise Differentiable Functions – the l_1 Solution to an Overdetermined Linear System.
SIAM J. Numer. Anal., 15, pp. 224-241, 1978.

[2] CONN, A.R. and PIETRZYKOWSKI, T.
A Penalty-function Method Converging Directly to a Constrained Optimum.
SIAM J. Numer. Anal., 14, pp. 348-375, 1977.

[3] BARTELS, R.H., CONN, A.R. and CHARALAMBOUS, C.
Minimisation Techniques for Piecewise Differentiable Functions – the l_∞ Solution to an Overdetermined Linear System.
The Johns Hopkins University, Mathematical Sciences Department, Technical Report No. 247, June 1976/Department of Combinatorics and Optimization, Research Report CORR 76/30.

[4] BARTELS, R.H., CONN, A.R. and SINCLAIR, J.W.
A Fortran Program for Solving Overdetermined Systems of Linear Equations in the l_1 Sense.
The Johns Hopkins University, Mathematical Sciences Department, Technical Report No. 236, January 1976/Department of Combinatorics and Optimization, Research Report CORR 76/7.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of equations in the over-determined system, m (i.e. the number of rows of the matrix A).

Constraint: M $\geq$ 2.

2: N – INTEGER. *Input*

On entry: the number of unknowns, n (the number of columns of the matrix A).

Constraint: M $\geq$ N $\geq$ 2.

3: MPL – INTEGER. *Input*

On entry: $m + l$, where l is the number of constraints (which may be zero).

Constraint: MPL $\geq$ M.

4: E(IE,MPL) – ***real*** array. *Input/Output*

On entry: the equation and constraint matrices stored in the following manner:
The first m columns contain the m rows of the matrix A; element E(i,j) specifying the element a_{ji} in the jth row and ith column of A (the coefficient of the ith unknown in the jth equation), for i = 1,2,...,n; j = 1,2,...,m. The next l columns contain the l rows of the constraint matrix C; element E(i,j+m) containing the element c_{ji} in the jth row and ith column of C (the coefficient of the ith unknown in the jth constraint), for i = 1,2,...,n; j = 1,2,...,l.

On exit: unchanged, except possibly to the extent of a small multiple of the ***machine precision***. (See Section 8.)

5: IE – INTEGER. *Input*

On entry: the first dimension of the array E as declared in the (sub)program from which E02GBF is called.

Constraint: IE ≥ N.

6: F(MPL) – ***real*** array. *Input*

On entry: F(i), for i = 1,2,...,m must contain b_i (the ith element of the right-hand side vector of the over-determined system of equations) and F(m+i), for i = 1,2,...,l must contain d_i (the ith element of the right-hand side vector of the constraints), where l is the number of constraints.

7: X(N) – ***real*** array. *Input/Output*

On entry: X(i) must contain an estimate of the ith unknown, for i = 1,2,...,n. If no better initial estimate for X(i) is available, set X(i) = 0.0.

On exit: the latest estimate of the ith unknown, for i = 1,2,...,n. If IFAIL = 0 on exit, these are the solution values.

8: MXS – INTEGER. *Input*

On entry: the maximum number of steps to be allowed for the solution of the unconstrained problem. Typically this may be a modest multiple of n. If, on entry, MXS is zero or negative, the value returned by X02BBF is used.

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

Monit can be used to print out the current values of any selection of its parameters. The frequency with which MONIT is called in E02GBF is controlled by IPRINT (see below).

Its specification is:

```
SUBROUTINE MONIT(N, X, NITER, K, EL1N)
INTEGER    N, NITER, K
real       X(N), EL1N
```

1: N – INTEGER. *Input*

On entry: the number n of unknowns (the number of columns of the matrix A).

2: X(N) – ***real*** array. *Input*

On entry: the latest estimate of the unknowns.

3: NITER – INTEGER. *Input*

On entry: the number of iterations so far carried out.

4: K – INTEGER. *Input*

On entry: the total number of equations and constraints which are currently active (i.e. the number of equations with zero residuals plus the number of constraints which are satisfied as equations).

5: EL1N – ***real***. *Input*

On entry: the l_1 norm of the current residuals of the over-determined system of equations.

MONIT must be declared as EXTERNAL in the (sub)program from which E02GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: IPRINT – INTEGER. *Input*

On entry: the frequency of iteration print out. If IPRINT > 0, then MONIT is called every IPRINT iterations and at the solution. If IPRINT = 0, then information is printed out at the solution only. Otherwise MONIT is not called (but a dummy routine must still be provided).

11: K – INTEGER. *Output*

On exit: the total number of equations and constraints which are then active (i.e. the number of equations with zero residuals plus the number of constraints which are satisfied as equalities).

12: EL1N – ***real***. *Output*

On exit: the l_1 norm (sum of absolute values) of the equation residuals.

13: INDX(MPL) – INTEGER array. *Output*

On exit: specifies which columns of E relate to the inactive equations and constraints. INDX(1) up to INDX(K) number the active columns and INDX(K+1) up to INDX(MPL) number the inactive columns.

14: W(IW) – ***real*** array. *Workspace*

15: IW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E02GBF is called.

Constraint: IW $\geq 3\times$MPL + $5\times$N + N^2 + (N+1)$\times$(N+2)/2.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The constraints cannot all be satisfied simultaneously: they are not compatible with one another. Hence no solution is possible.

IFAIL = 2

The limit imposed by MXS has been reached without finding a solution. Consider restarting from the current point by simply calling E02GBF again without changing the parameters.

IFAIL = 3

The routine has failed because of numerical difficulties; the problem is too ill-conditioned. Consider rescaling the unknowns.

IFAIL = 4

On entry, one or more of the following conditions are violated:

$M \geq N \geq 2$,
MPL $\geq$ M,
IW $\geq 3\times$MPL + $5\times$N + N^2 + (N+1)$\times$(N+2)/2,
IE $\geq$ N.

Alternatively elements 1 to M of one of the first MPL columns of the array E are all zero – this corresponds to a zero row in either of the matrices A or C.

7. Accuracy

The method is stable.

8. Further Comments

The effect of m and n on the time and on the number of iterations varies from problem to problem, but typically the number of iterations is a small multiple of n and the total time taken by the routine is approximately proportional to mn^2.

Linear dependencies among the rows or columns of A and C are not necessarily a problem to the algorithm. Solutions can be obtained from rank-deficient A and C. However, the algorithm requires that at every step the currently active columns of E form a linearly independent set. If this is not the case at any step, small, random perturbations of the order of rounding error are added to the appropriate columns of E. Normally this perturbation process will not affect the solution significantly. It does mean, however, that results may not be exactly reproducible.

9. Example

Suppose we wish to approximate in [0,1] a set of data by a curve of the form

$$y = ax^3 + bx^2 + cx + d$$

which has non-negative slope at the data points. Given points (t_i,y_i) we may form the equations

$$y_i = at_i^3 + bt_i^2 + ct_i + d$$

for the 6 data points, $i = 1,2,...,6$. The requirement of a non-negative slope at the data points demands

$$3at_i^2 + 2bt_i + c \geq 0$$

for each t_i and these form the constraints.

(Note that, for fitting with polynomials, it would usually be advisable to work with the polynomial expressed in Chebyshev series form (see the Chapter Introduction). The power series form is used here for simplicity of exposition.)

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.

```
*     E02GBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MPLMAX, IE, IW
      PARAMETER        (N=4,MPLMAX=12,IE=N,IW=3*MPLMAX+N*N+5*N+(N+1)
     +                 *(N+2)/2)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             EL1N, T, XI
      INTEGER          I, IFAIL, IPRINT, K, L, M, MXS
*     .. Local Arrays ..
      real             E(IE,MPLMAX), F(MPLMAX), W(IW), X(N)
      INTEGER          IWORK(MPLMAX)
*     .. External Subroutines ..
      EXTERNAL         E02GBF, MONIT
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02GBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M
      L = M
```

```
      IF (M.GT.0 .AND. M+L.LE.MPLMAX) THEN
         DO 20 I = 1, M
            READ (NIN,*) T, F(I)
            XI = 0.1e0*real(I-1)
            E(1,I) = 1.0e0
            E(2,I) = T
            E(3,I) = T*T
            E(4,I) = T*T*T
            E(1,M+I) = 0.0e0
            E(2,M+I) = 1.0e0
            E(3,M+I) = 2.0e0*T
            E(4,M+I) = 3.0e0*T*T
            F(M+I) = 0.0e0
   20    CONTINUE
         DO 40 I = 1, N
            X(I) = 0.0e0
   40    CONTINUE
         MXS = 50
*        * Set IPRINT=1 to obtain output from MONIT at each iteration *
         IPRINT = 0
         IFAIL = 1
*
         CALL E02GBF(M,N,M+L,E,IE,F,X,MXS,MONIT,IPRINT,K,EL1N,IWORK,W,
     +               IW,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
      END IF
      STOP
*
99999 FORMAT (1X,A,I2)
      END
*
      SUBROUTINE MONIT(N,X,NITER,K,EL1N)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             EL1N
      INTEGER          K, N, NITER
*     .. Array Arguments ..
      real             X(N)
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Results at iteration ', NITER
      WRITE (NOUT,*) 'X-values'
      WRITE (NOUT,99998) X
      WRITE (NOUT,99997) 'Norm of residuals =', EL1N
      RETURN
*
99999 FORMAT (1X,A,I5)
99998 FORMAT (1X,4F15.4)
99997 FORMAT (1X,A,e12.5)
      END
```

9.2. Program Data

```
E02GBF Example Program Data
  6
  0.00  0.00
  0.20  0.07
  0.40  0.07
  0.60  0.11
  0.80  0.27
  1.00  0.68
```

9.3. Program Results

```
E02GBF Example Program Results

Results at iteration    10
X-values
         0.0000          0.6943         -2.1482          2.1339
Norm of residuals = 0.95714E-02

IFAIL =  0
```

With IPRINT set to 1 in the example program, results similar to those below are obtained from MONIT:

```
E02GBF Example Program Results

Results at iteration     0
X-values
         0.0000          0.0000          0.0000          0.0000
Norm of residuals = 0.12000E+01

Results at iteration     1
X-values
         0.0700          0.0000          0.0000          0.0000
Norm of residuals = 0.92000E+00

Results at iteration     2
X-values
         0.1100          0.0000          0.0000          0.0000
Norm of residuals = 0.92000E+00

Results at iteration     3
X-values
         0.0700          0.0000          0.0000          0.0000
Norm of residuals = 0.92000E+00

Results at iteration     4
X-values
         0.0700          0.0000          0.0000          0.0000
Norm of residuals = 0.92000E+00

Results at iteration     5
X-values
         0.0735          0.0000         -0.2599          0.8665
Norm of residuals = 0.22858E+00

Results at iteration     6
X-values
         0.0060          0.0000         -0.2889          0.9628
Norm of residuals = 0.16655E+00

Results at iteration     7
X-values
         0.0378          0.3922         -1.5078          1.7578
Norm of residuals = 0.58438E-01

Results at iteration     8
X-values
         0.0319          0.4575         -1.7156          1.9063
Norm of residuals = 0.54875E-01

Results at iteration     9
X-values
         0.0000          0.6943         -2.1482          2.1339
Norm of residuals = 0.95714E-02
```

```
Results at iteration    10
X-values
        0.0000          0.6943          -2.1482          2.1339
Norm of residuals = 0.95714E-02

IFAIL =  0
```

E02GCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02GCF calculates an l_∞ solution to an over-determined system of linear equations.

2. Specification

```
      SUBROUTINE E02GCF (M, N, MDIM, NDIM, A, B, TOL, RELERR, X, RESMAX,
     1                   IRANK, ITER, IFAIL)
      INTEGER          M, N, MDIM, NDIM, IRANK, ITER, IFAIL
      real             A(NDIM,MDIM), B(M), TOL, RELERR, X(N), RESMAX
```

3. Description

Given a matrix A with m rows and n columns ($m \geq n$) and a vector b with m elements, the routine calculates an l_∞ solution to the over-determined system of equations

$$Ax = b.$$

That is to say, it calculates a vector x, with n elements, which minimizes the l_∞ norm of the residuals (the absolutely largest residual)

$$r(x) = \max_{1 \leq i \leq m} |r_i|$$

where the residuals r_i are given by

$$r_i = b_i - \sum_{j=1}^{n} a_{ij}x_j, \qquad i = 1,2,...,m.$$

Here a_{ij} is the element in row i and column j of A, b_i is the ith element of b and x_j the jth element of x. The matrix A need not be of full rank. The solution is not unique in this case, and may not be unique even if A is of full rank.

Alternatively, in applications where a complete minimization of the l_∞ norm is not necessary, the user may obtain an approximate solution, usually in shorter time, by giving an appropriate value to the parameter RELERR.

Typically in applications to data fitting, data consisting of m points with co-ordinates (t_i,y_i) is to be approximated in the l_∞ norm by a linear combination of known functions $\phi_j(t)$,

$$\alpha_1\phi_1(t) + \alpha_2\phi_2(t) + ... + \alpha_n\phi_n(t).$$

This is equivalent to finding an l_∞ solution to the over-determined system of equations

$$\sum_{j=1}^{n} \phi_j(t_i)\alpha_j = y_i, \qquad i = 1,2,...,m.$$

Thus if, for each value of i and j the element a_{ij} of the matrix A above is set equal to the value of $\phi_j(t_i)$ and b_i is set equal to y_i, the solution vector x will contain the required values of the α_j. Note that the independent variable t above can, instead, be a vector of several independent variables (this includes the case where each ϕ_i is a function of a different variable, or set of variables).

The algorithm is a modification of the simplex method of linear programming applied to the dual formation of the l_∞ problem (see Barrodale and Phillips [1] and [2]). The modifications are designed to improve the efficiency and stability of the simplex method for this particular application.

4. References

[1] BARRODALE, I. and PHILLIPS, C.
An Improved Algorithm for Discrete Chebyshev Linear Approximation.
Proc. 4th Manitoba Conf. on Numerical Mathematics, U. of Manitoba, Winnipeg, Canada, pp. 177-190, 1974.

[2] BARRODALE, I. and PHILLIPS, C.
Solution of an overdetermined system of linear equations in the Chebyshev Norm [F4] (Algorithm 495).
ACM Trans. Math. Software, 1, 3, pp. 264-270, 1975.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of equations, m (the number of rows of the matrix A).

Constraint: M ≥ N.

2: N – INTEGER. *Input*

On entry: the number of unknowns, n (the number of columns of the matrix A).

Constraint: N ≥ 1.

3: MDIM – INTEGER. *Input*

On entry: the second dimension of the array A as declared in the (sub)program from which E02GCF is called.

Constraint: MDIM ≥ M + 1.

4: NDIM – INTEGER. *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which E02GCF is called.

Constraint: NDIM ≥ N + 3.

5: A(NDIM,MDIM) – ***real*** array. *Input/Output*

On entry: A(j,i) must contain a_{ij}, element in the ith row and jth column of the matrix A for, $i = 1,2,...,m$; $j = 1,2,...,n$ (that is, the **transpose** of the matrix). The remaining elements need not be set. Preferably, the columns of the matrix A (rows of the parameter A) should be scaled before entry: see Section 7.

On exit: A contains the last simplex tableau.

6: B(M) – ***real*** array. *Input/Output*

On entry: b_i, the ith element of the vector b, for $i = 1,2,...,m$.

On exit: the ith residual r_i corresponding to the solution vector x, for $i = 1,2,...,m$. Note however that these residuals may contain few significant figures, especially when RESMAX is within one or two orders of magnitude of TOL. Indeed if RESMAX ≤ TOL, the elements B(i) may all be set to zero. It is therefore often advisable to compute the residuals directly.

7: TOL – ***real***. *Input*

On entry: a threshold below which numbers are regarded as zero. The recommended threshold value is $10.0 \times \varepsilon$, where ε is the ***machine precision***. If TOL ≤ 0.0 on entry, the recommended value is used within the routine. If premature termination occurs, a larger value for TOL may result in a valid solution.

Suggested value: 0.0.

8: RELERR – ***real.*** *Input/Output*

On entry: RELERR must be set to a bound on the relative error acceptable in the maximum residual at the solution.

If RELERR $\leq$ 0.0, then the l_∞ solution is computed, and RELERR is set to 0.0 on exit.

If RELERR > 0.0, then the routine obtains instead an approximate solution for which the largest residual is less than 1.0 + RELERR times that of the l_∞ solution; on exit, RELERR contains a smaller value such that the above bound still applies. (The usual result of this option, say with RELERR = 0.1, is a saving in the number of simplex iterations).

On exit: RELERR is altered as described above.

9: X(N) – ***real*** array. *Output*

On exit: if IFAIL = 0 or 1, X(j) contains the jth element of the solution vector x, for j = 1,2,...,n. Whether this is an l_∞ solution or an approximation to one, depends on the value of RELERR on entry.

10: RESMAX – ***real.*** *Output*

On exit: if IFAIL = 0 or 1, RESMAX contains the absolute value of the largest residual(s) for the solution vector x. (See B above.)

11: IRANK – INTEGER. *Output*

On exit: if IFAIL = 0 or 1, IRANK contains the computed rank of the matrix A.

12: ITER – INTEGER. *Output*

On exit: if IFAIL = 0 or 1, ITER contains the number of iterations taken by the simplex method.

13: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An optimal solution has been obtained but this may not be unique (perhaps simply because the matrix A is not of full rank, i.e. IRANK < N).

IFAIL = 2

The calculations have terminated prematurely due to rounding errors. Experiment with larger values of TOL or try rescaling the columns of the matrix (see Section 8).

IFAIL = 3

On entry, NDIM < N + 3
or MDIM < M + 1
or M < N
or N < 1.

7. Accuracy

Experience suggests that the computational accuracy of the solution x is comparable with the accuracy that could be obtained by applying Gaussian elimination with partial pivoting to the $n + 1$ equations which have residuals of largest absolute value. The accuracy therefore varies with the conditioning of the problem, but has been found generally very satisfactory in practice.

8. Further Comments

The effects of m and n on the time and on the number of iterations in the simplex method vary from problem to problem, but typically the number of iterations is a small multiple of n and the total time is approximately proportional to mn^2.

It is recommended that, before the routine is entered, the columns of the matrix A are scaled so that the largest element in each column is of the order of unity. This should improve the conditioning of the matrix, and also enable the parameter TOL to perform its correct function. The solution x obtained will then, of course, relate to the scaled form of the matrix. Thus if the scaling is such that, for each $j = 1,2,...,n$, the elements of the jth column are multiplied by the constant k_j, the element x_j of the solution vector x must be multiplied by k_j if it is desired to recover the solution corresponding to the original matrix A.

9. Example

Suppose we wish to approximate a set of data by a curve of the form

$$y = Ke^t + Le^{-t} + M$$

where K, L and M are unknown. Given values y_i at 5 points t_i we may form the over-determined set of equations for K, L and M

$$e^{t_i}K + e^{-t_i}L + M = y_i , \qquad i = 1,2,...,5.$$

E02GCF is used to solve these in the l_∞ sense.

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.

```
*     E02GCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MMAX, NDIM, MDIM
      PARAMETER        (N=3,MMAX=5,NDIM=N+3,MDIM=MMAX+1)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             RELERR, RESMAX, T, TOL
      INTEGER          I, IFAIL, IRANK, ITER, M
*     .. Local Arrays ..
      real             A(NDIM,MDIM), B(MMAX), X(N)
*     .. External Subroutines ..
      EXTERNAL         E02GCF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02GCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         DO 20 I = 1, M
            READ (NIN,*) T, B(I)
            A(1,I) = EXP(T)
            A(2,I) = EXP(-T)
            A(3,I) = 1.0e0
   20    CONTINUE
```

```
         TOL = 0.0e0
         RELERR = 0.0e0
         IFAIL = 1
*
         CALL E02GCF(M,N,MDIM,NDIM,A,B,TOL,RELERR,X,RESMAX,IRANK,ITER,
     +               IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.LE.1) THEN
            WRITE (NOUT,99999) 'RESMAX = ', RESMAX, '  Rank = ', IRANK,
     +        '  Iterations = ', ITER, '  IFAIL =', IFAIL
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Solution'
            WRITE (NOUT,99998) (X(I),I=1,N)
         ELSE
            WRITE (NOUT,99997) 'E02GCF fails with error', IFAIL
         END IF
       END IF
       STOP
*
99999 FORMAT (1X,A,e10.2,A,I5,A,I5,A,I5)
99998 FORMAT (1X,6F10.4)
99997 FORMAT (1X,A,I2)
       END
```

9.2. Program Data

```
E02GCF Example Program Data
  5
   0.0 4.501
   0.2 4.360
   0.4 4.333
   0.6 4.418
   0.8 4.625
```

9.3. Program Results

```
 E02GCF Example Program Results

 RESMAX =   0.10E-02  Rank =     3  Iterations =     4  IFAIL =    0

 Solution
     1.0049    2.0149    1.4822
```

E02RAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02RAF calculates the coefficients in a Padé approximant to a function from its user-supplied Maclaurin expansion.

2. Specification

```
SUBROUTINE E02RAF (IA, IB, C, IC, A, B, W, JW, IFAIL)
INTEGER          IA, IB, IC, JW, IFAIL
real             C(IC), A(IA), B(IB), W(JW)
```

3. Description

Given a power series

$$c_0 + c_1x + c_2x^2 + \ldots + c_{l+m}x^{l+m} + \ldots$$

this routine uses the coefficients c_i, for $i = 0,1,\ldots,l+m$, to form the $[l/m]$ Padé approximant of the form

$$\frac{a_0 + a_1x + a_2x^2 + \ldots + a_lx^l}{b_0 + b_1x + b_2x^2 + \ldots + b_mx^m}$$

with b_0 defined to be unity. The two sets of coefficients a_j, for $j = 0,1,\ldots,l$ and b_k, for $k = 0,1,\ldots,m$ in the numerator and denominator are calculated by direct solution of the Padé equations (see Graves-Morris [2]); these values are returned through the argument list unless the approximant is degenerate.

Padé approximation is a useful technique when values of a function are to be obtained from its Maclaurin expansion but convergence of the series is unacceptably slow or even non-existent. It is based on the hypothesis of the existence of a sequence of convergent rational approximations, as described in Baker and Graves-Morris [1] and [2].

Unless there are reasons to the contrary (as discussed in [1] Chapter 4, Section 2, Chapters 5 and 6), one normally uses the diagonal sequence of Padé approximants, namely

$$\{[m/m],\ m = 0,1,2,\ldots\}.$$

Subsequent evaluation of the approximant at a given value of x may be carried out using E02RBF.

4. References

[1] BAKER, G.A. Jr. and GRAVES-MORRIS, P.R.
Padé Approximants, Part 1: Basic Theory.
In: 'Encyclopaedia of Mathematics and its Applications'.
Addison-Wesley, 1981.

[2] GRAVES-MORRIS, P.R.
The Numerical Calculation of Padé Approximants.
In: 'Padé Approximation and its Applications', L. Wuytack, (Ed.).
Lecture Notes in Mathematics, 765, pp. 231-245, 1979.

5. Parameters

1: IA – INTEGER. *Input*

2: IB – INTEGER. *Input*

On entry: IA must specify $l + 1$ and IB must specify $m + 1$, where l and m are the degrees of the numerator and denominator of the approximant, respectively.

Constraint: IA and IB ≥ 1.

3: C(IC) – ***real*** array. *Input*

On entry: C(i) must specify, for $i = 1,2,...,l+m+1$, the coefficient of x^{i-1} in the given power series.

4: IC – INTEGER. *Input*

On entry: the dimension of the array C as declared in the (sub)program from which E02RAF is called.

Constraint: IC $\geq$ IA + IB – 1.

5: A(IA) – ***real*** array. *Output*

On exit: A(j+1), for $j = 1,2,...,l+1$, contains the coefficient a_j in the numerator of the approximant.

6: B(IB) – ***real*** array. *Output*

On exit: B(k+1), for $k = 1,2,...,m+1$, contains the coefficient b_k in the denominator of the approximant.

7: W(JW) – ***real*** array. *Workspace*

8: JW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E02RAF is called.

Constraint: JW $\geq$ IB$\times$(2$\times$IB+3).

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, JW < IB$\times$(2$\times$IB+3),
or IA < 1,
or IB < 1,
or IC < IA + IB – 1
(so there are insufficient coefficients in the given power series to calculate the desired approximant).

IFAIL = 2

The Padé approximant is degenerate.

7. Accuracy

The solution should be the best possible to the extent to which the solution is determined by the input coefficients. It is recommended that the user determines the locations of the zeros of the numerator and denominator polynomials, both to examine compatibility with the analytic structure of the given function and to detect defects. (Defects are nearby pole-zero pairs; defects close to $x = 0.0$ characterise ill-conditioning in the construction of the approximant.) Defects occur in regions where the approximation is necessarily inaccurate. The example program calls C02AGF to determine the above zeros.

It is easy to test the stability of the computed numerator and denominator coefficients by making small perturbations of the original Maclaurin series coefficients (e.g. c_l or c_{l+m}). These questions of intrinsic error of the approximants and computational error in their calculation are discussed in Baker and Graves-Morris [1] Chapter 2.

8. Further Comments

The time taken by the routine is approximately proportional to m^3.

9. Example

The example program calculates the [4/4] Padé approximant of e^x (whose power-series coefficients are first stored in the array CC). The poles and zeros are then calculated to check the character of the [4/4] Padé approximant.

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.

```
*     E02RAF Example Program Text.
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          L, M, IA, IB, IC, IW
      PARAMETER        (L=4,M=4,IA=L+1,IB=M+1,IC=IA+IB-1,IW=IB*(2*IB+3))
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      LOGICAL          SCALE
      PARAMETER        (SCALE=.TRUE.)
*     .. Local Scalars ..
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             AA(IA), BB(IB), CC(IC), DD(IA+IB), W(IW),
     +                 WORK(2*(L+M+1)), Z(2,L+M)
*     .. External Subroutines ..
      EXTERNAL         C02AGF, E02RAF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02RAF Example Program Results'
*     Power series coefficients in CC
      CC(1) = 1.0e0
      DO 20 I = 1, IC - 1
         CC(I+1) = CC(I)/real(I)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The given series coefficients are'
      WRITE (NOUT,99999) (CC(I),I=1,IC)
      IFAIL = 0
*
      CALL E02RAF(IA,IB,CC,IC,AA,BB,W,IW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Numerator coefficients'
      WRITE (NOUT,99999) (AA(I),I=1,IA)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Denominator coefficients'
      WRITE (NOUT,99999) (BB(I),I=1,IB)
*     Calculate zeros of the approximant using C02AGF
```

```
*     First need to reverse order of coefficients
      DO 40 I = 1, IA
         DD(IA-I+1) = AA(I)
   40 CONTINUE
      IFAIL = 0
*
      CALL C02AGF(DD,L,SCALE,Z,WORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Zeros of approximant are at'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     Real part     Imag part'
      WRITE (NOUT,99998) (Z(1,I),Z(2,I),I=1,L)
*     Calculate poles of the approximant using C02AGF
*     Reverse order of coefficients
      DO 60 I = 1, IB
         DD(IB-I+1) = BB(I)
   60 CONTINUE
      IFAIL = 0
*
      CALL C02AGF(DD,M,SCALE,Z,WORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Poles of approximant are at'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     Real part     Imag part'
      WRITE (NOUT,99998) (Z(1,I),Z(2,I),I=1,M)
      STOP
*
99999 FORMAT (1X,5e13.4)
99998 FORMAT (1X,2e13.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E02RAF Example Program Results

The given series coefficients are
   0.1000E+01   0.1000E+01   0.5000E+00   0.1667E+00   0.4167E-01
   0.8333E-02   0.1389E-02   0.1984E-03   0.2480E-04

Numerator coefficients
   0.1000E+01   0.5000E+00   0.1071E+00   0.1190E-01   0.5952E-03

Denominator coefficients
   0.1000E+01  -0.5000E+00   0.1071E+00  -0.1190E-01   0.5952E-03

Zeros of approximant are at

    Real part     Imag part
  -0.5792E+01   0.1734E+01
  -0.5792E+01  -0.1734E+01
  -0.4208E+01   0.5315E+01
  -0.4208E+01  -0.5315E+01

Poles of approximant are at

    Real part     Imag part
   0.5792E+01   0.1734E+01
   0.5792E+01  -0.1734E+01
   0.4208E+01   0.5315E+01
   0.4208E+01  -0.5315E+01
```

E02RBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E02RBF evaluates a rational function at a user-supplied point, given the numerator and denominator coefficients.

2. Specification

```
SUBROUTINE E02RBF (A, IA, B, IB, X, ANS, IFAIL)
INTEGER          IA, IB, IFAIL
real             A(IA), B(IB), X, ANS
```

3. Description

Given a real value x and the coefficients a_j, for $j = 0,...,l$ and b_k, for $k = 0,...,m$, E02RBF evaluates the rational function

$$\frac{\sum_{j=0}^{l} a_j \, x^j}{\sum_{k=0}^{m} b_k \, x^k},$$

using nested multiplication (Conte and de Boor [1]).

A particular use of E02RBF is to compute values of the Padé approximants determined by E02RAF.

4. References

[1] CONTE, S.D. and DE BOOR, C.
Elementary Numerical Analysis.
McGraw-Hill, p. 67, 1965.

[2] PETERS, G. and WILKINSON, J.H.
Practical problems arising in the solution of polynomial equations.
J. Inst. Math. Appl., 8, pp. 16-35, 1971.

5. Parameters

1: A(IA) – ***real*** array. *Input*

On entry: A(j+1), for $j = 1,2,...,l+1$, must contain the value of the coefficient a_j in the numerator of the rational function.

2: IA – INTEGER. *Input*

On entry: the value of $l + 1$, where l is the degree of the numerator.

Constraint: IA $\geq$ 1.

3: B(IB) – ***real*** array. *Input*

On entry: B(k+1), for $k = 1,2,...,m+1$, must contain the value of the coefficient b_k in the denominator of the rational function.

Constraint: if IB = 1, B(1) must be non-zero.

4: IB – INTEGER. *Input*

On entry: the value of $m + 1$, where m is the degree of the denominator.

Constraint: IB $\geq$ 1.

5: X – ***real.*** *Input*

On entry: the point x at which the rational function is to be evaluated.

6: ANS – ***real.*** *Output*

On exit: the result of evaluating the rational function at the given point x.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The rational function is being evaluated at or near a pole.

IFAIL = 2

On entry, IA < 1,
or IB < 1,
or B(1) = 0.0 when IB = 1 (so the denominator is identically zero).

7. Accuracy

A running error analysis for polynomial evaluation by nested multiplication using the recurrence suggested by Kahan (see Peters and Wilkinson [2]) is used to detect whether the user is attempting to evaluate the approximant at or near a pole.

8. Further Comments

The time taken by the routine is approximately proportional to $l + m$.

9. Example

The example program first calls E02RAF to calculate the 4/4 Padé approximant to e^x, and then uses E02RBF to evaluate the approximant at $x = 0.1,0.2,...,1.0$.

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.

```
*     E02RBF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          L, M, IA, IB, IC, IW
      PARAMETER        (L=4,M=4,IA=L+1,IB=M+1,IC=IA+IB-1,IW=IB*(2*IB+3))
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ANS, TVAL, X
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             AA(IA), BB(IB), CC(IC), W(IW)
*     .. External Subroutines ..
      EXTERNAL         E02RAF, E02RBF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP, real
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02RBF Example Program Results'
      CC(1) = 1.0e0
      DO 20 I = 1, IC - 1
         CC(I+1) = CC(I)/real(I)
   20 CONTINUE
      IFAIL = 0
*
      CALL E02RAF(IA,IB,CC,IC,AA,BB,W,IW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     X         Pade            True'
      DO 40 I = 1, 10
         X = real(I)/10.0e0
         IFAIL = 0
*
         CALL E02RBF(AA,IA,BB,IB,X,ANS,IFAIL)
*
         TVAL = EXP(X)
         WRITE (NOUT,99999) X, ANS, TVAL
   40 CONTINUE
      STOP
*
99999 FORMAT (1X,F6.1,3e15.5)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E02RBF Example Program Results

      X         Pade            True
    0.1    0.11052E+01    0.11052E+01
    0.2    0.12214E+01    0.12214E+01
    0.3    0.13499E+01    0.13499E+01
    0.4    0.14918E+01    0.14918E+01
    0.5    0.16487E+01    0.16487E+01
    0.6    0.18221E+01    0.18221E+01
    0.7    0.20138E+01    0.20138E+01
    0.8    0.22255E+01    0.22255E+01
    0.9    0.24596E+01    0.24596E+01
    1.0    0.27183E+01    0.27183E+01
```

E02ZAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

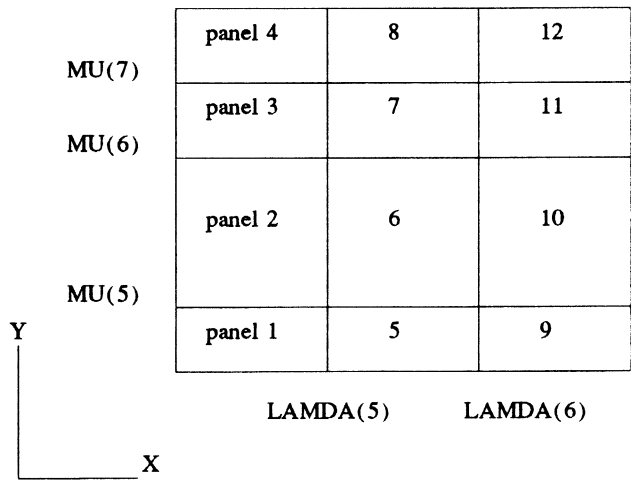
E02ZAF sorts two-dimensional data into rectangular panels.

2. Specification

```
      SUBROUTINE E02ZAF (PX, PY, LAMDA, MU, M, X, Y, POINT, NPOINT, ADRES,
     1                   NADRES, IFAIL)
      INTEGER          PX, PY, M, POINT(NPOINT), NPOINT, ADRES(NADRES),
     1                 NADRES, IFAIL
      real             LAMDA(PX), MU(PY), X(M), Y(M)
```

3. Description

A set of m data points with rectangular Cartesian co-ordinates x_r, y_r are sorted into panels defined by lines parallel to the y and x axes. The intercepts of these lines on the x and y axes are given in LAMDA(i), for i = 5,6,...,PX−4 and MU(j), for j = 5,6,...,PY−4, respectively. The subroutine orders the data so that all points in a panel occur before data in succeeding panels, where the panels are numbered from bottom to top and then left to right, with the usual arrangement of axes, as shown in the diagram. Within a panel the points maintain their original order.



A data point lying exactly on one or more panel sides is taken to be in the highest-numbered panel adjacent to the point. The subroutine does not physically rearrange the data, but provides the array POINT which contains a linked list for each panel, pointing to the data in that panel. The total number of panels is (PX−7)×(PY−7).

4. References

None.

5. Parameters

1: PX – INTEGER. *Input*

2: PY – INTEGER. *Input*

On entry: PX and PY must specify eight more than the number of intercepts on the x axis and y axis, respectively.

Constraint: PX ≥ 8 and PY ≥ 8.

3: LAMDA(PX) – ***real*** array. *Input*

On entry: LAMDA(5) to LAMDA(PX–4) must contain, in non-decreasing order, the intercepts on the x axis of the sides of the panels parallel to the y axis.

4: MU(PY) – ***real*** array. *Input*

On entry: MU(5) to MU(PY–4) must contain, in non-decreasing order, the intercepts on the y axis of the sides of the panels parallel to the x axis.

5: M – INTEGER. *Input*

On entry: the number m of data points.

6: X(M) – ***real*** array. *Input*

7: Y(M) – ***real*** array. *Input*

On entry: the co-ordinates of the rth data point (x_r,y_r), for $r = 1,2,...,m$.

8: POINT(NPOINT) – INTEGER array. *Output*

On exit: for $i = 1,2,...,$NADRES, POINT$(m+i)$ = I1 is the index of the first point in panel i, POINT(I1) = I2 is the index of the second point in panel i and so on.

POINT(IN) = 0 indicates that X(IN),Y(IN) was the last point in the panel.

The co-ordinates of points in panel i can be accessed in turn by means of the following instructions:

```
      IN = M + I
10    IN = POINT(IN)
      IF (IN .EQ. 0) GOTO 20
        XI = X(IN)
        YI = Y(IN)
           .
           .
           .
      GOTO 10
20    ...
```

9: NPOINT – INTEGER. *Input*

On entry: the dimension of the array POINT as declared in the (sub)program from which E02ZAF is called.

Constraint: NPOINT $\geq$ M + (PX–7)$\times$(PY–7).

10: ADRES(NADRES) – INTEGER array. *Workspace*

11: NADRES – INTEGER. *Input*

On entry: the value (PX–7)$\times$(PY–7), the number of panels into which the (x,y) plane is divided.

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The intercepts in the array LAMDA, or in the array MU, are not in non-decreasing order.

IFAIL = 2

On entry, PX < 8,
or PY < 8,
or M ≤ 0,
or NADRES ≠ (PX−7)×(PY−7),
or NPOINT < M + (PX−7)×(PY−7).

7. Accuracy

Not applicable.

8. Further Comments

The time taken by this routine is approximately proportional to $m \times \log(\text{NADRES})$.

This subroutine was written to sort two dimensional data in the manner required by routine E02DAF. The first 9 parameters of E02ZAF are the same as the parameters in E02DAF which have the same name.

9. Example

This example program reads in data points and the intercepts of the panel sides on the x and y axes; it calls E02ZAF to set up the index array POINT; and finally it prints the data points in panel order.

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.

```
*     E02ZAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, MAXPX, MAXPY, NADMAX, NPOINT
      PARAMETER        (MMAX=20,MAXPX=12,MAXPY=12,NADMAX=(MAXPX-7)
     +                 *(MAXPY-7),NPOINT=MMAX+NADMAX)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IADRES, IFAIL, M, NADRES, PX, PY
*     .. Local Arrays ..
      real             LAMDA(MAXPX), MU(MAXPY), X(MMAX), Y(MMAX)
      INTEGER          ADRES(NADMAX), POINT(NPOINT)
*     .. External Subroutines ..
      EXTERNAL         E02ZAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E02ZAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*) M
      IF (M.GT.0 .AND. M.LE.MMAX) THEN
         READ (NIN,*) PX, PY
         IF (PX.LE.MAXPX .AND. PY.LE.MAXPY) THEN
            NADRES = (PX-7)*(PY-7)
*           Read data points and intercepts of panel sides
            READ (NIN,*) (X(I),Y(I),I=1,M)
            IF (PX.GT.8) READ (NIN,*) (LAMDA(I),I=5,PX-4)
            IF (PY.GT.8) READ (NIN,*) (MU(I),I=5,PY-4)
*           Sort points into panel order
            IFAIL = 0
*
            CALL E02ZAF(PX,PY,LAMDA,MU,M,X,Y,POINT,NPOINT,ADRES,NADRES,
     +                  IFAIL)
*
```

```
*           Output points in panel order
            DO 60 I = 1, NADRES
               WRITE (NOUT,*)
               WRITE (NOUT,99999) 'Panel', I
               IADRES = M + I
   40          IADRES = POINT(IADRES)
               IF (IADRES.GT.0) THEN
                  WRITE (NOUT,99998) X(IADRES), Y(IADRES)
                  GO TO 40
               END IF
   60       CONTINUE
            GO TO 20
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,2F7.2)
      END
```

9.2. Program Data

```
E02ZAF Example Program Data
  10
   9
  10
0       0.77
0.70    1.06
1.44    0.33
0.21    0.44
1.01    0.50
1.84    0.02
0.71    1.95
1.00    1.20
0.54    0.04
1.53    0.18
1.00
0.80
1.20
0
```

9.3. Program Results

```
 E02ZAF Example Program Results

 Panel   1
    0.00   0.77
    0.21   0.44
    0.54   0.04

 Panel   2
    0.70   1.06

 Panel   3
    0.71   1.95

 Panel   4
    1.44   0.33
    1.01   0.50
    1.84   0.02
    1.53   0.18

 Panel   5

 Panel   6
    1.00   1.20
```

Chapter E04 – Minimizing or Maximizing a Function

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E04ABF	6	Minimum, function of one variable using function values only
E04BBF	6	Minimum, function of one variable, using first derivative
E04CCF	1	Unconstrained minimum, simplex algorithm, function of several variables using function values only (comprehensive)
E04DGF	12	Unconstrained minimum, preconditioned conjugate gradient algorithm, function of several variables using first derivatives (comprehensive)
E04DJF	12	Read optional parameter values for E04DGF from external file
E04DKF	12	Supply optional parameter values to E04DGF
E04FCF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (comprehensive)
E04FYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (easy-to-use)
E04GBF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm using first derivatives (comprehensive)
E04GDF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (comprehensive)
E04GYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm, using first derivatives (easy-to-use)
E04GZF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (easy-to-use)
E04HCF	6	Check user's routine for calculating first derivatives of function
E04HDF	6	Check user's routine for calculating second derivatives of function
E04HEF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (comprehensive)
E04HYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (easy-to-use)
E04JYF	18	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)
E04KDF	6	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (comprehensive)
E04KYF	18	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04KZF	18	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04LBF	6	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (comprehensive)
E04LYF	18	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (easy-to-use)
E04MFF	16	LP problem (dense)
E04MGF	16	Read optional parameter values for E04MFF from external file
E04MHF	16	Supply optional parameter values to E04MFF
E04MZF	18	Converts MPSX data file defining LP or QP problem to format required by E04NKF
E04NCF	12	Convex QP problem or linearly-constrained linear least-squares problem (dense)
E04NDF	12	Read optional parameter values for E04NCF from external file
E04NEF	12	Supply optional parameter values to E04NCF
E04NFF	16	QP problem (dense)

E04NGF	16	Read optional parameter values for E04NFF from external file
E04NHF	16	Supply optional parameter values to E04NFF
E04NKF	18	LP or QP problem (sparse)
E04NLF	18	Read optional parameter values for E04NKF from external file
E04NMF	18	Supply optional parameter values to E04NKF
E04UCF	12	Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
E04UDF	12	Read optional parameter values for E04UCF or E04UFF from external file
E04UEF	12	Supply optional parameter values to E04UCF or E04UFF
E04UFF	18	Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)
E04UGF	19	NLP problem (sparse)
E04UHF	19	Read optional parameter values for E04UGF from external file
E04UJF	19	Supply optional parameter values to E04UGF
E04UNF	17	Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)
E04UQF	14	Read optional parameter values for E04UNF from external file
E04URF	14	Supply optional parameter values to E04UNF
E04XAF	12	Estimate (using numerical differentiation) gradient and/or Hessian of a function
E04YAF	7	Check user's routine for calculating Jacobian of first derivatives
E04YBF	7	Check user's routine for calculating Hessian of a sum of squares
E04YCF	11	Covariance matrix for nonlinear least-squares problem (unconstrained)
E04ZCF	11	Check user's routines for calculating first derivatives of function and constraints

Chapter E04

Minimizing or Maximizing a Function

Contents

1 Scope of the Chapter

An optimization problem involves minimizing a function (called the **objective function**) of several variables, possibly subject to restrictions on the values of the variables defined by a set of **constraint functions**. The routines in the Library are concerned with function **minimization** only, since the problem of maximizing a given objective function $F(x)$ is equivalent to minimizing $-F(x)$.

This introduction is only a brief guide to the subject of optimization designed for the casual user. Anyone with a difficult or protracted problem to solve will find it beneficial to consult a more detailed text, such as Gill *et al.* [5] or Fletcher [3].

Users who are unfamiliar with the mathematics of the subject may find some sections difficult at first reading; if so, they should **concentrate** on Sections 2.1, 2.2, 2.5, 2.6 and 3.

2 Background to the Problems

2.1 Types of Optimization Problems

The solution of optimization problems by a single, all-purpose, method is cumbersome and inefficient. Optimization problems are therefore classified into particular categories, where each category is defined by the properties of the objective and constraint functions, as illustrated by some examples below.

Properties of Objective Function	**Properties of Constraints**
Nonlinear	Nonlinear
Sums of squares of nonlinear functions	Sparse linear
Quadratic	Linear
Sums of squares of linear functions	Bounds
Linear	None

For instance, a specific problem category involves the minimization of a nonlinear objective function subject to bounds on the variables. In the following sections we define the particular categories of problems that can be solved by routines contained in this chapter. Not every category is given special treatment in the current version of the Library; however, the long-term objective is to provide a comprehensive set of routines to solve problems in all such categories.

2.1.1 Unconstrained minimization

In unconstrained minimization problems there are no constraints on the variables. The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x)$$

where $x \in R^n$, that is, $x = (x_1, x_2, \ldots, x_n)^T$.

2.1.2 Nonlinear least-squares problems

Special consideration is given to the problem for which the function to be minimized can be expressed as a sum of squared functions. The least-squares problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}} \left\{ f^T f = \sum_{i=1}^{m} f_i^2(x) \right\}, x \in R^n$$

where the ith element of the m-vector f is the function $f_i(x)$.

2.1.3 Minimization subject to bounds on the variables

These problems differ from the unconstrained problem in that at least one of the variables is subject to a simple bound (or restriction) on its value, e.g., $x_5 \leq 10$, but no constraints of a more general form are present.

The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x), \quad x \in R^n$$

subject to $l_i \leq x_i \leq u_i$, $i = 1, 2, \ldots, n$.

This format assumes that upper and lower bounds exist on all the variables. By conceptually allowing $u_i = +\infty$ and $l_i = -\infty$ all the variables need not be restricted.

2.1.4 Minimization subject to linear constraints

A general linear constraint is defined as a constraint function that is linear in more than one of the variables, e.g. $3x_1 + 2x_2 \geq 4$. The various types of linear constraint are reflected in the following mathematical statement of the problem:

$$\underset{x}{\text{minimize}}\, F(x), \quad x \in R^n$$

subject to the

equality constraints:	$a_i^T x = b_i$	$i = 1, 2, \ldots, m_1$;
inequality constraints:	$a_i^T x \geq b_i$	$i = m_1 + 1, m_1 + 2, \ldots, m_2$;
	$a_i^T x \leq b_i$	$i = m_2 + 1, m_2 + 2, \ldots, m_3$;
range constraints:	$s_j \leq a_i^T x \leq t_j$	$i = m_3 + 1, m_3 + 2, \ldots, m_4$;
		$j = 1, 2, \ldots, m_4 - m_3$;
bounds constraints:	$l_i \leq x_i \leq u_i$	$i = 1, 2, \ldots, n$

where each a_i is a vector of length n; b_i, s_j and t_j are constant scalars; and any of the categories may be empty.

Although the bounds on x_i could be included in the definition of general linear constraints, we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a linear function, the linearly-constrained problem is termed a **linear programming** problem (LP problem); if $F(x)$ is a quadratic function, the problem is termed a **quadratic programming** problem (QP problem). For further discussion of LP and QP problems, including the dual formulation of such problems, see Dantzig [2].

2.1.5 Minimization subject to nonlinear constraints

A problem is included in this category if at least one constraint function is nonlinear, e.g. $x_1^2 + x_3 + x_4 - 2 \geq 0$. The mathematical statement of the problem is identical to that for the linearly-constrained case, except for the addition of the following constraints:

equality constraints:	$c_i(x) = 0$	$i = 1, 2, \ldots, m_5$;
inequality constraints:	$c_i(x) \geq 0$	$i = m_5 + 1, m_5 + 2, \ldots, m_6$;
range constraints:	$v_j \leq c_i(x) \leq w_j$	$i = m_6 + 1, m_6 + 2, \ldots, m_7$,
		$j = 1, 2, \ldots, m_7 - m_6$

where each c_i is a nonlinear function; v_j and w_j are constant scalars; and any category may be empty. Note that we do not include a separate category for constraints of the form $c_i(x) \leq 0$, since this is equivalent to $-c_i(x) \geq 0$.

Although the general linear constraints could be included in the definition of nonlinear constraints, again we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a nonlinear function, the nonlinearly-constrained problem is termed a **nonlinear programming** problem (NLP problem). For further discussion of NLP problems, see Gill *et al.* [5] or Fletcher [3].

2.1.6 Minimization subject to bounds on the objective function

In all of the above problem categories it is assumed that

$$a \leq F(x) \leq b$$

where $a = -\infty$ and $b = +\infty$. Problems in which a and/or b are finite can be solved by adding an extra constraint of the appropriate type (i.e., linear or nonlinear) depending on the form of $F(x)$. Further advice is given in Section 3.4.

2.2 Geometric Representation and Terminology

To illustrate the nature of optimization problems it is useful to consider the following example in two dimensions:

$$F(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1).$$

(This function is used as the example function in the documentation for the unconstrained routines.)

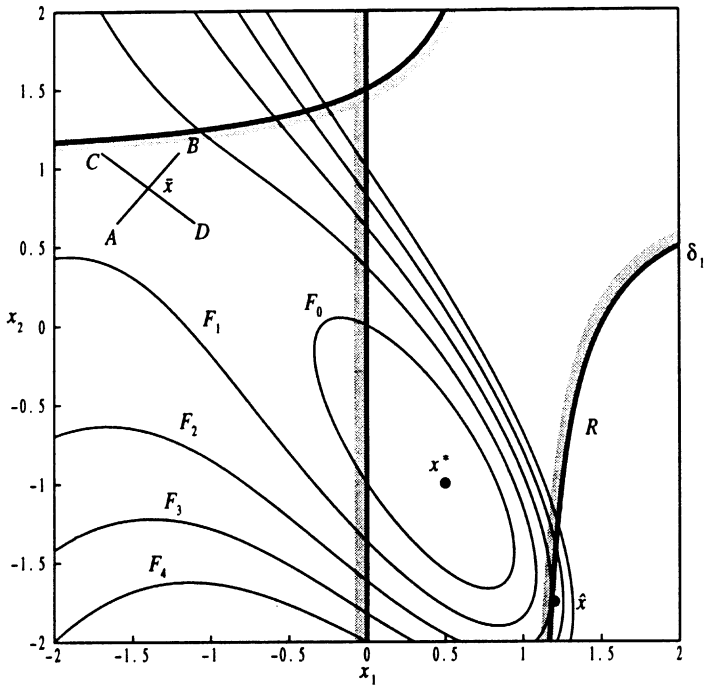


Figure 1

Figure 1 is a contour diagram of $F(x)$. The contours labelled $F_0, F_1, \ldots, F_4$ are isovalue contours, or lines along which the function $F(x)$ takes specific constant values. The point $x^* = (\frac{1}{2}, -1)^T$ is a **local unconstrained minimum**, that is, the value of $F(x^*)$ $(= 0)$ is less than at all the neighbouring points. A function may have several such minima. The lowest of the local minima is termed a **global minimum**. In the problem illustrated in Figure 1, x^* is the only local minimum. The point $\bar{x}$ is said to be a **saddle point** because it is a minimum along the line AB, but a maximum along CD.

If we add the constraint $x_1 \geq 0$ (a simple bound) to the problem of minimizing $F(x)$, the solution remains unaltered. In Figure 1 this constraint is represented by the straight line passing through $x_1 = 0$, and the shading on the line indicates the unacceptable region (i.e., $x_1 < 0$). The region in R^n satisfying the constraints of an optimization problem is termed the **feasible region**. A point satisfying the constraints is defined as a **feasible point**.

If we add the nonlinear constraint $c_1(x) : x_1 + x_2 - x_1x_2 - \frac{3}{2} \geq 0$, represented by the curved shaded line in Figure 1, then x^* is not a feasible point because $c_1(x^*) < 0$. The solution of the new constrained problem is $\hat{x} \simeq (1.1825, -1.7397)^T$, the feasible point with the smallest function value (where $F(\hat{x}) \simeq 3.0607$).

2.2.1 Gradient vector

The vector of first partial derivatives of $F(x)$ is called the **gradient vector**, and is denoted by $g(x)$, i.e.,

$$g(x) = \left[\frac{\partial F(x)}{\partial x_1}, \frac{\partial F(x)}{\partial x_2}, \ldots, \frac{\partial F(x)}{\partial x_n}\right]^T.$$

For the function illustrated in Figure 1,

$$g(x) = \begin{bmatrix} F(x) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_2 + 4x_1 + 2) \end{bmatrix}.$$

The gradient vector is of importance in optimization because it must be zero at an unconstrained minimum of any function with continuous first derivatives.

2.2.2 Hessian matrix

The matrix of second partial derivatives of a function is termed its **Hessian matrix**. The Hessian matrix of $F(x)$ is denoted by $G(x)$, and its (i,j)th element is given by $\partial^2 F(x)/\partial x_i \partial x_j$. If $F(x)$ has continuous second derivatives, then $G(x)$ must be positive semi-definite at any unconstrained minimum of F.

2.2.3 Jacobian matrix; matrix of constraint normals

In nonlinear least-squares problems, the matrix of first partial derivatives of the vector-valued function $f(x)$ is termed the **Jacobian matrix** of $f(x)$ and its (i,j)th component is $\partial f_i/\partial x_j$.

The vector of first partial derivatives of the constraint $c_i(x)$ is denoted by

$$a_i(x) = \left[\frac{\partial c_i(x)}{\partial x_1}, \frac{\partial c_i(x)}{\partial x_2}, \ldots, \frac{\partial c_i(x)}{\partial x_n}\right]^T.$$

The matrix whose columns are the vectors $\{a_i\}$ is termed the **matrix of constraint normals**. At a point $\hat{x}$, the vector $a_i(\hat{x})$ is orthogonal (normal) to the isovalue contour of $c_i(x)$ passing through $\hat{x}$; this relationship is illustrated for a two-dimensional function in Figure 2.

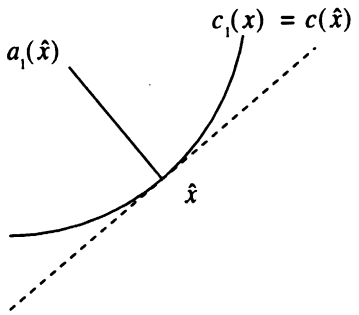


Figure 2

Note that if $c_i(x)$ is a linear constraint involving $a_i^T x$, then its vector of first partial derivatives is simply the vector a_i.

2.3 Sufficient Conditions for a Solution

All nonlinear functions will be assumed to have continuous second derivatives in the neighbourhood of the solution.

2.3.1 Unconstrained minimization

The following conditions are sufficient for the point x^* to be an unconstrained local minimum of $F(x)$:

(i) $\|g(x^*)\| = 0$; and
(ii) $G(x^*)$ is positive-definite,

where $\|g\|$ denotes the Euclidean length of g.

2.3.2 Minimization subject to bounds on the variables

At the solution of a bounds-constrained problem, variables which are not on their bounds are termed **free variables**. If it is known in advance which variables are on their bounds at the solution, the problem

can be solved as an unconstrained problem in just the free variables; thus, the sufficient conditions for a solution are similar to those for the unconstrained case, applied only to the free variables.

Sufficient conditions for a feasible point x^* to be the solution of a bounds-constrained problem are as follows:

(i) $\|\bar{g}(x^*)\| = 0$; and
(ii) $\bar{G}(x^*)$ is positive-definite; and
(iii) $g_j(x^*) < 0, x_j = u_j$; $g_j(x^*) > 0, x_j = l_j$,

where $\bar{g}(x)$ is the gradient of $F(x)$ with respect to the free variables, and $\bar{G}(x)$ is the Hessian matrix of $F(x)$ with respect to the free variables. The extra condition (iii) ensures that $F(x)$ cannot be reduced by moving off one or more of the bounds.

2.3.3 Linearly-constrained minimization

For the sake of simplicity, the following description does not include a specific treatment of bounds or range constraints, since the results for general linear inequality constraints can be applied directly to these cases.

At a solution x^*, of a linearly-constrained problem, the constraints which hold as equalities are called the **active** or **binding** constraints. Assume that there are t active constraints at the solution x^*, and let $\hat{A}$ denote the matrix whose columns are the columns of A corresponding to the active constraints, with $\hat{b}$ the vector similarly obtained from b; then

$$\hat{A}^T x^* = \hat{b}.$$

The matrix Z is defined as an $n \times (n-t)$ matrix satisfying:

$$\hat{A}^T Z = 0;$$
$$Z^T Z = I.$$

The columns of Z form an orthogonal basis for the set of vectors orthogonal to the columns of $\hat{A}$.

Define

$g_Z(x) = Z^T g(x)$, the **projected gradient vector** of $F(x)$;
$G_Z(x) = Z^T G(x) Z$, the **projected Hessian matrix** of $F(x)$.

At the solution of a linearly-constrained problem, the projected gradient vector must be zero, which implies that the gradient vector $g(x^*)$ can be written as a linear combination of the columns of $\hat{A}$, i.e., $g(x^*) = \sum_{i=1}^{t} \lambda_i^* \hat{a}_i = \hat{A}\lambda^*$. The scalar λ_i^* is defined as the **Lagrange-multiplier** corresponding to the ith active constraint. A simple interpretation of the ith Lagrange-multiplier is that it gives the gradient of $F(x)$ along the ith active constraint normal; a convenient definition of the Lagrange-multiplier vector (although not a recommended method for computation) is:

$$\lambda^* = (\hat{A}^T \hat{A})^{-1} \hat{A}^T g(x^*).$$

Sufficient conditions for x^* to be the solution of a linearly-constrained problem are:

(i) x^* is feasible, and $\hat{A}^T x^* = \hat{b}$; and
(ii) $\|g_Z(x^*)\| = 0$, or equivalently, $g(x^*) = \hat{A}\lambda^*$; and
(iii) $G_Z(x^*)$ is positive-definite; and
(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^T x^* \geq \hat{b}_i$;
$\lambda_i^* < 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^T x^* \leq \hat{b}_i$.
The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

2.3.4 Nonlinearly-constrained minimization

For nonlinearly-constrained problems, much of the terminology is defined exactly as in the linearly-constrained case. The set of active constraints at x again means the set of constraints that hold as equalities at x, with corresponding definitions of $\hat{c}$ and $\hat{A}$: the vector $\hat{c}(x)$ contains the active constraint functions, and the columns of $\hat{A}(x)$ are the gradient vectors of the active constraints. As before, Z is defined in terms of $\hat{A}(x)$ as a matrix such that:

$$\hat{A}^T Z = 0;$$
$$Z^T Z = I$$

where the dependence on x has been suppressed for compactness.

The projected gradient vector $g_Z(x)$ is the vector $Z^T g(x)$. At the solution x^* of a nonlinearly-constrained problem, the projected gradient must be zero, which implies the existence of Lagrange-multipliers corresponding to the active constraints, i.e., $g(x^*) = \hat{A}(x^*)\lambda^*$.

The **Lagrangian function** is given by:

$$L(x, \lambda) = F(x) - \lambda^T \hat{c}(x).$$

We define $g_L(x)$ as the gradient of the Lagrangian function; $G_L(x)$ as its Hessian matrix, and $\hat{G}_L(x)$ as its projected Hessian matrix, i.e., $\hat{G}_L = Z^T G_L Z$.

Sufficient conditions for x^* to be the solution of a nonlinearly-constrained problem are:

(i) x^* is feasible, and $\hat{c}(x^*) = 0$; and
(ii) $\|g_Z(x^*)\| = 0$, or, equivalently, $g(x^*) = \hat{A}(x^*)\lambda^*$; and
(iii) $\hat{G}_L(x^*)$ is positive-definite; and
(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint of the form $\hat{c}_i \geq 0$.

The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

Note that condition (ii) implies that the projected gradient of the Lagrangian function must also be zero at x^*, since the application of Z^T annihilates the matrix $\hat{A}(x^*)$.

2.4 Background to Optimization Methods

All the algorithms contained in this chapter generate an iterative sequence $\{x^{(k)}\}$ that converges to the solution x^* in the limit, except for some special problem categories (i.e., linear and quadratic programming). To terminate computation of the sequence, a convergence test is performed to determine whether the current estimate of the solution is an adequate approximation. The convergence tests are discussed in Section 2.6.

Most of the methods construct a sequence $\{x^{(k)}\}$ satisfying:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)},$$

where the vector $p^{(k)}$ is termed the **direction of search**, and $\alpha^{(k)}$ is the **steplength**. The steplength $\alpha^{(k)}$ is chosen so that $F(x^{(k+1)}) < F(x^{(k)})$ and is computed using one of the techniques for one-dimensional optimization referred to in Section 2.4.1.

2.4.1 One-dimensional optimization

The Library contains two special routines for minimizing a function of a single variable. Both routines are based on safeguarded polynomial approximation. One routine requires function evaluations only and fits a quadratic polynomial whilst the other requires function and gradient evaluations and fits a cubic polynomial. See Section 4.1. of Gill *et al.* [5].

2.4.2 Methods for unconstrained optimization

The distinctions among methods arise primarily from the need to use varying levels of information about derivatives of $F(x)$ in defining the search direction. We describe three basic approaches to unconstrained problems, which may be extended to other problem categories. Since a full description of the methods would fill several volumes, the discussion here can do little more than allude to the processes involved, and direct the user to other sources for a full explanation.

(a) **Newton-type** Methods (Modified Newton Methods)

Newton-type methods use the Hessian matrix $G(x^{(k)})$, or a finite-difference approximation to $G(x^{(k)})$, to define the search direction. The routines in the Library either require a subroutine that computes the elements of $G(x^{(k)})$ directly, or they approximate $G(x^{(k)})$ by finite-differences.

Newton-type methods are the most powerful methods available for general problems and will find the minimum of a quadratic function in one iteration. See Sections 4.4. and 4.5.1. of Gill *et al.* [5].

(b) **Quasi-Newton** Methods

Quasi-Newton methods approximate the Hessian $G(x^{(k)})$ by a matrix $B^{(k)}$ which is modified at each iteration to include information obtained about the curvature of F along the current search direction $p^{(k)}$. Although not as robust as Newton-type methods, quasi-Newton methods can be more efficient because $G(x^{(k)})$ is not computed directly, or approximated by finite-differences. Quasi-Newton methods minimize a quadratic function in n iterations. See Section 4.5.2 of Gill *et al.* [5].

(c) **Conjugate-Gradient** Methods

Unlike Newton-type and quasi-Newton methods, conjugate-gradient methods do not require the storage of an n by n matrix and so are ideally suited to solve large problems. Conjugate-gradient type methods are not usually as reliable or efficient as Newton-type, or quasi-Newton methods. See Section 4.8.3 of Gill *et al.* [5].

2.4.3 Methods for nonlinear least-squares problems

These methods are similar to those for unconstrained optimization, but exploit the special structure of the Hessian matrix to give improved computational efficiency.

Since

$$F(x) = \sum_{i=1}^{m} f_i^2(x)$$

the Hessian matrix $G(x)$ is of the form

$$G(x) = 2\left(J(x)^T J(x) + \sum_{i=1}^{m} f_i(x) G_i(x)\right),$$

where $J(x)$ is the Jacobian matrix of $f(x)$, and $G_i(x)$ is the Hessian matrix of $f_i(x)$.

In the neighbourhood of the solution, $\|f(x)\|$ is often small compared to $\|J(x)^T J(x)\|$ (for example, when $f(x)$ represents the goodness of fit of a nonlinear model to observed data). In such cases, $2J(x)^T J(x)$ may be an adequate approximation to $G(x)$, thereby avoiding the need to compute or approximate second derivatives of $\{f_i(x)\}$. See Section 4.7 of Gill *et al.* [5].

2.4.4 Methods for handling constraints

Bounds on the variables are dealt with by fixing some of the variables on their bounds and adjusting the remaining free variables to minimize the function. By examining estimates of the Lagrange-multipliers it is possible to adjust the set of variables fixed on their bounds so that eventually the bounds active at the solution should be correctly identified. This type of method is called an **active set method**. One feature of such methods is that, given an initial feasible point, all approximations $x^{(k)}$ are feasible. This approach can be extended to general linear constraints. At a point, x, the set of constraints which hold as equalities being used to predict, or approximate, the set of active constraints is called the **working set**.

Nonlinear constraints are more difficult to handle. If at all possible, it is usually beneficial to avoid including nonlinear constraints during the formulation of the problem. The methods currently implemented in the Library handle nonlinearly constrained problems by transforming them into a sequence of quadratic programming problems. A feature of such methods is that $x^{(k)}$ is not guaranteed to be feasible except in the limit, and this is certainly true of the routines currently in the Library. See Chapter 6, particularly Sections 6.4 and 6.5, of Gill *et al.* [5].

Anyone interested in a detailed description of methods for optimization should consult the references.

2.5 Scaling

Scaling (in a broadly defined sense) often has a significant influence on the performance of optimization methods. Since convergence tolerances and other criteria are necessarily based on an implicit definition of 'small' and 'large', problems with unusual or unbalanced scaling may cause difficulties for some algorithms. Although there are currently no user-callable scaling routines in the Library, scaling is automatically performed by default in the routines which solve sparse LP, QP or NLP problems. The following sections present some general comments on problem scaling.

2.5.1 Transformation of variables

One method of scaling is to transform the variables from their original representation, which may reflect the physical nature of the problem, to variables that have certain desirable properties in terms of optimization. It is generally helpful for the following conditions to be satisfied:

(i) the variables are all of similar magnitude in the region of interest;
(ii) a fixed change in any of the variables results in similar changes in $F(x)$. Ideally, a unit change in any variable produces a unit change in $F(x)$;
(iii) the variables are transformed so as to avoid cancellation error in the evaluation of $F(x)$.

Normally, users should restrict themselves to linear transformations of variables, although occasionally nonlinear transformations are possible. The most common such transformation (and often the most appropriate) is of the form

$$x_{\text{new}} = Dx_{\text{old}},$$

where D is a diagonal matrix with constant coefficients. Our experience suggests that more use should be made of the transformation

$$x_{\text{new}} = Dx_{\text{old}} + v,$$

where v is a constant vector.

Consider, for example, a problem in which the variable x_3 represents the position of the peak of a Gaussian curve to be fitted to data for which the extreme values are 150 and 170; therefore x_3 is known to lie in the range 150–170. One possible scaling would be to define a new variable $\bar{x}_3$, given by

$$\bar{x}_3 = \frac{x_3}{170}.$$

A better transformation, however, is given by defining $\bar{x}_3$ as

$$\bar{x}_3 = \frac{x_3 - 160}{10}.$$

Frequently, an improvement in the accuracy of evaluation of $F(x)$ can result if the variables are scaled before the routines to evaluate $F(x)$ are coded. For instance, in the above problem just mentioned of Gaussian curve fitting, x_3 may always occur in terms of the form $(x_3 - x_m)$, where x_m is a constant representing the mean peak position.

2.5.2 Scaling the objective function

The objective function has already been mentioned in the discussion of scaling the variables. The solution of a given problem is unaltered if $F(x)$ is multiplied by a positive constant, or if a constant value is added to $F(x)$. It is generally preferable for the objective function to be of the order of unity in the region of interest; thus, if in the original formulation $F(x)$ is always of the order of 10^{+5} (say), then the value of $F(x)$ should be multiplied by 10^{-5} when evaluating the function within an optimization routine. If a constant is added or subtracted in the computation of $F(x)$, usually it should be omitted – i.e., it is better to formulate $F(x)$ as $x_1^2 + x_2^2$ rather than as $x_1^2 + x_2^2 + 1000$ or even $x_1^2 + x_2^2 + 1$. The inclusion of such a constant in the calculation of $F(x)$ can result in a loss of significant figures.

2.5.3 Scaling the constraints

A 'well scaled' set of constraints has two main properties. Firstly, each constraint should be well conditioned with respect to perturbations of the variables. Secondly, the constraints should be balanced with respect to each other, i.e., all the constraints should have 'equal weight' in the solution process.

The solution of a linearly- or nonlinearly-constrained problem is unaltered if the ith constraint is multiplied by a positive weight w_i. At the approximation of the solution determined by a Library routine, any active linear constraints will (in general) be satisfied 'exactly' (i.e., to within the tolerance defined by ***machine precision***) if they have been properly scaled. This is in contrast to any active nonlinear constraints, which will not (in general) be satisfied 'exactly' but will have 'small' values (for example, $\hat{c}_1(x^*) = 10^{-8}$, $\hat{c}_2(x^*) = -10^{-6}$, and so on). In general, this discrepancy will be minimized if the constraints are weighted so that a unit change in x produces a similar change in each constraint.

A second reason for introducing weights is related to the effect of the size of the constraints on the Lagrange-multiplier estimates and, consequently, on the active set strategy. This means that different sets of weights may cause an algorithm to produce different sequences of iterates. Additional discussion is given in Gill *et al.* [5].

2.6 Analysis of Computed Results

2.6.1 Convergence criteria

The convergence criteria inevitably vary from routine to routine, since in some cases more information is available to be checked (for example, is the Hessian matrix positive-definite?), and different checks need to be made for different problem categories (for example, in constrained minimization it is necessary to verify whether a trial solution is feasible). Nonetheless, the underlying principles of the various criteria are the same; in non-mathematical terms, they are:

(i) is the sequence $\{x^{(k)}\}$ converging?
(ii) is the sequence $\{F^{(k)}\}$ converging?
(iii) are the necessary and sufficient conditions for the solution satisfied?

The decision as to whether a sequence is converging is necessarily speculative. The criterion used in the present routines is to assume convergence if the relative change occurring between two successive iterations is less than some prescribed quantity. Criterion (iii) is the most reliable but often the conditions cannot be checked fully because not all the required information may be available.

2.6.2 Checking results

Little *a priori* guidance can be given as to the quality of the solution found by a nonlinear optimization algorithm, since no guarantees can be given that the methods will not fail. Therefore, the user should always check the computed solution even if the routine reports success. Frequently a 'solution' may have been found even when the routine does not report a success. The reason for this apparent contradiction is that the routine needs to assess the accuracy of the solution. This assessment is not an exact process and consequently may be unduly pessimistic. Any 'solution' is in general only an approximation to the exact solution, and it is possible that the accuracy specified by the user is too stringent.

Further confirmation can be sought by trying to check whether or not convergence tests are almost satisfied, or whether or not some of the sufficient conditions are nearly satisfied. When it is thought that a routine has returned a non-zero value of IFAIL only because the requirements for 'success' were too stringent it may be worth restarting with increased convergence tolerances.

For nonlinearly-constrained problems, check whether the solution returned is feasible, or nearly feasible; if not, the solution returned is not an adequate solution.

Confidence in a solution may be increased by re-solving the problem with a different initial approximation to the solution. See Section 8.3 of Gill *et al.* [5] for further information.

2.6.3 Monitoring progress

Many of the routines in the chapter have facilities to allow the user to monitor the progress of the minimization process, and users are encouraged to make use of these facilities. Monitoring information can be a great aid in assessing whether or not a satisfactory solution has been obtained, and in indicating difficulties in the minimization problem or in the ability of the routine to cope with the problem.

The behaviour of the function, the estimated solution and first derivatives can help in deciding whether a solution is acceptable and what to do in the event of a return with a non-zero value of IFAIL.

2.6.4 Confidence intervals for least-squares solutions

When estimates of the parameters in a nonlinear least-squares problem have been found, it may be necessary to estimate the variances of the parameters and the fitted function. These can be calculated from the Hessian of $F(x)$ at the solution.

In many least-squares problems, the Hessian is adequately approximated at the solution by $G = 2J^TJ$ (see Section 2.4.3). The Jacobian, J, or a factorization of J is returned by all the comprehensive least-squares routines and, in addition, a routine is available in the Library to estimate variances of the parameters following the use of most of the nonlinear least-squares routines, in the case that $G = 2J^TJ$ is an adequate approximation.

Let H be the inverse of G, and S be the sum of squares, both calculated at the solution $\bar{x}$; an unbiased estimate of the **variance** of the ith parameter x_i is

$$\text{var}\,\bar{x}_i = \frac{2S}{m-n}H_{ii}$$

and an unbiased estimate of the covariance of $\bar{x}_i$ and $\bar{x}_j$ is

$$covar(\bar{x}_i, \bar{x}_j) = \frac{2S}{m-n}H_{ij}.$$

If x^* is the true solution, then the $100(1-\beta)\%$ **confidence interval** on $\bar{x}$ is

$$\bar{x}_i - \sqrt{\text{var}\,\bar{x}_i}.t_{(1-\beta/2,m-n)} < x_i^* < \bar{x}_i + \sqrt{\text{var}\,\bar{x}_i}.t_{(1-\beta/2,m-n)}, \quad i = 1,2,\ldots,n$$

where $t_{(1-\beta/2,m-n)}$ is the $100(1-\beta)/2$ percentage point of the $t-$distribution with $m-n$ degrees of freedom.

In the majority of problems, the residuals f_i, for $i = 1,2,\ldots,m$, contain the difference between the values of a model function $\phi(z,x)$ calculated for m different values of the independent variable z, and the corresponding observed values at these points. The minimization process determines the parameters, or constants x, of the fitted function $\phi(z,x)$. For any value, $\bar{z}$, of the independent variable z, an unbiased estimate of the **variance** of ϕ is

$$\text{var}\,\phi = \frac{2S}{m-n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\frac{\partial\phi}{\partial x_i}\right]_{\bar{z}}\left[\frac{\partial\phi}{\partial x_j}\right]_{\bar{z}}H_{ij}.$$

The $100(1-\beta)\%$ **confidence interval** on F at the point $\bar{z}$ is

$$\phi(\bar{z},\bar{x}) - \sqrt{\text{var}\,\phi}.t_{(\beta/2,m-n)} < \phi(\bar{z},x^*) < \phi(\bar{z},\bar{x}) + \sqrt{\text{var}\,\phi}.t_{(\beta/2,m-n)}.$$

For further details on the analysis of least-squares solutions see Bard [1] and Wolberg [7].

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

The choice of routine depends on several factors: the type of problem (unconstrained, etc.); the level of derivative information available (function values only, etc.); the experience of the user (there are easy-to-use versions of some routines); whether or not storage is a problem; and whether computational time has a high priority. Not all choices are catered for in the current version of the Library.

3.1 Easy-to-use and Comprehensive Routines

Many routines appear in the Library in two forms: a comprehensive form and an easy-to-use form. The objective in the easy-to-use forms is to make the routine simple to use by including in the calling sequence only those parameters absolutely essential to the definition of the problem, as opposed to parameters relevant to the solution method. The comprehensive routines have additional parameters which allow the experienced user to improve their efficiency by 'tuning' the method to a particular problem. For the casual or inexperienced user, this feature is of little value and may in some cases cause a failure because of a poor choice of some parameters.

In the easy-to-use routines, these extra parameters are determined either by fixing them at a known safe and reasonably efficient value, or by an auxiliary routine which generates a 'good' value automatically.

For routines introduced since Mark 12 of the Library a different approach has been adopted towards the choice of easy-to-use and comprehensive routines. The optimization routine has an easy-to-use parameter list, but additional parameters may be changed from their default values by calling an 'option' setting routine prior to the call to the main optimization routine. This approach has the advantages of allowing the options to be given in the form of keywords and requiring only those options that are to be different from their default values to be set.

3.2 Reverse Communication Routines

Most of the routines in this chapter are called just once in order to compute the minimum of a given objective function subject to a set of constraints on the variables. The objective function and nonlinear constraints (if any) are specified by the user and written as subroutines to a very rigid format described in the relevant routine document. Such subroutines usually appear in the argument list of the minimization routine.

For the majority of applications this is the simplest and most convenient usage. Sometimes however this approach can be restrictive:

(i) when the required format of the user's subroutine does not allow useful information to be passed conveniently to and from the user's calling program;
(ii) when the minimization routine is being called from another computer language, such as Visual Basic, which does not fully support procedure arguments in a way that is compatible with the Library.

A way around these problems is to supply **reverse communication** routines. Instead of performing complete optimizations, these routines perform one step in the solution process before returning to the calling program with an appropriate flag (IREVCM) set. The value of IREVCM determines whether the minimization process has finished or whether fresh information is required. In the latter case the user calculates this information (in the form of a vector or as a scalar, as appropriate) and re-enters the reverse communication routine with the information contained in appropriate arguments. Thus the user has the responsibility for providing the iterative loop in the minimization process, but as compensation, has an extremely flexible and basic user-interface to the reverse communication routine.

The only reverse communication routine in this chapter is E04UFF, which solves dense NLP problems and uses exactly the same method as E04UCF.

3.3 Service Routines

One of the most common errors in the use of optimization routines is that user-supplied subroutines do not evaluate the relevant partial derivatives correctly. Because exact gradient information normally enhances efficiency in all areas of optimization, the user should be encouraged to provide analytical derivatives whenever possible. However, mistakes in the computation of derivatives can result in serious and obscure run-time errors. Consequently, **service routines** are provided to perform an elementary check on the user-supplied gradients. These routines are inexpensive to use in terms of the number of calls they require to user-supplied routines.

The appropriate checking routines are as follows:

Minimization routine	**Checking routine(s)**
E04KDF	E04HCF
E04LBF	E04HCF and E04HDF
E04GBF	E04YAF
E04GDF	E04YAF
E04HEF	E04YAF and E04YBF

It should be noted that routines E04UCF, E04UFF, E04UGF and E04UNF each incorporate a check on the gradients being supplied. This involves verifying the gradients at the first point that satisfies the linear constraints and bounds. There is also an option to perform a more reliable (but more expensive) check on the individual gradient elements being supplied. Note that the checks are not infallible.

A second type of service routine computes a set of finite-differences to be used when approximating first derivatives. Such differences are required as input parameters by some routines that use only function evaluations.

E04YCF estimates selected elements of the variance-covariance matrix for the computed regression parameters following the use of a nonlinear least-squares routine.

E04XAF estimates the gradient and Hessian of a function at a point, given a routine to calculate function values only, or estimates the Hessian of a function at a point, given a routine to calculate function and gradient values.

3.4 Function Evaluations at Infeasible Points

All the routines for constrained problems will ensure that any evaluations of the objective function occur at points which **approximately** satisfy any **simple bounds** or **linear constraints**. Satisfaction of such constraints is only approximate because routines which estimate derivatives by finite-differences may require function evaluations at points which just violate such constraints even though the current iteration just satisfies them.

There is no attempt to ensure that the current iteration satisfies any nonlinear constraints. Users who wish to prevent their objective function being evaluated outside some known region (where it may be undefined or not practically computable), may try to confine the iteration within this region by imposing suitable simple bounds or linear constraints (but beware as this may create new local minima where these constraints are active).

Note also that some routines allow the user-supplied routine to return a parameter (IFLAG or MODE) with a negative value to force an immediate clean exit from the minimization routine when the objective function (or nonlinear constraints where appropriate) cannot be evaluated.

3.5 Related Problems

Apart from the standard types of optimization problem, there are other related problems which can be solved by routines in this or other chapters of the Library.

H02BBF solves **dense integer LP** problems, H02CBF solves **dense integer QP** problems, H02CEF solves **sparse integer QP** problems and H03ABF solves a special type of such problem known as a **'transportation' problem**.

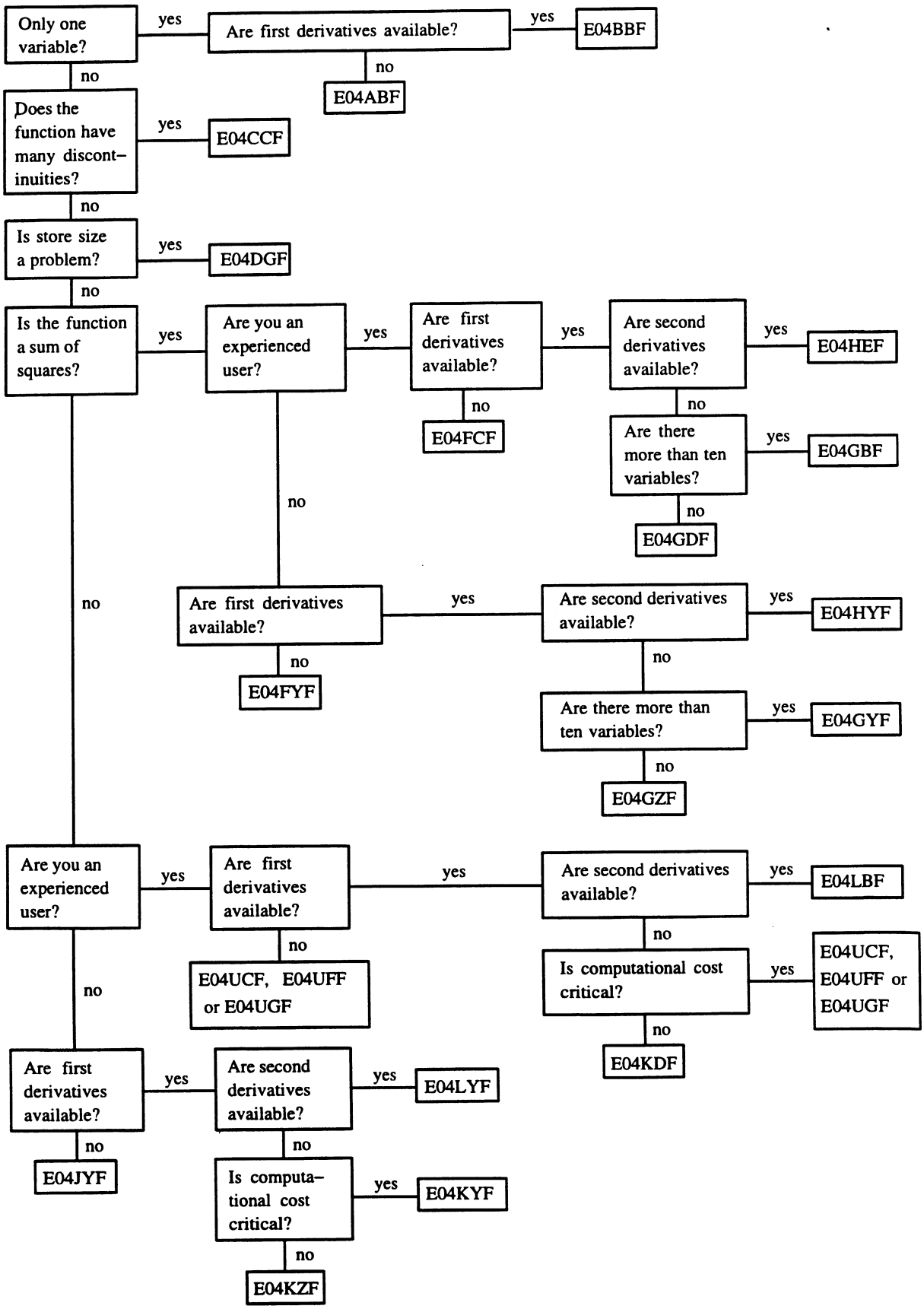
Several routines in Chapter F04 solve **linear least-squares problems**, i.e., minimize $\sum_{i=1}^{m} r_i(x)^2$ where $r_i(x) = b_i - \sum_{j=1}^{n} a_{ij}x_j$.

E02GAF solves an overdetermined system of linear equations in the l_1 norm, i.e., minimizes $\sum_{i=1}^{m} |r_i(x)|$, with r_i as above, and E02GBF solves the same problem subject to linear inequality constraints.

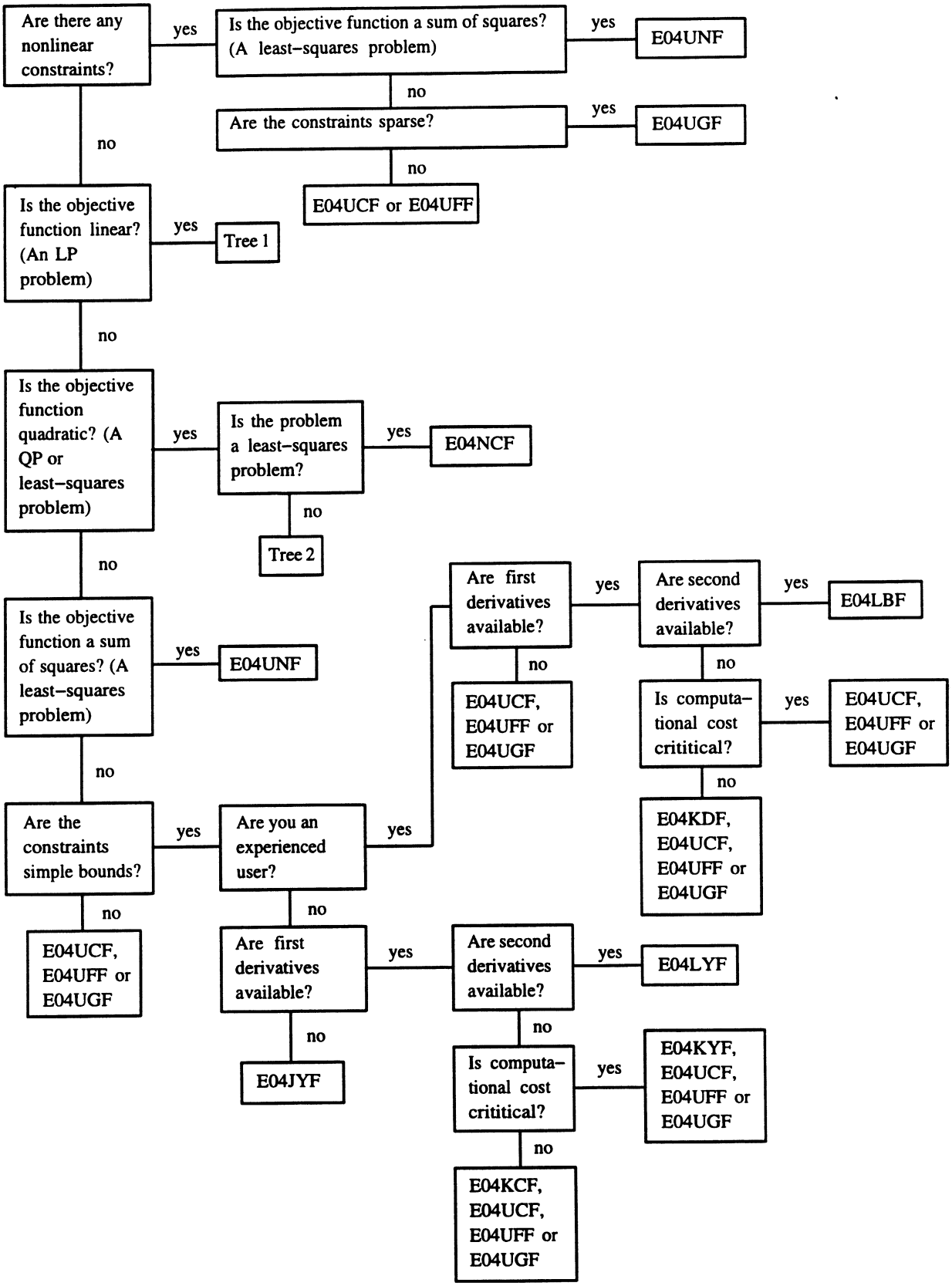
E02GCF solves an overdetermined system of linear equations in the l_∞ norm, i.e., minimizes $\max_i |r_i(x)|$, with r_i as above.

4 Decision Trees

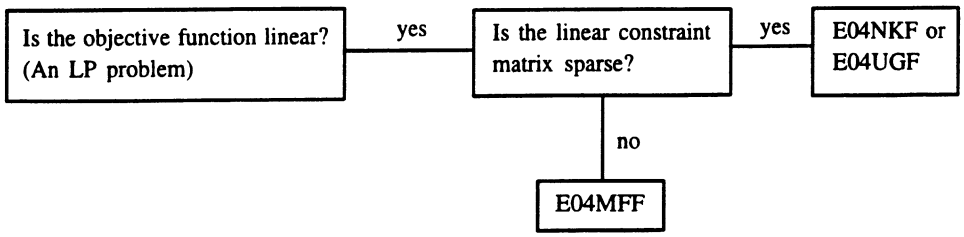
Selection chart for unconstrained problems



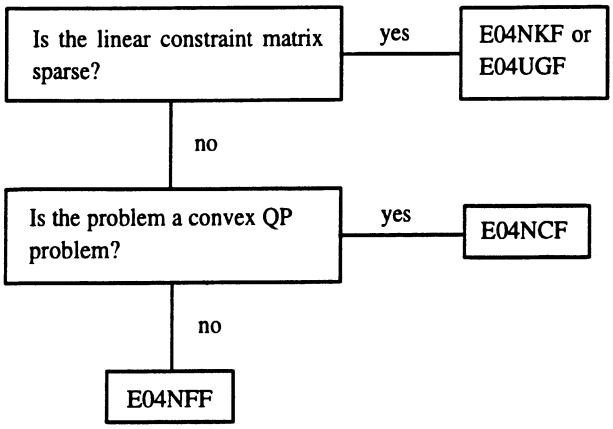
Selection chart for bound-constrained, linearly-constrained and nonlinearly-constrained problems



Tree 1



Tree 2



5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E04CGF	E04DBF	E04DEF	E04DFF	E04EBF	E04FDF
E04GCF	E04GEF	E04HFF	E04HBF	E04JAF	E04JBF
E04KAF	E04KBF	E04KCF	E04LAF	E04MBF	E04NAF
E04UAF	E04UPF	E04VCF	E04VDF		

6 References

[1] Bard Y (1974) *Nonlinear Parameter Estimation* Academic Press

[2] Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press

[3] Fletcher R (1987) *Practical Methods of Optimization* Wiley (2nd Edition)

[4] Gill P E and Murray W (ed.) (1974) *Numerical Methods for Constrained Optimization* Academic Press

[5] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

[6] Murray W (ed.) (1972) *Numerical Methods for Unconstrained Optimization* Academic Press

[7] Wolberg J R (1967) *Prediction Analysis* Van Nostrand

E04ABF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04ABF searches for a minimum, in a given finite interval, of a continuous function of a single variable, using function values only. The method (based on quadratic interpolation) is intended for functions which have a continuous first derivative (although it will usually work if the derivative has occasional discontinuities).

2. Specification

```
SUBROUTINE E04ABF (FUNCT, E1, E2, A, B, MAXCAL, X, F, IFAIL)
INTEGER          MAXCAL, IFAIL
real             E1, E2, A, B, X, F
EXTERNAL         FUNCT
```

3. Description

E04ABF is applicable to problems of the form:

Minimize $F(x)$ subject to $a \leq x \leq b$.

It normally computes a sequence of x values which tend in the limit to a minimum of $F(x)$ subject to the given bounds. It also progressively reduces the interval $[a,b]$ in which the minimum is known to lie. It uses the safeguarded quadratic-interpolation method described in Gill and Murray [1].

The user must supply a subroutine FUNCT to evaluate $F(x)$. The parameters E1 and E2 together specify the accuracy

$Tol(x) = \text{E1} \times |x| + \text{E2}$

to which the position of the minimum is required. Note that FUNCT is never called at any point which is closer than $Tol(x)$ to a previous point.

If the original interval $[a,b]$ contains more than one minimum, E04ABF will normally find one of the minima.

4. References

[1] GILL, P.E. and MURRAY, W.
Safeguarded steplength algorithms for optimization using descent methods.
National Physical Laboratory Report NAC 37, 1973.

5. Parameters

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

This routine must be supplied by the user to calculate the value of the function $F(x)$ at any point x in $[a,b]$. It should be tested separately before being used in conjunction with E04ABF.

Its specification is:

```
SUBROUTINE FUNCT(XC, FC)
real             XC, FC
```

1: XC – ***real***. *Input*

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

2: FC – ***real.*** *Output*

On exit: FC must be set to the value of the function F at the current point x.

FUNCT must be declared as EXTERNAL in the (sub)program from which E04ABF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: E1 – ***real.*** *Input/Output*

On entry: the relative accuracy to which the position of a minimum is required. (Note that, since E1 is a relative tolerance, the scaling of x is automatically taken into account.)

E1 should be no smaller than 2ϵ, and preferably not much less than $\sqrt{\epsilon}$, where ϵ is the ***machine precision.***

On exit: if the user sets E1 to 0.0 (or to any value less than ϵ), E04ABF will reset E1 to the default value $\sqrt{\epsilon}$ before starting the minimization process.

3: E2 – ***real.*** *Input/Output*

On entry: the absolute accuracy to which the position of a minimum is required. E2 should be no smaller than 2ϵ.

On exit: if the user sets E2 to 0.0 (or to any value less than ϵ), E04ABF will reset it to the default value $\sqrt{\epsilon}$.

4: A – ***real.*** *Input/Output*

On entry: the lower bound a of the interval containing a minimum.

On exit: an improved lower bound on the position of the minimum.

5: B – ***real.*** *Input/Output*

On entry: the upper bound b of the interval containing a minimum.

On exit: an improved upper bound on the position of the minimum.

6: MAXCAL – INTEGER. *Input/Output*

On entry: the maximum number of evaluations of $F(x)$ which the user is prepared to allow.

Constraint: MAXCAL $\geq$ 3. (Few problems will require more than 30.)

There will be an error exit (see Section 6) after MAXCAL calls of FUNCT.

On exit: the total number of times that FUNCT was actually called.

7: X – ***real.*** *Output*

On exit: the estimated position of the minimum.

8: F – ***real.*** *Output*

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

9: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, (A+E2) ≥ B,
or MAXCAL < 3,

IFAIL = 2

The number of calls of FUNCT has exceeded MAXCAL. This may have happened simply because MAXCAL was set too small for a particular problem, or may be due to a mistake in the user's routine FUNCT. If no mistake can be found in FUNCT, restart E04ABF (preferably with values of A and B given on exit from the previous call of E04ABF).

7. Accuracy

If $F(x)$ is δ–unimodal (see the Chapter Introduction) for some $\delta < Tol(x)$, where $Tol(x)$ = E1×|x| + E2, then, on exit, x approximates the minimum of $F(x)$ in the original interval $[a,b]$ with an error less than $3\times Tol(x)$.

8. Further Comments

Timing depends on the behaviour of $F(x)$, the accuracy demanded and the length of the interval $[a,b]$. Unless $F(x)$ can be evaluated very quickly, the run time will usually be dominated by the time spent in FUNCT.

If $F(x)$ has more than one minimum in the original interval $[a,b]$, E04ABF will determine an approximation x (and improved bounds a and b) for one of the minima.

If E04ABF finds an x such that $F(x-\delta_1) > F(x) < F(x+\delta_2)$ for some $\delta_1,\delta_2 \geq Tol(x)$, the interval $[x-\delta_1, x+\delta_2]$ will be regarded as containing a minimum, even if $F(x)$ is less than $F(x-\delta_1)$ and $F(x+\delta_2)$ only due to rounding errors in the user-supplied routine. Therefore FUNCT should be programmed to calculate $F(x)$ as accurately as possible, so that E04ABF will not be liable to find a spurious minimum.

9. Example

A sketch of the function

$$F(x) = \frac{\sin x}{x}$$

shows that it has a minimum somewhere in the range [3.5, 5.0]. The following program shows how E04ABF can be used to obtain a good approximation to the position of a minimum.

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.

```
*     E04ABF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B, EPS, F, T, X
      INTEGER          IFAIL, MAXCAL
*     .. External Subroutines ..
      EXTERNAL         E04ABF, FUNCT
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04ABF Example Program Results'
*     EPS and T are set to zero so that E04ABF will reset them to
*     the default values
      EPS = 0.0e0
      T = 0.0e0
*     The minimum is known to lie in the range (3.5, 5.0)
      A = 3.5e0
      B = 5.0e0
*     Allow 30 calls of FUNCT
      MAXCAL = 30
      IFAIL = 1
*
      CALL E04ABF(FUNCT,EPS,T,A,B,MAXCAL,X,F,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.1) THEN
         WRITE (NOUT,*) 'Parameter outside expected range'
      ELSE
         IF (IFAIL.EQ.2) THEN
            WRITE (NOUT,*)
     +        'Results after MAXCAL function evaluations are'
            WRITE (NOUT,*)
         END IF
         WRITE (NOUT,99999) 'The minimum lies in the interval ', A,
     +     ' to ', B
         WRITE (NOUT,99999) 'Its estimated position is ', X, ','
         WRITE (NOUT,99998) 'where the function value is ', F
         WRITE (NOUT,99997) MAXCAL, 'function evaluations were required'
      END IF
      STOP
*
99999 FORMAT (1X,A,F8.5,A,F8.5)
99998 FORMAT (1X,A,1P,e11.4)
99997 FORMAT (1X,I2,1X,A)
      END
*
      SUBROUTINE FUNCT(XC,FC)
*     Routine to evaluate F(x) at any point in (A, B)
*     .. Scalar Arguments ..
      real                FC, XC
*     .. Intrinsic Functions ..
      INTRINSIC           SIN
*     .. Executable Statements ..
      FC = SIN(XC)/XC
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E04ABF Example Program Results

The minimum lies in the interval  4.49341 to  4.49341
Its estimated position is  4.49341,
where the function value is -2.1723E-01
10 function evaluations were required
```

E04BBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04BBF searches for a minimum, in a given finite interval, of a continuous function of a single variable, using function and first derivative values. The method (based on cubic interpolation) is intended for functions which have a continuous first derivative (although it will usually work if the derivative has occasional discontinuities).

2. Specification

```
SUBROUTINE E04BBF (FUNCT, E1, E2, A, B, MAXCAL, X, F, G, IFAIL)
INTEGER          MAXCAL, IFAIL
real             E1, E2, A, B, X, F, G
EXTERNAL         FUNCT
```

3. Description

E04BBF is applicable to problems of the form:

Minimize $F(x)$ subject to $a \leq x \leq b$

when the first derivative $\frac{dF}{dx}$ can be calculated. The routine normally computes a sequence of x values which tend in the limit to a minimum of $F(x)$ subject to the given bounds. It also progressively reduces the interval $[a,b]$ in which the minimum is known to lie. It uses the safeguarded cubic-interpolation method described in Gill and Murray [1].

The subroutine FUNCT must be supplied by the user to evaluate $F(x)$ and its first derivative. The parameters E1 and E2 together specify the accuracy

$Tol(x) = \text{E1} \times |x| + \text{E2}$

to which the position of the minimum is required. Note that FUNCT is never called at a point which is closer than $Tol(x)$ to a previous point.

If the original interval $[a,b]$ contains more than one minimum, E04BBF will normally find one of the minima.

4. References

[1] GILL, P.E. and MURRAY, W.
Safeguarded steplength algorithms for optimization using descent methods.
National Physical Laboratory Report NAC 37, 1973.

5. Parameters

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

This routine must be supplied by the user to calculate the values of $F(x)$ and $\frac{dF}{dx}$ at any point x in $[a,b]$.

It should be tested separately before being used in conjunction with E04BBF.

Its specification is:

```
SUBROUTINE FUNCT(XC, FC, GC)
real             XC, FC, GC
```

1: XC – ***real***. *Input*

On entry: the point x at which the values of F and $\frac{dF}{dx}$ are required.

2: FC – ***real.*** *Output*

On exit: FC must be set to the value of the function F at the current point x.

3: GC – ***real.*** *Output*

On exit: GC must be set to the value of the first derivative $\frac{dF}{dx}$ at the current point x.

FUNCT must be declared as EXTERNAL in the (sub)program from which E04BBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: E1 – ***real.*** *Input/Output*

On entry: the relative accuracy to which the position of a minimum is required. (Note that, since E1 is a relative tolerance, the scaling of x is automatically taken into account.)

E1 should be no smaller than 2ε, and preferably not much less than $\sqrt{\varepsilon}$, where ε is the ***machine precision.***

On exit: if the user sets E1 to 0.0 (or to any value less than ε), E04BBF will reset E1 to the default value $\sqrt{\varepsilon}$ before starting the minimization process.

3: E2 – ***real.*** *Input/Output*

On entry: the absolute accuracy to which the position of a minimum is required. E2 should be no smaller than 2ε.

On exit: if the user sets E2 to 0.0 (or to any value less than ε), E04BBF will reset it to the default value $\sqrt{\varepsilon}$.

4: A – ***real.*** *Input/Output*

On entry: the lower bound a of the interval containing a minimum.

On exit: an improved lower bound on the position of the minimum.

5: B – ***real.*** *Input/Output*

On entry: the upper bound b of the interval containing a minimum.

On exit: an improved upper bound on the position of the minimum.

6: MAXCAL – INTEGER. *Input/Output*

On entry: MAXCAL must be set to the maximum number of calls of FUNCT which the user is prepared to allow.

Constraint: MAXCAL $\geq$ 2. (Few problems will require more than 20.)

There will be an error exit (see Section 6) after MAXCAL calls of FUNCT.

On exit: the total number of times than FUNCT was actually called.

7: X – ***real.*** *Output*

On exit: the estimated position of the minimum.

8: F – ***real.*** *Output*

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

9: G – ***real.*** *Output*

On exit: the value of the first derivative at the final point in X.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, (A+E2) ≥ B,
or MAXCAL < 2.

IFAIL = 2

The number of calls of FUNCT has exceeded MAXCAL. This may have happened simply because MAXCAL was set too small for a particular problem, or may be due to a mistake in the user's routine FUNCT. If no mistake can be found in FUNCT, restart E04BBF (preferably with the values of A and B given on exit from the previous call of E04BBF).

7. Accuracy

If $F(x)$ is δ-unimodal (see the Chapter Introduction) for some $\delta < Tol(x)$, where $Tol(x)$ = E1×$|x|$ + E2, then, on exit, x approximates the minimum of $F(x)$ in the original interval $[a,b]$ with an error less than $3\times Tol(x)$.

8. Further Comments

Timing depends on the behaviour of $F(x)$, the accuracy demanded and the length of the interval $[a,b]$. Unless $F(x)$ and $\frac{dF}{dx}$ can be evaluated very quickly, the run time will usually be dominated by the time spent in FUNCT.

If $F(x)$ has more than one minimum in the original interval $[a,b]$, E04BBF will determine an approximation x (and improved bounds a and b) for one of the minima.

If E04BBF finds an x such that $F(x-\delta_1) > F(x) < F(x+\delta_2)$ for some $\delta_1,\delta_2 \geq Tol(x)$, the interval $[x-\delta_1, x+\delta_2]$ will be regarded as containing a minimum, even if $F(x)$ is less than $F(x-\delta_1)$ and $F(x+\delta_2)$ only due to rounding errors in the user-supplied routine. Therefore FUNCT should be programmed to calculate $F(x)$ as accurately as possible, so that E04BBF will not be liable to find a spurious minimum. (For similar reasons, $\frac{dF}{dx}$ should be evaluated as accurately as possible.)

9. Example

A sketch of the function

$$F(x) = \frac{\sin x}{x}$$

shows that it has a minimum somewhere in the range [3.5, 5.0]. The following program shows how E04BBF can be used to obtain a good approximation to the position of a minimum.

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.

```
*     E04BBF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B, EPS, F, G, T, X
      INTEGER          IFAIL, MAXCAL
*     .. External Subroutines ..
      EXTERNAL         E04BBF, FUNCT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04BBF Example Program Results'
*     EPS and T are set to zero so that E04BBF will reset them to
*     the default values
      EPS = 0.0e0
      T = 0.0e0
*     The minimum is known to lie in the range (3.5, 5.0)
      A = 3.5e0
      B = 5.0e0
*     Allow 30 calls of FUNCT
      MAXCAL = 30
      IFAIL = 1
*
      CALL E04BBF(FUNCT,EPS,T,A,B,MAXCAL,X,F,G,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.1) THEN
         WRITE (NOUT,*) 'Parameter outside expected range'
      ELSE
         IF (IFAIL.EQ.2) THEN
            WRITE (NOUT,*) 'Results after MAXCAL calls of FUNCT are'
            WRITE (NOUT,*)
         END IF
         WRITE (NOUT,99999) 'The minimum lies in the interval ', A,
     +     ' to ', B
         WRITE (NOUT,99999) 'Its estimated position is ', X, ','
         WRITE (NOUT,99998) 'where the function value is ', F
         WRITE (NOUT,99998) 'and the gradient is ', G,
     +     ' (machine dependent)'
         WRITE (NOUT,99997) MAXCAL, ' calls of FUNCT were required'
      END IF
      STOP
*
99999 FORMAT (1X,A,F8.5,A,F8.5)
99998 FORMAT (1X,A,1P,e11.4,A)
99997 FORMAT (1X,I2,A)
      END
*
      SUBROUTINE FUNCT(XC,FC,GC)
*     Routine to evaluate F(x) and dF/dx at any point in (A, B)
*     .. Scalar Arguments ..
      real             FC, GC, XC
*     .. Intrinsic Functions ..
      INTRINSIC        COS, SIN
*     .. Executable Statements ..
      FC = SIN(XC)/XC
      GC = (COS(XC)-FC)/XC
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E04BBF Example Program Results

The minimum lies in the interval  4.49341 to  4.49341
Its estimated position is  4.49341,
where the function value is -2.1723E-01
and the gradient is  3.9434E-16 (machine dependent)
 6 calls of FUNCT were required
```

E04CCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04CCF minimizes a general function $F(x)$ of n independent variables $x = (x_1,x_2,...,x_n)^T$ by the Simplex method. Derivatives of the function need not be supplied.

2. Specification

```
      SUBROUTINE E04CCF (N, X, F, TOL, IW, W1, W2, W3, W4, W5, W6, FUNCT,
     1                   MONIT, MAXCAL, IFAIL)
      INTEGER           N, IW, MAXCAL, IFAIL
      real              X(N), F, TOL, W1(N), W2(N), W3(N), W4(N), W5(IW),
     1                  W6(IW,N)
      EXTERNAL          FUNCT, MONIT
```

3. Description

The routine finds an approximation to a minimum of a function of n variables. The user must supply a routine to calculate the value of the function for any set of values of the variables.

The method is iterative. A simplex of n+1 points is set up in the dimensional space of the variables (for example in 2 dimensions the simplex is a triangle) under the assumption that the problem has been scaled so that the values of the independent variables at the minimum are of order unity. The starting point provided by the user is the first vertex of the simplex, the remaining n vertices are generated by the routine (see Parkinson and Hutchinson [2]). The vertex of the simplex with the largest function value is reflected in the centre of gravity of the remaining vertices and the function value at this new point is compared with the remaining function values. Depending on the outcome of this test the new point is accepted or rejected, a further expansion move may be made, or a contraction may be carried out [1,2]. When no further progress can be made the sides of the simplex are reduced in length and the method is repeated.

The method tends to be slow, but it is robust and therefore very useful for functions that are subject to inaccuracies.

4. References

[1] NELDER, J.A. and MEAD, R.
A Simplex Method for Function Minimization.
Comput. J., 7, pp. 308-313, 1965.

[2] PARKINSON, J.M. and HUTCHINSON, D.
An investigation into the efficiency of variants of the simplex method. In: 'Numerical Methods for Nonlinear Optimization', F.A. Lootsma, (Ed.).
Academic Press, 1972.

5. Parameters

1: N – INTEGER. *Input*

On entry: the number n of independent variables.

Constraint: N > 0.

2: X(N) – ***real*** array. *Input/Output*

On entry: a guess at the position of the minimum. Note that the problem should be scaled so that the values of the X(i) are of order unity.

On exit: the value of x corresponding to the function value in F.

3: F – ***real.*** *Output*

On exit: the lowest function value found.

4: TOL – ***real.*** *Input*

On entry: the error tolerable in the result, as follows:

If f_i, for $i = 1,2,...,n+1$, are the individual function values at the vertices of a simplex and f_m is the mean of these values, then the routine will terminate when

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}(f_i - f_m)^2} < \text{TOL}.$$

Constraint: TOL must be greater than or equal to the ***machine precision.***

5: IW – INTEGER. *Input*

On entry: the value N + 1.

Constraint: IW = N + 1.

6: W1(N) – ***real*** array. *Workspace*
7: W2(N) – ***real*** array. *Workspace*
8: W3(N) – ***real*** array. *Workspace*
9: W4(N) – ***real*** array. *Workspace*
10: W5(IW) – ***real*** array. *Workspace*
11: W6(IW,N) – ***real*** array. *Workspace*

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

FUNCT must calculate the value of the function at XC and assign this value to FC. It should be tested separately before being used in conjunction with E04CCF (see the Chapter Introduction).

Its specification is:

```
SUBROUTINE FUNCT(N, XC, FC)
INTEGER    N
real       XC(N), FC
```

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 is required.

3: FC – ***real.*** *Output*

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

FUNCT must be declared as EXTERNAL in the (sub)program from which E04CCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

MONIT is called once every iteration in E04CCF. It can be used to print out the current values of any selection of its parameters but must not be used to change the values of the parameters.

Its specification is:

```
SUBROUTINE MONIT(FMIN, FMAX, SIM, N, IS, NCALL)
INTEGER    N, IS, NCALL
real       FMIN, FMAX, SIM(IS,N)
```

1: FMIN – ***real.*** *Input*

On entry: the smallest function value in the current simplex.

2: FMAX – ***real.*** *Input*

On entry: the largest function value in the current simplex.

3: SIM(IS,N) – ***real*** array. *Input*

On entry: the rows of SIM contain the position vectors of the vertices of the current simplex.

4: N – INTEGER. *Input*

On entry: the number of variables.

5: IS – INTEGER. *Input*

On entry: the first dimension of the array SIM.

6: NCALL – INTEGER. *Input*

On entry: the number of times that FUNCT has been called so far.

MONIT must be declared as EXTERNAL in the (sub)program from which E04CCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: MAXCAL – INTEGER. *Input*

On entry: the maximum number of function evaluations to be allowed.

Constraint: MAXCAL $\geq$ 1.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N < 1,
or TOL < ***machine precision***,
or IW $\neq$ N + 1,
or MAXCAL < 1.

IFAIL = 2

MAXCAL function evaluations have been completed, E04CCF has been terminated prematurely. Check the coding of the routine FUNCT before increasing the value of MAXCAL.

7. Accuracy

On a successful exit the accuracy will be as defined by TOL.

8. Further Comments

The time taken by the routine depends on the number of variables, the behaviour of the function and the distance of the starting point from the minimum. Each iteration consists of 1 or 2 function evaluations unless the size of the simplex is reduced, in which case n + 1 function evaluations are required.

9. Example

A simple program to locate a minimum of the function:

$$F = e^{x_1}(4x_1^2+2x_2^2+4x_1x_2+2x_2+1).$$

The input parameters to E04CCF are all set before the routine is called.

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.

```
*     E04CCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, IW
      PARAMETER        (N=2,IW=N+1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          IMONIT
*     .. Local Scalars ..
      real             F, TOL
      INTEGER          I, IFAIL, MAXCAL
*     .. Local Arrays ..
      real             SIM(IW,N), W1(N), W2(N), W3(N), W4(N), W5(IW),
     +                 X(N)
*     .. External Functions ..
      real             X02AJF
      EXTERNAL         X02AJF
*     .. External Subroutines ..
      EXTERNAL         E04CCF, FUNCT, MONIT
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Common blocks ..
      COMMON           /OUTP/IMONIT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04CCF Example Program Results'
*     ** Set IMONIT to 1 to obtain monitoring information **
      IMONIT = 0
      X(1) = -1.0e0
      X(2) = 1.0e0
      TOL = SQRT(X02AJF())
      MAXCAL = 100
      IFAIL = 0
*
      CALL E04CCF(N,X,F,TOL,IW,W1,W2,W3,W4,W5,SIM,FUNCT,MONIT,MAXCAL,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Final function value is ', F
      WRITE (NOUT,99999) 'at the point', (X(I),I=1,N)
      WRITE (NOUT,99998) 'This has error number', IFAIL
      STOP
*
99999 FORMAT (1X,A,2F12.4)
99998 FORMAT (1X,A,I3)
      END
*
      SUBROUTINE FUNCT(N,XC,FC)
*     .. Scalar Arguments ..
      real             FC
      INTEGER          N
*     .. Array Arguments ..
      real             XC(N)
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      FC = EXP(XC(1))*(4.0e0*XC(1)*(XC(1)+XC(2))+2.0e0*XC(2)*(XC(2)
     +     +1.0e0)+1.0e0)
      RETURN
      END
*
      SUBROUTINE MONIT(FMIN,FMAX,SIM,N,N1,NCALL)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             FMAX, FMIN
      INTEGER          N, N1, NCALL
*     .. Array Arguments ..
      real             SIM(N1,N)
*     .. Scalars in Common ..
      INTEGER          IMONIT
*     .. Local Scalars ..
      INTEGER          I, J
*     .. Common blocks ..
      COMMON           /OUTP/IMONIT
*     .. Executable Statements ..
      IF (IMONIT.NE.0) THEN
         WRITE (NOUT,99999) 'After', NCALL,
     +     '  function calls, the value is', FMIN, ' with simplex'
         WRITE (NOUT,99998) ((SIM(I,J),J=1,N),I=1,N1)
      END IF
      RETURN
*
99999 FORMAT (1X,A,I5,A,F10.4,A)
99998 FORMAT (1X,2F12.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E04CCF Example Program Results

 Final function value is       0.0000
 at the point      0.5000     -0.9999
 This has error number  0
```

E04DGF – NAG Fortran Library Routine Document

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.

Note. *This routine uses* optional parameters *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Section 1 to Section 9 of this document. Refer to the additional Section 10 and Section 11 for a description of the algorithm and the specification of the optional parameters.*

Warning: the specification of the optional parameter **Maximum Step Length** changed at Mark 16.

1 Purpose

E04DGF minimizes an unconstrained nonlinear function of several variables using a pre-conditioned, limited memory quasi-Newton conjugate gradient method. First derivatives (or an 'acceptable' finite difference approximation to them) are required. It is intended for use on large scale problems.

2 Specification

```
   SUBROUTINE E04DGF(OBJFUN, ITER, OBJF, OBJGRD, X, IWORK, WORK,
  1                  IUSER, USER, IFAIL)
   INTEGER           ITER, IWORK(N+1), IUSER(*), IFAIL
   real              OBJF, OBJGRD(N), X(N), WORK(13*N), USER(*)
   EXTERNAL          OBJFUN
```

3 Description

E04DGF is designed to solve unconstrained minimization problems of the form

$$\underset{x \in R^n}{\text{minimize}} F(x), \quad \text{subject to} \quad -\infty \leq x \leq \infty,$$

where x is an n element vector.

The user must supply an initial estimate of the solution.

For maximum reliability, it is preferable to provide all first partial derivatives. If all of the derivatives cannot be provided, users are recommended to obtain approximate values (using finite differences) by calling E04XAF from within OBJFUN. This is illustrated in Section 9 of the document for E04DJF.

The method used by E04DGF is described in Section 10.

4 References

[1] Gill P E and Murray W (1979) Conjugate-gradient methods for large-scale nonlinear optimization *Technical Report SOL 79-15* Department of Operations Research, Stanford University

[2] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

5 Parameters

1: N — INTEGER *Input*

On entry: n, the number of variables.

Constraint: N > 0.

2: OBJFUN — SUBROUTINE, supplied by the user. *External Procedure*

On exit: the number of iterations performed.

Its specification is:

```
SUBROUTINE OBJFUN(MODE, N, X, OBJF, OBJGRD, NSTATE, IUSER, USER)
INTEGER           MODE, N, NSTATE, IUSER(*)
real              X(N), OBJF, OBJGRD(N), USER(*)
```

1: MODE — INTEGER *Input/Output*

On entry: MODE indicates which values must be assigned during each call of OBJFUN. Only the following values need be assigned:

if MODE = 0, OBJF;

if MODE = 2, OBJF and OBJGRD.

On exit: MODE may be set to a negative value if the user wishes to terminate the solution to the current problem, and in this case E04DGF will terminate with IFAIL set to MODE.

2: N — INTEGER *Input*

On entry: n, the number of variables.

3: X(N) — ***real*** array *Input*

On entry: x, the vector of variables at which the objective function and its gradient are to be evaluated.

4: OBJF — ***real*** *Output*

On exit: the value of the objective function at x.

5: OBJGRD(N) — ***real*** array *Output*

On exit: if MODE = 2, OBJGRD(i) must contain the value of $\frac{\partial F}{\partial x_i}$ evaluated at x, for $i = 1, 2, \ldots, n$.

6: NSTATE — INTEGER *Input*

On entry: NSTATE will be 1 on the first call of OBJFUN by E04DGF, and 0 for all subsequent calls. Thus, if the user wishes, NSTATE may be tested within OBJFUN in order to perform certain calculations once only. For example, the user may read data or initialise COMMON blocks when NSTATE = 1.

7: IUSER(*) — INTEGER array *User Workspace*

8: USER(*) — ***real*** array *User Workspace*

OBJFUN is called from E04DGF with the parameters IUSER and USER as supplied to E04DGF. The user is free to use arrays IUSER and USER to supply information to OBJFUN as an alternative to using COMMON.

OBJFUN must be declared as EXTERNAL in the (sub)program from which E04DGF is called, and should be tested separately before being used in conjunction with E04DGF. See also the optional parameter **Verify** in Section 11.2. Parameters denoted as *Input* must **not** be changed by this procedure.

3: ITER — INTEGER *Output*

On exit: the total number of iterations performed.

4: OBJF — ***real*** *Output*

On exit: the value of the objective function at the final iterate.

5: OBJGRD(N) — ***real*** array *Output*

On exit: the gradient of the objective function at the final iterate (or its finite difference approximation).

6: X(N) — ***real*** array *Input/Output*

On entry: an initial estimate of the solution.

On exit: the final estimate of the solution.

7: IWORK(N+1) — INTEGER array *Workspace*

8: WORK(13*N) — ***real*** array *Workspace*

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

This array is not used by E04DGF, but is passed directly to routine OBJFUN and may be used to supply information to OBJFUN.

9: IUSER(*) — INTEGER array *User Workspace*

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

This array is not used by E04DGF, but is passed directly to routine OBJFUN and may be used to supply information to OBJFUN.

10: USER(*) — ***real*** array *User Workspace*

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

This array is not used by E04DGF, but is passed directly to routine OBJFUN and may be used to supply information to OBJFUN.

11: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit**.

E04DGF returns with IFAIL = 0 if the following three conditions are satisfied:

(i) $F_{k-1} - F_k < \tau_F(1 + |F_k|)$
(ii) $\|x_{k-1} - x_k\| < \sqrt{\tau_F}(1 + \|x_k\|)$
(iii) $\|g_k\| \leq \sqrt[3]{\tau_F}(1 + |F_k|)$ or $\|g_k\| < \epsilon_A$

where τ_F is the value of the optional parameter **Optimality Tolerance** (default value = $\epsilon^{0.8}$; see Section 11.2) and ϵ_A is the absolute error associated with computing the objective function.

For a full discussion on termination criteria see Gill *et al.* [2] Chapter 8.

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 specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04DGF because the user set MODE < 0 in routine OBJFUN. The value of IFAIL will be the same as the user's setting of MODE.

IFAIL = 1

Not used by this routine.

IFAIL = 2

Not used by this routine.

IFAIL = 3

The limiting number of iterations (as determined by the optional parameter **Iteration Limit** (default value = $\max(50, 5n)$; see Section 11.2) has been reached.

If the algorithm appears to be making satisfactory progress, then **Iteration Limit** may be too small. If so, increase its value and rerun E04DGF. If the algorithm seems to be making little or no progress, then the user should check for incorrect gradients as described below under IFAIL = 7.

IFAIL = 4

The computed upper bound on the step length taken during the linesearch was too small. A rerun with an increased value of the optional parameter **Maximum Step Length** (ρ say) may be successful unless $\rho \geq 10^{20}$ (the default value; see Section 11.2), in which case the current point cannot be improved upon.

IFAIL = 5

Not used by this routine.

IFAIL = 6

The conditions for an acceptable solution (see parameter IFAIL in Section 5) have not all been met, but a lower point could not be found.

If routine OBJFUN computes the objective function and its gradient correctly, then this may occur because an overly stringent accuracy has been requested, i.e., the value of the optional parameter **Optimality Tolerance** (default value = $\epsilon^{0.8}$; see Section 11.2) is too small or if $\alpha_k \simeq 0$. In this case the user should apply the three tests described above under IFAIL = 0 to determine whether or not the final solution is acceptable. For a discussion of attainable accuracy see Gill and Murray [2].

If many iterations have occurred in which essentially no progress has been made or E04DGF has failed to move from the initial point, routine OBJFUN may be incorrect. The user should refer to the comments below under IFAIL = 7 and check the gradients using the optional parameter **Verify** (default value = 0; see Section 11.2). Unfortunately, there may be small errors in the objective gradients that cannot be detected by the verification process. Finite-difference approximations to first derivatives are catastrophically affected by even small inaccuracies.

IFAIL = 7

The user-provided derivatives of the objective function appear to be incorrect.

Large errors were found in the derivatives of the objective function. This value of IFAIL will occur if the verification process indicated that at least one gradient element had no correct figures. The user should refer to the printed output to determine which elements are suspected to be in error.

As a first step, the user should check that the code for the objective values is correct – for example, by computing the function at a point where the correct value is known. However, care should be taken that the chosen point fully tests the evaluation of the function. It is remarkable how often the values $x = 0$ or $x = 1$ are used to test function evaluation procedures, and how often the special properties of these numbers make the test meaningless.

Special care should be used in this test if computation of the objective function involves subsidiary data communicated in COMMON storage. Although the first evaluation of the function may be correct, subsequent calculations may be in error because some of the subsidiary data has accidentally been overwritten.

Errors in programming the function may be quite subtle in that the function value is 'almost' correct. For example, the function may not be accurate to full precision because of the inaccurate calculation of a subsidiary quantity, or the limited accuracy of data upon which the function depends. A common error on machines where numerical calculations are usually performed in double precision is to include even one single precision constant in the calculation of the function; since some compilers do not convert such constants to double precision, half the correct figures may be lost by such a seemingly trivial error.

IFAIL = 8

The gradient ($g = \frac{\partial F}{\partial x}$) at the starting point x_0 is 'too small'. More precisely, the value of $g(x_0)^T g(x_0)$ is less than $\epsilon_m |F(x_0)|$, where ϵ_m is the ***machine precision***.

The problem should be rerun from a different starting point.

IFAIL = 9

An input parameter is invalid.

7 Accuracy

On successful exit (IFAIL = 0) the accuracy of the solution will be as defined by the optional parameter **Optimality Tolerance** (default value = $\epsilon^{0.8}$; see Section 11.2).

8 Further Comments

To evaluate an 'acceptable' set of finite difference intervals using E04XAF requires 2 function evaluations per variable for a well-scaled problem and up to 6 function evaluations per variable for a badly scaled problem.

8.1 Description of Printed Output

This section describes the (default) intermediate printout and final printout produced by E04DGF. The level of printed output can be controlled by the user (see the description of the optional parameter **Print Level** in Section 11.2). Note that the intermediate printout and final printout are produced only if Print Level $\geq$ 10 (the default).

The following line of summary output (< 80 characters) is produced at every iteration. In all cases, the values of the quantities are those in effect *on completion* of the given iteration.

`Itn`	is the iteration count.
`Step`	is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.
`Nfun`	is the cumulated number of evaluations of the objective function needed forthe linesearch. Evaluations needed for the verification of the gradients by finite differences are not included. `Nfun` is printed as a guide to the amount of work required for the linesearch. E04DGF will perform at most 11 function evaluations per iteration.
`Objective`	is the value of the objective function at x_k.
`Norm G`	is the Euclidean norm of the gradient of the objective function at x_k.
`Norm X`	is the Euclidean norm of x_k.
`Norm (X(k-1)-X(k))`	is the Euclidean norm of $x_{k-1} - x_k$.

The following describes the printout for each variable.

`Variable`	gives the name (`Varbl`) and index j, for $j = 1, 2, \ldots, n$ of the variable.
`Value`	is the value of the variable at the final iteration.
`Gradient Value`	is the value of the gradient of the objective function with respect to the jth variable at the final iteration.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

9 Example

To find a minimum of the function

$$F = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1).$$

The initial point is

$$x_0 = (-1.0, 1.0)^T,$$

and $F(x_0) = 1.8394$ (to five figures).

The optimal solution is

$$x^* = (0.5, -1.0)^T,$$

and $F(x^*) = 0$.

The document for E04DJF includes an example program to solve the same problem using some of the optional parameters described in Section 11. The remainder of this document is intended for more advanced users. Section 10 contains a description of the algorithm which may be needed in order to understand Section 11. Section 11 describes the optional parameters which may be set by calls to E04DJF and/or E04DKF.

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.

```
*     E04DGF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=10)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             OBJF
      INTEGER          I, IFAIL, ITER, N
*     .. Local Arrays ..
      real             OBJGRD(NMAX), USER(1), WORK(13*NMAX), X(NMAX)
      INTEGER          IUSER(1), IWORK(NMAX+1)
*     .. External Subroutines ..
      EXTERNAL         E04DGF, OBJFUN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04DGF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.LE.NMAX) THEN
*
*        Read X from data file
*
         READ (NIN,*) (X(I),I=1,N)
*
*        Solve the problem
*
         IFAIL = -1
*
```

```
         CALL E04DGF(N,OBJFUN,ITER,OBJF,OBJGRD,X,IWORK,WORK,IUSER,USER,
     +               IFAIL)
*
      END IF
      STOP
      END
*
      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
*     Routine to evaluate F(x) and its 1st derivatives.
*     .. Parameters ..
      real              ONE, TWO, FOUR
      PARAMETER         (ONE=1.0e0,TWO=2.0e0,FOUR=4.0e0)
*     .. Scalar Arguments ..
      real              OBJF
      INTEGER           MODE, N, NSTATE
*     .. Array Arguments ..
      real              OBJGRD(N), USER(*), X(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      real              EXPX1, X1, X2
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      X1 = X(1)
      X2 = X(2)
*
      EXPX1 = EXP(X1)
      OBJF = EXPX1*(FOUR*X1**2+TWO*X2**2+FOUR*X1*X2+TWO*X2+ONE)
*
      IF (MODE.EQ.2) THEN
         OBJGRD(1) = FOUR*EXPX1*(TWO*X1+X2) + OBJF
         OBJGRD(2) = TWO*EXPX1*(TWO*X2+TWO*X1+ONE)
      END IF
*
      RETURN
      END
```

9.2 Program Data

```
E04DGF Example Program Data
  2                          :Value of N
 -1.0  1.0                   :End of X
```

9.3 Program Results

```
 E04DGF Example Program Results

 *** E04DGF
 *** Start of NAG Library implementation details ***

 Implementation title: Generalised Base Version
            Precision: FORTRAN double precision
         Product Code: FLBAS19D
                 Mark: 19A

 *** End of NAG Library implementation details ***
```

```
Parameters
----------

Variables.............        2

Maximum step length....  1.00E+20      EPS (machine precision)  1.11E-16
Optimality tolerance...  3.26E-12      Linesearch tolerance...  9.00E-01

Est. opt. function val.       None     Function precision.....  4.37E-15
Verify level...........          0

Iteration limit........         50     Print level............         10


Verification of the objective gradients.
----------------------------------------

The objective gradients seem to be ok.

Directional derivative of the objective   -1.47151776E-01
Difference approximation                  -1.47151796E-01
```

Itn	Step	Nfun	Objective	Norm G	Norm X	Norm (X(k-1)-X(k))
0		1	1.839397E+00	8.2E-01	1.4E+00	
1	3.7E-01	3	1.724275E+00	2.8E-01	1.3E+00	3.0E-01
2	1.6E+01	8	6.083488E-02	9.2E-01	9.3E-01	2.2E+00
3	1.6E-03	14	5.367978E-02	1.0E+00	9.6E-01	3.7E-02
4	4.8E-01	16	1.783392E-04	5.8E-02	1.1E+00	1.6E-01
5	1.0E+00	17	1.671122E-05	2.0E-02	1.1E+00	6.7E-03
6	1.0E+00	18	1.101991E-07	1.7E-03	1.1E+00	2.4E-03
7	1.0E+00	19	2.332133E-09	1.8E-04	1.1E+00	1.5E-04
8	1.0E+00	20	9.130924E-11	3.3E-05	1.1E+00	3.0E-05
9	1.0E+00	21	1.085455E-12	4.7E-06	1.1E+00	7.0E-06
10	1.0E+00	22	5.308300E-14	1.2E-06	1.1E+00	6.4E-07

```
Exit from E04DGF after    10 iterations.


Variable          Value      Gradient value
Varbl    1     0.500000          9.1E-07
Varbl    2    -1.000000          8.3E-07

Exit E04DGF - Optimal solution found.

Final objective value =   0.5308300E-13
```

10 Algorithmic Details

This section contains a description of the method used by E04DGF.

E04DGF uses a pre-conditioned conjugate gradient method and is based upon algorithm PLMA as described in Gill and Murray [1] and Gill *et al.* [2] Section 4.8.3.

The algorithm proceeds as follows:

Let x_0 be a given starting point and let k denote the current iteration, starting with $k = 0$. The iteration requires g_k, the gradient vector evaluated at x_k, the kth estimate of the minimum. At each iteration a vector p_k (known as the direction of search) is computed and the new estimate x_{k+1} is given by $x_k + \alpha_k p_k$

where α_k (the step length) minimizes the function $F(x_k + \alpha_k p_k)$ with respect to the scalar α_k. A choice of initial step α_0 is taken as

$$\alpha_0 = \min\{1, 2 \times |F_k - F_{est}|/g_k^T g_k\}$$

where F_{est} is a user-supplied estimate of the function value at the solution. If F_{est} is not specified, the software always chooses the unit step length for α_0. Subsequent step length estimates are computed using cubic interpolation with safeguards.

A quasi-Newton method can be used to compute the search direction p_k by updating the inverse of the approximate Hessian (H_k) and computing

$$p_{k+1} = -H_{k+1} g_{k+1} \tag{1}$$

The updating formula for the approximate inverse is given by

$$H_{k+1} = H_k - \frac{1}{y_k^T s_k}\left(H_k y_k s_k^T + s_k y_k^T H_k\right) + \frac{1}{y_k^T s_k}\left(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}\right) s_k s_k^T \tag{2}$$

where $y_k = g_{k-1} - g_k$ and $s_k = x_{k+1} - x_k = \alpha_k p_k$.

The method used by E04DGF to obtain the search direction is based upon computing p_{k+1} as $-H_{k+1} g_{k+1}$ where H_{k+1} is a matrix obtained by updating the identity matrix with a limited number of quasi-Newton corrections. The storage of an n by n matrix is avoided by storing only the vectors that define the rank two corrections - hence the term 'limited-memory' quasi-Newton method. The precise method depends upon the number of updating vectors stored. For example, the direction obtained with the 'one-step' limited memory update is given by (1) using (2) with H_k equal to the identity matrix, viz.

$$p_{k+1} = -g_{k+1} + \frac{1}{y_k^T s_k}\left(s_k^T g_{k+1} y_k + y_k^T g_{k+1} s_k\right) - \frac{s_k^T g_{k+1}}{y_k^T s_k}\left(1 + \frac{y_k^T y_k}{y_k^T s_k}\right) s_k.$$

E04DGF uses a two-step method described in detail in Gill and Murray [1] in which restarts and pre-conditioning are incorporated. Using a limited-memory quasi-Newton formula, such as the one above, guarantees p_{k+1} to be a descent direction if all the inner products $y_k^T s_k$ are positive for all vectors y_k and s_k used in the updating formula.

11 Optional Parameters

Several optional parameters in E04DGF define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of E04DGF these optional parameters have associated *default values* that are appropriate for most problems. Therefore, the user need only specify those optional parameters whose values are to be different from their default values. The remainder of this section can be skipped by users who wish to use the default values for *all* optional parameters. A complete list of optional parameters and their default values is given in Section 11.1.
Optional parameters may be specified by calling one, or both, of the routines E04DJF and E04DKF prior to a call to E04DGF. E04DJF reads options from an external options file, with **Begin** and **End** as the first and last lines respectively and each intermediate line defining a single optional parameter. For example,

```
Begin
  Print Level = 1
End
```

The call

```
CALL E04DJF (IOPTNS, INFORM)
```

can then be used to read the file on unit IOPTNS. INFORM will be zero on successful exit. E04DJF should be consulted for a full description of this method of supplying optional parameters.

E04DKF can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
CALL E04DKF ('Print Level = 1')
```

E04DKF should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by the user are set to their default values. Optional parameters specified by the user are unaltered by E04DGF (unless they define invalid values) and so remain in effect for subsequent calls unless altered by the user.

11.1 Optional Parameter Checklist and Default Values

For easy reference, the following list shows all the valid keywords and their default values. The symbol ϵ represents the ***machine precision*** (see X02AJF).

Optional Parameters	**Default Values**
Defaults	
Estimated optimal function value	
Function precision	$\epsilon^{0.9}$
Iteration limit	$\max(50, 5n)$
Linesearch tolerance	0.9
List/Nolist	**List**
Maximum step length	10^{20}
Optimality tolerance	$\epsilon^{0.8}$
Print level	10
Start objective check at variable	1
Stop objective check at variable	n
Verify level	0

11.2 Description of the Optional Parameters

The following list (in alphabetical order) gives the valid options. For each option, we give the keyword, any essential optional qualifiers, the default value, and the definition. The minimum abbreviation of each keyword is underlined. If no characters of an optional qualifier are underlined, the qualifier may be omitted. The letter *a* denotes a phrase (character string) that qualifies an option. The letters i and r denote INTEGER and ***real*** values required with certain options. The number ϵ is a generic notation for ***machine precision*** (see X02AJF), and ϵ_R denotes the relative precision of the objective function.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Estimated Optimal Function Value r

This value of r specifies the user-supplied guess of the optimum objective function value F_{est}. This value is used by E04DGF to calculate an initial step length α_0 (see Section 10). If the value of r is not specified by the user (the default), then this has the effect of setting α_0 to unity. It should be noted that for badly scaled functions a unit step along the steepest descent direction will often compute the objective function at very large values of x.

Function Precision r Default $= \epsilon^{0.9}$

The parameter defines ϵ_R, which is intended to be a measure of the accuracy with which the problem function $F(x)$ can be computed. If $r < \epsilon$ or $r \geq 1$, the default value is used.

The value of ϵ_R should reflect the relative precision of $1 + |F(x)|$; i.e., ϵ_R acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 10^{-6}. In contrast, if $F(x)$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 10^{-10}. The choice of ϵ_R can be quite complicated for badly scaled problems; see Chapter 8 of Gill and Murray [2], for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_R should be large enough so that E04DGF will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Iteration Limit i Default = $\max(50, 5n)$

Iters

Itns

The value of i specifies the maximum number of iterations allowed before termination. If $i < 0$, the default value is used.

Problems whose Hessian matrices at the solution contain sets of clustered eigenvalues are likely to be minimized in significantly fewer than n iterations. Problems without this property may require anything between n and $5n$ iterations, with approximately $2n$ iterations being a common figure for moderately difficult problems.

Linesearch Tolerance r Default = 0.9

The value r ($0 \leq r < 1$) controls the accuracy with which the step α taken during each iteration approximates a minimum of the function along the search direction (the smaller the value of r, the more accurate the linesearch). The default value $r = 0.9$ requests an inaccurate search, and is appropriate for most problems. A more accurate search may be appropriate when it is desirable to reduce the number of iterations - for example, if the objective function is cheap to evaluate. If $r < 0$ or $r \geq 1$, the default value is used.

List Default = List

Nolist

Normally each optional parameter specification is printed as it is supplied. **Nolist** may be used to suppress the printing and **List** may be used to restore printing.

Maximum Step Length r Default = 10^{20}

If $r > 0$, the maximum allowable step length for the linesearch is taken as $\min\left(\frac{1}{\text{X02AMF}()}, \frac{r}{\|p_k\|}\right)$. If $r \leq 0$, the default value is used.

Optimality Tolerance r Default = $\epsilon_R^{0.8}$

The parameter r ($\epsilon_R \leq r < 1$) specifies the accuracy to which the user wishes the final iterate to approximate a solution of the problem. Broadly speaking, r indicates the number of correct figures desired in the objective function at the solution. For example, if r is 10^{-6} and E04DGF terminates successfully, the final value of F should have approximately six correct figures. E04DGF will terminate successfully if the iterative sequence of x-values is judged to have converged and the final point satisfies the termination criteria (as described under the parameter IFAIL in Section 5, where τ_F represents **Optimality Tolerance**). If $r < \epsilon_R$ or $r \geq 1$, the default value is used.

Print Level i Default = 10

The value i controls the amount of printout produced by E04DGF, as indicated below. A detailed description of the printout is given in Section 8.1 (summary output at each iteration and the final solution).

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output (< 80 characters; see Section 8.1) for each iteration (no printout of the final solution).
10	The final solution and one line of summary output for each iteration.

Start Objective Check at Variable i_1 Default = 1

Stop Objective Check at Variable i_2 Default = n

These keywords take effect only if **Verify Level** > 0 (see below). They may be used to control the verification of gradient elements computed by routine OBJFUN. For example, if the first 30 elements of the objective gradient appeared to be correct in an earlier run, so that only element 31 remains questionable, it is reasonable to specify **Start Objective Check at Variable** 31. If the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, the above choice would also be appropriate.

If $i_1 \leq 0$ or $i_1 > \max(1, \min(n, i_2))$, the default value is used. If $i_2 \leq 0$ or $i_2 > n$, the default value is used.

Verify Level	i	Default = 0
Verify	No	
Verify Level	-1	
Verify Level	0	
Verify		
Verify	Yes	
Verify Objective Gradients		
Verify Gradients		
Verify Level	1	

These keywords refer to finite-difference checks on the gradient elements computed by the user-provided routine OBJFUN. It is possible to specify **Verify Level** in several ways, as indicated above. For example, the objective gradient will be verified if **Verify**, **Verify Yes**, **Verify Gradients**, **Verify Objective Gradients** or **Verify Level** = 1 is specified. If $i = -1$, then no checking will be performed. If $i = 0$ or 1, then the objective gradient will be verified at the user-supplied initial estimate of the solution. If $i = 0$ only a 'cheap' test will be performed, requiring one call to OBJFUN. If $i = 1$, a more reliable (but more expensive) check will be made on individual gradient elements, within the ranges specified by the **Start** and **Stop** keywords as described above. A result of the form `OK` or `BAD`? is printed by E04DGF to indicate whether or not each element appears to be correct. If $i < -1$ or $i > 1$, the default value is used.

E04DJF – NAG Fortran Library Routine Document

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

To supply optional parameters to E04DGF from an external file.

2 Specification

```
SUBROUTINE E04DJF(IOPTNS, INFORM)
INTEGER           IOPTNS, INFORM
```

3 Description

E04DJF may be used to supply values for optional parameters to E04DGF. E04DJF reads an external file and each line of the file defines a single optional parameter. It is only necessary to supply values for those parameters whose values are to be different from their default values.

Each optional parameter is defined by a single character string of up to 72 characters, consisting of one or more items. The items associated with a given option must be separated by spaces, or equal signs [=]. Alphabetic characters may be upper or lower case. The string

```
Print level = 1
```

is an example of a string used to set an optional parameter. For each option the string contains one or more of the following items:

(a) A mandatory keyword.
(b) A phrase that qualifies the keyword.
(c) A number that specifies an INTEGER or ***real*** value. Such numbers may be up to 16 contiguous characters in Fortran 77's I, F, E or D formats, terminated by a space if this is not the last item on the line.

Blank strings and comments are ignored. A comment begins with an asterisk (*) and all subsequent characters in the string are regarded as part of the comment.

The file containing the options must start with **begin** and must finish with **end**. An example of a valid options file is:

```
Begin * Example options file
  Print level = 5
End
```

Normally each line of the file is printed as it is read, on the current advisory message unit (see X04ABF), but printing may be suppressed using the keyword **nolist**. To suppress printing of **begin**, **nolist** must be the first option supplied as in the file:

```
Begin
  Nolist
  Print level = 5
End
```

Printing will automatically be turned on again after a call to E04DGF and may be turned on again at any time by the user by using the keyword **list**.

Optional parameter settings are preserved following a call to E04DGF, and so the keyword **defaults** is provided to allow the user to reset all the optional parameters to their default values prior to a subsequent call to E04DGF.

A complete list of optional parameters, their abbreviations, synonyms and default values is given in Section 11 of the document for E04DGF.

4 References

None.

5 Parameters

1: IOPTNS — INTEGER *Input*

On entry: the unit number of the options file to be read.

Constraint: $0 \leq \text{IOPTNS} \leq 99$.

2: INFORM — INTEGER *Output*

On exit: contains zero if an options file with the correct structure has been read and a value > 0 otherwise, as indicated below.

INFORM = 1

IOPTNS is not in the range $[0, 99]$.

INFORM = 2

begin was found, but end-of-file was found before **end** was found.

INFORM = 3

end-of-file was found before **begin** was found.

6 Error Indicators and Warnings

If a line is not recognized as a valid option, then a warning message is output on the current advisory message unit (see X04ABF).

7 Accuracy

Not applicable.

8 Further Comments

E04DKF may also be used to supply optional parameters to E04DGF.

9 Example

This example solves the same problem as the example for E04DGF, but in addition illustrates the use of E04DJF and E04DKF to set optional parameters for E04DGF.

In this example the options file read by E04DJF is appended to the data file for the program (see Section 9.1). It would usually be more convenient in practice to keep the data file and the options file separate.

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.

```
*     E04DJF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=10)
```

```
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             OBJF
      INTEGER          I, IFAIL, INFORM, ITER, N
*     .. Local Arrays ..
      real             OBJGRD(NMAX), USER(4*NMAX), WORK(13*NMAX),
     +                 X(NMAX)
      INTEGER          IUSER(NMAX), IWORK(NMAX+1)
*     .. External Subroutines ..
      EXTERNAL         E04DGF, E04DJF, E04DKF, OBJFN1, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04DJF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.LE.NMAX) THEN
*
*        Read X from data file
*
         READ (NIN,*) (X(I),I=1,N)
*
*        Set two options using E04DKF
*
         CALL E04DKF(' Verify Level = -1 ')
*
         CALL E04DKF(' Maximum Step Length = 100.0 ')
*
*        Set the unit number for advisory messages to NOUT
*
         CALL X04ABF(1,NOUT)
*
*        Read the options file for the remaining options
*
         CALL E04DJF(NIN,INFORM)
*
         IF (INFORM.NE.0) THEN
            WRITE (NOUT,99999) 'E04DJF terminated with INFORM = ',
     +        INFORM
            STOP
         END IF
*
*        Solve the problem
*
         IFAIL = -1
*
         CALL E04DGF(N,OBJFN1,ITER,OBJF,OBJGRD,X,IWORK,WORK,IUSER,USER,
     +               IFAIL)
*
      END IF
      STOP
*
99999 FORMAT (1X,A,I3)
      END
*
```

```
      SUBROUTINE OBJFN1(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
*     Routine to evaluate F(x) and approximate its 1st derivatives
*     .. Scalar Arguments ..
      real              OBJF
      INTEGER           MODE, N, NSTATE
*     .. Array Arguments ..
      real              OBJGRD(N), USER(*), X(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      real              EPSRF
      INTEGER           I, IFAIL, IMODE, IWARN, LHES, MSGLVL
*     .. Local Arrays ..
      real              USE(1)
      INTEGER           IUSE(1)
*     .. External Subroutines ..
      EXTERNAL          E04XAF, OBJFN2
*     .. Executable Statements ..
      IF (MODE.EQ.0) THEN
*        Evaluate F(x) only
         CALL OBJFN2(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSE,USE)
*
      ELSE IF (MODE.EQ.2) THEN
*        Evaluate F(x) and approximate its 1st derivatives
         MSGLVL = 0
         EPSRF = 0.0e0
         IMODE = 0
         LHES = N
         DO 20 I = 1, N
            USER(I) = 0.0e0
   20    CONTINUE
         IFAIL = 1
*
         CALL E04XAF(MSGLVL,N,EPSRF,X,IMODE,OBJFN2,LHES,USER(1),OBJF,
     +               OBJGRD,USER(N+1),USER(2*N+1),IWARN,USER(3*N+1),
     +               IUSE,USE,IUSER,IFAIL)
*
      END IF
*
      RETURN
      END
      SUBROUTINE OBJFN2(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSE,USE)
*     Routine to evaluate F(x)
*     .. Scalar Arguments ..
      real              OBJF
      INTEGER           MODE, N, NSTATE
*     .. Array Arguments ..
      real              OBJGRD(N), USE(*), X(N)
      INTEGER           IUSE(*)
*     .. Local Scalars ..
      real              X1, X2
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      X1 = X(1)
      X2 = X(2)
*
      OBJF = EXP(X1)*(4.0e0*X1**2+2.0e0*X2**2+4.0e0*X1*X2+2.0e0*X2+
```

```
     +       1.0e0)
*
      RETURN
      END
```

9.2 Program Data

```
E04DJF Example Program Data
  2                                        :Value of N
 -1.0  1.0                                 :End of X
Begin   Example options file for  E04DJF
   Iteration Limit  =  25                  *  (Default =  50)
   Print Level      =   1                  *  (Default =  10)
End
```

9.3 Program Results

```
 E04DJF Example Program Results


 Calls to E04DKF
 ---------------

       Verify Level = -1
       Maximum Step Length = 100.0


 OPTIONS file
 ------------

      Begin   Example options file for  E04DJF
         Iteration Limit  =  25                  *  (Default =  50)
         Print Level      =   1                  *  (Default =  10)
      End

 *** E04DGF
 *** Start of NAG Library implementation details ***

 Implementation title: Generalised Base Version
            Precision: FORTRAN double precision
         Product Code: FLBAS18D
                 Mark: 17A

 *** End of NAG Library implementation details ***

 Parameters
 ----------

 Variables.............         2

 Maximum step length....  1.00E+02       EPS (machine precision)  1.11E-16
 Optimality tolerance...  3.26E-12       Linesearch tolerance...  9.00E-01

 Est. opt. function val.      None       Function precision.....  4.37E-15
 Verify level...........        -1

 Iteration limit........        25       Print level............         1
```

```
Exit from E04DGF after    10 iterations.


Variable           Value      Gradient value
Varbl    1      0.500000           9.1E-07
Varbl    2     -1.000000           8.3E-07

Exit E04DGF - Optimal solution found.

Final objective value =   0.5308300E-13
```

E04DKF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

To supply individual optional parameters to E04DGF.

2. Specification

```
SUBROUTINE E04DKF (STRING)
CHARACTER*(*) STRING
```

3. Description

E04DKF may be used to supply values for optional parameters to E04DGF. It is only necessary to call E04DKF for those parameters whose values are to be different from their default values. One call to E04DKF sets one parameter value.

Each optional parameter is defined by a single character string of up to 72 characters, consisting of one or more items. The items associated with a given option must be separated by spaces, or equal signs [=]. Alphabetic characters may be upper or lower case. The string

```
Print Level = 1
```

is an example of a string used to set an optional parameter. For each option the string contains one or more of the following items:

(a) A mandatory keyword.

(b) A phrase that qualifies the keyword.

(c) A number that specifies an INTEGER or ***real*** value. Such numbers may be up to 16 contiguous characters in Fortran 77's I, F, E or D formats, terminated by a space if this is not the last item on the line.

Blank strings and comments are ignored. A comment begins with an asterisk (*) and all subsequent characters in the string are regarded as part of the comment.

Normally, each user-specified option is printed as it is defined, on the current advisory message unit (see X04ABF), but this printing may be suppressed using the keyword **nolist**. Thus the statement

```
CALL E04DKF ('Nolist')
```

suppresses printing of this and subsequent options. Printing will automatically be turned on again after a call to E04DGF, and may be turned on again at any time by the user, by using the keyword **list**.

Optional parameter settings are preserved following a call to E04DGF, and so the keyword **defaults** is provided to allow the user to reset all the optional parameters to their default values by the statement,

```
CALL E04DKF ('Defaults')
```

prior to a subsequent call to E04DGF.

A complete list of optional parameters, their abbreviations, synonyms and default values is given in Section 11 of the routine document for E04DGF.

4. References

None.

5. Parameters

1: STRING – CHARACTER*(*). *Input*

On entry: a single valid option string (as described in Section 3 above and in Section 11 of the routine document for E04DGF).

6. Error Indicators and Warnings

If the parameter STRING is not recognized as a valid option string, then a warning message is output on the current advisory message unit (see X04ABF).

7. Accuracy

Not applicable.

8. Further Comments

E04DJF may also be used to supply optional parameters to E04DGF.

9. Example

See the example for E04DJF.

E04FCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04FCF is a comprehensive algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). No 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 E04FCF (M, N, LSQFUN, LSQMON, IPRINT, MAXCAL, ETA, XTOL,
     1                   STEPMX, X, FSUMSQ, FVEC, FJAC, LJ, S, V, LV,
     2                   NITER, NF, IW, LIW, W, LW, IFAIL)
      INTEGER           M, N, IPRINT, MAXCAL, LJ, LV, NITER, NF,
     1                  IW(LIW), LIW, LW, IFAIL
      real              ETA, XTOL, STEPMX, X(N), FSUMSQ, FVEC(M),
     1                  FJAC(LJ,N), S(N), V(LV,N), W(LW)
      EXTERNAL          LSQFUN, LSQMON
```

3. Description

This routine is essentially identical to the subroutine LSQNDN in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine LSQFUN to calculate the values of the $f_i(x)$ at any point x.

From a starting point $x^{(1)}$ supplied by the user, the routine generates a sequence of points $x^{(2)}, x^{(3)}, \ldots$, which is intended to converge to a local minimum of $F(x)$. The sequence of points is given by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$$

where the vector $p^{(k)}$ is a direction of search, and $\alpha^{(k)}$ is chosen such that $F(x^{(k)} + \alpha^{(k)} p^{(k)})$ is approximately a minimum with respect to $\alpha^{(k)}$.

The vector $p^{(k)}$ used depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced, then $p^{(k)}$ is an approximation to the Gauss-Newton direction; otherwise additional function evaluations are made so as to enable $p^{(k)}$ to be a more accurate approximation to the Newton direction.

The method is designed to ensure that steady progress is made whatever the starting point, and to have the rapid ultimate convergence of Newton's method.

4. References

[1] GILL, P.E. and MURRAY, W.
Algorithms for the Solution of the Nonlinear Least-squares Problem.
SIAM J. Numer. Anal., 15, pp. 977-992, 1978.

5. Parameters

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \text{N} \leq \text{M}$.

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

LSQFUN must calculate the vector of values $f_i(x)$ at any point x. (However, if the user does not wish to calculate the residuals at a particular x, there is the option of setting a parameter to cause E04FCF to terminate immediately.)

Its specification is:

```
SUBROUTINE LSQFUN(IFLAG, M, N, XC, FVECC, IW, LIW, W, LW)
INTEGER    IFLAG, M, N, IW(LIW), LIW, LW
real       XC(N), FVECC(M), W(LW)
```

1: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG has a non-negative value.

On exit: if LSQFUN resets IFLAG to some negative number, E04FCF will terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: M – INTEGER. *Input*

3: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

4: XC(N) – ***real*** array. *Input*

On entry: the point x at which the values of the f_i are required.

5: FVECC(M) – ***real*** array. *Output*

On exit: unless IFLAG is reset to a negative number, on exit FVECC(i) must contain the value of f_i at the point x, for $i = 1,2,...,m$.

6: IW(LIW) – INTEGER array. *Workspace*

7: LIW – INTEGER. *Input*

8: W(LW) – ***real*** array. *Workspace*

9: LW – INTEGER. *Input*

LSQFUN is called with these parameters as in the call to E04FCF, so users can pass quantities to LSQFUN from the (sub)program which calls E04FCF by using partitions of IW and W beyond those used as workspace by E04FCF. However, because of the danger of mistakes in partitioning, it is recommended that this facility be used very selectively, e.g. for stable applications packages which need to pass their own variable dimension workspace to LSQFUN. It is **recommended** that the normal method for passing information from the user's (sub)program to LSQFUN should be via COMMON. In any case, users **must not change** LIW, LW or the elements of IW and W used as workspace by E04FCF.

Note: LSQFUN should be tested separately before being used in conjunction with E04FCF.

LSQFUN must be declared as EXTERNAL in the (sub)program from which E04FCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

If IPRINT ≥ 0, the user must supply a subroutine LSQMON which is suitable for monitoring the minimization process. LSQMON must not change the values of any of its parameters.

If IPRINT < 0, the dummy routine E04FDZ can be used as LSQMON. (In some implementations the name of this routine is FDZE04; refer to the Users' Note for your implementation.)

Its specification is:

```
SUBROUTINE LSQMON(M, N, XC, FVECC, FJACC, LJC, S, IGRADE, NITER, NF,
1          IW, LIW, W, LW)
INTEGER    M, N, LJC, IGRADE, NITER, NF, IW(LIW), LIW, LW
real       XC(N), FVECC(M), FJACC(LJC,N), S(N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

4: FVECC(M) – ***real*** array. *Input*

On entry: the values of the residuals f_i at the current point x.

5: FJACC(LJC,N) – ***real*** array. *Input*

On entry: FJACC(i,j) contains the value of $\frac{\partial f_i}{\partial x_j}$, at the current point x, for $i = 1,2,...,m$; $j = 1,2,...,n$.

6: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

7: S(N) – ***real*** array. *Input*

On entry: the singular values of the current approximation to the Jacobian matrix. Thus S may be useful as information about the structure of the user's problem.

8: IGRADE – INTEGER. *Input*

On entry: E04FCF estimates the dimension of the subspace for which the Jacobian matrix can be used as a valid approximation to the curvature (see Gill and Murray [1]). This estimate is called the grade of the Jacobian matrix, and IGRADE gives its current value.

9: NITER – INTEGER. *Input*

On entry: the number of iterations which have been performed in E04FCF.

10: NF – INTEGER. *Input*

On entry: the number of times that LSQFUN has been called so far. (However, for intermediate calls of LSQMON, NF is calculated on the assumption that the latest linear search has been successful. If this is not the case, then the n evaluations allowed for approximating the Jacobian at the new point will not in fact have been made. NF will be accurate at the final call of LSQMON.)

11: IW(LIW) – INTEGER array. *Workspace*

12: LIW – INTEGER. *Input*

13: W(LW) – ***real*** array. *Workspace*

14: LW – INTEGER. *Input*

These parameters correspond to the parameters IW, LIW, W, LW of E04FCF. They are included in LSQMON's parameter list primarily for when E04FCF is called by other Library routines.

Note: the user should normally print the sum of squares of residuals, so as to be able to examine the sequence of values of $F(x)$ mentioned in Section 7. It is usually helpful to print XC, the estimated gradient of the sum of squares, NITER and NF.

LSQMON must be declared as EXTERNAL in the (sub)program from which E04FCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: IPRINT – INTEGER. *Input*

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

If IPRINT > 0, LSQMON is called once every IPRINT iterations and just before exit from E04FCF.

If IPRINT = 0, LSQMON is just called at the final point.

If IPRINT < 0, LSQMON 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 limit set by the user on the number of times that LSQFUN may be called by E04FCF. There will be an error exit (see Section 6) after MAXCAL calls of LSQFUN.

Suggested value: MAXCAL = 400×n.

Constraint: MAXCAL ≥ 1.

7: ETA – ***real***. *Input*

Every iteration of E04FCF involves a linear minimization i.e. minimization of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$.

On entry: specifies how accurately the linear minimizations are to be performed. The minimum with respect to $\alpha^{(k)}$ 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 performed by E04FCF, they will increase the number of calls of LSQFUN made each iteration. On balance it is usually more efficient to perform a low accuracy minimization.

Suggested value: ETA = 0.5 (ETA = 0.0 if N = 1).

Constraint: 0.0 ≤ ETA < 1.0.

8: XTOL – ***real***. *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 prior to a normal exit, is such that

$\|x_{sol}-x_{true}\| < \text{XTOL}\times(1.0+\|x_{true}\|)$,

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 = 1.0E−5, then x_{sol} is usually accurate to about 5 decimal places. (For further details see Section 7.)

If $F(x)$ and the variables are scaled roughly as described in Section 8 and ε is the ***machine precision***, then a setting of order XTOL = $\sqrt{\varepsilon}$ will usually be appropriate. If XTOL is set to 0.0 or some positive value less than 10ε, E04FCF will use 10ε instead of XTOL, since 10ε is probably the smallest reasonable setting.

Constraint: XTOL ≥ 0.0.

9: STEPMX – ***real***. *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency, a slight overestimate is preferable.) E04FCF will ensure that, for each iteration,

$$\sum_{j=1}^{n} (x_j^{(k)}-x_j^{(k-1)})^2 \leq (\text{STEPMX})^2$$

where k is the iteration number. Thus, if the problem has more than one solution, E04FCF is most likely to find the one nearest to the starting point. On difficult problems, a realistic choice can prevent the sequence $x^{(k)}$ entering a region where the problem is ill-behaved and can help avoid overflow in the evaluation of $F(x)$. However, an underestimate of STEPMX can lead to inefficiency.

Suggested value: STEPMX = 100000.0.

Constraint: STEPMX ≥ XTOL.

10: 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 $j = 1,2,...,n$.

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.

11: FSUMSQ – ***real***. *Output*

On exit: the value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in X.

12: FVEC(M) – ***real*** array. *Output*

On exit: the value of the residual $f_i(x)$ at the final point given in X, for $i = 1,2,...,m$.

13: FJAC(LJ,N) – ***real*** array. *Output*

On exit: the estimate of the first derivative $\frac{\partial f_i}{\partial x_j}$ at the final point given in X, for $i = 1,2,...,m$; $j = 1,2,...,n$.

14: LJ – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which E04FCF is called.

Constraint: LJ ≥ M.

15: S(N) – ***real*** array. *Output*

On exit: the singular values of the estimated Jacobian matrix at the final point. Thus S may be useful as information about the structure of the user's problem.

16: V(LV,N) – ***real*** array. *Output*

On exit: the matrix V associated with the singular value decomposition

$$J = USV^T$$

of the estimated Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalised eigenvectors of J^TJ.

17: LV – INTEGER. *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which E04FCF is called.

Constraint: LV ≥ N.

18: NITER – INTEGER. *Output*

On exit: the number of iterations which have been performed in E04FCF.

19: NF – INTEGER. *Output*

On exit: the number of times that the residuals have been evaluated (i.e. number of calls of LSQFUN).

20: IW(LIW) – INTEGER array. *Workspace*
21: LIW – INTEGER. *Input*

On entry: the length of IW as declared in the (sub)program from which E04FCF is called.

Constraint: LIW ≥ 1.

22: W(LW) – ***real*** array. *Workspace*
23: LW – INTEGER. *Input*

On entry: the length of W as declared in the (sub)program from which E04FCF is called.

Constraints: LW ≥ 6×N + M×N + 2×M + N×(N−1)/2, if N > 1
LW ≥ 7 + 3×M, if N = 1.

24: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL < 0

A negative value of IFAIL indicates an exit from E04FCF because the user has set IFLAG negative in LSQFUN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N < 1,
or M < N,
or MAXCAL < 1,
or ETA < 0.0,
or ETA ≥ 1.0,
or XTOL < 0.0,
or STEPMX < XTOL,
or LJ < M,
or LV < N,
or LIW < 1,
or LW < 6×N + M×N + 2×M + N×(N−1)/2, when N > 1,
or LW < 7 + 3×M, when N = 1.

When this exit occurs, no values will have been assigned to FSUMSQ, or to the elements of FVEC, FJAC, S or V.

IFAIL = 2

There have been MAXCAL calls of LSQFUN. If steady reductions in the sum of squares, $F(x)$, 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 satisfied, but a lower point could not be found. This could be because XTOL has been set so small that rounding errors in the evaluation of the residuals make attainment of the convergence conditions impossible.

IFAIL = 4

The method for computing the singular value decomposition of the estimated Jacobian matrix has failed to converge in a reasonable number of sub-iterations. It may be worth applying E04FCF again starting with an initial approximation which is not too close to the point at which the failure occurred.

The values IFAIL = 2, 3 and 4 may also be caused by mistakes in LSQFUN, 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 E04FCF when (B1, B2 and B3) or B4 or B5 hold, where

$$\text{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\varepsilon)\times(1.0+\|x^{(k)}\|)$$

$$\text{B2} \equiv |F^{(k)}-F^{(k-1)}| < (\text{XTOL}+\varepsilon)^2\times(1.0+F^{(k)})$$

$$\text{B3} \equiv \|g^{(k)}\| < (\varepsilon^{1/3}+\text{XTOL})\times(1.0+F^{(k)})$$

$$\text{B4} \equiv F^{(k)} < \varepsilon^2$$

$$\text{B5} \equiv \|g^{(k)}\| < (\varepsilon\times\sqrt{F^{(k)}})^{\frac{1}{2}}$$

and where $\|.\|$ and ε are as defined in Section 5, and $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of estimated first derivatives at $x^{(k)}$. If IFAIL = 0 then the vector in X on exit, x_{sol}, is almost certainly an estimate of x_{true}, the position of the minimum to the accuracy specified by XTOL.

If IFAIL = 3, then x_{sol} may still be a good estimate of x_{true}, but to verify this the user should make the following checks. If

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

(b) $g(x_{sol})^T g(x_{sol}) < 10\varepsilon$,

where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum. When (b) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. The values of $F(x^{(k)})$ can be calculated in LSQMON, and the vector $g(x_{sol})$ can be calculated from the contents of FVEC and FJAC on exit from E04FCF.

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

8. Further Comments

The number of iterations required depends on the number of variables, the number of residuals, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04FCF varies, but for $m \gg n$ is approximately $n\times m^2 + O(n^3)$. In addition, each iteration makes at least $n + 1$ calls of LSQFUN. So, unless the residuals can be evaluated very quickly, the run time will be dominated by the time spent in LSQFUN.

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 of $F(x)$ at the solution is well-conditioned. 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 E04FCF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in the arrays S and V. See E04YCF for further details.

9. Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04FCF Example Program Text.
*     Mark 15 Revised.  NAG Copyright 1991.
*     .. Parameters ..
      INTEGER           N, M, NT, LIW, LV, LJ, LW
      PARAMETER         (N=3,M=15,NT=3,LIW=1,LV=N,LJ=M,
     +                  LW=6*N+M*N+2*M+N*(N-1)/2)
      INTEGER           NIN, NOUT
      PARAMETER         (NIN=5,NOUT=6)
*     .. Arrays in Common ..
      real              T(M,NT), Y(M)
*     .. Local Scalars ..
      real              ETA, FSUMSQ, STEPMX, XTOL
      INTEGER           I, IFAIL, IPRINT, J, MAXCAL, NF, NITER
*     .. Local Arrays ..
      real              FJAC(M,N), FVEC(M), G(N), S(N), V(LV,N), W(LW),
     +                  X(N)
      INTEGER           IW(LIW)
*     .. External Functions ..
      real              X02AJF
      EXTERNAL          X02AJF
*     .. External Subroutines ..
      EXTERNAL          E04FCF, LSQFUN, LSQGRD, LSQMON
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Common blocks ..
      COMMON            Y, T
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04FCF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
      DO 20 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
   20 CONTINUE
*     * Set IPRINT to 1 to obtain output from LSQMON at each iteration *
      IPRINT = -1
      MAXCAL = 400*N
      ETA = 0.5e0
      XTOL = 10.0e0*SQRT(X02AJF())
*     We estimate that the minimum will be within 10 units of the
*     starting point
      STEPMX = 10.0e0
*     Set up the starting point
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
      IFAIL = 1
*
      CALL E04FCF(M,N,LSQFUN,LSQMON,IPRINT,MAXCAL,ETA,XTOL,STEPMX,X,
     +            FSUMSQ,FVEC,FJAC,LJ,S,V,LV,NITER,NF,IW,LIW,W,LW,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
         CALL LSQGRD(M,N,FVEC,FJAC,LJ,G)
         WRITE (NOUT,99997) 'The estimated gradient is', (G(J),J=1,N)
         WRITE (NOUT,*) '                                (machine dependent)'
         WRITE (NOUT,*) 'and the residuals are'
         WRITE (NOUT,99996) (FVEC(I),I=1,M)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
99997 FORMAT (1X,A,1P,3e12.3)
99996 FORMAT (1X,1P,e9.1)
      END
*
      SUBROUTINE LSQFUN(IFLAG,M,N,XC,FVECC,IW,LIW,W,LW)
*     Routine to evaluate the residuals
*     .. Parameters ..
      INTEGER           MDEC, NT
      PARAMETER         (MDEC=15,NT=3)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LIW, LW, M, N
*     .. Array Arguments ..
      real              FVECC(M), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Arrays in Common ..
      real              T(MDEC,NT), Y(MDEC)
*     .. Local Scalars ..
      INTEGER           I
*     .. Common blocks ..
      COMMON            Y, T
*     .. Executable Statements ..
      DO 20 I = 1, M
         FVECC(I) = XC(1) + T(I,1)/(XC(2)*T(I,2)+XC(3)*T(I,3)) - Y(I)
   20 CONTINUE
      RETURN
      END
*
```

```
      SUBROUTINE LSQMON(M,N,XC,FVECC,FJACC,LJC,S,IGRADE,NITER,NF,IW,LIW,
     +                  W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER           NDEC
      PARAMETER         (NDEC=3)
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      INTEGER           IGRADE, LIW, LJC, LW, M, N, NF, NITER
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), S(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Local Scalars ..
      real              FSUMSQ, GTG
      INTEGER           J
*     .. Local Arrays ..
      real              G(NDEC)
*     .. External Functions ..
      real              F06EAF
      EXTERNAL          F06EAF
*     .. External Subroutines ..
      EXTERNAL          LSQGRD
*     .. Executable Statements ..
      FSUMSQ = F06EAF(M,FVECC,1,FVECC,1)
      CALL LSQGRD(M,N,FVECC,FJACC,LJC,G)
      GTG = F06EAF(N,G,1,G,1)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '  Itn      F evals        SUMSQ             GTG         Grade'
      WRITE (NOUT,99999) NITER, NF, FSUMSQ, GTG, IGRADE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '       X                    G           Singular values'
      DO 20 J = 1, N
         WRITE (NOUT,99998) XC(J), G(J), S(J)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,I4,6X,I5,6X,1P,e13.5,6X,1P,e9.1,6X,I3)
99998 FORMAT (1X,1P,e13.5,10X,1P,e9.1,10X,1P,e9.1)
      END
*
      SUBROUTINE LSQGRD(M,N,FVECC,FJACC,LJC,G)
*     Routine to evaluate gradient of the sum of squares
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), G(N)
*     .. Local Scalars ..
      real              SUM
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 J = 1, N
         SUM = 0.0e0
         DO 20 I = 1, M
            SUM = SUM + FJACC(I,J)*FVECC(I)
   20    CONTINUE
         G(J) = SUM + SUM
   40 CONTINUE
      RETURN
      END
```

9.2. Program Data

```
E04FCF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3. Program Results

```
E04FCF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
The estimated gradient is  -6.181E-12  -3.170E-11   3.409E-11
                           (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

With IPRINT set to 1 in the example program, intermediate results similar to those below are obtaned from LSQMON:

```
E04FCF Example Program Results

  Itn       F evals        SUMSQ             GTG         Grade
   0          4        1.02104E+01         1.0E+03         3

       X                    G            Singular values
  5.00000E-01            2.1E+01             5.0E+00
  1.00000E+00           -1.7E+01             2.6E+00
  1.50000E+00           -1.6E+01             9.6E-02

  Itn       F evals        SUMSQ             GTG         Grade
   1          8        1.98730E-01         8.1E+00         3

       X                    G            Singular values
  8.24763E-02            1.9E+00             4.2E+00
  1.13575E+00           -1.5E+00             1.8E+00
  2.06664E+00           -1.5E+00             6.6E-02
```

```
 Itn       F evals        SUMSQ             GTG         Grade
  2          12        9.23238E-03         3.6E-02         3

      X                    G            Singular values
 8.24402E-02            1.4E-01            4.1E+00
 1.13805E+00           -9.5E-02            1.6E+00
 2.31707E+00           -9.5E-02            6.1E-02

 Itn       F evals        SUMSQ             GTG         Grade
  3          16        8.21492E-03         1.3E-06         3

      X                    G            Singular values
 8.24150E-02            8.2E-04            4.1E+00
 1.13323E+00           -5.8E-04            1.6E+00
 2.34337E+00           -5.8E-04            6.1E-02

 Itn       F evals        SUMSQ             GTG         Grade
  4          25        8.21488E-03         1.5E-14         2

      X                    G            Singular values
 8.24107E-02           -7.5E-08            4.1E+00
 1.13304E+00            8.6E-08            1.6E+00
 2.34369E+00            4.2E-08            6.1E-02

 Itn       F evals        SUMSQ             GTG         Grade
  5          29        8.21488E-03         2.2E-21         2

      X                    G            Singular values
 8.24106E-02           -6.2E-12            4.1E+00
 1.13304E+00           -3.2E-11            1.6E+00
 2.34370E+00            3.4E-11            6.1E-02

On exit, the sum of squares is      0.0082
at the point       0.0824      1.1330      2.3437
The estimated gradient is  -6.181E-12  -3.170E-11   3.409E-11
                           (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

E04FYF – NAG Fortran Library Routine Document

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.

Note. This routine was introduced into the NAG Fortran Library at Mark 19 and may therefore not be available to all users of the NAG Fortran SMP Library.

1 Purpose

E04FYF is an easy-to-use algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). No derivatives are required.

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 E04FYF(M, N, LSFUN1, X, FSUMSQ, W, LW, IUSER, USER,
1                 IFAIL)
INTEGER           M, N, LW, IUSER(*), IFAIL
real              X(N), FSUMSQ, W(LW), USER(*)
EXTERNAL          LSFUN1
```

3 Description

This routine is essentially identical to the subroutine LSNDN1 in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to evaluate functions $f_i(x)$ at any point x.

From a starting point supplied by the user, a sequence of points is generated which is intended to converge to a local minimum of the sum of squares. These points are generated using estimates of the curvature of $F(x)$.

4 References

[1] Gill P E and Murray W (1978) Algorithms for the solution of the nonlinear least-squares problem *SIAM J. Numer. Anal.* **15** 977–992

5 Parameters

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \text{N} \leq \text{M}$.

3: LSFUN1 — SUBROUTINE, supplied by the user. *External Procedure*

This routine must be supplied by the user to calculate the vector of values $f_i(x)$ at any point x. It should be tested separately before being used in conjunction with E04FYF (see the Chapter Introduction).

Its specification is:

```
SUBROUTINE LSFUN1(M, N, XC, FVECC, IUSER, USER)
INTEGER           M, N, IUSER(*)
real              XC(N), FVECC(M), USER(*)
```

1: M — INTEGER *Input*
2: N — INTEGER *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) — ***real*** array *Input*

On entry: the point x at which the values of the f_i are required.

4: FVECC(M) — ***real*** array *Output*

On exit: FVECC(i) must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

5: IUSER(*) — INTEGER array *User Workspace*
6: USER(*) — ***real*** array *User Workspace*

LSFUN1 is called from E04FYF with the parameters IUSER and USER as supplied to E04FYF. The user is free to use the arrays IUSER and USER to supply information to LSFUN1 as an alternative to using COMMON.

LSFUN1 must be declared as EXTERNAL in the (sub)program from which E04FYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: 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 $j = 1, 2, \ldots, n$.

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.

5: FSUMSQ — ***real*** *Output*

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

6: W(LW) — ***real*** array *Workspace*
7: LW — INTEGER *Input*

On entry: the length of W as declared in the (sub)program from which E04FYF is called.

Constraints:

LW $\geq 7 \times$ N + N $\times$ N + 2 $\times$ M $\times$ N + 3 $\times$ M + N $\times$ (N−1)/2, if N > 1,
LW $\geq 9 + 5 \times$ M, if N = 1.

8: IUSER(*) — INTEGER array *User Workspace*

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

IUSER is not used by E04FYF, but is passed directly to LSFUN1 and may be used to pass information to those routines.

9: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04FYF, but is passed directly to LSFUN1 and may be used to pass information to those routines.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to −1 before entry. **It is then 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 specified by the routine:

IFAIL = 1

On entry, N < 1,
or M < N,
or LW < 7 × N + N × N + 2 × M × N + 3 × M + N × (N−1)/2, when N > 1,
or LW < 9 + 5 × M, when N = 1.

IFAIL = 2

There have been 400 × n calls of LSFUN1, yet the algorithm does not seem to have converged. This may be due to an awkward function or to a poor starting point, so it is worth restarting E04FYF from the final point held in X.

IFAIL = 3

The final point does not satisfy the conditions for acceptance as a minimum, but no lower point could be found.

IFAIL = 4

An auxiliary routine has been unable to complete a singular value decomposition in a reasonable number of sub-iterations.

IFAIL = 5, 6, 7 and 8

There is some doubt about whether the point x found by E04FYF is a minimum of $F(x)$. 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.

If the user is not satisfied with the result (e.g. because IFAIL lies between 3 and 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. Repeated failure may indicate some defect in the formulation of the problem.

7 Accuracy

If the problem is reasonably well scaled and 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 the components of x and between $t - 1$ (if $F(x)$ is of order 1 at the minimum) and $2t - 2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

8 Further Comments

The number of iterations required depends on the number of variables, the number of residuals and their behaviour, and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04FYF varies, but for $m \gg n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least $n + 1$ calls of LSFUN1. So, unless the residuals can be evaluated very quickly, the run time will be dominated by the time spent in LSFUN1.

Ideally, the problem should be scaled so that the minimum value of the sum of squares is in the range $(0, +1)$, and so that at points a unit distance away from the solution the sum of squares 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 E04FYF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in segments of the workspace array W. See E04YCF for further details.

9 Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

9.1 Program Text

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.

```
*     E04FYF Example Program Text.
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          N, M, NT, LW
      PARAMETER        (N=3,M=15,NT=3,LW=7*N+N*N+2*M*N+3*M+N*(N-1)/2)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FSUMSQ
```

```
      INTEGER          I, IFAIL, J, K
*     .. Local Arrays ..
      real             T(M,NT), USER(M+M*NT), W(LW), X(N), Y(M)
      INTEGER          IUSER(1)
*     .. External Subroutines ..
      EXTERNAL         E04FYF, LSFUN1
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04FYF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
*
      IUSER(1) = NT
      K = M
      DO 40 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
         USER(I) = Y(I)
         DO 20 J = 1, NT
            USER(K+J) = T(I,J)
   20    CONTINUE
         K = K + NT
   40 CONTINUE
*
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
*
      IFAIL = 1
*
      CALL E04FYF(M,N,LSFUN1,X,FSUMSQ,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
      END
*
      SUBROUTINE LSFUN1(M,N,XC,FVECC,IUSER,USER)
*     .. Scalar Arguments ..
      INTEGER          M, N
*     .. Array Arguments ..
      real             FVECC(M), USER(*), XC(N)
      INTEGER          IUSER(*)
*     .. Local Scalars ..
      INTEGER          I, K
```

```
*     .. Executable Statements ..
      K = M
      DO 20 I = 1, M
         FVECC(I) = XC(1) + USER(K+1)/(XC(2)*USER(K+2)+XC(3)*USER(K+3))
     +             - USER(I)
         K = K + IUSER(1)
   20 CONTINUE
      RETURN
      END
```

9.2 Program Data

```
E04FYF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3 Program Results

```
E04FYF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
```

E04GBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04GBF is a comprehensive quasi-Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First 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 E04GBF (M, N, LSQLIN, LSQFUN, LSQMON, IPRINT, MAXCAL, ETA,
     1                   XTOL, STEPMX, X, FSUMSQ, FVEC, FJAC, LJ, S, V,
     2                   LV, NITER, NF, IW, LIW, W, LW, IFAIL)
      INTEGER           M, N, IPRINT, MAXCAL, LJ, LV, NITER, NF,
     1                  IW(LIW), LIW, LW, IFAIL
      real              ETA, XTOL, STEPMX, X(N), FSUMSQ, FVEC(M),
     1                  FJAC(LJ,N), S(N), V(LV,N), W(LW)
      EXTERNAL          LSQLIN, LSQFUN, LSQMON
```

3. Description

This routine is essentially identical to the subroutine LSQFDQ in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form:

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to calculate the values of the $f_i(x)$ and their first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x.

From a starting point $x^{(1)}$ supplied by the user, the routine generates a sequence of points $x^{(2)}, x^{(3)}, \ldots$, which is intended to converge to a local minimum of $F(x)$. The sequence of points is given by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$$

where the vector $p^{(k)}$ is a direction of search, and $\alpha^{(k)}$ is chosen such that $F(x^{(k)} + \alpha^{(k)} p^{(k)})$ is approximately a minimum with respect to $\alpha^{(k)}$.

The vector $p^{(k)}$ used depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced, then $p^{(k)}$ is the Gauss-Newton direction; otherwise the second derivatives of the $f_i(x)$ are taken into account using a quasi-Newton updating scheme.

The method is designed to ensure that steady progress is made whatever the starting point, and to have the rapid ultimate convergence of Newton's method.

4. References

[1] GILL, P.E. and MURRAY, W.
Algorithms for the Solution of the Nonlinear Least-squares Problem.
SIAM J. Numer. Anal., 15, pp. 977-992, 1978.

5. Parameters

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathrm{N} \leq \mathrm{M}$.

3: LSQLIN – SUBROUTINE, supplied by the NAG Fortran Library. *External Procedure*

This parameter enables the user to specify whether the linear minimizations (i.e. minimizations of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$) are to be performed by a routine which just requires the evaluation of the $f_i(x)$ (E04FCV), or by a routine which also requires the first derivatives of the $f_i(x)$ (E04HEV).

It will often be possible to evaluate the first derivatives of the residuals in about the same amount of computer time that is required for the evaluation of the residuals themselves – if this is so then E04GBF should be called with routine E04HEV as the parameter LSQLIN. However, if the evaluation of the derivatives takes more than about 4 times as long as the evaluation of the residuals, then E04FCV will usually be preferable. If in doubt, use E04HEV as it is slightly more robust.

The routine names E04FCV and E04HEV may be implementation dependent: see the Users' Note for your implementation for details.

Whichever subroutine is used must be declared as EXTERNAL in the (sub)program from which E04GBF is called.

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

LSQFUN must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point x. (However, if the user does not wish to calculate the residuals or first derivatives at a particular x, there is the option of setting a parameter to cause E04GBF to terminate immediately.)

Its specification is:

```
 SUBROUTINE LSQFUN (IFLAG, M, N, XC, FVECC, FJACC, LJC,
1           IW, LIW, W, LW)
 INTEGER    IFLAG, M, N, LJC, IW(LIW), LIW, LW
 real       XC(N), FVECC(M), FJACC(LJC,N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG will be set to 0, 1 or 2. The value 0 indicates that only the residuals need to be evaluated, the value 1 indicates that only the Jacobian matrix needs to be evaluated, and the value 2 indicates that both the residuals and the Jacobian matrix must be calculated. (If E04HEV is used as E04GBF's parameter LSQLIN, LSQFUN will always be called with IFLAG set to 2.)

On exit: if it is not possible to evaluate the $f_i(x)$ or their first derivatives at the point given in XC (or if it is wished to stop the calculations for any other reason), the user should reset IFLAG to some negative number and return control to E04GBF. E04GBF will then terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: M – INTEGER. *Input*

3: N – INTEGER. *Input*

On entry: the number m and n of residuals and variables, respectively.

4: XC(N) – ***real*** array. *Input*

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

5: FVECC(M) – ***real*** array. *Output*

On exit: unless IFLAG = 1 on entry, or IFLAG is reset to a negative number, then FVECC(i) must contain the value of f_i at the point x, for $i = 1,2,...,m$.

6: FJACC(LJC,N) – ***real*** array. *Output*

On exit: unless IFLAG = 0 on entry, or IFLAG is reset to a negative number, then FJACC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1,2,..,m$; $j = 1,2,...,n$.

7: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC as declared in the (sub)program from which E04GBF is called.

8: IW(LIW) – INTEGER array. *Workspace*

9: LIW – INTEGER. *Input*

10: W(LW) – ***real*** array. *Workspace*

11: LW – INTEGER. *Input*

LSQFUN is called with E04GBF's parameters IW, LIW, W, LW as these parameters. They are present so that, when other library routines require the solution of a minimization subproblem, constants needed for the evaluation of residuals can be passed through IW and W. Similarly, users could pass quantities to LSQFUN from the segment which calls E04GBF by using partitions of IW and W beyond those used as workspace by E04GBF. However, because of the danger of mistakes in partitioning, it is **recommended** that users should pass information to LSQFUN via COMMON and **not use IW or W** at all. In any case users **must not change** the elements of IW and W used as workspace by E04GBF.

Note: LSQFUN should be tested separately before being used in conjunction with E04GBF.

LSQFUN must be declared as EXTERNAL in the (sub)program from which E04GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: LSQMON – SUBROUTINE, supplied by the user. *External Procedure*

If IPRINT ≥ 0, the user must supply a subroutine LSQMON which is suitable for monitoring the minimization process. LSQMON must not change the values of any of its parameters.

If IPRINT < 0, the NAG Fortran Library dummy routine E04FDZ can be used as LSQMON. (In some implementations the name of this routine is FDZE04; refer to the Users' Note for your implementation.)

Its specification is:

```
 SUBROUTINE LSQMON(M, N, XC, FVECC, FJACC, LJC, S, IGRADE, NITER, NF,
1           IW, LIW, W, LW)
 INTEGER    M, N, LJC, IGRADE, NITER, NF, IW(LIW), LIW, LW
 real       XC(N), FVECC(M), FJACC(LJC,N), S(N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

4: FVECC(M) – ***real*** array. *Input*

On entry: the value of the residuals f_i at the current point x.

5: FJACC(LJC,N) – ***real*** array. *Input*

On entry: FJACC(i,j) contains the value of $\frac{\partial f_i}{\partial x_j}$ at the current point x, for $i = 1,2,...,m$; $j = 1,2,...,n$.

6: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

7: S(N) – ***real*** array. *Input*

On entry: the singular values of the current Jacobian matrix. Thus S may be useful as information about the structure of the user's problem.

8: IGRADE – INTEGER. *Input*

On entry: E04GBF estimates the dimension of the subspace for which the Jacobian matrix can be used as a valid approximation to the curvature (see Gill and Murray [1]). This estimate is called the grade of the Jacobian matrix, and IGRADE gives its current value.

9: NITER – INTEGER. *Input*

On entry: the number of iterations which have been performed in E04GBF.

10: NF – INTEGER. *Input*

On entry: the number of evaluations of the residuals. (If E04HEV is used as LSQLIN, NF is also the number of evaluations of the Jacobian matrix.)

11: IW(LIW) – INTEGER array. *Workspace*

12: LIW – INTEGER. *Input*

13: W(LW) – ***real*** array. *Workspace*

14: LW – INTEGER. *Input*

As in LSQFUN, these parameters correspond to the parameters IW, LIW, W, LW of E04GBF. They are included in LSQMON's parameter list primarily for when E04GBF is called by other library routines.

Note: the user should normally print the sum of squares of residuals, so as to be able to examine the sequence of values of $F(x)$ mentioned in Section 7. It is usually helpful to also print XC, the gradient of the sum of squares, NITER and NF.

LSQMON must be declared as EXTERNAL in the (sub)program from which E04GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: IPRINT – INTEGER. *Input*

On entry: the frequency with which LSQMON is to be called. If IPRINT > 0, LSQMON is called once every IPRINT iterations and just before exit from E04GBF. If IPRINT = 0, LSQMON is just called at the final point. If IPRINT < 0, LSQMON is not called at all.

IPRINT should normally be set to a small positive number.

Suggested value: IPRINT = 1.

7: MAXCAL – INTEGER. *Input*

On entry: this parameter enables the user to limit the number of times that LSQFUN is called by E04GBF. There will be an error exit (see Section 6) after MAXCAL calls of LSQFUN.

Suggested value: MAXCAL = $75\times n$ if E04FCV is used as LSQLIN,
MAXCAL = $50\times n$ if E04HEV is used as LSQLIN.

Constraint: MAXCAL $\geq$ 1.

8: ETA – ***real***. *Input*

On entry: every iteration of E04GBF involves a linear minimization (i.e. minimization of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$). ETA specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha^{(k)}$ 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 performed by E04GBF, they will increase the number of calls of LSQFUN made every iteration. On balance it is usually more efficient to perform a low accuracy minimization.

Suggested value: ETA = 0.9 if N > 1 and E04HEV is used as LSQLIN,
ETA = 0.5 if N > 1 and E04FCV is uses as LSQLIN,
ETA = 0.0 if N = 1.

Constraint: 0.0 $\leq$ ETA < 1.0.

9: XTOL – ***real***. *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 prior to a normal exit, is such that

$$\|x_{sol}-x_{true}\| < \text{XTOL}\times(1.0+\|x_{true}\|),$$

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 = 1.0E−5, then x_{sol} is usually accurate to about 5 decimal places. (For further details see Section 7.)

If $F(x)$ and the variables are scaled roughly as described in Section 8 and ε is the ***machine precision***, then a setting of order XTOL = $\sqrt{\varepsilon}$ will usually be appropriate. If XTOL is set to 0.0 or some positive value less than 10ε, E04GBF will use 10ε instead of XTOL, since 10ε is probably the smallest reasonable setting.

Constraint: XTOL $\geq$ 0.0.

10: STEPMX – ***real***. *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency, a slight overestimate is preferable.) E04GBF will ensure that, for each iteration,

$$\sum_{j=1}^{n}(x_j^{(k)}-x_j^{(k-1)})^2 \leq (\text{STEPMX})^2$$

where k is the iteration number. Thus, if the problem has more than one solution, E04GBF 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 help avoid 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.

11: 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 $j = 1,2,...,N$.

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.

12: FSUMSQ – ***real***. *Output*

On exit: the value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in X.

13: FVEC(M) – ***real*** array. *Output*

On exit: the value of the residual $f_i(x)$ at the final point given in X, for $i = 1,2,...,m$.

14: FJAC(LJ,N) – ***real*** array. *Output*

On exit: the value of the first derivative $\frac{\partial f_i}{\partial x_j}$ evaluated at the final point given in X, for $i = 1,2,...,m$; $j = 1,2,...,n$.

15: LJ – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which E04GBF is called.

Constraint: LJ ≥ M.

16: S(N) – ***real*** array. *Output*

On exit: the singular values of the Jacobian matrix at the final point. Thus S may be useful as information about the structure of the user's problem.

17: V(LV,N) – ***real*** array. *Output*

On exit: the matrix V associated with the singular value decomposition

$$J = USV^T$$

of the Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalised eigenvectors of J^TJ.

18: LV – INTEGER. *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which E04GBF is called.

Constraint: LV ≥ N.

19: NITER – INTEGER. *Output*

On exit: the number of iterations which have been performed in E04GBF.

20: NF – INTEGER. *Output*

On exit: the number of times that the residuals have been evaluated (i.e. the number of calls of LSQFUN). If E04HEV is used as LSQLIN, NF is also the number of times that the Jacobian matrix has been evaluated.

21: IW(LIW) – INTEGER array. *Workspace*

22: LIW – INTEGER. *Input*

On entry: the length of IW as declared in the (sub)program from which E04GBF is called.

Constraint: LIW ≥ 1.

23: W(LW) – ***real*** array. *Workspace*
24: LW – INTEGER. *Input*

On entry: the length of W as declared in the (sub)program from which E04GBF is called.

Constraints: LW ≥ 7×N + M×N + 2×M + N×N, if N > 1
LW ≥ 9 + 3×M, if N = 1.

25: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL < 0

A negative value of IFAIL indicates an exit from E04GBF because the user has set IFLAG negative in LSQFUN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N < 1,
or M < N,
or MAXCAL < 1,
or ETA < 0.0,
or ETA ≥ 1.0,
or XTOL < 0.0,
or STEPMX < XTOL,
or LJ < M,
or LV < N,
or LIW < 1,
or LW < 7×N + M×N + 2×M + N×N, when N > 1,
or LW < 9 + 3×M, when N = 1.

When this exit occurs, no values will have been assigned to FSUMSQ, or to the elements of FVEC, FJAC, S or V.

IFAIL = 2

There have been MAXCAL calls of LSQFUN. If steady reductions in the sum of squares, $F(x)$, 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 satisfied, but a lower point could not be found. This could be because XTOL has been set so small that rounding errors in the evaluation of the residuals and derivatives make attainment of the convergence conditions impossible.

IFAIL = 4

The method for computing the singular value decomposition of the Jacobian matrix has failed to converge in a reasonable number of sub-iterations. It may be worth applying E04GBF again starting with an initial approximation which is not too close to the point at which the failure occurred.

The values IFAIL = 2, 3 and 4 may also be caused by mistakes in LSQFUN, 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 E04GBF when (B1, B2 and B3) or B4 or B5 hold, where

$$\text{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\varepsilon) \times (1.0+\|x^{(k)}\|)$$

$$\text{B2} \equiv |F^{(k)} - F^{(k-1)}| < (\text{XTOL}+\varepsilon)^2 \times (1.0+F^{(k)})$$

$$\text{B3} \equiv \|g^{(k)}\| < \varepsilon^{\frac{1}{3}} \times (1.0+F^{(k)})$$

$$\text{B4} \equiv F^{(k)} < \varepsilon^2$$

$$\text{B5} \equiv \|g^{(k)}\| < \left(\varepsilon \times \sqrt{F^{(k)}}\right)^{\frac{1}{2}}$$

and where $\|.\|$ and ε are as defined in Section 5 (XTOL), and $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of first derivatives at $x^{(k)}$.

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

If IFAIL = 3, then x_{sol} may still be a good estimate of x_{true}, but to verify this the user should make the following checks. If

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

(b) $g(x_{sol})^T g(x_{sol}) < 10\varepsilon$, where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum. When (b) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. The values of $F(x^{(k)})$ can be calculated in LSQMON, and the vector $g(x_{sol})$ can be calculated from the contents of FVEC and FJAC on exit from E04GBF.

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

8. Further Comments

The number of iterations required depends on the number of variables, the number of residuals, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04GBF varies, but for $m \gg n$ is approximately $n \times m^2 + \text{O}(n^3)$. In addition, each iteration makes at least one call of LSQFUN. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSQFUN.

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 of $F(x)$ at the solution is well-conditioned. 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 E04GBF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in the arrays S and V. See E04YCF for further details.

9. Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

Before calling E04GBF, the program calls E04YAF to check LSQFUN. It uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04GBF Example Program Text.
*     Mark 15 Revised.  NAG Copyright 1991.
*     .. Parameters ..
      INTEGER          M, N, NT, LIW, LJ, LV, LW
      PARAMETER        (M=15,N=3,NT=3,LIW=1,LJ=M,LV=N,
     +                 LW=7*N+M*N+2*M+N*N)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Arrays in Common ..
      real             T(M,NT), Y(M)
*     .. Local Scalars ..
      real             ETA, FSUMSQ, STEPMX, XTOL
      INTEGER          I, IFAIL, IPRINT, J, MAXCAL, NF, NITER
*     .. Local Arrays ..
      real             FJAC(LJ,N), FVEC(M), G(N), S(N), V(LV,N), W(LW),
     +                 X(N)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             X02AJF
      EXTERNAL         X02AJF
*     .. External Subroutines ..
      EXTERNAL         E04GBF, E04HEV, E04YAF, LSQFUN, LSQGRD, LSQMON
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Common blocks ..
      COMMON           Y, T
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04GBF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
      DO 20 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
   20 CONTINUE
*     Check LSQFUN by calling E04YAF at an arbitrary point
      X(1) = 0.19e0
      X(2) = -1.34e0
      X(3) = 0.88e0
      IFAIL = 0
*
      CALL E04YAF(M,N,LSQFUN,X,FVEC,FJAC,LJ,IW,LIW,W,LW,IFAIL)
*
*     Continue setting parameters for E04GBF
*     * Set IPRINT to 1 to obtain output from LSQMON at each iteration *
      IPRINT = -1
      MAXCAL = 50*N
*     Since E04HEV is being used as LSQLIN, we set ETA to 0.9
      ETA = 0.9e0
      XTOL = 10.0e0*SQRT(X02AJF())
*     We estimate that the minimum will be within 10 units of the
*     starting point
      STEPMX = 10.0e0
*     Set up the starting point
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
      IFAIL = 1
*
      CALL E04GBF(M,N,E04HEV,LSQFUN,LSQMON,IPRINT,MAXCAL,ETA,XTOL,
     +            STEPMX,X,FSUMSQ,FVEC,FJAC,LJ,S,V,LV,NITER,NF,IW,LIW,W,
     +            LW,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
         CALL LSQGRD(M,N,FVEC,FJAC,LJ,G)
         WRITE (NOUT,99997) 'The corresponding gradient is',
     +     (G(J),J=1,N)
         WRITE (NOUT,*) '                                (machine dependent)'
         WRITE (NOUT,*) 'and the residuals are'
         WRITE (NOUT,99996) (FVEC(I),I=1,M)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
99997 FORMAT (1X,A,1P,3e12.3)
99996 FORMAT (1X,1P,e9.1)
      END
*
      SUBROUTINE LSQFUN(IFLAG,M,N,XC,FVECC,FJACC,LJC,IW,LIW,W,LW)
*     Routine to evaluate the residuals and their 1st derivatives.
*     This routine is also suitable for use when E04FCV is used as
*     LSQLIN, since it can deal with IFLAG = 0 as well as IFLAG = 2.
*     .. Parameters ..
      INTEGER           MDEC, NT
      PARAMETER         (MDEC=15,NT=3)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LIW, LJC, LW, M, N
```

```
*     .. Array Arguments ..
      real                FJACC(LJC,N), FVECC(M), W(LW), XC(N)
      INTEGER             IW(LIW)
*     .. Arrays in Common ..
      real                T(MDEC,NT), Y(MDEC)
*     .. Local Scalars ..
      real                DENOM, DUMMY
      INTEGER             I
*     .. Common blocks ..
      COMMON              Y, T
*     .. Executable Statements ..
      DO 20 I = 1, M
         DENOM = XC(2)*T(I,2) + XC(3)*T(I,3)
         FVECC(I) = XC(1) + T(I,1)/DENOM - Y(I)
         IF (IFLAG.EQ.0) GO TO 20
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = T(I,1)*T(I,2)*DUMMY
         FJACC(I,3) = T(I,1)*T(I,3)*DUMMY
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE LSQMON(M,N,XC,FVECC,FJACC,LJC,S,IGRADE,NITER,NF,IW,LIW,
     +                  W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER             NDEC
      PARAMETER           (NDEC=3)
      INTEGER             NOUT
      PARAMETER           (NOUT=6)
*     .. Scalar Arguments ..
      INTEGER             IGRADE, LIW, LJC, LW, M, N, NF, NITER
*     .. Array Arguments ..
      real                FJACC(LJC,N), FVECC(M), S(N), W(LW), XC(N)
      INTEGER             IW(LIW)
*     .. Local Scalars ..
      real                FSUMSQ, GTG
      INTEGER             J
*     .. Local Arrays ..
      real                G(NDEC)
*     .. External Functions ..
      real                F06EAF
      EXTERNAL            F06EAF
*     .. External Subroutines ..
      EXTERNAL            LSQGRD
*     .. Executable Statements ..
      FSUMSQ = F06EAF(M,FVECC,1,FVECC,1)
      CALL LSQGRD(M,N,FVECC,FJACC,LJC,G)
      GTG = F06EAF(N,G,1,G,1)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '  Itn      F evals        SUMSQ              GTG         Grade'
      WRITE (NOUT,99999) NITER, NF, FSUMSQ, GTG, IGRADE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '       X                    G           Singular values'
      DO 20 J = 1, N
         WRITE (NOUT,99998) XC(J), G(J), S(J)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,I4,6X,I5,6X,1P,e13.5,6X,1P,e9.1,6X,I3)
99998 FORMAT (1X,1P,e13.5,10X,1P,e9.1,10X,1P,e9.1)
      END
*
```

```
      SUBROUTINE LSQGRD(M,N,FVECC,FJACC,LJC,G)
*     Routine to evaluate gradient of the sum of squares
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), G(N)
*     .. Local Scalars ..
      real              SUM
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 J = 1, N
         SUM = 0.0e0
         DO 20 I = 1, M
            SUM = SUM + FJACC(I,J)*FVECC(I)
   20    CONTINUE
         G(J) = SUM + SUM
   40 CONTINUE
      RETURN
      END
```

9.2. Program Data

```
E04GBF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3. Program Results

```
E04GBF Example Program Results

On exit, the sum of squares is       0.0082
at the point       0.0824      1.1330      2.3437
The corresponding gradient is   1.199E-09  -1.865E-11   1.807E-11
                               (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

With IPRINT set to 1 in the example program, intermediate results similar to those below are obtained form LSQMON:

```
E04GBF Example Program Results

 Itn       F evals        SUMSQ            GTG         Grade
  0          1        1.02104E+01         1.0E+03         3

      X                    G            Singular values
 5.00000E-01             2.1E+01            5.0E+00
 1.00000E+00            -1.7E+01            2.6E+00
 1.50000E+00            -1.6E+01            9.6E-02

 Itn       F evals        SUMSQ            GTG         Grade
  1          2        1.98730E-01         8.1E+00         3

      X                    G            Singular values
 8.24763E-02             1.9E+00            4.2E+00
 1.13575E+00            -1.5E+00            1.8E+00
 2.06664E+00            -1.5E+00            6.6E-02

 Itn       F evals        SUMSQ            GTG         Grade
  2          3        9.23238E-03         3.6E-02         3

      X                    G            Singular values
 8.24402E-02             1.4E-01            4.1E+00
 1.13805E+00            -9.5E-02            1.6E+00
 2.31707E+00            -9.5E-02            6.1E-02

 Itn       F evals        SUMSQ            GTG         Grade
  3          4        8.21492E-03         1.3E-06         3

      X                    G            Singular values
 8.24150E-02             8.2E-04            4.1E+00
 1.13323E+00            -5.8E-04            1.6E+00
 2.34337E+00            -5.8E-04            6.1E-02

 Itn       F evals        SUMSQ            GTG         Grade
  4          5        8.21488E-03         2.5E-15         2

      X                    G            Singular values
 8.24107E-02             3.4E-08            4.1E+00
 1.13304E+00             8.9E-09            1.6E+00
 2.34369E+00            -3.5E-08            6.1E-02

 Itn       F evals        SUMSQ            GTG         Grade
  5          6        8.21488E-03         2.2E-17         0

      X                    G            Singular values
 8.24106E-02             9.5E-11            4.1E+00
 1.13304E+00             3.5E-09            1.6E+00
 2.34369E+00            -3.2E-09            6.1E-02

 Itn       F evals        SUMSQ            GTG         Grade
  6          7        8.21488E-03         1.4E-18         0

      X                    G            Singular values
 8.24106E-02             1.2E-09            4.1E+00
 1.13304E+00            -1.9E-11            1.6E+00
 2.34370E+00             1.8E-11            6.1E-02
```

```
On exit, the sum of squares is       0.0082
at the point       0.0824       1.1330       2.3437
The corresponding gradient is   1.199E-09  -1.865E-11   1.807E-11
                             (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

E04GDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04GDF is a comprehensive modified Gauss-Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First 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 E04GDF (M, N, LSQFUN, LSQMON, IPRINT, MAXCAL, ETA, XTOL,
     1                   STEPMX, X, FSUMSQ, FVEC, FJAC, LJ, S, V, LV,
     2                   NITER, NF, IW, LIW, W, LW, IFAIL)
      INTEGER           M, N, IPRINT, MAXCAL, LJ, LV, NITER, NF,
     1                  IW(LIW), LIW, LW, IFAIL
      real              ETA, XTOL, STEPMX, X(N), FSUMSQ, FVEC(M),
     1                  FJAC(LJ,N), S(N), V(LV,N), W(LW)
      EXTERNAL          LSQFUN, LSQMON
```

3. Description

This routine is essentially identical to the subroutine LSQFDN in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to calculate the values of the $f_i(x)$ and their first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point x.

From a starting point $x^{(1)}$ supplied by the user, the routine generates a sequence of points $x^{(2)}, x^{(3)}, \ldots$, which is intended to converge to a local minimum of $F(x)$. The sequence of points is given by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$$

where the vector $p^{(k)}$ is a direction of search, and $\alpha^{(k)}$ is chosen such that $F(x^{(k)} + \alpha^{(k)} p^{(k)})$ is approximately a minimum with respect to $\alpha^{(k)}$.

The vector $p^{(k)}$ used depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced, then $p^{(k)}$ is the Gauss-Newton direction; otherwise finite difference estimates of the second derivatives of the $f_i(x)$ are taken into account.

The method is designed to ensure that steady progress is made whatever the starting point, and to have the rapid ultimate convergence of Newton's method.

4. References

[1] GILL, P.E. and MURRAY, W.
Algorithms for the Solution of the Nonlinear Least-squares Problem.
SIAM J. Numer. Anal., 15, pp. 977-992, 1978.

5. Parameters

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq N \leq M$.

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

LSQFUN must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point x. (However, if the user does not wish to calculate the residuals or first derivatives at a particular x, there is the option of setting a parameter to cause E04GDF to terminate immediately.)

Its specification is:

```
 SUBROUTINE LSQFUN (IFLAG, M, N, XC, FVECC, FJACC, LJC,
1           IW, LIW, W, LW)
 INTEGER    IFLAG, M, N, LJC, IW(LIW), LIW, LW
 real       XC(N), FVECC(M), FJACC(LJC,N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: IFLAG – INTEGER. *Input/Output*

On entry: to LSQFUN, IFLAG will be set to 1 or 2. The value 1 indicates that only the Jacobian matrix needs to be evaluated, and the value 2 indicates that both the residuals and the Jacobian matrix must be calculated.

On exit: if it is not possible to evaluate the $f_i(x)$ or their first derivatives at the point given in XC (or if it wished to stop the calculations for any other reason), the user should reset IFLAG to some negative number and return control to E04GDF. E04GDF will then terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: M – INTEGER. *Input*

3: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

4: XC(N) – ***real*** array. *Input*

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

5: FVECC(M) – ***real*** array. *Output*

On exit: unless IFLAG = 1 on entry or IFLAG is reset to a negative number, then FVECC(i) must contain the value of f_i at the point x, for $i = 1,2,...,m$.

6: FJACC(LJC,N) – ***real*** array. *Output*

On exit: unless IFLAG is reset to a negative number FJACC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1,2,...,m$; $j = 1,2,...,n$.

7: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

8: IW(LIW) – INTEGER array. *Workspace*
9: LIW – INTEGER. *Input*
10: W(LW) – ***real*** array. *Workspace*
11: LW – INTEGER. *Input*

LSQFUN is called with E04GDF's parameters IW, LIW, W, LW as these parameters. They are present so that, when other library routines require the solution of a minimization subproblem, constants needed for the evaluation of residuals can be passed through IW and W. Similarly, users could pass quantities to LSQFUN from the segment which calls E04GDF by using partitions of IW and W beyond those used as workspace by E04GDF. However, because of the danger of mistakes in partitioning, it is **recommended** that users should pass information to LSQFUN via COMMON and **not use IW or W** at all. In any case users **must not change** the elements of IW and W used as workspace by E04GDF.

Note: LSQFUN should be tested separately before being used in conjunction with E04GDF.

LSQFUN must be declared as EXTERNAL in the (sub)program from which E04GDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

If IPRINT ≥ 0, the user must supply a subroutine LSQMON which is suitable for monitoring the minimization process. LSQMON must not change the values of any of its parameters.

If IPRINT < 0, the dummy routine E04FDZ can be used as LSQMON. (In some implementations the name of this routine is FDZE04; refer to the Users' Note for your implementation.)

Its specification is:

```
 SUBROUTINE LSQMON (M, N, XC, FVECC, FJACC, LJC, S, IGRADE, NITER,
1           NF, IW, LIW, W, LW)
 INTEGER    M, N, LJC, IGRADE, NITER, NF, IW(LIW), LIW, LW
 real       XC(N), FVECC(M), FJACC(LJC,N), S(N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M – INTEGER. *Input*
2: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

4: FVECC(M) – ***real*** array. *Input*

On entry: the values of the residuals f_i at the current point x.

5: FJACC(LJC,N) – ***real*** array. *Input*

On entry: FJACC(i,j) contains the value of $\frac{\partial f_i}{\partial x_j}$ at the current point x, for $i = 1,2,...,m$; $j = 1,2,...,n$

6: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

7: S(N) – ***real*** array. *Input*

On entry: the singular values of the current Jacobian matrix. Thus S may be useful as information about the structure of the user's problem. (If IPRINT > 0, LSQMON is called at the initial point before the singular values have been calculated. So the elements of S are set to zero for the first call of LSQMON.)

8: IGRADE – INTEGER. *Input*

On entry: E04GDF estimates the dimension of the subspace for which the Jacobian matrix can be used as a valid approximation to the curvature (see Gill and Murray [1]). This estimate is called the grade of the Jacobian matrix, and IGRADE gives its current value.

9: NITER – INTEGER. *Input*

On entry: the number of iterations which have been performed in E04GDF.

10: NF – INTEGER. *Input*

On entry: the number of times that LSQFUN has been called so far with IFLAG = 2. (In addition to these calls monitored by NF, LSQFUN is called not more than N times per iteration with IFLAG set to 1.)

11: IW(LIW) – INTEGER array. *Workspace*

12: LIW – INTEGER. *Input*

13: W(LW) – ***real*** array. *Workspace*

14: LW – INTEGER. *Input*

As in LSQFUN, these parameters correspond to the parameters IW, LIW, W, LW of E04GDF. They are included in LSQMON's parameter list primarily for when E04GDF is called by other library routines.

Note: the user should normally print the sum of squares of residuals, so as to be able to examine the sequence of values of $F(x)$ mentioned in Section 7. It is usually also helpful to print XC, the gradient of the sum of squares, NITER and NF.

LSQMON must be declared as EXTERNAL in the (sub)program from which E04GDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: IPRINT – INTEGER. *Input*

On entry: the frequency with which LSQMON is to be called. If IPRINT > 0, LSQMON is called once every IPRINT iterations and just before exit from E04GDF. If IPRINT = 0, LSQMON is just called at the final point. If IPRINT < 0, LSQMON 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: this parameter enables the user to limit the number of times that LSQFUN is called by E04GDF. There will be an error exit (see Section 6) after MAXCAL evaluations of the residuals (i.e. calls of LSQFUN with IFLAG set to 2). It should be borne in mind that, in addition to the calls of FUNCT which are limited directly by MAXCAL, there will be calls of LSQFUN (with IFLAG set to 1) to evaluate only first derivatives.

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

Constraint: MAXCAL $\geq$ 1.

7: ETA – ***real***. *Input*

On entry: every iteration of E04GDF involves a linear minimization (i.e. minimization of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$. ETA specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha^{(k)}$ 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, they will tend to increase the number of calls of LSQFUN (with IFLAG set to 2) needed for each linear minimization. On balance it is usually efficient to perform a low accuracy linear minimization.

Suggested value: ETA = 0.5 (ETA = 0.0 if N = 1).

Constraint: 0.0 ≤ ETA < 1.0.

8: XTOL – ***real***. *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 prior to a normal exit, is such that

$$\|x_{sol}-x_{true}\| < \text{XTOL}\times(1.0+\|x_{true}\|)$$

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 = 1.0E−5, then x_{sol} is usually accurate to about 5 decimal places. (For further details see Section 7.)

If $F(x)$ and the variables are scaled roughly as described in Section 8 and ϵ is the ***machine precision***, then a setting of order XTOL = $\sqrt{\epsilon}$ will usually be appropriate. If XTOL is set to 0.0 or some positive value less than 10ϵ, E04GDF will use 10ϵ instead of XTOL, since 10ϵ is probably the smallest reasonable setting.

Constraint: XTOL ≥ 0.0.

9: STEPMX – ***real***. *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency, a slight overestimate is preferable.) E04GDF will ensure that, for each iteration,

$$\sum_{j=1}^{n} (x_j^{(k)}-x_j^{(k-1)})^2 \leq (\text{STEPMX})^2$$

where k is the iteration number. Thus, if the problem has more than one solution, E04GDF 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 help avoid overflow in the evaluation of $F(x)$. However, an underestimate of STEPMX can lead to inefficiency.

Suggested value: STEPMX = 100000.0.

Constraint: STEPMX ≥ XTOL.

10: 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 (j = 1,2,...,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.

11: FSUMSQ – ***real***. *Output*

On exit: the value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in X.

12: FVEC(M) – ***real*** array. *Output*

On exit: the value of the residual $f_i(x)$ at the final point given in X, for $i = 1,2,...,m$.

13: FJAC(LJ,N) – ***real*** array. *Output*

On exit: the value of the first derivative $\frac{\partial f_i}{\partial x_j}$ evaluated at the final point given in X, for $i = 1,2,...,m$; $j = 1,2,...,n$.

14: LJ – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which E04GDF is called.

Constraint: LJ ≥ M.

15: S(N) – ***real*** array. *Output*

On exit: the singular values of the Jacobian matrix at the final point. Thus S may be useful as information about the structure of the user's problem.

16: V(LV,N) – ***real*** array. *Output*

On exit: the matrix V associated with the singular value decomposition

$$J = USV^T$$

of the Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalised eigenvectors of J^TJ.

17: LV – INTEGER. *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which E04GDF is called.

Constraint: LV ≥ N.

18: NITER – INTEGER. *Output*

On exit: the number of iterations which have been performed in E04GDF.

19: NF – INTEGER. *Output*

On exit: the number of times that the residuals have been evaluated (i.e. number of calls of LSQFUN with IFLAG set to 2).

20: IW(LIW) – INTEGER array. *Workspace*

21: LIW – INTEGER. *Input*

On entry: the length of IW as declared in the (sub)program from which E04GDF is called.

Constraint: LIW ≥ 1.

22: W(LW) – ***real*** array. *Workspace*

23: LW – INTEGER. *Input*

On entry: the length of W as declared in the (sub)program from which E04GDF is called.

Constraints: LW ≥ 7×N + M×N + 2×M + N×N, if N > 1.
LW ≥ 9 + 3×M, if N = 1.

24: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL < 0

A negative value of IFAIL indicates an exit from E04GDF because the user has set IFLAG negative in LSQFUN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N < 1,
or M < N,
or MAXCAL < 1,
or ETA < 0.0,
or ETA ≥ 1.0,
or XTOL < 0.0,
or STEPMX < XTOL,
or LJ < M,
or LV < N,
or LIW < 1,
or LW < 7×N + M×N + 2×M + N×N when N > 1,
or LW < 9 + 3×M when N = 1.

When this exit occurs, no values will have been assigned to FSUMSQ, or to the elements of FVEC, FJAC, S or V.

IFAIL = 2

There have been MAXCAL evaluations of the residuals. If steady reductions in the sum of squares, $F(x)$, were monitored up to the point where this exit occurred, then the exit probably occurred simply becuase 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 satisfied, but a lower point could not be found. This could be because XTOL has been set so small that rounding errors in the evaluation of the residuals and derivatives make attainment of the convergence conditions impossible.

IFAIL = 4

The method for computing the singular value decomposition of the Jacobian matrix has failed to converge in a reasonable number of sub-iterations. It may be worth applying E04GDF again starting with an initial approximation which is not too close to the point at which the failure occurred.

The values IFAIL = 2, 3 and 4 may also be caused by mistakes in LSQFUN, 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 E04GDF when the matrix of approximate second derivatives of $F(x)$ is positive-definite, and when (B1, B2 and B3) or B4 or B5 hold, where

$$\text{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\varepsilon) \times (1.0+\|x^{(k)}\|)$$

$$\text{B2} \equiv |F^{(k)}-F^{(k-1)}| < (\text{XTOL}+\varepsilon)^2 \times (1.0+F^{(k)})$$

$$\text{B3} \equiv \|g^{(k)}\| < \varepsilon^{1/3} \times (1.0+F^{(k)})$$

$$\text{B4} \equiv F^{(k)} < \varepsilon^2$$

$$\text{B5} \equiv \|g^{(k)}\| < \left(\varepsilon \times \sqrt{F^{(k)}}\right)^{1/2}$$

and where $\|.\|$ and ε are as defined in Section 5, and $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of estimated first derivatives at $x^{(k)}$.

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

If IFAIL = 3, then x_{sol} may still be a good estimate of x_{true}, but to verify this the user should make the following checks. If

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

(b) $g(x_{sol})^T g(x_{sol}) < 10\varepsilon$, where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum. When (b) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. The values of $F(x^{(k)})$ can be calculated in LSQMON, and the vector $g(x_{sol})$ can be calculated from the contents of FVEC and FJAC on exit from E04GDF.

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

8. Further Comments

The number of iterations required depends on the number of variables, the number of residuals, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04GDF varies, but for $m \gg n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least one call of LSQFUN. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSQFUN.

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 aproximately one unit. This will usually imply that the Hessian matrix of $F(x)$ at the solution is well-conditioned. 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 E04GDF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in the arrays S and V. See E04YCF for further details.

9. Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

Before calling E04GDF, the program calls E04YAF to check LSQFUN. It uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04GDF Example Program Text.
*     Mark 15 Revised.  NAG Copyright 1991.
*     .. Parameters ..
      INTEGER          N, M, NT, LV, LJ, LIW, LW
      PARAMETER        (N=3,M=15,NT=3,LV=N,LJ=M,LIW=1,
     +                 LW=7*N+M*N+2*M+N*N)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Arrays in Common ..
      real             T(M,NT), Y(M)
*     .. Local Scalars ..
      real             ETA, FSUMSQ, STEPMX, XTOL
      INTEGER          I, IFAIL, IPRINT, J, MAXCAL, NF, NITER
*     .. Local Arrays ..
      real             FJAC(LJ,N), FVEC(M), G(N), S(N), V(LV,N), W(LW),
     +                 X(N)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             X02AJF
      EXTERNAL         X02AJF
*     .. External Subroutines ..
      EXTERNAL         E04GDF, E04YAF, LSQFUN, LSQGRD, LSQMON
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Common blocks ..
      COMMON           Y, T
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04GDF Example Program Results'
```

```
*     Skip heading in data file
      READ (NIN,*)
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
      DO 20 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
   20 CONTINUE
*     Check LSQFUN by calling E04YAF at an arbitrary point. Since
*     E04YAF only checks the derivatives calculated when IFLAG = 2,
*     a separate program should be run before using E04YAF or
*     E04GDF to check that LSQFUN gives the same values for the
*     elements of FJACC when IFLAG is set to 1 as when IFLAG is
*     set to 2.
      X(1) = 0.19e0
      X(2) = -1.34e0
      X(3) = 0.88e0
      IFAIL = 0
*
      CALL E04YAF(M,N,LSQFUN,X,FVEC,FJAC,LJ,IW,LIW,W,LW,IFAIL)
*
*     Continue setting parameters for E04GDF
*     * Set IPRINT to 1 to obtain output from LSQMON at each iteration *
      IPRINT = -1
      MAXCAL = 50*N
      ETA = 0.9e0
      XTOL = 10.0e0*SQRT(X02AJF())
*     We estimate that the minimum will be within 10 units of the
*     starting point
      STEPMX = 10.0e0
*     Set up the starting point
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
      IFAIL = 1
*
      CALL E04GDF(M,N,LSQFUN,LSQMON,IPRINT,MAXCAL,ETA,XTOL,STEPMX,X,
     +            FSUMSQ,FVEC,FJAC,LJ,S,V,LV,NITER,NF,IW,LIW,W,LW,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
         CALL LSQGRD(M,N,FVEC,FJAC,LJ,G)
         WRITE (NOUT,99997) 'The corresponding gradient is',
     +     (G(J),J=1,N)
         WRITE (NOUT,*) '                                (machine dependent)'
         WRITE (NOUT,*) 'and the residuals are'
         DO 40 I = 1, M
            WRITE (NOUT,99996) FVEC(I)
   40    CONTINUE
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
99997 FORMAT (1X,A,1P,3e12.3)
99996 FORMAT (1X,1P,e9.1)
      END
*
```

```
      SUBROUTINE LSQFUN(IFLAG,M,N,XC,FVECC,FJACC,LJC,IW,LIW,W,LW)
*     Routine to evaluate the residuals and their 1st derivatives.
*     A COMMON variable could be updated here to count the
*     number of calls of LSQFUN with IFLAG set to 1 (since NF
*     in LSQMON only counts calls with IFLAG set to 2)
*     .. Parameters ..
      INTEGER           MDEC, NT
      PARAMETER         (MDEC=15,NT=3)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LIW, LJC, LW, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Arrays in Common ..
      real              T(MDEC,NT), Y(MDEC)
*     .. Local Scalars ..
      real              DENOM, DUMMY
      INTEGER           I
*     .. Common blocks ..
      COMMON            Y, T
*     .. Executable Statements ..
      DO 20 I = 1, M
         DENOM = XC(2)*T(I,2) + XC(3)*T(I,3)
         IF (IFLAG.EQ.2) FVECC(I) = XC(1) + T(I,1)/DENOM - Y(I)
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = T(I,1)*T(I,2)*DUMMY
         FJACC(I,3) = T(I,1)*T(I,3)*DUMMY
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE LSQMON(M,N,XC,FVECC,FJACC,LJC,S,IGRADE,NITER,NF,IW,LIW,
     +                  W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER           NDEC
      PARAMETER         (NDEC=3)
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      INTEGER           IGRADE, LIW, LJC, LW, M, N, NF, NITER
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), S(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Local Scalars ..
      real              FSUMSQ, GTG
      INTEGER           J
*     .. Local Arrays ..
      real              G(NDEC)
*     .. External Functions ..
      real              F06EAF
      EXTERNAL          F06EAF
*     .. External Subroutines ..
      EXTERNAL          LSQGRD
*     .. Executable Statements ..
      FSUMSQ = F06EAF(M,FVECC,1,FVECC,1)
      CALL LSQGRD(M,N,FVECC,FJACC,LJC,G)
      GTG = F06EAF(N,G,1,G,1)
```

```
*     A COMMON variable giving the number of calls of
*     LSQFUN with IFLAG set to 1 could be printed here
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  ' Itns    F evals          SUMSQ             GTG        grade'
      WRITE (NOUT,99999) NITER, NF, FSUMSQ, GTG, IGRADE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '       X                   G           Singular values'
      DO 20 J = 1, N
         WRITE (NOUT,99998) XC(J), G(J), S(J)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,I4,6X,I5,6X,1P,e13.5,6X,1P,e9.1,6X,I3)
99998 FORMAT (1X,1P,e13.5,10X,1P,e9.1,10X,1P,e9.1)
      END
*
      SUBROUTINE LSQGRD(M,N,FVECC,FJACC,LJC,G)
*     Routine to evaluate gradient of the sum of squares
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), G(N)
*     .. Local Scalars ..
      real              SUM
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 J = 1, N
         SUM = 0.0e0
         DO 20 I = 1, M
            SUM = SUM + FJACC(I,J)*FVECC(I)
   20    CONTINUE
         G(J) = SUM + SUM
   40 CONTINUE
      RETURN
      END
```

9.2. Program Data

```
E04GDF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3. Program Results

```
E04GDF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
The corresponding gradient is  -6.060E-12   9.031E-11   9.385E-11
                            (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

With IPRINT set to 1 in the example program, intermediate results similar to those below are obtained form LSQMON:

```
E04GDF Example Program Results

 Itns     F evals          SUMSQ             GTG         grade
   0           1       1.02104E+01         1.0E+03         3

       X                    G            Singular values
 5.00000E-01             2.1E+01             5.0E+00
 1.00000E+00            -1.7E+01             2.6E+00
 1.50000E+00            -1.6E+01             9.6E-02

 Itns     F evals          SUMSQ             GTG         grade
   1           2       1.98730E-01         8.1E+00         3

       X                    G            Singular values
 8.24763E-02             1.9E+00             4.2E+00
 1.13575E+00            -1.5E+00             1.8E+00
 2.06664E+00            -1.5E+00             6.6E-02

 Itns     F evals          SUMSQ             GTG         grade
   2           3       9.23238E-03         3.6E-02         3

       X                    G            Singular values
 8.24402E-02             1.4E-01             4.1E+00
 1.13805E+00            -9.5E-02             1.6E+00
 2.31707E+00            -9.5E-02             6.1E-02

 Itns     F evals          SUMSQ             GTG         grade
   3           4       8.21492E-03         1.3E-06         3

       X                    G            Singular values
 8.24150E-02             8.2E-04             4.1E+00
 1.13323E+00            -5.8E-04             1.6E+00
 2.34337E+00            -5.8E-04             6.1E-02
```

```
 Itns     F evals          SUMSQ             GTG         grade
   4           6       8.21488E-03        2.5E-15         2

       X                   G           Singular values
 8.24107E-02           3.4E-08            4.1E+00
 1.13304E+00           8.9E-09            1.6E+00
 2.34369E+00          -3.5E-08            6.1E-02

 Itns     F evals          SUMSQ             GTG         grade
   5          10       8.21488E-03        1.7E-20         0

       X                   G           Singular values
 8.24106E-02          -6.1E-12            4.1E+00
 1.13304E+00           9.0E-11            1.6E+00
 2.34370E+00           9.4E-11            6.1E-02

On exit, the sum of squares is      0.0082
at the point       0.0824      1.1330      2.3437
The corresponding gradient is  -6.060E-12   9.031E-11   9.385E-11
                             (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

E04GYF – NAG Fortran Library Routine Document

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

E04GYF is an easy-to-use quasi-Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First derivatives are required.

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 E04GYF(M, N, LSFUN2, X, FSUMSQ, W, LW, IUSER, USER,
1                 IFAIL)
INTEGER           M, N, LW, IUSER(*), IFAIL
real              X(N), FSUMSQ, W(LW), USER(*)
EXTERNAL          LSFUN2
```

3 Description

This routine is similar to the subroutine LSFDQ2 in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to evaluate the residuals and their first derivatives at any point x.

Before attempting to minimize the sum of squares, the algorithm checks the user-supplied routine for consistency. Then, from a starting point supplied by the user, a sequence of points is generated which is intended to converge to a local minimum of the sum of squares. These points are generated using estimates of the curvature of $F(x)$.

4 References

[1] Gill P E and Murray W (1978) Algorithms for the solution of the nonlinear least-squares problem *SIAM J. Numer. Anal.* **15** 977–992

5 Parameters

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \text{N} \leq \text{M}$.

3: LSFUN2 — SUBROUTINE, supplied by the user. *External Procedure*

This routine must be supplied by the user to calculate the vector of values $f_i(x)$ and the Jacobian matrix of first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04GYF (see the the Chapter Introduction).

Its specification is:

```
SUBROUTINE LSFUN2(M, N, XC, FVECC, FJACC, LJC, IUSER, USER)
INTEGER           M, N, LJC, IUSER(*)
real              XC(N), FVECC(M), FJACC(LJC,N), USER(*)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) — ***real*** array *Input*

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

4: FVECC(M) — ***real*** array *Output*

On exit: FVECC(i) must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

5: FJACC(LJC,N) — ***real*** array *Output*

On exit: FJACC(i, j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

6: LJC — INTEGER *Input*

On entry: the first dimension of the array FJACC.

7: IUSER(*) — INTEGER array *User Workspace*

8: USER(*) — ***real*** array *User Workspace*

LSFUN2 is called from E04GYF with the parameters IUSER and USER as supplied to E04GYF. The user is free to use the arrays IUSER and USER to supply information to LSFUN2 as an alternative to using COMMON.

LSFUN2 must be declared as EXTERNAL in the (sub)program from which E04GYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: 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 first derivatives calculated by LSFUN2 at the starting point, and so is more likely to detect an error in the user's routine 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.

5: FSUMSQ — ***real*** *Output*

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

6: W(LW) — ***real*** array *Workspace*

7: LW — INTEGER *Input*

On entry: the length of W as declared in the (sub)program from which E04GYF is called.

Constraints:

LW $\geq 8 \times$ N $+ 2 \times$ N $\times$ N $+ 2 \times$ M $\times$ N $+ 3 \times$ M, if N > 1,
LW $\geq 11 + 5 \times$ M, if N $= 1$.

8: IUSER(*) — INTEGER array *User Workspace*

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

IUSER is not used by E04GYF, but is passed directly to LSFUN2 and may be used to pass information to those routines.

9: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04GYF, but is passed directly to LSFUN2 and may be used to pass information to those routines.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to −1 before entry. **It is then 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 specified by the routine:

IFAIL = 1

On entry, N < 1,
or M < N,
or LW < 8 × N + 2 × N × N + 2 × M × N + 3 × M, when N > 1,
or LW < 11 + 5 × M, when N = 1.

IFAIL = 2

There have been 50 × n calls of LSFUN2, yet the algorithm does not seem to have converged. This may be due to an awkward function or to a poor starting point, so it is worth restarting E04GYF from the final point held in X.

IFAIL = 3

The final point does not satisfy the conditions for acceptance as a minimum, but no lower point could be found.

IFAIL = 4

An auxiliary routine has been unable to complete a singular value decomposition in a reasonable number of sub-iterations.

IFAIL = 5, 6, 7 and 8

There is some doubt about whether the point X found by E04GYF is a minimum of $F(x)$. 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

It is very likely that the user has made an error in forming the derivatives $\frac{\partial f_i}{\partial x_j}$ in LSFUN2.

If the user is not satisfied with the result (e.g. because IFAIL lies between 3 and 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. Repeated failure may indicate some defect in the formulation of the problem.

7 Accuracy

If the problem is reasonably well scaled and a successful exit is made then, for a computer with a mantissa of t decimals, one would expect to get $t/2 - 1$ decimals accuracy in the components of x and between $t - 1$ (if $F(x)$ is of order 1 at the minimum) and $2t - 2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

8 Further Comments

The number of iterations required depends on the number of variables, the number of residuals and their behaviour, and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04GYF varies, but for $m \gg n$ is approximately $n \times m^2 + \mathrm{O}(n^3)$. In addition, each iteration makes at least one call of LSFUN2. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSFUN2.

Ideally the problem should be scaled so that the minimum value of the sum of squares is in the range $(0, 1)$ and so that at points a unit distance away from the solution the sum of squares 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 E04GYF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in segments of the workspace array W. See E04YCF for further details.

9 Example

To find the least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04GYF Example Program Text.
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          M, N, NT, LW
      PARAMETER        (M=15,N=3,NT=3,LW=8*N+2*N*N+2*M*N+3*M)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FSUMSQ
      INTEGER          I, IFAIL, J, K
*     .. Local Arrays ..
      real             T(M,NT), USER(M+M*NT), W(LW), X(N), Y(M)
      INTEGER          IUSER(1)
*     .. External Subroutines ..
      EXTERNAL         E04GYF, LSFUN2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04GYF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
*
      IUSER(1) = NT
      K = M
      DO 40 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
         USER(I) = Y(I)
         DO 20 J = 1, NT
            USER(K+J) = T(I,J)
   20    CONTINUE
         K = K + NT
   40 CONTINUE
*
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
*
      IFAIL = 1
*
      CALL E04GYF(M,N,LSFUN2,X,FSUMSQ,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
      END IF
      STOP
*
```

```
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
      END
*
      SUBROUTINE LSFUN2(M,N,XC,FVECC,FJACC,LJC,IUSER,USER)
*     Routine to evaluate the residuals and their 1st derivatives.
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), USER(*), XC(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      real              DENOM, DUMMY
      INTEGER           I, K
*     .. Executable Statements ..
      K = M
      DO 20 I = 1, M
         DENOM = XC(2)*USER(K+2) + XC(3)*USER(K+3)
         FVECC(I) = XC(1) + USER(K+1)/DENOM - USER(I)
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = USER(K+1)*USER(K+2)*DUMMY
         FJACC(I,3) = USER(K+1)*USER(K+3)*DUMMY
         K = K + IUSER(1)
   20 CONTINUE
      RETURN
      END
```

9.2 Program Data

```
E04GYF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3 Program Results

```
E04GYF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
```

E04GZF – NAG Fortran Library Routine Document

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

E04GZF is an easy-to-use modified Gauss–Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First derivatives are required.

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 E04GZF(M, N, LSFUN2, X, FSUMSQ, W, LW, IUSER, USER,
1                 IFAIL)
INTEGER           M, N, LW, IUSER(*), IFAIL
real              X(N), FSUMSQ, W(LW), USER(*)
EXTERNAL          LSFUN2
```

3 Description

This routine is similar to the subroutine LSFDN2 in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to evaluate the residuals and their first derivatives at any point x.

Before attempting to minimize the sum of squares, the algorithm checks the user-supplied routine for consistency. Then, from a starting point supplied by the user, a sequence of points is generated which is intended to converge to a local minimum of the sum of squares. These points are generated using estimates of the curvature of $F(x)$.

4 References

[1] Gill P E and Murray W (1978) Algorithms for the solution of the nonlinear least-squares problem *SIAM J. Numer. Anal.* **15** 977–992

5 Parameters

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \text{N} \leq \text{M}$.

3: LSFUN2 — SUBROUTINE, supplied by the user. *External Procedure*

This routine must be supplied by the user to calculate the vector of values $f_i(x)$ and the Jacobian matrix of first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04GZF.

Its specification is:

```
SUBROUTINE LSFUN2(M, N, XC, FVECC, FJACC, LJC, IUSER, USER)
INTEGER           M, N, LJC, IUSER(*)
real              XC(N), FVECC(M), FJACC(LJC,N), USER(*)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) — ***real*** array *Input*

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

4: FVECC(M) — ***real*** array *Output*

On exit: FVECC(i) must be set to the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

5: FJACC(LJC,N) — ***real*** array *Output*

On exit: FJACC(i, j) must be set to the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

6: LJC — INTEGER *Input*

On entry: the first dimension of the array FJACC.

7: IUSER(*) — INTEGER array *User Workspace*

8: USER(*) — ***real*** array *User Workspace*

LSFUN2 is called from E04GZF with the parameters IUSER and USER as supplied to E04GZF. The user is free to use the arrays IUSER and USER to supply information to LSFUN2 as an alternative to using COMMON.

LSFUN2 must be declared as EXTERNAL in the (sub)program from which E04GZF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: 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 first derivatives calculated by LSFUN2 at the starting point, and so is more likely to detect any error in the user's routines 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.

5: FSUMSQ — ***real*** *Output*

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

6: W(LW) — ***real*** array *Workspace*

7: LW — INTEGER *Input*

On entry: the length of W as declared in the (sub)program from which E04GZF is called.

Constraints:

LW $\geq 8 \times$ N $+ 2 \times$ N $\times$ N $+ 2 \times$ M $\times$ N $+ 3 \times$ M, if N > 1,

LW $\geq 11 + 5 \times$ M, if N $= 1$.

8: IUSER(*) — INTEGER array *User Workspace*

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

IUSER is not used by E04GZF, but is passed directly to LSFUN2 and may be used to pass information to those routines.

9: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04GZF, but is passed directly to LSFUN2 and may be used to pass information to those routines.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to −1 before entry. **It is then 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 specified by the routine:

IFAIL = 1

On entry, N < 1,
or M < N,
or LW < 8 × N + 2 × N × N + 2 × M × N + 3 × M, when N > 1,
or LW < 11 + 5 × M, when N = 1.

IFAIL = 2

There have been $50 \times n$ calls of LSFUN2, yet the algorithm does not seem to have converged. This may be due to an awkward function or to a poor starting point, so it is worth restarting E04GZF from the final point held in X.

IFAIL = 3

The final point does not satisfy the conditions for acceptance as a minimum, but no lower point could be found.

IFAIL = 4

An auxiliary routine has been unable to complete a singular value decomposition in a reasonable number of sub-iterations.

IFAIL = 5, 6, 7 and 8

There is some doubt about whether the point x found by E04GZF is a minimum of $F(x)$. 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

It is very likely that the user has made an error in forming the derivatives $\frac{\partial f_i}{\partial x_j}$ in LSFUN2.

If the user is not satisfied with the result (e.g. because IFAIL lies between 3 and 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. Repeated failure may indicate some defect in the formulation of the problem.

7 Accuracy

If the problem is reasonably well scaled and 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 the components of x and between $t - 1$ (if $F(x)$ is of order 1 at the minimum) and $2t - 2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

8 Further Comments

The number of iterations required depends on the number of variables, the number of residuals and their behaviour, and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04GZF varies, but for $m \gg n$ is approximately $n \times m^2 + \mathrm{O}(n^3)$. In addition, each iteration makes at least one call of LSFUN2. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSFUN2.

Ideally, the problem should be scaled so that the minimum value of the sum of squares is in the range $(0, +1)$, and so that at points a unit distance away from the solution the sum of squares is aproximately 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 E04GZF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in segments of the workspace array W. See E04YCF for further details.

9 Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04GZF Example Program Text.
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          M, N, NT, LW
      PARAMETER        (M=15,N=3,NT=3,LW=8*N+2*N*N+2*M*N+3*M)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             FSUMSQ
      INTEGER          I, IFAIL, J, K
*     .. Local Arrays ..
      real             T(M,NT), USER(M+M*NT), W(LW), X(N), Y(M)
      INTEGER          IUSER(1)
*     .. External Subroutines ..
      EXTERNAL         E04GZF, LSFUN3
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04GZF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
*
      IUSER(1) = NT
      K = M
      DO 40 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
         USER(I) = Y(I)
         DO 20 J = 1, NT
            USER(K+J) = T(I,J)
   20    CONTINUE
         K = K + NT
   40 CONTINUE
*
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
*
      IFAIL = 1
*
      CALL E04GZF(M,N,LSFUN3,X,FSUMSQ,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
      END IF
      STOP
*
```

```
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
      END
*
      SUBROUTINE LSFUN3(M,N,XC,FVECC,FJACC,LJC,IUSER,USER)
*     Routine to evaluate the residuals and their 1st derivatives.
*      DOUBLE PRECISION  T(MDEC,NT), Y(MDEC)
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), USER(*), XC(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      real              DENOM, DUMMY
      INTEGER           I, K
*     .. Executable Statements ..
      K = M
      DO 20 I = 1, M
         DENOM = XC(2)*USER(K+2) + XC(3)*USER(K+3)
         FVECC(I) = XC(1) + USER(K+1)/DENOM - USER(I)
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = USER(K+1)*USER(K+2)*DUMMY
         FJACC(I,3) = USER(K+1)*USER(K+3)*DUMMY
         K = K + IUSER(1)
   20 CONTINUE
      RETURN
      END
```

9.2 Program Data

```
E04GZF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3 Program Results

```
E04GZF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
```

E04HCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04HCF checks that a user-supplied routine for evaluating an objective function and its first derivatives produces derivative values which are consistent with the function values calculated.

2. Specification

```
SUBROUTINE E04HCF (N, FUNCT, X, F, G, IW, LIW, W, LW, IFAIL)
INTEGER          N, IW(LIW), LIW, LW, IFAIL
real             X(N), F, G(N), W(LW)
EXTERNAL         FUNCT
```

3. Description

Routines for minimizing a function of several variables may require the user to supply a subroutine to evaluate the objective function $F(x_1,x_2,...,x_n)$ and its first derivatives. E04HCF is designed to check the derivatives calculated by such user-supplied routines. As well as the routine to be checked (FUNCT), the user must supply a point $x = (x_1,x_2,...,x_n)^T$ at which the check will be made. Note that E04HCF checks routines of the form required for E04KDF and E04LBF.

E04HCF first calls FUNCT to evaluate F and its first derivatives $g_j = \dfrac{\partial F}{\partial x_j}$, for $j = 1,2,...,n$ at x.

The components of the user-supplied derivatives along two orthogonal directions (defined by unit vectors p_1 and p_2, say) are then calculated; these will be g^Tp_1 and g^Tp_2 respectively. The same components are also estimated by finite differences, giving quantities

$$v_k = \frac{F(x+hp_k) - F(x)}{h}, \qquad k = 1,2$$

where h is a small positive scalar. If the relative difference between v_1 and g^Tp_1 or between v_2 and g^Tp_2 is judged too large, an error indicator is set.

4. References

None.

5. Parameters

1: N – INTEGER. *Input*

On entry: the number n of independent variables in the objective function.

Constraint: N $\geq$ 1.

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

FUNCT must evaluate the function and its first derivatives at a given point. (The minimization routines mentioned in Section 3 give the user the option of resetting a parameter of FUNCT to cause the minimization process to terminate immediately. E04HCF will also terminate immediately, without finishing the checking process, if the parameter in question is reset.)

Its specification is:

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

1: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG will be set to 2.

On exit: if the user resets IFLAG to a negative number in FUNCT and returns control to E04HCF, E04HCF will terminate immediately with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FC – ***real***. *Output*

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

5: GC(N) – ***real*** array. *Output*

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

6: IW(LIW) – INTEGER array. *Workspace*

7: LIW – INTEGER. *Input*

8: W(LW) – ***real*** array. *Workspace*

9: LW – INTEGER. *Input*

These parameters are present so that FUNCT will be of the form required by the minimization routines mentioned in Section 3. FUNCT is called with E04HCF's parameters IW, LIW, W, LW as these parameters. If the advice given in the minimization routine documents is being followed, the user will have no reason to examine or change any elements of IW or W. In any case, FUNCT **must not change** the first 3×N elements of W.

The actual parameter used as FUNCT must be declared as EXTERNAL in the (sub)program from which E04HCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: X(N) – ***real*** array. *Input*

On entry: X(j), for $j = 1,2,...,n$ must be set to the co-ordinates of a suitable point at which to check the derivatives calculated by FUNCT. 'Obvious' settings, such as 0.0 or 1.0, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors could go undetected. Similarly, it is preferable that no two elements of X should be the same.

4: F – ***real***. *Output*

On exit: unless the user sets IFLAG negative in the first call of FUNCT, F contains the value of the objective function $F(x)$ at the point given by the user in X.

5: G(N) – ***real*** array. *Output*

On exit: unless the user sets IFLAG negative in the first call of FUNCT, G(j) contains the value of the derivative $\frac{\partial F}{\partial x_j}$ at the point given in X, as calculated by FUNCT, for $j = 1,2,...,n$.

6: IW(LIW) – INTEGER array. *Workspace*

This array is in the parameter list so that it can be used by other library routines for passing INTEGER quantities to FUNCT. It is not examined or changed by E04HCF. The general user must provide an array IW but is advised not to use it.

7: LIW – INTEGER. *Input*

On entry: the length of the array IW as declared in the (sub)program from which E04HCF is called.

Constraint: LIW ≥ 1.

8: W(LW) – ***real*** array. *Workspace*

9: LW – INTEGER. *Input*

On entry: the length of the array W as declared in the (sub)program from which E04HCF is called.

Constraint: LW ≥ 3×N.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04HCF because the user has set IFLAG negative in FUNCT. The setting of IFAIL will be the same as the user's setting of IFLAG. The check on FUNCT will not have been completed.

IFAIL = 1

On entry, N < 1,
or LIW < 1,
or LW < 3×N.

IFAIL = 2

The user should check carefully the derivation and programming of expressions for the derivatives of $F(x)$, because it is very unlikely that FUNCT is calculating them correctly.

7. Accuracy

IFAIL is set to 2 if

$$(v_k - g^T p_k)^2 \geq h \times ((g^T p_k)^2 + 1)$$

for $k = 1$ or 2. (See Section 3 for definitions of the quantities involved.) The scalar h is set equal to $\sqrt{\varepsilon}$, where ε is the ***machine precision*** as given by X02AJF.

8. Further Comments

The user-supplied routine FUNCT is called 3 times.

Before using E04HCF to check the calculation of first derivatives, the user should be confident that FUNCT is calculating F correctly. The usual way of checking the calculation of the function is to compare values of $F(x)$ calculated by FUNCT at non-trivial points x with values calculated independently. ('Non-trivial' means that, as when setting x before calling E04HCF, co-ordinates such as 0.0 or 1.0 should be avoided.)

E04HCF only checks the derivatives calculated by a user-supplied routine when IFLAG = 2. So, if FUNCT is intended for use in conjunction with a minimization routine which may set IFLAG to 1, the user must check that, for given settings of the XC(j), FUNCT produces the same values for the GC(j) when IFLAG is set to 1 as when IFLAG is set to 2.

9. Example

Suppose that it is intended to use E04KDF to minimize

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

The following program could be used to check the first derivatives calculated by the routine FUNCT. (The tests of whether IFLAG = 0 or 1 in FUNCT are present ready for when FUNCT is called by E04KDF. E04HCF will always call FUNCT with IFLAG set to 2.)

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.

```
*     E04HCF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LIW, LW
      PARAMETER        (N=4,LIW=1,LW=3*N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             F
      INTEGER          IFAIL, J
*     .. Local Arrays ..
      real             G(N), W(LW), X(N)
      INTEGER          IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         E04HCF, FUNCT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04HCF Example Program Results'
*     Set up an arbitrary point at which to check the 1st derivatives
      X(1) = 1.46e0
      X(2) = -0.82e0
      X(3) = 0.57e0
      X(4) = 1.21e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The test point is'
      WRITE (NOUT,99999) (X(J),J=1,N)
      IFAIL = 1
*
      CALL E04HCF(N,FUNCT,X,F,G,IW,LIW,W,LW,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.LT.0) THEN
         WRITE (NOUT,99998) 'IFLAG was set to ', IFAIL, 'in FUNCT'
      ELSE IF (IFAIL.EQ.1) THEN
         WRITE (NOUT,*) 'A parameter is outside its expected range'
      ELSE
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,*)
     +        '1st derivatives are consistent with function values'
         ELSE
```

```
            WRITE (NOUT,*)
     +        'Probable error in calculation of 1st derivatives'
          END IF
          WRITE (NOUT,*)
          WRITE (NOUT,99997)
     +      'At the test point, FUNCT gives the function value', F
          WRITE (NOUT,*) 'and the 1st derivatives'
          WRITE (NOUT,99996) (G(J),J=1,N)
        END IF
        STOP
*
99999 FORMAT (1X,4F10.4)
99998 FORMAT (1X,A,I3,A)
99997 FORMAT (1X,A,1P,e12.4)
99996 FORMAT (1X,1P,4e12.3)
      END
*
      SUBROUTINE FUNCT(IFLAG,N,XC,FC,GC,IW,LIW,W,LW)
*     Routine to evaluate objective function and its 1st derivatives.
*     .. Scalar Arguments ..
      real                FC
      INTEGER             IFLAG, LIW, LW, N
*     .. Array Arguments ..
      real                GC(N), W(LW), XC(N)
      INTEGER             IW(LIW)
*     .. Executable Statements ..
      IF (IFLAG.NE.1) THEN
         FC = (XC(1)+10.0e0*XC(2))**2 + 5.0e0*(XC(3)-XC(4))**2 + (XC(2)
     +        -2.0e0*XC(3))**4 + 10.0e0*(XC(1)-XC(4))**4
      END IF
      IF (IFLAG.NE.0) THEN
         GC(1) = 2.0e0*(XC(1)+10.0e0*XC(2)) + 40.0e0*(XC(1)-XC(4))**3
         GC(2) = 20.0e0*(XC(1)+10.0e0*XC(2)) + 4.0e0*(XC(2)-2.0e0*XC(3))
     +           **3
         GC(3) = 10.0e0*(XC(3)-XC(4)) - 8.0e0*(XC(2)-2.0e0*XC(3))**3
         GC(4) = 10.0e0*(XC(4)-XC(3)) - 40.0e0*(XC(1)-XC(4))**3
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 E04HCF Example Program Results

 The test point is
     1.4600   -0.8200    0.5700    1.2100

 1st derivatives are consistent with function values

 At the test point, FUNCT gives the function value  6.2273E+01
 and the 1st derivatives
   -1.285E+01  -1.649E+02   5.384E+01   5.775E+00
```

E04HDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04HDF checks that a user-supplied routine for calculating second derivatives of an objective function is consistent with a user-supplied routine for calculating the corresponding first derivatives.

2. Specification

```
SUBROUTINE E04HDF (N, FUNCT, HESS, X, G, HESL, LH, HESD, IW, LIW, W,
1                  LW, IFAIL)
INTEGER        N, LH, IW(LIW), LIW, LW, IFAIL
real           X(N), G(N), HESL(LH), HESD(N), W(LW)
EXTERNAL       FUNCT, HESS
```

3. Description

Routines for minimizing a function $F(x_1,x_2,...,x_n)$ of the variables $x_1,x_2,...,x_n$ may require the user to provide a subroutine to evaluate the second derivatives of F. E04HDF is designed to check the second derivatives calculated by such user-supplied routines. As well as the routine to be checked (HESS), the user must supply a routine (FUNCT) to evaluate the first derivatives, and a point $x = (x_1,x_2,...,x_n)^T$ at which the checks will be made. Note that E04HDF checks routines of the form required for E04LBF.

E04HDF first calls FUNCT and HESS to evaluate the first and second derivatives of F at x. The user-supplied Hessian matrix (H, say) is projected onto two orthogonal vectors y and z to give the scalars y^THy and z^THz respectively. The same projections of the Hessian matrix are also estimated by finite differences, giving

$$p = (y^Tg(x+hy)-y^Tg(x))/h$$

and $$q = (z^Tg(x+hz)-z^Tg(x))/h$$

respectively, where $g(\ )$ denotes the vector of first derivatives at the point in brackets and h is a small positive scalar. If the relative difference between p and y^THy or between q and z^THz is judged too large, an error indicator is set.

4. References

None.

5. Parameters

1: N – INTEGER. *Input*

On entry: the number n of independent variables in the objective function.

Constraint: N $\geq$ 1.

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

FUNCT must evaluate the function and its first derivatives at a given point. (E04LBF gives the user the option of resetting a parameter of FUNCT to cause the minimization process to terminate immediately. E04HDF will also terminate immediately, without finishing the checking process, if the parameter in question is reset.)

Its specification is:

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

1: IFLAG – INTEGER. *Input/Output*

On entry: to FUNCT, IFLAG will be set to 2.

On exit: if the user sets IFLAG to some negative number in FUNCT and returns control to E04HDF, E04HDF will terminate immediately with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FC – ***real***. *Output*

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

5: GC(N) – ***real*** array. *Output*

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

6: IW(LIW) – INTEGER array. *Workspace*

7: LIW – INTEGER. *Input*

8: W(LW) – ***real*** array. *Workspace*

9: LW – INTEGER. *Input*

These parameters are present so that FUNCT will be of the form required by E04LBF. FUNCT is called with E04HDF's parameters IW, LIW, W, LW as these parameters. If the advice given in E04LBF routine document is being followed, the user will have no reason to examine or change any elements of IW or W. In any case, FUNCT **must not change** the first $5 \times n$ elements of W.

E04HCF should be used to check the first derivatives calculated by FUNCT before E04HDF is used to check the second derivatives, since E04HDF assumes that the first derivatives are correct. The actual parameter used as FUNCT must be declared as EXTERNAL in the (sub)program from which E04HDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

HESS must evaluate the second derivatives of the function at a given point. (As with FUNCT, a parameter can be set to cause immediate termination.)

Its specification is:

```
SUBROUTINE HESS(IFLAG, N, XC, FHESL, LH, FHESD, IW, LIW, W, LW)
INTEGER    IFLAG, N, LH, IW(LIW), LIW, LW
real       XC(N), FHESL(LH), FHESD(N), W(LW)
```

1: IFLAG – INTEGER. *Input/Output*

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

On exit: if HESS resets IFLAG to a negative number, E04HDF will terminate immediately with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FHESL(LH) – ***real*** array. *Output*

On exit: unless IFLAG is reset, HESS 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. FHESL($(i-1)(i-2)/2+j$) must be set to the value of $\frac{\partial^2 F}{\partial x_i \partial x_j}$ at the point x, for $i = 2,3,...,n$; $j = 1,2,...,i-1$.

(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*** array. *Input/Output*

On entry: contains the value of $\frac{\partial F}{\partial x_j}$ at the point x, for $j = 1,2,...,n$. Routines written to take advantage of a similar feature of E04LBF can be tested as they stand by E04HDF.

On exit: unless IFLAG is reset, HESS must place the diagonal elements of the second derivative matrix of F (evaluated at the point x) in FHESD, i.e. FHESD(j) must be set to the value of $\frac{\partial^2 F}{\partial x_j^2}$ at the point x, for $j = 1,2,...,n$.

7: IW(LIW) – INTEGER array. *Workspace*

8: LIW – INTEGER. *Input*

9: W(LW) – ***real*** array. *Workspace*

10: LW – INTEGER. *Input*

As in FUNCT, these parameters correspond to the parameters IW, LIW, W and LW of E04HDF. HESS **must not change** the first 5×N elements of W.

HESS must be declared as EXTERNAL in the (sub)program from which E04HDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: X(N) – ***real*** array. *Input*

On entry: X(j), for $j = 1,2,...,n$ must contain the co-ordinates of a suitable point at which to check the derivatives calculated by FUNCT. 'Obvious' settings, such as 0.0 or 1.0, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors could go undetected. Similarly, it is advisable that no two elements of X should be the same.

5: G(N) – ***real*** array. *Output*

On exit: unless the user sets IFLAG negative in the first call of FUNCT, G(j) contains the value of the first derivative $\frac{\partial F}{\partial x_j}$ at the point given in X, as calculated by FUNCT, for $j = 1,2,...,N$.

6: HESL(LH) – ***real*** array. *Output*

On exit: unless the user sets IFLAG negative in HESS, HESL contains the strict lower triangle of the second derivative matrix of F, as evaluated by HESS at the point given in X, stored by rows.

7: LH – INTEGER. *Input*

On entry: the dimension of the array HESL as declared in the (sub)program from which E04HDF is called.

Constraint: LH ≥ max(1,N×(N−1)/2).

8: HESD(N) – ***real*** array. *Output*

On exit: unless the user set IFLAG negative in HESS, HESD contains the diagonal elements of the second derivative matrix of F, as evaluated by HESS at the point given in X.

9: IW(LIW) – INTEGER array. *Workspace*

This array is in the parameter list so that it can be used by other library routines for passing INTEGER quantities to FUNCT or HESS. It is not examined or changed by E04HDF. The general user must provide an array IW, but is advised not to use it.

10: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which E04HDF is called.

Constraint: LIW $\geq$ 1.

11: W(LW) – ***real*** array. *Workspace*

12: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E04HDF is called.

Constraint: LW $\geq$ 5×N.

13: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04HDF because the user has set IFLAG negative in FUNCT or HESS. The setting of IFLAG will be the same as the user's setting of IFLAG. The check on HESS will not have been completed.

IFAIL = 1

On entry, N < 1,
or LH < max(1,N×(N−1)/2),
or LIW < 1,
or LW < 5×N.

IFAIL = 2

The user should check carefully the derivation and programming of expressions for the second derivatives of $F(x)$, because it is very unlikely that HESS is calculating them correctly.

7. Accuracy

IFAIL is set to 2 if

$$|y^T H y - p| \geq \sqrt{h} \times (|y^T H y| + 1.0)$$
or $$|z^T H z - q| \geq \sqrt{h} \times (|z^T H z| + 1.0)$$

where h is set equal to $\sqrt{\varepsilon}$ (ε being the ***machine precision*** as given by X02AJF) and other quantities are as defined in Section 3.

8. Further Comments

E04HDF calls HESS once and FUNCT three times.

9. Example

Suppose that it is intended to use E04LBF to minimize

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

The following program could be used to check the second derivatives calculated by the routine HESS required. (The call of E04HDF is preceded by a call of E04HCF to check the routine FUNCT which calculates the first derivatives.)

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.

```
*     E04HDF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LH, LIW, LW
      PARAMETER        (N=4,LH=N*(N-1)/2,LIW=1,LW=5*N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             F
      INTEGER          I, IFAIL, J, K
*     .. Local Arrays ..
      real             G(N), HESD(N), HESL(LH), W(LW), X(N)
      INTEGER          IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         E04HCF, E04HDF, FUNCT, HESS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04HDF Example Program Results'
*     Set up an arbitrary point at which to check the derivatives
      X(1) = 1.46e0
      X(2) = -0.82e0
      X(3) = 0.57e0
      X(4) = 1.21e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The test point is'
      WRITE (NOUT,99999) (X(J),J=1,N)
*     Check the 1st derivatives
      IFAIL = 0
*
      CALL E04HCF(N,FUNCT,X,F,G,IW,LIW,W,LW,IFAIL)
*
*     Check the 2nd derivatives
      IFAIL = 1
*
      CALL E04HDF(N,FUNCT,HESS,X,G,HESL,LH,HESD,IW,LIW,W,LW,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.LT.0) THEN
         WRITE (NOUT,99998) 'IFLAG was set to ', IFAIL,
     +     'in FUNCT or HESS'
      ELSE IF (IFAIL.EQ.1) THEN
         WRITE (NOUT,*) 'A parameter is outside its expected range'
      ELSE
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,*)
     +        '2nd derivatives are consistent with 1st derivatives'
         ELSE IF (IFAIL.EQ.2) THEN
            WRITE (NOUT,*)
     +        'Probable error in calculation of 2nd derivatives'
         END IF
         WRITE (NOUT,*)
```

```
            WRITE (NOUT,99997)
     +        'At the test point, FUNCT gives the function value', F
            WRITE (NOUT,*) 'and the 1st derivatives'
            WRITE (NOUT,99996) (G(J),J=1,N)
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        'HESS gives the lower triangle of the Hessian matrix'
            WRITE (NOUT,99995) HESD(1)
            K = 1
            DO 20 I = 2, N
               WRITE (NOUT,99995) (HESL(J),J=K,K+I-2), HESD(I)
               K = K + I - 1
   20       CONTINUE
         END IF
         STOP
*
99999 FORMAT (1X,4F9.4)
99998 FORMAT (1X,A,I3,A)
99997 FORMAT (1X,A,1P,e12.4)
99996 FORMAT (1X,1P,4e12.3)
99995 FORMAT (1X,1P,4e12.3)
      END
*
      SUBROUTINE FUNCT(IFLAG,N,XC,FC,GC,IW,LIW,W,LW)
*     Routine to evaluate objective function and its 1st derivatives.
*     .. Scalar Arguments ..
      real              FC
      INTEGER           IFLAG, LIW, LW, N
*     .. Array Arguments ..
      real              GC(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Executable Statements ..
      FC = (XC(1)+10.0e0*XC(2))**2 + 5.0e0*(XC(3)-XC(4))**2 + (XC(2)
     +     -2.0e0*XC(3))**4 + 10.0e0*(XC(1)-XC(4))**4
      GC(1) = 2.0e0*(XC(1)+10.0e0*XC(2)) + 40.0e0*(XC(1)-XC(4))**3
      GC(2) = 20.0e0*(XC(1)+10.0e0*XC(2)) + 4.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(3) = 10.0e0*(XC(3)-XC(4)) - 8.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(4) = 10.0e0*(XC(4)-XC(3)) - 40.0e0*(XC(1)-XC(4))**3
      RETURN
      END
*
      SUBROUTINE HESS(IFLAG,N,XC,FHESL,LH,FHESD,IW,LIW,W,LW)
*     Routine to evaluate 2nd derivatives
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LH, LIW, LW, N
*     .. Array Arguments ..
      real              FHESD(N), FHESL(LH), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Executable Statements ..
      FHESD(1) = 2.0e0 + 120.0e0*(XC(1)-XC(4))**2
      FHESD(2) = 200.0e0 + 12.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESD(3) = 10.0e0 + 48.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESD(4) = 10.0e0 + 120.0e0*(XC(1)-XC(4))**2
      FHESL(1) = 20.0e0
      FHESL(2) = 0.0e0
      FHESL(3) = -24.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESL(4) = -120.0e0*(XC(1)-XC(4))**2
      FHESL(5) = 0.0e0
      FHESL(6) = -10.0e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
E04HDF Example Program Results

The test point is
   1.4600  -0.8200   0.5700   1.2100

2nd derivatives are consistent with 1st derivatives

At the test point, FUNCT gives the function value  6.2273E+01
and the 1st derivatives
  -1.285E+01  -1.649E+02   5.384E+01   5.775E+00

HESS gives the lower triangle of the Hessian matrix
   9.500E+00
   2.000E+01   2.461E+02
   0.000E+00  -9.220E+01   1.944E+02
  -7.500E+00   0.000E+00  -1.000E+01   1.750E+01
```

E04HEF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04HEF is a comprehensive modified Gauss-Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). 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 E04HEF (M, N, LSQFUN, LSQHES, LSQMON, IPRINT, MAXCAL, ETA,
     1                   XTOL, STEPMX, X, FSUMSQ, FVEC, FJAC, LJ, S, V,
     2                   LV, NITER, NF, IW, LIW, W, LW, IFAIL)
      INTEGER           M, N, IPRINT, MAXCAL, LJ, LV, NITER, NF,
     1                  IW(LIW), LIW, LW, IFAIL
      real              ETA, XTOL, STEPMX, X(N), FSUMSQ, FVEC(M),
     1                  FJAC(LJ,N), S(N), V(LV,N), W(LW)
      EXTERNAL          LSQFUN, LSQHES, LSQMON
```

3. Description

This routine is essentially identical to the subroutine LSQSDN in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form:

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply subroutines to calculate the values of the $f_i(x)$ and their first derivatives and second derivatives at any point x.

From a starting point $x^{(1)}$ supplied by the user, the routine generates a sequence of points $x^{(2)}, x^{(3)}, \ldots$, which is intended to converge to a local minimum of $F(x)$. The sequence of points is given by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$$

where the vector $p^{(k)}$ is a direction of search, and $\alpha^{(k)}$ is chosen such that $F(x^{(k)} + \alpha^{(k)} p^{(k)})$ is approximately a minimum with respect to $\alpha^{(k)}$.

The vector $p^{(k)}$ used depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced, then $p^{(k)}$ is the Gauss-Newton direction; otherwise the second derivatives of the $f_i(x)$ are taken into account.

The method is designed to ensure that steady progress is made whatever the starting point, and to have the rapid ultimate convergence of Newton's method.

4. References

[1] GILL, P.E. and MURRAY, W.
Algorithms for the Solution of the Nonlinear Least-squares Problem.
SIAM J. Numer. Anal., 15, pp. 977-992, 1978.

5. Parameters

1: M – INTEGER. *Input*
2: N – INTEGER. *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathrm{N} \leq \mathrm{M}$.

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

LSQFUN must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point x. (However, if the user does not wish to calculate the residuals or first derivatives at a particular x, there is the option of setting a parameter to cause E04HEF to terminate immediately.)

Its specification is:

```
SUBROUTINE LSQFUN (IFLAG, M, N, XC, FVECC, FJACC, LJC, IW, LIW, W, LW)
INTEGER    IFLAG, M, N, LJC, IW(LIW), LIW, LW
real       XC(N), FVECC(M), FJACC(LJC,N), W(LW)
```

Important: the dimension declaration FJACC must contain the variable LJC, not an integer constant.

1: IFLAG – INTEGER. *Input/Output*

On entry: to LSQFUN, IFLAG will be set to 2.

On exit: if it is not possible to evaluate the $f_i(x)$ or their first derivatives at the point given in XC (or if it wished to stop the calculations for any other reason), the user should reset IFLAG to some negative number and return control to E04HEF. E04HEF will then terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: M – INTEGER. *Input*
3: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

4: XC(N) – ***real*** array. *Input*

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

5: FVECC(M) – ***real*** array. *Output*

On exit: unless IFLAG is reset to a negative number, FVECC(i) must contain the value of f_i at the point x, for $i = 1,2,...,m$.

6: FJACC(LJC,N) – ***real*** array. *Output*

On exit: unless IFLAG is reset to a negative number, FJACC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1,2,...,m$; $j = 1,2,...,n$.

7: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

8: IW(LIW) – INTEGER array. *Workspace*
9: LIW – INTEGER. *Input*
10: W(LW) – ***real*** array. *Workspace*
11: LW – INTEGER. *Input*

LSQFUN is called with E04HEF's parameters IW, LIW, W, LW as these parameters. They are present so that, when other library routines require the solution of a minimization subproblem, constants needed for the evaluation of residuals can be passed through IW and W. Similarly, users could pass quantities

of LSQFUN from the segment which calls E04HEF by using partitions of IW and W beyond those used as workspace by E04HEF. However, because of the danger of mistakes in partitioning, it is recommended that users should pass information to LSQFUN via COMMON and **not use IW or W** at all. In any case users **must not change** the elements of IW and W used as workspace by E04HEF.

Note: LSQFUN should be tested separately before being used in conjunction with E04HEF.

LSQFUN must be declared as EXTERNAL in the (sub)program from which E04HEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

LSQHES must calculate the elements of the symmetric matrix

$$B(x) = \sum_{i=1}^{m} f_i(x)G_i(x),$$

at any point x, where $G_i(x)$ is the Hessian matrix of $f_i(x)$. (As with LSQFUN, there is the option of causing E04HEF to terminate immediately.)

Its specification is:

```
SUBROUTINE LSQHES (IFLAG, M, N, FVECC, XC, B, LB, IW, LIW, W, LW)
INTEGER     IFLAG, M, N, LB, IW(LIW), LIW, LW
real        FVECC(M), XC(N), B(LB), W(LW)
```

1: IFLAG – INTEGER. *Input/Output*

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

On exit: if LSQHES resets IFLAG to some negative number, E04HEF will terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: M – INTEGER. *Input*

3: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

4: FVECC(M) – ***real*** array. *Input*

On entry: the value of the residual f_i at the point x, for $i = 1,2,...,m$, so that the values of the f_i can be used in the calculation of the elements of B.

5: XC(N) – ***real*** array. *Input*

On entry: the point x at which the elements of B are to be evaluated.

6: B(LB) – ***real*** array. *Output*

On exit: unless IFLAG is reset to a negative number, B must contain the lower triangle of the matrix $B(x)$, evaluated at the point x, stored by rows. (The upper triangle is not required because the matrix is symmetric.) i.e. B$(j(j-1)/2+k)$ must contain $\sum_{i=1}^{m} f_i \frac{\partial^2 f_i}{\partial x_j \partial x_k}$ evaluated at the point x, for $j = 1,2,...,n$ and $k = 1,2,...,j$.

7: LB – INTEGER. *Input*

On entry: the length of the array B.

8: IW(LIW) – INTEGER array. *Workspace*

9: LIW – INTEGER. *Input*

10: W(LW) – ***real*** array. *Workspace*

11: LW – INTEGER. *Input*

As in LSQFUN, these parameters correspond to the parameters IW, LIW, W, LW of E04HEF. LSQHES **must not change** the sections of IW and W required as workspace by E04HEF. Again, it is recommended that the user should pass quantities to LSQHES via COMMON and not use IW or W at all.

Note: LSQHES should be tested separately before being used in conjuncton with E04HEF.

LSQHES must be declared as EXTERNAL in the (sub)program from which E04HEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: LSQMON – SUBROUTINE, supplied by the user. *External Procedure*

If IPRINT ≥ 0, the user must supply a subroutine LSQMON which is suitable for monitoring the minimization process. LSQMON must not change the values of any of its parameters.

If IPRINT < 0, the dummy routine E04FDZ can be used as LSQMON. (In some implementations the name of this routine is FDZE04; refer to the Users' Note for your implementation.)

Its specification is:

```
 SUBROUTINE LSQMON (M, N, XC, FVECC, FJACC, LJC, S, IGRADE, NITER, NF,
1           IW, LIW, W, LW)
 INTEGER    M, N, LJC, IGRADE, NITER, NF, IW(LIW), LIW, LW
 real       XC(N), FVECC(M), FJACC(LJC,N), S(N), W(LW)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M – INTEGER. *Input*

2: N – INTEGER. *Input*

On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

4: FVECC(M) – ***real*** array. *Input*

On entry: the values of the residuals f_i at the point x.

5: FJACC(LJC,N) – ***real*** array. *Input*

On entry: FJACC(i,j) contains the value of $\frac{\partial f_i}{\partial x_j}$ at the current point x, for $i = 1,2,...,m$; $j = 1,2,...,n$.

6: LJC – INTEGER. *Input*

On entry: the first dimension of the array FJACC.

7: S(N) – ***real*** array. *Input*

On entry: the singular values of the current Jacobian matrix. Thus S may be useful as information about the structure of the user's problem. (If IPRINT > 0, LSQMON is called at the initial point before the singular values have been calculated. So the elements of S are set to zero for the first call of LSQMON.)

8: IGRADE – INTEGER. *Input*

On entry: E04HEF estimates the dimension of the subspace for which the Jacobian matrix can be used as a valid approximation to the curvature (see Gill and Murray [1]). This estimate is called the grade of the Jacobian matrix, and IGRADE gives its current value.

9: NITER – INTEGER. *Input*

On entry: the number of iterations which have been performed in E04HEF.

10: NF – INTEGER. *Input*

On entry: the number of times that LSQFUN has been called so far. Thus NF gives the number of evaluations of the residuals and the Jacobian matrix.

11: IW(LIW) – INTEGER array. *Workspace*
12: LIW – INTEGER. *Input*
13: W(LW) – ***real*** array. *Workspace*
14: LW – INTEGER. *Input*

As in LSQFUN and LSQHES, these parameters correspond to the parameters IW, LIW, W, LW of E04HEF. They are included in LSQMON's parameter list primarily for when E04HEF is called by other library routines.

Note: the user should normally print the sum of squares of residuals, so as to be able to examine the sequence of values of $F(x)$ mentioned in Section 7. It is usually helpful to also print XC, the gradient of the sum of squares, NITER and NF.

LSQMON must be declared as EXTERNAL in the (sub)program from which E04HEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: IPRINT – INTEGER. *Input*

On entry: IPRINT specifies the frequency with which LSQMON is to be called. If IPRINT > 0, LSQMON is called once every IPRINT iterations and just before exit from E04HEF. If IPRINT = 0, LSQMON is just called at the final point. If IPRINT < 0, LSQMON is not called at all.

IPRINT should normally be set to a small positive number.

Suggested value: IPRINT = 1.

7: MAXCAL – INTEGER. *Input*

On entry: this parameter is present so as to enable the user to limit the number of times that LSQFUN is called by E04HEF. There will be an error exit (see Section 6) after MAXCAL calls of LSQFUN.

Suggested value: MAXCAL = 50×n.

Constraint: MAXCAL ≥ 1.

8: ETA – ***real***. *Input*

On entry: every iteration of E04HEF involves a linear minimization (i.e. minimization of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$). ETA must lie in the range 0.0 ≤ ETA < 1.0, and specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha^{(k)}$ 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 performed by E04HEF, they will increase the number of calls of LSQFUN made each iteration. On balance it is usually more efficient to perform a low accuracy minimization.

Suggested value: ETA = 0.5 (ETA = 0.0 if N = 1).

Constraint: 0.0 ≤ ETA < 1.0.

9: XTOL – ***real***. *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 prior to a normal exit, is such that

$$\|x_{sol}-x_{true}\| < \text{XTOL}\times(1.0+\|x_{true}\|),$$

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 = 1.0E−5, then x_{sol} is usually accurate to about 5 decimal places. (For further details see Section 7.)

If $F(x)$ and the variables are scaled roughly as described in Section 8 and ϵ is the ***machine precision***, then a setting of order XTOL = $\sqrt{\epsilon}$ will usually be appropriate. If XTOL is set to 0.0 or some positive value less than 10ϵ, E04HEF will use 10ϵ instead of XTOL, since 10ϵ is probably the smallest reasonable setting.

Constraint: XTOL $\geq$ 0.0.

10: STEPMX – ***real***. *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency, a slight overestimate is preferable.) E04HEF will ensure that, for each iteration

$$\sum_{j=1}^{n} (x_j^{(k)} - x_j^{(k-1)})^2 \leq (\text{STEPMX})^2,$$

where k is the iteration number. Thus, if the problem has more than one solution, E04HEF 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 help avoid 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.

11: 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,...,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.

12: FSUMSQ – ***real***. *Output*

On exit: the value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in X.

13: FVEC(M) – ***real*** array. *Output*

On exit: the value of the residual $f_i(x)$ at the final point in X, for i = 1,2,...,m.

14: FJAC(LJ,N) – ***real*** array. *Output*

On exit: the value of the first derivative $\dfrac{\partial f_i}{\partial x_j}$ evaluated at the final point given in X, for i = 1,2,...,m; j = 1,2,...,n.

15: LJ – INTEGER. *Input*

On entry: the dimension of the array FJAC as declared in the (sub)program from which E04HEF is called.

Constraint: LJ $\geq$ M.

16: S(N) – ***real*** array. *Output*

On exit: the singular values of the Jacobian matrix at the final point. Thus S may be useful as information about the structure of the user's problem.

17: V(LV,N) – ***real*** array. *Output*

On exit: the matrix V associated with the singular value decomposition

$$J = USV^T$$

of the Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalised eigenvectors of J^TJ.

18: LV – INTEGER. *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which E04HEF is called.

Constraint: LV ≥ N.

19: NITER – INTEGER. *Output*

On exit: the number of iterations which have been performed in E04HEF.

20: NF – INTEGER. *Output*

On exit: the number of times that the residuals and Jacobian matrix have been evaluated (i.e. number of calls of LSQFUN).

21: IW(LIW) – INTEGER array. *Workspace*

22: LIW – INTEGER. *Input*

On entry: the length of IW as declared in the (sub)program from which E04HEF is called.

Constraint: LIW ≥ 1.

23: W(LW) – ***real*** array. *Workspace*

24: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E04HEF is called.

Constraints: LW ≥ 7×N + 2×M×N + M + N×N, if N > 1,
LW ≥ 9 + 3×M, if N = 1.

25: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL < 0

A negative value of IFAIL indicates an exit from E04HEF because the user has set IFLAG negative in LSQFUN or LSQHES. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N < 1,
or M < N,
or MAXCAL < 1,
or ETA < 0.0,
or ETA ≥ 1.0,
or XTOL < 0.0,
or STEPMX < XTOL,
or LJ < M,
or LV < N,
or LIW < 1,

or LW < 7×N + M×N + 2×M + N^2, when N > 1,
or LW < 9 + 3×M, when N = 1.

When this exit occurs, no values will have been assigned to FSUMSQ, or to the elements of FVEC, FJAC, S or V.

IFAIL = 2

There have been MAXCAL calls of LSQFUN. If steady reductions in the sum of squares, $F(x)$, 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 satisified, but a lower point could not be found. This could be because XTOL has been set so small that rounding errors in the evaluation of the residuals and derivatives make attainment of the convergence conditions impossible.

IFAIL = 4

The method for computing the singular value decomposition of the Jacobian matrix has failed to converge in a reasonable number of sub-iterations. It may be worth applying E04HEF again starting with an initial approximation which is not too close to the point at which the failure occurred.

The values IFAIL = 2, 3 and 4 may also be caused by mistakes in LSQFUN or LSQHES, 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 E04HEF when the matrix of second derivatives of $F(x)$ is positive-definite, and when (B1, B2 and B3) or B4 or B5 hold, where

B1 ≡ $\alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\varepsilon) \times (1.0+\|x^{(k)}\|)$

B2 ≡ $|F^{(k)} - F^{(k-1)}| < (\text{XTOL}+\varepsilon)^2 \times (1.0+F^{(k)})$

B3 ≡ $\|g^{(k)}\| < \varepsilon^{\frac{1}{3}} \times (1.0+F^{(k)})$

B4 ≡ $F^{(k)} < \varepsilon^2$

B5 ≡ $\|g^{(k)}\| < \sqrt{\varepsilon \times \sqrt{F^{(k)}}}$

and where $\|.\|$ and ε are as defined in Section 5, and $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of first derivatives at $x^{(k)}$.

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

If IFAIL = 3, then x_{sol} may still be a good estimate of x_{true}, but to verify this the user should make the following checks. If

(1) the sequence $\{F(x^{(k)})\}$ converges to $F(x_{sol})$ at a superlinear or a fast linear rate, and
(2) $g(x_{sol})^T g(x_{sol}) < 10\varepsilon$,

where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum. When (2) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. The values of $F(x^{(k)})$ can be calculated in LSQMON, and the vector $g(x_{sol})$ can be calculated from the contents of FVEC and FJAC on exit from E04HEF.

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

8. Further Comments

The number of iterations required depends on the number of variables, the number of residuals, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04HEF varies, but for $m \gg n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least one call of LSQFUN, and some iterations may call LSQHES. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSQFUN (and, to a lesser extent, in LSQHES).

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 of $F(x)$ at the solution is well-conditioned. 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 E04HEF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in the arrays S and V. See E04YCF for further details.

9. Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table:

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

Before calling E04HEF, the program calls E04YAF and E04YBF to check LSQFUN and LSQHES. It uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04HEF Example Program Text.
*     Mark 15 Revised.  NAG Copyright 1991.
*     .. Parameters ..
      INTEGER          N, M, NT, LJ, LV, LB, LIW, LW
      PARAMETER        (N=3,M=15,NT=3,LJ=M,LV=N,LB=N*(N+1)/2,LIW=1,
     +                 LW=7*N+M*N+2*M+N*N)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Arrays in Common ..
      real             T(M,NT), Y(M)
*     .. Local Scalars ..
      real             ETA, FSUMSQ, STEPMX, XTOL
      INTEGER          I, IFAIL, IPRINT, J, MAXCAL, NF, NITER
*     .. Local Arrays ..
      real             B(LB), FJAC(LJ,N), FVEC(M), G(N), S(N), V(LV,N),
     +                 W(LW), X(N)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             X02AJF
      EXTERNAL         X02AJF
*     .. External Subroutines ..
      EXTERNAL         E04HEF, E04YAF, E04YBF, LSQFUN, LSQGRD, LSQHES,
     +                 LSQMON
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Common blocks ..
      COMMON           Y, T
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04HEF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
      DO 20 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
   20 CONTINUE
*     Set up an arbitrary point at which to check the derivatives
      X(1) = 0.19e0
      X(2) = -1.34e0
      X(3) = 0.88e0
*     Check the 1st derivatives
      IFAIL = 0
*
      CALL E04YAF(M,N,LSQFUN,X,FVEC,FJAC,LJ,IW,LIW,W,LW,IFAIL)
*
*     Check the evaluation of B
      IFAIL = 0
*
      CALL E04YBF(M,N,LSQFUN,LSQHES,X,FVEC,FJAC,LJ,B,LB,IW,LIW,W,LW,
     +            IFAIL)
*
*     Continue setting parameters for E04HEF
*     * Set IPRINT to 1 to obtain output from LSQMON at each iteration *
      IPRINT = -1
      MAXCAL = 50*N
      ETA = 0.9e0
      XTOL = 10.0e0*SQRT(X02AJF())
*     We estimate that the minimum will be within 10 units of the
*     starting point
      STEPMX = 10.0e0
```

```
*     Set up the starting point
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
      IFAIL = 1
*
      CALL E04HEF(M,N,LSQFUN,LSQHES,LSQMON,IPRINT,MAXCAL,ETA,XTOL,
     +            STEPMX,X,FSUMSQ,FVEC,FJAC,LJ,S,V,LV,NITER,NF,IW,LIW,W,
     +            LW,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
         CALL LSQGRD(M,N,FVEC,FJAC,LJ,G)
         WRITE (NOUT,99997) 'The corresponding gradient is',
     +     (G(J),J=1,N)
         WRITE (NOUT,*) '                                (machine dependent)'
         WRITE (NOUT,*) 'and the residuals are'
         WRITE (NOUT,99996) (FVEC(I),I=1,M)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
99997 FORMAT (1X,A,1P,3e12.3)
99996 FORMAT (1X,1P,e9.1)
      END
*
      SUBROUTINE LSQFUN(IFLAG,M,N,XC,FVECC,FJACC,LJC,IW,LIW,W,LW)
*     Routine to evaluate the residuals and their 1st derivatives
*     .. Parameters ..
      INTEGER           NT, MDEC
      PARAMETER         (NT=3,MDEC=15)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LIW, LJC, LW, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Arrays in Common ..
      real              T(MDEC,NT), Y(MDEC)
*     .. Local Scalars ..
      real              DENOM, DUMMY
      INTEGER           I
*     .. Common blocks ..
      COMMON            Y, T
*     .. Executable Statements ..
      DO 20 I = 1, M
         DENOM = XC(2)*T(I,2) + XC(3)*T(I,3)
         FVECC(I) = XC(1) + T(I,1)/DENOM - Y(I)
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = T(I,1)*T(I,2)*DUMMY
         FJACC(I,3) = T(I,1)*T(I,3)*DUMMY
   20 CONTINUE
      RETURN
      END
*
```

```
      SUBROUTINE LSQHES(IFLAG,M,N,FVECC,XC,B,LB,IW,LIW,W,LW)
*     Routine to compute the lower triangle of the matrix B
*     (stored by rows in the array B)
*     .. Parameters ..
      INTEGER           NT, MDEC
      PARAMETER         (NT=3,MDEC=15)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LB, LIW, LW, M, N
*     .. Array Arguments ..
      real              B(LB), FVECC(M), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Arrays in Common ..
      real              T(MDEC,NT), Y(MDEC)
*     .. Local Scalars ..
      real              DUMMY, SUM22, SUM32, SUM33
      INTEGER           I
*     .. Common blocks ..
      COMMON            Y, T
*     .. Executable Statements ..
      B(1) = 0.0e0
      B(2) = 0.0e0
      SUM22 = 0.0e0
      SUM32 = 0.0e0
      SUM33 = 0.0e0
      DO 20 I = 1, M
         DUMMY = 2.0e0*T(I,1)/(XC(2)*T(I,2)+XC(3)*T(I,3))**3
         SUM22 = SUM22 + FVECC(I)*DUMMY*T(I,2)**2
         SUM32 = SUM32 + FVECC(I)*DUMMY*T(I,2)*T(I,3)
         SUM33 = SUM33 + FVECC(I)*DUMMY*T(I,3)**2
   20 CONTINUE
      B(3) = SUM22
      B(4) = 0.0e0
      B(5) = SUM32
      B(6) = SUM33
      RETURN
      END
*
      SUBROUTINE LSQMON(M,N,XC,FVECC,FJACC,LJC,S,IGRADE,NITER,NF,IW,LIW,
     +                  W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER           NDEC
      PARAMETER         (NDEC=3)
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      INTEGER           IGRADE, LIW, LJC, LW, M, N, NF, NITER
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), S(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Local Scalars ..
      real              FSUMSQ, GTG
      INTEGER           J
*     .. Local Arrays ..
      real              G(NDEC)
*     .. External Functions ..
      real              F06EAF
      EXTERNAL          F06EAF
*     .. External Subroutines ..
      EXTERNAL          LSQGRD
```

```
*     .. Executable Statements ..
      FSUMSQ = F06EAF(M,FVECC,1,FVECC,1)
      CALL LSQGRD(M,N,FVECC,FJACC,LJC,G)
      GTG = F06EAF(N,G,1,G,1)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  ' Itns    F evals          SUMSQ             GTG        grade'
      WRITE (NOUT,99999) NITER, NF, FSUMSQ, GTG, IGRADE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '       X                   G           Singular values'
      DO 20 J = 1, N
         WRITE (NOUT,99998) XC(J), G(J), S(J)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,I4,6X,I5,6X,1P,e13.5,6X,1P,e9.1,6X,I3)
99998 FORMAT (1X,1P,e13.5,10X,1P,e9.1,10X,1P,e9.1)
      END
*
      SUBROUTINE LSQGRD(M,N,FVECC,FJACC,LJC,G)
*     Routine to evaluate gradient of the sum of squares
*     .. Scalar Arguments ..
      INTEGER           LJC, M, N
*     .. Array Arguments ..
      real              FJACC(LJC,N), FVECC(M), G(N)
*     .. Local Scalars ..
      real              SUM
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 J = 1, N
         SUM = 0.0e0
         DO 20 I = 1, M
            SUM = SUM + FJACC(I,J)*FVECC(I)
   20    CONTINUE
         G(J) = SUM + SUM
   40 CONTINUE
      RETURN
      END
```

9.2. Program Data

```
E04HEF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

9.3. Program Results

```
E04HEF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
The corresponding gradient is  -6.060E-12   9.030E-11   9.385E-11
                          (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

With IPRINT set to 1 in the example program, intermediate output similar to that below is obtained from LSQMON:

```
E04HEF Example Program Results

 Itns     F evals          SUMSQ             GTG         grade
   0           1       1.02104E+01         1.0E+03         3

       X                    G           Singular values
 5.00000E-01            2.1E+01             5.0E+00
 1.00000E+00           -1.7E+01             2.6E+00
 1.50000E+00           -1.6E+01             9.6E-02

 Itns     F evals          SUMSQ             GTG         grade
   1           2       1.98730E-01         8.1E+00         3

       X                    G           Singular values
 8.24763E-02            1.9E+00             4.2E+00
 1.13575E+00           -1.5E+00             1.8E+00
 2.06664E+00           -1.5E+00             6.6E-02

 Itns     F evals          SUMSQ             GTG         grade
   2           3       9.23238E-03         3.6E-02         3

       X                    G           Singular values
 8.24402E-02            1.4E-01             4.1E+00
 1.13805E+00           -9.5E-02             1.6E+00
 2.31707E+00           -9.5E-02             6.1E-02

 Itns     F evals          SUMSQ             GTG         grade
   3           4       8.21492E-03         1.3E-06         3

       X                    G           Singular values
 8.24150E-02            8.2E-04             4.1E+00
 1.13323E+00           -5.8E-04             1.6E+00
 2.34337E+00           -5.8E-04             6.1E-02
```

```
 Itns     F evals          SUMSQ             GTG         grade
   4          5        8.21488E-03         2.5E-15         2

       X                    G           Singular values
 8.24107E-02            3.4E-08             4.1E+00
 1.13304E+00            8.9E-09             1.6E+00
 2.34369E+00           -3.5E-08             6.1E-02

 Itns     F evals          SUMSQ             GTG         grade
   5          6        8.21488E-03         1.7E-20         0

       X                    G           Singular values
 8.24106E-02           -6.1E-12             4.1E+00
 1.13304E+00            9.0E-11             1.6E+00
 2.34370E+00            9.4E-11             6.1E-02

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
The corresponding gradient is  -6.060E-12   9.030E-11   9.385E-11
                             (machine dependent)
and the residuals are
 -5.9E-03
 -2.7E-04
  2.7E-04
  6.5E-03
 -8.2E-04
 -1.3E-03
 -4.5E-03
 -2.0E-02
  8.2E-02
 -1.8E-02
 -1.5E-02
 -1.5E-02
 -1.1E-02
 -4.2E-03
  6.8E-03
```

E04HYF – NAG Fortran Library Routine Document

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

E04HYF is an easy-to-use modified Gauss–Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First and second derivatives are required.

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 E04HYF(M, N, LSFUN2, LSHES2, X, FSUMSQ, W, LW, IUSER,
     1                  USER, IFAIL)
      INTEGER           M, N, LW, IUSER(*), IFAIL
      real              X(N), FSUMSQ, W(LW), USER(*)
      EXTERNAL          LSFUN2, LSHES2
```

3 Description

This routine is similar to the subroutine LSSDN2 in the National Physical Laboratory Algorithms Library. It is applicable to problems of the form:

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a subroutine to evaluate the residuals and their first derivatives at any point x, and a subroutine to evaluate the elements of the second derivative term of the Hessian matrix of $F(x)$.

Before attempting to minimize the sum of squares, the algorithm checks the user-supplied routines for consistency. Then, from a starting point supplied by the user, a sequence of points is generated which is intended to converge to a local minimum of the sum of squares. These points are generated using estimates of the curvature of $F(x)$.

4 References

[1] Gill P E and Murray W (1978) Algorithms for the solution of the nonlinear least-squares problem *SIAM J. Numer. Anal.* **15** 977–992

5 Parameters

1: M — INTEGER *Input*

2: N — INTEGER *Input*

On entry: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \text{N} \leq \text{M}$.

3: LSFUN2 — SUBROUTINE, supplied by the user. *External Procedure*

This routine must be supplied by the user to calculate the vector of values $f_i(x)$ and the Jacobian matrix of first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04HYF (see the the Chapter Introduction).

Its specification is:

```
SUBROUTINE LSFUN2(M, N, XC, FVECC, FJACC, LJC, IUSER, USER)
INTEGER           M, N, LJC, IUSER(*)
real              XC(N), FVECC(M), FJACC(LJC,N), USER(*)
```

Important: the dimension declaration for FJACC must contain the variable LJC, not an integer constant.

1: M — INTEGER *Input*
2: N — INTEGER *Input*
On entry: the numbers m and n of residuals and variables, respectively.

3: XC(N) — ***real*** array *Input*
On entry: the point x at which the values of the f_i and the $\frac{\partial f_i}{\partial x_j}$ are required.

4: FVECC(M) — ***real*** array *Output*
On exit: FVECC(i) must be set to the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

5: FJACC(LJC,N) — ***real*** array *Output*
On exit: FJACC(i, j) must be set to the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

6: LJC — INTEGER *Input*
On entry: the first dimension of the array FJACC.

7: IUSER(*) — INTEGER array *User Workspace*
8: USER(*) — ***real*** array *User Workspace*
LSFUN2 is called from E04HYF with the parameters IUSER and USER as supplied to E04HYF. The user is free to use the arrays IUSER and USER to supply information to LSFUN2 as an alternative to using COMMON.

LSFUN2 must be declared as EXTERNAL in the (sub)program from which E04HYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: LSHES2 — SUBROUTINE, supplied by the user. *External Procedure*
This routine must be supplied by the user to calculate the elements of the symmetric matrix

$$B(x) = \sum_{i=1}^{m} f_i(x) G_i(x),$$

at any point x, where $G_i(x)$ is the Hessian matrix of $f_i(x)$. It should be tested separately before being used in conjunction with E04HYF (see the Chapter Introduction).

Its specification is:

```
SUBROUTINE LSHES2(M, N, FVECC, XC, B, LB, IUSER, USER)
INTEGER           M, N, LB, IUSER(*)
real              FVECC(M), XC, B(LB), USER(*)
```

1: M — INTEGER *Input*
2: N — INTEGER *Input*
On entry: the numbers m and n of residuals and variables, respectively.

3: FVECC(M) — ***real*** array *Input*
On entry: the value of the residual f_i at the point in XC, for $i = 1, 2, \ldots, m$, so that the values of the f_i can be used in the calculation of the elements of B.

4: XC — ***real*** *Input*

On entry: the point x at which the elements of B are to be evaluated.

5: B(LB) — ***real*** array *Input*

On exit: B must contain the lower triangle of the matrix $B(x)$, evaluated at the point x, stored by rows. (The upper triangle is not required because the matrix is symmetric.) More precisely, B$(j(j-1)/2+k)$ must contain $\sum_{i=1}^{m} f_i \frac{\partial^2 f_i}{\partial x_j \partial x_k}$ evaluated at the point x, for $j = 1, 2, \ldots, n$; $k = 1, 2, \ldots, j$.

6: LB — INTEGER *Input*

On entry: the length of the array B.

7: IUSER(*) — INTEGER array *User Workspace*
8: USER(*) — ***real*** array *User Workspace*

LSHES2 is called from E04HYF with the parameters IUSER and USER as supplied to E04HYF. The user is free to use the arrays IUSER and USER to supply information to LSHES2 as an alternative to using COMMON.

LSHES2 must be declared as EXTERNAL in the (sub)program from which E04HYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: 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 LSFUN2 and LSHES2 at the starting point, and so is more likely to detect any error in the user's routines 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.

6: FSUMSQ — ***real*** *Output*

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

7: W(LW) — ***real*** array *Workspace*

8: LW — INTEGER *Input*

On entry: the length of W as declared in the (sub)program from which E04HYF is called.

Constraints:

LW $\geq 8 \times$ N $+ 2 \times$ N $\times$ N $+ 2 \times$ M $\times$ N $+ 3 \times$ M, if N > 1,
LW $\geq 11 + 5 \times$ M, if N $= 1$.

9: IUSER(*) — INTEGER array *User Workspace*

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

IUSER is not used by E04HYF, but is passed directly to LSFUN2 and LSHES2 and may be used to pass information to those routines.

10: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04HYF, but is passed directly to LSFUN2 and LSHES2 and may be used to pass information to those routines.

11: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to −1 before entry. **It is then 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 specified by the routine:

IFAIL = 1

On entry, N < 1,
or M < N,
or LW < 8 × N + 2 × N × N + 2 × M × N + 3 × M, when N > 1,
or LW < 11 + 5 × M, when N = 1.

IFAIL = 2

There have been 50 × n calls of LSFUN2, yet the algorithm does not seem to have converged. This may be due to an awkward function or to a poor starting point, so it is worth restarting E04HYF from the final point held in X.

IFAIL = 3

The final point does not satisfy the conditions for acceptance as a minimum, but no lower point could be found.

IFAIL = 4

An auxiliary routine has been unable to complete a singular value decomposition in a reasonable number of sub-iterations.

IFAIL = 5, 6, 7 and 8

There is some doubt about whether the point x found by E04HYF is a minimum of $F(x)$. 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

It is very likely that the user has made an error in forming the derivatives $\frac{\partial f_i}{\partial x_j}$ in LSFUN2.

IFAIL = 10

It is very likely that the user has made an error in forming the quantities B_{jk} in LSHES2.

If the user is not satisfied with the result (e.g. because IFAIL lies between 3 and 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. Repeated failure may indicate some defect in the formulation of the problem.

7 Accuracy

If the problem is reasonably well scaled and 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 the components of x and between $t - 1$ (if $F(x)$ is of order 1 at the minimum) and $2t - 2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

8 Further Comments

The number of iterations required depends on the number of variables, the number of residuals and their behaviour, and the distance of the starting point from the solution. The number of multiplications performed per iteration of E04HYF varies, but for $m \gg n$ is approximately $n \times m^2 + \mathrm{O}(n^3)$. In addition, each iteration makes at least one call of LSFUN2, and some iterations may call LSHES2. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in LSFUN2 (and, to a lesser extent, in LSHES2).

Ideally, the problem should be scaled so that the minimum value of the sum of squares is in the range $(0, +1)$, and so that at points a unit distance away from the solution the sum of squares 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 E04HYF will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to E04YCF, using information returned in segments of the workspace array W. See E04YCF for further details.

9 Example

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

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.

```
*     E04HYF Example Program Text.
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          M, N, NT, LW
      PARAMETER        (M=15,N=3,NT=3,LW=8*N+2*N*N+2*M*N+3*M)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
```

```
*     .. Local Scalars ..
      real             FSUMSQ
      INTEGER          I, IFAIL, J, K
*     .. Local Arrays ..
      real             T(M,NT), USER(M+M*NT), W(LW), X(N), Y(M)
      INTEGER          IUSER(1)
*     .. External Subroutines ..
      EXTERNAL         E04HYF, LSFUN3, LSHES3
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04HYF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Observations of TJ (J = 1, 2, 3) are held in T(I, J)
*     (I = 1, 2, . . . , 15)
*
      IUSER(1) = NT
      K = M
      DO 40 I = 1, M
         READ (NIN,*) Y(I), (T(I,J),J=1,NT)
         USER(I) = Y(I)
         DO 20 J = 1, NT
            USER(K+J) = T(I,J)
   20    CONTINUE
         K = K + NT
   40 CONTINUE
*
      X(1) = 0.5e0
      X(2) = 1.0e0
      X(3) = 1.5e0
*
      IFAIL = 1
*
      CALL E04HYF(M,N,LSFUN3,LSHES3,X,FSUMSQ,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) 'On exit, the sum of squares is', FSUMSQ
         WRITE (NOUT,99998) 'at the point', (X(J),J=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,3F12.4)
      END

      SUBROUTINE LSFUN3(M,N,XC,FVECC,FJACC,LJC,IUSER,USER)
*     Routine to evaluate the residuals and their 1st derivatives.
*     .. Scalar Arguments ..
      INTEGER          LJC, M, N
*     .. Array Arguments ..
      real             FJACC(LJC,N), FVECC(M), USER(*), XC(N)
      INTEGER          IUSER(*)
```

```
*     .. Local Scalars ..
      real               DENOM, DUMMY
      INTEGER            I, K
*     .. Executable Statements ..
      K = M
      DO 20 I = 1, M
         DENOM = XC(2)*USER(K+2) + XC(3)*USER(K+3)
         FVECC(I) = XC(1) + USER(K+1)/DENOM - USER(I)
         FJACC(I,1) = 1.0e0
         DUMMY = -1.0e0/(DENOM*DENOM)
         FJACC(I,2) = USER(K+1)*USER(K+2)*DUMMY
         FJACC(I,3) = USER(K+1)*USER(K+3)*DUMMY
         K = K + IUSER(1)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE LSHES3(M,N,FVECC,XC,B,LB,IUSER,USER)
*     Routine to compute the lower triangle of the matrix B
*     (stored by rows in the array B).
*     .. Scalar Arguments ..
      INTEGER            LB, M, N
*     .. Array Arguments ..
      real               B(LB), FVECC(M), USER(*), XC(N)
      INTEGER            IUSER(*)
*     .. Local Scalars ..
      real               DUMMY, SUM22, SUM32, SUM33
      INTEGER            I, K
*     .. Executable Statements ..
      B(1) = 0.0e0
      B(2) = 0.0e0
      SUM22 = 0.0e0
      SUM32 = 0.0e0
      SUM33 = 0.0e0
      K = M
      DO 20 I = 1, M
         DUMMY = 2.0e0*USER(K+1)/(XC(2)*USER(K+2)+XC(3)*USER(K+3))**3
         SUM22 = SUM22 + FVECC(I)*DUMMY*USER(K+2)**2
         SUM32 = SUM32 + FVECC(I)*DUMMY*USER(K+2)*USER(K+3)
         SUM33 = SUM33 + FVECC(I)*DUMMY*USER(K+3)**2
         K = K + IUSER(1)
   20 CONTINUE
      B(3) = SUM22
      B(4) = 0.0e0
      B(5) = SUM32
      B(6) = SUM33
      RETURN
      END
```

9.2 Program Data

```
E04HYF Example Program Data
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
```

```
0.35  7.0  9.0  7.0
0.39  8.0  8.0  8.0
0.37  9.0  7.0  7.0
0.58 10.0  6.0  6.0
0.73 11.0  5.0  5.0
0.96 12.0  4.0  4.0
1.34 13.0  3.0  3.0
2.10 14.0  2.0  2.0
4.39 15.0  1.0  1.0
```

9.3 Program Results

```
E04HYF Example Program Results

On exit, the sum of squares is      0.0082
at the point      0.0824      1.1330      2.3437
```

E04JYF – NAG Fortran Library Routine Document

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

E04JYF 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 of the independent variables $x_1, x_2, \ldots, x_n$, using function values only.

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 E04JYF(N, IBOUND, FUNCT1, BL, BU, X, F, 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, W(LW), USER(*)
EXTERNAL          FUNCT1
```

3 Description

This routine is applicable to problems of the form:

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

when derivatives of $F(x)$ are unavailable.

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 value of $F(x)$ at any point x.

From a starting point supplied by the user there is generated, on the basis of estimates of the gradient and 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 finite-difference approximations to the derivatives of $F(x)$ with respect to the free variables, is known. A unit lower triangular matrix L and a diagonal matrix D (both of dimension n_z), such that LDL^T is a positive-definite approximation 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 estimated 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. Most iterations calculate g_z using forward differences, but central differences are used when they seem necessary.

There are two sets of convergence criteria – a weaker and a stronger. Whenever the weaker criteria are satisfied, the Lagrange-multipliers are estimated for all the active constraints. If any Lagrange-multiplier estimate is significantly negative, then one of the variables associated with a negative Lagrange-multiplier estimate is released from its bound and the next search direction is computed in the extended subspace (i.e., n_z is increased). Otherwise minimization continues in the current subspace provided that this is practicable. When it is not, or when the stronger convergence criteria 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

[1] 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 ≥ 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

3: FUNCT1 — SUBROUTINE, supplied by the user. *External Procedure*

This routine must be supplied by the user to calculate the value of the function $F(x)$ at any point x. It should be tested separately before being used with E04JYF (see the Chapter Introduction).

Its specification is:

```
SUBROUTINE FUNCT1(N, XC, FC, IUSER, USER)
INTEGER           N, IUSER(*)
real              XC(N), FC, 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 value is required.

3: FC — ***real*** *Output*

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

4:	IUSER(*) — INTEGER array	*User Workspace*
5:	USER(*) — ***real*** array	*User Workspace*

FUNCT1 is called from E04JYF with the parameters IUSER and USER as supplied to E04JYF. The user is free to use the arrays IUSER and USER to supply information to FUNCT1 as an alternative to using COMMON.

FUNCT1 must be declared as EXTERNAL in the (sub)program from which E04JYF 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; E04JYF will then set the remaining elements of BL equal to BL(1).

On exit: the lower bounds actually used by E04JYF.

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; E04JYF will then set the remaining elements of BU equal to BU(1).

On exit: the upper bounds actually used by E04JYF.

6: X(N) — ***real*** array *Input/Output*

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

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: 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(N+1) contains the number of free variables (i.e., n_z). The rest of the array is used as workspace.

9: LIW — INTEGER *Input*

On entry: the length of IW as declared in the (sub)program from which E04JYF is called.

Constraint: LIW $\geq$ N + 2.

10: W(LW) — ***real*** array *Output*

On exit: if IFAIL = 0, 3 or 5, W(i) contains a finite-difference approximation to the ith element of the projected gradient vector g_z, for $i = 1, 2, \ldots$,N. In addition, W(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.

11: LW — INTEGER *Input*

On entry: the length of W as declared in the (sub)program from which E04JYF is called.

Constraint: LW $\geq$ max(N$\times$(N$-$1)/2+12$\times$N,13).

12: IUSER(*) — INTEGER array *User Workspace*

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

IUSER is not used by E04JYF, but is passed directly to FUNCT1 and may be used to pass information to those routines.

13: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04JYF, but is passed directly to FUNCT1 and may be used to pass information to those routines.

14: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified 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(13,12$\times$N+N$\times$(N$-$1)/2).

IFAIL = 2

There have been 400 $\times$ 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, 6, 7 and 8

There is some doubt about whether the point x found by E04JYF 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 FUNCT1, that the user's problem has no finite solution, or that the problem needs rescaling (see Section 8).

IFAIL = 10

The computed set of forward-difference intervals (stored in W(9*N+1),W(9*N+2),...,W(10*N)) is such that X(i) +W(9*N+i) $\leq$ X(i) for some i.

This is an unlikely failure, but if it occurs the user should attempt to select another starting point.

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 and the gradient can be calculated, it may be advisable to change to a routine which uses gradients (see the Chapter Introduction).

7 Accuracy

A successful exit (IFAIL = 0) is made from E04JYF when (B1, B2 and B3) or B4 hold, and the local search confirms a minimum, where

$$\text{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (x_{tol} + \sqrt{\epsilon}) \times (1.0 + \|x^{(k)}\|)$$
$$\text{B2} \equiv |F^{(k)} - F^{(k-1)}| < (x_{tol}^2 + \epsilon) \times (1.0 + |F^{(k)}|)$$
$$\text{B3} \equiv \|g_z^{(k)}\| < (\epsilon^{1/3} + x_{tol}) \times (1.0 + |F^{(k)}|)$$
$$\text{B4} \equiv \|g_z^{(k)}\| < 0.01 \times \sqrt{\epsilon}.$$

(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

(1) the sequence $\{F(x^{(k)})\}$ converges to F(x_{sol}) at a superlinear or a fast linear rate,
(2) $\|g_z(x_{xol})\|^2 < 10.0 \times \epsilon$ and
(3) $k < 1.0/\|g_z(x_{sol})\|$,

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (2) 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 E04JYF is roughly proportional to n^2. In addition, each iteration makes at least $m+1$ calls of FUNCT1,

where m is the number of variables not fixed on bounds. So, unless $F(x)$ can be evaluated very quickly, the run time will be dominated by the time spent in FUNCT1.

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 E04JYF will take less computer time.

9 Example

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

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

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

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.

```
*     E04JYF Example Program Text.
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          N, LIW, LW
      PARAMETER        (N=4,LIW=N+2,LW=N*(N-1)/2+12*N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             F
      INTEGER          IBOUND, IFAIL, J
*     .. Local Arrays ..
      real             BL(N), BU(N), USER(N), W(LW), X(N)
      INTEGER          IUSER(N), IW(6)
*     .. External Subroutines ..
      EXTERNAL         E04JYF, FUNCT1
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04JYF 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 E04JYF(N,IBOUND,FUNCT1,BL,BU,X,F,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)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,F8.4)
99997 FORMAT (1X,A,4F9.4)
      END
*
      SUBROUTINE FUNCT1(N,XC,FC,IUSER,USER)
*     Routine to evaluate objective function.
*     .. Scalar Arguments ..
      real              FC
      INTEGER           N
*     .. Array Arguments ..
      real              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
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
E04JYF Example Program Results

Error exit type  5 - see routine document

Function value on exit is   2.4338
at the point   1.0000  -0.0852   0.4093   1.0000
```

E04KDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

E04KDF 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 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 E04KDF (N, FUNCT, MONIT, IPRINT, MAXCAL, ETA, XTOL, DELTA,
     1                   STEPMX, IBOUND, BL, BU, X, HESL, LH, HESD,
     2                   ISTATE, F, G, IW, LIW, W, LW, IFAIL)
      INTEGER           N, IPRINT, MAXCAL, IBOUND, LH, ISTATE(N), IW(LIW),
     1                  LIW, LW, IFAIL
      real              ETA, XTOL, DELTA, STEPMX, BL(N), BU(N), X(N),
     1                  HESL(LH), HESD(N), F, G(N), W(LW)
      EXTERNAL          FUNCT, MONIT
```

3. Description

This routine is applicable to problems of the form:

Minimize $F(x_1,x_2,...,x_n)$ subject to $l_j \leq x_j \leq u_j$, for $j = 1,2,...,n$.

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

A typical iteration starts at the current point x where n_z (say) variables are free from their bounds. The vector g_z, whose elements are the derivatives of $F(x)$ with respect to the free variables, is known. The matrix of second derivatives with respect to the free variables, H, is estimated by finite differences. (Note that g_z and H are both of 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 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 until the stronger convergence criteria are satisfied. If at this point there are no negative or near-zero Lagrange-multiplier estimates, the process is terminated.

If the user specifies that the problem is unconstrained, E04KDF 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 E04KDF will act as an unconstrained minimization algorithm.

4. References

[1] GILL, P.E. and MURRAY, W.
Safeguarded steplength algorithms for optimization using descent methods.
National Physical Laboratory Report NAC 37, 1974.

[2] GILL, P.E. and MURRAY, W.
Newton-type methods for unconstrained and linearly constrained optimization.
Mathematical Programming, 7, pp. 311-350, 1974.

[3] GILL, P.E. and MURRAY, W.
Minimization subject to bounds on the variables.
National Physical Laboratory Report NAC 72, 1976.

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 a specified point. (However, if the user does not wish to calculate F or its first derivatives at a particular x, there is the option of setting a parameter to cause E04KDF to terminate immediately.)

Its specification is:

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

1: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG will have been set to 1 or 2. The value 1 indicates that only the first derivatives of F need be supplied, and the value 2 indicates that both F itself and its first derivatives must be calculated.

On exit: if it is not possible to evaluate F or its first derivatives at the point given in XC (or if it is wished to stop the calculations for any other reason) the user should reset IFLAG to a negative number and return control to E04KDF. E04KDF will then terminate immediately, with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FC – ***real***. *Output*

On exit: unless IFLAG = 1 on entry or IFLAG is reset, FUNCT must set FC to the value of the objective function F at the current point x.

5: GC(N) – ***real*** array. *Output*

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

6: IW(LIW) – INTEGER array. *Workspace*

7: LIW – INTEGER. *Input*

8: W(LW) – ***real*** array. *Workspace*

9: LW – INTEGER. *Input*

FUNCT is called with the same parameters IW, LIW, W, LW as for E04KDF. 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, users **could** use elements 3,4,...,LIW of IW and elements from max(8,7×N+N×(N−1)/2) + 1 onwards of W for passing quantities to FUNCT from the (sub)program which calls E04KDF. However, because of the danger of mistakes in partitioning, it is recommended that users should pass information to FUNCT via COMMON and not use IW or W at all. In any case users must not change the first 2 elements of IW or the first max(8,7×N+N×(N−1)/2) elements of W.

Note: FUNCT should be tested separately before being used in conjunction with E04KDF. It must be declared as EXTERNAL in the (sub)program from which E04KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

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

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

Its specification is:

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

1: N – INTEGER. *Input*

On entry: the number n of variables.

2: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

3: FC – ***real***. *Input*

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

4: GC(N) – ***real*** array. *Input*

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

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(j) is positive, its value gives the position of x_j in the sequence of free variables.

6: GPJNRM – ***real***. *Input*

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

7: COND – ***real***. *Input*

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

8: POSDEF – LOGICAL. *Input*

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

9: NITER – INTEGER. *Input*

On entry: the number of iterations (as outlined in Section 3) which have been performed by E04KDF so far.

10: NF – INTEGER. *Input*

On entry: the number of evaluations of $F(x)$ so far, i.e. the number of calls of FUNCT with IFLAG set to 2. Each such call of FUNCT also calculates the first derivatives of F. (In addition to these calls monitored by NF, FUNCT is called with IFLAG set to 1 not more than N times per iteration.)

11: IW(LIW) – INTEGER array. *Workspace*
12: LIW – INTEGER. *Input*
13: W(LW) – ***real*** array. *Workspace*
14: LW – INTEGER. *Input*

As in FUNCT, these parameters correspond to the parameters IW, LIW, W, LW of E04KDF. They are included in MONIT's parameter list primarily when E04KDF is called by other library routines.

The user should normally print FC, GPJNRM and COND to be able to compare the quantities mentioned in Section 7. It is usually helpful to examine XC, POSDEF and NF too. MONIT must be declared as EXTERNAL in the (sub)program from which E04KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: IPRINT – INTEGER. *Input*

On entry: the frequency with which MONIT is to be called. If IPRINT > 0, MONIT is called once every IPRINT iterations and just before exit from E04KDF. If IPRINT = 0, MONIT is just called at the final point. If IPRINT < 0, MONIT is not called at all. IPRINT should normally be set to a small positive number.

Suggested value: IPRINT = 1.

5: MAXCAL – INTEGER. *Input*

On entry: the maximum permitted number of evaluations of $F(x)$, i.e. the maximum permitted number of calls of FUNCT with IFLAG set to 2. It should be borne in mind that, in addition to the calls of FUNCT which are limited directly by MAXCAL, there will be calls of FUNCT (with IFLAG set to 1) to evaluate only first derivatives.

Suggested value: MAXCAL = 50×N.

Constraint: MAXCAL ≥ 1.

6: ETA – ***real.*** *Input*

On entry: every iteration of E04KDF 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 large values (say 0.9).

Although accurate linear minimizations will generally reduce the number of iterations (and hence the number of calls of FUNCT to estimate the second derivatives), they will tend to increase the number of calls of FUNCT needed for each linear minimization. On balance, it is usually efficient to perform a low accuracy linear minimization when n is small and a high accuracy minimization when n is large.

Suggested value: ETA = 0.5 if $1 < n < 10$,
ETA = 0.1 if $10 \leq n \leq 20$,
ETA = 0.01 if $n > 20$.

If N = 1, ETA should be set to 0.0 (also when the problem is effectively 1-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.

7: XTOL – ***real.*** *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 prior to a normal exit, is such that $\|x_{sol}-x_{true}\| <$ XTOL$\times(1.0+\|x_{true}\|)$ 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 5 decimal places. (For further details see Section 7.)

If the problem is scaled as described in Section 8.2 and ε is the ***machine precision***, then $\sqrt{\varepsilon}$ is probably the smallest reasonable choice for XTOL. This is because, normally, to machine accuracy, $F(x+\sqrt{\varepsilon}e_j) = F(x)$, for any j where e_j is the jth column of the identity matrix. If the user sets XTOL to 0.0 (or any positive value less than ε), E04KDF will use $10.0\times\sqrt{\varepsilon}$ instead of XTOL.

Suggested value: XTOL = 0.0.

Constraint: XTOL ≥ 0.0.

8: DELTA – ***real.*** *Input*

On entry: the differencing interval to be used for approximating the second derivatives of $F(x)$. Thus, for the finite difference approximations, the first derivatives of $F(x)$ are evaluated at points which are DELTA apart. If ε is the ***machine precision***, then $\sqrt{\varepsilon}$ will usually be a suitable setting for DELTA. If the user sets DELTA to 0.0 (or to any positive value less than ε), E04KDF will automatically use $\sqrt{\varepsilon}$ as the differencing interval.

Suggested value: DELTA = 0.0.

Constraint: DELTA ≥ 0.0.

9: STEPMX – ***real***. *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency a slight overestimate is preferable.) E04KDF 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, E04KDF 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 ≥ XTOL.

10: IBOUND – INTEGER. *Input*

On entry: indicates 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. IBOUND should be set to one of the following values:

IBOUND = 0

if the variables are bounded and the user will be 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

IBOUND = 4

if the problem is unconstrained. (The IBOUND = 4 option is provided for consistency with other routines. In E04KDF it produces the same effect as IBOUND = 1.)

Constraint: 0 ≤ IBOUND ≤ 4.

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

On entry: the fixed lower bounds l_j.

If IBOUND is set to 0, the user must set BL(j) to l_j, for j = 1,2,...,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, the user must set BL(1) to l_1; E04KDF will then set the remaining elements of BL equal to BL(1).

If IBOUND is set to 1, 2 or 4, BL will be initialised by E04KDF.

On exit: the lower bounds actually used by E04KDF, e.g. If IBOUND = 2, BL(1) = BL(2) = ... = BL(n) = 0.0.

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

On entry: the fixed upper bounds u_j.

If IBOUND is set to 0, the user must set BU(j) to u_j, for j = 1,2,...,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, the user must set BU(1) to u_1; E04KDF will then set the remaining elements of BU equal to BU(1).

If IBOUND is set to 1, 2 or 4, BU will be initialised by E04KDF.

On exit: the upper bounds actually used by E04KDF, e.g. if IBOUND = 2, BU(1) = BU(2) = ... = BU(n) = 10^6.

13: 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,...,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*** array. *Output*

See description of HESD below.

15: LH – INTEGER. *Input*

On entry: the length of HESL as declared in the (sub)program from which E04KDF is called.

Constraint: LH ≥ max(N×(N−1)/2,1).

16: HESD(N) – ***real*** 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 E04KDF (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***. *Output*

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

19: G(N) – ***real*** 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. *Workspace*

21: LIW – INTEGER. *Input*

On entry: the length of IW as declared in the (sub)program from which E04KDF is called.

Constraint: LIW ≥ 2.

22: W(LW) – ***real*** array. *Workspace*

23: LW – INTEGER. *Input*

On entry: the length of W as declared in the (sub)program from which E04KDF is called.

Constraint: LW ≥ max(7×N+N×(N−1)/2,8).

24: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04KDF because the user has set IFLAG negative in FUNCT. The value of IFAIL will be the same as the user's 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 DELTA < 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,N×(N−1)/2),
or LIW < 2,
or LW < max(8,7×N+N×(N−1)/2).

(Note that if the user has set XTOL or DELTA to 0.0, E04KDF uses the default values 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 funtion 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 recognise 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. The user should investigate as for IFAIL = 3.

The values IFAIL = 2, 3 and 5 may also be caused by mistakes in FUNCT, 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 E04KDF when $H^{(k)}$ is positive-definite and when (B1, B2 and B3) or B4 hold, where

B1 $\equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\sqrt{\varepsilon}) \times (1.0+\|x^{(k)}\|)$

B2 $\equiv |F^{(k)}-F^{(k-1)}| < (\text{XTOL}^2+\varepsilon) \times (1.0+|F^{(k)}|)$

B3 $\equiv \|g_z^{(k)}\| < (\varepsilon^{\frac{1}{3}}+\text{XTOL}) \times (1.0+|F^{(k)}|)$

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

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

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

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

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

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (2) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. 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 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 E04KDF is $\frac{n_z^3}{6} + O(n_z^2)$. In addition, each iteration makes n_z calls of FUNCT (with IFLAG set to 1) in approximating the projected Hessian matrix, and at least one other call of FUNCT (with IFLAG set to 2). So, unless $F(x)$ and its first derivatives can be evaluated very quickly, the run time will be dominated by the time spent in FUNCT.

8.2. Scaling

Ideally, the problem should be scaled so that, at the solution, $F(x)$ and the corresponding values of 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 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 E04KDF 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 estimated 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 E04KDF 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 UNCKDF(N,FUNCT,MONIT,IPRINT,MAXCAL,ETA,XTOL,DELTA,
     *                  STEPMX,X,HESL,LH,HESD,F,G,IWORK,LIWORK,WORK,
     *                  LWORK,IFAIL)
C
C     A ROUTINE TO APPLY E04KDF 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 E04KDF 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 E04KDF 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 E04KDF.
C
```

```
C     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
C     .. Scalar Arguments ..
      real              DELTA, 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, MONIT
C     .. Local Scalars ..
      INTEGER           IBOUND, J, JBL, JBU, NH
      LOGICAL           TOOBIG
C     .. External Subroutines ..
      EXTERNAL          E04KDF
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 E04KDF(N,FUNCT,MONIT,IPRINT,MAXCAL,ETA,XTOL,DELTA,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

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

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

starting from the initial guess $(3, -1, 0, 1)$. Before calling E04KDF, the program calls E04HCF to check the first derivatives calculated by FUNCT.

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.

```
*     E04KDF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LH, LIW, LW
      PARAMETER        (N=4,LH=N*(N-1)/2,LIW=2,LW=7*N+N*(N-1)/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             DELTA, ETA, F, STEPMX, XTOL
      INTEGER          IBOUND, IFAIL, IPRINT, J, MAXCAL
*     .. Local Arrays ..
      real             BL(N), BU(N), G(N), HESD(N), HESL(LH), W(LW),
     +                 X(N)
      INTEGER          ISTATE(N), IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         E04HCF, E04KDF, FUNCT, MONIT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04KDF Example Program Results'
*     Check FUNCT by calling E04HCF at an arbitrary point. Since E04HCF
*     only checks the derivatives calculated when IFLAG = 2, a separate
*     program should be run before using E04HCF or E04KDF to check that
*     FUNCT gives the same values for the GC(J) when IFLAG is set to 1
*     as when IFLAG is set to 2.
      X(1) = 1.46e0
      X(2) = -0.82e0
      X(3) = 0.57e0
      X(4) = 1.21e0
      IFAIL = 0
*
      CALL E04HCF(N,FUNCT,X,F,G,IW,LIW,W,LW,IFAIL)
*
*     Continue setting parameters for E04KDF
*     * Set IPRINT to 1 to obtain output from MONIT at each iteration *
      IPRINT = -1
      MAXCAL = 50*N
      ETA = 0.5e0
*     Set XTOL and DELTA to zero so that E04KDF will use the default
*     values
      XTOL = 0.0e0
      DELTA = 0.0e0
*     We estimate that the minimum will be within 4 units of the
*     starting point
      STEPMX = 4.0e0
      IBOUND = 0
      BL(1) = 1.0e0
      BU(1) = 3.0e0
      BL(2) = -2.0e0
      BU(2) = 0.0e0
*     X(3) is not bounded, 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
*     Set up starting point
      X(1) = 3.0e0
      X(2) = -1.0e0
      X(3) = 0.0e0
      X(4) = 1.0e0
      IFAIL = 1
*
```

```
      CALL E04KDF(N,FUNCT,MONIT,IPRINT,MAXCAL,ETA,XTOL,DELTA,STEPMX,
     +            IBOUND,BL,BU,X,HESL,LH,HESD,ISTATE,F,G,IW,LIW,W,LW,
     +            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,99998) 'at the point', (X(J),J=1,N)
         WRITE (NOUT,*)
     +     'The corresponding (machine dependent) gradient is'
         WRITE (NOUT,99997) (G(J),J=1,N)
         WRITE (NOUT,99996) 'ISTATE contains', (ISTATE(J),J=1,N)
         WRITE (NOUT,99995) 'and HESD contains', (HESD(J),J=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,4F12.4)
99997 FORMAT (24X,1P,4e12.3)
99996 FORMAT (1X,A,4I5)
99995 FORMAT (1X,A,4e12.4)
      END
*
      SUBROUTINE FUNCT(IFLAG,N,XC,FC,GC,IW,LIW,W,LW)
*     Routine to evaluate objective function and its 1st derivatives.
*     A COMMON variable could be updated here to count the number of
*     calls of FUNCT with IFLAG = 1 (since NF in MONIT only counts
*     calls with IFLAG = 2)
*     .. Scalar Arguments ..
      real              FC
      INTEGER           IFLAG, LIW, LW, N
*     .. Array Arguments ..
      real              GC(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Executable Statements ..
      IF (IFLAG.NE.1) FC = (XC(1)+10.0e0*XC(2))**2 + 5.0e0*(XC(3)-XC(4))
     +                     **2 + (XC(2)-2.0e0*XC(3))**4 + 10.0e0*(XC(1)
     +                     -XC(4))**4
      GC(1) = 2.0e0*(XC(1)+10.0e0*XC(2)) + 40.0e0*(XC(1)-XC(4))**3
      GC(2) = 20.0e0*(XC(1)+10.0e0*XC(2)) + 4.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(3) = 10.0e0*(XC(3)-XC(4)) - 8.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(4) = 10.0e0*(XC(4)-XC(3)) - 40.0e0*(XC(1)-XC(4))**3
      RETURN
      END
*
      SUBROUTINE MONIT(N,XC,FC,GC,ISTATE,GPJNRM,COND,POSDEF,NITER,NF,IW,
     +                 LIW,W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              COND, FC, GPJNRM
      INTEGER           LIW, LW, N, NF, NITER
      LOGICAL           POSDEF
*     .. Array Arguments ..
      real              GC(N), W(LW), XC(N)
      INTEGER           ISTATE(N), IW(LIW)
*     .. Local Scalars ..
      INTEGER           ISJ, J
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +' Itn     Fn evals              Fn value            Norm of proj g
     +radient'
      WRITE (NOUT,99999) NITER, NF, FC, GPJNRM
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  ' J           X(J)                G(J)           Status'
      DO 20 J = 1, N
         ISJ = ISTATE(J)
         IF (ISJ.GT.0) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Free'
         ELSE IF (ISJ.EQ.-1) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Upper Bound'
         ELSE IF (ISJ.EQ.-2) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Lower Bound'
         ELSE IF (ISJ.EQ.-3) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Constant'
         END IF
   20 CONTINUE
      IF (COND.NE.0.0e0) THEN
         IF (COND.GT.1.0e6) 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
*           The following statement is included so that this MONIT
*           can be used in conjunction with either of the routines
*           E04KDF or E04LBF
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        'Projected Hessian matrix is not positive definite'
         END IF
         RETURN
      END IF
*
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
```

9.2. Program Data

None.

9.3. Program Results

```
E04KDF Example Program Results

Error exit type  3 - see routine document

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  -5.871E-10   1.176E-09   5.907E+00
ISTATE contains   -2    1    2   -2
and HESD contains  0.2098E+03  0.4738E+02  0.4552E+02  0.0000E+00
```

With IPRINT set to 1 in the example program, intermediate results similar to those below are obtained from MONIT:

```
 Itn     Fn evals              Fn value            Norm of proj gradient
  0          1                2.1500E+02                1.4401E+02

 J          X(J)                 G(J)           Status
 1          3.0000E+00           3.0600E+02     Upper Bound
 2         -1.0000E+00          -1.4400E+02     Free
 3          0.0000E+00          -2.0000E+00     Free
 4          1.0000E+00          -3.1000E+02     Lower Bound

 Estimated condition number of projected Hessian =   3.83E+00

 Itn     Fn evals              Fn value            Norm of proj gradient
  1          2                1.6306E+02                3.2033E+02

 J          X(J)                 G(J)           Status
 1          3.0000E+00           3.2033E+02     Free
 2         -2.8328E-01          -3.5132E-02     Free
 3          3.3106E-01           7.0265E-02     Free
 4          1.0000E+00          -3.1331E+02     Lower Bound

 Estimated condition number of projected Hessian =   9.46E+00

  .
  .
```

intermediate results omitted

```
  .
  .
 Estimated condition number of projected Hessian =   4.43E+00

 Itn     Fn evals              Fn value            Norm of proj gradient
  9         10                2.4338E+00                1.3153E-09

 J          X(J)                 G(J)           Status
 1          1.0000E+00           2.9535E-01     Lower Bound
 2         -8.5233E-02          -5.8718E-10     Free
 3          4.0930E-01           1.1770E-09     Free
 4          1.0000E+00           5.9070E+00     Lower Bound

 Estimated condition number of projected Hessian =   4.43E+00

 Itn     Fn evals              Fn value            Norm of proj gradient
 10         11                2.4338E+00                1.3153E-09

 J          X(J)                 G(J)           Status
 1          1.0000E+00           2.9535E-01     Lower Bound
 2         -8.5233E-02          -5.8718E-10     Free
 3          4.0930E-01           1.1770E-09     Free
 4          1.0000E+00           5.9070E+00     Lower Bound

 Estimated condition number of projected Hessian =   4.43E+00
```

E04KYF – NAG Fortran Library Routine Document

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,
1                 LW, 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:

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

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

[1] 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 ≥ 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

Constraint: $0 \leq$ IBOUND ≤ 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 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(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 length of IW, as declared in the (sub)program from which E04KYF is called.

Constraint: LIW $\geq$ 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 $i = 1, 2, \ldots,$ N. In addition, W(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 length of W, as declared in the (sub)program from which E04KYF is called.

Constraint: LW $\geq$ max(10×N+N×(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 those routines.

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 those routines.

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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified 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×N+N×(N−1)/2).

IFAIL = 2

There have been 100 × 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, 6, 7 and 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

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

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

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

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

(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

(1) the sequence $\{F(x^{(k)})\}$ converges to $F(x_{sol})$ at a superlinear or a fast linear rate,
(2) $\|g_z(x_{xol})\|^2 < 10.0 \times \epsilon$ and
(3) $k < 1.0/\|g_z(x_{sol})\|$,

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (2) 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

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

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

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
```

E04KZF – NAG Fortran Library Routine Document

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

E04KZF 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 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 E04KZF(N, IBOUND, FUNCT2, BL, BU, X, F, G, IW, LIW, W,
     1                  LW, 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:

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

when first derivatives are known.

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 $\mathrm{F}(x)$ 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 gradient of the curvature of $\mathrm{F}(x)$, a sequence of feasible points which is intended to converge to a local minimum of the constrained function.

4 References

[1] 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: $\mathrm{N} \geq 1$.

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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

Constraint: $0 \leq$ IBOUND ≤ 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 derivatives $\frac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04KZF (see the the 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 E04KZF with the parameters IUSER and USER as supplied to E04KZF. 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 E04KZF 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; E04KZF will then set the remaining elements of BL equal to BL(1).

On exit: the lower bounds actually used by E04KZF.

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; E04KZF will then set the remaining elements of BU equal to BU(1).

On exit: the upper bounds actually used by E04KZF.

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 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 *Workspace*

10: LIW — INTEGER *Input*

On entry: the length of IW, as declared in the (sub)program from which E04KZF is called.

Constraint: LIW $\geq$ N + 2.

11: W(LW) — ***real*** array *Workspace*

12: LW — INTEGER *Input*

On entry: the length of W, as declared in the (sub)program from which E04KZF is called.

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

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

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

IUSER is not used by E04KZF, but is passed directly to FUNCT2 and may be used to pass information to those routines.

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

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

USER is not used by E04KZF, but is passed directly to FUNCT2 and may be used to pass information to those routines.

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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified 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,N× (N+7)).

IFAIL = 2

There has been a large number of 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 to make the significance of IFAIL = 5 etc. consistent in the easy-to-use routines.)

IFAIL = 5, 6, 7 and 8

There is some doubt about whether the point x found by E04KZF 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 and 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 and it is possible to calculate second derivatives it may be advisable to change to a routine which uses second derivatives (see the Chapter Introduction).

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 E04KZF is roughly proportional to $n^3 + \mathrm{O}(n^2)$. In addition, each iteration makes at least $m + 1$ calls of FUNCT2 where m is the number of variables not fixed on bounds. 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 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 E04KZF 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 FUNCT2 as efficient as possible. This has not been done in the example program for reasons of clarity.)

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.

```
*     E04KZF Example Program Text.
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          N, LIW, LW
      PARAMETER        (N=4,LIW=N+2,LW=N*(N+7))
      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         E04KZF, FUNCT2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04KZF 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 E04KZF(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,99998) 'at the point', (X(J),J=1,N)
         WRITE (NOUT,*)
     +     'the corresponding (machine dependent) gradient is'
         WRITE (NOUT,99997) (G(J),J=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,4F12.4)
99997 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

```
E04KZF Example Program Results

Error exit type  5 - see routine document

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 -0.5872E-09  0.1177E-08  0.5907E+01
```

E04LBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

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, HESS, MONIT, IPRINT, MAXCAL, ETA, XTOL,
     1                   STEPMX, IBOUND, BL, BU, X, HESL, LH, HESD,
     2                   ISTATE, F, G, IW, LIW, W, LW, IFAIL)
      INTEGER           N, IPRINT, MAXCAL, IBOUND, LH, ISTATE(N), IW(LIW),
     1                  LIW, LW, IFAIL
      real              ETA, XTOL, STEPMX, BL(N), BU(N), X(N), HESL(LH),
     1                  HESD(N), F, G(N), W(LW)
      EXTERNAL          FUNCT, HESS, MONIT
```

3. Description

This routine is applicable to problems of the form:

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

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 $l_1 = l_2 = ... = l_n$ and $u_1 = u_2 = ... = u_n$. It is possible to specify that a particular x_j should be held constant. The user must supply a starting point, a subroutine FUNCT to calculate the value of $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at any point x, and a subroutine HESS 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 the user specifies 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

[1] GILL, P.E. and MURRAY, W.
Safeguarded steplength algorithms for optimization using descent methods.
National Physical Laboratory Report NAC 37, 1974.

[2] GILL, P.E. and MURRAY, W.
Newton-type methods for unconstrained and linearly constrained optimization.
Mathematical Programming, 7, pp. 311-350, 1974.

[3] GILL, P.E. and MURRAY, W.
Minimization subject to bounds on the variables.
National Physical Laboratory Report NAC 72, 1976.

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 the user does 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.)

Its specification is:

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

1: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG 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) the user should reset IFLAG to some negative number and return control to E04LBF. E04LBF will then terminate immediately with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FC – ***real***. *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*** 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 $j = 1,2,...,n$.

6: IW(LIW) – INTEGER array. *Workspace*
7: LIW – INTEGER. *Input*
8: W(LW) – ***real*** 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, users **could** use elements 3,4,...,LIW of IW and elements from max(8,7×N+N×(N−1)/2) + 1 onwards of W for passing quantities to FUNCT from the (sub)program which calls E04LBF. However, because of the danger of mistakes in partitioning, it is recommended that users should pass information to FUNCT via COMMON 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×N+N×(N−1)/2) elements of W.

Note: FUNCT should be tested separately before being used in conjunction with E04LBF. It must be declared as EXTERNAL in the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

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

Its specification is:

```
SUBROUTINE HESS(IFLAG, N, XC, FHESL, LH, FHESD, IW, LIW, W, LW)
INTEGER    IFLAG, N, LH, IW(LIW), LIW, LW
real       XC(N), FHESL(LH), FHESD(N), W(LW)
```

1: IFLAG – INTEGER. *Input/Output*

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

On exit: if HESS resets IFLAG to some negative number, E04LBF will terminate immediately with IFAIL set to the user's setting of IFLAG.

2: N – INTEGER. *Input*

On entry: the number n of variables.

3: XC(N) – ***real*** array. *Input*

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

4: FHESL(LH) – ***real*** array. *Output*

On exit: unless IFLAG is reset, HESS 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 FHESL $((i-1)(i-2)/2+j) = \left.\dfrac{\partial^2 F}{\partial x_i \partial x_j}\right|_{XC}$, for $i = 2,3,...,n$; $j = 1,2,...,i-1$. (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*** array. *Input/Output*

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

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

On exit: unless IFLAG is reset, HESS must place the diagonal elements of the second derivative matrix of F (evaluated at the point x) in FHESD, i.e. set

$$\text{FHESD}(j) = \left.\frac{\partial^2 F}{\partial x_j^2}\right|_{\text{XC}}, \qquad j = 1,2,\ldots,n.$$

7: IW(LIW) – INTEGER array. *Workspace*
8: LIW – INTEGER. *Input*
9: W(LW) – ***real*** array. *Workspace*
10: LW – INTEGER. *Input*

As in FUNCT, these parameters correspond to the parameters IW, LIW, W, LW of E04LBF. HESS **must not change** the first two elements of IW or the first max(8, 7×N+N×(N−1)/2) elements of W. Again, it is recommended that the user should pass quantities to HESS via COMMON and not use IW or W at all.

Note: HESS should be tested separately before being used in conjunction with E04LBF. It must be declared as EXTERNAL in the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

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

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

Its specification is:

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

1: N – INTEGER. *Input*

On entry: the number n of variables.

2: XC(N) – ***real*** array. *Input*

On entry: the co-ordinates of the current point x.

3: FC – ***real***. *Input*

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

4: GC(N) – ***real*** array. *Input*

On entry: the value of $\dfrac{\partial F}{\partial x_j}$ 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***. *Input*

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

7: COND – ***real.*** *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 HESS below). 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: POSDEF 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*** array. *Workspace*

14: LW – INTEGER. *Input*

As in FUNCT, and HESS, 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.

The user should normally print out FC, GPJNRM and COND so as to be able to compare the quantities mentioned in Section 7. It is usually helpful to examine XC, POSDEF and NF as well. MONIT must be declared as EXTERNAL in the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: IPRINT – INTEGER. *Input*

On entry: the frequency with which MONIT is to be called. If IPRINT > 0, MONIT is called once every IPRINT iterations and just before exit from E04LBF. If IPRINT = 0, MONIT is just called at the final point. If 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×N.

Constraint: MAXCAL ≥ 1.

7: ETA – ***real.*** *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 1-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 ≤ ETA < 1.0.

8: XTOL – ***real.*** *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 prior to a normal exit, is such that $\|x_{sol}-x_{true}\| < \text{XTOL}\times(1.0+\|x_{true}\|)$, 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 5 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 $\sqrt{\varepsilon}$ is probably the smallest reasonable choice for XTOL. (This is because, normally, to machine accuracy, $F(x+\sqrt{\varepsilon}e_j) = F(x)$ where e_j is any column of the identity matrix.) If the user sets XTOL to 0.0 (or any positive value less than ε), E04LBF will use $10.0\times\sqrt{\varepsilon}$ instead of XTOL.

Suggested value: XTOL = 0.0.

Constraint: XTOL ≥ 0.0.

9: STEPMX – ***real.*** *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (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 ≥ 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. IBOUND should be set to one of the following values:

IBOUND = 0

if the variables are bounded and the user will be 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

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 ≤ IBOUND ≤ 4.

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

On entry: the fixed lower bounds l_j.

If IBOUND is set to 0, the user must set BL(j) to l_j, for j = 1,2,...,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, the user 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 initialised by E04LBF.

On exit: the lower bounds actually used by E04LBF, e.g. If IBOUND = 2, BL(1) = BL(2) = ... = BL(N) = 0.0.

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

On entry: the fixed upper bounds u_j.

If IBOUND is set to 0, the user must set BU(j) to u_j, for j = 1,2,...,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, the user 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 initialised by E04LBF.

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

13: X(N) – ***real*** array. *Input/Output*

On entry: X(j) must be set to a guess at the jth component of the position of minimum, for j = 1,2,...,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*** array. *Output*

See description of HESD below.

15: LH – INTEGER. *Input*

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

Constraint: LH ≥ max(N×(N−1)/2,1).

16: HESD(N) – ***real*** 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***. *Output*

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

19: G(N) – ***real*** 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. *Workspace*
21: LIW – INTEGER. *Input*

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

Constraint: LIW $\geq$ 2.

22: W(LW) – ***real*** array. *Workspace*
23: LW – INTEGER. *Input*

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

Constraint: LW $\geq$ max(7×N+N×(N–1)/2,8).

24: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04LBF because the user has set IFLAG negative in FUNCT or HESS. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N < 1,
or MAXCAL < 1,
or ETA < 0.0,
or ETA $\geq$ 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,N×(N−1)/2),
or LIW < 2,
or LW < max(8,7×N+N×(N−1)/2).

(Note that if the user has 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 recognise the point as a minimum.

IFAIL = 4

Not used. (This is done to make the significance of IFAIL = 5 similar for E04JBF, E04KBF, 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. The user should investigate as for IFAIL = 3.

The values IFAIL = 2, 3 and 5 may also be caused by mistakes in FUNCT or HESS, 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

$$\text{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{XTOL}+\sqrt{\varepsilon}) \times (1.0+\|x^{(k)}\|)$$

$$\text{B2} \equiv |F^{(k)}-F^{(k-1)}| < (\text{XTOL}^2+\varepsilon) \times (1.0+|F^{(k)}|)$$

$$\text{B3} \equiv \|g_z^{(k)}\| < (\varepsilon^{\frac{1}{3}}+\text{XTOL}) \times (1.0+|F^{(k)}|)$$

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

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

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

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

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

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (2) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. 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 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 E04LBF is $\frac{n_z^3}{6} + O(n_z^2)$. In addition, each iteration makes one call of HESS 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 HESS.

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 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 E04LBF 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 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,HESS,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, HESS, 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,HESS,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

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

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

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

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.

```
*     E04LBF Example Program Text.
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LH, LIW, LW
      PARAMETER        (N=4,LH=N*(N-1)/2,LIW=2,LW=7*N+N*(N-1)/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ETA, F, STEPMX, XTOL
      INTEGER          IBOUND, IFAIL, IPRINT, J, MAXCAL
*     .. Local Arrays ..
      real             BL(N), BU(N), G(N), HESD(N), HESL(LH), W(LW),
     +                 X(N)
      INTEGER          ISTATE(N), IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         E04HCF, E04HDF, E04LBF, FUNCT, HESS, MONIT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04LBF Example Program Results'
*     Set up an arbitrary point at which to check the derivatives
      X(1) = 1.46e0
      X(2) = -0.82e0
      X(3) = 0.57e0
      X(4) = 1.21e0
*     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,HESS,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 = 0.9e0
*     Set XTOL to zero so that E04LBF will use the default tolerance
      XTOL = 0.0e0
```

```
*     We estimate that the minimum will be within 4 units of the
*     starting point
      STEPMX = 4.0e0
      IBOUND = 0
      BL(1) = 1.0e0
      BU(1) = 3.0e0
      BL(2) = -2.0e0
      BU(2) = 0.0e0
*     X(3) is not bounded, 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
*     Set up starting point
      X(1) = 3.0e0
      X(2) = -1.0e0
      X(3) = 0.0e0
      X(4) = 1.0e0
      IFAIL = 1
*
      CALL E04LBF(N,FUNCT,HESS,MONIT,IPRINT,MAXCAL,ETA,XTOL,STEPMX,
     +            IBOUND,BL,BU,X,HESL,LH,HESD,ISTATE,F,G,IW,LIW,W,LW,
     +            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)
         WRITE (NOUT,99995) 'ISTATE contains', (ISTATE(J),J=1,N)
         WRITE (NOUT,99994) 'and HESD contains', (HESD(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 (23X,1P,4e12.3)
99995 FORMAT (1X,A,4I5)
99994 FORMAT (1X,A,4e12.4)
      END
*
      SUBROUTINE FUNCT(IFLAG,N,XC,FC,GC,IW,LIW,W,LW)
*     Routine to evaluate objective function and its 1st derivatives.
*     .. Scalar Arguments ..
      real              FC
      INTEGER           IFLAG, LIW, LW, N
*     .. Array Arguments ..
      real              GC(N), W(LW), XC(N)
      INTEGER           IW(LIW)
*     .. Executable Statements ..
      FC = (XC(1)+10.0e0*XC(2))**2 + 5.0e0*(XC(3)-XC(4))**2 + (XC(2)
     +     -2.0e0*XC(3))**4 + 10.0e0*(XC(1)-XC(4))**4
      GC(1) = 2.0e0*(XC(1)+10.0e0*XC(2)) + 40.0e0*(XC(1)-XC(4))**3
      GC(2) = 20.0e0*(XC(1)+10.0e0*XC(2)) + 4.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(3) = 10.0e0*(XC(3)-XC(4)) - 8.0e0*(XC(2)-2.0e0*XC(3))**3
      GC(4) = 10.0e0*(XC(4)-XC(3)) - 40.0e0*(XC(1)-XC(4))**3
      RETURN
      END
*
```

```
      SUBROUTINE HESS(IFLAG,N,XC,FHESL,LH,FHESD,IW,LIW,W,LW)
*     Routine to evaluate 2nd derivatives
*     .. Scalar Arguments ..
      INTEGER          IFLAG, LH, LIW, LW, N
*     .. Array Arguments ..
      real             FHESD(N), FHESL(LH), W(LW), XC(N)
      INTEGER          IW(LIW)
*     .. Executable Statements ..
      FHESD(1) = 2.0e0 + 120.0e0*(XC(1)-XC(4))**2
      FHESD(2) = 200.0e0 + 12.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESD(3) = 10.0e0 + 48.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESD(4) = 10.0e0 + 120.0e0*(XC(1)-XC(4))**2
      FHESL(1) = 20.0e0
      FHESL(2) = 0.0e0
      FHESL(3) = -24.0e0*(XC(2)-2.0e0*XC(3))**2
      FHESL(4) = -120.0e0*(XC(1)-XC(4))**2
      FHESL(5) = 0.0e0
      FHESL(6) = -10.0e0
      RETURN
      END
*
      SUBROUTINE MONIT(N,XC,FC,GC,ISTATE,GPJNRM,COND,POSDEF,NITER,NF,IW,
     +                 LIW,W,LW)
*     Monitoring routine
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             COND, FC, GPJNRM
      INTEGER          LIW, LW, N, NF, NITER
      LOGICAL          POSDEF
*     .. Array Arguments ..
      real             GC(N), W(LW), XC(N)
      INTEGER          ISTATE(N), IW(LIW)
*     .. Local Scalars ..
      INTEGER          ISJ, J
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +' Itn     Fn evals              Fn value             Norm of proj g
     +radient'
      WRITE (NOUT,99999) NITER, NF, FC, GPJNRM
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  ' J           X(J)                 G(J)           Status'
      DO 20 J = 1, N
         ISJ = ISTATE(J)
         IF (ISJ.GT.0) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Free'
         ELSE IF (ISJ.EQ.-1) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Upper Bound'
         ELSE IF (ISJ.EQ.-2) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Lower Bound'
         ELSE IF (ISJ.EQ.-3) THEN
            WRITE (NOUT,99998) J, XC(J), GC(J), '    Constant'
         END IF
   20 CONTINUE
      IF (COND.NE.0.0e0) THEN
         IF (COND.GT.1.0e6) 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
```

```
*           The following statement is included so that this MONIT
*           can also be used in conjunction with E04KDF
            WRITE (NOUT,*)
            WRITE (NOUT,*)
     +        'Projected Hessian matrix is not positive definite'
          END IF
          RETURN
        END IF
*
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
```

9.2. Program Data

None.

9.3. Program Results

```
 E04LBF Example Program Results

 Error exit type  3 - see routine document

 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  0.4552E+02  0.1750E+02
```

With IPRINT set to 1 in the example program, intermediate results similar to those below are obtained from MONIT:

```
 Itn     Fn evals              Fn value            Norm of proj gradient
  0          1                2.1500E+02                1.4401E+02

 J           X(J)                G(J)          Status
 1          3.0000E+00          3.0600E+02    Upper Bound
 2         -1.0000E+00         -1.4400E+02    Free
 3          0.0000E+00         -2.0000E+00    Free
 4          1.0000E+00         -3.1000E+02    Lower Bound

 Estimated condition number of projected Hessian =   3.83E+00

 Itn     Fn evals              Fn value            Norm of proj gradient
  1          2                1.6306E+02                3.2033E+02

 J           X(J)                G(J)          Status
 1          3.0000E+00          3.2033E+02    Free
 2         -2.8328E-01         -3.5132E-02    Free
 3          3.3106E-01          7.0265E-02    Free
 4          1.0000E+00         -3.1331E+02    Lower Bound

 Estimated condition number of projected Hessian =   9.46E+00
  .
  .
```

intermediate results omitted

```
  .
  .
```

```
 Itn      Fn evals              Fn value              Norm of proj gradient
  9          10                2.4338E+00                   1.3120E-09

 J           X(J)                 G(J)            Status
 1         1.0000E+00           2.9535E-01     Lower Bound
 2        -8.5233E-02          -5.8675E-10     Free
 3         4.0930E-01           1.1735E-09     Free
 4         1.0000E+00           5.9070E+00     Lower Bound

Estimated condition number of projected Hessian =   4.43E+00

 Itn      Fn evals              Fn value              Norm of proj gradient
 10          11                2.4338E+00                   1.3120E-09

 J           X(J)                 G(J)            Status
 1         1.0000E+00           2.9535E-01     Lower Bound
 2        -8.5233E-02          -5.8675E-10     Free
 3         4.0930E-01           1.1735E-09     Free
 4         1.0000E+00           5.9070E+00     Lower Bound

Estimated condition number of projected Hessian =   4.43E+00
```

E04LYF – NAG Fortran Library Routine Document

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,
     1                  LIW, W, LW, 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, HESS2
```

3 Description

This routine is applicable to problems of the form:

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

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 $l_1 = l_2 = \ldots = l_n$ 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 and a subroutine to calculate the second derivatives.

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.

4 References

[1] 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 ≥ 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$.

Constraint: $0 \leq$ IBOUND ≤ 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 derivatives $\dfrac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04LYF (see the the 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 its 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 E04LYF with the parameters IUSER and USER as supplied to E04LYF. 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 E04LYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

This routine must be supplied by the user to evaluate the elements $H_{ij} = (\partial^2 F)/(\partial x_i \partial x_j)$ 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 Chapter Introduction).

Its specification is:

```
SUBROUTINE HESS2(N, XC, HESLC, LH, HESDC, IUSER, USER)
INTEGER          N, LH, IUSER(*)
real             XC(N), HESLC(LH), HESDC(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 derivatives are required.

3: HESLC(LH) — ***real*** 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$((i-1)(i-2)/2+j) = \frac{\partial^2 F}{\partial x_i \partial x_j}$ for $i = 2, 3, \ldots, n$; $j = 1, 2, \ldots, i-1$. (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*** array *Output*
On exit: HESDC must contain the diagonal elements of the second derivative matrix, i.e., set HESDC$(j) = \frac{\partial^2 F}{\partial x_j^2}$ for $j = 1, 2, \ldots, n$.

6: IUSER(*) — INTEGER array *User Workspace*
7: USER(*) — ***real*** array *User Workspace*
HESS2 is called from E04LYF with the parameters IUSER and USER as supplied to E04LYF. The user is free to use the arrays IUSER and USER to supply information to HESS2 as an alternative to using COMMON.

HESS2 must be 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*** 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, the user 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*** 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, the user 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*** 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 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.

8: F — ***real*** *Output*

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

9: 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.

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*** 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*

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

IUSER is not used by E04LYF, but is passed directly to FUNCT2 and HESS2 and may be used to pass information to those routines.

15: USER(*) — ***real*** array *User Workspace*

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

USER is not used by E04LYF, but is passed directly to FUNCT2 and HESS2 and may be used to pass information to those routines.

16: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified 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,N$\times$ (N+7)).

IFAIL = 2

There have been 50 × 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, 6, 7 and 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 or the variables has become very large ($\sim 10^6$). This indicates that there is a mistake in FUNCT2 or HESS2, 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.

IFAIL = 11

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

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.

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 $n^3 + \mathrm{O}(n^2)$. In addition, each iteration makes one call of HESS2 and a 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 FUNCT2 and HESS2.

Ideally the problem should be scaled so that at the solution the value of $F(x)$ and the corresponding values of $x_1, x_{2,n}$ 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 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

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

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

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.

```
*     E04LYF Example Program Text.
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          N, LIW, LW
      PARAMETER        (N=4,LIW=N+2,LW=N*(N+7))
      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         E04LYF, FUNCT2, HESS2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04LYF 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 E04LYF(N,IBOUND,FUNCT2,HESS2,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,99998) 'at the point', (X(J),J=1,N)
         WRITE (NOUT,*)
     +     'The corresponding (machine dependent) gradient is'
         WRITE (NOUT,99997) (G(J),J=1,N)
       END IF
       STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,4F9.4)
99997 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
*
      SUBROUTINE HESS2(N,XC,HESLC,LH,HESDC,IUSER,USER)
*     Routine to evaluate 2nd derivatives.
*     .. Scalar Arguments ..
      INTEGER           LH, N
*     .. Array Arguments ..
      real              HESDC(N), HESLC(LH), USER(1), XC(N)
      INTEGER           IUSER(1)
*     .. Local Scalars ..
      real              X1, X2, X3, X4
*     .. Executable Statements ..
      X1 = XC(1)
      X2 = XC(2)
      X3 = XC(3)
      X4 = XC(4)
      HESDC(1) = 2.0e0 + 120.0e0*(X1-X4)**2
```

```
      HESDC(2) = 200.0e0 + 12.0e0*(X2-2.0e0*X3)**2
      HESDC(3) = 10.0e0 + 48.0e0*(X2-2.0e0*X3)**2
      HESDC(4) = 10.0e0 + 120.0e0*(X1-X4)**2
      HESLC(1) = 20.0e0
      HESLC(2) = 0.0e0
      HESLC(3) = -24.0e0*(X2-2.0e0*X3)**2
      HESLC(4) = -120.0e0*(X1-X4)**2
      HESLC(5) = 0.0e0
      HESLC(6) = -10.0e0
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
E04LYF Example Program Results

Error exit type  5 - see routine document

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
```