D01ALF – 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

D01ALF is a general purpose integrator which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a, b]$:

$$I = \int_a^b f(x)\,dx$$

where the integrand may have local singular behaviour at a finite number of points within the integration interval.

2 Specification

```
 SUBROUTINE D01ALF(F, A, B, NPTS, POINTS, EPSABS, EPSREL, RESULT,
1                  ABSERR, W, LW, IW, LIW, IFAIL)
 INTEGER           NPTS, LW, IW(LIW), LIW, IFAIL
 real              F, A, B, POINTS(NPTS), EPSABS, EPSREL, RESULT,
1                  ABSERR, W(LW)
 EXTERNAL          F
```

3 Description

D01ALF is based upon the QUADPACK routine QAGP (Piessens *et al.* [3]). It is very similar to D01AJF, but allows the user to supply 'break-points', points at which the function is known to be difficult. It is an adaptive routine, using the Gauss 10-point and Kronrod 21-point rules. The algorithm described by de Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ϵ-algorithm (Wynn [4]) to perform extrapolation. The user-supplied 'break-points' always occur as the end-points of some sub-interval during the adaptive process. The local error estimation is described by Piessens *et al.* [3].

4 References

[1] de Doncker E (1978) An adaptive extrapolation algorithm for automatic integration *SIGNUM Newsl.* **13 (2)** 12–18

[2] Malcolm M A and Simpson R B (1976) Local versus global strategies for adaptive quadrature *ACM Trans. Math. Software* **1** 129–146

[3] Piessens R, de Doncker–Kapenga E, Überhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer-Verlag

[4] Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96

5 Parameters

1: F — ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real             X
```

1: X — ***real*** *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01ALF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A — ***real*** *Input*

On entry: the lower limit of integration, a.

3: B — ***real*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: NPTS — INTEGER *Input*

On entry: the number of user-supplied break-points within the integration interval.

Constraint: NPTS ≥ 0.

5: POINTS(NPTS) — ***real*** array *Input*

On entry: the user-specified break-points.

Constraint: the break-points must all lie within the interval of integration (but may be supplied in any order).

6: EPSABS — ***real*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

7: EPSREL — ***real*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

8: RESULT — ***real*** *Input*

On entry: the approximation to the integral I.

9: ABSERR — ***real*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I - \text{RESULT}|$.

10: W(LW) — ***real*** array *Output*

On exit: details of the computation, as described in Section 8.

11: LW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01ALF is called.. The value of LW (together with that of LIW below) imposes a bound on the number of sub-intervals into which the interval of integration may be divided by the routine. The number of sub-intervals cannot exceed $(\text{LW} - 2 \times \text{NPTS} - 4)/4$. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: $\text{LW} \geq 2 \times \text{NPTS} + 8$.

12: IW(LIW) — INTEGER array *Output*

On exit: IW(1) contains the actual number of sub-intervals used. The rest of the array is used as workspace.

13: LIW — INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01ALF is called.. The number of sub-intervals into which the interval of integration may be divided cannot exceed $(\text{LIW} - \text{NPTS} - 2)/2$.

Suggested value: $\text{LIW} = \text{LW}/2$.

Constraint: $\text{LIW} \geq \text{NPTS} + 4$.

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**.

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 $= 1$

The maximum number of subdivisions allowed with the given workspace has been reached, without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g., a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) it should be supplied to the routine as an element of the vector POINTS. If necessary, another integrator should be used, which is designed for handling the type of difficulty involved. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL $= 2$

Round-off error prevents the requested tolerance from being achieved. The error may be underestimated. Consider requesting less accuracy.

IFAIL $= 3$

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL $= 1$.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the result returned is the best which can be obtained. The same advice applies as in the case $IFAIL = 1$.

IFAIL = 5

The integral is probably divergent, or slowly convergent. Please note that divergence can also occur with any other non-zero value of IFAIL.

IFAIL = 6

The input is invalid: break-points are specified outside the integration range, NPTS > LIMIT or NPTS < 0. RESULT and ABSERR are set to zero.

IFAIL = 7

On entry, LW < 2 × NPTS + 8,
or LIW < NPTS + 4.

7 Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I - \text{RESULT}| \leq tol,$$

where

$$tol = \max\{|\text{EPSABS}|, |\text{EPSREL}| \times |I|\},$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR which, in normal circumstances, satisfies

$$|I - \text{RESULT}| \leq \text{ABSERR} \leq tol.$$

8 Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

If IFAIL $\neq 0$ on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the sub-intervals used by D01ALF along with the integral contributions and error estimates over these sub-intervals.

Specifically, for $i = 1, 2, \ldots, n$, let r_i denote the approximation to the value of the integral over the sub-interval $[a_i, b_i]$ in the partition of $[a, b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)\,dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$ unless D01ALF terminates while testing for divergence of the integral (see Piessens *et al.* [3], Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

$$\begin{aligned} a_i &= \text{W}(i), \\ b_i &= \text{W}(n+i), \\ e_i &= \text{W}(2n+i) \text{ and} \\ r_i &= \text{W}(3n+i). \end{aligned}$$

9 Example

To compute

$$\int_0^1 \frac{1}{\sqrt{|x - 1/7|}}\,dx.$$

A break-point is specified at $x = 1/7$, at which point the integrand is infinite. (For definiteness the function FST returns the value 0.0 at this point.)

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.

```
*     D01ALF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NPTS, LW, LIW
      PARAMETER        (NPTS=1,LW=800,LIW=LW/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             POINTS(NPTS), W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST
      EXTERNAL         FST
*     .. External Subroutines ..
      EXTERNAL         D01ALF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ALF Example Program Results'
      EPSABS = 0.0e0
      EPSREL = 1.0e-03
      A = 0.0e0
      B = 1.0e0
      POINTS(1) = 1.0e0/7.0e0
      KOUNT = 0
      IFAIL = -1
*
      CALL D01ALF(FST,A,B,NPTS,POINTS,EPSABS,EPSREL,RESULT,ABSERR,W,LW,
     +            IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,99999) 'POINTS(1) - given break-point = ', POINTS(1)
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997)
     +     '  RESULT - approximation to the integral = ', RESULT
         WRITE (NOUT,99998)
     +     '  ABSERR - estimate of the absolute error = ', ABSERR
         WRITE (NOUT,99996)
     +     '  KOUNT  - number of function evaluations = ', KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
```

```
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT
*     .. Local Scalars ..
      real              A
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      A = ABS(X-1.0e0/7.0e0)
      FST = 0.0e0
      IF (A.NE.0.0e0) FST = A**(-0.5e0)
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D01ALF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     1.0000
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-02
POINTS(1) - given break-point =     0.1429

  RESULT - approximation to the integral =   2.60757
  ABSERR - estimate of the absolute error =  0.60E-13
  KOUNT  - number of function evaluations =  462
IW(1)  - number of subintervals used =   12
```