NAG Library Routine Document

D05BEF

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

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,     &
                   WORK, LWK, NCT, IFAIL)

INTEGER             IORDER, NMESH, LWK, NCT(NMESH/32+1), IFAIL
REAL (KIND=nag_wp) 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

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

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

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

then the solution $y(t)$ is unique and has the form $y(t)=t^{\beta-1/2}z(t)$, (see Lubich (1987)). It is evident from (1) that $f(0)=0$. You are required to provide the value of $y(t)$ at $t=0$. If $y(0)$ is unknown, Section 9 gives a description of how an approximate value can be obtained.

The routine uses a fractional BDF linear multi-step method selected by you 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 Lubich (1987) and to Baker and Derakhshan (1987) and Hairer *et al.* (1988) 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 you 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

Baker C T H and Derakhshan M S (1987) FFT techniques in the numerical solution of convolution equations *J. Comput. Appl. Math.* **20** 5–24

Gorenflo R and Pfeiffer A (1991) On analysis and discretization of nonlinear Abel integral equations of first kind *Acta Math. Vietnam* **16** 211–262

Hairer E, Lubich Ch and Schlichte M (1988) Fast numerical solution of weakly singular Volterra integral equations *J. Comput. Appl. Math.* **23** 87–98

Lubich Ch (1987) Fractional linear multistep methods for Abel–Volterra integral equations of the first kind *IMA J. Numer. Anal* **7** 97–106

5 Parameters

1: CK – REAL (KIND=nag_wp) FUNCTION, supplied by the user. *External Procedure*

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

The specification of CK is:

```
FUNCTION CK (T)
REAL (KIND=nag_wp) CK
REAL (KIND=nag_wp) T
```

1: T – REAL (KIND=nag_wp) *Input*

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

CK must either be a module subprogram USEd by, or 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 (KIND=nag_wp) FUNCTION, supplied by the user. *External Procedure*

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

The specification of CF is:

```
FUNCTION CF (T)
REAL (KIND=nag_wp) CF
REAL (KIND=nag_wp) T
```

1: T – REAL (KIND=nag_wp) *Input*

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

CF must either be a module subprogram USEd by, or 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 (KIND=nag_wp) FUNCTION, supplied by the user. *External Procedure*

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

The specification of CG is:

```
FUNCTION CG (S, Y)
REAL (KIND=nag_wp) CG
REAL (KIND=nag_wp) S, Y
```

1: S – REAL (KIND=nag_wp) *Input*

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

2: Y – REAL (KIND=nag_wp) *Input*

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

CG must either be a module subprogram USEd by, or 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' (**I**nitial call).

If INITWT = 'S' (**S**ubsequent call), 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: p, the order of the BDF method to be used.

Suggested value: IORDER = 4.

Constraint: $4 \leq \text{IORDER} \leq 6$.

6: TLIM – REAL (KIND=nag_wp) *Input*

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

Constraint: TLIM $> 10 \times$ ***machine precision***.

7: TOLNL – REAL (KIND=nag_wp) *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 9).

Suggested value: TOLNL $= \sqrt{\epsilon}$ where ϵ is the ***machine precision***.

Constraint: TOLNL $> 10 \times$ ***machine precision***.

8: NMESH – INTEGER *Input*

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

Constraint: NMESH $= 2^m + 2 \times \text{IORDER} - 1$, where $m \geq 1$.

9: YN(NMESH) – REAL (KIND=nag_wp) array *Input/Output*

On entry: YN(1) must contain the value of $y(t)$ at $t = 0$ (see Section 9).

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

10: WORK(LWK) – REAL (KIND=nag_wp) array *Communication Array*

On entry: if INITWT = 'S', WORK must contain fractional weights computed by a previous call of D05BEF (see description of INITWT).

On exit: contains fractional weights which may be used by a subsequent call of D05BEF.

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 $\geq (2 \times \text{IORDER} + 6) \times \text{NMESH} + 8 \times \text{IORDER}^2 - 16 \times \text{IORDER} + 1$.

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

13: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value −1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value −1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

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 ≠ $2^m + 2 \times$ 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 9 for further details).

IFAIL = 3

The routine cannot compute the solution at a specific step due to an error in the solution of the single nonlinear equation (3). Relaxing the value of TOLNL and/or increasing the value of NMESH may overcome this problem (see Section 9 for further details).

IFAIL = −99

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL = −399

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL = −999

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

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 $\sqrt{\epsilon}$, where ϵ is the ***machine precision***, should be sufficient.

8 Parallelism and Performance

D05BEF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D05BEF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

Also when solving (1) the initial value $y(0)$ is required. This value may be computed from the limit relation (see Gorenflo and Pfeiffer (1991))

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

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 Gorenflo and Pfeiffer (1991), 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 you 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. C05QDF is used for this purpose. If a failure with IFAIL $= 2$ occurs (corresponding to an error exit from C05QDF), you 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), you 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 you 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.

10 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 Hairer *et al.* (1988))

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

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

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

Example 2

In this example we consider the equation

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

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

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

10.1 Program Text

```
!   D05BEF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.
    Module d05befe_mod

!     D05BEF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                                 :: cf1, cf2, cg1, cg2, ck1, ck2,    &
                                                sol1, sol2
!     .. Parameters ..
      Integer, Parameter, Public             :: iorder = 4
      Integer, Parameter, Public             :: nmesh = 2**6 + 2*iorder - 1
      Integer, Parameter, Public             :: nout = 6
      Integer, Parameter, Public             :: lct = nmesh/32 + 1
      Integer, Parameter, Public             ::                                  &
                     lwk = (2*iorder+6)*nmesh + 8*iorder*iorder- 16*iorder + 1
    Contains
      Function sol1(t)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. Function Return Value ..
        Real (Kind=nag_wp)                   :: sol1
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: c, pi, t1
!       .. Intrinsic Procedures ..
        Intrinsic                            :: exp, sqrt
!       .. Executable Statements ..
        t1 = 1.0_nag_wp + t
        c = 1.0_nag_wp/sqrt(2.0_nag_wp*x01aaf(pi))
```

```
        sol1 = c*(1.0_nag_wp/(t**1.5_nag_wp))*exp(-t1*t1/(2.0_nag_wp*t))

        Return

      End Function sol1
      Function sol2(t)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: sol2
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t
!       .. Executable Statements ..
        sol2 = 1.0_nag_wp/(1.0_nag_wp+t)

        Return

      End Function sol2
      Function ck1(t)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: ck1
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t
!       .. Intrinsic Procedures ..
        Intrinsic                           :: exp
!       .. Executable Statements ..
        ck1 = exp(-0.5_nag_wp*t)

        Return

      End Function ck1
      Function cf1(t)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: cf1
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: a, pi, t1
!       .. Intrinsic Procedures ..
        Intrinsic                           :: exp, sqrt
!       .. Executable Statements ..
        t1 = 1.0_nag_wp + t
        a = 1.0_nag_wp/sqrt(x01aaf(pi)*t)
        cf1 = -a*exp(-0.5_nag_wp*t1*t1/t)

        Return

      End Function cf1
      Function cg1(s,y)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: cg1
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: s, y
!       .. Executable Statements ..
        cg1 = y

        Return

      End Function cg1
      Function ck2(t)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: ck2
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t
```

```
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: pi
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        ck2 = sqrt(x01aaf(pi))

        Return

      End Function ck2
      Function cf2(t)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                   :: cf2
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: st1
!       .. Intrinsic Procedures ..
        Intrinsic                            :: log, sqrt
!       .. Executable Statements ..
        st1 = sqrt(1.0_nag_wp+t)
        cf2 = -2.0_nag_wp*log(st1+sqrt(t))/st1

        Return

      End Function cf2
      Function cg2(s,y)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                   :: cg2
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: s, y
!       .. Executable Statements ..
        cg2 = y

        Return

      End Function cg2
    End Module d05befe_mod
    Program d05befe

!     D05BEF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d05bef, nag_wp, x02ajf
      Use d05befe_mod, Only: cf1, cf2, cg1, cg2, ck1, ck2, iorder, lct, lwk,   &
                             nmesh, nout, sol1, sol2
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                     :: err, errmax, h, hi1, soln, t,    &
                                                tlim, tolnl
      Integer                                :: i, ifail
!     .. Local Arrays ..
      Real (Kind=nag_wp)                     :: work(lwk), yn(nmesh)
      Integer                                :: nct(lct)
!     .. Intrinsic Procedures ..
      Intrinsic                              :: abs, mod, real, sqrt
!     .. Executable Statements ..
      Write (nout,*) 'D05BEF Example Program Results'

      tlim = 7.0_nag_wp
      tolnl = sqrt(x02ajf())
      h = tlim/real(nmesh-1,kind=nag_wp)
      yn(1) = 0.0_nag_wp

      ifail = 0
      Call d05bef(ck1,cf1,cg1,'Initial',iorder,tlim,tolnl,nmesh,yn,work,lwk, &
        nct,ifail)
```

```
      Write (nout,*)
      Write (nout,*) 'Example 1'
      Write (nout,*)
      Write (nout,99997) h
      Write (nout,*)
      Write (nout,*) '     T        Approximate'
      Write (nout,*) '                Solution '
      Write (nout,*)

      errmax = 0.0_nag_wp

      Do i = 2, nmesh
        hi1 = real(i-1,kind=nag_wp)*h
        err = abs(yn(i)-sol1(hi1))

        If (err>errmax) Then
          errmax = err
          t = hi1
          soln = yn(i)
        End If

        If (i>5 .And. mod(i,5)==1) Then
          Write (nout,99998) hi1, yn(i)
        End If

      End Do

      Write (nout,*)
      Write (nout,99999) errmax, t, soln
      Write (nout,*)

      tlim = 5.0_nag_wp
      h = tlim/real(nmesh-1,kind=nag_wp)
      yn(1) = 1.0_nag_wp

      ifail = 0
      Call d05bef(ck2,cf2,cg2,'Subsequent',iorder,tlim,tolnl,nmesh,yn,work, &
        lwk,nct,ifail)

      Write (nout,*)
      Write (nout,*) 'Example 2'
      Write (nout,*)
      Write (nout,99997) h
      Write (nout,*)
      Write (nout,*) '     T        Approximate'
      Write (nout,*) '                Solution '
      Write (nout,*)

      errmax = 0.0_nag_wp

      Do i = 1, nmesh
        hi1 = real(i-1,kind=nag_wp)*h
        err = abs(yn(i)-sol2(hi1))

        If (err>errmax) Then
          errmax = err
          t = hi1
          soln = yn(i)
        End If

        If (i>7 .And. mod(i,7)==1) Then
          Write (nout,99998) hi1, yn(i)
        End If

      End Do

      Write (nout,*)
      Write (nout,99999) errmax, t, soln
```

```
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 Program d05befe
```

10.2 Program Data

None.

10.3 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
```