NAG Library Routine Document

D02PEF

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

D02PEF solves an initial value problem for a first-order system of ordinary differential equations using Runge–Kutta methods.

2 Specification

```
SUBROUTINE D02PEF (F, N, TWANT, TGOT, YGOT, YPGOT, YMAX, IUSER, RUSER,     &
                   IWSAV, RWSAV, IFAIL)

INTEGER            N, IUSER(*), IWSAV(130), IFAIL
REAL (KIND=nag_wp) TWANT, TGOT, YGOT(N), YPGOT(N), YMAX(N), RUSER(*),      &
                   RWSAV(32*N+350)
EXTERNAL           F
```

3 Description

D02PEF and its associated routines (D02PQF, D02PTF and D02PUF) solve an initial value problem for a first-order system of ordinary differential equations. The routines, based on Runge–Kutta methods and derived from RKSUITE (see Brankin *et al.* (1991)), integrate

$$y' = f(t, y) \quad \text{given} \quad y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

D02PEF is designed for the usual task, namely to compute an approximate solution at a sequence of points. You must first call D02PQF to specify the problem and how it is to be solved. Thereafter you call D02PEF repeatedly with successive values of TWANT, the points at which you require the solution, in the range from TSTART to TEND (as specified in D02PQF). In this manner D02PEF returns the point at which it has computed a solution TGOT (usually TWANT), the solution there (YGOT) and its derivative (YPGOT). If D02PEF encounters some difficulty in taking a step toward TWANT, then it returns the point of difficulty (TGOT) and the solution and derivative computed there (YGOT and YPGOT, respectively).

In the call to D02PQF you can specify either the first step size for D02PEF to attempt or that it computes automatically an appropriate value. Thereafter D02PEF estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to D02PEF by a call to D02PTF. The local error is controlled at every step as specified in D02PQF. If you wish to assess the true error, you must set METHOD to a positive value in the call to D02PQF. This assessment can be obtained after any call to D02PEF by a call to D02PUF.

For more complicated tasks, you are referred to routines D02PFF, D02PRF and D02PSF, all of which are used by D02PEF.

4 References

Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: A suite of Runge–Kutta codes for the initial value problems for ODEs *SoftReport 91-S1* Southern Methodist University

5 Arguments

1: F – SUBROUTINE, supplied by the user. *External Procedure*

F must evaluate the functions f_i (that is the first derivatives y_i') for given values of the arguments t, y_i.

The specification of F is:

```
SUBROUTINE F (T, N, Y, YP, IUSER, RUSER)
INTEGER            N, IUSER(*)
REAL (KIND=nag_wp) T, Y(N), YP(N), RUSER(*)
```

1: T – REAL (KIND=nag_wp) *Input*

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

2: N – INTEGER *Input*

On entry: n, the number of ordinary differential equations in the system to be solved.

3: Y(N) – REAL (KIND=nag_wp) array *Input*

On entry: the current values of the dependent variables, y_i, for $i = 1, 2, \ldots, n$.

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

On exit: the values of f_i, for $i = 1, 2, \ldots, n$.

5: IUSER(∗) – INTEGER array *User Workspace*

6: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

F is called with the arguments IUSER and RUSER as supplied to D02PEF. You should use the arrays IUSER and RUSER to supply information to F.

F must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02PEF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER *Input*

On entry: n, the number of ordinary differential equations in the system to be solved.

Constraint: N ≥ 1.

3: TWANT – REAL (KIND=nag_wp) *Input*

On entry: t, the next value of the independent variable where a solution is desired.

Constraint: TWANT must be closer to TEND than the previous value of TGOT (or TSTART on the first call to D02PEF); see D02PQF for a description of TSTART and TEND. TWANT must not lie beyond TEND in the direction of integration.

4: TGOT – REAL (KIND=nag_wp) *Output*

On exit: t, the value of the independent variable at which a solution has been computed. On successful exit with IFAIL $= 0$, TGOT will equal TWANT. On exit with IFAIL > 1, a solution has still been computed at the value of TGOT but in general TGOT will not equal TWANT.

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

On entry: on the first call to D02PEF, YGOT need not be set. On all subsequent calls YGOT must remain unchanged.

On exit: an approximation to the true solution at the value of TGOT. At each step of the integration to TGOT, the local error has been controlled as specified in D02PQF. The local error has still been controlled even when TGOT $\neq$ TWANT, that is after a return with IFAIL > 1.

6: YPGOT(N) – REAL (KIND=nag_wp) array *Output*

On exit: an approximation to the first derivative of the true solution at TGOT.

7: YMAX(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: on the first call to D02PEF, YMAX need not be set. On all subsequent calls YMAX must remain unchanged.

On exit: YMAX(i) contains the largest value of $|y_i|$ computed at any step in the integration so far.

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

IUSER and RUSER are not used by D02PEF, but are passed directly to F and should be used to pass information to this routine.

10: IWSAV(130) – INTEGER array *Communication Array*
11: RWSAV($32 \times \mathrm{N} + 350$) – REAL (KIND=nag_wp) array *Communication Array*

On entry: these must be the same arrays supplied in a previous call to D02PQF. They must remain unchanged between calls.

On exit: information about the integration for use on subsequent calls to D02PEF or other associated routines.

12: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output arguments may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

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

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, a previous call to the setup routine has not been made or the communication arrays have become corrupted.

On entry, N $= \langle value \rangle$, but the value passed to the setup routine was N $= \langle value \rangle$.

On entry, the communication arrays have become corrupted, or a catastrophic error has already been detected elsewhere. You cannot continue integrating the problem.

TEND (setup) had already been reached in a previous call.
To start a new problem, you will need to call the setup routine.

TWANT does not lie in the direction of integration. TWANT = ⟨*value*⟩.

TWANT is too close to the last value of TGOT (TSTART on setup).
When using the method of order 8 at setup, these must differ by at least ⟨*value*⟩. Their absolute difference is ⟨*value*⟩.

TWANT lies beyond TEND (setup) in the direction of integration, but is very close to TEND. You may have intended TWANT = TEND.
|TWANT − TEND| = ⟨*value*⟩.

TWANT lies beyond TEND (setup) in the direction of integration.
TWANT = ⟨*value*⟩ and TEND = ⟨*value*⟩.

You cannot call this routine after it has returned an error.
You must call the setup routine to start another problem.

You cannot call this routine when you have specified, in the setup routine, that the step integrator will be used.

IFAIL = 2

This routine is being used inefficiently because the step size has been reduced drastically many times to obtain answers at many points. Using the order 4 and 5 pair method at setup is more appropriate here. You can continue integrating this problem.

IFAIL = 3

Approximately ⟨*value*⟩ function evaluations have been used to compute the solution since the integration started or since this message was last printed. However, you can continue integrating the problem.

IFAIL = 4

Approximately ⟨*value*⟩ function evaluations have been used to compute the solution since the integration started or since this message was last printed. Your problem has been diagnosed as stiff. If the situation persists, it will cost roughly ⟨*value*⟩ times as much to reach TEND (setup) as it has cost to reach the current time. You should probably call routines intended for stiff problems. However, you can continue integrating the problem.

IFAIL = 5

In order to satisfy your error requirements the solver has to use a step size of ⟨*value*⟩ at the current time, ⟨*value*⟩. This step size is too small for the ***machine precision***, and is smaller than ⟨*value*⟩.

IFAIL = 6

The global error assessment algorithm failed at start of integration.
The integration is being terminated.

The global error assessment may not be reliable for times beyond ⟨*value*⟩.
The integration is being terminated.

IFAIL = −99

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL = −399

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

The accuracy of integration is determined by the arguments TOL and THRESH in a prior call to D02PQF (see the routine document for D02PQF for further details and advice). Note that only the local error at each step is controlled by these arguments. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential system.

8 Parallelism and Performance

D02PEF 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

If D02PEF returns with IFAIL $= 5$ and the accuracy specified by TOL and THRESH is really required then you should consider whether there is a more fundamental difficulty. For example, the solution may contain a singularity. In such a region the solution components will usually be large in magnitude. Successive output values of YGOT and YMAX should be monitored (or D02PFF should be used since this takes one integration step at a time) with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary.

Performance statistics are available after any return from D02PEF by a call to D02PTF. If METHOD > 0 in the call to D02PQF, global error assessment is available after a return from D02PEF with IFAIL $= 0$, 2, 3, 4, 5 or 6 by a call to D02PUF.

After a failure with IFAIL $= 5$ or 6 each of the diagnostic routines D02PTF and D02PUF may be called only once.

If D02PEF returns with IFAIL $= 4$ then it is advisable to change to another code more suited to the solution of stiff problems. D02PEF will not return with IFAIL $= 4$ if the problem is actually stiff but it is estimated that integration can be completed using less function evaluations than already computed.

10 Example

This example solves the equation

$$y'' = -y, \quad y(0) = 0, \quad y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0, 2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. Relative error control is used with threshold values of 1.0E$-$8 for each solution component and compute the solution at intervals of length $\pi/4$ across the range. A low-order Runge–Kutta method (see D02PQF) is also used with tolerances TOL $=$ 1.0E$-$3 and TOL $=$ 1.0E$-$4 in turn so that the solutions can be compared.

See also Section 10 in D02PUF.

10.1 Program Text

```
!   D02PEF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02pefe_mod

!     D02PEF 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                             :: f
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: tol0 = 1.0E-3_nag_wp
      Integer, Parameter, Public         :: liwsav = 130, n = 2, nin = 5,        &
                                            nout = 6, npts = 8
      Integer, Parameter, Public         :: lrwsav = 350 + 32*n
    Contains

      Subroutine f(t,n,y,yp,iuser,ruser)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)            :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
        Real (Kind=nag_wp), Intent (In) :: y(n)
        Real (Kind=nag_wp), Intent (Out) :: yp(n)
        Integer, Intent (Inout)         :: iuser(*)
!       .. Executable Statements ..
        yp(1) = y(2)
        yp(2) = -y(1)
        Return
      End Subroutine f
    End Module d02pefe_mod

    Program d02pefe

!     D02PEF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02pef, d02pqf, d02ptf, nag_wp
      Use d02pefe_mod, Only: f, liwsav, lrwsav, n, nin, nout, npts, tol0
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                 :: hnext, hstart, tend, tgot, tinc,     &
                                            tol, tstart, twant, waste
      Integer                            :: fevals, i, ifail, j, method,         &
                                            stepcost, stepsok
!     .. Local Arrays ..
      Real (Kind=nag_wp)                 :: ruser(1), thresh(n), ygot(n),        &
                                            yinit(n), ymax(n), ypgot(n)
      Real (Kind=nag_wp), Allocatable    :: rwsav(:)
      Integer                            :: iuser(1)
      Integer, Allocatable               :: iwsav(:)
!     .. Intrinsic Procedures ..
      Intrinsic                          :: real
!     .. Executable Statements ..
      Write (nout,*) 'D02PEF Example Program Results'

      Allocate (iwsav(liwsav),rwsav(lrwsav))

!     Set initial conditions and input for D02PQF

!     Skip heading in data file
```

```
      Read (nin,*)
      Read (nin,*) method
      Read (nin,*) tstart, tend
      Read (nin,*) yinit(1:n)
      Read (nin,*) hstart
      Read (nin,*) thresh(1:n)

!     Set control for output

      tinc = (tend-tstart)/real(npts,kind=nag_wp)

      tol = 10.0_nag_wp*tol0

loop: Do i = 1, 2
        tol = tol*0.1_nag_wp

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d02pqf(n,tstart,tend,yinit,tol,thresh,method,hstart,iwsav,rwsav,  &
          ifail)

        Write (nout,99999) tol
        Write (nout,99998)
        Write (nout,99997) tstart, yinit(1:n)
        twant = tstart
        Do j = 1, npts
          twant = twant + tinc

          ifail = 0
          Call d02pef(f,n,twant,tgot,ygot,ypgot,ymax,iuser,ruser,iwsav,rwsav,  &
            ifail)

          Write (nout,99997) tgot, ygot(1:n)
        End Do

        ifail = 0
        Call d02ptf(fevals,stepcost,waste,stepsok,hnext,iwsav,rwsav,ifail)
        Write (nout,99996) fevals

      End Do loop

99999 Format (/,' Calculation with TOL = ',1P,E8.1)
99998 Format (/,'    t         y1        y2',/)
99997 Format (1X,F6.3,2(3X,F7.3))
99996 Format (/,' Cost of the integration in evaluations of F is',I6)
    End Program d02pefe
```

10.2 Program Data

```
D02PEF Example Program Data
   1                              : method
   0.0     6.28318530717958647692 : tstart, tend
   0.0     1.0                    : yinit(1:n)
   0.0                            : hstart
   1.0E-8  1.0E-8                 : thresh(1:n)
```

10.3 Program Results

```
 D02PEF Example Program Results

 Calculation with TOL =  1.0E-03

    t         y1        y2

  0.000     0.000     1.000
  0.785     0.707     0.707
  1.571     0.999    -0.000
  2.356     0.706    -0.706
  3.142    -0.000    -0.999
```

```
 3.927    -0.706    -0.706
 4.712    -0.998     0.000
 5.498    -0.705     0.706
 6.283     0.001     0.997

Cost of the integration in evaluations of F is   430

Calculation with TOL =  1.0E-04

   t         y1        y2

 0.000     0.000     1.000
 0.785     0.707     0.707
 1.571     1.000    -0.000
 2.356     0.707    -0.707
 3.142    -0.000    -1.000
 3.927    -0.707    -0.707
 4.712    -1.000     0.000
 5.498    -0.707     0.707
 6.283     0.000     1.000

Cost of the integration in evaluations of F is   892
```