NAG Library Routine Document

D02PFF

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

D02PFF is a one-step routine for solving an initial value problem for a first-order system of ordinary differential equations using Runge–Kutta methods.

2 Specification

```
SUBROUTINE D02PFF (F, N, TNOW, YNOW, YPNOW, IUSER, RUSER, IWSAV, RWSAV,    &
                   IFAIL)

INTEGER            N, IUSER(*), IWSAV(130), IFAIL
REAL (KIND=nag_wp) TNOW, YNOW(N), YPNOW(N), RUSER(*), RWSAV(32*N+350)
EXTERNAL           F
```

3 Description

D02PFF and its associated routines (D02PQF, D02PRF, D02PSF, 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.

D02PFF is designed to be used in complicated tasks when solving systems of ordinary differential equations. You must first call D02PQF to specify the problem and how it is to be solved. Thereafter you (repeatedly) call D02PFF to take one integration step at a time from TSTART in the direction of TEND (as specified in D02PQF). In this manner D02PFF returns an approximation to the solution YNOW and its derivative YPNOW at successive points TNOW. If D02PFF encounters some difficulty in taking a step, the integration is not advanced and the routine returns with the same values of TNOW, YNOW and YPNOW as returned on the previous successful step. D02PFF tries to advance the integration as far as possible subject to passing the test on the local error and not going past TEND.

In the call to D02PQF you can specify either the first step size for D02PFF to attempt or that it computes automatically an appropriate value. Thereafter D02PFF estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to D02PFF 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 D02PFF by a call to D02PUF.

If you want answers at specific points there are two ways to proceed:

(i) The more efficient way is to step past the point where a solution is desired, and then call D02PSF to get an answer there. Within the span of the current step, you can get all the answers you want at very little cost by repeated calls to D02PSF. This is very valuable when you want to find where something happens, e.g., where a particular solution component vanishes. You cannot proceed in this way with METHOD $= 3$ or -3.

(ii) The other way to get an answer at a specific point is to set TEND to this value and integrate to TEND. D02PFF will not step past TEND, so when a step would carry it past, it will reduce the step size so as to produce an answer at TEND exactly. After getting an answer there (TNOW = TEND), you can reset TEND to the next point where you want an answer, and repeat. TEND could be reset by a call to D02PQF, but you should not do this. You should use D02PRF instead because it is both easier to use and much more efficient. This way of getting answers at specific points can be used with any of the available methods, but it is the only way with METHOD $= 3$ or -3. It can be inefficient. Should this be the case, the code will bring the matter to your attention.

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 $i = 1, 2, \dots, n$.

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

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

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 D02PFF. 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 D02PFF 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: TNOW – REAL (KIND=nag_wp) *Output*

On exit: t, the value of the independent variable at which a solution has been computed.

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

On exit: an approximation to the solution at TNOW. The local error of the step to TNOW was no greater than permitted by the specified tolerances (see D02PQF).

5: YPNOW(N) – REAL (KIND=nag_wp) array *Output*

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

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

IUSER and RUSER are not used by D02PFF, but are passed directly to F and should be used to pass information to this routine.

8: IWSAV(130) – INTEGER array *Communication Array*
9: 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 D02PFF or other associated routines.

10: 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 $= 1$

A call to this routine cannot be made after it has returned an error.
The setup routine must be called to start another problem.

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, as specified in the setup routine, has already been reached. To start a new problem, you will need to call the setup routine. To continue integration beyond TEND then D02PRF must first be called to reset TEND to a new end value.

IFAIL $= 2$

More than 100 output points have been obtained by integrating to TEND (as specified in the setup routine). They have been so clustered that it would probably be (much) more efficient to use the interpolation routine (if $|\mathrm{METHOD}| = 3$, switch to $|\mathrm{METHOD}| = 2$ at setup). However, you can continue integrating the problem.

IFAIL $= 3$

Approximately $\langle$*value*$\rangle$ 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 $\langle$*value*$\rangle$ 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 $\langle$*value*$\rangle$ 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 $\langle$*value*$\rangle$ at the current time, $\langle$*value*$\rangle$. This step size is too small for the ***machine precision***, and is smaller than $\langle$*value*$\rangle$.

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 $\langle$*value*$\rangle$.
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

D02PFF 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 D02PFF 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 YNOW should be monitored 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 D02PFF (except when IFAIL $= 1$) by a call to D02PTF. If METHOD > 0 in the call to D02PQF, global error assessment is available after any return from D02PFF (except when IFAIL $= 1$) 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 D02PFF returns with IFAIL $= 4$ then it is advisable to change to another code more suited to the solution of stiff problems. D02PFF 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 $y_2 = 1.0$. We use relative error control with threshold values of 1.0E$-$8 for each solution component and print the solution at each integration step across the range. We use a medium order Runge–Kutta method (METHOD $= 2$) with tolerances TOL $=$ 1.0E$-$4 and TOL $=$ 1.0E$-$5 in turn so that we may compare the solutions.

10.1 Program Text

```
!   D02PFF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02pffe_mod

!     D02PFF 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 :: tol1 = 1.0E-4_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: tol2 = 1.0E-5_nag_wp
      Integer, Parameter, Public      :: liwsav = 130, n = 2, nin = 5,       &
                                         nout = 6
      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 d02pffe_mod

    Program d02pffe

!     D02PFF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02pff, d02pqf, d02ptf, nag_wp
      Use d02pffe_mod, Only: f, liwsav, lrwsav, n, nin, nout, tol1, tol2
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: hnext, hstart, tend, tnow, tol,      &
                                          tstart, waste
      Integer                          :: i, ifail, method, stpcst, stpsok,    &
                                          totf
!     .. Local Arrays ..
      Real (Kind=nag_wp)               :: ruser(1)
      Real (Kind=nag_wp), Allocatable  :: rwsav(:), thres(:), ynow(:),         &
                                          ypnow(:), ystart(:)
      Integer                          :: iuser(1)
      Integer, Allocatable             :: iwsav(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02PFF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) method
      Allocate (thres(n),iwsav(liwsav),rwsav(lrwsav),ynow(n),ypnow(n),         &
        ystart(n))

!     Set initial conditions and input for D02PQF

      Read (nin,*) tstart, tend
      Read (nin,*) ystart(1:n)
      Read (nin,*) hstart
      Read (nin,*) thres(1:n)

      Do i = 1, 2
        If (i==1) Then
          tol = tol1
        End If
        If (i==2) Then
          tol = tol2
        End If

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

        Write (nout,99999) tol
        Write (nout,99998)
        Write (nout,99997) tstart, ystart(1:n)

loop:   Do
          ifail = 0
          Call d02pff(f,n,tnow,ynow,ypnow,iuser,ruser,iwsav,rwsav,ifail)

          If (ifail==0) Then
            Write (nout,99997) tnow, ynow(1:n)
            If (tnow>=tend) Then
```

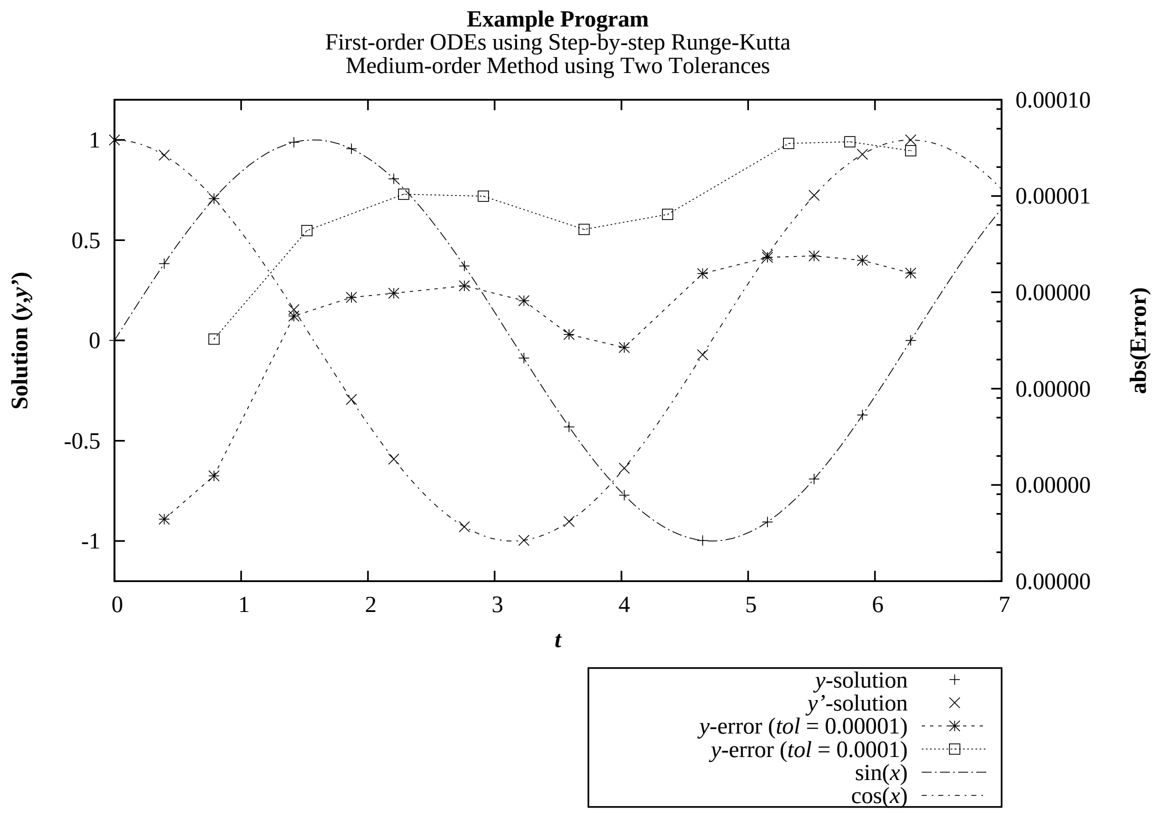
```
              Exit loop
            End If
          Else
            Exit loop
          End If

        End Do loop

        ifail = 0
        Call d02ptf(totf,stpcst,waste,stpsok,hnext,iwsav,rwsav,ifail)
        Write (nout,99996) totf
      End Do

99999 Format (/,' Calculation with TOL = ',E8.1)
99998 Format (/,'    t         y1        y2',/)
99997 Format (1X,F6.3,2(3X,F8.4))
99996 Format (/,' Cost of the integration in evaluations of F is',I6)
    End Program d02pffe
```

10.2 Program Data

```
D02PFF Example Program Data
   2                                : method
   0.0  6.28318530717958647692      : tstart, tend
   0.0  1.0                         : ystart(1:n)
   0.0                              : hstart
   1.0E-8  1.0E-8                   : thres(1:n)
```

10.3 Program Results

```
 D02PFF Example Program Results

 Calculation with TOL =  0.1E-03

    t         y1        y2

  0.000     0.0000     1.0000
  0.785     0.7071     0.7071
  1.519     0.9987     0.0513
  2.282     0.7573    -0.6531
  2.911     0.2285    -0.9735
  3.706    -0.5348    -0.8450
  4.364    -0.9399    -0.3414
  5.320    -0.8209     0.5710
  5.802    -0.4631     0.8863
  6.283     0.0000     1.0000

 Cost of the integration in evaluations of F is   204

 Calculation with TOL =  0.1E-04

    t         y1        y2

  0.000     0.0000     1.0000
  0.393     0.3827     0.9239
  0.785     0.7071     0.7071
  1.416     0.9881     0.1538
  1.870     0.9557    -0.2943
  2.204     0.8062    -0.5916
  2.761     0.3711    -0.9286
  3.230    -0.0880    -0.9961
  3.587    -0.4304    -0.9026
  4.022    -0.7710    -0.6368
  4.641    -0.9974    -0.0717
  5.152    -0.9049     0.4256
  5.521    -0.6903     0.7235
  5.902    -0.3718     0.9283
  6.283     0.0000     1.0000

 Cost of the integration in evaluations of F is   314
```



Example Program
First-order ODEs using Step-by-step Runge-Kutta
Medium-order Method using Two Tolerances
Solution (y,y')
abs(Error)
t
1
0.5
0
-0.5
-1
0.00010
0.00001
0.00000
0.00000
0.00000
0.00000
0 1 2 3 4 5 6 7
y-solution
y'-solution
y-error (tol = 0.00001)
y-error (tol = 0.0001)
sin(x)
cos(x)