NAG Library Routine Document

D02PUF

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

D02PUF provides details about global error assessment computed during an integration with either D02PEF or D02PFF.

2 Specification

```
SUBROUTINE D02PUF (N, RMSERR, ERRMAX, TERRMX, IWSAV, RWSAV, IFAIL)
INTEGER            N, IWSAV(130), IFAIL
REAL (KIND=nag_wp) RMSERR(N), ERRMAX, TERRMX, RWSAV(32*N+350)
```

3 Description

D02PUF and its associated routines (D02PEF, D02PFF, D02PQF, D02PRF, D02PSF and D02PTF) solve the 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.

After a call to D02PEF or D02PFF, D02PUF can be called for information about error assessment, if this assessment was specified in the setup routine D02PQF. A more accurate 'true' solution $\hat{y}$ is computed in a secondary integration. The error is measured as specified in D02PQF for local error control. At each step in the primary integration, an average magnitude μ_i of component y_i is computed, and the error in the component is

$$\frac{|y_i - \hat{y}_i|}{\max(\mu_i, \text{THRESH}(i))}.$$

It is difficult to estimate reliably the true error at a single point. For this reason the RMS (root-mean-square) average of the estimated global error in each solution component is computed. This average is taken over all steps from the beginning of the integration through to the current integration point. If all has gone well, the average errors reported will be comparable to TOL (see D02PQF). The maximum error seen in any component in the integration so far and the point where the maximum error first occurred are also reported.

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 Parameters

1: N – INTEGER *Input*

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

Constraint: N ≥ 1.

2: RMSERR(N) – REAL (KIND=nag_wp) array *Output*

On exit: RMSERR(i) approximates the RMS average of the true error of the numerical solution for the ith solution component, for $i = 1, 2, \ldots, n$. The average is taken over all steps from the beginning of the integration to the current integration point.

3: ERRMAX – REAL (KIND=nag_wp) *Output*

On exit: the maximum weighted approximate true error taken over all solution components and all steps.

4: TERRMX – REAL (KIND=nag_wp) *Output*

On exit: the first value of the independent variable where an approximate true error attains the maximum value, ERRMAX.

5: IWSAV(130) – INTEGER array *Communication Array*

6: 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 D02PEF or D02PFF. They must remain unchanged between calls.

On exit: information about the integration for use on subsequent calls to D02PEF or D02PFF or other associated routines.

7: 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 $= 1$

No error assessment is available since the integrator has not actually taken any successful steps.

No error assessment is available since you did not ask for it in your call to the setup routine.

On entry, a previous call to the setup routine has not been made or the communication arrays have become corrupted, or a catastrophic error has already been detected elsewhere.
You cannot continue integrating the problem.

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

You cannot call this routine before you have called the integrator.

You have already made one call to this routine after the integrator could not achieve specified accuracy.
You cannot call this routine again.

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

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

If the integration has proceeded 'well' and the problem is smooth enough, stable and not too difficult then the values returned in the arguments RMSERR and ERRMAX should be comparable to the value of TOL specified in the prior call to D02PQF.

10 Example

This example integrates a two body problem. The equations for the coordinates $(x(t), y(t))$ of one body as functions of time t in a suitable frame of reference are

$$x'' = -\frac{x}{r^3}$$

$$y'' = -\frac{y}{r^3}, \quad r = \sqrt{x^2 + y^2}.$$

The initial conditions

$$\begin{aligned} x(0) &= 1 - \epsilon, \quad x'(0) = 0 \\ y(0) &= 0, \qquad\quad y'(0) = \sqrt{\frac{1+\epsilon}{1-\epsilon}} \end{aligned}$$

lead to elliptic motion with $0 < \epsilon < 1$. $\epsilon = 0.7$ is selected and the system of ODEs is reposed as

$$y_1' = y_3$$

$$y_2' = y_4$$

$$y_3' = -\frac{y_1}{r^3}$$

$$y_4' = -\frac{y_2}{r^3}$$

over the range $[0, 3\pi]$. Relative error control is used with threshold values of 1.0E−10 for each solution component and a high-order Runge–Kutta method (METHOD = 3) with tolerance TOL = 1.0E−6.

Note that for illustration purposes since it is not necessary for this problem, this example integrates to the end of the range regardless of efficiency concerns (i.e., returns from D02PEF with IFAIL $= 2$, 3 or 4).

10.1 Program Text

```
!   D02PUF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module d02pufe_mod

!     D02PUF 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 ..
      Integer, Parameter, Public            :: liwsav = 130, n = 4, 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(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: r
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        r = sqrt(y(1)**2+y(2)**2)
        yp(1) = y(3)
        yp(2) = y(4)
        yp(3) = -y(1)/r**3
        yp(4) = -y(2)/r**3
        Return
      End Subroutine f
    End Module d02pufe_mod

    Program d02pufe

!     D02PUF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02pef, d02pqf, d02ptf, d02puf, nag_wp
      Use d02pufe_mod, Only: f, liwsav, lrwsav, n, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                     :: errmax, hnext, hstart, tend,     &
                                                terrmx, tgot, tol, tstart,       &
                                                twant, waste
      Integer                                :: fevals, ifail, method, stepcost, &
                                                stepsok
!     .. Local Arrays ..
      Real (Kind=nag_wp)                     :: rmserr(n), ruser(1), thresh(n),  &
                                                ygot(n), yinit(n), ymax(n),      &
                                                ypgot(n)
```

```
      Real (Kind=nag_wp), Allocatable      :: rwsav(:)
      Integer                              :: iuser(1)
      Integer, Allocatable                 :: iwsav(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02PUF Example Program Results'

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

!     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, tol
      Read (nin,*) thresh(1:n)

!     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 = tend

integ: Do
        ifail = -1
        Call d02pef(f,n,twant,tgot,ygot,ypgot,ymax,iuser,ruser,iwsav,rwsav, &
          ifail)

        If (ifail<2 .Or. ifail>4) Then
          Exit integ
        End If

      End Do integ

      If (ifail==0) Then

!       Print solution.
        Write (nout,99997) tgot, ygot(1:n)

!       Compute and print error estimates.
        ifail = 0
        Call d02puf(n,rmserr,errmax,terrmx,iwsav,rwsav,ifail)

        Write (nout,99996) rmserr(1:n)
        Write (nout,99995) errmax, terrmx

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

        Write (nout,99994) fevals
      End If

99999 Format (/' Calculation with TOL = ',E8.1)
99998 Format (/'    t          y1         y2         y3         y4'/)
99997 Format (1X,F6.3,4(3X,F8.4))
99996 Format (/' Componentwise error assessment'/9X,4(2X,E9.2))
99995 Format (/' Worst global error observed was ',E9.2, &
        ' - it occurred at T = ',F6.3)
99994 Format (/' Cost of the integration in evaluations of F is',I6)
    End Program d02pufe
```

10.2 Program Data

```
D02PUF Example Program Data
   3                                                 : method
   0.0                          9.42477796076937971538 : tstart, tend
   0.3      0.0      0.0        2.38047614284761666599 : yinit(1:n)
   0.0      1.0E-6                                    : hstart, tol
   1.0E-10  1.0E-10  1.0E-10    1.0E-10                : thresh(1:n)
```

10.3 Program Results

```
 D02PUF Example Program Results

 Calculation with TOL =  0.1E-05

    t          y1          y2          y3          y4

  0.000      0.3000      0.0000      0.0000      2.3805
  9.425     -1.7000      0.0000     -0.0000     -0.4201

 Componentwise error assessment
            0.38E-05    0.71E-05    0.69E-05    0.21E-05

 Worst global error observed was  0.34E-04 - it occurred at T =  6.302

 Cost of the integration in evaluations of F is  1361
```