NAG Library Routine Document

D02PGF

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

D02PGF is a reverse communication one-step routine for solving an initial value problem for a first-order system of ordinary differential equations using Runge–Kutta methods. The direct communication version of this routine is D02PFF. See Section 3.3.3 in How to Use the NAG Library and its Documentation for the difference between forward and reverse communication.

2 Specification

```
SUBROUTINE D02PGF (IREVCM, N, T, Y, YP, IWSAV, RWSAV, IFAIL)
INTEGER            IREVCM, N, IWSAV(130), IFAIL
REAL (KIND=nag_wp) T, Y(N), YP(N), RWSAV(32*N+350)
```

3 Description

D02PGF and its associated routines (D02PHF, D02PJF, D02PQF, D02PRF, 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.

D02PGF 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 D02PGF in reverse communication loops to take one integration step at a time from TSTART in the direction of TEND (as specified in D02PQF). In this manner D02PGF returns an approximation to the solution Y and its derivative YP at successive points T. If D02PGF encounters some difficulty in taking a step, the integration is not advanced and the routine returns with the same values of T, Y and YP as returned on the previous successful step. D02PGF 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 D02PGF to attempt or it computes automatically an appropriate value. Thereafter D02PGF estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after a completed step by D02PGF 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 D02PGF 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 D02PHF and D02PJF 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 D02PJF. This is very valuable when you want to find where something happens, e.g., where a particular solution component vanishes.

(ii) Alternatively, set TEND to the desired value and integrate to TEND. D02PGF 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 (T = 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 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

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **IREVCM**. Between intermediate exits and re-entries, **all arguments other than those specified by the value of IREVCM must remain unchanged**.

1: IREVCM – INTEGER *Input/Output*

On initial entry: IREVCM must be set to zero to indicate that a new step is being taken.

On intermediate re-entry: IREVCM should remain unchanged.

On intermediate exit: IREVCM returns a value > 0 to indicate that a function evaluation is required prior to re-entry; the value of the derivatives $y' = f(t, y)$ must be returned in YP where the value of t is supplied in T and the values $y(t)$ are supplied in the array Y. The value of IREVCM indicates the reason for the function evaluation as follows:

IREVCM $= 1$
For initial entry values of T and Y.

IREVCM $= 2$
To determine stiffness of system.

IREVCM $= 3$
For the stages of the primary step.

IREVCM $= 4$
A final stage of the primary step.

IREVCM $= 5$
For the stages of a secondary step (if global error assessment is required).

On final exit:

IREVCM $= -1$
Successful exit; T, Y and YP contain the solution at the end of a successful integration step.

IREVCM $= -2$
Error exit; IFAIL should be interrogated to determine the nature of the error.

2: N – INTEGER *Input*

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

Constraint: N ≥ 1. This must be the same value as supplied in a previous call to D02PQF.

3: T – REAL (KIND=nag_wp) *Output*

On intermediate exit: T contains the value of the independent variable t at which the derivatives y' are to be evaluated.

On final exit: the value of t at which a solution has been computed following a successful step.

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

On intermediate exit: Y contains the value of the solution y at which the derivatives y' are to be evaluated.

On final exit: the approximation to the solution computed following a successful step.

5: YP(N) – REAL (KIND=nag_wp) array *Input*

On initial entry: YP need not be set.

On intermediate re-entry: YP must contain the value of the derivatives $y' = f(t, y)$ where t is supplied in T and y is supplied in the array Y.

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

7: 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 D02PGF or other associated routines.

8: IFAIL – INTEGER *Input/Output*

On initial 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 final 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.

IREVCM < 0 on entry.

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, 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. 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

D02PGF 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 D02PGF returns with IFAIL > 1 then the values returned in T and Y are for the last successful step, or the initial conditions supplied if no successful step has been taken.

If D02PGF 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 Y 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 D02PGF (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 D02PGF (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 D02PGF returns with IFAIL $= 4$ then it is advisable to change to another code more suited to the solution of stiff problems. D02PGF 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 regular intervals using the interpolation routines D02PHF and D02PJF within integration steps across the range; points on the range at which y_1 or y_2 change sign are also evaluated using a combination of the root finding routine C05AZF and the interpolation routines. We use a medium order Runge–Kutta method (METHOD $= 2$) with tolerance TOL $=$ 1.0E−5.

10.1 Program Text

```
!   D02PGF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02pgfe_mod

!     D02PGF Example Program Module:
!            Parameters

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
```

```
      Implicit None
!     .. Accessibility Statements ..
      Private
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: tol = 1.0E-5_nag_wp
      Integer, Parameter, Public       :: liwsav = 130, n = 2, nin = 5,        &
                                          nout = 6
      Integer, Parameter, Public       :: lrwsav = 350 + 32*n
      Integer, Parameter, Public       :: lwcomm = 6*n
    End Module d02pgfe_mod

    Program d02pgfe

!     D02PGF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: c05azf, d02pgf, d02phf, d02pjf, d02pqf, d02ptf,   &
                             nag_wp
      Use d02pgfe_mod, Only: liwsav, lrwsav, lwcomm, n, nin, nout, tol
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: hnext, hstart, t, t1, t2, tend,      &
                                          tnow, tout, tprev, waste
      Integer                          :: i, icheck, ifail, ind, irevcm, j,    &
                                          method, nchange, stpcst, stpsok,     &
                                          totf
!     .. Local Arrays ..
      Real (Kind=nag_wp)               :: c(17)
      Real (Kind=nag_wp), Allocatable  :: rwsav(:), thres(:), troot(:),        &
                                          wcomm(:), y(:), ynow(:), yout(:),    &
                                          yp(:), ypnow(:), yprev(:)
      Integer, Allocatable             :: iroot(:), iwsav(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02PGF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) method
      Allocate (thres(n),iwsav(liwsav),rwsav(lrwsav),ynow(n),ypnow(n),         &
        yprev(n),wcomm(lwcomm),yout(n),iroot(n),y(n),yp(n),troot(n))

!     Set initial conditions and input for D02PQF

      Read (nin,*) t, tend
      Read (nin,*) ynow(1:n)
      Read (nin,*) hstart
      Read (nin,*) thres(1:n)

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

      Write (nout,99999) tol
      Write (nout,99998)
      Write (nout,99997) t, ynow(1:n)

      tout = 0.1_nag_wp
      tnow = t
      Do While (tnow<tend)
!       Integrate by one time step using reverse communication
        tprev = tnow
        yprev(1:n) = ynow(1:n)
        ifail = 1
        irevcm = 0
        Do While (irevcm>=0)
          Call d02pgf(irevcm,n,tnow,ynow,ypnow,iwsav,rwsav,ifail)
          If (irevcm>0) Then
            ypnow(1) = ynow(2)
            ypnow(2) = -ynow(1)
          End If
```

```
        End Do
        If (ifail==1 .Or. ifail>4) Then
          Write (6,99996) ifail, tnow
          Go To 100
        End If

!       Detect sign changes in last step
        iroot(1:n) = 0
        nchange = 0
        Do i = 1, n
          If (ynow(i)*yprev(i)<0.0_nag_wp) Then
            nchange = nchange + 1
            iroot(nchange) = i
          End If
        End Do

!       Interpolate for values of t at increments of 0.1 within step and
!       find roots of components of y.
        If (tnow>=tout .Or. nchange>0) Then
!         Set up interpolation by reverse communication
          ifail = 0
          irevcm = 0
          Do While (irevcm>=0)
            ifail = 0
            Call d02phf(irevcm,n,n,t,y,yp,wcomm,lwcomm,iwsav,rwsav,ifail)
            If (irevcm>0) Then
              yp(1) = y(2)
              yp(2) = -y(1)
            End If
          End Do

          icheck = 1
!         If there are sign changes: find roots
          Do i = 1, nchange
            j = iroot(i)
            t1 = tprev
            t2 = tnow
            ind = 1
            ifail = 0
            Do While (ind/=0)
              Call c05azf(t1,t2,y(j),tol,1,c,ind,ifail)
              If (ind>1) Then
                ifail = 0
                Call d02pjf(icheck,n,n,t1,0,y,wcomm,lwcomm,iwsav,rwsav,ifail)
                icheck = 0
              End If
            End Do
            troot(i) = t1
          End Do
!         Evaluate Interpolant at increments and print any contained roots
          ifail = 0
          Do While (tnow>=tout)
            Do i = 1, nchange
              If (troot(i)<tout .And. iroot(i)>0) Then
                Write (nout,99995) iroot(i), troot(i)
                iroot(i) = -iroot(i)
              End If
            End Do
            Call d02pjf(icheck,n,n,tout,0,yout,wcomm,lwcomm,iwsav,rwsav,ifail)
            icheck = 0
            Write (nout,99997) tout, yout(1:n)
            tout = tout + 0.1_nag_wp
          End Do
!         Print any remaining roots up to current time
          Do i = 1, nchange
            If (iroot(i)>0) Then
              Write (nout,99995) iroot(i), troot(i)
            End If
          End Do
        End If
      End Do
```

```
!     Print some integration statistics
      ifail = 0
      Call d02ptf(totf,stpcst,waste,stpsok,hnext,iwsav,rwsav,ifail)
      Write (nout,99994) totf
100   Continue

99999 Format (/,' Calculation with TOL = ',E8.1)
99998 Format (/,'    t         y1        y2',/)
99997 Format (1X,F6.3,2(3X,F8.4))
99996 Format (/,'D02PGF returned with ifail = ',I3,' at t = ',E12.5,'.')
99995 Format ('Component ',I2,' has a root at t = ',F7.4)
99994 Format (/,' Cost of the integration in evaluations of F is',I6)
    End Program d02pgfe
```

10.2 Program Data

```
D02PGF 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

```
 D02PGF Example Program Results

 Calculation with TOL =  0.1E-04

    t         y1        y2

  0.000     0.0000     1.0000
  0.100     0.0998     0.9950
  0.200     0.1987     0.9801
  0.300     0.2955     0.9553
  0.400     0.3894     0.9211
  0.500     0.4794     0.8776
  0.600     0.5646     0.8253
  0.700     0.6442     0.7648
  0.800     0.7174     0.6967
  0.900     0.7833     0.6216
  1.000     0.8415     0.5403
  1.100     0.8912     0.4536
  1.200     0.9320     0.3624
  1.300     0.9636     0.2675
  1.400     0.9854     0.1700
  1.500     0.9975     0.0707
Component  2 has a root at t =  1.5708
  1.600     0.9996    -0.0292
  1.700     0.9917    -0.1288
  1.800     0.9738    -0.2272
  1.900     0.9463    -0.3233
  2.000     0.9093    -0.4161
  2.100     0.8632    -0.5048
  2.200     0.8085    -0.5885
  2.300     0.7457    -0.6663
  2.400     0.6755    -0.7374
  2.500     0.5985    -0.8011
  2.600     0.5155    -0.8569
  2.700     0.4274    -0.9041
  2.800     0.3350    -0.9422
  2.900     0.2392    -0.9710
  3.000     0.1411    -0.9900
  3.100     0.0416    -0.9991
Component  1 has a root at t =  3.1416
  3.200    -0.0584    -0.9983
  3.300    -0.1577    -0.9875
  3.400    -0.2555    -0.9668
  3.500    -0.3508    -0.9365
```

```
  3.600    -0.4425    -0.8968
  3.700    -0.5298    -0.8481
  3.800    -0.6119    -0.7910
  3.900    -0.6878    -0.7259
  4.000    -0.7568    -0.6536
  4.100    -0.8183    -0.5748
  4.200    -0.8716    -0.4903
  4.300    -0.9162    -0.4008
  4.400    -0.9516    -0.3073
  4.500    -0.9775    -0.2108
  4.600    -0.9937    -0.1122
  4.700    -0.9999    -0.0124
Component  2 has a root at t =  4.7124
  4.800    -0.9962     0.0875
  4.900    -0.9825     0.1865
  5.000    -0.9589     0.2837
  5.100    -0.9258     0.3780
  5.200    -0.8835     0.4685
  5.300    -0.8323     0.5544
  5.400    -0.7728     0.6347
  5.500    -0.7055     0.7087
  5.600    -0.6313     0.7756
  5.700    -0.5507     0.8347
  5.800    -0.4646     0.8855
  5.900    -0.3739     0.9275
  6.000    -0.2794     0.9602
  6.100    -0.1822     0.9833
  6.200    -0.0831     0.9965
Component  1 has a root at t =  6.2832

 Cost of the integration in evaluations of F is   356
```