NAG Fortran Library Routine Document

D02EJF

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

D02EJF integrates a stiff system of first-order ordinary differential equations over an interval with suitable initial conditions, using a variable-order, variable-step method implementing the Backward Differentiation Formulae (BDF), until a user-specified function, if supplied, of the solution is zero, and returns the solution at points specified by the user, if desired.

2 Specification

```
      SUBROUTINE D02EJF(X, XEND, N, Y, FCN, PEDERV, TOL, RELABS, OUTPUT, G, W,
     1                  IW, IFAIL)
      INTEGER           N, IW, IFAIL
      real              X, XEND, Y(N), TOL, G, W(IW)
      CHARACTER*1       RELABS
      EXTERNAL          FCN, PEDERV, OUTPUT, G
```

3 Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \quad i = 1, 2, \ldots, n,$$

from $x = \mathrm{X}$ to $x = \mathrm{XEND}$ using a variable-order, variable-step method implementing the BDF. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_n$ (see Section 5). The initial values of $y_1, y_2, \ldots, y_n$ must be given at $x = \mathrm{X}$.

The solution is returned via the user-supplied routine OUTPUT at points specified by the user, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x, y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately by C^1 interpolation to the solution. It is assumed that $g(x, y)$ is a continuous function of the variables, so that a solution of $g(x, y) = 0.0$ can be determined by searching for a change in sign in $g(x, y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x, y) = 0.0$, is controlled by the parameters TOL and RELABS. The Jacobian of the system $y' = f(x, y)$ may be supplied in routine PEDERV, if it is available.

For a description of BDF and their practical implementation see Hall and Watt (1976).

4 References

Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

5 Parameters

1: X – ***real*** *Input/Output*

On entry: the initial value of the independent variable x.

Constraint: $\mathrm{X} \neq \mathrm{XEND}$.

On exit: if G is supplied by the user, X contains the point where $g(x,y) = 0.0$, unless $g(x,y) \neq 0.0$ anywhere on the range X to XEND, in which case, X will contain XEND. If G is not supplied X contains XEND, unless an error has occurred, when it contains the value of x at the error.

2: XEND – ***real*** *Input*

On entry: the final value of the independent variable. If XEND < X, integration will proceed in the negative direction.

Constraint: XEND $\neq$ X.

3: N – INTEGER *Input*

On entry: the number of differential equations, n.

Constraint: N $\geq$ 1.

4: Y(N) – ***real*** array *Input/Output*

On entry: the initial values of the solution $y_1, y_2, \ldots, y_n$ at $x =$ X.

On exit: the computed values of the solution at the final point $x =$ X.

5: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e., the derivatives y_i') for given values of its arguments $x, y_1, \ldots, y_n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real            X, Y(n), F(n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array *Input*

On entry: the value of the variable y_i, for $i = 1, 2, \ldots, n$.

3: F(n) – ***real*** array *Output*

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

FCN must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: PEDERV – SUBROUTINE, supplied by the user. *External Procedure*

PEDERV must evaluate the Jacobian of the system (that is, the partial derivatives $\frac{\partial f_i}{\partial y_j}$) for given values of the variables $x, y_1, y_2, \ldots, y_n$.

Its specification is:

```
SUBROUTINE PEDERV(X, Y, PW)
real               X, Y(n), PW(n,n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array *Input*

On entry: the value of the variable y_i, for $i = 1, 2, \ldots, n$.

3: PW(n,n) – ***real*** array *Output*

On exit: the value of $\frac{\partial f_i}{\partial y_j}$, for $i, j = 1, 2, \ldots, n$.

PEDERV must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

If the user does not wish to supply the Jacobian, the actual argument PEDERV **must** be the dummy routine D02EJY. (D02EJY is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation dependent: see the User's Note for your implementation for details.)

7: TOL – ***real*** *Input/Output*

On entry: TOL must be set to a **positive** tolerance for controlling the error in the integration. Hence TOL affects the determination of the position where $g(x, y) = 0.0$, if G is supplied.

D02EJF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution. However, the actual relation between TOL and the accuracy achieved cannot be guaranteed. The user is strongly recommended to call D02EJF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, the user might compare the results obtained by calling D02EJF with TOL $= 10^{-p}$ and TOL $= 10^{-p-1}$ if p correct decimal digits are required in the solution.

Constraint: TOL > 0.0.

On exit: normally unchanged. However if the range X to XEND is so short that a small change in TOL is unlikely to make any change in the computed solution, then, on return, TOL has its sign changed.

8: RELABS – CHARACTER*1 *Input*

On entry: the type of error control. At each step in the numerical solution an estimate of the local error, EST, is made. For the current step to be accepted the following condition must be satisfied:

$$\mathrm{EST} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i/(\tau_r \times |y_i| + \tau_a))^2} \leq 1.0$$

where τ_r and τ_a are defined by

RELABS	τ_r	τ_a
'M'	TOL	TOL
'A'	0.0	TOL
'R'	TOL	ϵ
'D'	TOL	ϵ

where ϵ is a small machine-dependent number and e_i is an estimate of the local error at y_i, computed internally. If the appropriate condition is not satisfied, the step size is reduced and the solution is recomputed on the current step. If the user wishes to measure the error in the computed solution in terms of the number of correct decimal places, then RELABS should be set to 'A' on entry, whereas if the error requirement is in terms of the number of correct significant digits, then RELABS should be set to 'R'. If the user prefers a mixed error test, then RELABS should be set to 'M', otherwise if the user has no preference, RELABS should be set to the default 'D'. Note that in this case 'D' is taken to be 'R'.

Constraint: RELABS = 'A', 'M', 'R' or 'D'.

9: OUTPUT – SUBROUTINE, supplied by the user. *External Procedure*

OUTPUT permits access to intermediate values of the computed solution (for example to print or plot them), at successive user-specified points. It is initially called by D02EJF with XSOL=X (the initial value of x). The user must reset XSOL to the next point (between the current XSOL and XEND) where OUTPUT is to be called, and so on at each call to OUTPUT. If, after a call to OUTPUT, the reset point XSOL is beyond XEND, D02EJF will integrate to XEND with no further calls to OUTPUT; if a call to OUTPUT is required at the point XSOL=XEND, then XSOL must be given precisely the value XEND.

Its specification is:

```
SUBROUTINE OUTPUT(XSOL, Y)
real               XSOL, Y(n)
```

where n is the actual value of N in the call of D02EJF.

1: XSOL – ***real*** *Input/Output*

On entry: the value of the independent variable x.

On exit: the user must set XSOL to the next value of x at which OUTPUT is to be called.

2: Y(n) – ***real*** array *Input*

On entry: the computed solution at the point XSOL.

OUTPUT must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

If the user does not wish to access intermediate output, the actual argument OUTPUT **must** be the dummy routine D02EJX. (D02EJX is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the the Users' Note for your implementation for details.)

10: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x, y)$ for specified values x, y. It specifies the function g for which the first position x where $g(x, y) = 0$ is to be found.

Its specification is:

```
real FUNCTION G(X, Y)
real            X, Y(n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array *Input*

On entry: the value of the variable y_i, for $i = 1, 2, \dots, n$.

G must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

If the user does not require the root finding option, the actual argument G **must** be the dummy routine D02EJW. (D02EJW is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the the Users' Note for your implementation for details.)

11: W(IW) – ***real*** array *Workspace*

12: IW – INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D02EJF is called.

Constraint: $IW \geq (12 + N) \times N + 50$.

13: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

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, for users 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.**

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, $TOL \leq 0.0$,
or $X = XEND$,
or $N \leq 0$,
or $RELABS \neq$ 'M', 'A', 'R', 'D',
or $IW < (12 + N) \times N + 50$.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point $x = X$. (See Section 5 for a discussion of this error test.) The components $Y(1), Y(2), \ldots, Y(n)$ contain the computed values of the solution at the current point $x = X$. If the user has supplied G, then no point at which $g(x, y)$ changes sign has been located up to the point $x = X$.

IFAIL = 3

TOL is too small for D02EJF to take an initial step. X and $Y(1), Y(2), \ldots, Y(n)$ retain their initial values.

IFAIL = 4

XSOL lies behind X in the direction of integration, after the initial call to OUTPUT, if the OUTPUT option was selected.

IFAIL = 5

A value of XSOL returned by OUTPUT lies behind the last value of XSOL in the direction of integration, if the OUTPUT option was selected.

IFAIL = 6

At no point in the range X to XEND did the function $g(x, y)$ change sign, if G was supplied. It is assumed that $g(x, y) = 0$ has no solution.

IFAIL = 7

A serious error has occurred in an internal call to C05AZF. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 8

A serious error has occurred in an internal call to D02XKF. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 9

A serious error has occurred in an internal call to an interpolation routine. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

The accuracy of the computation of the solution vector Y may be controlled by varying the local error tolerance TOL. In general, a decrease in local error tolerance should lead to an increase in accuracy. Users are advised to choose RELABS = 'R' unless they have a good reason for a different choice. It is particularly appropriate if the solution decays.

If the problem is a root-finding one, then the accuracy of the root determined will depend strongly on $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y_i}$, for $i = 1, 2, \ldots, n$. Large values for these quantities may imply large errors in the root.

8 Further Comments

If more than one root is required, then to determine the second and later roots D02EJF may be called again starting a short distance past the previously determined roots. Alternatively the user may construct his own root finding code using D02NBF (and other routines in Chapter D02M/N, D02XKF and C05AZF.

If it is easy to code, the user should supply the routine PEDERV. However, it is important to be aware that if PEDERV is coded incorrectly, a very inefficient integration may result and possibly even a failure to complete the integration (IFAIL = 2).

9 Example

We illustrate the solution of five different problems. In each case the differential system is the well-known stiff Robertson problem.

$$\begin{aligned} a' &= -0.04a - 10^4 bc \\ b' &= 0.04a - 10^4 bc - 3 \times 10^7 b^2 \\ c' &= 3 \times 10^7 b^2 \end{aligned}$$

with initial conditions $a = 1.0$, $b = c = 0.0$ at $x = 0.0$. We solve each of the following problems with local error tolerances 1.0E−3 and 1.0E−4.

(i) To integrate to $x = 10.0$ producing output at intervals of 2.0 until a point is encountered where $a = 0.9$. The Jacobian is calculated numerically.

(ii) As (i) but with the Jacobian calculated analytically.

(iii) As (i) but with no intermediate output.

(iv) As (i) but with no termination on a root-finding condition.

(v) Integrating the equations as in (i) but with no intermediate output and no root-finding termination condition.

9.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     D02EJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, IW
      PARAMETER        (N=3,IW=(12+N)*N+50)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             H, XEND
      INTEGER          K
*     .. Local Scalars ..
      real             TOL, X
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             W(IW), Y(N)
*     .. External Functions ..
      real             D02EJW, G
      EXTERNAL         D02EJW, G
*     .. External Subroutines ..
      EXTERNAL         D02EJF, D02EJX, D02EJY, FCN, OUT, PEDERV
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, K
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02EJF Example Program Results'
      XEND = 10.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1: calculating Jacobian internally,'
      WRITE (NOUT,*) ' intermediate output, root-finding'
      DO 20 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X          Y(1)          Y(2)          Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',OUT,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2: calculating Jacobian by PEDERV,'
      WRITE (NOUT,*) ' intermediate output, root-finding'
      DO 40 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X          Y(1)          Y(2)          Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,PEDERV,TOL,'Default',OUT,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
```

```
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 3: calculating Jacobian internally,'
      WRITE (NOUT,*) ' no intermediate output, root-finding'
      DO 60 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',D02EJX,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   60 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 4: calculating Jacobian internally,'
      WRITE (NOUT,*) ' intermediate output, no root-finding'
      DO 80 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X          Y(1)          Y(2)          Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',OUT,D02EJW,W,
     +               IW,IFAIL)
*
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   80 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 5: calculating Jacobian internally,'
      WRITE (NOUT,*)
     +  ' no intermediate output, no root-finding (integrate to XEND)'
      DO 100 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         WRITE (NOUT,*) '     X          Y(1)          Y(2)          Y(3)'
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',D02EJX,D02EJW,
     +               W,IW,IFAIL)
*
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
  100 CONTINUE
      STOP
*
```

```
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.3)
99997 FORMAT (1X,A,3F13.5)
99996 FORMAT (1X,F8.2,3F13.5)
      END
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=3)
*     .. Scalar Arguments ..
      real            T
*     .. Array Arguments ..
      real            F(N), Y(N)
*     .. Executable Statements ..
      F(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
      F(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
      F(3) = 3.0e7*Y(2)*Y(2)
      RETURN
      END
*
      SUBROUTINE PEDERV(X,Y,PW)
*     .. Parameters ..
      INTEGER            N
      PARAMETER          (N=3)
*     .. Scalar Arguments ..
      real               X
*     .. Array Arguments ..
      real               PW(N,N), Y(N)
*     .. Executable Statements ..
      PW(1,1) = -0.04e0
      PW(1,2) = 1.0e4*Y(3)
      PW(1,3) = 1.0e4*Y(2)
      PW(2,1) = 0.04e0
      PW(2,2) = -1.0e4*Y(3) - 6.0e7*Y(2)
      PW(2,3) = -1.0e4*Y(2)
      PW(3,1) = 0.0e0
      PW(3,2) = 6.0e7*Y(2)
      PW(3,3) = 0.0e0
      RETURN
      END
*
      real FUNCTION G(T,Y)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=3)
*     .. Scalar Arguments ..
      real             T
*     .. Array Arguments ..
      real             Y(N)
*     .. Executable Statements ..
      G = Y(1) - 0.9e0
      RETURN
      END
*
      SUBROUTINE OUT(X,Y)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=3)
      INTEGER         NOUT
      PARAMETER       (NOUT=6)
*     .. Scalar Arguments ..
      real            X
*     .. Array Arguments ..
      real            Y(N)
*     .. Scalars in Common ..
      real            H, XEND
      INTEGER         I
*     .. Local Scalars ..
      INTEGER         J
*     .. Intrinsic Functions ..
```

```
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, I
*     .. Executable Statements ..
      WRITE (NOUT,99999) X, (Y(J),J=1,N)
      X = XEND - real(I)*H
      I = I - 1
      RETURN
*
99999 FORMAT (1X,F8.2,3F13.5)
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02EJF Example Program Results

 Case 1: calculating Jacobian internally,
  intermediate output, root-finding

  Calculation with TOL = 0.1E-02
      X         Y(1)         Y(2)         Y(3)
     0.00      1.00000      0.00000      0.00000
     2.00      0.94163      0.00003      0.05834
     4.00      0.90551      0.00002      0.09447
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998

  Calculation with TOL = 0.1E-03
      X         Y(1)         Y(2)         Y(3)
     0.00      1.00000      0.00000      0.00000
     2.00      0.94161      0.00003      0.05837
     4.00      0.90551      0.00002      0.09446
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998


 Case 2: calculating Jacobian by PEDERV,
  intermediate output, root-finding

  Calculation with TOL = 0.1E-02
      X         Y(1)         Y(2)         Y(3)
     0.00      1.00000      0.00000      0.00000
     2.00      0.94163      0.00003      0.05834
     4.00      0.90551      0.00002      0.09447
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998

  Calculation with TOL = 0.1E-03
      X         Y(1)         Y(2)         Y(3)
     0.00      1.00000      0.00000      0.00000
     2.00      0.94161      0.00003      0.05837
     4.00      0.90551      0.00002      0.09446
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998


 Case 3: calculating Jacobian internally,
  no intermediate output, root-finding

  Calculation with TOL = 0.1E-02
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998

  Calculation with TOL = 0.1E-03
   Root of Y(1)-0.9 at  4.377
   Solution is      0.90000      0.00002      0.09998
```

```
Case 4: calculating Jacobian internally,
 intermediate output, no root-finding

 Calculation with TOL = 0.1E-02
     X         Y(1)         Y(2)         Y(3)
    0.00      1.00000      0.00000      0.00000
    2.00      0.94163      0.00003      0.05834
    4.00      0.90551      0.00002      0.09447
    6.00      0.87930      0.00002      0.12068
    8.00      0.85858      0.00002      0.14140
   10.00      0.84136      0.00002      0.15862

 Calculation with TOL = 0.1E-03
     X         Y(1)         Y(2)         Y(3)
    0.00      1.00000      0.00000      0.00000
    2.00      0.94161      0.00003      0.05837
    4.00      0.90551      0.00002      0.09446
    6.00      0.87926      0.00002      0.12072
    8.00      0.85854      0.00002      0.14145
   10.00      0.84136      0.00002      0.15863


Case 5: calculating Jacobian internally,
 no intermediate output, no root-finding (integrate to XEND)

 Calculation with TOL = 0.1E-02
     X         Y(1)         Y(2)         Y(3)
    0.00      1.00000      0.00000      0.00000
   10.00      0.84136      0.00002      0.15862

 Calculation with TOL = 0.1E-03
     X         Y(1)         Y(2)         Y(3)
    0.00      1.00000      0.00000      0.00000
   10.00      0.84136      0.00002      0.15863
```