# NAG 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 you, if desired.

## 2 Specification

```
SUBROUTINE D02EJF (X, XEND, N, Y, FCN, PEDERV, TOL, RELABS, OUTPUT, G,     &
                   W, IW, IFAIL)

INTEGER             N, IW, IFAIL
REAL (KIND=nag_wp) X, XEND, Y(N), TOL, G, W(IW)
CHARACTER(1)        RELABS
EXTERNAL            FCN, PEDERV, OUTPUT, G
```

## 3 Description

D02EJF 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 FCN, 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 OUTPUT at points specified by you, 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 arguments TOL and RELABS. The Jacobian of the system $y' = f(x, y)$ may be supplied in 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 Arguments

1: X – REAL (KIND=nag_wp) *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 you, 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 (KIND=nag_wp) *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*: $n$, the number of differential equations.

*Constraint*: N $\geq 1$.

4: Y(N) – REAL (KIND=nag_wp) 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 $x, y_1, \ldots, y_n$.

The specification of FCN is:

```
SUBROUTINE FCN (X, Y, F)
REAL (KIND=nag_wp) X, Y(n), F(n)
```

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

1: X – REAL (KIND=nag_wp) *Input*

*On entry*: $x$, the value of the independent variable.

2: Y($n$) – REAL (KIND=nag_wp) array *Input*

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

3: F($n$) – REAL (KIND=nag_wp) array *Output*

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

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02EJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

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

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

The specification of PEDERV is:

```
SUBROUTINE PEDERV (X, Y, PW)
REAL (KIND=nag_wp) X, Y(n), PW(*)
```

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

1: X – REAL (KIND=nag_wp) *Input*

*On entry*: $x$, the value of the independent variable.

2: Y($n$) – REAL (KIND=nag_wp) array *Input*

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

3: PW($*$) – REAL (KIND=nag_wp) array *Output*

*On exit*: PW($n \times (i-1) + j$) must contain the value of $\dfrac{\partial f_i}{\partial y_j}$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

PEDERV must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02EJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

If you do not wish to supply the Jacobian, the actual argument PEDERV **must** be the dummy routine D02EJY. (D02EJY is included in the NAG Library.)

7: TOL – REAL (KIND=nag_wp) *Input/Output*

*On entry*: 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. You are 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, you 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:

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

where $\tau_r$ and $\tau_a$ are defined by

| RELABS | $\tau_r$ | $\tau_a$ |
|---|---|---|
| `M' | TOL | TOL |
| `A' | 0.0 | TOL |
| `R' | TOL | $\epsilon$ |
| `D' | TOL | $\epsilon$ |

where $\epsilon$ 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 you wish 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 you prefer a mixed error test, then RELABS should be set to

`M', otherwise if you have 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 NAG Library or 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 $x$). You 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.

The specification of OUTPUT is:

```
SUBROUTINE OUTPUT (XSOL, Y)
REAL (KIND=nag_wp) XSOL, Y(n)
```

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

1: XSOL – REAL (KIND=nag_wp) *Input/Output*

*On entry*: $x$, the value of the independent variable.

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

2: Y($n$) – REAL (KIND=nag_wp) array *Input*

*On entry*: the computed solution at the point XSOL.

OUTPUT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02EJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

If you do not wish to access intermediate output, the actual argument OUTPUT **must** be the dummy routine D02EJX. (D02EJX is included in the NAG Library.)

10: G – REAL (KIND=nag_wp) 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.

The specification of G is:

```
FUNCTION G (X, Y)
REAL (KIND=nag_wp) G
REAL (KIND=nag_wp) X, Y(n)
```

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

1: X – REAL (KIND=nag_wp) *Input*

*On entry*: $x$, the value of the independent variable.

2: Y($n$) – REAL (KIND=nag_wp) array *Input*

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

G must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02EJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

If you do not require the root-finding option, the actual argument G **must** be the dummy routine D02EJW. (D02EJW is included in the NAG Library.)

11: W(IW) – REAL (KIND=nag_wp) 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 $\geq (12 + \mathrm{N}) \times \mathrm{N} + 50$.

13: 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, if you are not familiar with this argument, 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 $= 1$

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

IFAIL $= 2$

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

IFAIL $= 3$

TOL is too small for D02EJF to take an initial step. X and $\mathrm{Y}(1), \mathrm{Y}(2), \ldots, \mathrm{Y}(\mathrm{N})$ 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 the 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 $= 7$ (C05AZF)

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

IFAIL $= 8$ (D02XKF)

A serious error has occurred in an internal call to the specified routine. 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 (sub) program calls and array dimensions. Seek expert help.

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 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. You are advised to choose RELABS = 'R' unless you 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 $\dfrac{\partial g}{\partial x}$ and $\dfrac{\partial g}{\partial y_i}$, for $i = 1, 2, \ldots, n$. Large values for these quantities may imply large errors in the root.

## 8 Parallelism and Performance

D02EJF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D02EJF 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 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 you may construct your own root-finding code using D02NBF (and other routines in Sub-chapter D02M–N), C05AZF and D02XKF.

If it is easy to code, you should supply 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 (see IFAIL = 2).

## 10 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.

### 10.1 Program Text

```
!   D02EJF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02ejfe_mod

!     Data for D02EJF example program

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                            :: fcn, g, output, pederv
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter     :: alpha = 0.04_nag_wp
      Real (Kind=nag_wp), Parameter     :: beta = 1.0E4_nag_wp
      Real (Kind=nag_wp), Parameter     :: gamma = 3.0E7_nag_wp
      Real (Kind=nag_wp), Parameter     :: zero = 0.0_nag_wp
      Integer, Parameter, Public        :: n = 3, nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save :: h, xend
    Contains
      Subroutine fcn(x,y,f)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: f(*)
        Real (Kind=nag_wp), Intent (In) :: y(*)
```

```
!       .. Executable Statements ..
        f(1) = -alpha*y(1) + beta*y(2)*y(3)
        f(2) = alpha*y(1) - beta*y(2)*y(3) - gamma*y(2)*y(2)
        f(3) = gamma*y(2)*y(2)
        Return
      End Subroutine fcn
      Subroutine pederv(x,y,pw)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: pw(*)
        Real (Kind=nag_wp), Intent (In) :: y(*)
!       .. Executable Statements ..
        pw(1) = -alpha
        pw(2) = alpha
        pw(3) = zero
        pw(4) = beta*y(3)
        pw(5) = -beta*y(3) - 2.0_nag_wp*gamma*y(2)
        pw(6) = 2.0_nag_wp*gamma*y(2)
        pw(7) = beta*y(2)
        pw(8) = -beta*y(2)
        pw(9) = zero
        Return
      End Subroutine pederv
      Function g(x,y)

!       .. Function Return Value ..
        Real (Kind=nag_wp)             :: g
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: y(*)
!       .. Executable Statements ..
        g = y(1) - 0.9E0_nag_wp
        Return
      End Function g
      Subroutine output(xsol,y)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: xsol
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: y(*)
!       .. Local Scalars ..
        Integer                        :: j
!       .. Intrinsic Procedures ..
        Intrinsic                      :: abs
!       .. Executable Statements ..
        Write (nout,99999) xsol, (y(j),j=1,n)
        xsol = xsol + h
!       Make sure we exactly hit xsol = xend
        If (abs(xsol-xend)<h/4.0E0_nag_wp) Then
          xsol = xend
        End If
        Return

99999   Format (1X,F8.2,3F13.5)
      End Subroutine output
    End Module d02ejfe_mod
    Program d02ejfe

!     D02EJF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02ejf, d02ejw, d02ejx, d02ejy, nag_wp
      Use d02ejfe_mod, Only: fcn, g, h, n, nin, nout, output, pederv, xend
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)             :: tol, x, xinit
      Integer                        :: i, icase, ifail, iw, j, kinit
```

```
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: w(:), y(:), yinit(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: real
!     .. Executable Statements ..
      Write (nout,*) 'D02EJF Example Program Results'
      iw = (12+n)*n + 50
      Allocate (w(iw),y(n),yinit(n))
!     Skip heading in data file
      Read (nin,*)
!     xinit: initial x value, xend: final x value
!     y: initial solution values
      Read (nin,*) xinit, xend
      Read (nin,*) yinit(1:n)
      Read (nin,*) kinit
      Do icase = 1, 5
        If (icase/=2) Then
          Write (nout,99995) icase, 'Jacobian internally'
        Else
          Write (nout,99995) icase, 'Jacobian by PEDERV'
        End If
        Select Case (icase)
        Case (1,2)
          Write (nout,99994) 'intermediate output, root-finding'
        Case (3)
          Write (nout,99994) 'no intermediate output, root-finding'
        Case (4)
          Write (nout,99994) 'intermediate output, no root-finding'
        Case (5)
          Write (nout,99994)                                                    &
            'no intermediate output, no root-finding (integrate to XEND)'
        End Select
        Do j = 3, 4
          tol = 10.0E0_nag_wp**(-j)
          Write (nout,99999) ' Calculation with TOL =', tol
          x = xinit
          y(1:n) = yinit(1:n)
          If (icase/=3) Then
            Write (nout,*) '     X         Y(1)         Y(2)         Y(3)'
            h = (xend-x)/real(kinit+1,kind=nag_wp)
          End If
          ifail = 0
          Select Case (icase)
          Case (1)
            Call d02ejf(x,xend,n,y,fcn,d02ejy,tol,'Default',output,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)
          Case (2)
            Call d02ejf(x,xend,n,y,fcn,pederv,tol,'Default',output,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)
          Case (3)
            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)
          Case (4)
            ifail = 0
            Call d02ejf(x,xend,n,y,fcn,d02ejy,tol,'Default',output,d02ejw,w,   &
              iw,ifail)
          Case (5)
            Write (nout,99996) x, (y(i),i=1,n)
            Call d02ejf(x,xend,n,y,fcn,d02ejy,tol,'Default',d02ejx,d02ejw,w,   &
              iw,ifail)
            Write (nout,99996) x, (y(i),i=1,n)
          End Select
          If (tol<0.0E0_nag_wp) Then
            Write (nout,*) '  Range too short for TOL'
          End If
```

```
        End Do
        If (icase<5) Then
          Write (nout,*)
        End If
      End Do

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)
99995 Format (/,1X,'Case ',I1,': calculating ',A,',')
99994 Format (8X,A)
    End Program d02ejfe
```

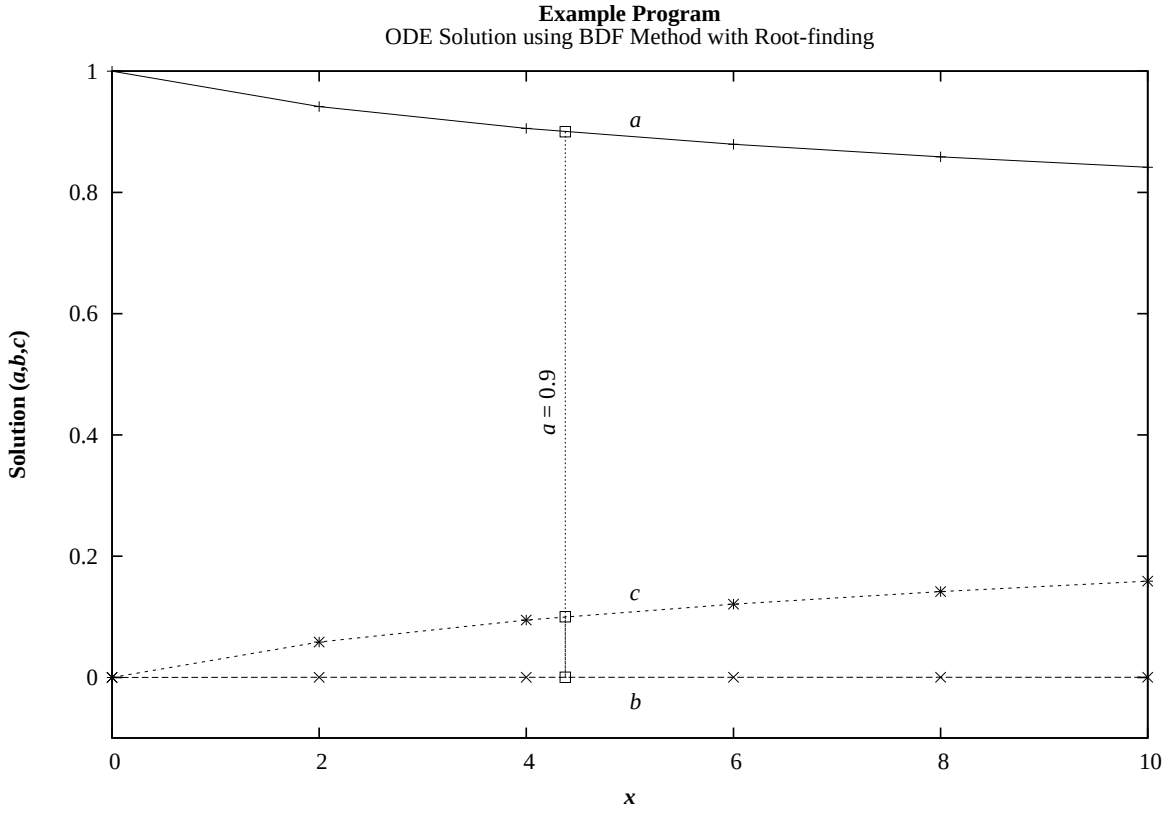
## 10.2 Program Data

```
D02EJF Example Program Data
   0.0 10.0             : xinit, xend
   1.0, 0.0, 0.0        : yinit
   4                    : kinit
```

## 10.3 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
```


Example Program
ODE Solution using BDF Method with Root-finding
Solution (a,b,c)
x
a
b
c
a = 0.9
0
0.2
0.4
0.6
0.8
1
0
2
4
6
8
10