NAG Library Routine Document

D02CJF

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

D02CJF integrates a system of first-order ordinary differential equations over a range with suitable initial conditions, using a variable-order, variable-step Adams' method 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 D02CJF (X, XEND, N, Y, FCN, TOL, RELABS, OUTPUT, G, W, IFAIL)
INTEGER             N, IFAIL
REAL (KIND=nag_wp) X, XEND, Y(N), TOL, G, W(28+21*N)
CHARACTER(1)        RELABS
EXTERNAL            FCN, OUTPUT, G
```

3 Description

D02CJF 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 $x =$ X to $x =$ XEND using a variable-order, variable-step Adams' method. The system is defined by FCN, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_n$. The initial values of $y_1, y_2, \ldots, y_n$ must be given at $x =$ X.

The solution is returned via 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.

For a description of Adams' methods 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: X $\neq$ XEND.

On exit: if g is supplied by you, it 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 by you it 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 ≥ 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(*), F(*)
```

1: X – REAL (KIND=nag_wp) *Input*

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

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

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

3: F($*$) – 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 D02CJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

6: TOL – REAL (KIND=nag_wp) *Input*

On entry: 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.

D02CJF 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 D02CJF 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 D02CJF with TOL $= 10.0^{-p}$ and TOL $= 10.0^{-p-1}$ where p correct decimal digits are required in the solution.

Constraint: TOL > 0.0.

7: 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:

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

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	TOL

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 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 `M'.

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

8: 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 D02CJF 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, D02CJF 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(*)
```

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

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

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

2: Y$(*)$ – 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 D02CJF 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 D02CJX. (D02CJX is included in the NAG Library.)

9: 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(*)
```

1: X – REAL (KIND=nag_wp) *Input*

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

2: Y$(*)$ – 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 D02CJF 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 D02CJW. (D02CJW is included in the NAG Library.)

10: W$(28 + 21 \times \mathrm{N})$ – REAL (KIND=nag_wp) array *Workspace*

11: 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 ≤ 0.0,
or N ≤ 0,
or RELABS $\neq$ 'M', 'A', 'R' or 'D',
or X $=$ XEND.

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 9 for a discussion of this error exit.) The components Y(1), Y(2), $\ldots$, Y(N) contain the computed values of the solution at the current point $x =$ X. If you have 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 D02CJF to take an initial step. X and Y(1), Y(2), $\ldots$, Y(N) retain their initial values.

IFAIL $= 4$

XSOL has not been reset or 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 has not been reset or 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. Check all subroutine calls and array sizes. 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 = 'M' unless you have a good reason for a different choice.

If the problem is a root-finding one, then the accuracy of the root determined will depend on the properties of $g(x, y)$. You should try to code G without introducing any unnecessary cancellation errors.

8 Parallelism and Performance

D02CJF is not threaded in any implementation.

9 Further Comments

If more than one root is required then D02QFF should be used.

If D02CJF fails with IFAIL $= 3$, then it can be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If D02CJF fails with IFAIL $= 2$, it is probable that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. You should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL $= 2$, unless overflow occurs first. Numerical integration cannot be continued through a singularity, and analytic treatment should be considered;

(b) for 'stiff' equations where the solution contains rapidly decaying components, the routine will use very small steps in x (internally to D02CJF) to preserve stability. This will exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL $= 2$. Adams' methods are not efficient in such cases, and you should try D02EJF.

10 Example

This example illustrates the solution of four different problems. In each case the differential system (for a projectile) is

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi} \\ \phi' &= \frac{-0.032}{v^2} \end{aligned}$$

over an interval X $= 0.0$ to XEND $= 10.0$ starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We solve each of the following problems with local error tolerances 1.0E$-$4 and 1.0E$-$5.

(i) To integrate to $x = 10.0$ producing output at intervals of 2.0 until a point is encountered where $y = 0.0$.

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

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

(iv) As (i) but with no intermediate output and no root-finding termination condition.

10.1 Program Text

```
!   D02CJF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02cjfe_mod

!     Data for D02CJF example program

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                            :: fcn, g, output
!     .. Parameters ..
      Integer, Parameter, Public        :: n = 3, nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save :: h, xend
!     n: number of differential equations
    Contains
      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
      Subroutine fcn(x,y,f)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: alpha = -0.032E0_nag_wp
        Real (Kind=nag_wp), Parameter  :: beta = -0.02E0_nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: f(*)
        Real (Kind=nag_wp), Intent (In) :: y(*)
!       .. Intrinsic Procedures ..
        Intrinsic                       :: cos, tan
!       .. Executable Statements ..
        f(1) = tan(y(3))
        f(2) = alpha*tan(y(3))/y(2) + beta*y(2)/cos(y(3))
        f(3) = alpha/y(2)**2
        Return
      End Subroutine fcn
      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)
        Return
      End Function g
    End Module d02cjfe_mod
    Program d02cjfe

!     D02CJF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02cjf, d02cjw, d02cjx, nag_wp
      Use d02cjfe_mod, Only: fcn, g, h, n, nin, nout, output, 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,*) 'D02CJF Example Program Results'
      iw = 21*n + 28
      Allocate (w(iw),y(n),yinit(n))
!     Skip heading in data file
      Read (nin,*)
!     xinit: initial x value, xend: final x value.
      Read (nin,*) xinit
      Read (nin,*) xend
      Read (nin,*) yinit(1:n)
      Read (nin,*) kinit
      Do icase = 1, 4
        Write (nout,*)
        Select Case (icase)
        Case (1)
          Write (nout,99995) icase, 'intermediate output, root-finding'
```

```
        Case (2)
          Write (nout,99995) icase, 'no intermediate output, root-finding'
        Case (3)
          Write (nout,99995) icase, 'intermediate output, no root-finding'
        Case (4)
          Write (nout,99995) icase,                                          &
            'no intermediate output, no root-finding ( integrate to XEND)'
        End Select
        Do j = 4, 5
          tol = 10.0E0_nag_wp**(-j)
          Write (nout,*)
          Write (nout,99999) ' Calculation with TOL =', tol
          x = xinit
          y(1:n) = yinit(1:n)
          If (icase/=2) Then
            Write (nout,*) '     X         Y(1)         Y(2)         Y(3)'
            h = (xend-x)/real(kinit+1,kind=nag_wp)
          End If
!         ifail: behaviour on error exit
!                =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
          ifail = 0
          Select Case (icase)
          Case (1)
            Call d02cjf(x,xend,n,y,fcn,tol,'Default',output,g,w,ifail)
            Write (nout,99998) '  Root of Y(1) = 0.0 at', x
            Write (nout,99997) '  Solution is', (y(i),i=1,n)
          Case (2)
            Call d02cjf(x,xend,n,y,fcn,tol,'Default',d02cjx,g,w,ifail)
            Write (nout,99998) '  Root of Y(1) = 0.0 at', x
            Write (nout,99997) '  Solution is', (y(i),i=1,n)
          Case (3)
            Call d02cjf(x,xend,n,y,fcn,tol,'Default',output,d02cjw,w,ifail)
          Case (4)
            Write (nout,99996) x, (y(i),i=1,n)
            Call d02cjf(x,xend,n,y,fcn,tol,'Default',d02cjx,d02cjw,w,ifail)
            Write (nout,99996) x, (y(i),i=1,n)
          End Select
        End Do
        If (icase<4) 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,': ',A)
    End Program d02cjfe
```

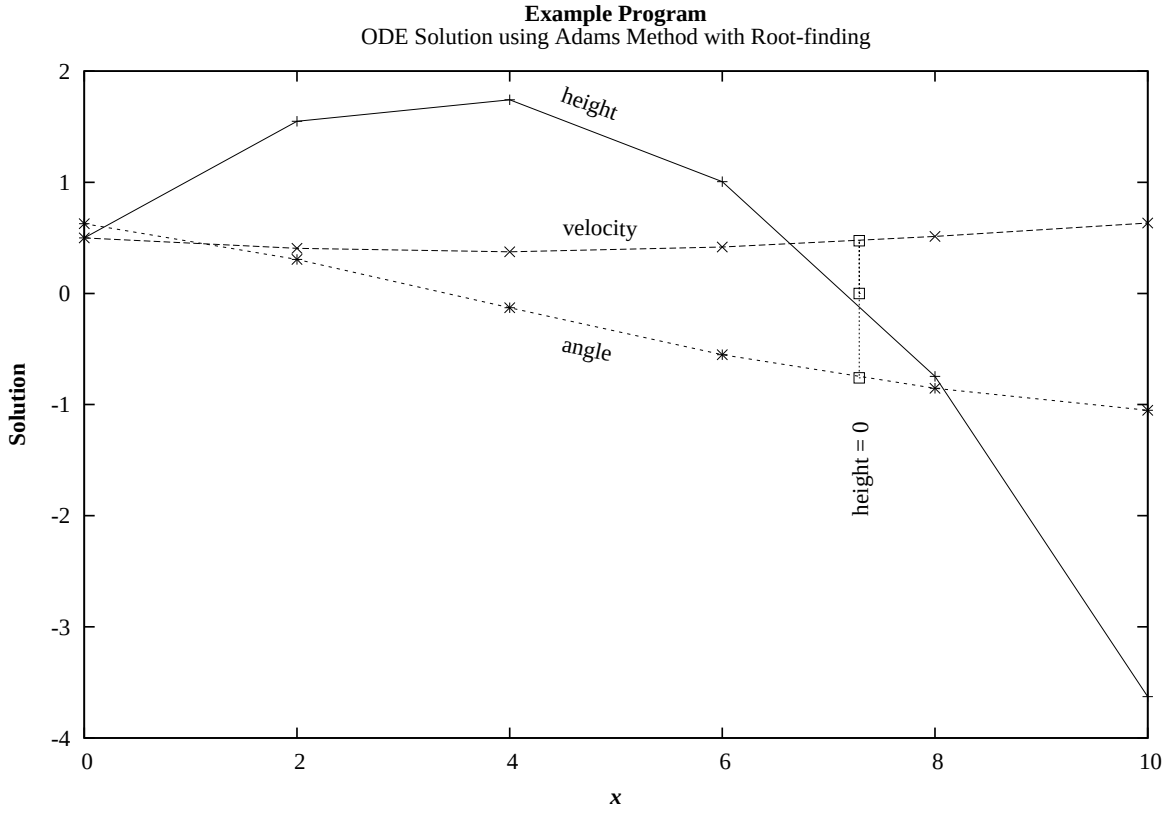
10.2 Program Data

```
D02CJF Example Program Data
   0.0                                    : xinit
  10.0                                    : xend
   0.5  0.5  6.28318530717958647692E-1    : yinit
   4                                      : kinit
```

10.3 Program Results

```
 D02CJF Example Program Results

 Case 1: intermediate output, root-finding

  Calculation with TOL = 0.1E-03
      X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54931      0.40548      0.30662
    4.00      1.74229      0.37433     -0.12890
    6.00      1.00554      0.41731     -0.55068
```

```
  Root of Y(1) = 0.0 at  7.288
  Solution is     -0.00000      0.47486     -0.76011

 Calculation with TOL = 0.1E-04
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54933      0.40548      0.30662
    4.00      1.74232      0.37433     -0.12891
    6.00      1.00552      0.41731     -0.55069
  Root of Y(1) = 0.0 at  7.288
  Solution is     -0.00000      0.47486     -0.76010


Case 2: no intermediate output, root-finding

 Calculation with TOL = 0.1E-03
  Root of Y(1) = 0.0 at  7.288
  Solution is     -0.00000      0.47486     -0.76011

 Calculation with TOL = 0.1E-04
  Root of Y(1) = 0.0 at  7.288
  Solution is     -0.00000      0.47486     -0.76010


Case 3: intermediate output, no root-finding

 Calculation with TOL = 0.1E-03
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54931      0.40548      0.30662
    4.00      1.74229      0.37433     -0.12890
    6.00      1.00554      0.41731     -0.55068
    8.00     -0.74589      0.51299     -0.85371
   10.00     -3.62813      0.63325     -1.05152

 Calculation with TOL = 0.1E-04
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54933      0.40548      0.30662
    4.00      1.74232      0.37433     -0.12891
    6.00      1.00552      0.41731     -0.55069
    8.00     -0.74601      0.51299     -0.85372
   10.00     -3.62829      0.63326     -1.05153


Case 4: no intermediate output, no root-finding ( integrate to XEND)

 Calculation with TOL = 0.1E-03
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
   10.00     -3.62813      0.63325     -1.05152

 Calculation with TOL = 0.1E-04
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
   10.00     -3.62829      0.63326     -1.05153
```


Example Program
ODE Solution using Adams Method with Root-finding
height
velocity
angle
height = 0
Solution
x
2
1
0
-1
-2
-3
-4
0
2
4
6
8
10