```
c
c     Main Program to Solve System of IVP's by the following methods
c     1. 2nd-order Adams-Bashford Method.
c     2. 2nd-order Predictor-Corrector Method.
c     3. 2nd-order Adams-Moulten Method.
c
      Program IVP
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (N=2)
      DOUBLE PRECISION Y(N),YOLD(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
      OPEN(11,FILE='IVP.txt')
c
      h     = 1.d-1
      x1    = 0.d0
      x2    = 3.d0
      tol   = 1.d-12
      itmax = 10
      phi1  = 1.d0
      phi2  = 5.d-1
c
c     provide initial conditions
      YOLD(1) = 1.d0
      YOLD(2) = 0.d0
c
c     Integrate IVP's using 2nd-order Adams-Bashford Method
c      call Adams_Bashford(N,YOLD,Y)
c
c     Integrate IVP's using 2nd-order Adams-Bashford Method
c      call Predictor_Corrector(N,YOLD,Y)
c
c
c     Integrate IVP's using 2nd-order Adams-Moulten Method
      call Adams_Moulten(N,YOLD,Y)
c
      stop
      end
c
c     END of main Program
c
c
c
c
c
c
c
c
c
c     Subroutine Func to Input RHS functions of IVP
      subroutine Func(N,Y,FUN)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION Y(N),FUN(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
c
      do i = 1, N
        FUN(i) = 0.d0
      enddo
c
      FUN(1) =-phi1*Y(1)
      FUN(2) = phi1*Y(1)-phi2*Y(2)*Y(2)
c
      return
      end
c
c
c
c
c
c
c
c
c
c     Subroutine Adams_Bashford to integrate IVP's
      subroutine Adams_Bashford(N,YOLD,Y)
```

```
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION Y(N),YOLD(N),FUNOLD1(N),FUNOLD2(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
c
      istep = 0
      xval  = x1
c
c     Print the Initial solution to ivp.txt
      write(11,*)
      write(11,'(12x,a,13x,a,13x,a)')  'X','Y(1)','Y(2)'
      write(11,'(3f16.6)') xval,YOLD(1),YOLD(2)
c
c     Evaluate the RHS Functions of IVP's for the Initial Solution
      call Func(N,YOLD,FUNOLD1)
c
c     Start Integration Loop
  10  xval  = xval + h
      istep = istep + 1
c
c     Set Termination Criteria for the Integration Loop
      if((xval-x2) .gt. 1.d-14) return
c
c     For the first time step apply 1st-order explicit Euler
      if (istep .eq. 1) then
        do i = 1, N
          Y(i) = YOLD(i)+h*FUNOLD1(i)
        enddo
      else
c
c     Apply 2nd-order AB Formula for Remaining Time Steps
        do i = 1, N
          Y(i) = YOLD(i)+h/2.d0*(3.d0*FUNOLD1(i)-FUNOLD2(i))
        enddo
      endif
c
c     Print the solution for the current time step to shooting.txt
      write(11,'(3f16.6)') xval,Y(1),Y(2)
c
c     Store Previous Time Values
      do i = 1, N
        YOLD(i)    = Y(i)
        FUNOLD2(i) = FUNOLD1(i)
      enddo
c
c     Evaluate the RHS Function at Current Time
      call Func(N,YOLD,FUNOLD1)
      go to 10
c
      end
c
c
c
c
c
c
c
c
c     Subroutine Predictor-Corrector to integrate IVP's
      subroutine Predictor_Corrector(N,YOLD,Y)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION Y(N),YOLD(N),FUNOLD1(N),FUNOLD2(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
c
      istep = 0
      xval  = x1
c
c     Print the Initial solution to ivp.txt
      write(11,*)
      write(11,'(12x,a,13x,a,13x,a)')  'X','Y(1)','Y(2)'
      write(11,'(3f16.6)') xval,YOLD(1),YOLD(2)
c
c     Evaluate the RHS Function for the Initial Solution
```

```
      call Func(N,YOLD,FUNOLD1)
c
c     Start Integration Loop
  10  xval  = xval + h
      istep = istep + 1
c
c     Set Termination Criteria for the Integration Loop
      if((xval-x2) .gt. 1.d-14) return
c
c     For the first time step apply 1st-order explicit Euler for predictor
c     and 1st-order implicit Euler for corrector
      if (istep .eq. 1) then
c       predicor
        do i = 1, N
          Y(i) = YOLD(i)+h*FUNOLD1(i)
        enddo
c       evaluate RHS function at predicted variable
        call Func(N,Y,FUNOLD1)
c       corrector
        do i = 1, N
          Y(i) = YOLD(i)+h*FUNOLD1(i)
        enddo
      else
c
c     Apply 2nd-order predictor-corrector method for Remaining Time Steps
c       predicor
        do i = 1, N
          Y(i) = YOLD(i)+h/2.d0*(3.d0*FUNOLD1(i)-FUNOLD2(i))
        enddo
c       evaluate RHS function at predicted variable
        call Func(N,Y,FUNOLD1)
c       corrector
        do i = 1, N
          Y(i) = YOLD(i)+h/2.d0*(FUNOLD1(i)+FUNOLD2(i))
        enddo
      endif
c
c     Print the solution for the current time step to ivp.txt
      write(11,'(3f16.6)') xval,Y(1),Y(2)
c
c     Store Previous Time Values
      do i = 1, N
        YOLD(i)    = Y(i)
        FUNOLD2(i) = FUNOLD1(i)
      enddo
c
c     Evaluate the RHS Function at Current Time
      call Func(N,YOLD,FUNOLD1)
      go to 10
c
      end
c
c
c
c
c
c
c
c
c     Subroutine Adams_Moulten to integrate IVP's
      subroutine Adams_Moulten(N,YOLD,Y)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION Y(N),Y_G(N),FUN(N),FUNOLD(N),YOLD(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
c
      istep = 0
      xval  = x1
c
c     Print the Initial solution to ivp.txt
      write(11,*)
      write(11,'(12x,a,13x,a,13x,a)')  'X','Y(1)','Y(2)'
      write(11,'(3f16.6)') xval,YOLD(1),YOLD(2)
```

```
c
c     Evaluate the RHS Function for the Initial Solution
      call Func(N,YOLD,FUNOLD)
c
c     Start Integration Loop
  10  xval  = xval + h
      istep = istep + 1
      write(*,'(4x,a,f10.4)') 'xval =',xval
c
c     Set Termination Criteria for the Integration Loop
      if((xval-x2) .gt. 1.d-14) return
c
c     Initial guess for current intehration step is the solution for previous step + tolerance
      do i = 1, N
        Y_G(i) = YOLD(i)+tol
      enddo
      call Newton(N,YOLD,Y_G,Y)
c
c     Evaluate the RHS Function for current integration step
      call Func(N,Y,FUN)
c
c     Store previous integration values
      do i = 1, N
        YOLD(i)   = Y(i)
        FUNOLD(i) = FUN(i)
      enddo
c
c     Print the solution for the current time step to ivp.txt
      write(11,'(3f16.6)') xval,Y(1),Y(2)
c
      go to 10
c
      return
      end
c
c
c
c
c
c
c
c
c     Subroutine Newton to to solve a system of algebraic equations iteratively
      subroutine Newton(N,YOLD,UNKNON_G,UNKNON)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION A(N,N),UNKNON_G(N),UNKNON(N),B(N)
      DOUBLE PRECISION YOLD(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
      integer ipvt(N)
c
c     initialize matrix A and vector B
      do i = 1, N
      UNKNON(i) = UNKNON_G(i)
      B(i) = 0.d0
      do j = 1, N
         A(i,j)=0.d0
      enddo
      enddo
c
c     Start Iterative Solution
      iter = 0
  10  iter = iter + 1
      if (iter .gt. itmax) go to 12
c
c     Input system of nonlinear algebraic equations
c     on return B contains the residuals
      call residuals(N,YOLD,UNKNON,B)
c
c     Check the norm of the residuals
      rnorm = 0.d0
      do i = 1, N
      rnorm = rnorm + B(i)**2
```

```
      enddo
      rnorm = dsqrt(rnorm)
      write(*,'(3x,a,i3,5x,a,e17.10)')
     !                   'iteration #',iter, 'Tolerance =', rnorm
      if (rnorm .lt. tol) return
c
c     Evaluate the Jacobian Matrix
c     IFLAG = 0 ==> analatical jacobian
c     IFLAG = 1 ==> numerical jacobian
      IFLAG = 1
      call jacobian(IFLAG,N,UNKNON,A)
c
c     LU-decompose matrix A
c     Note that matrix A will contains on return the L and U matrixes (original is destroyed)
      call dgefa(A,N,N,ipvt,info)
c
c     Solve the equations: A .(Xiter+1-Xiter) =-B, by backsubstitution
c     Note that vector B will contains the unknowns X (original is destroyed)
c     for Newton Raphson method B = Xiter+1 - Xiter
      do i = 1, N
        B(i) = -B(i)
      enddo
c
      call dgesl(A,N,N,ipvt,B,info)
c
      do i= 1, N
        UNKNON(i) = UNKNON(i)+B(i)
      enddo
      go to 10
  12  write(11,*) 'No Convergence !'
      write(*,*)  'No Convergence !'
      stop
      return
      end
c
c
c
c
c
c
c
c
c     Subroutine Residuals to Input System of Non-Linear Equations
      subroutine Residuals(N,YOLD,UNKNON,R)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION UNKNON(N),YOLD(N),FUN(N),FUNOLD(N),R(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
c
      do i = 1, N
      R(i) = 0.d0
      enddo
c
c     Evaluate the RHS Functions of IVP's for YOLD
      call Func(N,YOLD,FUNOLD)
c
c     Evaluate the RHS Functions of IVP's for UNKNON
      call Func(N,UNKNON,FUN)
c
c     Evaluate the Residual Equations
      do i = 1, N
        R(i) = UNKNON(i)-YOLD(i)-h/2.d0*(FUN(i)+FUNOLD(i))
      enddo
c
      return
      end
c
c
c
c
c
c
c
```

```
c
c     Subroutine jacobian to Evaluate the Jacobian Matrix
      subroutine jacobian(IFLAG,N,UNKNON,A)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION A(N,N),R(N),RJ(N),UNKNON(N),UNKNONJ(N),DUMMY(N)
      COMMON x1,x2,h,phi1,phi2,itmax,tol
      parameter (epsl=1.d-5)
c
      do i = 1, N
      do j = 1, N
        A(i,j)=0.d0
      enddo
      enddo
c
      if (IFLAG .gt. 0) go to 10
c
c     Evaluate the jacobian matrix analatically
      A(1,1) = 1.d0+h/2.d0*phi1
      A(1,2) = 0.d0
      A(2,1) =-h/2.d0*phi1
      A(2,2) = 1.d0+h*phi2*UNKNON(2)
      return
c
c     Evaluate the jacobian matrix numerically
 10   do i = 1, N
        UNKNONJ(i)= UNKNON(i)
        DUMMY(i)  = 0.d0
      enddo
      call residuals(N,DUMMY,UNKNON,R)
      do i = 1, N
        diff  = dmax1(epsl,dabs(epsl*UNKNON(i)))
        UNKNONJ(i) = UNKNON(i)+diff
        call residuals(N,DUMMY,UNKNONJ,RJ)
        do j = 1, N
          A(j,i) = (RJ(j)-R(j))/diff
        enddo
        UNKNONJ(i)=UNKNON(i)
      enddo
      return
      end
c
c
c
c
c
c
c
c     subroutine dgefa to LU decompose matrix A
      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)
c
c     dgefa factors a double precision matrix by gaussian elimination.
c
c     dgefa is usually called by dgeco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for dgeco) = (1 + 9/n)*(time for dgefa) .
c
c     on entry
c
c        a       double precision(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
```

```
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that dgesl or dgedi will divide by zero
c                     if called.  use  rcond  in dgeco for a reliable
c                     indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,dscal,idamax
c
c     internal variables
c
      double precision t
      integer idamax,j,k,kp1,l,nm1
c
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
c
         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
c
c        zero pivot implies this column already triangularized
c
         if (a(l,k) .eq. 0.0d0) go to 40
c
c           interchange if necessary
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -1.0d0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
```

```
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d0) info = n
      return
      end
      subroutine dgesl(a,lda,n,ipvt,b,job)
      integer lda,n,ipvt(1),job
      double precision a(lda,1),b(1)
c
c     dgesl solves the double precision system
c     a * x = b  or  trans(a) * x = b
c     using the factors computed by dgeco or dgefa.
c
c     on entry
c
c        a       double precision(lda, n)
c                the output from dgeco or dgefa.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c        ipvt    integer(n)
c                the pivot vector from dgeco or dgefa.
c
c        b       double precision(n)
c                the right hand side vector.
c
c        job     integer
c                = 0         to solve  a*x = b ,
c                = nonzero   to solve  trans(a)*x = b  where
c                            trans(a)  is the transpose.
c
c     on return
c
c        b       the solution vector  x .
c
c     error condition
c
c        a division by zero will occur if the input factor contains a
c        zero on the diagonal.  technically this indicates singularity
c        but it is often caused by improper arguments or improper
c        setting of lda .  it will not occur if the subroutines are
c        called correctly and if dgeco has set rcond .gt. 0.0
c        or dgefa has set info .eq. 0 .
c
c     to compute  inverse(a) * c  where  c  is a matrix
c     with  p  columns
c           call dgeco(a,lda,n,ipvt,rcond,z)
c           if (rcond is too small) go to ...
c           do 10 j = 1, p
c              call dgesl(a,lda,n,ipvt,c(1,j),0)
c        10 continue
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,ddot
c
c     internal variables
c
      double precision ddot,t
      integer k,kb,l,nm1
c
      nm1 = n - 1
      if (job .ne. 0) go to 50
```

```
c
c        job = 0 , solve  a * x = b
c        first solve  l*y = b
c
         if (nm1 .lt. 1) go to 30
         do 20 k = 1, nm1
            l = ipvt(k)
            t = b(l)
            if (l .eq. k) go to 10
               b(l) = b(k)
               b(k) = t
   10       continue
            call daxpy(n-k,t,a(k+1,k),1,b(k+1),1)
   20    continue
   30    continue
c
c        now solve  u*x = y
c
         do 40 kb = 1, n
            k = n + 1 - kb
            b(k) = b(k)/a(k,k)
            t = -b(k)
            call daxpy(k-1,t,a(1,k),1,b(1),1)
   40    continue
      go to 100
   50 continue
c
c        job = nonzero, solve  trans(a) * x = b
c        first solve  trans(u)*y = b
c
         do 60 k = 1, n
            t = ddot(k-1,a(1,k),1,b(1),1)
            b(k) = (b(k) - t)/a(k,k)
   60    continue
c
c        now solve trans(l)*x = y
c
         if (nm1 .lt. 1) go to 90
         do 80 kb = 1, nm1
            k = n - kb
            b(k) = b(k) + ddot(n-k,a(k+1,k),1,b(k+1),1)
            l = ipvt(k)
            if (l .eq. k) go to 70
               t = b(l)
               b(l) = b(k)
               b(k) = t
   70       continue
   80    continue
   90    continue
  100 continue
      return
      end
      integer function idamax(n,dx,incx)
c
c     finds the index of element having max. absolute value.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dmax
      integer i,incx,ix,n
c
      idamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      ix = 1
      dmax = dabs(dx(1))
```

```
      ix = ix + incx
      do 10 i = 2,n
         if(dabs(dx(ix)).le.dmax) go to 5
         idamax = i
         dmax = dabs(dx(ix))
    5    ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
   20 dmax = dabs(dx(1))
      do 30 i = 2,n
         if(dabs(dx(i)).le.dmax) go to 30
         idamax = i
         dmax = dabs(dx(i))
   30 continue
      return
      end
      subroutine daxpy(n,da,dx,incx,dy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end
      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
```

```
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     *   dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
      subroutine  dscal(n,da,dx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to one.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision da,dx(*)
      integer i,incx,m,mp1,n,nincx
c
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dx(i) = da*dx(i)
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
```

```
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
     dx(i) = da*dx(i)
30 continue
   if( n .lt. 5 ) return
40 mp1 = m + 1
   do 50 i = mp1,n,5
     dx(i) = da*dx(i)
     dx(i + 1) = da*dx(i + 1)
     dx(i + 2) = da*dx(i + 2)
     dx(i + 3) = da*dx(i + 3)
     dx(i + 4) = da*dx(i + 4)
50 continue
   return
   end
```