

```

c
c Shooting Method
program shoot_2eq
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
PARAMETER (N=4)
double precision Y(N),YOLD(N),A(N/2,N/2),B(N/2)
integer ipvt(N/2)
COMMON h,x1,x2,tol,itmax
OPEN(11,FILE='shoot_2eq.txt')
c
h      = 5.d-2
x1     = 0
x2     = 1.d0
tol    = 1.d-12
eps1   = 1.d-5
itmax  = 10
c
c provide initial condition for YOLD(N)
Beta_1 = 1.d0
Beta_2 = 2.d0
z1_1   = 0.5d0
z1_2   = z1_1+eps1
z2_1   = 0.6d0
z2_2   = z2_1+eps1
c
c Integrate IVP's for the first guess of IC's
YOLD(1) = 0.d0
YOLD(2) = z1_1
YOLD(3) = 0.d0
YOLD(4) = z2_1
call Adams_Bashford(N,YOLD,Y)
phi1_1  = Y(1)
phi2_1  = Y(3)
c
c Integrate IVP's for the second guess of IC's
YOLD(1) = 0.d0
YOLD(2) = z1_2
YOLD(3) = 0.d0
YOLD(4) = z2_1
call Adams_Bashford(N,YOLD,Y)
phi1_2  = Y(1)
phi2_2  = Y(3)
c
c Integrate IVP's for the third guess of IC's
YOLD(1) = 0.d0
YOLD(2) = z1_1
YOLD(3) = 0.d0
YOLD(4) = z2_2
call Adams_Bashford(N,YOLD,Y)
phi1_3  = Y(1)
phi2_3  = Y(3)
c
c Construct the Jacobian for the IC's
A(1,1) = (phi1_2-phi1_1)/(z1_2-z1_1)
A(1,2) = (phi1_3-phi1_1)/(z2_2-z2_1)
A(2,1) = (phi2_2-phi2_1)/(z1_2-z1_1)
A(2,2) = (phi2_3-phi2_1)/(z2_2-z2_1)
c
c LU-decompose matrix A
c Note that matrix A will contains on return the L and U matrixes (original is destroyed)
c
call dgefa(A,N/2,N/2,ipvt,info)
ishoot = 0
c
c Shooting Method Loop
10 ishoot = ishoot + 1
c
c Set Termination Criteria for the Shooting Method Loop
if (ishoot .gt. itmax) go to 1000
B(1) = -(phi1_1-beta_1)
B(2) = -(phi2_1-beta_2)
c
c Solve the equations by backsubstitution

```

```

c      Note that vector B will contains the unknowns X (original is destroyed)
c
c      call dgesl(A,N/2,N/2,ipvt,B,info)
z1_1   = z1_1+B(1)
z2_1   = z2_1+B(2)
c
c      Integrate IVP's for the third guess of IC's
YOLD(1) = 0.d0
YOLD(2) = z1_1
YOLD(3) = 0.d0
YOLD(4) = z2_1
call Adams_Bashford(N,YOLD,Y)
phil_1 = Y(1)
phi2_1 = Y(3)
R1_1   = phil_1-beta_1
R2_1   = phi2_1-beta_2
rnorm  = dsqrt(R1_1*R1_1+R2_1*R2_1)
write(*,'(5x,a,i3,5x,a,f20.12)')
!      'iteration =',ishoot,'error =',rnorm
if (rnorm .lt. tol) stop
go to 10
1000 write(11,*) 'No Convergence !'
write(*,*) 'No Convergence !'
c
c      stop
c      end
c
c      END of main Program
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 h,x1,x2,tol,itmax
c
do i = 1, N
FUN(i) = 0.d0
enddo
c
FUN(1) = Y(2)
FUN(2) = Y(1)+Y(3)
FUN(3) = Y(4)
FUN(4) = Y(1)+Y(3)
c
return
end
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 h,x1,x2,tol,itmax
c
istep = 0
xval  = x1
c
c      Print the Initial solution to output.txt
write(11,*)
write(11,'(13x,a,18x,a,16x,a,16x,a,16x,a)') 't','Y(1)','Y(2)',
!      'Y(3)','Y(4)'
write(11,'(5f20.12)') xval,YOLD(1),YOLD(2),YOLD(3),YOLD(4)
c
c      Evaluate the RHS Function for the Initial Solution
call Func(N,YOLD,FUNOLD1)
c
c      Start Integration Loop
c      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
c if (istep .eq. 1) then
c   do i = 1, N
c     Y(i) = YOLD(i)+h*FUNOLD1(i)
c   enddo
c else
c
c Apply 2nd-order AB Formula for Remaining Time Steps
c   do i = 1, N
c     Y(i) = YOLD(i)+h/2.d0*(3.d0*FUNOLD1(i)-FUNOLD2(i))
c   enddo
c endif
c
c Print the solution for the current time step to shooting.txt
c write(11,'(5f20.12)') xval,Y(1),Y(2),Y(3),Y(4)
c
c Store Previous Time Values
c do i = 1, N
c   YOLD(i) = Y(i)
c   FUNOLD2(i) = FUNOLD1(i)
c enddo
c
c Evaluate the RHS Function at Current Time
c call Func(N,YOLD,FUNOLD1)
c go to 10
c
c end
c
c END of main Program
c
c subroutine dgefa(a,lda,n,ipvt,info)
c integer lda,n,ipvt(1),info
c 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
c      double precision t
c      integer idamax,j,k,kp1,l,nml
c
c      gaussian elimination with partial pivoting
c
c      info = 0
c      nml = n - 1
c      if (nml .lt. 1) go to 70
c      do 60 k = 1, nml
c         kp1 = k + 1
c
c         find l = pivot index
c
c         l = idamax(n-k+1,a(k,k),1) + k - 1
c         ipvt(k) = l
c
c         zero pivot implies this column already triangularized
c
c         if (a(l,k) .eq. 0.0d0) go to 40
c
c         interchange if necessary
c
c         if (l .eq. k) go to 10
c         t = a(l,k)
c         a(l,k) = a(k,k)
c         a(k,k) = t
10      continue
c
c         compute multipliers
c
c         t = -1.0d0/a(k,k)
c         call dscal(n-k,t,a(k+1,k),1)
c
c         row elimination with column indexing
c
c         do 30 j = kp1, n
c         t = a(l,j)
c         if (l .eq. k) go to 20
c         a(l,j) = a(k,j)
c         a(k,j) = t
20      continue
c         call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
30      continue
c         go to 50
40      continue
c         info = k
50      continue
60 continue
70 continue
c      ipvt(n) = n
c      if (a(n,n) .eq. 0.0d0) info = n
c      return
c      end
c      subroutine dgesl(a,lda,n,ipvt,b,job)
c      integer lda,n,ipvt(1),job
c      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

```

```

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
c double precision ddot,t
c integer k,kb,l,nml
c
c nml = n - 1
c if (job .ne. 0) go to 50
c
c      job = 0 , solve  a * x = b
c      first solve  l*y = b
c
c      if (nml .lt. 1) go to 30
c      do 20 k = 1, nml
c          l = ipvt(k)
c          t = b(l)
c          if (l .eq. k) go to 10
c          b(l) = b(k)
c          b(k) = t
c      10      continue
c      call daxpy(n-k,t,a(k+1,k),1,b(k+1),1)
c      20      continue
c      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(1)
          b(1) = 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
      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
      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   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 mpl = m + 1
  do 50 i = mpl,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,mpl,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
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 mpl = m + 1
  do 50 i = mpl,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

```



