

```

c
IMPLICIT DOUBLE PRECISION (a-h,o-z)
parameter (N=10,N1=N+1)
DOUBLE PRECISION Y(N1),Y_G(N1)
COMMON /CHEB/ z1,z2,PI,Alpha,Beta
COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
COMMON /COLLEC/ D(N1,N1),DD(N1,N1)
COMMON /PARAM/ Phi,Bi
OPEN(11,FILE='Chebyshive_Collocation.txt')

c
PI    = 4.d0*DATAN(1.d0)
z1    = 0.d0
z2    = 1.d0
Phi   = 1.d0
Bi    = 1.d0

c
call Chebyshev

c
do i = 1, N1
    Y_G(i) = 1.d0/(dexp(-1.d0)-dexp(-4.d0))*
!      (dexp(-Z(i))-dexp(-4.d0*Z(i)))*0.9
enddo

c
call Newton(Y_G,Y)

c
do i = 1, N1
    ye = 1.d0/(dexp(-1.d0)-dexp(-4.d0))*
!      (dexp(-Z(i))-dexp(-4.d0*Z(i)))
    write(11,'(3f16.6)') Z(i),Y(i),ye
    write(* ,'(3f16.6)') Z(i),Y(i),ye
enddo

c
1000 stop
end

c
c
c
c
c
c
c
c
Subroutine Newton to solve a system of algebraic equations iteratively
subroutine Newton(UNKNON_G,UNKNON)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
parameter (N=10,N1=N+1)
DOUBLE PRECISION A(N1,N1),UNKNON_G(N1),UNKNON(N1),B(N1)
COMMON /COLLEC/ D(N1,N1),DD(N1,N1)
integer ipvt(N1)

c
itmax = 10
tol    = 1.d-12

c
initialize matrix A and vector B
do i = 1, N1
UNKNON(i) = UNKNON_G(i)
B(i) = 0.d0
do j = 1, N1
    A(i,j)=0.d0
enddo
enddo

c
Start Iterative Solution
iter = 0
10 iter = iter + 1
if (iter .gt. itmax) go to 12

c
Input system of nonlinear equations
on return B contains the residuals
call residuals(UNKNON,B)

c
Check the norm of the residuals
rnorm = 0.d0

```



```

        enddo
c
c      Boundary Conditions
c      In Collocation, the first equation for the first BC and the last equation for the second BC
c
c      R(1 ) = Y(1 )-0.d0
c      R(N1) = Y(N1)-1.d0
c
c      return
c      end
c
c
c
c
c
c
c
c
c
c      Subroutine jacobian to Evaluate the Jacobian Matrix
c      subroutine jacobian(IFLAG,UNKNON,A)
c      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
c      parameter (N=10,N1=N+1)
c      double precision A(N1,N1),UNKNON(N1),UNKNONJ(N1),R(N1),RJ(N1)
c      parameter (epsl=1.d-5)
c
c      do i = 1, N1
c      do j = 1, N1
c      A(i,j)=0.d0
c      enddo
c      enddo
c
c      if (IFLAG .gt. 0) go to 10
c
c      Evaluate the jacobian matrix analatically
c      return
c
c      Evaluate the jacobian matrix numerically
10  do i = 1, N1
c      UNKNONJ(i) = UNKNON(i)
c      enddo
c
c      call residuals(UNKNON,R)
c
c      do i = 1, N1
c      diff = dmax1(epsl,dabs(epsl*UNKNON(i)))
c      UNKNONJ(i) = UNKNON(i)+diff
c      call residuals(UNKNONJ,RJ)
c      do j = 1, N1
c      A(j,i) = (RJ(j)-R(j))/diff
c      enddo
c      UNKNONJ(i)=UNKNON(i)
c      enddo
c      return
c      end
c
c
c
c      SUBROUTINE Chebyshev
c      IMPLICIT DOUBLE PRECISION (a-h,o-z)
c      parameter (N=10,N1=N+1)
c      DOUBLE PRECISION XCHEB(N1)
c      COMMON /CHEB/ z1,z2,PI,Alpha,Beta
c      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
c      COMMON /COLLEC/ D(N1,N1),DD(N1,N1)
c
c      Vector XCHEB (1,-1) contains the Gauss-Lobatto grid.
c      do i = 1, N1
c      XCHEB(i) = dcos((i-1)*pi/N)
c      CBAR(i) = 1.d0
c      C(i) = 1.d0
c      enddo
c

```

```

C(1)      = 2.d0
CBAR(1)   = 2.d0
CBAR(N1)  = 2.d0

c
c  Vector X maps XCHEB to the interval (z1,z2) ==> X(i) = Alpha + Beta*XCHEB(i)
Alpha = (z1+z2)/2.d0
Beta  = (z1-z2)/2.d0

c
do i = 1, N1
  Z(i) = Alpha + Beta*XCHEB(i)
enddo

c
do i = 1, N1
do j = 1, N1
  CT(i,j) = 2.d0/N/CBAR(i)/CBAR(j)*dcos(pi*(i-1)*(j-1)/N)
  CTINV(i,j) = dcos(pi*(i-1)*(j-1)/N)
enddo
enddo

c
do i = 1, N1
  ii = i-1
  do j = 1, N1
    jj = j-1
    if (ii .ne. jj) then
      D(i,j) = CBAR(i)/CBAR(j)*(-1)**(ii+jj)/(XCHEB(i)-XCHEB(j))
    else
      if (ii .ne. 0 .and. ii .ne. N) then
        D(i,j) = -XCHEB(j)/2.d0/(1.d0-XCHEB(j)**2)
      endif
    endif
  enddo
enddo
D( 1, 1) = (2.d0*db1e(N)**2+1.d0)/6.d0
D(N1,N1) = -(2.d0*db1e(N)**2+1.d0)/6.d0

c
do i=1,N1
  do j=1,N1
    DD(i,j) = 0.d0
    do k=1,N1
      DD(i,j) = DD(i,j) + D(i,k)*D(k,j)
    enddo
  enddo
enddo

c
do i = 1, N1
  do j = 1, N1
    D(i,j) = D(i,j)/BETA
    DD(i,j) = DD(i,j)/BETA/BETA
  enddo
enddo

c
return
end

c
c
c
SUBROUTINE Derivative(YHAT,D1YHAT,D2YHAT)
IMPLICIT DOUBLE PRECISION (a-h,o-z)
parameter (N=10,N1=N+1)
DOUBLE PRECISION YHAT(N1),D1YHAT(N1),D2YHAT(N1)
COMMON /CHEB/ z1,z2,PI,Alpha,Beta
COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)

c
do i = 1, N1
  D1YHAT(i) = 0.d0
  D2YHAT(i) = 0.d0
enddo

c
do i = 1, N1
  ri = db1e(i) - 1.d0
c  1st-order derivatives
  do ip = i + 1, N1, 2

```



```

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,kpl,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     kpl = 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

```

```

do 30 j = kp1, n
  t = a(1,j)
  if (1 .eq. k) go to 20
  a(1,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
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
10      continue
c          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
c      do 40 kb = 1, n
c          k = n + 1 - kb
c          b(k) = b(k)/a(k,k)
c          t = -b(k)
c          call daxpy(k-1,t,a(1,k),1,b(1),1)
40      continue
c      go to 100
50      continue
c
c      job = nonzero, solve trans(a) * x = b
c      first solve trans(u)*y = b
c
c      do 60 k = 1, n
c          t = ddot(k-1,a(1,k),1,b(1),1)
c          b(k) = (b(k) - t)/a(k,k)
60      continue
c
c      now solve trans(l)*x = y
c
c      if (nml .lt. 1) go to 90
c      do 80 kb = 1, nml
c          k = n - kb
c          b(k) = b(k) + ddot(n-k,a(k+1,k),1,b(k+1),1)
c          l = ipvt(k)
c          if (l .eq. k) go to 70
c              t = b(l)
c              b(l) = b(k)
c              b(k) = t
70      continue
80      continue
90      continue
100     continue
c      return
c      end
c      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
c      double precision dx(*),dmax
c      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(i)).le.dmax) go to 5
    idamax = i
    dmax = dabs(dx(i))
  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      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,mpl,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 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
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
c

```