```
c
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION Y(N1),YHAT(N1),YHAT_G(N1)
      COMMON /CHEB/ z1,z2,PI,Alpha,Beta
      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
      COMMON /PARAM/ Phi,Bi
      OPEN(11,FILE='chebyshive_tau.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(i) = 1.d0/(dexp(-1.d0)-dexp(-4.d0))*
     !        (dexp(-Z(i))-dexp(-4.d0*Z(i)))
      enddo
c
      call resp(Y,YHAT_G)
c
      call Newton(YHAT_G,YHAT)
c
      call spre(YHAT,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,'(3f20.14)') Z(i),Y(i),ye
         write(* ,'(3f20.14)') Z(i),Y(i),ye
      enddo
c
1000  stop
      end
c
c
c
c
c
c
c
c     Subroutine Newton to to solve a system of algebraic equations iteratively
      subroutine Newton(UNKNON_G,UNKNON)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION A(N1,N1),UNKNON_G(N1),UNKNON(N1),B(N1)
      integer ipvt(N1)
c
      itmax = 10
      tol   = 1.d-12
c
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
c     Start Iterative Solution
      iter = 0
  10  iter = iter + 1
      if (iter .gt. itmax) go to 12
c
c     Input system of nonlinear equations
c     on return B contains the residuals
      call residuals(UNKNON,B)
c
```

```
c     Check the norm of the residuals
      rnorm = 0.d0
      do i = 1, N1
      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,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,N1,N1,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, N1
        B(i) = -B(i)
      enddo
c
      call dgesl(A,N1,N1,ipvt,B,info)
c
      do i= 1, N1
        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
c
c
c
c     Subroutine Residuals to Input System of Non-Linear Equations
      SUBROUTINE RESIDUALS(YHAT,RHAT)
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION Y(N1),YHAT(N1),R(N1),RHAT(N1)
      DOUBLE PRECISION D1YHAT(N1),D2YHAT(N1),D1Y(N1),D2Y(N1)
      COMMON /CHEB/ z1,z2,PI,Alpha,Beta
      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
      COMMON /PARAM/ Phi,Bi
c
      do i = 1, N1
        R(i)    = 0.d0
        RHAT(i) = 0.d0
      enddo
c
c     Evaluate the derivatines in the spectral domain
c
      call Derivative(YHAT,D1YHAT,D2YHAT)
c
      call spre(YHAT,Y)
      call spre(D1YHAT,D1Y)
      call spre(D2YHAT,D2Y)
```

```
c
c     Construct Residuals in the Physical Domain then Transform to Spectral Domain
c
      do i = 1, N1
        R(i) = (D2Y(i)+5.d0*D1Y(i)+4.d0*Y(i))*Y(i)
      enddo
c
c     Boundary Conditions
c     In physical domain, the first equation for the first BC and the last equation for the second
BC
c
      R(1 ) = Y(1 )-0.d0
      R(N1) = Y(N1)-1.d0
c
c     Transform the residuals back to the spectral domain
c
      call resp(R,RHAT)
c
      return
      end
c
c
c
c
c
c
c
c
c     Subroutine jacobian to Evaluate the Jacobian Matrix
      subroutine jacobian(IFLAG,UNKNON,A)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (N=16,N1=N+1)
      double precision A(N1,N1),UNKNON(N1),UNKNONJ(N1),FUN(N1),FUNJ(N1)
      parameter (epsl=1.d-5)
c
      do i = 1, N1
      do j = 1, N1
        A(i,j)=0.d0
      enddo
      enddo
c
      if (IFLAG .gt. 0) go to 10
c
c     Evaluate the jacobian matrix analatically
      return
c
c     Evaluate the jacobian matrix numerically
 10   do i = 1, N1
        UNKNONJ(i)=UNKNON(i)
      enddo
      call residuals(UNKNON,FUN)
      do i = 1, N1
        diff  = dmax1(epsl,dabs(epsl*UNKNON(i)))
        UNKNONJ(i) = UNKNON(i)+diff
        call residuals(UNKNONJ,FUNJ)
        do j = 1, N1
          A(j,i) = (FUNJ(j)-FUN(j))/diff
        enddo
        UNKNONJ(i)=UNKNON(i)
      enddo
      return
      end
c
c
c
      SUBROUTINE Chebyshev
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION XCHEB(N1)
      COMMON /CHEB/ z1,z2,PI,Alpha,Beta
      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
c
```

```
c     Vector XCHEB (1,-1) contains the Gauss-Lobatto grid.
      do i = 1, N1
        XCHEB(i) = dcos((i-1)*pi/N)
        CBAR(i)  = 1.d0
        C(i)     = 1.d0
      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
      return
      end
c
c
c
      SUBROUTINE Derivative(YHAT,D1YHAT,D2YHAT)
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (N=16,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 = dble(i) - 1.d0
c       1st-oder derivatives
        do ip = i + 1, N1, 2
          rp = dble(ip) - 1.d0
          D1YHAT(i) = D1YHAT(i) + (2.d0/C(i)*rp)/Beta*YHAT(ip)
        enddo
c       2nd-oder derivatives
        do ip = i + 2, N1, 2
          rp = dble(ip) - 1.d0
          D2YHAT(i) = D2YHAT(i) +
     !    1.d0/C(i)*rp*(rp*rp-ri*ri)/Beta/Beta*YHAT(ip)
        enddo
      enddo
c
      return
      end
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      subroutine RESP(AR,AS)
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      implicit double precision (a-h,o-z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION AR(N1),AS(N1)
      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
c
c     invert chebychev
c
      do i = 1, N1
        AS(i) = 0.d0
```

```
        do j = 1, N1
          AS(i) = AS(i) + CT(i,j) * AR(j)
        enddo
      enddo
c
      return
      end
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      subroutine SPRE(AS,AR)
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      implicit double precision (a-h,o-z)
      parameter (N=16,N1=N+1)
      DOUBLE PRECISION AR(N1),AS(N1)
      COMMON /TRANS/ C(N1),CBAR(N1),CT(N1,N1),CTINV(N1,N1),Z(N1)
c
c     invert chebychev
c
      do i = 1, N1
        AR(i) = 0.d0
        do j = 1, N1
          AR(i) = AR(i) + CTINV(i,j) * AS(j)
        enddo
      enddo
c
      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
c
```