

```

c
  IMPLICIT DOUBLE PRECISION (a-h,o-z)
  parameter (N=6,N1=N+1)
  DOUBLE PRECISION A(N1,N1),B(N1)
  INTEGER IPVT(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)
  OPEN(11,FILE='Chebyshev_Collocation.txt')

c
  PI      = 4.d0*ATAN(1.d0)
  z1      = 0.d0
  z2      = 1.d0

c
  call Chebyshev

c
  call MATRIX(A)

c
  call dgefa(A,N1,N1,ipvt,info)

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

c
  call dgesl(A,N1,N1,ipvt,B,job)

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),B(i),ye
    write(* ,'(3f16.6)') Z(i),B(i),ye
  enddo

c
1000 stop
end

c
c
c
  SUBROUTINE MATRIX(A)
  IMPLICIT DOUBLE PRECISION (a-h,o-z)
  parameter (N=6,N1=N+1)
  DOUBLE PRECISION A(N1,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)

c
  do i = 1, N1
    do j = 1, N1
      A(i,j) = 0.d0
    enddo
  enddo

c
  Construct Residuals using Chebyshev Collocation

c
  do i = 2, N1-1
    A(i,i) = 4.d0
    do j = 1, N1
      A(i,j) = A(i,j)+DD(i,j)+5.d0*D(i,j)
    enddo
  enddo

c
  Implement boundary conditions

c
  A( 1, 1) = 1.d0
  A(N1,N1) = 1.d0

c
  return
end

c
c
c

```

```

SUBROUTINE Chebyshev
IMPLICIT DOUBLE PRECISION (a-h,o-z)
parameter (N=6,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)
COMMON /COLLEC/ D(N1,N1),DD(N1,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
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
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
subroutine RESP(AR,AS)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
implicit double precision (a-h,o-z)

```



```

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
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
c 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 = kpl, 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
c 20      continue
c          call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
c 30      continue
c      go to 50
c 40      continue
c          info = k
c 50      continue
c 60      continue
c 70      continue
c      ipvt(n) = n
c      if (a(n,n) .eq. 0.0d0) info = n
c      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
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

```

```

        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       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      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
c      do 10 i = 1,nincx,incx
c        dx(i) = da*dx(i)
10 continue
  return
c
c      code for increment 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
    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
```

c