```
c
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (NX=16,NY=16,NX1=NX+1,NY1=NY+1)
      DOUBLE PRECISION A(NX1*NY1,NX1*NY1),B(NX1*NY1)
      INTEGER IPVT(NX1*NY1)
      COMMON /CHEB/ X1,X2,Y1,Y2,PI,Alpha,Beta
      COMMON /TRANS/ X(NX1),Y(NY1)
      COMMON /COLLEC/ DX(NX1,NX1),DDX(NX1,NX1),DY(NY1,NY1),DDY(NY1,NY1)
      OPEN(11,FILE='Elliptic_PDE.txt')
c
      PI    = 4.d0*ATAN(1.d0)
      x1    = 0.d0
      x2    = 1.d0
      y1    = 0.d0
      y2    = 1.d0
c
      call Chebyshev
c
      call MATRIX(A)
c      do i = 1, NX1*NY1
c         write(11,'(24f8.2)') (A(i,j) , j = 1, NX1*NY1)
c      enddo
c
      call dgefa(A,NX1*NY1,NX1*NY1,ipvt,info)
c
      do i = 1, NX1*NY1
         B(i) = 0.d0
      enddo
c
c     Implement boundary conditions
c
      do j = 1, NY1
         B((  1-1)*NY1+j) = 1.d0
         B((NX1-1)*NY1+j) = 1.d0
      enddo
c
      do i = 1, NX1
         B((i-1)*NY1+  1) = 1.d0
      enddo
c
      do i = 2, NX1-1
         B((i-1)*NY1+NY1) = 0.d0
      enddo
c
      call dgesl(A,NX1*NY1,NX1*NY1,ipvt,B,job)
c
      write(11,'(20x,17f10.4)') (X(i), i = 1, NX1)
      write(11,*)
      do j = 1, NY1
      write(11,'(f10.4,10x,17f10.4)') Y(j),(B((i-1)*NY1+j), i = 1, NX1)
      enddo
c      do j = 1, NY1
c      write(11,'(9f10.4)') B(j),B(NY1+j),B(2*NY1+j),B(3*NY1+j),
c     !B(4*NY1+j),B(5*NY1+j),B(6*NY1+j),B(7*NY1+j),B(8*NY1+j)
c      enddo
c
1000  stop
      end
c
c
c
      SUBROUTINE MATRIX(A)
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (NX=16,NY=16,NX1=NX+1,NY1=NY+1)
      DOUBLE PRECISION A(NX1*NY1,NX1*NY1)
      COMMON /CHEB/ X1,X2,Y1,Y2,PI,Alpha,Beta
      COMMON /TRANS/ X(NX1),Y(NY1)
      COMMON /COLLEC/ DX(NX1,NX1),DDX(NX1,NX1),DY(NY1,NY1),DDY(NY1,NY1)
c
      do i = 1, NX1*NY1
      do j = 1, NX1*NY1
        A(i,j) = 0.d0
```

```
      enddo
      enddo
c
c     Construct Residuals using Chebyshev Collocation
c
      do i = 2, NX1-1
        do j = 2, NY1-1
          do n = 1, NX1
            A((i-1)*NY1+j,(n-1)*NY1+j) = A((i-1)*NY1+j,(n-1)*NY1+j) +
     !                                   DDX(i,n)
          enddo
          do m = 1, NY1
            A((i-1)*NY1+j,(i-1)*NY1+m) = A((i-1)*NY1+j,(i-1)*NY1+m) +
     !                                   DDY(j,m)
          enddo
        enddo
      enddo
c
c     Implement boundary conditions
c
      do j = 1, NY1
         A((  1-1)*NY1+j,(  1-1)*NY1+j) = 1.d0
         A((NX1-1)*NY1+j,(NX1-1)*NY1+j) = 1.d0
      enddo
c
      do i = 1, NX1
         A((i-1)*NY1+  1,(i-1)*NY1+  1) = 1.d0
      enddo
c
      do i = 2, NX1-1
         A((i-1)*NY1+NY1,(i-1)*NY1+NY1) = 1.d0
      enddo
c
      return
      end
c
c
c
      SUBROUTINE Chebyshev
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      parameter (NX=16,NY=16,NX1=NX+1,NY1=NY+1)
      DOUBLE PRECISION XCHEB(NX1),CX(NX1),CBARX(NX1)
      DOUBLE PRECISION YCHEB(NY1),CY(NY1),CBARY(NY1)
      COMMON /CHEB/ X1,X2,Y1,Y2,PI,Alpha,Beta
      COMMON /TRANS/ X(NX1),Y(NY1)
      COMMON /COLLEC/ DX(NX1,NX1),DDX(NX1,NX1),DY(NY1,NY1),DDY(NY1,NY1)
c
c     Vector XCHEB (1,-1) contains the Gauss-Lobatto grid.
      do i = 1, NX1
        XCHEB(i) = dcos((i-1)*pi/NX)
        CBARX(i)  = 1.d0
        CX(i)     = 1.d0
      enddo
      do i = 1, NY1
        YCHEB(i)  = dcos((i-1)*pi/NY)
        CBARY(i)  = 1.d0
        CY(i)     = 1.d0
      enddo
c
      CX(1)      = 2.d0
      CBARX(1)   = 2.d0
      CBARX(NX1) = 2.d0
      CY(1)      = 2.d0
      CBARY(1)   = 2.d0
      CBARY(NY1) = 2.d0
c
c     Vector X maps XCHEB to the interval (x1,x2) ==> X(i) = Alpha + Beta*XCHEB(i)
      Alpha1 = (x1+x2)/2.d0
      Beta1  = (x1-x2)/2.d0
      Alpha2 = (y1+y2)/2.d0
      Beta2  = (y1-y2)/2.d0
c
```

```
      do i = 1, NX1
        X(i) = Alpha1 + Beta1*XCHEB(i)
      enddo
c
      do i = 1, NY1
        Y(i) = Alpha2 + Beta2*YCHEB(i)
      enddo
c
      do i = 1, NX1
         ii = i-1
         do j = 1, NX1
            jj = j-1
            if (ii .ne. jj) then
              DX(i,j) = CBARX(i)/CBARX(j)*(-1)**(ii+jj)
     !                  /(XCHEB(i)-XCHEB(j))
            else
              if (ii .ne. 0 .and. ii .ne. NX) then
              DX(i,j) = -XCHEB(j)/2.d0/(1.d0-XCHEB(j)**2)
              endif
            endif
         enddo
      enddo
      DX( 1, 1) = (2.d0*dble(NX)**2+1.d0)/6.d0
      DX(NX1,NX1) =-(2.d0*dble(NX)**2+1.d0)/6.d0
c
      do i=1,NX1
        do j=1,NX1
          DDX(i,j) = 0.d0
          do k=1,NX1
            DDX(i,j) = DDX(i,j) + DX(i,k)*DX(k,j)
          enddo
        enddo
      enddo
c
      do i = 1, NX1
         do j = 1, NX1
             DX(i,j) =  DX(i,j)/BETA1
            DDX(i,j) = DDX(i,j)/BETA1/BETA1
         enddo
      enddo
c
      do i = 1, NY1
         ii = i-1
         do j = 1, NY1
            jj = j-1
            if (ii .ne. jj) then
              DY(i,j) = CBARY(i)/CBARY(j)*(-1)**(ii+jj)
     !                  /(YCHEB(i)-YCHEB(j))
            else
              if (ii .ne. 0 .and. ii .ne. NY) then
              DY(i,j) = -YCHEB(j)/2.d0/(1.d0-YCHEB(j)**2)
              endif
            endif
         enddo
      enddo
      DY( 1, 1) = (2.d0*dble(NY)**2+1.d0)/6.d0
      DY(NY1,NY1) =-(2.d0*dble(NY)**2+1.d0)/6.d0
c
      do i=1,NY1
        do j=1,NY1
          DDY(i,j) = 0.d0
          do k=1,NY1
            DDY(i,j) = DDY(i,j) + DY(i,k)*DY(k,j)
          enddo
        enddo
      enddo
c
      do i = 1, NY1
         do j = 1, NY1
             DY(i,j) =  DY(i,j)/BETA2
            DDY(i,j) = DDY(i,j)/BETA2/BETA2
         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
```