```
C     * one-s and three-p orbitals on a simple cubic lattice *
C     * including spin-orbit coupling                        *
C     * Then do One Dimensional DOS for the [001] surface    *
C
C     SSS = s to s sigma hopping
C     SPS = s to p sigma hopping
C     PPS = p to p sigma hopping
C     S0, P0 = onsite energies of s and p orbital
C     The input is given in the file fort.30
C     ZPLANES = counter that controls the number of z-slices
C     TRAJECT(3,120) = Array that contains the k-points
C     AR(D,D) = real part of the matrix element
C     AI(D,D) = imaginary part of the matrix element
C     If an element is a + ib, then AR = a and AI = b
C     PIE = mathematical constant
C     SOC = strength of spin-orbit coupling
C     I,J,K,L,M,O = counters for loops
C     N = counter for numbering the output files
C     The output files are numbered from fort.31
C     E(D) = array for eigenvalues
C     VECR, VECI = array for real and imaginary elements of eigenvectors
C     S(D), T(D), U(2,D) = Only used by subroutines
C     ONLYEIGEN = 0 for only eigenvalues,
C               = non-zero for both eigenvalues and eigenvectors
C     IFAIL = keeps track if diagonalization routine crashes


      INTEGER           D
      PARAMETER (D = 8)
      DOUBLE PRECISION  AR(D,D), AI(D,D)
      DOUBLE PRECISION  PPS, SPS, SSS, P0, S0, ZPLANES
      DOUBLE PRECISION  PIE, SOC
      INTEGER           I, J, K, L, M, N, O
      DOUBLE PRECISION  TRAJECT(3,120)
      DOUBLE PRECISION  E(D), VECR(D,D), VECI(D,D)
      DOUBLE PRECISION  S(D), T(D), U(2,D)
      INTEGER           IFAIL, ONLYEIGEN



      PIE = 3.1415926535897
      ONLYEIGEN = 2
      N = 31

C     *****     Get the parameters from fort.30 file       *****
      READ(30,*) P0, S0, PPS, SPS, SSS, SOC


C     *****  We will calculate the Energies for the values *****
C     *****  of k in the trajectory that goes from         *****
C     *****  Gamma (0,0,0) to X(1,0,0) to M (1,1,0) to  *****
C     *****  to Gamma (0,0,0)                              *****

C     *****     Define the trajectory for the band structure     *****

C     ** First empty the array ***********************************

      DO 33 I=1,120
      TRAJECT(1,1) = 0.0
      TRAJECT(2,1) = 0.0
      TRAJECT(3,1) = 0.0
33    CONTINUE

C     **** We define 40 kpoints from Gamma to X, 40 kpoints between***
```

```
C     **** X to M and 40 kpoints from M to Gamma                  ****
C     **** 120 kpoints in total                                   ****
C     **** We slice the Z-axis in steps of 0.03125 (ZPALNES)      ****
C     **** which results in 32 slices                             ****
C     **** Hence Eigenvalues are placed in fort.31 until fort.64  ****

      ZPLANES = 0.0
      DO 100 O=1,33
      TRAJECT(1,1) = 0.0
      TRAJECT(2,1) = 0.0
      TRAJECT(3,1) = ZPLANES

C     **** Gamma to X *********************

      DO 10 I=1,40
      TRAJECT(1,I)=(I*1.0)/40.0
      TRAJECT(2,I)=0.
      TRAJECT(3,I)=ZPLANES
10    CONTINUE


C     ****** X to M ************************

      DO 11 I=41,80
      TRAJECT(1,I)=1
      TRAJECT(2,I)=(I*1.0-40.0)/40.0
      TRAJECT(3,I)=ZPLANES
11    CONTINUE


C     ******* M to Gamma ********************

      DO 12 I=81,120
      TRAJECT(1,I)=I*1.0-(I*1.0-1.0)-((I*1.0-80)/40.0)
      TRAJECT(2,I)=I*1.0-(I*1.0-1.0)-((I*1.0-80)/40.0)
      TRAJECT(3,I)=ZPLANES
12    CONTINUE



C     *****  an iteration for each point of the trajectory.      *****
C     ***** Start by getting the values of X, Y, Z               *****
      DO 130 I=1,120
      X = TRAJECT(1,I)
      Y = TRAJECT(2,I)
      Z = TRAJECT(3,I)



C     *****     Empty the arrays that will be used     *****
      DO 145 J=1,D
      E(J) = 0.
      S(J) = 0.
      T(J) = 0.
      U(1,J) = 0.
      U(2,J) = 0.
      DO 155 K=1,D
      AR(J,K) = 0.
      AI(J,K) = 0.
      VECR(J,K) = 0.
      VECI(J,K) = 0.
155   CONTINUE
145   CONTINUE
```

```
C     ***** The full 8 x 8 matrix.                        *****
C     ***** Here, we only provide the non-zero elements  *****

       AR(1,1)   = P0 + (2.0*PPS*COS(PIE*X))
       AI(1,2)   = -0.5*SOC
       AI(1,4)   = 2.0*SPS*SIN(PIE*X)
       AR(1,7)   = 0.5*SOC
       AR(2,2)   = P0 + (2.0*PPS*COS(PIE*Y))
       AI(2,4)   = 2.0*SPS*SIN(PIE*Y)
       AI(2,7)   = -0.5*SOC
       AR(3,3)   = P0 + (2.0*PPS*COS(PIE*Z))
       AI(3,4)   = 2.0*SPS*SIN(PIE*Z)
       AR(3,5)   = -0.5*SOC
       AI(3,6)   = 0.5*SOC
       AR(4,4)   = S0+(2.0*SSS*(COS(PIE*X)+COS(PIE*Y)+COS(PIE*Z)))
       AR(5,5)   = P0 + (2.0*PPS*COS(PIE*X))
       AI(5,6)   = 0.5*SOC
       AI(5,8)   = 2.0*SPS*SIN(PIE*X)
       AR(6,6)   = P0 + (2.0*PPS*COS(PIE*Y))
       AI(6,8)   = 2.0*SPS*SIN(PIE*Y)
       AR(7,7)   = P0 + (2.0*PPS*COS(PIE*Z))
       AI(7,8)   = 2.0*SPS*SIN(PIE*Z)
       AR(8,8)   = S0+(2.0*SSS*(COS(PIE*X)+COS(PIE*Y)+COS(PIE*Z)))
       AI(2,1)   = -1.0*AI(1,2)
       AI(4,1)   = -1.0*AI(1,4)
       AI(4,2)   = -1.0*AI(2,4)
       AI(4,3)   = -1.0*AI(3,4)
       AR(5,3)   = AR(3,5)
       AI(6,3)   = -1.0*AI(3,6)
       AI(6,5)   = -1.0*AI(5,6)
       AR(7,1)   = AR(1,7)
       AI(7,2)   = -1.0*AI(2,7)
       AI(8,5)   = -1.0*AI(5,8)
       AI(8,6)   = -1.0*AI(6,8)
       AI(8,7)   = -1.0*AI(7,8)


C     *****     Call the Diagonalization Routine     *****
      CALL DIAG(D,D,AR,AI,E,ONLYEIGEN,VECR,VECI,S,T,U,IFAIL)



C     ***** Write only                                    *****
C     ***** the 8 eigenvalues found in the file fort.N  *****
C     ***** Format: row number, 8 eigen values          *****

      WRITE(N,185) I,E(1),E(2),E(3),E(4),E(5),E(6),E(7),E(8)
185   FORMAT(I3,8F10.6)

C     *****  Write the Eigenvectors in fort.N+33 onwards *****

C      DO 245 L=1,D
C      DO 255 M=1,D
C      WRITE(N+33,186) VECR(L,M), VECI(L,M)
C255   CONTINUE
C245   CONTINUE
C186   FORMAT(2F10.6)

130   CONTINUE

      ZPLANES = ZPLANES+0.03125
      N = N+1
100   CONTINUE
```

```
      END

C#####################################################################
      SUBROUTINE DIAG(nm,n,ar,ai,w,matz,zr,zi,fv1,fv2,fm1,ierr)
      CALL ch(nm,n,ar,ai,w,matz,zr,zi,fv1,fv2,fm1,ierr)
      RETURN
      END

C**********************************************************************

      subroutine ch(nm,n,ar,ai,w,matz,zr,zi,fv1,fv2,fm1,ierr)

c**********************************************************************72
c
cc CH gets eigenvalues and eigenvectors of a complex Hermitian matrix.
c
c     this subroutine calls the recommended sequence of
c     routines from the eigensystem subroutine package (eispack)
c     to find the eigenvalues and eigenvectors (if desired)
c     of a complex hermitian matrix.
c
c     on input
c
c        nm  must be set to the row dimension of the two-dimensional
c        array parameters as declared in the calling program
c        dimension statement.
c
c        n  is the order of the matrix  a=(ar,ai).
c
c        ar  and  ai  contain the real and imaginary parts,
c        respectively, of the complex hermitian matrix.
c
c        matz  is an integer variable set equal to zero if
c        only eigenvalues are desired.  otherwise it is set to
c        any non-zero integer for both eigenvalues and eigenvectors.
c
c     on output
c
c        w  contains the eigenvalues in ascending order.
c
c        zr  and  zi  contain the real and imaginary parts,
c        respectively, of the eigenvectors if matz is not zero.
c
c        ierr  is an integer output variable set equal to an error
c           completion code described in the documentation for tqlrat
c           and tql2.  the normal completion code is zero.
c
c        fv1, fv2, and  fm1  are temporary storage arrays.
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
      integer i,j,n,nm,ierr,matz
      double precision ar(nm,n),ai(nm,n),w(n),zr(nm,n),zi(nm,n)
      double precision fv1(n),fv2(n),fm1(2,n)
c
C
C
      if (n .le. nm) go to 10
```

```
      ierr = 10 * n
      go to 50
c
   10 call  htridi(nm,n,ar,ai,w,fv1,fv2,fm1)
      if (matz .ne. 0) go to 20
c     ..... find eigenvalues only .....
      call  tqlrat(n,w,fv2,ierr)
      go to 50
c     ..... find both eigenvalues and eigenvectors .....
   20 do 40 i = 1, n
c
         do 30 j = 1, n
            zr(j,i) = 0.0d0
   30    continue
c
         zr(i,i) = 1.0d0
   40 continue
c
      call  tql2(nm,n,w,fv1,zr,ierr)
      if (ierr .ne. 0) go to 50
      call  htribk(nm,n,ar,ai,fm1,n,zr,zi)
   50 return
      end
      subroutine cinvit(nm,n,ar,ai,wr,wi,select,mm,m,zr,zi,
     x                  ierr,rm1,rm2,rv1,rv2)

c**********************************************************************72
c
cc CINVIT gets eigenvectors from eigenvalues, for a complex Hessenberg
matrix.
c
c     this subroutine is a translation of the algol procedure cx invit
c     by peters and wilkinson.
c     handbook for auto. comp. vol.ii-linear algebra, 418-439(1971).
c
c     this subroutine finds those eigenvectors of a complex upper
c     hessenberg matrix corresponding to specified eigenvalues,
c     using inverse iteration.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        ar and ai contain the real and imaginary parts,
c          respectively, of the hessenberg matrix.
c
c        wr and wi contain the real and imaginary parts, respectively,
c          of the eigenvalues of the matrix.  the eigenvalues must be
c          stored in a manner identical to that of subroutine  comlr,
c          which recognizes possible splitting of the matrix.
c
c        select specifies the eigenvectors to be found.  the
c          eigenvector corresponding to the j-th eigenvalue is
c          specified by setting select(j) to .true..
c
c        mm should be set to an upper bound for the number of
c          eigenvectors to be found.
c
c     on output
c
```

```
c        ar, ai, wi, and select are unaltered.
c
c        wr may have been altered since close eigenvalues are perturbed
c          slightly in searching for independent eigenvectors.
c
c        m is the number of eigenvectors actually found.
c
c        zr and zi contain the real and imaginary parts, respectively,
c          of the eigenvectors.  the eigenvectors are normalized
c          so that the component of largest magnitude is 1.
c          any vector which fails the acceptance test is set to zero.
c
c        ierr is set to
c          zero       for normal return,
c          -(2*n+1)   if more than mm eigenvectors have been specified,
c          -k         if the iteration corresponding to the k-th
c                     value fails,
c          -(n+k)     if both error situations occur.
c
c        rm1, rm2, rv1, and rv2 are temporary storage arrays.
c
c     the algol procedure guessvec appears in cinvit in line.
c
c     calls cdiv for complex division.
c     calls pythag for  dsqrt(a*a + b*b) .
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
      integer i,j,k,m,n,s,ii,mm,mp,nm,uk,ip1,its,km1,ierr
      double precision ar(nm,n),ai(nm,n),wr(n),wi(n),zr(nm,mm),
     x       zi(nm,mm),rm1(n,n),rm2(n,n),rv1(n),rv2(n)
      double precision x,y,eps3,norm,normv,epslon,growto,ilambd,pythag,
     x       rlambd,ukroot
      logical select(n)

      ierr = 0
      uk = 0
      s = 1
c
      do 980 k = 1, n
         if (.not. select(k)) go to 980
         if (s .gt. mm) go to 1000
         if (uk .ge. k) go to 200
c     ..... check for possible splitting .....
         do 120 uk = k, n
            if (uk .eq. n) go to 140
            if (ar(uk+1,uk) .eq. 0.0d0 .and. ai(uk+1,uk) .eq. 0.0d0)
     x         go to 140
  120    continue
c     ..... compute infinity norm of leading uk by uk
c                (hessenberg) matrix .....
  140    norm = 0.0d0
         mp = 1
c
         do 180 i = 1, uk
            x = 0.0d0
c
            do 160 j = mp, uk
  160       x = x + pythag(ar(i,j),ai(i,j))
c
            if (x .gt. norm) norm = x
```

```
            mp = i
  180    continue
c     ..... eps3 replaces zero pivot in decomposition
c                and close roots are modified by eps3 .....
         if (norm .eq. 0.0d0) norm = 1.0d0
         eps3 = epslon(norm)
c     ..... growto is the criterion for growth .....
         ukroot = uk
         ukroot = dsqrt(ukroot)
         growto = 0.1d0 / ukroot
  200    rlambd = wr(k)
         ilambd = wi(k)
         if (k .eq. 1) go to 280
         km1 = k - 1
         go to 240
c     ..... perturb eigenvalue if it is close
c                to any previous eigenvalue .....
  220    rlambd = rlambd + eps3
c     ..... for i=k-1 step -1 until 1 do -- .....
  240    do 260 ii = 1, km1
            i = k - ii
            if (select(i) .and. dabs(wr(i)-rlambd) .lt. eps3 .and.
     x         dabs(wi(i)-ilambd) .lt. eps3) go to 220
  260    continue
c
         wr(k) = rlambd
c     ..... form upper hessenberg (ar,ai)-(rlambd,ilambd)*i
c                and initial complex vector .....
  280    mp = 1
c
         do 320 i = 1, uk
c
            do 300 j = mp, uk
               rm1(i,j) = ar(i,j)
               rm2(i,j) = ai(i,j)
  300       continue
c
            rm1(i,i) = rm1(i,i) - rlambd
            rm2(i,i) = rm2(i,i) - ilambd
            mp = i
            rv1(i) = eps3
  320    continue
c     ..... triangular decomposition with interchanges,
c                replacing zero pivots by eps3 .....
         if (uk .eq. 1) go to 420
c
         do 400 i = 2, uk
            mp = i - 1
            if (pythag(rm1(i,mp),rm2(i,mp)) .le.
     x          pythag(rm1(mp,mp),rm2(mp,mp))) go to 360
c
            do 340 j = mp, uk
               y = rm1(i,j)
               rm1(i,j) = rm1(mp,j)
               rm1(mp,j) = y
               y = rm2(i,j)
               rm2(i,j) = rm2(mp,j)
               rm2(mp,j) = y
  340       continue
c
  360       if (rm1(mp,mp) .eq. 0.0d0 .and. rm2(mp,mp) .eq. 0.0d0)
     x         rm1(mp,mp) = eps3
            call cdiv(rm1(i,mp),rm2(i,mp),rm1(mp,mp),rm2(mp,mp),x,y)
            if (x .eq. 0.0d0 .and. y .eq. 0.0d0) go to 400
```

```
c
           do 380 j = i, uk
              rm1(i,j) = rm1(i,j) - x * rm1(mp,j) + y * rm2(mp,j)
              rm2(i,j) = rm2(i,j) - x * rm2(mp,j) - y * rm1(mp,j)
  380      continue
c
  400   continue
c
  420   if (rm1(uk,uk) .eq. 0.0d0 .and. rm2(uk,uk) .eq. 0.0d0)
     x     rm1(uk,uk) = eps3
        its = 0
c     ..... back substitution
c                for i=uk step -1 until 1 do -- .....
  660   do 720 ii = 1, uk
           i = uk + 1 - ii
           x = rv1(i)
           y = 0.0d0
           if (i .eq. uk) go to 700
           ip1 = i + 1
c
           do 680 j = ip1, uk
              x = x - rm1(i,j) * rv1(j) + rm2(i,j) * rv2(j)
              y = y - rm1(i,j) * rv2(j) - rm2(i,j) * rv1(j)
  680      continue
c
  700      call cdiv(x,y,rm1(i,i),rm2(i,i),rv1(i),rv2(i))
  720   continue
c     ..... acceptance test for eigenvector
c                and normalization .....
        its = its + 1
        norm = 0.0d0
        normv = 0.0d0
c
        do 780 i = 1, uk
           x = pythag(rv1(i),rv2(i))
           if (normv .ge. x) go to 760
           normv = x
           j = i
  760      norm = norm + x
  780   continue
c
        if (norm .lt. growto) go to 840
c     ..... accept vector .....
        x = rv1(j)
        y = rv2(j)
c
        do 820 i = 1, uk
           call cdiv(rv1(i),rv2(i),x,y,zr(i,s),zi(i,s))
  820   continue
c
        if (uk .eq. n) go to 940
        j = uk + 1
        go to 900
c     ..... in-line procedure for choosing
c                a new starting vector .....
  840   if (its .ge. uk) go to 880
        x = ukroot
        y = eps3 / (x + 1.0d0)
        rv1(1) = eps3
c
        do 860 i = 2, uk
  860   rv1(i) = y
c
        j = uk - its + 1
```

```
         rv1(j) = rv1(j) - eps3 * x
         go to 660
c     ..... set error -- unaccepted eigenvector .....
  880    j = 1
         ierr = -k
c     ..... set remaining vector components to zero .....
  900    do 920 i = j, n
            zr(i,s) = 0.0d0
            zi(i,s) = 0.0d0
  920    continue
c
  940    s = s + 1
  980 continue
c
      go to 1001
c     ..... set error -- underestimate of eigenvector
c                space required .....
 1000 if (ierr .ne. 0) ierr = ierr - n
      if (ierr .eq. 0) ierr = -(2 * n + 1)
 1001 m = s - 1
      return
      end
C
C
C
C*********************************************************************
      subroutine htridi(nm,n,ar,ai,d,e,e2,tau)
C
c*********************************************************************72
c
cc HTRIDI tridiagonalizes a complex hermitian matrix.
c
c     this subroutine is a translation of a complex analogue of
c     the algol procedure tred1, num. math. 11, 181-195(1968)
c     by martin, reinsch, and wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 212-226(1971).
c
c     this subroutine reduces a complex hermitian matrix
c     to a real symmetric tridiagonal matrix using
c     unitary similarity transformations.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        ar and ai contain the real and imaginary parts,
c          respectively, of the complex hermitian input matrix.
c          only the lower triangle of the matrix need be supplied.
c
c     on output
c
c        ar and ai contain information about the unitary trans-
c          formations used in the reduction in their full lower
c          triangles.  their strict upper triangles and the
c          diagonal of ar are unaltered.
c
c        d contains the diagonal elements of the the tridiagonal matrix.
c
c        e contains the subdiagonal elements of the tridiagonal
c          matrix in its last n-1 positions.  e(1) is set to zero.
```

```
c
c        e2 contains the squares of the corresponding elements of e.
c          e2 may coincide with e if the squares are not needed.
c
c        tau contains further information about the transformations.
c
c     calls pythag for  dsqrt(a*a + b*b) .
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
      integer i,j,k,l,n,ii,nm,jp1
      double precision ar(nm,n),ai(nm,n),d(n),e(n),e2(n),tau(2,n)
      double precision f,g,h,fi,gi,hh,si,scale,pythag

      tau(1,n) = 1.0d0
      tau(2,n) = 0.0d0
c
      do 100 i = 1, n
  100 d(i) = ar(i,i)
c     .......... for i=n step -1 until 1 do -- ..........
      do 300 ii = 1, n
         i = n + 1 - ii
         l = i - 1
         h = 0.0d0
         scale = 0.0d0
         if (l .lt. 1) go to 130
c     .......... scale row (algol tol then not needed) ..........
         do 120 k = 1, l
  120    scale = scale + dabs(ar(i,k)) + dabs(ai(i,k))
c
         if (scale .ne. 0.0d0) go to 140
         tau(1,l) = 1.0d0
         tau(2,l) = 0.0d0
  130    e(i) = 0.0d0
         e2(i) = 0.0d0
         go to 290
c
  140    do 150 k = 1, l
            ar(i,k) = ar(i,k) / scale
            ai(i,k) = ai(i,k) / scale
            h = h + ar(i,k) * ar(i,k) + ai(i,k) * ai(i,k)
  150    continue
c
         e2(i) = scale * scale * h
         g = dsqrt(h)
         e(i) = scale * g
         f = pythag(ar(i,l),ai(i,l))
c     .......... form next diagonal element of matrix t ..........
         if (f .eq. 0.0d0) go to 160
         tau(1,l) = (ai(i,l) * tau(2,i) - ar(i,l) * tau(1,i)) / f
         si = (ar(i,l) * tau(2,i) + ai(i,l) * tau(1,i)) / f
         h = h + f * g
         g = 1.0d0 + g / f
         ar(i,l) = g * ar(i,l)
         ai(i,l) = g * ai(i,l)
         if (l .eq. 1) go to 270
         go to 170
  160    tau(1,l) = -tau(1,i)
         si = tau(2,i)
         ar(i,l) = g
  170    f = 0.0d0
```

```
c
         do 240 j = 1, l
            g = 0.0d0
            gi = 0.0d0
c     ..... form element of a*u .....
            do 180 k = 1, j
               g = g + ar(j,k) * ar(i,k) + ai(j,k) * ai(i,k)
               gi = gi - ar(j,k) * ai(i,k) + ai(j,k) * ar(i,k)
  180       continue
c
            jp1 = j + 1
            if (l .lt. jp1) go to 220
c
            do 200 k = jp1, l
               g = g + ar(k,j) * ar(i,k) - ai(k,j) * ai(i,k)
               gi = gi - ar(k,j) * ai(i,k) - ai(k,j) * ar(i,k)
  200       continue
c     ..... form element of p .....
  220       e(j) = g / h
            tau(2,j) = gi / h
            f = f + e(j) * ar(i,j) - tau(2,j) * ai(i,j)
  240    continue
c
         hh = f / (h + h)
c     ..... form reduced a .....
         do 260 j = 1, l
            f = ar(i,j)
            g = e(j) - hh * f
            e(j) = g
            fi = -ai(i,j)
            gi = tau(2,j) - hh * fi
            tau(2,j) = -gi
c
            do 260 k = 1, j
               ar(j,k) = ar(j,k) - f * e(k) - g * ar(i,k)
     x                           + fi * tau(2,k) + gi * ai(i,k)
               ai(j,k) = ai(j,k) - f * tau(2,k) - g * ai(i,k)
     x                           - fi * e(k) - gi * ar(i,k)
  260    continue
c
  270    do 280 k = 1, l
            ar(i,k) = scale * ar(i,k)
            ai(i,k) = scale * ai(i,k)
  280    continue
c
         tau(2,l) = -si
  290    hh = d(i)
         d(i) = ar(i,i)
         ar(i,i) = hh
         ai(i,i) = scale * dsqrt(h)
  300 continue
c
      return
      end
C
C
C
C*********************************************************************
      subroutine tqlrat(n,d,e2,ierr)
C
c**********************************************************************72
c
cc TQLRAT computes all eigenvalues of a real symmetric tridiagonal matrix.
c
```

```
C     This subroutine is a translation of the Algol procedure tqlrat,
C     Algorithm 464, Comm. ACM 16, 689(1973) by Reinsch.
C
C     This subroutine finds the eigenvalues of a symmetric
C     tridiagonal matrix by the rational QL method.
C
C     On input
C
C        N is the order of the matrix.
C
C        D contains the diagonal elements of the input matrix.
C
C        E2 contains the squares of the subdiagonal elements of the
C          input matrix in its last N-1 positions.  E2(1) is arbitrary.
C
C      On output
C
C        D contains the eigenvalues in ascending order.  If an
C          error exit is made, the eigenvalues are correct and
C          ordered for indices 1,2,...IERR-1, but may not be
C          the smallest eigenvalues.
C
C        E2 has been destroyed.
C
C        IERR is set to
C          zero       for normal return,
C          J          if the J-th eigenvalue has not been
C                     determined after 30 iterations.
C
C     Calls PYTHAG for  DSQRT(A*A + B*B) .
C
C     Questions and comments should be directed to Burton S. Garbow,
C     Mathematics and Computer Science Div, Argonne National Laboratory
C
C     This version dated August 1987.
C     Modified by C. Moler to fix underflow/overflow difficulties,
C     especially on the VAX and other machines where epslon(1.0d0)**2
C     nearly underflows.  See the loop involving statement 102 and
C     the two statements just before statement 200.
C
      integer i,j,l,m,n,ii,l1,mml,ierr
      double precision d(n),e2(n)
      double precision b,c,f,g,h,p,r,s,t,epslon,pythag

      ierr = 0
      if (n .eq. 1) go to 1001
c
      do 100 i = 2, n
  100 e2(i-1) = e2(i)
c
      f = 0.0d0
      t = 0.0d0
      e2(n) = 0.0d0
c
      do 290 l = 1, n
         j = 0
         h = dabs(d(l)) + dsqrt(e2(l))
         if (t .gt. h) go to 105
         t = h
         b = epslon(t)
         c = b * b
         if (c .ne. 0.0d0) go to 105
c        spliting tolerance underflowed.  look for larger value.
         do 102 i = l, n
```

```
            h = dabs(d(i)) + dsqrt(e2(i))
            if (h .gt. t) t = h
  102    continue
         b = epslon(t)
         c = b * b
c     .......... look for small squared sub-diagonal element ..........
  105    do 110 m = l, n
            if (e2(m) .le. c) go to 120
c     .......... e2(n) is always zero, so there is no exit
c                through the bottom of the loop ..........
  110    continue
c
  120    if (m .eq. l) go to 210
  130    if (j .eq. 30) go to 1000
         j = j + 1
c     .......... form shift ..........
         l1 = l + 1
         s = dsqrt(e2(l))
         g = d(l)
         p = (d(l1) - g) / (2.0d0 * s)
         r = pythag(p,1.0d0)
         d(l) = s / (p + dsign(r,p))
         h = g - d(l)
c
         do 140 i = l1, n
  140    d(i) = d(i) - h
c
         f = f + h
c     .......... rational ql transformation ..........
         g = d(m)
         if (g .eq. 0.0d0) g = b
         h = g
         s = 0.0d0
         mml = m - l
c     .......... for i=m-1 step -1 until l do -- ..........
         do 200 ii = 1, mml
            i = m - ii
            p = g * h
            r = p + e2(i)
            e2(i+1) = s * r
            s = e2(i) / r
            d(i+1) = h + s * (h + d(i))
            g = d(i) - e2(i) / g
c           avoid division by zero on next pass
            if (g .eq. 0.0d0) g = epslon(d(i))
            h = g * (p / r)
  200    continue
c
         e2(l) = s * g
         d(l) = h
c     .......... guard against underflow in convergence test ..........
         if (h .eq. 0.0d0) go to 210
         if (dabs(e2(l)) .le. dabs(c/h)) go to 210
         e2(l) = h * e2(l)
         if (e2(l) .ne. 0.0d0) go to 130
  210    p = d(l) + f
c     .......... order eigenvalues ..........
         if (l .eq. 1) go to 250
c     .......... for i=l step -1 until 2 do -- ..........
         do 230 ii = 2, l
            i = l + 2 - ii
            if (p .ge. d(i-1)) go to 270
            d(i) = d(i-1)
  230    continue
```

```
c
  250    i = 1
  270    d(i) = p
  290 continue
c
      go to 1001
c     ......... set error -- no convergence to an
c               eigenvalue after 30 iterations .........
 1000 ierr = l
 1001 return
      end
C
C
C
C*********************************************************************
      subroutine tql2(nm,n,d,e,z,ierr)
C
c*********************************************************************72
c
cc TQL2 computes all eigenvalues/vectors, real symmetric tridiagonal
matrix.
c
c     this subroutine is a translation of the algol procedure tql2,
c     num. math. 11, 293-306(1968) by bowdler, martin, reinsch, and
c     wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 227-240(1971).
c
c     this subroutine finds the eigenvalues and eigenvectors
c     of a symmetric tridiagonal matrix by the ql method.
c     the eigenvectors of a full symmetric matrix can also
c     be found if  tred2  has been used to reduce this
c     full matrix to tridiagonal form.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        d contains the diagonal elements of the input matrix.
c
c        e contains the subdiagonal elements of the input matrix
c          in its last n-1 positions.  e(1) is arbitrary.
c
c        z contains the transformation matrix produced in the
c          reduction by  tred2, if performed.  if the eigenvectors
c          of the tridiagonal matrix are desired, z must contain
c          the identity matrix.
c
c      on output
c
c        d contains the eigenvalues in ascending order.  if an
c          error exit is made, the eigenvalues are correct but
c          unordered for indices 1,2,...,ierr-1.
c
c        e has been destroyed.
c
c        z contains orthonormal eigenvectors of the symmetric
c          tridiagonal (or full) matrix.  if an error exit is made,
c          z contains the eigenvectors associated with the stored
c          eigenvalues.
c
```

```
c        ierr is set to
c          zero       for normal return,
c          j          if the j-th eigenvalue has not been
c                     determined after 30 iterations.
c
c     calls pythag for  dsqrt(a*a + b*b) .
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
      integer i,j,k,l,m,n,ii,l1,l2,nm,mml,ierr
      double precision d(n),e(n),z(nm,n)
      double precision c,c2,c3,dl1,el1,f,g,h,p,r,s,s2,tst1,tst2,pythag

      ierr = 0
      if (n .eq. 1) go to 1001
c
      do 100 i = 2, n
  100 e(i-1) = e(i)
c
      f = 0.0d0
      tst1 = 0.0d0
      e(n) = 0.0d0
c
      do 240 l = 1, n
         j = 0
         h = dabs(d(l)) + dabs(e(l))
         if (tst1 .lt. h) tst1 = h
c     ..... look for small sub-diagonal element .....
         do 110 m = l, n
            tst2 = tst1 + dabs(e(m))
            if (tst2 .eq. tst1) go to 120
c     ..... e(n) is always zero, so there is no exit
c                through the bottom of the loop .....
  110    continue
c
  120    if (m .eq. l) go to 220
  130    if (j .eq. 30) go to 1000
         j = j + 1
c     ..... form shift .....
         l1 = l + 1
         l2 = l1 + 1
         g = d(l)
         p = (d(l1) - g) / (2.0d0 * e(l))
         r = pythag(p,1.0d0)
         d(l) = e(l) / (p + dsign(r,p))
         d(l1) = e(l) * (p + dsign(r,p))
         dl1 = d(l1)
         h = g - d(l)
         if (l2 .gt. n) go to 145
c
         do 140 i = l2, n
  140    d(i) = d(i) - h
c
  145    f = f + h
c     ..... ql transformation .....
         p = d(m)
         c = 1.0d0
         c2 = c
         el1 = e(l1)
         s = 0.0d0
         mml = m - l
```

```
c     .......... for i=m-1 step -1 until l do -- ..........
         do 200 ii = 1, mml
            c3 = c2
            c2 = c
            s2 = s
            i = m - ii
            g = c * e(i)
            h = c * p
            r = pythag(p,e(i))
            e(i+1) = s * r
            s = e(i) / r
            c = p / r
            p = c * d(i) - s * g
            d(i+1) = h + s * (c * g + s * d(i))
c     .......... form vector ..........
            do 180 k = 1, n
               h = z(k,i+1)
               z(k,i+1) = s * z(k,i) + c * h
               z(k,i) = c * z(k,i) - s * h
  180       continue
c
  200    continue
c
         p = -s * s2 * c3 * el1 * e(l) / dl1
         e(l) = s * p
         d(l) = c * p
         tst2 = tst1 + dabs(e(l))
         if (tst2 .gt. tst1) go to 130
  220    d(l) = d(l) + f
  240 continue

c     .......... order eigenvalues and eigenvectors ..........
      do 300 ii = 2, n
         i = ii - 1
         k = i
         p = d(i)
c
         do 260 j = ii, n
            if (d(j) .ge. p) go to 260
            k = j
            p = d(j)
  260    continue
c
         if (k .eq. i) go to 300
         d(k) = d(i)
         d(i) = p
c
         do 280 j = 1, n
            p = z(j,i)
            z(j,i) = z(j,k)
            z(j,k) = p
  280    continue
c
  300 continue
c
      go to 1001
c     .......... set error -- no convergence to an
c                eigenvalue after 30 iterations ..........
 1000 ierr = l
 1001 return
      end
C
C
C
```

```
C*********************************************
      subroutine cdiv(ar,ai,br,bi,cr,ci)
C
c**********************************************************************72
c
cc CDIV emulates complex division, using real arithmetic.
c
c     complex division, (cr,ci) = (ar,ai)/(br,bi)
c
      double precision ar,ai,br,bi,cr,ci
      double precision s,ars,ais,brs,bis

      s = dabs(br) + dabs(bi)
      ars = ar/s
      ais = ai/s
      brs = br/s
      bis = bi/s
      s = brs**2 + bis**2
      cr = (ars*brs + ais*bis)/s
      ci = (ais*brs - ars*bis)/s
      return
      end
C
C
C
C*****************************************************
      double precision function epslon (x)
C
c**********************************************************************72
c
cc EPSLON estimate unit roundoff in quantities of size X.
c
c     this program should function properly on all systems
c     satisfying the following two assumptions,
c        1.  the base used in representing floating point
c            numbers is not a power of three.
c        2.  the quantity  a  in statement 10 is represented to
c            the accuracy used in floating point variables
c            that are stored in memory.
c     the statement number 10 and the go to 10 are intended to
c     force optimizing compilers to generate code satisfying
c     assumption 2.
c     under these assumptions, it should be true that,
c            a  is not exactly equal to four-thirds,
c            b  has a zero for its last bit or digit,
c            c  is not exactly equal to one,
c            eps  measures the separation of 1.0 from
c                 the next larger floating point number.
c     the developers of eispack would appreciate being informed
c     about any systems where these assumptions do not hold.
c
c     this version dated 4/6/83.
c
      double precision a,b,c,eps
      double precision x

      a = 4.0d0/3.0d0
   10 b = a - 1.0d0
      c = b + b + b
      eps = dabs(c-1.0d0)
      if (eps .eq. 0.0d0) go to 10
      epslon = eps*dabs(x)
      return
      end
```

```
C
C
C
C***********************************************************************
      subroutine htribk(nm,n,ar,ai,tau,m,zr,zi)
C
c***********************************************************************72
c
cc HTRIBK determines eigenvectors by undoing the HTRIDI transformation.
c
c     this subroutine is a translation of a complex analogue of
c     the algol procedure trbak1, num. math. 11, 181-195(1968)
c     by martin, reinsch, and wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 212-226(1971).
c
c     this subroutine forms the eigenvectors of a complex hermitian
c     matrix by back transforming those of the corresponding
c     real symmetric tridiagonal matrix determined by  htridi.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        ar and ai contain information about the unitary trans-
c          formations used in the reduction by  htridi  in their
c          full lower triangles except for the diagonal of ar.
c
c        tau contains further information about the transformations.
c
c        m is the number of eigenvectors to be back transformed.
c
c        zr contains the eigenvectors to be back transformed
c          in its first m columns.
c
c     on output
c
c        zr and zi contain the real and imaginary parts,
c          respectively, of the transformed eigenvectors
c          in their first m columns.
c
c     note that the last component of each returned vector
c     is real and that vector euclidean norms are preserved.
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
      integer i,j,k,l,m,n,nm
      double precision ar(nm,n),ai(nm,n),tau(2,n),zr(nm,m),zi(nm,m)
      double precision h,s,si

      if (m .eq. 0) go to 200
c     ..... transform the eigenvectors of the real symmetric
c                tridiagonal matrix to those of the hermitian
c                tridiagonal matrix. .....
      do 50 k = 1, n
c
         do 50 j = 1, m
            zi(k,j) = -zr(k,j) * tau(2,k)
```

```
            zr(k,j) = zr(k,j) * tau(1,k)
   50 continue
c
      if (n .eq. 1) go to 200
c     .......... recover and apply the householder matrices ..........
      do 140 i = 2, n
         l = i - 1
         h = ai(i,i)
         if (h .eq. 0.0d0) go to 140
c
         do 130 j = 1, m
            s = 0.0d0
            si = 0.0d0
c
            do 110 k = 1, l
               s = s + ar(i,k) * zr(k,j) - ai(i,k) * zi(k,j)
               si = si + ar(i,k) * zi(k,j) + ai(i,k) * zr(k,j)
  110       continue
c     .......... double divisions avoid possible underflow ..........
            s = (s / h) / h
            si = (si / h) / h
c
            do 120 k = 1, l
               zr(k,j) = zr(k,j) - s * ar(i,k) - si * ai(i,k)
               zi(k,j) = zi(k,j) - si * ar(i,k) + s * ai(i,k)
  120       continue
c
  130    continue
c
  140 continue
c
  200 return
      end
C
C
C
C*************************
      double precision function pythag(a,b)
C
c*********************************************************************72
c
cc PYTHAG computes SQRT ( A**2 + B**2 ) carefully.
c
c     finds dsqrt(a**2+b**2) without overflow or destructive underflow
c
      double precision a,b
      double precision p,r,s,t,u

      p = dmax1(dabs(a),dabs(b))
      if (p .eq. 0.0d0) go to 20
      r = (dmin1(dabs(a),dabs(b))/p)**2
   10 continue
         t = 4.0d0 + r
         if (t .eq. 4.0d0) go to 20
         s = r/t
         u = 1.0d0 + 2.0d0*s
         p = u*p
         r = (s/u)**2 * r
      go to 10
   20 pythag = p
      return
      end
```