December 5, 2014
0:01

Contents

1 beigen

Old faithful Jacobi diagonisation routine. This version is modified to diagonalise a "blocked" matrix. The matrix is assumed to be *nblock* sub-matrices along the principal diagonal of dimension *iblock*(1), *iblock*(2) , ... *iblock*(*nblock*) with zeroes off these blocks. Of course, if *nblock* = 1 then *iblock*(1) = *n*.

NAME

beigen

SYNOPSIS

```
subroutine beigen(H,U,n,init,nblock,iblock);
implicit double precision (a-h,o-z);
double precision H(*), U(*);
integer n, init, nblock, iblock(*);
```

DESCRIPTION

Diagonalises a symmetric matrix which consists of *nblock* symmetric matrices along the diagonal of H. The eigenvalues and eigenvectors are returned in order of ascending eigenvalue.

ARGUMENTS

H Input/Outpu:t Matrix to be diagonalised. On output, the eigenvalues are on the main diagonal of H.

U Input/Output: If *init* is non-zero U contains a guess at the eigenvector matrix, if not it is initialised to the unit matrix. Ouput is the eigenvectors.

n Input: The size of the matrices overall,

init Input: zero if U is a sensible starting-point for the process, non-zero if not.

nblock Input: The number of sub-matrices in H.

iblock Input: *iblock*(1) ... *iblock*(*nblock*) are the dimensions of the sub-matrices.

DIAGNOSTICS

None, possibility of infinite loop but unlikely.

"beigen.f" 1 ≡

```
subroutine beigen(H, U, n, init, nblock, iblock)
⟨ Interface declarations 1.1 ⟩
data zero, eps, one, two, four/0.0 · 10^+00D, 1.0 · 10^-20D, 1.0 · 10^+00D, 2.0 · 10^+00D, 4.0 · 10^+00D/
⟨ Initialize U 1.2 ⟩
nmax = 0
do nsym = 1, nblock     /* Loop over the nsym sub-matrices */
    /* nmin and nmax are the limits of the current sub-matrix */
  nmin = nmax + 1
  nmax = nmax + iblock(nsym)
    /* Start sweep through off-diagonal elements; the sweep is repeated until the largest off-diagonal
       element of H is less than eps */
  do while(hmax > eps)
    ⟨ Reduce current off-diagonal to zero 1.3 ⟩
  end do
end do     /* End of nsym loop on blocks */
⟨ Sort the eigenvalues and vectors 1.7 ⟩
return
END
```

⟨ Interface declarations 1.1 ⟩ ≡

```
implicit double precision (a − h, o − z)
double precision H(∗), U(∗)
integer n, init, nblock, iblock(∗)
```

This code is used in section 1.

This simply sets U to the unit matrix. It is used if *init* is zero. If *init* is not zero, the incoming U is assumed to be a sensible starting-point for the calculation.

```
⟨ Initialize U 1.2 ⟩ ≡
    if (init ≡ 0) then
      do i = 1, n
        ii = n ∗ (i − 1) + i
        do j = 1, n
          ij = n ∗ (j − 1) + i
          U(ij) = zero
        end do
        U(ii) = one
      end do
    end if
```

This code is used in section 1.

This is the central algorithm which calclautes and uses the "angle" which reduces the current off-diagonal element of H to zero. The effect on the other off-diagonals is computed and the largest off-diagonal element saved for convergence testing

```
⟨ Reduce current off-diagonal to zero 1.3 ⟩ ≡
    hmax = zero    /∗ hmax keeps track of the largest off-diagonal element of H ∗/
    do i = nmin + 1, nmax
      jtop = i − 1
      do j = nmin, jtop
        ⟨ Calculate Rotation Angle 1.4 ⟩
        ⟨ Apply Rotation to U 1.5 ⟩
        ⟨ Apply Rotation to H 1.6 ⟩
      end do
    end do
```

This code is used in section 1.

tan is $tan(2\theta)$ where θ is the rotation angle which makes $H(ij)$ vanish. c and s are $cos\theta$ and $sin\theta$ obtained by the usual "half-angle formula" from $tan(2\theta)$

⟨ Calculate Rotation Angle 1.4 ⟩ ≡

```
    /* positions of matrix elements */
  ii = n * (i - 1) + i;
  jj = n * (j - 1) + j
  ij = n * (j - 1) + i;
  ji = n * (i - 1) + j
  hii = H(ii);
  hjj = H(jj);
  hij = H(ij);
  hsq = hij * hij
  if (hsq > hmax)
    hmax = hsq
  if (hsq < eps)
    cycle    /* omit zero H(ij) */
  del = hii - hjj
  sign = one
  if (del < zero) then
    sign = -one
    del = -del
  end if
  denom = del + dsqrt(del * del + four * hsq)

  tan = two * sign * hij / denom
  c = one / dsqrt(one + tan * tan)
  s = c * tan
```

This code is used in section 1.3.

⟨ Apply Rotation to U 1.5 ⟩ ≡

```
do k = 1, n
  kj = n * (j - 1) + k
  ki = n * (i - 1) + k
  jk = n * (k - 1) + j
  ik = n * (k - 1) + i
  temp = c * U(kj) - s * U(ki)
  U(ki) = s * U(kj) + c * U(ki)
  U(kj) = temp
    /* If k is niether i or j then apply the current rotation */
  if ((i ≡ k) | (j ≡ k))
    cycle
  temp = c * H(kj) - s * H(ki)
  H(ki) = s * H(kj) + c * H(ki)
  H(kj) = temp
  H(ik) = H(ki)
  H(jk) = H(kj)
end do
    /* This does not make any off-diagonal element zero; in fact it will, in general, re-generate ones
       which have been zeroized in other cycles */
```

This code is used in section 1.3.

⟨ Apply Rotation to H 1.6 ⟩ ≡

```
H(ii) = c * c * hii + s * s * hjj + two * c * s * hij
H(jj) = c * c * hjj + s * s * hii - two * c * s * hij
H(ij) = zero
H(ji) = zero
    /* This is the key step; it generates one zero off-diagonal element */
```

This code is used in section 1.3.

Now Sort the eigenvalues and eigenvectors into ascending order. OVERALL; i.e. not within each block.

If it is required to sort the eigenvalues and vectors into ascending order *within* each block then this coding must be changed. For example, one may wish to occupy the lowest orbitals of each of several symmetry types to generate specific states of the molecule.

⟨ Sort the eigenvalues and vectors 1.7 ⟩ ≡

```
nmax = 0
do nsym = 1, nblock
  nmin = nmax + 1
  nmax = nmax + iblock(nsym)
  call epsort(H, U, n, nmin, nmax)
end do
```

This code is used in section 1.

2 epsort

Sort eigenvectors from n1 to n2 into eigenvalue order

```
"beigen.f" 2 ≡
    subroutine epsort(H, U, n, n1, n2)
      implicit double precision (a − h, o − z)
      double precision H(∗), U(∗)
      integer n, n1, n2

      double precision temp
      integer iq, jq, ii, jj, k, ilr, imr

      iq = (n1 − 2) ∗ n
      do i = n1, n2
        iq = iq + n
        ii = (i − 1) ∗ n + i
        jq = n ∗ (i − 2)
        do j = i, n2
          jq = jq + n
          jj = (j − 1) ∗ n + j
          if (H(ii) < H(jj))
            cycle     /∗ this means H(1) is lowest! ∗/
          temp = H(ii)
          H(ii) = H(jj)
          H(jj) = temp
          do k = 1, n
            ilr = iq + k
            imr = jq + k
            temp = U(ilr)
            U(ilr) = U(imr)
            U(imr) = temp;
          end do
        end do
      end do
      return
    end
```

3 INDEX

⟨ Apply Rotation to H 1.6 ⟩ Used in section 1.3.
⟨ Apply Rotation to U 1.5 ⟩ Used in section 1.3.
⟨ Calculate Rotation Angle 1.4 ⟩ Used in section 1.3.
⟨ Initialize U 1.2 ⟩ Used in section 1.
⟨ Interface declarations 1.1 ⟩ Used in section 1.
⟨ Reduce current off-diagonal to zero 1.3 ⟩ Used in section 1.
⟨ Sort the eigenvalues and vectors 1.7 ⟩ Used in section 1.

COMMAND LINE: `"fweave beigen.web"`.
WEB FILE: `"beigen.web"`.
CHANGE FILE: `(none)`.
GLOBAL LANGUAGE: FORTRAN.