NUMERICAL RECIPES
Webnote No. 17, Rev. 1

Implementations in Unsymmeig

The routine hqr finds eigenvalues, but not eigenvectors.

eigen_unsym.h

```
void Unsymmeig::hqr()
```

Finds all eigenvalues of an upper Hessenberg matrix a[0..n-1][0..n-1]. On input a can be exactly as output from elmhes §11.6; on output it is destroyed. The complex eigenvalues are returned in wri[0..n-1].

```
{
    Int nn,m,l,k,j,its,i,mmin;
    Doub z,y,x,w,v,u,t,s,r,q,p,anorm=0.0;

    const Doub EPS=numeric_limits<Doub>::epsilon();
    for (i=0;i<n;i++)                         Compute matrix norm for possible use in lo-
        for (j=MAX(i-1,0);j<n;j++)                cating single small subdiagonal element.
            anorm += abs(a[i][j]);
    nn=n-1;
    t=0.0;                                    Gets changed only by an exceptional shift.
    while (nn >= 0) {                         Begin search for next eigenvalue.
        its=0;
        do {
            for (l=nn;l>0;l--) {              Begin iteration: look for single small subdi-
                s=abs(a[l-1][l-1])+abs(a[l][l]);          agonal element.
                if (s == 0.0) s=anorm;
                if (abs(a[l][l-1]) <= EPS*s) {
                    a[l][l-1] = 0.0;
                    break;
                }
            }
            x=a[nn][nn];
            if (l == nn) {                    One root found.
                wri[nn--]=x+t;
            } else {
                y=a[nn-1][nn-1];
                w=a[nn][nn-1]*a[nn-1][nn];
                if (l == nn-1) {              Two roots found...
                    p=0.5*(y-x);
                    q=p*p+w;
                    z=sqrt(abs(q));
                    x += t;
                    if (q >= 0.0) {           ...a real pair.
                        z=p+SIGN(z,p);
                        wri[nn-1]=wri[nn]=x+z;
                        if (z != 0.0) wri[nn]=x-w/z;
                    } else {                  ...a complex pair.
                        wri[nn]=Complex(x+p,-z);
                        wri[nn-1]=conj(wri[nn]);
                    }
                    nn -= 2;
                } else {                      No roots found. Continue iteration.
                    if (its == 30) throw("Too many iterations in hqr");
if (its == 10 || its == 20) {          Form exceptional shift.
    t += x;
    for (i=0;i<nn+1;i++) a[i][i] -= x;
    s=abs(a[nn][nn-1])+abs(a[nn-1][nn-2]);
    y=x=0.75*s;
    w = -0.4375*s*s;
}
++its;
for (m=nn-2;m>=1;m--) {                Form shift and then look for
    z=a[m][m];                             2 consecutive small sub-
    r=x-z;                                 diagonal elements.
    s=y-z;
    p=(r*s-w)/a[m+1][m]+a[m][m+1];     Equation (Webnote 16.21).
    q=a[m+1][m+1]-z-r-s;
    r=a[m+2][m+1];
    s=abs(p)+abs(q)+abs(r);            Scale to prevent overflow or
    p /= s;                                underflow.
    q /= s;
    r /= s;
    if (m == l) break;
    u=abs(a[m][m-1])*(abs(q)+abs(r));
    v=abs(p)*(abs(a[m-1][m-1])+abs(z)+abs(a[m+1][m+1]));
    if (u <= EPS*v) break;             Equation (Webnote 16.24).
}
for (i=m;i<nn-1;i++) {
    a[i+2][i]=0.0;
    if (i != m) a[i+2][i-1]=0.0;
}
for (k=m;k<nn;k++) {
Double QR step on rows l to nn and columns m to nn.
    if (k != m) {
        p=a[k][k-1];                   Begin setup of Householder
        q=a[k+1][k-1];                     vector.
        r=0.0;
        if (k+1 != nn) r=a[k+2][k-1];
        if ((x=abs(p)+abs(q)+abs(r)) != 0.0) {
            p /= x;                    Scale to prevent overflow or
            q /= x;                        underflow.
            r /= x;
        }
    }
    if ((s=SIGN(sqrt(p*p+q*q+r*r),p)) != 0.0) {
        if (k == m) {
            if (l != m)
            a[k][k-1] = -a[k][k-1];
        } else
            a[k][k-1] = -s*x;
        p += s;                        Equations (Webnote 16.22).
        x=p/s;
        y=q/s;
        z=r/s;
        q /= p;
        r /= p;
        for (j=k;j<nn+1;j++) {         Row modification.
            p=a[k][j]+q*a[k+1][j];
            if (k+1 != nn) {
                p += r*a[k+2][j];
                a[k+2][j] -= p*z;
            }
            a[k+1][j] -= p*y;
            a[k][j] -= p*x;
        }
        mmin = nn < k+3 ? nn : k+3;
        for (i=l;i<mmin+1;i++) {       Column modification.
```

```
                                    p=x*a[i][k]+y*a[i][k+1];
                                    if (k+1 != nn) {
                                        p += z*a[i][k+2];
                                        a[i][k+2] -= p*r;
                                    }
                                    a[i][k+1] -= p*q;
                                    a[i][k] -= p;
                                }
                            }
                        }
                    }
                }
            } while (l+1 < nn);
        }
    }
```

To find the eigenvectors as well as the eigenvalues, the routine needs to keep track of the similarity transformations and then apply them to the eigenvector matrix in the usual way. Note that the routine uses complex division. If your compiler doesn't do a good job of this, you may have to write your own routine for it as described in §5.5.

eigen_unsym.h

```
void Unsymmeig::hqr2()
```

Finds all eigenvalues of an upper Hessenberg matrix `a[0..n-1][0..n-1]`. On input `a` can be exactly as output from `elmhes` and `eltran` §11.6. On output, `wri[0..n-1]` contains the eigenvalues of `a`, while `zz[0..n-1][0..n-1]` is a matrix whose columns contain the corresponding eigenvectors. The eigenvalues are not sorted, except that complex conjugate pairs appear consecutively with the eigenvalue having the positive imaginary part first. For a complex eigenvalue, only the eigenvector corresponding to the eigenvalue with positive imaginary part is stored, with real part in `zz[0..n-1][i]` and imaginary part in `h.zz[0..n-1][i+1]`. The eigenvectors are not normalized.

```
{
    Int nn,m,l,k,j,its,i,mmin,na;
    Doub z,y,x,w,v,u,t,s,r,q,p,anorm=0.0,ra,sa,vr,vi;

    const Doub EPS=numeric_limits<Doub>::epsilon();
    for (i=0;i<n;i++)                           Compute matrix norm for possible use in lo-
        for (j=MAX(i-1,0);j<n;j++)                   cating single small subdiagonal element.
            anorm += abs(a[i][j]);
    nn=n-1;
    t=0.0;                                      Gets changed only by an exceptional shift.
    while (nn >= 0) {                           Begin search for next eigenvalue.
        its=0;
        do {
            for (l=nn;l>0;l--) {                Begin iteration: look for single small subdi-
                s=abs(a[l-1][l-1])+abs(a[l][l]);           agonal element.
                if (s == 0.0) s=anorm;
                if (abs(a[l][l-1]) <= EPS*s) {
                    a[l][l-1] = 0.0;
                    break;
                }
            }
            x=a[nn][nn];
            if (l == nn) {                      One root found.
                wri[nn]=a[nn][nn]=x+t;
                nn--;
            } else {
                y=a[nn-1][nn-1];
                w=a[nn][nn-1]*a[nn-1][nn];
                if (l == nn-1) {                Two roots found...
                    p=0.5*(y-x);
                    q=p*p+w;
            z=sqrt(abs(q));
            x += t;
            a[nn][nn]=x;
            a[nn-1][nn-1]=y+t;
            if (q >= 0.0) {           ...a real pair.
                z=p+SIGN(z,p);
                wri[nn-1]=wri[nn]=x+z;
                if (z != 0.0) wri[nn]=x-w/z;
                x=a[nn][nn-1];
                s=abs(x)+abs(z);
                p=x/s;
                q=z/s;
                r=sqrt(p*p+q*q);
                p /= r;
                q /= r;
                for (j=nn-1;j<n;j++) {      Row modification.
                    z=a[nn-1][j];
                    a[nn-1][j]=q*z+p*a[nn][j];
                    a[nn][j]=q*a[nn][j]-p*z;
                }
                for (i=0;i<=nn;i++) {       Column modification.
                    z=a[i][nn-1];
                    a[i][nn-1]=q*z+p*a[i][nn];
                    a[i][nn]=q*a[i][nn]-p*z;
                }
                for (i=0;i<n;i++) {         Accumulate transformations.
                    z=zz[i][nn-1];
                    zz[i][nn-1]=q*z+p*zz[i][nn];
                    zz[i][nn]=q*zz[i][nn]-p*z;
                }
            } else {                  ...a complex pair.
                wri[nn]=Complex(x+p,-z);
                wri[nn-1]=conj(wri[nn]);
            }
            nn -= 2;
        } else {                      No roots found.  Continue iteration.
            if (its == 30) throw("Too many iterations in hqr");
            if (its == 10 || its == 20) {          Form exceptional shift.
                t += x;
                for (i=0;i<nn+1;i++) a[i][i] -= x;
                s=abs(a[nn][nn-1])+abs(a[nn-1][nn-2]);
                y=x=0.75*s;
                w = -0.4375*s*s;
            }
            ++its;
            for (m=nn-2;m>=1;m--) {                Form shift and then look for
                z=a[m][m];                             2 consecutive small sub-
                r=x-z;                                 diagonal elements.
                s=y-z;
                p=(r*s-w)/a[m+1][m]+a[m][m+1];     Equation (Webnote 16.21).
                q=a[m+1][m+1]-z-r-s;
                r=a[m+2][m+1];
                s=abs(p)+abs(q)+abs(r);            Scale to prevent overflow or
                p /= s;                                underflow.
                q /= s;
                r /= s;
                if (m == 1) break;
                u=abs(a[m][m-1])*(abs(q)+abs(r));
                v=abs(p)*(abs(a[m-1][m-1])+abs(z)+abs(a[m+1][m+1]));
                if (u <= EPS*v) break;             Equation (Webnote 16.24).
            }
            for (i=m;i<nn-1;i++) {
                a[i+2][i]=0.0;
                if (i != m) a[i+2][i-1]=0.0;
                }
                for (k=m;k<nn;k++) {
```
Double QR step on rows `l` to `nn` and columns `m` to `nn`.
```
                    if (k != m) {
                        p=a[k][k-1];                    Begin setup of Householder
                        q=a[k+1][k-1];                      vector.
                        r=0.0;
                        if (k+1 != nn) r=a[k+2][k-1];
                        if ((x=abs(p)+abs(q)+abs(r)) != 0.0) {
                            p /= x;                     Scale to prevent overflow or
                            q /= x;                         underflow.
                            r /= x;
                        }
                    }
                    if ((s=SIGN(sqrt(p*p+q*q+r*r),p)) != 0.0) {
                        if (k == m) {
                            if (l != m)
                            a[k][k-1] = -a[k][k-1];
                        } else
                            a[k][k-1] = -s*x;
                        p += s;                         Equations (Webnote 16.22).
                        x=p/s;
                        y=q/s;
                        z=r/s;
                        q /= p;
                        r /= p;
                        for (j=k;j<n;j++) {             Row modification.
                            p=a[k][j]+q*a[k+1][j];
                            if (k+1 != nn) {
                                p += r*a[k+2][j];
                                a[k+2][j] -= p*z;
                            }
                            a[k+1][j] -= p*y;
                            a[k][j] -= p*x;
                        }
                        mmin = nn < k+3 ? nn : k+3;
                        for (i=0;i<mmin+1;i++) {        Column modification.
                            p=x*a[i][k]+y*a[i][k+1];
                            if (k+1 != nn) {
                                p += z*a[i][k+2];
                                a[i][k+2] -= p*r;
                            }
                            a[i][k+1] -= p*q;
                            a[i][k] -= p;
                        }
                        for (i=0; i<n; i++) {           Accumulate transformations.
                            p=x*zz[i][k]+y*zz[i][k+1];
                            if (k+1 != nn) {
                                p += z*zz[i][k+2];
                                zz[i][k+2] -= p*r;
                            }
                            zz[i][k+1] -= p*q;
                            zz[i][k] -= p;
                        }
                    }
                }
            }
        }
    } while (l+1 < nn);
}
```
All roots found. Backsubstitute to find vectors of upper triangular form.
```
if (anorm != 0.0) {
    for (nn=n-1;nn>=0;nn--) {
        p=real(wri[nn]);
        q=imag(wri[nn]);
        na=nn-1;
        if (q == 0.0) {                         Real vector.
            m=nn;
            a[nn][nn]=1.0;
            for (i=nn-1;i>=0;i--) {
                w=a[i][i]-p;
                r=0.0;
                for (j=m;j<=nn;j++)
                    r += a[i][j]*a[j][nn];
                if (imag(wri[i]) < 0.0) {
                    z=w;
                    s=r;
                } else {
                    m=i;

                    if (imag(wri[i]) == 0.0) {
                        t=w;
                        if (t == 0.0)
                            t=EPS*anorm;
                        a[i][nn]=-r/t;
                    } else {                    Solve real equations.
                        x=a[i][i+1];
                        y=a[i+1][i];
                        q=SQR(real(wri[i])-p)+SQR(imag(wri[i]));
                        t=(x*s-z*r)/q;
                        a[i][nn]=t;
                        if (abs(x) > abs(z))
                            a[i+1][nn]=(-r-w*t)/x;
                        else
                            a[i+1][nn]=(-s-y*t)/z;
                    }
                    t=abs(a[i][nn]);            Overflow control.
                    if (EPS*t*t > 1)
                        for (j=i;j<=nn;j++)
                            a[j][nn] /= t;
                }
            }
        } else if (q < 0.0) {                   Complex vector, only do one case.
            m=na;                               Last vector component chosen imag-
            if (abs(a[nn][na]) > abs(a[na][nn])) {     inary so that eigenvec-
                a[na][na]=q/a[nn][na];                 tor matrix is triangular.
                a[na][nn]=-(a[nn][nn]-p)/a[nn][na];
            } else {
                Complex temp=Complex(0.0,-a[na][nn])/Complex(a[na][na]-p,q);
                a[na][na]=real(temp);
                a[na][nn]=imag(temp);
            }
            a[nn][na]=0.0;
            a[nn][nn]=1.0;
            for (i=nn-2;i>=0;i--) {
                w=a[i][i]-p;
                ra=sa=0.0;
                for (j=m;j<=nn;j++) {
                    ra += a[i][j]*a[j][na];
                    sa += a[i][j]*a[j][nn];
                }
                if (imag(wri[i]) < 0.0) {
                    z=w;
                    r=ra;
                    s=sa;
                } else {
                    m=i;
                    if (imag(wri[i]) == 0.0) {
                        Complex temp = Complex(-ra,-sa)/Complex(w,q);
                        a[i][na]=real(temp);
                        a[i][nn]=imag(temp);
                    } else {                    Solve complex equations.
                        x=a[i][i+1];
                        y=a[i+1][i];
                        vr=SQR(real(wri[i])-p)+SQR(imag(wri[i]))-q*q;
                        vi=2.0*q*(real(wri[i])-p);
                        if (vr == 0.0 && vi == 0.0)
                            vr=EPS*anorm*(abs(w)+abs(q)+abs(x)+abs(y)+abs(z));
                        Complex temp=Complex(x*r-z*ra+q*sa,x*s-z*sa-q*ra)/
                            Complex(vr,vi);
                        a[i][na]=real(temp);
                        a[i][nn]=imag(temp);
                        if (abs(x) > abs(z)+abs(q)) {
                            a[i+1][na]=(-ra-w*a[i][na]+q*a[i][nn])/x;
                            a[i+1][nn]=(-sa-w*a[i][nn]-q*a[i][na])/x;
                        } else {
                            Complex temp=Complex(-r-y*a[i][na],-s-y*a[i][nn])/
                                Complex(z,q);
                            a[i+1][na]=real(temp);
                            a[i+1][nn]=imag(temp);
                        }
                    }
                }
                t=MAX(abs(a[i][na]),abs(a[i][nn]));  Overflow control.
                if (EPS*t*t > 1)
                    for (j=i;j<=nn;j++) {
                        a[j][na] /= t;
                        a[j][nn] /= t;
                    }
            }
        }
    }
    for (j=n-1;j>=0;j--)                        Multiply by transformation matrix to
        for (i=0;i<n;i++) {                         give vectors of original full ma-
            z=0.0;                                  trix.
            for (k=0;k<=j;k++)
                z += zz[i][k]*a[k][j];
            zz[i][j]=z;
        }
    }
}
```

Here are the routines used by `Unsymmeig` to sort the eigenvalues and eigenvectors:

eigen_unsym.h

```
void Unsymmeig::sort()
Sort the eigenvalues in descending order of their real parts using straight insertion.
{
    Int i;
    for (Int j=1;j<n;j++) {
        Complex x=wri[j];
        for (i=j-1;i>=0;i--) {
            if (real(wri[i]) >= real(x)) break;
            wri[i+1]=wri[i];
        }
        wri[i+1]=x;
    }
}
void Unsymmeig::sortvecs()
Sort the eigenvalues in descending order of their real parts using straight insertion, and simultaneously rearrange the eigenvectors.
```

```
{
    Int i;
    VecDoub temp(n);
    for (Int j=1;j<n;j++) {
        Complex x=wri[j];
        for (Int k=0;k<n;k++)
            temp[k]=zz[k][j];
        for (i=j-1;i>=0;i--) {
            if (real(wri[i]) >= real(x)) break;
            wri[i+1]=wri[i];
            for (Int k=0;k<n;k++)
                zz[k][i+1]=zz[k][i];
        }
        wri[i+1]=x;
        for (Int k=0;k<n;k++)
            zz[k][i+1]=temp[k];
    }
}
```

CITED REFERENCES AND FURTHER READING:

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer).

Golub, G.H., and Van Loan, C.F. 1996, *Matrix Computations*, 3rd ed. (Baltimore: Johns Hopkins University Press), §7.5.

Smith, B.T., et al. 1976, *Matrix Eigensystem Routines — EISPACK Guide*, 2nd ed., vol. 6 of Lecture Notes in Computer Science (New York: Springer).