```
            }
        }
        if (i != m) {                       Interchange rows and columns.
            for (j=m-1;j<=n;j++) SWAP(a[i][j],a[m][j])
            for (j=1;j<=n;j++) SWAP(a[j][i],a[j][m])
        }
        if (x) {                            Carry out the elimination.
            for (i=m+1;i<=n;i++) {
                if ((y=a[i][m-1]) != 0.0) {
                    y /= x;
                    a[i][m-1]=y;
                    for (j=m;j<=n;j++)
                        a[i][j] -= y*a[m][j];
                    for (j=1;j<=n;j++)
                        a[j][m] += y*a[j][i];
                }
            }
        }
    }
}
```

CITED REFERENCES AND FURTHER READING:

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag). [1]

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

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §6.5.4. [3]

11.6 The QR Algorithm for Real Hessenberg Matrices

Recall the following relations for the QR algorithm with shifts:

$$\mathbf{Q}_s \cdot (\mathbf{A}_s - k_s\mathbf{1}) = \mathbf{R}_s \tag{11.6.1}$$

where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular, and

$$\begin{aligned}\mathbf{A}_{s+1} &= \mathbf{R}_s \cdot \mathbf{Q}_s^T + k_s\mathbf{1}\\ &= \mathbf{Q}_s \cdot \mathbf{A}_s \cdot \mathbf{Q}_s^T\end{aligned} \tag{11.6.2}$$

The QR transformation preserves the upper Hessenberg form of the original matrix $\mathbf{A} \equiv \mathbf{A}_1$, and the workload on such a matrix is $O(n^2)$ per iteration as opposed to $O(n^3)$ on a general matrix. As $s \to \infty$, $\mathbf{A}_s$ converges to a form where the eigenvalues are either isolated on the diagonal or are eigenvalues of a 2×2 submatrix on the diagonal.

As we pointed out in §11.3, shifting is essential for rapid convergence. A key difference here is that a nonsymmetric real matrix can have complex eigenvalues.

This means that good choices for the shifts k_s may be complex, apparently necessitating complex arithmetic.

Complex arithmetic can be avoided, however, by a clever trick. The trick depends on a result analogous to the lemma we used for implicit shifts in §11.3. The lemma we need here states that if **B** is a nonsingular matrix such that

$$\mathbf{B} \cdot \mathbf{Q} = \mathbf{Q} \cdot \mathbf{H} \tag{11.6.3}$$

where **Q** is orthogonal and **H** is upper Hessenberg, then **Q** and **H** are fully determined by the first column of **Q**. (The determination is unique if **H** has positive subdiagonal elements.) The lemma can be proved by induction analogously to the proof given for tridiagonal matrices in §11.3.

The lemma is used in practice by taking two steps of the QR algorithm, either with two real shifts k_s and k_{s+1}, or with complex conjugate values k_s and $k_{s+1} = k_s*$. This gives a real matrix $\mathbf{A}_{s+2}$, where

$$\mathbf{A}_{s+2} = \mathbf{Q}_{s+1} \cdot \mathbf{Q}_s \cdot \mathbf{A}_s \cdot \mathbf{Q}_s^T \cdot \mathbf{Q}_{s+1}^T \cdot \tag{11.6.4}$$

The **Q**'s are determined by

$$\mathbf{A}_s - k_s \mathbf{1} = \mathbf{Q}_s^T \cdot \mathbf{R}_s \tag{11.6.5}$$

$$\mathbf{A}_{s+1} = \mathbf{Q}_s \cdot \mathbf{A}_s \cdot \mathbf{Q}_s^T \tag{11.6.6}$$

$$\mathbf{A}_{s+1} - k_{s+1} \mathbf{1} = \mathbf{Q}_{s+1}^T \cdot \mathbf{R}_{s+1} \tag{11.6.7}$$

Using (11.6.6), equation (11.6.7) can be rewritten

$$\mathbf{A}_s - k_{s+1} \mathbf{1} = \mathbf{Q}_s^T \cdot \mathbf{Q}_{s+1}^T \cdot \mathbf{R}_{s+1} \cdot \mathbf{Q}_s \tag{11.6.8}$$

Hence, if we define

$$\mathbf{M} = (\mathbf{A}_s - k_{s+1} \mathbf{1}) \cdot (\mathbf{A}_s - k_s \mathbf{1}) \tag{11.6.9}$$

equations (11.6.5) and (11.6.8) give

$$\mathbf{R} = \mathbf{Q} \cdot \mathbf{M} \tag{11.6.10}$$

where

$$\mathbf{Q} = \mathbf{Q}_{s+1} \cdot \mathbf{Q}_s \tag{11.6.11}$$

$$\mathbf{R} = \mathbf{R}_{s+1} \cdot \mathbf{R}_s \tag{11.6.12}$$

Equation (11.6.4) can be rewritten

$$\mathbf{A}_s \cdot \mathbf{Q}^T = \mathbf{Q}^T \cdot \mathbf{A}_{s+2} \tag{11.6.13}$$

Thus suppose we can somehow find an upper Hessenberg matrix **H** such that

$$\mathbf{A}_s \cdot \overline{\mathbf{Q}}^T = \overline{\mathbf{Q}}^T \cdot \mathbf{H} \tag{11.6.14}$$

where $\overline{\mathbf{Q}}$ is orthogonal. If $\overline{\mathbf{Q}}^T$ has the same first column as $\mathbf{Q}^T$ (i.e., $\overline{\mathbf{Q}}$ has the same first row as **Q**), then $\overline{\mathbf{Q}} = \mathbf{Q}$ and $\mathbf{A}_{s+2} = \mathbf{H}$.

The first row of $\mathbf{Q}$ is found as follows. Equation (11.6.10) shows that $\mathbf{Q}$ is the orthogonal matrix that triangularizes the real matrix $\mathbf{M}$. Any real matrix can be triangularized by premultiplying it by a sequence of Householder matrices $\mathbf{P}_1$ (acting on the first column), $\mathbf{P}_2$ (acting on the second column), $\dots$, $\mathbf{P}_{n-1}$. Thus $\mathbf{Q} = \mathbf{P}_{n-1}\cdots\mathbf{P}_2\cdot\mathbf{P}_1$, and the first row of $\mathbf{Q}$ is the first row of $\mathbf{P}_1$ since $\mathbf{P}_i$ is an $(i-1)\times(i-1)$ identity matrix in the top left-hand corner. We now must find $\overline{\mathbf{Q}}$ satisfying (11.6.14) whose first row is that of $\mathbf{P}_1$.

The Householder matrix $\mathbf{P}_1$ is determined by the first column of $\mathbf{M}$. Since $\mathbf{A}_s$ is upper Hessenberg, equation (11.6.9) shows that the first column of $\mathbf{M}$ has the form $[p_1, q_1, r_1, 0, ..., 0]^T$, where

$$\begin{aligned} p_1 &= a_{11}^2 - a_{11}(k_s + k_{s+1}) + k_s k_{s+1} + a_{12}a_{21} \\ q_1 &= a_{21}(a_{11} + a_{22} - k_s - k_{s+1}) \\ r_1 &= a_{21}a_{32} \end{aligned} \tag{11.6.15}$$

Hence

$$\mathbf{P}_1 = 1 - 2\mathbf{w}_1\cdot\mathbf{w}_1^T \tag{11.6.16}$$

where $\mathbf{w}_1$ has only its first 3 elements nonzero (cf. equation 11.2.5). The matrix $\mathbf{P}_1\cdot\mathbf{A}_s\cdot\mathbf{P}_1^T$ is therefore upper Hessenberg with 3 extra elements:

$$\mathbf{P}_1\cdot\mathbf{A}_1\cdot\mathbf{P}_1^T = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \mathbf{x} & \times & \times & \times & \times & \times & \times \\ \mathbf{x} & \mathbf{x} & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{bmatrix} \tag{11.6.17}$$

This matrix can be restored to upper Hessenberg form without affecting the first row by a sequence of Householder similarity transformations. The first such Householder matrix, $\mathbf{P}_2$, acts on elements 2, 3, and 4 in the first column, annihilating elements 3 and 4. This produces a matrix of the same form as (11.6.17), with the 3 extra elements appearing one column over:

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ & \mathbf{x} & \times & \times & \times & \times & \times \\ & \mathbf{x} & \mathbf{x} & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{bmatrix} \tag{11.6.18}$$

Proceeding in this way up to $\mathbf{P}_{n-1}$, we see that at each stage the Householder matrix $\mathbf{P}_r$ has a vector $\mathbf{w}_r$ that is nonzero only in elements r, $r+1$, and $r+2$. These elements are determined by the elements r, $r+1$, and $r+2$ in the $(r-1)$st column of the current matrix. Note that the preliminary matrix $\mathbf{P}_1$ has the same structure as $\mathbf{P}_2, \ldots, \mathbf{P}_{n-1}$.

The result is that

$$\mathbf{P}_{n-1} \cdots \mathbf{P}_2 \cdot \mathbf{P}_1 \cdot \mathbf{A}_s \cdot \mathbf{P}_1^T \cdot \mathbf{P}_2^T \cdots \mathbf{P}_{n-1}^T = \mathbf{H} \tag{11.6.19}$$

where $\mathbf{H}$ is upper Hessenberg. Thus

$$\overline{\mathbf{Q}} = \mathbf{Q} = \mathbf{P}_{n-1} \cdots \mathbf{P}_2 \cdot \mathbf{P}_1 \tag{11.6.20}$$

and

$$\mathbf{A}_{s+2} = \mathbf{H} \tag{11.6.21}$$

The shifts of origin at each stage are taken to be the eigenvalues of the 2×2 matrix in the bottom right-hand corner of the current $\mathbf{A}_s$. This gives

$$\begin{aligned} k_s + k_{s+2} &= a_{n-1,n-1} + a_{nn} \\ k_s k_{s+1} &= a_{n-1,n-1} a_{nn} - a_{n-1,n} a_{n,n-1} \end{aligned} \tag{11.6.22}$$

Substituting (11.6.22) in (11.6.15), we get

$$\begin{aligned} p_1 &= a_{21} \left\{ [(a_{nn} - a_{11})(a_{n-1,n-1} - a_{11}) - a_{n-1,n} a_{n,n-1}]/a_{21} + a_{12} \right\} \\ q_1 &= a_{21} [a_{22} - a_{11} - (a_{nn} - a_{11}) - (a_{n-1,n-1} - a_{11})] \\ r_1 &= a_{21} a_{32} \end{aligned} \tag{11.6.23}$$

We have judiciously grouped terms to reduce possible roundoff when there are small off-diagonal elements. Since only the ratios of elements are relevant for a Householder transformation, we can omit the factor a_{21} from (11.6.23).

In summary, to carry out a double QR step we construct the Householder matrices $\mathbf{P}_r,\ r = 1, \ldots, n-1$. For $\mathbf{P}_1$ we use p_1, q_1, and r_1 given by (11.6.23). For the remaining matrices, p_r, q_r, and r_r are determined by the $(r, r-1)$, $(r+1, r-1)$, and $(r+2, r-1)$ elements of the current matrix. The number of arithmetic operations can be reduced by writing the nonzero elements of the $2\mathbf{w} \cdot \mathbf{w}^T$ part of the Householder matrix in the form

$$2\mathbf{w} \cdot \mathbf{w}^T = \begin{bmatrix} (p \pm s)/(\pm s) \\ q/(\pm s) \\ r/(\pm s) \end{bmatrix} \cdot [\,1 \quad q/(p \pm s) \quad r/(p \pm s)\,] \tag{11.6.24}$$

where

$$s^2 = p^2 + q^2 + r^2 \tag{11.6.25}$$

(We have simply divided each element by a piece of the normalizing factor; cf. the equations in §11.2.)

If we proceed in this way, convergence is usually very fast. There are two possible ways of terminating the iteration for an eigenvalue. First, if $a_{n,n-1}$ becomes "negligible," then a_{nn} is an eigenvalue. We can then delete the nth row and column of the matrix and look for the next eigenvalue. Alternatively, $a_{n-1,n-2}$ may become negligible. In this case the eigenvalues of the 2×2 matrix in the lower right-hand corner may be taken to be eigenvalues. We delete the nth and $(n-1)$st rows and columns of the matrix and continue.

The test for convergence to an eigenvalue is combined with a test for negligible subdiagonal elements that allows splitting of the matrix into submatrices. We find the largest i such that $a_{i,i-1}$ is negligible. If $i = n$, we have found a single eigenvalue. If $i = n - 1$, we have found two eigenvalues. Otherwise we continue the iteration on the submatrix in rows i to n (i being set to unity if there is no small subdiagonal element).

After determining i, the submatrix in rows i to n is examined to see if the *product* of any two consecutive subdiagonal elements is small enough that we can work with an even smaller submatrix, starting say in row m. We start with $m = n - 2$ and decrement it down to $i + 1$, computing p, q, and r according to equations (11.6.23) with 1 replaced by m and 2 by $m + 1$. If these were indeed the elements of the special "first" Householder matrix in a double QR step, then applying the Householder matrix would lead to nonzero elements in positions $(m+1, m-1)$, $(m+2, m-1)$, and $(m+2, m)$. We require that the first two of these elements be small compared with the local diagonal elements $a_{m-1,m-1}$, a_{mm} and $a_{m+1,m+1}$. A satisfactory approximate criterion is

$$|a_{m,m-1}|(|q| + |r|) \ll |p|(|a_{m+1,m+1}| + |a_{mm}| + |a_{m-1,m-1}|) \qquad (11.6.26)$$

Very rarely, the procedure described so far will fail to converge. On such matrices, experience shows that if one double step is performed with any shifts that are of order the norm of the matrix, convergence is subsequently very rapid. Accordingly, if ten iterations occur without determining an eigenvalue, the usual shifts are replaced for the next iteration by shifts defined by

$$\begin{aligned} k_s + k_{s+1} &= 1.5 \times (|a_{n,n-1}| + |a_{n-1,n-2}|) \\ k_s k_{s+1} &= (|a_{n,n-1}| + |a_{n-1,n-2}|)^2 \end{aligned} \qquad (11.6.27)$$

The factor 1.5 was arbitrarily chosen to lessen the likelihood of an "unfortunate" choice of shifts. This strategy is repeated after 20 unsuccessful iterations. After 30 unsuccessful iterations, the routine reports failure.

The operation count for the QR algorithm described here is $\sim 5k^2$ per iteration, where k is the current size of the matrix. The typical average number of iterations per eigenvalue is ~ 1.8, so the total operation count for all the eigenvalues is $\sim 3n^3$. This estimate neglects any possible efficiency due to splitting or sparseness of the matrix.

The following routine `hqr` is based algorithmically on the above description, in turn following the implementations in [1,2].

```
#include <math.h>
#include "nrutil.h"

void hqr(float **a, int n, float wr[], float wi[])
```
Finds all eigenvalues of an upper Hessenberg matrix a[1..n][1..n]. On input a can be exactly as output from elmhes §11.5; on output it is destroyed. The real and imaginary parts of the eigenvalues are returned in wr[1..n] and wi[1..n], respectively.
```
{
    int nn,m,l,k,j,its,i,mmin;
    float z,y,x,w,v,u,t,s,r,q,p,anorm;

    anorm=0.0;                          Compute matrix norm for possible use in lo-
    for (i=1;i<=n;i++)                     cating single small subdiagonal element.
        for (j=IMAX(i-1,1);j<=n;j++)
            anorm += fabs(a[i][j]);
    nn=n;
    t=0.0;                              Gets changed only by an exceptional shift.
    while (nn >= 1) {                   Begin search for next eigenvalue.
        its=0;
        do {
            for (l=nn;l>=2;l--) {       Begin iteration: look for single small subdi-
                s=fabs(a[l-1][l-1])+fabs(a[l][l]);       agonal element.
                if (s == 0.0) s=anorm;
                if ((float)(fabs(a[l][l-1]) + s) == s) break;
            }
            x=a[nn][nn];
            if (l == nn) {              One root found.
                wr[nn]=x+t;
                wi[nn--]=0.0;
            } 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(fabs(q));
                    x += t;
                    if (q >= 0.0) {     ...a real pair.
                        z=p+SIGN(z,p);
                        wr[nn-1]=wr[nn]=x+z;
                        if (z) wr[nn]=x-w/z;
                        wi[nn-1]=wi[nn]=0.0;
                    } else {            ...a complex pair.
                        wr[nn-1]=wr[nn]=x+p;
                        wi[nn-1]= -(wi[nn]=z);
                    }
                    nn -= 2;
                } else {                No roots found. Continue iteration.
                    if (its == 30) nrerror("Too many iterations in hqr");
                    if (its == 10 || its == 20) {        Form exceptional shift.
                        t += x;
                        for (i=1;i<=nn;i++) a[i][i] -= x;
                        s=fabs(a[nn][nn-1])+fabs(a[nn-1][nn-2]);
                        y=x=0.75*s;
                        w = -0.4375*s*s;
                    }
                    ++its;
                    for (m=(nn-2);m>=l;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 (11.6.23).
                        q=a[m+1][m+1]-z-r-s;
                        r=a[m+2][m+1];
```

```
                s=fabs(p)+fabs(q)+fabs(r);        Scale to prevent overflow or
                p /= s;                                underflow.
                q /= s;
                r /= s;
                if (m == l) break;
                u=fabs(a[m][m-1])*(fabs(q)+fabs(r));
                v=fabs(p)*(fabs(a[m-1][m-1])+fabs(z)+fabs(a[m+1][m+1]));
                if ((float)(u+v) == v) break;     Equation (11.6.26).
            }
            for (i=m+2;i<=nn;i++) {
                a[i][i-2]=0.0;
                if (i != (m+2)) a[i][i-3]=0.0;
            }
            for (k=m;k<=nn-1;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 != (nn-1)) r=a[k+2][k-1];
                    if ((x=fabs(p)+fabs(q)+fabs(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 (11.6.24).
                    x=p/s;
                    y=q/s;
                    z=r/s;
                    q /= p;
                    r /= p;
                    for (j=k;j<=nn;j++) {         Row modification.
                        p=a[k][j]+q*a[k+1][j];
                        if (k != (nn-1)) {
                            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;i++) {       Column modification.
                        p=x*a[i][k]+y*a[i][k+1];
                        if (k != (nn-1)) {
                            p += z*a[i][k+2];
                            a[i][k+2] -= p*r;
                        }
                        a[i][k+1] -= p*q;
                        a[i][k] -= p;
                    }
                }
            }
        }
    }
    } while (l < nn-1);
  }
}
```

CITED REFERENCES AND FURTHER READING:

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag). [1]

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd 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-Verlag). [2]

11.7 Improving Eigenvalues and/or Finding Eigenvectors by Inverse Iteration

The basic idea behind inverse iteration is quite simple. Let $\mathbf{y}$ be the solution of the linear system

$$(\mathbf{A} - \tau\mathbf{1}) \cdot \mathbf{y} = \mathbf{b} \tag{11.7.1}$$

where $\mathbf{b}$ is a random vector and τ is close to some eigenvalue λ of $\mathbf{A}$. Then the solution $\mathbf{y}$ will be close to the eigenvector corresponding to λ. The procedure can be iterated: Replace $\mathbf{b}$ by $\mathbf{y}$ and solve for a new $\mathbf{y}$, which will be even closer to the true eigenvector.

We can see why this works by expanding both $\mathbf{y}$ and $\mathbf{b}$ as linear combinations of the eigenvectors $\mathbf{x}_j$ of $\mathbf{A}$:

$$\mathbf{y} = \sum_j \alpha_j \mathbf{x}_j \qquad \mathbf{b} = \sum_j \beta_j \mathbf{x}_j \tag{11.7.2}$$

Then (11.7.1) gives

$$\sum_j \alpha_j(\lambda_j - \tau)\mathbf{x}_j = \sum_j \beta_j \mathbf{x}_j \tag{11.7.3}$$

so that

$$\alpha_j = \frac{\beta_j}{\lambda_j - \tau} \tag{11.7.4}$$

and

$$\mathbf{y} = \sum_j \frac{\beta_j \mathbf{x}_j}{\lambda_j - \tau} \tag{11.7.5}$$

If τ is close to λ_n, say, then provided β_n is not accidentally too small, $\mathbf{y}$ will be approximately $\mathbf{x}_n$, up to a normalization. Moreover, each iteration of this procedure gives another power of $\lambda_j - \tau$ in the denominator of (11.7.5). Thus the convergence is rapid for well-separated eigenvalues.

Suppose at the kth stage of iteration we are solving the equation

$$(\mathbf{A} - \tau_k\mathbf{1}) \cdot \mathbf{y} = \mathbf{b}_k \tag{11.7.6}$$