```
                f += e[j]*a[i][j];
            }
            hh=f/(h+h);                     Form K, equation (11.2.11).
            for (j=1;j<=l;j++) {            Form q and store in e overwriting p.
                f=a[i][j];
                e[j]=g=e[j]-hh*f;
                for (k=1;k<=j;k++)          Reduce a, equation (11.2.13).
                    a[j][k] -= (f*e[k]+g*a[i][k]);
            }
        }
    } else
        e[i]=a[i][l];
    d[i]=h;
}
/* Next statement can be omitted if eigenvectors not wanted */
d[1]=0.0;
e[1]=0.0;
/* Contents of this loop can be omitted if eigenvectors not
        wanted except for statement d[i]=a[i][i]; */
for (i=1;i<=n;i++) {                        Begin accumulation of transformation ma-
    l=i-1;                                      trices.
    if (d[i]) {                             This block skipped when i=1.
        for (j=1;j<=l;j++) {
            g=0.0;
            for (k=1;k<=l;k++)              Use u and u/H stored in a to form P·Q.
                g += a[i][k]*a[k][j];
            for (k=1;k<=l;k++)
                a[k][j] -= g*a[k][i];
        }
    }
    d[i]=a[i][i];                           This statement remains.
    a[i][i]=1.0;                            Reset row and column of a to identity
    for (j=1;j<=l;j++) a[j][i]=a[i][j]=0.0;     matrix for next iteration.
}
}
```

CITED REFERENCES AND FURTHER READING:

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §5.1. [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).

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag). [2]

11.3 Eigenvalues and Eigenvectors of a Tridiagonal Matrix

Evaluation of the Characteristic Polynomial

Once our original, real, symmetric matrix has been reduced to tridiagonal form, one possible way to determine its eigenvalues is to find the roots of the characteristic polynomial $p_n(\lambda)$ directly. The characteristic polynomial of a tridiagonal matrix can be evaluated for any trial value of λ by an efficient recursion relation (see [1], for example). The polynomials of lower degree produced during the recurrence form a

Sturmian sequence that can be used to localize the eigenvalues to intervals on the real axis. A root-finding method such as bisection or Newton's method can then be employed to refine the intervals. The corresponding eigenvectors can then be found by inverse iteration (see §11.7).

Procedures based on these ideas can be found in [2,3]. If, however, more than a small fraction of all the eigenvalues and eigenvectors are required, then the factorization method next considered is much more efficient.

The QR and QL Algorithms

The basic idea behind the QR algorithm is that any real matrix can be decomposed in the form

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R} \tag{11.3.1}$$

where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular. For a general matrix, the decomposition is constructed by applying Householder transformations to annihilate successive columns of $\mathbf{A}$ below the diagonal (see §2.10).

Now consider the matrix formed by writing the factors in (11.3.1) in the opposite order:

$$\mathbf{A}' = \mathbf{R} \cdot \mathbf{Q} \tag{11.3.2}$$

Since $\mathbf{Q}$ is orthogonal, equation (11.3.1) gives $\mathbf{R} = \mathbf{Q}^T \cdot \mathbf{A}$. Thus equation (11.3.2) becomes

$$\mathbf{A}' = \mathbf{Q}^T \cdot \mathbf{A} \cdot \mathbf{Q} \tag{11.3.3}$$

We see that $\mathbf{A}'$ is an orthogonal transformation of $\mathbf{A}$.

You can verify that a QR transformation preserves the following properties of a matrix: symmetry, tridiagonal form, and Hessenberg form (to be defined in §11.5).

There is nothing special about choosing one of the factors of $\mathbf{A}$ to be upper triangular; one could equally well make it lower triangular. This is called the QL algorithm, since

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{L} \tag{11.3.4}$$

where $\mathbf{L}$ is lower triangular. (The standard, but confusing, nomenclature R and L stands for whether the *right* or *left* of the matrix is nonzero.)

Recall that in the Householder reduction to tridiagonal form in §11.2, we started in the nth (last) column of the original matrix. To minimize roundoff, we then exhorted you to put the biggest elements of the matrix in the lower right-hand corner, if you can. If we now wish to diagonalize the resulting tridiagonal matrix, the QL algorithm will have smaller roundoff than the QR algorithm, so we shall use QL henceforth.

The QL algorithm consists of a *sequence* of orthogonal transformations:

$$\begin{aligned}\mathbf{A}_s &= \mathbf{Q}_s \cdot \mathbf{L}_s \\ \mathbf{A}_{s+1} &= \mathbf{L}_s \cdot \mathbf{Q}_s \qquad (= \mathbf{Q}_s^T \cdot \mathbf{A}_s \cdot \mathbf{Q}_s)\end{aligned} \tag{11.3.5}$$

The following (nonobvious!) theorem is the basis of the algorithm for a general matrix $\mathbf{A}$: (i) If $\mathbf{A}$ has eigenvalues of different absolute value $|\lambda_i|$, then $\mathbf{A}_s \to$ [lower triangular form] as $s \to \infty$. The eigenvalues appear on the diagonal in increasing order of absolute magnitude. (ii) If $\mathbf{A}$ has an eigenvalue $|\lambda_i|$ of multiplicity p, $\mathbf{A}_s \to$ [lower triangular form] as $s \to \infty$, except for a diagonal block matrix of order p, whose eigenvalues $\to \lambda_i$. The proof of this theorem is fairly lengthy; see, for example, [4].

The workload in the QL algorithm is $O(n^3)$ per iteration for a general matrix, which is prohibitive. However, the workload is only $O(n)$ per iteration for a tridiagonal matrix and $O(n^2)$ for a Hessenberg matrix, which makes it highly efficient on these forms.

In this section we are concerned only with the case where $\mathbf{A}$ is a real, symmetric, tridiagonal matrix. All the eigenvalues λ_i are thus real. According to the theorem, if any λ_i has a multiplicity p, then there must be at least $p-1$ zeros on the sub- and superdiagonal. Thus the matrix can be split into submatrices that can be diagonalized separately, and the complication of diagonal blocks that can arise in the general case is irrelevant.

In the proof of the theorem quoted above, one finds that in general a super-diagonal element converges to zero like

$$a_{ij}^{(s)} \sim \left(\frac{\lambda_i}{\lambda_j}\right)^s \tag{11.3.6}$$

Although $\lambda_i < \lambda_j$, convergence can be slow if λ_i is close to λ_j. Convergence can be accelerated by the technique of *shifting*: If k is any constant, then $\mathbf{A} - k\mathbf{1}$ has eigenvalues $\lambda_i - k$. If we decompose

$$\mathbf{A}_s - k_s\mathbf{1} = \mathbf{Q}_s \cdot \mathbf{L}_s \tag{11.3.7}$$

so that

$$\begin{aligned}\mathbf{A}_{s+1} &= \mathbf{L}_s \cdot \mathbf{Q}_s + k_s\mathbf{1} \\ &= \mathbf{Q}_s^T \cdot \mathbf{A}_s \cdot \mathbf{Q}_s\end{aligned} \tag{11.3.8}$$

then the convergence is determined by the ratio

$$\frac{\lambda_i - k_s}{\lambda_j - k_s} \tag{11.3.9}$$

The idea is to choose the shift k_s at each stage to maximize the rate of convergence. A good choice for the shift initially would be k_s close to λ_1, the smallest eigenvalue. Then the first row of off-diagonal elements would tend rapidly to zero. However, λ_1 is not usually known *a priori*. A very effective strategy in practice (although there is no proof that it is optimal) is to compute the eigenvalues of the leading 2×2 diagonal submatrix of **A**. Then set k_s equal to the eigenvalue closer to a_{11}.

More generally, suppose you have already found $r - 1$ eigenvalues of **A**. Then you can *deflate* the matrix by crossing out the first $r - 1$ rows and columns, leaving

$$\mathbf{A} = \begin{bmatrix} 0 & & & \cdots & \cdots & & & 0 \\ & \cdots & & & & & & \\ & & 0 & & & & & \\ \vdots & & & d_r & e_r & & & \vdots \\ \vdots & & & e_r & d_{r+1} & & & \\ & & & & & \cdots & & 0 \\ & & & & & & d_{n-1} & e_{n-1} \\ 0 & & & \cdots & & 0 & e_{n-1} & d_n \end{bmatrix} \tag{11.3.10}$$

Choose k_s equal to the eigenvalue of the leading 2×2 submatrix that is closer to d_r. One can show that the convergence of the algorithm with this strategy is generally cubic (and at worst quadratic for degenerate eigenvalues). This rapid convergence is what makes the algorithm so attractive.

Note that with shifting, the eigenvalues no longer necessarily appear on the diagonal in order of increasing absolute magnitude. The routine `eigsrt` (§11.1) can be used if required.

As we mentioned earlier, the QL decomposition of a general matrix is effected by a sequence of Householder transformations. For a tridiagonal matrix, however, it is more efficient to use plane rotations $\mathbf{P}_{pq}$. One uses the sequence $\mathbf{P}_{12}, \mathbf{P}_{23}, \ldots, \mathbf{P}_{n-1,n}$ to annihilate the elements $a_{12}, a_{23}, \ldots, a_{n-1,n}$. By symmetry, the subdiagonal elements $a_{21}, a_{32}, \ldots, a_{n,n-1}$ will be annihilated too. Thus each $\mathbf{Q}_s$ is a product of plane rotations:

$$\mathbf{Q}_s^T = \mathbf{P}_1^{(s)} \cdot \mathbf{P}_2^{(s)} \cdots \mathbf{P}_{n-1}^{(s)} \tag{11.3.11}$$

where $\mathbf{P}_i$ annihilates $a_{i,i+1}$. Note that it is $\mathbf{Q}^T$ in equation (11.3.11), not $\mathbf{Q}$, because we defined $\mathbf{L} = \mathbf{Q}^T \cdot \mathbf{A}$.

QL Algorithm with Implicit Shifts

The algorithm as described so far can be very successful. However, when the elements of **A** differ widely in order of magnitude, subtracting a large k_s from the diagonal elements can lead to loss of accuracy for the small eigenvalues. This difficulty is avoided by the QL algorithm with *implicit shifts*. The implicit QL algorithm is mathematically equivalent to the original QL algorithm, but the computation does not require $k_s\mathbf{1}$ to be actually subtracted from **A**.

The algorithm is based on the following lemma: If **A** is a symmetric nonsingular matrix and $\mathbf{B} = \mathbf{Q}^T \cdot \mathbf{A} \cdot \mathbf{Q}$, where **Q** is orthogonal and **B** is tridiagonal with positive off-diagonal elements, then **Q** and **B** are fully determined when the last row of $\mathbf{Q}^T$ is specified. Proof: Let $\mathbf{q}_i^T$ denote the ith row vector of the matrix $\mathbf{Q}^T$. Then $\mathbf{q}_i$ is the ith column vector of the matrix $\mathbf{Q}$. The relation $\mathbf{B} \cdot \mathbf{Q}^T = \mathbf{Q}^T \cdot \mathbf{A}$ can be written

$$\begin{bmatrix} \beta_1 & \gamma_1 & & & \\ \alpha_2 & \beta_2 & \gamma_2 & & \\ & & \vdots & & \\ & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} \\ & & & \alpha_n & \beta_n \end{bmatrix} \cdot \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_{n-1}^T \\ \mathbf{q}_n^T \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_{n-1}^T \\ \mathbf{q}_n^T \end{bmatrix} \cdot \mathbf{A} \qquad (11.3.12)$$

The nth row of this matrix equation is

$$\alpha_n \mathbf{q}_{n-1}^T + \beta_n \mathbf{q}_n^T = \mathbf{q}_n^T \cdot \mathbf{A} \qquad (11.3.13)$$

Since $\mathbf{Q}$ is orthogonal,

$$\mathbf{q}_n^T \cdot \mathbf{q}_m = \delta_{nm} \qquad (11.3.14)$$

Thus if we postmultiply equation (11.3.13) by $\mathbf{q}_n$, we find

$$\beta_n = \mathbf{q}_n^T \cdot \mathbf{A} \cdot \mathbf{q}_n \qquad (11.3.15)$$

which is known since $\mathbf{q}_n$ is known. Then equation (11.3.13) gives

$$\alpha_n \mathbf{q}_{n-1}^T = \mathbf{z}_{n-1}^T \qquad (11.3.16)$$

where

$$\mathbf{z}_{n-1}^T \equiv \mathbf{q}_n^T \cdot \mathbf{A} - \beta_n \mathbf{q}_n^T \qquad (11.3.17)$$

is known. Therefore

$$\alpha_n^2 = \mathbf{z}_{n-1}^T \mathbf{z}_{n-1}, \qquad (11.3.18)$$

or

$$\alpha_n = |\mathbf{z}_{n-1}| \qquad (11.3.19)$$

and

$$\mathbf{q}_{n-1}^T = \mathbf{z}_{n-1}^T / \alpha_n \qquad (11.3.20)$$

(where α_n is nonzero by hypothesis). Similarly, one can show by induction that if we know $\mathbf{q}_n, \mathbf{q}_{n-1}, \dots, \mathbf{q}_{n-j}$ and the α's, β's, and γ's up to level $n-j$, one can determine the quantities at level $n-(j+1)$.

To apply the lemma in practice, suppose one can somehow find a tridiagonal matrix $\overline{\mathbf{A}}_{s+1}$ such that

$$\overline{\mathbf{A}}_{s+1} = \overline{\mathbf{Q}}_s^T \cdot \overline{\mathbf{A}}_s \cdot \overline{\mathbf{Q}}_s \qquad (11.3.21)$$

where $\overline{\mathbf{Q}}_s^T$ is orthogonal and has the same last row as $\mathbf{Q}_s^T$ in the original QL algorithm. Then $\overline{\mathbf{Q}}_s = \mathbf{Q}_s$ and $\overline{\mathbf{A}}_{s+1} = \mathbf{A}_{s+1}$.

Now, in the original algorithm, from equation (11.3.11) we see that the last row of $\mathbf{Q}_s^T$ is the same as the last row of $\mathbf{P}_{n-1}^{(s)}$. But recall that $\mathbf{P}_{n-1}^{(s)}$ is a plane rotation designed to annihilate the $(n-1,n)$ element of $\mathbf{A}_s - k_s\mathbf{1}$. A simple calculation using the expression (11.1.1) shows that it has parameters

$$c = \frac{d_n - k_s}{\sqrt{e_n^2 + (d_n - k_s)^2}} \quad , \quad s = \frac{-e_{n-1}}{\sqrt{e_n^2 + (d_n - k_s)^2}} \qquad (11.3.22)$$

The matrix $\mathbf{P}_{n-1}^{(s)} \cdot \mathbf{A}_s \cdot \mathbf{P}_{n-1}^{(s)T}$ is tridiagonal with 2 extra elements:

$$\begin{bmatrix} \cdots & & & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \mathbf{x} \\ & & & \times & \times & \times \\ & & & \mathbf{x} & \times & \times \end{bmatrix} \qquad (11.3.23)$$

We must now reduce this to tridiagonal form with an orthogonal matrix whose last row is $[0, 0, \dots, 0, 1]$ so that the last row of $\overline{\mathbf{Q}}_s^T$ will stay equal to $\mathbf{P}_{n-1}^{(s)}$. This can be done by a sequence of Householder or Givens transformations. For the special form of the matrix (11.3.23), Givens is better. We rotate in the plane $(n-2, n-1)$ to annihilate the $(n-2, n)$ element. [By symmetry, the $(n, n-2)$ element will also be zeroed.] This leaves us with tridiagonal form except for extra elements $(n-3, n-1)$ and $(n-1, n-3)$. We annihilate these with a rotation in the $(n-3, n-2)$ plane, and so on. Thus a sequence of $n-2$ Givens rotations is required. The result is that

$$\mathbf{Q}_s^T = \overline{\mathbf{Q}}_s^T = \overline{\mathbf{P}}_1^{(s)} \cdot \overline{\mathbf{P}}_2^{(s)} \cdots \overline{\mathbf{P}}_{n-2}^{(s)} \cdot \mathbf{P}_{n-1}^{(s)} \tag{11.3.24}$$

where the $\overline{\mathbf{P}}$'s are the Givens rotations and $\mathbf{P}_{n-1}$ is the same plane rotation as in the original algorithm. Then equation (11.3.21) gives the next iterate of $\mathbf{A}$. Note that the shift k_s enters implicitly through the parameters (11.3.22).

The following routine `tqli` ("Tridiagonal QL Implicit"), based algorithmically on the implementations in [2,3], works extremely well in practice. The number of iterations for the first few eigenvalues might be 4 or 5, say, but meanwhile the off-diagonal elements in the lower right-hand corner have been reduced too. The later eigenvalues are liberated with very little work. The average number of iterations per eigenvalue is typically $1.3 - 1.6$. The operation count per iteration is $O(n)$, with a fairly large effective coefficient, say, $\sim 20n$. The total operation count for the diagonalization is then $\sim 20n \times (1.3 - 1.6)n \sim 30n^2$. If the eigenvectors are required, the statements indicated by comments are included and there is an additional, much larger, workload of about $3n^3$ operations.

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

void tqli(float d[], float e[], int n, float **z)
```

QL algorithm with implicit shifts, to determine the eigenvalues and eigenvectors of a real, symmetric, tridiagonal matrix, or of a real, symmetric matrix previously reduced by `tred2` §11.2. On input, `d[1..n]` contains the diagonal elements of the tridiagonal matrix. On output, it returns the eigenvalues. The vector `e[1..n]` inputs the subdiagonal elements of the tridiagonal matrix, with `e[1]` arbitrary. On output `e` is destroyed. When finding only the eigenvalues, several lines may be omitted, as noted in the comments. If the eigenvectors of a tridiagonal matrix are desired, the matrix `z[1..n][1..n]` is input as the identity matrix. If the eigenvectors of a matrix that has been reduced by `tred2` are required, then `z` is input as the matrix output by `tred2`. In either case, the `k`th column of `z` returns the normalized eigenvector corresponding to `d[k]`.

```
{
    float pythag(float a, float b);
    int m,l,iter,i,k;
    float s,r,p,g,f,dd,c,b;

    for (i=2;i<=n;i++) e[i-1]=e[i];              Convenient to renumber the el-
    e[n]=0.0;                                        ements of e.
    for (l=1;l<=n;l++) {
        iter=0;
        do {
            for (m=l;m<=n-1;m++) {                   Look for a single small subdi-
                dd=fabs(d[m])+fabs(d[m+1]);              agonal element to split
                if ((float)(fabs(e[m])+dd) == dd) break;     the matrix.
            }
            if (m != l) {
                if (iter++ == 30) nrerror("Too many iterations in tqli");
                g=(d[l+1]-d[l])/(2.0*e[l]);          Form shift.
                r=pythag(g,1.0);
                g=d[m]-d[l]+e[l]/(g+SIGN(r,g));      This is d_m - k_s.
                s=c=1.0;
                p=0.0;
```

```
            for (i=m-1;i>=l;i--) {                A plane rotation as in the origi-
                f=s*e[i];                             nal QL, followed by Givens
                b=c*e[i];                             rotations to restore tridiag-
                e[i+1]=(r=pythag(f,g));               onal form.
                if (r == 0.0) {                   Recover from underflow.
                    d[i+1] -= p;
                    e[m]=0.0;
                    break;
                }
                s=f/r;
                c=g/r;
                g=d[i+1]-p;
                r=(d[i]-g)*s+2.0*c*b;
                d[i+1]=g+(p=s*r);
                g=c*r-b;
                /* Next loop can be omitted if eigenvectors not wanted*/
                for (k=1;k<=n;k++) {              Form eigenvectors.
                    f=z[k][i+1];
                    z[k][i+1]=s*z[k][i]+c*f;
                    z[k][i]=c*z[k][i]-s*f;
                }
            }
            if (r == 0.0 && i >= l) continue;
            d[l] -= p;
            e[l]=g;
            e[m]=0.0;
        }
    } while (m != l);
  }
}
```


CITED REFERENCES AND FURTHER READING:

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America), pp. 331–335. [1]

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag). [2]

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). [3]

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §6.6.6. [4]

11.4 Hermitian Matrices

The complex analog of a real, symmetric matrix is a Hermitian matrix, satisfying equation (11.0.4). Jacobi transformations can be used to find eigenvalues and eigenvectors, as also can Householder reduction to tridiagonal form followed by QL iteration. Complex versions of the previous routines `jacobi`, `tred2`, and `tqli` are quite analogous to their real counterparts. For working routines, consult [1,2].

An alternative, using the routines in this book, is to convert the Hermitian problem to a real, symmetric one: If $\mathbf{C} = \mathbf{A} + i\mathbf{B}$ is a Hermitian matrix, then the $n \times n$ complex eigenvalue problem

$$(\mathbf{A} + i\mathbf{B}) \cdot (\mathbf{u} + i\mathbf{v}) = \lambda(\mathbf{u} + i\mathbf{v}) \tag{11.4.1}$$