```
        for (j=i+1;j<=n;j++)
            if (d[j] >= p) p=d[k=j];
        if (k != i) {
            d[k]=d[i];
            d[i]=p;
            for (j=1;j<=n;j++) {
                p=v[j][i];
                v[j][i]=v[j][k];
                v[j][k]=p;
            }
        }
    }
}
```

CITED REFERENCES AND FURTHER READING:

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §8.4.

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). [1]

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

11.2 Reduction of a Symmetric Matrix to Tridiagonal Form: Givens and Householder Reductions

As already mentioned, the optimum strategy for finding eigenvalues and eigenvectors is, first, to reduce the matrix to a simple form, only then beginning an iterative procedure. For symmetric matrices, the preferred simple form is tridiagonal. The *Givens reduction* is a modification of the Jacobi method. Instead of trying to reduce the matrix all the way to diagonal form, we are content to stop when the matrix is tridiagonal. This allows the procedure to be carried out *in a finite number of steps,* unlike the Jacobi method, which requires iteration to convergence.

Givens Method

For the Givens method, we choose the rotation angle in equation (11.1.1) so as to zero an element that is *not* at one of the four "corners," i.e., not a_{pp}, a_{pq}, or a_{qq} in equation (11.1.3). Specifically, we first choose $\mathbf{P}_{23}$ to annihilate a_{31} (and, by symmetry, a_{13}). Then we choose $\mathbf{P}_{24}$ to annihilate a_{41}. In general, we choose the sequence

$$\mathbf{P}_{23}, \mathbf{P}_{24}, \dots, \mathbf{P}_{2n}; \mathbf{P}_{34}, \dots, \mathbf{P}_{3n}; \dots ; \mathbf{P}_{n-1,n}$$

where $\mathbf{P}_{jk}$ annihilates $a_{k,j-1}$. The method works because elements such as a'_{rp} and a'_{rq}, with $r \neq p$ $r \neq q$, are linear combinations of the old quantities a_{rp} and a_{rq}, by equation (11.1.4). Thus, if a_{rp} and a_{rq} have already been set to zero, they remain zero as the reduction proceeds. Evidently, of order $n^2/2$ rotations are required, and the number of multiplications in a straightforward implementation is of order $4n^3/3$, not counting those for keeping track of the product of the transformation matrices, required for the eigenvectors.

The Householder method, to be discussed next, is just as stable as the Givens reduction and it is a factor of 2 more efficient, so the Givens method is not generally used. Recent work (see [1]) has shown that the Givens reduction can be reformulated to reduce the number of operations by a factor of 2, and also avoid the necessity of taking square roots. This appears to make the algorithm competitive with the Householder reduction. However, this "fast Givens" reduction has to be monitored to avoid overflows, and the variables have to be periodically rescaled. There does not seem to be any compelling reason to prefer the Givens reduction over the Householder method.

Householder Method

The Householder algorithm reduces an $n \times n$ symmetric matrix **A** to tridiagonal form by $n - 2$ orthogonal transformations. Each transformation annihilates the required part of a whole column and whole corresponding row. The basic ingredient is a Householder matrix **P**, which has the form

$$\mathbf{P} = \mathbf{1} - 2\mathbf{w} \cdot \mathbf{w}^T \tag{11.2.1}$$

where **w** is a real vector with $|\mathbf{w}|^2 = 1$. (In the present notation, the *outer* or matrix product of two vectors, **a** and **b** is written $\mathbf{a} \cdot \mathbf{b}^T$, while the *inner* or scalar product of the vectors is written as $\mathbf{a}^T \cdot \mathbf{b}$.) The matrix **P** is orthogonal, because

$$\begin{aligned}\mathbf{P}^2 &= (\mathbf{1} - 2\mathbf{w} \cdot \mathbf{w}^T) \cdot (\mathbf{1} - 2\mathbf{w} \cdot \mathbf{w}^T)\\ &= \mathbf{1} - 4\mathbf{w} \cdot \mathbf{w}^T + 4\mathbf{w} \cdot (\mathbf{w}^T \cdot \mathbf{w}) \cdot \mathbf{w}^T\\ &= \mathbf{1}\end{aligned} \tag{11.2.2}$$

Therefore $\mathbf{P} = \mathbf{P}^{-1}$. But $\mathbf{P}^T = \mathbf{P}$, and so $\mathbf{P}^T = \mathbf{P}^{-1}$, proving orthogonality.

Rewrite **P** as

$$\mathbf{P} = \mathbf{1} - \frac{\mathbf{u} \cdot \mathbf{u}^T}{H} \tag{11.2.3}$$

where the scalar H is

$$H \equiv \frac{1}{2}|\mathbf{u}|^2 \tag{11.2.4}$$

and **u** can now be any vector. Suppose **x** is the vector composed of the first column of **A**. Choose

$$\mathbf{u} = \mathbf{x} \mp |\mathbf{x}|\mathbf{e}_1 \tag{11.2.5}$$

where $\mathbf{e}_1$ is the unit vector $[1, 0, \dots, 0]^T$, and the choice of signs will be made later. Then

$$\begin{aligned}\mathbf{P}\cdot\mathbf{x} &= \mathbf{x} - \frac{\mathbf{u}}{H}\cdot(\mathbf{x} \mp |\mathbf{x}|\mathbf{e}_1)^T\cdot\mathbf{x}\\ &= \mathbf{x} - \frac{2\mathbf{u}\cdot(|\mathbf{x}|^2 \mp |\mathbf{x}|x_1)}{2|\mathbf{x}|^2 \mp 2|\mathbf{x}|x_1}\\ &= \mathbf{x} - \mathbf{u}\\ &= \pm|\mathbf{x}|\mathbf{e}_1\end{aligned} \tag{11.2.6}$$

This shows that the Householder matrix $\mathbf{P}$ acts on a given vector $\mathbf{x}$ to zero all its elements except the first one.

To reduce a symmetric matrix $\mathbf{A}$ to tridiagonal form, we choose the vector $\mathbf{x}$ for the first Householder matrix to be the lower $n-1$ elements of the first column. Then the lower $n-2$ elements will be zeroed:

$$\begin{aligned}\mathbf{P}_1\cdot\mathbf{A} &= \left[\begin{array}{c|cccc} 1 & 0 & 0 & \cdots & 0\\ \hline 0 & & & & \\ 0 & & & & \\ \vdots & & {}^{(n-1)}\mathbf{P}_1 & & \\ 0 & & & & \end{array}\right]\cdot\left[\begin{array}{c|cccc} a_{11} & a_{12} & a_{13} & \cdots & a_{1n}\\ \hline a_{21} & & & & \\ a_{31} & & & & \\ \vdots & & & \text{irrelevant} & \\ a_{n1} & & & & \end{array}\right]\\ &= \left[\begin{array}{c|cccc} a_{11} & a_{12} & a_{13} & \cdots & a_{1n}\\ \hline k & & & & \\ 0 & & & & \\ \vdots & & & \text{irrelevant} & \\ 0 & & & & \end{array}\right]\end{aligned} \tag{11.2.7}$$

Here we have written the matrices in partitioned form, with ${}^{(n-1)}\mathbf{P}$ denoting a Householder matrix with dimensions $(n-1)\times(n-1)$. The quantity k is simply plus or minus the magnitude of the vector $[a_{21}, \dots, a_{n1}]^T$.

The complete orthogonal transformation is now

$$\mathbf{A}' = \mathbf{P}\cdot\mathbf{A}\cdot\mathbf{P} = \left[\begin{array}{c|cccc} a_{11} & k & 0 & \cdots & 0\\ \hline k & & & & \\ 0 & & & & \\ \vdots & & & \text{irrelevant} & \\ 0 & & & & \end{array}\right] \tag{11.2.8}$$

We have used the fact that $\mathbf{P}^T = \mathbf{P}$.

Now choose the vector **x** for the second Householder matrix to be the bottom $n-2$ elements of the second column, and from it construct

$$\mathbf{P}_2 \equiv \left[\begin{array}{cc|ccc} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & {}^{(n-2)}\mathbf{P}_2 & \\ 0 & 0 & & & \end{array}\right] \qquad (11.2.9)$$

The identity block in the upper left corner insures that the tridiagonalization achieved in the first step will not be spoiled by this one, while the $(n-2)$-dimensional Householder matrix ${}^{(n-2)}\mathbf{P}_2$ creates one additional row and column of the tridiagonal output. Clearly, a sequence of $n-2$ such transformations will reduce the matrix **A** to tridiagonal form.

Instead of actually carrying out the matrix multiplications in $\mathbf{P}\cdot\mathbf{A}\cdot\mathbf{P}$, we compute a vector

$$\mathbf{p} \equiv \frac{\mathbf{A}\cdot\mathbf{u}}{H} \qquad (11.2.10)$$

Then

$$\mathbf{A}\cdot\mathbf{P} = \mathbf{A}\cdot(1-\frac{\mathbf{u}\cdot\mathbf{u}^T}{H}) = \mathbf{A}-\mathbf{p}\cdot\mathbf{u}^T$$
$$\mathbf{A}' = \mathbf{P}\cdot\mathbf{A}\cdot\mathbf{P} = \mathbf{A}-\mathbf{p}\cdot\mathbf{u}^T-\mathbf{u}\cdot\mathbf{p}^T+2K\mathbf{u}\cdot\mathbf{u}^T$$

where the scalar K is defined by

$$K = \frac{\mathbf{u}^T\cdot\mathbf{p}}{2H} \qquad (11.2.11)$$

If we write

$$\mathbf{q} \equiv \mathbf{p} - K\mathbf{u} \qquad (11.2.12)$$

then we have

$$\mathbf{A}' = \mathbf{A} - \mathbf{q}\cdot\mathbf{u}^T - \mathbf{u}\cdot\mathbf{q}^T \qquad (11.2.13)$$

This is the computationally useful formula.

Following [2], the routine for Householder reduction given below actually starts in the nth column of **A**, not the first as in the explanation above. In detail, the equations are as follows: At stage m $(m = 1, 2, \ldots, n-2)$ the vector **u** has the form

$$\mathbf{u}^T = [a_{i1}, a_{i2}, \ldots, a_{i,i-2},\ a_{i,i-1} \pm \sqrt{\sigma},\ 0, \ldots, 0] \qquad (11.2.14)$$

Here

$$i \equiv n - m + 1 = n, n-1, \ldots, 3 \qquad (11.2.15)$$

and the quantity σ ($|x|^2$ in our earlier notation) is

$$\sigma = (a_{i1})^2 + \cdots + (a_{i,i-1})^2 \qquad (11.2.16)$$

We choose the sign of σ in (11.2.14) to be the same as the sign of $a_{i,i-1}$ to lessen roundoff error.

Variables are thus computed in the following order: $\sigma, \mathbf{u}, H, \mathbf{p}, K, \mathbf{q}, \mathbf{A}'$. At any stage m, $\mathbf{A}$ is tridiagonal in its last $m-1$ rows and columns.

If the eigenvectors of the final tridiagonal matrix are found (for example, by the routine in the next section), then the eigenvectors of $\mathbf{A}$ can be obtained by applying the accumulated transformation

$$\mathbf{Q} = \mathbf{P}_1 \cdot \mathbf{P}_2 \cdots \mathbf{P}_{n-2} \tag{11.2.17}$$

to those eigenvectors. We therefore form $\mathbf{Q}$ by recursion after all the $\mathbf{P}$'s have been determined:

$$\begin{aligned} \mathbf{Q}_{n-2} &= \mathbf{P}_{n-2} \\ \mathbf{Q}_j &= \mathbf{P}_j \cdot \mathbf{Q}_{j+1}, \qquad j = n-3, \ldots, 1 \\ \mathbf{Q} &= \mathbf{Q}_1 \end{aligned} \tag{11.2.18}$$

Input for the routine below is the real, symmetric matrix `a[1..n][1..n]`. On output, `a` contains the elements of the orthogonal matrix `q`. The vector `d[1..n]` is set to the diagonal elements of the tridiagonal matrix $\mathbf{A}'$, while the vector `e[1..n]` is set to the off-diagonal elements in its components 2 through `n`, with `e[1]=0`. Note that since `a` is overwritten, you should copy it before calling the routine, if it is required for subsequent computations.

No extra storage arrays are needed for the intermediate results. At stage m, the vectors $\mathbf{p}$ and $\mathbf{q}$ are nonzero only in elements $1, \ldots, i$ (recall that $i = n - m + 1$), while $\mathbf{u}$ is nonzero only in elements $1, \ldots, i-1$. The elements of the vector `e` are being determined in the order $n, n-1, \ldots$, so we can store $\mathbf{p}$ in the elements of `e` not already determined. The vector $\mathbf{q}$ can overwrite $\mathbf{p}$ once $\mathbf{p}$ is no longer needed. We store $\mathbf{u}$ in the ith row of `a` and $\mathbf{u}/H$ in the ith column of `a`. Once the reduction is complete, we compute the matrices $\mathbf{Q}_j$ using the quantities $\mathbf{u}$ and $\mathbf{u}/H$ that have been stored in `a`. Since $\mathbf{Q}_j$ is an identity matrix in the last $n-j+1$ rows and columns, we only need compute its elements up to row and column $n-j$. These can overwrite the $\mathbf{u}$'s and $\mathbf{u}/H$'s in the corresponding rows and columns of `a`, which are no longer required for subsequent $\mathbf{Q}$'s.

The routine `tred2`, given below, includes one further refinement. If the quantity σ is zero or "small" at any stage, one can skip the corresponding transformation. A simple criterion, such as

$$\sigma < \frac{\text{smallest positive number representable on machine}}{\text{machine precision}}$$

would be fine most of the time. A more careful criterion is actually used. Define the quantity

$$\epsilon = \sum_{k=1}^{i-1} |a_{ik}| \tag{11.2.19}$$

If $\epsilon = 0$ to machine precision, we skip the transformation. Otherwise we redefine

$$a_{ik} \quad \text{becomes} \quad a_{ik}/\epsilon \tag{11.2.20}$$

and use the scaled variables for the transformation. (A Householder transformation depends only on the ratios of the elements.)

Note that when dealing with a matrix whose elements vary over many orders of magnitude, it is important that the matrix be permuted, insofar as possible, so that the smaller elements are in the top left-hand corner. This is because the reduction is performed starting from the bottom right-hand corner, and a mixture of small and large elements there can lead to considerable rounding errors.

The routine `tred2` is designed for use with the routine `tqli` of the next section. `tqli` finds the eigenvalues and eigenvectors of a symmetric, tridiagonal matrix. The combination of `tred2` and `tqli` is the most efficient known technique for finding all the eigenvalues and eigenvectors (or just all the eigenvalues) of a real, symmetric matrix.

In the listing below, the statements indicated by comments are required only for subsequent computation of eigenvectors. If only eigenvalues are required, omission of the commented statements speeds up the execution time of `tred2` by a factor of 2 for large n. In the limit of large n, the operation count of the Householder reduction is $2n^3/3$ for eigenvalues only, and $4n^3/3$ for both eigenvalues and eigenvectors.

```
#include <math.h>

void tred2(float **a, int n, float d[], float e[])
Householder reduction of a real, symmetric matrix a[1..n][1..n]. On output, a is replaced
by the orthogonal matrix Q effecting the transformation. d[1..n] returns the diagonal ele-
ments of the tridiagonal matrix, and e[1..n] the off-diagonal elements, with e[1]=0. Several
statements, as noted in comments, can be omitted if only eigenvalues are to be found, in which
case a contains no useful information on output. Otherwise they are to be included.
{
    int l,k,j,i;
    float scale,hh,h,g,f;

    for (i=n;i>=2;i--) {
        l=i-1;
        h=scale=0.0;
        if (l > 1) {
            for (k=1;k<=l;k++)
                scale += fabs(a[i][k]);
            if (scale == 0.0)                 Skip transformation.
                e[i]=a[i][l];
            else {
                for (k=1;k<=l;k++) {
                    a[i][k] /= scale;         Use scaled a's for transformation.
                    h += a[i][k]*a[i][k];     Form σ in h.
                }
                f=a[i][l];
                g=(f >= 0.0 ? -sqrt(h) : sqrt(h));
                e[i]=scale*g;
                h -= f*g;                     Now h is equation (11.2.4).
                a[i][l]=f-g;                  Store u in the ith row of a.
                f=0.0;
                for (j=1;j<=l;j++) {
                /* Next statement can be omitted if eigenvectors not wanted */
                    a[j][i]=a[i][j]/h;        Store u/H in ith column of a.
                    g=0.0;                    Form an element of A · u in g.
                    for (k=1;k<=j;k++)
                        g += a[j][k]*a[i][k];
                    for (k=j+1;k<=l;k++)
                        g += a[k][j]*a[i][k];
                    e[j]=g/h;                 Form element of p in temporarily unused
                                                 element of e.
                    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