# 55
# Symmetric Matrix Eigenvalue Techniques

Ivan Slapničar
*University of Split*

The eigenvalue decomposition (EVD) is an infinite iterative procedure — finding eigenvalues is equivalent to finding zeros of the characteristic polynomial, and, by the results of Abel and Galois, there is no algebraic formula for roots of the polynomial of degree greater than four. However, the number of arithmetic operations required to compute EVD to some prescribed accuracy is also finite — EVD of a general symmetric matrix requires $O(n^3)$ operations, while for matrices with special structure this number can be smaller. For example, the EVD of a tridiagonal matrix can be computed in $O(n^2)$ operations (see Sections 55.5 and 55.6).

Basic methods for the symmetric eigenvalue computations are the power method, the inverse iteration method, and the QR iteration method (see Section 55.1). Since direct application of those methods to a general symmetric matrix requires $O(n^4)$ operations, the most commonly used algorithms consist of two steps: the given matrix is first reduced to tridiagonal form, followed by the computation of the EVD of the tridiagonal matrix by QR iteration, the divide and conquer method, bisection and inverse iteration, or the method of multiple relatively robust representations. Two other methods are the Jacobi method, which does not require tridiagonalization, and the Lanczos method, which computes only a part of the tridiagonal matrix.

Design of an efficient algorithm must take into account the target computer, the desired speed and accuracy, the specific goal (whether all or some eigenvalues and eigenvectors are desired), and the matrix size and structure (small or large, dense or sparse, tridiagonal, etc.). For example, if only some eigenvalues and eigenvectors are required, one can use the methods of Sections 55.5, 55.6, and 55.8. If high relative accuracy is desired and the matrix is positive definite, the Jacobi method is the method of choice. If the matrix is sparse, the Lanczos method should be used. We shall cover the most commonly used algorithms, like those which are implemented in LAPACK (see Chapter 93) and MATLAB® (see Chapter 88). The algorithms provided in this chapter are intended to assist the reader in understanding the methods. Since the actual software is very complex, the reader is advised to use professional software in practice.

Efficient algorithms should be designed to use BLAS, and especially BLAS 3, as much as possible (see Chapter 92). The reasons are twofold: First, calling predefined standardized routines makes programs shorter and more easily readable, and second, processor vendors can optimize sets of standardized routines for their processor beyond the level given by compiler optimization. Examples of such optimized libraries are the *Intel Math Kernel Library* and *AMD Core Math Library*. Both libraries contain processor optimized BLAS, LAPACK, and FFT routines.

This chapter deals only with the computation of EVD of real symmetric matrices. The need to compute EVD of a complex Hermitian matrix (see Chapter 9) does not arise often in applications, and it is theoretically and numerically similar to the real symmetric case addressed here. All algorithms described in this chapter have their Hermitian counterparts (see e.g., [ABB99], [LSY98], and Chapters 88, 93, and 94).

The chapter is organized as follows: In Section 55.1, we describe basic methods for EVD computations. These methods are necessary to understand algorithms of Sections 55.3 to 55.6. In Section 55.2, we describe tridiagonalization by Householder reflections and Givens rotations. In Sections 55.3 to 55.6, we describe methods for computing the EVD of a tridiagonal matrix — QR iteration, the divide and conquer method, bisection and inverse iteration, and the method of multiple relatively robust representations, respectively. The Jacobi method is described in Section 55.7 and the Lanczos method is described in Section 55.8. For each method, we also describe the existing LAPACK or Matlab implementations. The respective timings of the methods are given in Section 55.9.

## 55.1 Basic Methods

The reader is referred to Sections 4.4, 9.1, 9.2, 9.3, 16.1, and 16.2 for more information on eigenvalues and their locations.

**Definitions:**

The **eigenvalue decomposition** (EVD) of a real symmetric matrix $A = [a_{ij}]$ is given by $A = U\Lambda U^T$, where $U$ is a $n \times n$ real orthonormal matrix, $U^TU = UU^T = I_n$, and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ is a real diagonal matrix.

The numbers $\lambda_i$ are the **eigenvalues** of $A$, the columns $\mathbf{u}_i$, $i = 1, \dots, n$, of $U$ are the **eigenvectors** of $A$, and $A\mathbf{u}_i = \lambda_i \mathbf{u}_i$, $i = 1, \dots, n$.

If $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$, we say that $\lambda_1$ is the **dominant eigenvalue**.

**Deflation** is a process of reducing the size of the matrix whose EVD is to be determined, given that one eigenvector is known (see Fact 4 for details).

The **shifted matrix** of the matrix $A$ is the matrix $A - \mu I$, where $\mu$ is the **shift**.

The simplest method for computing the EVD (also in the unsymmetric case) is the **power method**: given starting vector $\mathbf{x}_0$, the method computes the sequences

$$\nu_k = \mathbf{x}_k^T A \mathbf{x}_k, \qquad \mathbf{x}_{k+1} = A\mathbf{x}_k / \|A\mathbf{x}_k\|, \qquad k = 0, 1, 2, \dots, \tag{55.1}$$

until convergence. Normalization of $\mathbf{x}_k$ can be performed in any norm and serves the numerical stability of the algorithm (avoiding overflow or underflow).

**Inverse iteration** is the power method applied to the inverse of a shifted matrix, starting from $\mathbf{x}_0$:

$$\nu_k = \mathbf{x}_k^T A \mathbf{x}_k, \quad \mathbf{v}_{k+1} = (A - \mu I)^{-1} \mathbf{x}_k, \quad \mathbf{x}_{k+1} = \mathbf{v}_{k+1} / \|\mathbf{v}_{k+1}\|, \quad k = 0, 1, 2, \dots. \tag{55.2}$$

Given starting $n \times p$ matrix $X_0$ with orthonormal columns, the **orthogonal iteration** (also **subspace iteration**) forms the sequence of matrices

$$Y_{k+1} = AX_k, \qquad Y_{k+1} = X_{k+1}R_{k+1} \quad \text{(QR factorization)}, \qquad k = 0, 1, 2, \dots, \tag{55.3}$$

where $X_{k+1}R_{k+1}$ is the reduced QR factorization of $Y_{k+1}$ ($X_{k+1}$ is an $n \times p$ matrix with orthonormal columns and $R_{k+1}$ is an upper triangular $p \times p$ matrix).

Starting from the matrix $A_0 = A$, the **QR iteration** forms the sequence of matrices

$$A_k = Q_k R_k \quad \text{(QR factorization)}, \qquad A_{k+1} = R_k Q_k, \qquad k = 0, 1, 2, \ldots \tag{55.4}$$

Given the shift $\mu$, the **shifted QR iteration** forms the sequence of matrices

$$A_k - \mu I = Q_k R_k \quad \text{(QR factorization)}, \quad A_{k+1} = R_k Q_k + \mu I, \quad k = 0, 1, 2, \ldots \tag{55.5}$$

**Facts:**

Facts 1 to 14 can be found in [GV96, Chap. 8.2], [Par80, Chap. 4, 5], [Ste01, Chaps. 2.1, 2.2.1, 2.2.2], and [Dem97, Chap. 4].

1. If $\lambda_1$ is the dominant eigenvalue and if $\mathbf{x}_0$ is not orthogonal to $\mathbf{u}_1$, then in Eq. (55.1) $\nu_k \to \lambda_1$ and $\mathbf{x}_k \to \mathbf{u}_1$. In other words, the power method converges to the dominant eigenvalue and its eigenvector.
2. The convergence of the power method is linear in the sense that
$$|\lambda_1 - \nu_k| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), \qquad \|\mathbf{u}_1 - \mathbf{x}_k\|_2 = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$
More precisely,
$$|\lambda_1 - \nu_k| \approx \left|\frac{c_2}{c_1}\right| \left|\frac{\lambda_2}{\lambda_1}\right|^k,$$
where $c_i$ is the coefficient of the $i$-th eigenvector in the linear combination expressing the starting vector $\mathbf{x}_0$.
3. Since $\lambda_1$ is not readily available, the convergence is in practice determined using residuals. If $\|A\mathbf{x}_k - \nu_k \mathbf{x}_k\|_2 \leq tol$, where $tol$ is a user prescribed stopping criterion, then $|\lambda_1 - \nu_k| \leq tol$.
4. After computing the dominant eigenpair, we can perform deflation to reduce the given EVD to the one of size $n-1$. Let $Y = [\mathbf{u}_1 \quad X]$ be an orthogonal matrix. Then
$$\begin{bmatrix}\mathbf{u}_1 & X\end{bmatrix}^T A \begin{bmatrix}\mathbf{u}_1 & X\end{bmatrix} = \begin{bmatrix}\lambda_1 & 0 \\ 0 & A_1\end{bmatrix},$$
where $A_1 = X^T A X$.
5. The EVD of the shifted matrix $A - \mu I$ is given by $U(\Lambda - \mu I)U^T$. Sometimes we can choose shift $\mu$ such that the shifted matrix $A - \mu I$ has better ratio between the dominant eigenvalue and the absolutely closest one, than the original matrix. In this case, applying the power method to the shifted matrix will speed up the convergence.
6. Inverse iteration requires solving the system of linear equations $(A - \mu I)\mathbf{v}_{k+1} = \mathbf{x}_k$ for $\mathbf{v}_{k+1}$ in each step. At the beginning, we must compute the LU factorization of $A - \mu I$, which requires $2n^3/3$ operations and in each subsequent step we must solve two triangular systems, which requires $2n^2$ operations.
7. If $\mu$ is very close to some eigenvalue of $A$, then the eigenvalues of the shifted matrix satisfy $|\lambda_1| \gg |\lambda_2| \geq \cdots \geq |\lambda_n|$, so the convergence of the inverse iteration method is very fast.
8. If $\mu$ is very close to some eigenvalue of $A$, then the matrix $A - \mu I$ is nearly singular, so the solutions of linear systems may have large errors. However, these errors are almost entirely in the direction of the dominant eigenvector so the inverse iteration method is both fast and accurate.

9. We can further increase the speed of convergence of inverse iterations by substituting the shift $\mu$ with the Rayleigh quotient $\nu_k$ in each step, at the cost of computing new LU factorization each time. See Section 9.2 for more information about the Rayleigh quotient.
10. If
$$|\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n|,$$
then the subspace iteration given in Eq. (55.3) converges such that
$$X_k \rightarrow [\mathbf{u}_1, \ldots, \mathbf{u}_p], \qquad X_k^T A X_k \rightarrow \operatorname{diag}(\lambda_1, \ldots, \lambda_p),$$
at a speed which is proportional to $|\lambda_{p+1}/\lambda_p|^k$.
11. If $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, then the sequence of matrices $A_k$ generated by the QR iteration given in Eq. (55.4) converges to diagonal matrix $\Lambda$. However, this result is not of practical use, since the convergence may be very slow and each iteration requires $O(n^3)$ operations. Careful implementation, like the one described in Section 55.3, is needed to construct a useful algorithm.
12. The QR iteration is equivalent to orthogonal iteration starting with the matrix $X_0 = I$. More precisely, the matrices $X_k$ from Eq. (55.3) and $A_k$ from (Eq. 55.4) satisfy $X_k^T A X_k = A_k$.
13. Matrices $A_k$ and $A_{k+1}$ from Eq. (55.4) and Eq. (55.5) are orthogonally similar. In both cases
$$A_{k+1} = Q_k^T A_k Q_k.$$
14. The QR iteration method is essentially equivalent to the power method and the shifted QR iteration method is essentially equivalent to the inverse power method on the shifted matrix.
15. [Wil65, Chaps. 3, 5, 6, 7] [TB97, Chap. V] Let $U\Lambda U^T$ and $\tilde{U}\tilde{\Lambda}\tilde{U}^T$ be the exact and the computed EVDs of $A$, respectively, such that the diagonals of $\Lambda$ and $\tilde{\Lambda}$ are in the same order. Numerical methods generally compute the EVD with the errors bounded by
$$|\lambda_i - \tilde{\lambda}_i| \leq \phi\epsilon\|A\|_2, \qquad \|\mathbf{u}_i - \tilde{\mathbf{u}}_i\|_2 \leq \psi\epsilon \frac{\|A\|_2}{\min_{j\neq i} |\lambda_i - \tilde{\lambda}_j|},$$
where $\epsilon$ is machine precision and $\phi$ and $\psi$ are slowly growing polynomial functions of $n$ which depend upon the algorithm used (typically $O(n)$ or $O(n^2)$). Such bounds are obtained by combining perturbation bounds, like, for example, those of Section 21.1, with the floating-point error analysis of the respective algorithms.
16. For some types of matrices it is possible to compute EVD more accurately than stated in Fact 15. Two examples of such matrices are "well-behaved" matrices and arrowhead matrices. Eigenvalue problems for "well-behaved" matrices satisfy relative perturbation bounds of Section 21.6. The EVD for such matrices can be computed with high relative accuracy as described in Sections 59.4 and 59.5. Arrowhead matrices have the form
$$A = \begin{bmatrix} D & \mathbf{z} \\ \mathbf{z}^T & \alpha \end{bmatrix},$$
where $D = \operatorname{diag}(d_1, \ldots, d_{n-1})$, $\mathbf{z} = \begin{bmatrix} \zeta_1 & \cdots & \zeta_{n-1} \end{bmatrix}$, and $\alpha$ is a scalar. For such matrices all eigenvalues and all components of each eigenvector can be computed to nearly full accuracy in $O(n^2)$ operations. See [JSB12] for details.

**Examples:**

1. The eigenvalue decomposition of the matrix

$$A = \begin{bmatrix} 4.5013 & 0.6122 & 2.1412 & 2.0390 \\ 0.6122 & 2.6210 & -0.4941 & -1.2164 \\ 2.1412 & -0.4941 & 1.1543 & -0.1590 \\ 2.0390 & -1.2164 & -0.1590 & -0.9429 \end{bmatrix}$$

computed by the MATLAB command `[U,Lambda]=eig(A)` is $A = U\Lambda U^T$ with (properly rounded to four decimal places)

$$U \begin{bmatrix} -0.3697 & 0.2496 & 0.1003 & -0.8894 \\ 0.2810 & -0.0238 & 0.9593 & -0.0153 \\ 0.3059 & -0.8638 & -0.1172 & -0.3828 \\ 0.8311 & 0.4370 & -0.2366 & -0.2495 \end{bmatrix}, \quad \Lambda = \begin{bmatrix} -2.3197 & 0 & 0 & 0 \\ 0 & 0.6024 & 0 & 0 \\ 0 & 0 & 3.0454 & 0 \\ 0 & 0 & 0 & 6.0056 \end{bmatrix}.$$

2. Let $A$, $U$, and $\Lambda$ be as in the Example 1, and set $\mathbf{x}_0 = [1 \quad 1 \quad 1 \quad 1]^T$. The power method in Eq. (55.1) gives $\mathbf{x}_6 = [0.8893 \quad 0.0234 \quad 0.3826 \quad 0.2496]^T$. By setting $\mathbf{u}_1 = -U_{:,4}$ we have $\|\mathbf{u}_1 - \mathbf{x}_6\|_2 = 0.0081$. Here (Fact 2), $c_2 = 0.7058$, $c_1 = -1.5370$, and

$$\left|\frac{c_2}{c_1}\right| \left|\frac{\lambda_2}{\lambda_1}\right|^6 = 0.0078.$$

Similarly, $\|\mathbf{u}_1 - \mathbf{x}_{50}\|_2 = 1.3857 \cdot 10^{-15}$. However, for a different (bad) choice of the starting vector, $\mathbf{x}_0 = [0 \quad 1 \quad 0 \quad 0]^T$, where $c_2 = 0.9593$ and $c_1 = -0.0153$, we have $\|\mathbf{u}_1 - \mathbf{x}_6\|_2 = 0.7956$.

3. The deflation matrix $Y$ and the deflated matrix $A_1$ (Fact 4) for the above example are equal to (correctly rounded):

$$Y = \begin{bmatrix} -0.8894 & -0.0153 & -0.3828 & -0.2495 \\ -0.0153 & 0.9999 & -0.0031 & -0.0020 \\ -0.3828 & -0.0031 & 0.9224 & -0.0506 \\ -0.2495 & -0.0020 & -0.0506 & 0.9670 \end{bmatrix},$$

$$A_1 = \begin{bmatrix} 6.0056 & 0 & 0 & 0 \\ 0 & 2.6110 & -0.6379 & -1.3154 \\ 0 & -0.6379 & 0.2249 & -0.8952 \\ 0 & -1.3154 & -0.8952 & -1.5078 \end{bmatrix}.$$

4. Let $A$ and $\mathbf{x}_0$ be as in Example 2. For the shift $\mu = 6$, the inverse iteration method in Eq. (55.2) gives $\|\mathbf{u}_1 - \mathbf{x}_6\|_2 = 6.5187 \cdot 10^{-16}$, so the convergence is much faster than in Example 2 (Fact 7).
5. Let $A$ be as in Example 1. Applying six steps of the QR iteration in Eq. (55.4) gives

$$A_6 = \begin{bmatrix} 6.0055 & -0.0050 & -0.0118 & -0.0000 \\ -0.0050 & 3.0270 & 0.3134 & 0.0002 \\ -0.0118 & 0.3134 & -2.3013 & -0.0017 \\ -0.0000 & 0.0002 & -0.0017 & 0.6024 \end{bmatrix}.$$

and applying six steps of the shifted QR iteration in Eq. (55.5) with $\mu = 6$ gives

$$A_6 = \begin{bmatrix} -2.3123 & 0.1452 & -0.0215 & -0.0000 \\ 0.1452 & 0.6623 & 0.4005 & 0.0000 \\ -0.0215 & 0.4005 & 2.9781 & -0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 6.0056 \end{bmatrix}.$$

In this case both methods converge. The convergence toward the matrix where the eigenvalue nearest to the shift can be deflated is faster for the shifted iterations.

## 55.2 Tridiagonalization

The QR iteration in Eq. (55.4) and the shifted QR iteration in Eq. (55.5) require $O(n^3)$ operations (one QR factorization) for each step, which makes these algorithms highly unpractical. However, if the starting matrix is tridiagonal, one step of these iterations requires only $O(n)$ operations. As a consequence, the practical algorithm consists of three steps:

1. Reduce $A$ to tridiagonal form $T$ by orthogonal similarities, $X^T AX = T$.
2. Compute the EVD of $T$, $T = Q\Lambda Q^T$.
3. Multiply $U = XQ$.

The EVD of $A$ is then $A = U\Lambda U^T$. Reduction to tridiagonal form can be performed by using Householder reflectors or Givens rotations and it is a finite process requiring $O(n^3)$ operations. Reduction to tridiagonal form is a considerable compression of data since an EVD of $T$ can be computed very quickly. The EVD of $T$ can be efficiently computed by various methods such as QR iteration, the divide and conquer method (DC), bisection and inverse iteration, or the method of multiple relatively robust representations (MRRR). These methods are described in subsequent sections.

**Facts:**

All the following facts, except Fact 6, can be found in [Par80, Chap. 7], [TB97, pp. 196–201], [GV96, Chap. 8.3.1], [Ste01, pp. 158–162], and [Wil65, pp. 345–367].

1. Tridiagonal form is not unique (see Examples 1 and 2).
2. The reduction of $A$ to tridiagonal matrix by Householder reflections is performed as follows. Let us partition $A$ as
$$A = \left[\begin{array}{c|c} a_{11} & \mathbf{a}^T \\ \hline \mathbf{a} & B \end{array}\right].$$
Let $H$ be the appropriate Householder reflection (see Section 51.5), that is,
$$\mathbf{v} = \mathbf{a} + \operatorname{sign}(a_{21})\|\mathbf{a}\|_2\mathbf{e}_1, \qquad H = I - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}},$$
and let
$$H_1 = \left[\begin{array}{c|c} 1 & \mathbf{0}^T \\ \hline \mathbf{0} & H \end{array}\right].$$
Then
$$H_1AH_1 = \left[\begin{array}{c|c} a_{11} & \mathbf{a}^T H \\ \hline H\mathbf{a} & HBH \end{array}\right] = \left[\begin{array}{c|c} a_{11} & \nu\mathbf{e}_1^T \\ \hline \nu\mathbf{e}_1 & A_1 \end{array}\right], \qquad \nu = -\operatorname{sign}(a_{21})\|\mathbf{a}\|_2.$$
This step annihilates all elements in the first column below the first subdiagonal and all elements in the first row to the right of the first subdiagonal. Applying this procedure recursively yields the triangular matrix $T = X^T AX$, $X = H_1H_2 \cdots H_{n-2}$.

3. $H$ does not depend on the normalization of $\mathbf{v}$. The normalization $v_1 = 1$ is useful since $\mathbf{a}_{2:n}$ can be overwritten by $\mathbf{v}_{2:n}$ and $v_1$ does not need to be stored.
4. Forming $H$ explicitly and then computing $A_1 = HBH$ requires $O(n^3)$ operations, which would ultimately yield an $O(n^4)$ algorithm. However, we do not need to form the matrix $H$ explicitly — given $\mathbf{v}$, we can overwrite $B$ with $HBH$ in just $O(n^2)$ operations by using one matrix-vector multiplication and two rank-one updates.
5. The entire tridiagonalization algorithm is as follows:

```
Algorithm 1: Tridiagonalization by Householder reflections
Input: real symmetric $n \times n$ matrix $A$
Output: the main diagonal and sub- and superdiagonal of $A$ are overwritten by $T$,
        the Householder vectors are stored in the lower triangular part of $A$
        below the first subdiagonal
for $j = 1 : n-2$
    $\mu = \text{sign}(a_{j+1,j})\|A_{j+1:n,j}\|_2$
    if $\mu \neq 0$, then
        $\beta = a_{j+1,j} + \mu$
        $\mathbf{v}_{j+2:n} = A_{j+2:n,j}/\beta$
    endif
    $a_{j+1,j} = -\mu$
    $a_{j,j+1} = -\mu$
    $v_{j+1} = 1$
    $\gamma = -2/\mathbf{v}_{j+1:n}^T\mathbf{v}_{j+1:n}$
    $\mathbf{w} = \gamma A_{j+1:n,j+1:n}\mathbf{v}_{j+1:n}$
    $\mathbf{q} = \mathbf{w} + \frac{1}{2}\gamma\mathbf{v}_{j+1:n}(\mathbf{v}_{j+1:n}^T\mathbf{w})$
    $A_{j+1:n,j+1:n} = A_{j+1:n,j+1:n} + \mathbf{v}_{j+1:n}\mathbf{q}^T + \mathbf{q}\mathbf{v}_{j+1:n}^T$
    $A_{j+2:n,j} = \mathbf{v}_{j+2:n}$
endfor
```

6. [DHS89] When symmetry is exploited in performing rank-2 update, Algorithm 1 requires $4n^3/3$ operations. Another important enhancement is the derivation of the block-version of the algorithm. Instead of performing rank-2 update on $B$, thus obtaining $A_1$, we can accumulate $p$ transformations and perform rank-$2p$ update. In the first $p$ steps, the algorithm is modified to update only columns and rows $1, \ldots, p$, which are needed to compute the first $p$ Householder vectors. Then the matrix $A$ is updated by $A - UV^T - VU^T$, where $U$ and $V$ are $n \times p$ matrices. This algorithm is rich in matrix–matrix multiplications (roughly one half of the operations is performed using BLAS 3 routines), but it requires extra workspace for $U$ and $V$.
7. If the matrix $X$ is needed explicitly, it can be computed from the stored Householder vectors by Algorithm 2. In order to minimize the operation count, the computation starts from the smallest matrix and the size is gradually increased; that is, the algorithm computes the sequence of matrices

$$H_{n-2}, \quad H_{n-3}H_{n-2}, \ldots, \quad X = H_1 \cdots H_{n-2}.$$

A column-oriented version is possible as well, and the operation count in both cases is $4n^3/3$. If the Householder matrices $H_i$ are accumulated in the order in which they are generated, the operation count is $2n^3$.

```
**Algorithm 2:** Computation of the tridiagonalizing matrix $X$
Input: output from Algorithm 1
Output: matrix $X$ such that $X^TAX = T$, where $A$ is the input of Algorithm 1
        and $T$ is tridiagonal
$X = I_n$
for $j = n-2 : -1 : 1$
    $v_{j+1} = 1$
    $\mathbf{v}_{j+2:n} = A_{j+2:n,j}$
    $\gamma = -2/\mathbf{v}^T_{j+1:n}\mathbf{v}_{j+1:n}$
    $\mathbf{w} = \gamma X^T_{j+1:n,j+1:n}\mathbf{v}_{j+1:n}$
    $X_{j+1:n,j+1:n} = X_{j+1:n,j+1:n} + \mathbf{v}_{j+1:n}\mathbf{w}^T$
endfor
```

8. The error bounds for Algorithms 1 and 2 are as follows: The matrix $\tilde{T}$ computed by Algorithm 1 is equal to the matrix, which would be obtained by exact tridiagonalization of some perturbed matrix $A+E$ (backward error), where $\|E\|_2 \leq \psi\epsilon\|A\|_2$ and $\psi$ is a slowly increasing function of $n$. The matrix $\tilde{X}$ computed by Algorithm 2 satisfies $\tilde{X} = X + F$, where $\|F\|_2 \leq \phi\epsilon$ and $\phi$ is a slowly increasing function of $n$.
9. Givens rotation parameters $c$ and $s$ are computed as in Fact 51.5.13. Tridiagonalization by Givens rotations is performed as follows:

```
**Algorithm 3:** Tridiagonalization by Givens rotations
Input: real symmetric $n \times n$ matrix $A$
Output: the matrix $X$ such that $X^TAX = T$ is tridiagonal, main diagonal
        and sub- and superdiagonal of $A$ are overwritten by $T$
$X = I_n$
for $j = 1 : n-2$
    for $i = j+2 : n$
        set $x = a_{j+1,j}$ and $y = a_{i,j}$
        compute $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ via Fact 51.5.13
        $\begin{bmatrix} A_{j+1,j:n} \\ A_{i,j:n} \end{bmatrix} = G \begin{bmatrix} A_{j+1,j:n} \\ A_{i,j:n} \end{bmatrix}$
        $\begin{bmatrix} A_{j:n,j+1} & A_{j:n,i} \end{bmatrix} = \begin{bmatrix} A_{j:n,j+1} & A_{j:n,i} \end{bmatrix} G^T$
        $\begin{bmatrix} X_{1:n,j+1} & X_{1:n,i} \end{bmatrix} = \begin{bmatrix} X_{1:n,j+1} & X_{1:n,i} \end{bmatrix} G^T$
    endfor
endfor
```

10. Algorithm 3 requires $(n-1)(n-2)/2$ plane rotations, which amounts to $4n^3$ operations if symmetry is properly exploited. The operation count is reduced to $8n^3/3$ if fast rotations are used. Fast rotations are obtained by factoring out absolutely larger of $c$ and $s$ from $G$.
11. The Givens rotations in Algorithm 3 can be performed in different orderings. For example, the elements in the first column and row can be annihilated by rotations in the planes $(n-1,n)$, $(n-2,n-1)$, $\ldots(2,3)$. Since Givens rotations act more selectively than Householder reflectors, they can be useful if $A$ has some special structure. For example, Givens rotations are used to efficiently tridiagonalize symmetric band matrices (see Example 4).

12. Error bounds for Algorithm 3 are the same as the ones for Algorithms 1 and 2 (Fact 8), but with slightly different functions $\psi$ and $\phi$.

**Examples:**

1. Algorithms 1 and 2 applied to the matrix $A$ from Example 55.1.1 give

$$T = \begin{bmatrix} 4.5013 & -3.0194 & 0 & 0 \\ -3.0194 & -0.3692 & 1.2804 & 0 \\ 0 & 1.2804 & 0.5243 & -0.9303 \\ 0 & 0 & -0.9303 & 2.6774 \end{bmatrix},$$

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.2028 & 0.4417 & -0.8740 \\ 0 & -0.7091 & -0.6817 & -0.1800 \\ 0 & -0.6753 & 0.5833 & 0.4514 \end{bmatrix}.$$

2. Tridiagonalization is implemented in the Matlab function `T = hess(A)` (`[X,T] = hess(A)` if $X$ is to be computed, as well). In fact, the function `hess` is more general and it computes the Hessenberg form of a general square matrix. For the same matrix $A$ as above, the matrices $T$ and $X$ computed by `hess` are:

$$T = \begin{bmatrix} 2.6562 & 1.3287 & 0 & 0 \\ 1.3287 & 2.4407 & 2.4716 & 0 \\ 0 & 2.4716 & 3.1798 & 2.3796 \\ 0 & 0 & 2.3796 & -0.9429 \end{bmatrix}, \quad X = \begin{bmatrix} 0.4369 & 0.2737 & 0.8569 & 0 \\ 0.7889 & 0.3412 & -0.5112 & 0 \\ -0.4322 & 0.8993 & -0.0668 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}.$$

3. The block version of tridiagonal reduction is implemented in the LAPACK subroutine DSYTRD (file `dsytrd.f`). The computation of $X$ is implemented in the subroutine DORGTR. The size of the required extra workspace (in elements) is $lwork = nb * n$, where $nb$ is the optimal block size (here, $nb = 64$), and it is determined automatically by the subroutines. The timings are given in Section 55.9.
4. Computation of Givens rotation in Algorithm 3 is implemented in the Matlab functions `planerot` and `givens`, BLAS 1 subroutine DROTG, and LAPACK subroutine DLARTG. These implementations avoid unnecessary overflow or underflow by appropriately scaling $x$ and $y$. Plane rotations (multiplications with $G$) are implemented in the BLAS 1 subroutine DROT. LAPACK subroutines DLAR2V, DLARGV, and DLARTV generate and apply multiple plane rotations. LAPACK subroutine DSBTRD tridiagonalizes a symmetric band matrix by using Givens rotations.

## 55.3 Implicitly Shifted QR Method

This method is named after the fact that, for a tridiagonal matrix, each step of the shifted QR iterations given by Eq. (55.5) can be elegantly implemented without explicitly computing the shifted matrix $A_k - \mu I$.

**Definitions:**

**Wilkinson's shift** $\mu$ is the eigenvalue of the bottom right $2 \times 2$ submatrix of $T$, which is closer to $t_{n,n}$.

**Facts:**

The following facts can be found in [GV96, pp. 417–422], [Ste01, pp. 163–171], [TB97, pp. 211–224], [Par80, Chap. 8], [Dem97, Chap. 5.3.1], and [Wil65, Chaps. 8.50, 8.54].

$T = [t_{ij}]$ is a real symmetric tridiagonal matrix of order $n$ and $T = Q\Lambda Q^T$ is its EVD.

1. The stable formula for the Wilkinson's shift is

$$\mu = t_{n,n} - \frac{t_{n,n-1}^2}{\tau + \operatorname{sign}(\tau)\sqrt{\tau^2 + t_{n,n-1}^2}}, \qquad \tau = \frac{t_{n-1,n-1} - t_{n,n}}{2}.$$

2. The following recursive function implements the implicitly shifted QR method given by Eq. (55.5):

```
Algorithm 4: Implicitly shifted QR method for tridiagonal matrices
Input: real symmetric tridiagonal $n \times n$ matrix $T$
Output: the diagonal of $T$ is overwritten by its eigenvalues
function $T = QR\_iteration(T)$
    repeat      % one sweep
        compute a suitable shift $\mu$
        set $x = t_{11} - \mu$ and $y = t_{21}$
        compute $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ via Fact 51.5.13
        $\begin{bmatrix} T_{1,1:3} \\ T_{2,1:3} \end{bmatrix} = G \begin{bmatrix} T_{1,1:3} \\ T_{2,1:3} \end{bmatrix}$
        $\begin{bmatrix} T_{1:3,1} & T_{1:3,2} \end{bmatrix} = \begin{bmatrix} T_{1:3,1} & T_{1:3,2} \end{bmatrix} G^T$
        for $i = 2 : n-1$
            set $x = t_{i,i-1}$ and $y = t_{i+1,i-1}$
            compute $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ via Fact 51.5.13
            $\begin{bmatrix} T_{i,i-1:i+2} \\ T_{i+1,i-1:i+2} \end{bmatrix} = G \begin{bmatrix} T_{i,i-1:i+2} \\ T_{i+1,i-1:i+2} \end{bmatrix}$
            $\begin{bmatrix} T_{i-1:i+2,i} & T_{i-1:i+2,i+1} \end{bmatrix} = \begin{bmatrix} T_{i-1:i+2,i} & T_{i-1:i+2,i+1} \end{bmatrix} G^T$
        endfor
    until $|t_{i,i+1}| \leq \epsilon\sqrt{|t_{i,i} \cdot t_{i+1,i+1}|}$   for some $i$   % deflation
    set $t_{i+1,i} = 0$ and $t_{i,i+1} = 0$
    $T_{1:i,1:i} = QR\_iteration(T_{1:i,1:i})$
    $T_{i+1:n,i+1:n} = QR\_iteration(T_{i+1:n,i+1:n})$
```

3. Wilkinson's shift (Fact 1) is the most commonly used shift. With Wilkinson's shift, the algorithm always converges in the sense that $t_{n-1,n} \to 0$. The convergence is quadratic, that is, $|[T_{k+1}]_{n-1,n}| \leq c|[T_k]_{n-1,n}|^2$ for some constant $c$, where $T_k$ is the matrix after the $k$-th sweep. Even more, the convergence is usually cubic. However, it can also happen that some $t_{i,i+i}$, $i \neq n-1$, becomes sufficiently small before $t_{n-1,n}$, so the practical program has to check for deflation at each step.
4. The plane rotation parameters at the start of the sweep are computed as if the shifted matrix $T - \mu I$ has been formed. Since the rotation is applied to the original $T$ and not to $T - \mu I$, this creates new nonzero elements at the positions $(3,1)$ and $(1,3)$, the so-called **bulge**. The subsequent rotations simply chase the bulge out of the lower

$$
\begin{array}{cccccc}
x & x & * & & & \\
x & x & x & & & \\
* & x & \times & \times & & \\
 & & \times & \times & \times & \\
 & & & \times & \times & \times \\
 & & & & \times & \times
\end{array}
\quad
\begin{array}{cccccc}
\times & x & 0 & & & \\
x & x & x & * & & \\
0 & x & x & x & & \\
 & * & x & \times & \times & \\
 & & & \times & \times & \times \\
 & & & & \times & \times
\end{array}
\quad
\begin{array}{cccccc}
\times & \times & & & & \\
\times & \times & x & 0 & & \\
 & x & x & x & * & \\
 & 0 & x & x & x & \\
 & & * & x & \times & \times \\
 & & & & \times & \times
\end{array}
\quad
\begin{array}{cccccc}
\times & \times & & & & \\
\times & \times & \times & & & \\
 & \times & \times & x & 0 & \\
 & & x & x & x & * \\
 & & 0 & x & x & x \\
 & & & * & x & \times
\end{array}
\quad
\begin{array}{cccccc}
\times & \times & & & & \\
\times & \times & \times & & & \\
 & \times & \times & \times & & \\
 & & \times & \times & x & 0 \\
 & & & x & x & x \\
 & & & 0 & x & x
\end{array}
$$

**FIGURE 55.1** Chasing the bulge in one sweep of the implicit QR iteration for $n = 6$.

right corner of the matrix. The rotation in the $(2, 3)$ plane sets the elements $(3, 1)$ and $(1, 3)$ back to zero, but it generates two new nonzero elements at positions $(4, 2)$ and $(2, 4)$; the rotation in the $(3, 4)$ plane sets the elements $(4, 2)$ and $(2, 4)$ back to zero, but it generates two new nonzero elements at positions $(5, 3)$ and $(3, 5)$, etc. The procedure is illustrated in Figure 55.1: "$x$" denotes the elements that are transformed by the current plane rotation, "$*$" denotes the newly generated nonzero elements (the bulge), and 0 denotes the zeros that are reintroduced by the current plane rotation.

The effect of this procedure is the following. At the end of the first sweep, the resulting matrix $T_1$ is equal to the the matrix that would have been obtained by factorizing $T - \mu I = QR$ and computing $T_1 = RQ + \mu I$ as in Eq. (55.5).

5. Since the convergence of Algorithm 4 is quadratic (or even cubic), an eigenvalue is isolated after just a few steps, which requires $O(n)$ operations. This means that $O(n^2)$ operations are needed to compute all eigenvalues.
6. If the eigenvector matrix $Q$ is desired, the plane rotations need to be accumulated similarly to the accumulation of $X$ in Algorithm 3. This accumulation requires $O(n^3)$ operations (see Example 2 below and Fact 55.9.5). Another, usually faster, algorithm to compute $Q$ is given in Fact 55.9.9.
7. The computed eigenvalue decomposition $T = Q\Lambda Q^T$ satisfies the error bounds from Fact 55.1.15 with $A$ replaced by $T$ and $U$ replaced by $Q$. The deflation criterion implies $|t_{i,i+1}| \leq \epsilon \|T\|_F$, which is within these bounds.
8. Combining Algorithms 1, 2, and 4 we get the the following algorithm:

```
Algorithm 5: Real symmetric eigenvalue decomposition
Input: real symmetric $n \times n$ matrix $A$
Output: eigenvalue matrix $\Lambda$ and, optionally, eigenvector matrix $U$ of $A$
if only eigenvalues are required, then
    Compute $T$ by Algorithm 1
    $T = QR\_iteration(T)$        % Algorithm 4
    $\Lambda = \mathrm{diag}(T)$
else
    Compute $T$ by Algorithm 1
    Compute $X$ by Algorithm 2
    $T = QR\_iteration(T)$        % with rotations accumulated in $Q$
    $\Lambda = \mathrm{diag}(T)$
    $U = XQ$
endif
```

9. The EVD computed by Algorithm 5 satisfies the error bounds given in Fact 55.1.15. However, the algorithm tends to perform better on matrices, which are graded downward, that is, on matrices that exhibit systematic decrease in the size of the matrix elements as we move along the diagonal. For such matrices, the tiny eigenvalues can usually be computed with higher relative accuracy (although counterexamples can be easily constructed). If the tiny eigenvalues are of interest, it should be checked whether

there exists a symmetric permutation that moves larger elements to the upper left corner, thus converting the given matrix to the one that is graded downward.

**Examples:**

1. For the matrix $T$ from Example 55.2.1, after one sweep of Algorithm 4, we have

$$T = \begin{bmatrix} 2.9561 & 3.9469 & 0 & 0 \\ 3.9469 & 0.8069 & -0.7032 & 0 \\ 0 & -0.7032 & 0.5253 & 0.0091 \\ 0 & 0 & 0.0091 & 3.0454 \end{bmatrix}.$$

2. Algorithm 4 is implemented in the LAPACK subroutine DSTEQR. This routine can compute just the eigenvalues, or both eigenvalues and eigenvectors. To avoid double indices, the diagonal and subdiagonal entries of $T$ are stored in one-dimensional vectors, $d_i = T_{ii}$ and $e_i = T_{i+1,i}$, respectively. The timings are given in Section 55.9.
3. Algorithm 5 is implemented in the Matlab routine `eig`. The command `Lambda = eig(A)` returns only the eigenvalues, `[U,Lambda]=eig(A)` returns the eigenvalues and the eigenvectors (see Example 55.1.1).
4. The LAPACK implementation of Algorithm 5 is given in the subroutine DSYEV. To compute only eigenvalues, DSYEV calls DSYTRD and DSTEQR without eigenvector option. To compute both eigenvalues and eigenvectors, DSYEV calls DSYTRD, DORGTR, and DSTEQR with the eigenvector option. The timings are given in Section 55.9.

## 55.4 Divide and Conquer Method

This is currently the fastest method for computing the EVD of a real symmetric tridiagonal matrix $T$. It is based on splitting the given tridiagonal matrix into two matrices, and then computing the EVDs of the smaller matrices and computing the final EVD from the two EVDs. The method was first introduced in [Cup81], but numerically stable and efficient implementation was first derived in [GE95].

**Facts:**

The following facts can be found in [Dem97, pp. 216–228], [Ste01, pp. 171–185], and [GE95]. $T = [t_{ij}]$ is a real symmetric tridiagonal matrix of order $n$ and $T = U\Lambda U^T$ is its EVD.

1. Let $T$ be partitioned as

$$T = \left[\begin{array}{cccc|cccc} d_1 & e_1 & & & & & & \\ e_1 & d_2 & e_2 & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & e_{k-1} & d_k & e_k & & & \\ \hline & & & e_k & d_{k+1} & e_{k+1} & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & e_{n-2} & d_{n-1} & e_{n-1} \\ & & & & & & e_{n-1} & d_n \end{array}\right] \equiv \begin{bmatrix} T_1 & e_k \mathbf{e}_k \mathbf{e}_1^T \\ e_k \mathbf{e}_1 \mathbf{e}_k^T & T_2 \end{bmatrix}.$$

We assume that $T$ is unreduced, that is, $e_i \neq 0$ for all $i$. Further, we assume that

$e_i > 0$ for all $i$, which can be easily attained by diagonal similarity with a diagonal matrix of signs (see Example 1 below). Let

$$\hat{T}_1 = T_1 - e_k \mathbf{e}_k \mathbf{e}_k^T, \qquad \hat{T}_2 = T_2 - e_k \mathbf{e}_1 \mathbf{e}_1^T. \tag{55.6}$$

In other words, $\hat{T}_1$ is equal to $T_1$ except that $d_k$ is replaced by $d_k - e_k$, and $\hat{T}_2$ is equal to $T_2$ except that $d_{k+1}$ is replaced by $d_{k+1} - e_k$.

Let $\hat{T}_i = \hat{U}_i \hat{\Lambda}_i \hat{U}_i^T$, $i = 1, 2$, be the respective EVDs and let $\mathbf{v} = \begin{bmatrix} \hat{U}_1^T \mathbf{e}_k \\ \hat{U}_2^T \mathbf{e}_1 \end{bmatrix}$ ($\mathbf{v}$ consists of the last column of $\hat{U}_1^T$ and the first column of $\hat{U}_2^T$). Set $\hat{U} = \hat{U}_1 \oplus \hat{U}_2$ and $\hat{\Lambda} = \hat{\Lambda}_1 \oplus \hat{\Lambda}_2$. Then

$$T = \begin{bmatrix} \hat{U}_1 & \\ & \hat{U}_2 \end{bmatrix} \left[ \begin{bmatrix} \hat{\Lambda}_1 & \\ & \hat{\Lambda}_2 \end{bmatrix} + e_k \mathbf{v}\mathbf{v}^T \right] \begin{bmatrix} \hat{U}_1^T & \\ & \hat{U}_2^T \end{bmatrix} = \hat{U}(\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T)\hat{U}^T. \tag{55.7}$$

If

$$\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T = X \Lambda X^T$$

is the EVD of the rank-one modification of the diagonal matrix $\hat{\Lambda}$, then $T = U \Lambda U^T$, where $U = \hat{U}X$ is the EVD of $T$. Thus, the original tridiagonal eigenvalue problem is reduced to two smaller tridiagonal eigenvalue problems and one eigenvalue problem for the rank-one update of a diagonal matrix.

2. If the matrix $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ is permuted such that $\hat{\lambda}_1 \geq \cdots \geq \hat{\lambda}_n$, then $\lambda_i$ and $\hat{\lambda}_i$ are interlaced, that is,

$$\lambda_1 \geq \hat{\lambda}_1 \geq \lambda_2 \geq \hat{\lambda}_2 \geq \cdots \geq \lambda_{n-1} \geq \hat{\lambda}_{n-1} \geq \lambda_n \geq \hat{\lambda}_n.$$

   Moreover, if $\hat{\lambda}_{i-1} = \hat{\lambda}_i$ for some $i$, then one eigenvalue is obviously known exactly, that is, $\lambda_i = \hat{\lambda}_i$. In this case, $\lambda_i$ can be deflated by applying to $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ a plane rotation in the $(i-1, i)$ plane, where the Givens rotation parameters $c$ and $s$ are computed from $v_{i-1}$ and $v_i$ as in Fact 51.5.13.

3. If all $\hat{\lambda}_i$ are different, then the eigenvalues $\lambda_i$ of $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ are solutions of the so-called secular equation,

$$1 + e_k \sum_{i=1}^{n} \frac{v_i^2}{\hat{\lambda}_i - \lambda} = 0.$$

   The eigenvalues can be computed by bisection, or by some faster zero finder of the Newton type, and they need to be computed as accurately as possible.

4. Once the eigenvalues $\lambda_i$ of $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ are known, the corresponding eigenvectors are

$$\mathbf{x}_i = (\hat{\Lambda} - \lambda_i I)^{-1} \mathbf{v}.$$

5. Each $\lambda_i$ and $\mathbf{x}_i$ in Facts 3 and 4 is computed in $O(n)$ operations, respectively, so the overall computational cost for computing the EVD of $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ is $O(n^2)$.
6. The accuracy of the computed EVD is given by Fact 55.1.15. However, if some eigenvalues are too close, they may not be computed with sufficient relative accuracy. As a consequence, the eigenvectors computed by using Fact 4 may not be sufficiently orthogonal. One remedy to this problem is to solve the secular equation from Fact 3 in double of the working precision. A better remedy is based on the solution of the following inverse eigenvalue problem. If $\hat{\lambda}_1 > \cdots > \hat{\lambda}_n$ and $\lambda_1 > \hat{\lambda}_1 > \lambda_2 > \hat{\lambda}_2 > \cdots > \lambda_{n-1} > \hat{\lambda}_{n-1} > \lambda_n > \hat{\lambda}_n$, then $\lambda_i$ are the exact eigenvalues of the matrix $\hat{\Lambda} + e_k \hat{\mathbf{v}}\hat{\mathbf{v}}^T$, where

$$\hat{v}_i = \operatorname{sign} v_i \sqrt{\frac{\prod_{j=1}^{n} (\lambda_j - \hat{\lambda}_i)}{\prod_{j=1, j\neq i}^{n} (\hat{\lambda}_j - \hat{\lambda}_i)}}.$$

Instead of computing $\mathbf{x}_i$ according to Fact 4, we compute $\hat{\mathbf{x}}_i = (\hat{\Lambda} - \lambda_i I)^{-1}\hat{\mathbf{v}}$. The eigenvector matrix of $T$ is now computed as $U = \hat{U}\hat{X}$, where $\hat{X} = \begin{bmatrix}\mathbf{x}_1 & \cdots & \mathbf{x}_n\end{bmatrix}$, instead of $U = \hat{U}X$ as in Fact 1. See also Fact 8.

7. The algorithm for the divide and conquer method is the following:

```
Algorithm 6: Divide and conquer method
Input: real symmetric tridiagonal $n \times n$ matrix $T$ with $t_{i-1,i} > 0$ for all $i$
Output: eigenvalue matrix $\Lambda$ and eigenvector matrix $U$ of $T$
 function $(\Lambda, U) = Divide\_and\_Conquer(T)$
    if $n = 1$, then
       $U = 1$
       $\Lambda = T$
    else
       $k = floor(n/2)$
       form $\hat{T}_1$ and $\hat{T}_2 =$ as in Eq. (55.6) in Fact 1
       $(\hat{\Lambda}_1, \hat{U}_1) = Divide\_and\_Conquer(\hat{T}_1)$
       $(\hat{\Lambda}_2, \hat{U}_2) = Divide\_and\_Conquer(\hat{T}_2)$
       form $\hat{\Lambda} + e_k \mathbf{v}\mathbf{v}^T$ as in Eq. (55.7) in Fact 1
       compute the eigenvalues $\lambda_i$ via Fact 3
       compute $\hat{\mathbf{v}}$ via Fact 6
       $\hat{\mathbf{x}}_i = (\hat{\Lambda} - \lambda_i I)^{-1}\hat{\mathbf{v}}$
       $U = \begin{bmatrix}\hat{U}_1 & \\ & \hat{U}_2\end{bmatrix}\hat{X}$
    endif
```

8. The rationale for the approach of Fact 6 and Algorithm 6 is the following: The computations of $\hat{\mathbf{v}}$ and $\hat{\mathbf{x}}_i$ involve only subtractions of exact quantities, so there is no cancellation. Thus, all entries of each $\hat{\mathbf{x}}_i$ are computed with high relative accuracy so $\hat{\mathbf{x}}_i$ are mutually orthonormal to working precision. Also, the transition from the matrix $\hat{\Lambda} + e_k\mathbf{v}\mathbf{v}^T$ to the matrix $\hat{\Lambda} + e_k\hat{\mathbf{v}}\hat{\mathbf{v}}^T$ induces only perturbations that are bounded by $\epsilon\|T\|$. Thus, the EVD computed by Algorithm 6 satisfies the error bounds given in Fact 55.1.15, producing at the same time numerically orthogonal eigenvectors. For details, see [Dem97, pp. 224–226] and [GE95].
9. Although Algorithm 6 requires $O(n^3)$ operations (this is due to the computation of $U$ in the last line), it is in practice usually faster than Algorithm 4 from Fact 55.3.2. This is due to deflations which are performed when solving the secular equation from Fact 3, resulting in matrix $\hat{X}$ having many zeros.
10. The operation count of Algorithm 6 can be reduced to $O(n^2 \log n)$ if the Fast Multipole Method, originally used in particle simulation, is used for solving the secular equation from Fact 3 and for multiplying $\hat{U}\hat{X}$ in the last line of Algorithm 6. For details, see [Dem97, pp. 227–228] and [GE95].

**Examples:**

1. Let $T$ be the matrix from Example 55.2.1 pre- and postmultiplied by the matrix $D = \text{diag}(1, -1, -1, 1)$:

$$T = \begin{bmatrix} 4.5013 & 3.0194 & 0 & 0 \\ 3.0194 & -0.3692 & 1.2804 & 0 \\ 0 & 1.2804 & 0.5243 & 0.9303 \\ 0 & 0 & 0.9303 & 2.6774 \end{bmatrix}.$$

The EVDs of the matrices $\hat{T}_1$ and $\hat{T}_2$ from Eq. (55.6) in Fact 1 are

$$\hat{T}_1 = \begin{bmatrix} 4.5013 & 3.0194 \\ 3.0194 & -1.6496 \end{bmatrix}, \quad \hat{U}_1 = \begin{bmatrix} 0.3784 & -0.9256 \\ -0.9256 & -0.3784 \end{bmatrix}, \quad \hat{\Lambda}_1 = \begin{bmatrix} -2.8841 & 0 \\ 0 & 5.7358 \end{bmatrix},$$
$$\hat{T}_2 = \begin{bmatrix} -0.7561 & 0.9303 \\ 0.9303 & 2.6774 \end{bmatrix}, \quad \hat{U}_2 = \begin{bmatrix} -0.9693 & -0.2458 \\ 0.2458 & -0.9693 \end{bmatrix}, \quad \hat{\Lambda}_2 = \begin{bmatrix} -0.9920 & 0 \\ 0 & 2.9132 \end{bmatrix},$$

so, in Eq. (55.7) in Fact 1, we have

$$\hat{\Lambda} = \operatorname{diag}(-2.8841, 5.7358, -0.9920, 2.9132),$$
$$\mathbf{v} = [-0.9256 \quad -0.3784 \quad -0.9693 \quad -0.2458]^T.$$

2. Algorithm 6 is implemented in the LAPACK subroutine DSTEDC. This routine can compute just the eigenvalues or both eigenvalues and eigenvectors. The routine requires workspace of approximately $n^2$ elements. The timings are given in Section 55.9.

## 55.5 Bisection and Inverse Iteration

The bisection method is convenient if only part of the spectrum is needed. If the eigenvectors are needed as well, they can be efficiently computed by the inverse iteration method (see Facts 55.1.7 and 55.1.8).

**Facts:**

The following facts can be found in [Dem97, pp. 228–213] and [Par80, pp. 65–75].

$A$ is a real symmetric $n \times n$ matrix and $T$ is a real symmetric tridiagonal $n \times n$ matrix.

1. (Sylvester's theorem) For a real nonsingular matrix $X$, the matrices $A$ and $X^T AX$ have the same inertia. (See also Section 9.3.)
2. Let $\alpha, \beta \in \mathbb{R}$ with $\alpha < \beta$. The number of eigenvalues of $A$ in the interval $[\alpha, \beta)$ is equal to $\nu(A - \beta I) - \nu(A - \alpha I)$. By systematically choosing the intervals $[\alpha, \beta)$, the bisection method pinpoints each eigenvalue of $A$ to any desired accuracy.
3. In the factorization $T - \mu I = LDL^T$, where $D = \operatorname{diag}(d_1, \ldots, d_n)$ and $L$ is the unit lower bidiagonal matrix, the elements of $D$ are computed by the recursion

$$d_1 = t_{11} - \mu, \quad d_i = (t_{ii} - \mu) - t_{i,i-1}^2/d_{i-1}, \quad i = 2, \ldots n,$$

and the subdiagonal elements of $L$ are given by $l_{i+1,i} = t_{i+1,i}/d_i$. By Fact 1 the matrices $T$ and $D$ have the same inertia; thus, the above recursion enables an efficient implementation of the bisection method for $T$.
4. The factorization from Fact 3 is essentially Gaussian elimination without pivoting. Nevertheless, if $d_i \neq 0$ for all $i$, the above recursion is very stable (see [Dem97, Lemma 5.4] for details).
5. Even when $d_{i-1} = 0$ for some $i$, if the IEEE arithmetic is used, the computation will continue and the inertia will be computed correctly. Namely, in that case, we would have $d_i = -\infty$, $l_{i+1,i} = 0$, and $d_{i+1} = t_{i+1.i+1} - \mu$. For details, see [Dem97, pp. 230–231] and the references therein.

6. Computing one eigenvalue of $T$ by using the recursion from Fact 3 and bisection requires $O(n)$ operations. For a computed eigenvalue, the corresponding eigenvector is computed by inverse iteration given by Eq. (55.2). The convergence is very fast (Fact 55.1.7), so the cost of computing each eigenvector is also $O(n)$ operations. Therefore, the overall cost for computing all eigenvalues and eigenvectors is $O(n^2)$ operations.
7. Both bisection and inverse iteration are highly parallel since each eigenvalue and eigenvector can be computed independently.
8. If some of the eigenvalues are too close, the corresponding eigenvectors computed by inverse iteration may not be sufficiently orthogonal. In this case, it is necessary to orthogonalize these eigenvectors (for example, by the modified Gram–Schmidt procedure). If the number of close eigenvalues is too large, the overall operation count can increase to $O(n^3)$.
9. The EVD computed by bisection and inverse iteration satisfies the error bounds from Fact 55.1.15.

**Examples:**

1. The bisection method for tridiagonal matrices is implemented in the LAPACK subroutine DSTEBZ. This routine can compute all eigenvalues in a given interval or the eigenvalues from $\lambda_l$ to $\lambda_k$, where $l < k$, and the eigenvalues are ordered from smallest to largest. Inverse iteration (with reorthogonalization) is implemented in the LAPACK subroutine DSTEIN. The timings for computing half of the largest eigenvalues and the corresponding eigenvectors are given in Section 55.9.

## 55.6 Multiple Relatively Robust Representations

The computation of the tridiagonal EVD which satisfies the error bounds of Fact 55.1.15 such that the eigenvectors are orthogonal to working precision, all in $O(n^2)$ operations, has been the “holy grail” of numerical linear algebra for a long time. The method of Multiple Relatively Robust Representations (MRRR) does the job, except in some exceptional cases. The key idea is to implement inverse iteration more carefully. The practical algorithm is quite elaborate and only main ideas are described here.

**Facts:**

The following facts can be found in [Dhi97], [DP04], and [DPV06].

$T = [t_{ij}]$ denotes a real symmetric tridiagonal matrix of order $n$. $D$, $D_+$, and $D_-$ are diagonal matrices with the $i$-th diagonal entry denoted by $d_i$, $D_+(i)$, and $D_-(i)$, respectively. $L$ and $L_+$ are unit lower bidiagonal matrices and $U_-$ is a unit upper bidiagonal matrix, where we denote $(L)_{i+1,i}$ by $l_i$, $(L_+)_{i+1,i}$ by $L_+(i)$, and $(U_-)_{i,i+1}$ by $U_-(i)$.

1. Instead of working with the given $T$, the MRRR method works with the factorization $T = LDL^T$ (computed, for example, as in Fact 55.5.3 with $\mu = 0$). If $T$ is positive definite, then all eigenvalues of $LDL^T$ are determined to high relative accuracy in the sense that small relative changes in the elements of $L$ and $D$ cause only small relative changes in the eigenvalues. If $T$ is indefinite, then the tiny eigenvalues of $LDL^T$ are determined to high relative accuracy in the same sense. The bisection method based on Algorithms 7a and 7b computes the well-determined eigenvalues of $LDL^T$ to high relative accuracy; that is, the computed eigenvalue $\hat{\lambda}$ satisfies $|\lambda - \hat{\lambda}| = O(n\epsilon|\hat{\lambda}|)$.

2. The MRRR method is based on the following three algorithms:

```
**Algorithm 7a:** Differential stationary qd transform
Input: factors $L$ and $D$ of $T$ and the computed eigenvalue $\hat{\lambda}$
Output: matrices $D_+$ and $L_+$ such that $LDL^T - \hat{\lambda}I = L_+D_+L_+^T$ and vector $\mathbf{s}$
$s_1 = -\hat{\lambda}$
for $i = 1 : n-1$
    $D_+(i) = s_i + d_i$
    $L_+(i) = (d_i l_i)/D_+(i)$
    $s_{i+1} = L_+(i) l_i s_i - \hat{\lambda}$
endfor
$D_+(n) = s_n + d_n$
```

```
**Algorithm 7b:** Differential progressive qd transform
Input: factors $L$ and $D$ of $T$ and the computed eigenvalue $\hat{\lambda}$
Output: matrices $D_-$ and $U_-$ such that $LDL^T - \hat{\lambda}I = U_-D_-U_-^T$ and vector $\mathbf{p}$
$p_n = d_n - \hat{\lambda}$
for $i = n-1 : -1 : 1$
    $D_-(i+1) = d_i l_i^2 + p_{i+1}$
    $t = d_i/D_-(i+1)$
    $U_-(i) = l_i t$
    $p_i = p_{i+1} t - \hat{\lambda}$
endfor
$D_-(1) = p_1$
```

```
**Algorithm 7c:** Eigenvector computation
Input: output of Algorithms 7a and 7b and the computed eigenvalue $\hat{\lambda}$
Output: index $r$ and the eigenvector $\mathbf{u}$ such that $LDL^T\mathbf{u} = \hat{\lambda}\mathbf{u}$
for $i = 1 : n-1$
    $\gamma_i = s_i + \frac{d_i}{D_-(i+1)} p_{i+1}$
endfor
$\gamma_n = s_n + p_n + \hat{\lambda}$
find $r$ such that $|\gamma_r| = \min_i |\gamma_i|$
$u_r = 1$
for $i = r-1 : -1 : 1$
    $u_i = -L_+(i) u_{i+1}$
endfor
for $i = r : n-1$
    $u_{i+1} = -U_-(i) u_i$
endfor
$\mathbf{u} = \mathbf{u}/\|\mathbf{u}\|_2$
```

3. Algorithm 7a is accurate in the sense that small relative perturbations (of the order of few $\epsilon$) in the elements $l_i$, $d_i$, and the computed elements $L_+(i)$ and $D_+(i)$ make $LDL^T - \hat{\lambda}I = L_+D_+L_+^T$ an exact equality. Similarly, Algorithm 7b is accurate in the sense that small relative perturbations in the elements $l_i$, $d_i$, and the computed elements $U_-(i)$ and $D_-(i)$ make $LDL^T - \hat{\lambda}I = U_-D_-U_-^T$ an exact equality.
4. The idea behind Algorithm 7c is the following: Index $r$ is the index of the column

of the matrix $(LDL^T - \hat{\lambda}I)^{-1}$ with the largest norm. Since the matrix $LDL^T - \hat{\lambda}I$ is nearly singular, the eigenvector is computed in just one step of inverse iteration given by Eq. (55.2) starting from the vector $\gamma_r \mathbf{e}_r$. Further, $LDL^T - \hat{\lambda}I = N\Delta N^T$, where $N\Delta N^T$ is the the so-called twisted factorization obtained from $L_+$, $D_+$, $U_-$, and $D_-$:

$$\begin{aligned}\Delta &= \operatorname{diag}(D_+(1), \dots, D_+(r-1), \gamma_r, D_-(r+1), \dots, D_-(n)),\\ N_{ii} &= 1,\\ N_{i+1,i} &= L_+(i), \quad i = 1, \dots, r-1,\\ N_{i,i+1} &= U_-(i), \quad i = r, \dots, n-1.\end{aligned}$$

Since $\Delta \mathbf{e}_r = \gamma_r \mathbf{e}_r$ and $N\mathbf{e}_r = \mathbf{e}_r$, solving $N\Delta N^T \mathbf{u} = \gamma_r \mathbf{e}_r$ is equivalent to solving $N^T \mathbf{u} = \mathbf{e}_r$, which is exactly what is done by Algorithm 7c.

5. If an eigenvalue $\lambda$ is well separated from other eigenvalues in the relative sense (the quantity $\min_{\mu \in \sigma(A), \mu \neq \lambda} |\lambda - \mu|/|\lambda|$ is large, say greater than $10^{-3}$), then the computed vector $\hat{\mathbf{u}}$ satisfies $\|\sin\Theta(\mathbf{u}, \hat{\mathbf{u}})\|_2 = O(n\epsilon)$. If all eigenvalues are well separated from each other, then the computed EVD satisfies error bounds of Fact 55.1.15 and the computed eigenvectors are numerically orthogonal, that is, $|\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_j| = O(n\epsilon)$ for $i \neq j$.
6. If there is a cluster of poorly separated eigenvalues which is itself well separated from the rest of $\sigma(A)$, the MRRR method chooses a shift $\mu$ which is near one end of the cluster and computes a new factorization $LDL^T - \mu I = L_+ D_+ L_+^T$. The eigenvalues within the cluster are then recomputed by bisection as in Fact 1 and their corresponding eigenvectors are computed by Algorithms 7a, 7b, and 7c. When properly implemented, this procedure results in the computed EVD, which satisfies the error bounds of Fact 55.1.15 and the computed eigenvectors are numerically orthogonal.

**Examples:**

1. The MRRR method is implemented in the LAPACK subroutine DSTEGR. This routine can compute just the eigenvalues, or both eigenvalues and eigenvectors. The timings are given in Section 55.9.

## 55.7 Jacobi Method

The Jacobi method is the oldest method for EVD computations [Jac846]. The method does not require tridiagonalization. Instead, the method computes a sequence of orthogonally similar matrices which converge to $\Lambda$. In each step a simple plane rotation, which sets one off-diagonal element to zero, is performed.

**Definitions:**

$A$ is a real symmetric matrix of order $x$ and $A = U\Lambda U^T$ is its EVD.

The **Jacobi method** forms a sequence of matrices,

$$A_0 = A, \qquad A_{k+1} = G(i_k, j_k, c, s) A_k G(i_k, j_k, c, s)^T, \qquad k = 1, 2, \dots,$$

where $G(i_k, j_k, c, s)$ is the plane rotation matrix defined in Section 51.5. The parameters $c$ and $s$ are chosen such that $[A_{k+1}]_{i_k j_k} = [A_{k+1}]_{j_k i_k} = 0$ and are computed as described in Fact 1.

The plane rotation with $c$ and $s$ as above is also called the **Jacobi rotation**.

The **off-norm** of $A$ is defined as $\operatorname{off}(A) = (\sum_i \sum_{j \neq i} a_{ij}^2)^{1/2}$, that is, off-norm is the Frobenius norm of the matrix consisting of all off-diagonal elements of $A$.

The choice of **pivot elements** $[A_k]_{i_k j_k}$ is called the **pivoting strategy**.

The **optimal pivoting** strategy, originally used by Jacobi, chooses pivoting elements such that $|[A_k]_{i_k j_k}| = \max_{i<j} |[A_k]_{ij}|$.

The **row cyclic** pivoting strategy chooses pivot elements in the systematic row-wise order,

$$(1,2),(1,3),\ldots,(1,n),(2,3),(2,4),\ldots,(2,n),(3,4),\ldots,(n-1,n).$$

Similarly, the column-cyclic strategy chooses pivot elements column-wise.

One pass through all matrix elements is called **cycle** or **sweep**.

**Facts:**

Facts 1 to 8 can be found in [Wil65, pp. 265–282], [Par80, Chap. 9], [GV96, Chap. 8.4], and [Dem97, Chap. 5.3.5].

1. The Jacobi rotations parameters $c$ and $s$ are computed as follows: If $[A_k]_{i_k j_k} = 0$, then $c = 1$ and $s = 0$, otherwise

$$\tau = \frac{[A_k]_{i_k i_k} - [A_k]_{j_k j_k}}{2[A_k]_{i_k j_k}}, \qquad t = \frac{\operatorname{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}, \qquad c = \frac{1}{\sqrt{1+t^2}}, \qquad s = c \cdot t.$$

2. After each rotation, the off-norm decreases, that is,

$$\operatorname{off}^2(A_{k+1}) = \operatorname{off}^2(A_k) - 2[A_k]^2_{i_k j_k}.$$

   With the appropriate pivoting strategy, the method converges in the sense that

$$\operatorname{off}(A_k) \to 0, \qquad A_k \to \Lambda, \qquad \prod_{k=1}^{\infty} R^T_{(i_k,j_k)} \to U.$$

3. For the optimal pivoting strategy the square of the pivot element is greater than the average squared element, $[A_k]^2_{i_k j_k} \ge \operatorname{off}^2(A)\frac{1}{n(n-1)}$. Thus,

$$\operatorname{off}^2(A_{k+1}) \le \left(1 - \frac{2}{n(n-1)}\right) \operatorname{off}^2(A_k)$$

   and the method converges.
4. For the row cyclic and the column cyclic pivoting strategies, the method converges. The convergence is ultimately quadratic in the sense that

$$\operatorname{off}(A_{k+n(n-1)/2}) \le \gamma \operatorname{off}^2(A_k)$$

   for some constant $\gamma$, provided $\operatorname{off}(A_k)$ is sufficiently small.
5. We have the following algorithm:

```
Algorithm 8: Jacobi method with row-cyclic pivoting strategy
Input: real symmetric $n \times n$ matrix $A$
Output: the eigenvalue matrix $\Lambda$ and the eigenvector matrix $U$
$U = I_n$
repeat      % one cycle
    for $i = 1 : n-1$
        for $j = i+1 : n$
            compute $c$ and $s$ according to Fact 1
            $\begin{bmatrix} A_{i,1:n} \\ A_{j,1:n} \end{bmatrix} = G(i,j,c,s) \begin{bmatrix} A_{i,1:n} \\ A_{j,1:n} \end{bmatrix}$
            $\begin{bmatrix} A_{1:n,i} & A_{1:n,j} \end{bmatrix} = \begin{bmatrix} A_{1:n,i} & A_{1:n,j} \end{bmatrix} G(i,j,c,s)^T$
            $\begin{bmatrix} U_{1:n,i} & U_{1:n,j} \end{bmatrix} = \begin{bmatrix} U_{1:n,i} & U_{1:n,j} \end{bmatrix} G(i,j,c,s)^T$
        endfor
    endfor
until  $\mathrm{off}(A) \leq tol$  for some user defined stopping criterion $tol$
$\Lambda = \mathrm{diag}(A)$
```

6. Detailed implementation of the Jacobi method can be found in [Rut66] and [WR71].
7. The EVD computed by the Jacobi method satisfies the error bounds from Fact 55.1.15.
8. The Jacobi method is suitable for parallel computation. There exist convergent parallel strategies that enable simultaneous execution of several rotations.
9. [GV96, p. 429] The Jacobi method is simple, but it is slower than the methods based on tridiagonalization. It is conjectured that standard implementations require $O(n^3 \log n)$ operations. More precisely, each cycle clearly requires $O(n^3)$ operations and it is conjectured that $\log n$ cycles are needed until convergence.
10. [DV92], [DV07] If $A$ is positive definite, the method can be modified such that it reaches the speed of the methods based on tridiagonalization and at the same time computes the eigenvalues with high relative accuracy. See Chapter 59 for details.

**Examples:**

1. Let $A$ be the matrix from Example 55.1.1. After executing two cycles of Algorithm 8, we have

$$A = \begin{bmatrix} 6.0054 & -0.0192 & 0.0031 & 0.0003 \\ -0.0192 & 3.0455 & -0.0005 & -0.0000 \\ 0.0031 & -0.0005 & 0.6024 & -0.0000 \\ 0.0003 & -0.0000 & 0.0000 & -2.3197 \end{bmatrix}.$$

## 55.8 Lanczos Method

If the matrix $A$ is large and sparse and if only some eigenvalues and their eigenvectors are desired, sparse matrix methods are the methods of choice. For example, the power method can be useful to compute the eigenvalue with the largest modulus. The basic operation in the power method is matrix-vector multiplication, and this can be performed very fast if $A$ is sparse. Moreover, $A$ need not be stored in the computer — the input for the algorithm can be just a program which, given some vector $\mathbf{x}$, computes the product $A\mathbf{x}$. An "improved" version of the power method, which efficiently computes several eigenvalues (either largest in modulus or near some target value $\mu$) and the corresponding eigenvectors, is the Lanczos method.

**Definitions:**

$A$ is a real symmetric matrix of order $n$.

Given a nonzero vector $\mathbf{x}$ and an index $k < n$, the **Krylov matrix** is defined as $K_k = [\mathbf{x} \quad A\mathbf{x} \quad A^2\mathbf{x} \quad \cdots \quad A^{k-1}\mathbf{x}]$.

**Facts:**

The following facts can be found in [Par80, Chap. 13], [GV96, Chap. 9], [Dem97, Chap. 7], and [Ste01, Chap. 5.3].

1. The Lanczos method is based on the following observation. If $K_k = XR$ is the $QR$ factorization of the matrix $K_k$ (see Sections 5.5 and 51.5), then the $k \times k$ matrix $T = X^T AX$ is tridiagonal. The matrices $X$ and $T$ can be computed by using only matrix-vector products in just $O(kn)$ operations. Let $T = Q\Lambda Q^T$ be the EVD of $T$ (computed by any of the methods from Sections 55.3 to 55.6). Then $\lambda_i$ approximate well some of the largest and smallest eigenvalues of $A$. The columns of the matrix $U = XQ$ approximate the corresponding eigenvectors of $A$. We have the following algorithm:

```
Algorithm 9: Lanczos method
Input: real symmetric $n \times n$ matrix $A$, unit vector $\mathbf{x}$ and index $k < n$
Output: matrices $\Lambda$ and $U$
$X_{:,1} = \mathbf{x}$
for $i = 1 : k$
    $\mathbf{z} = A\,X_{:,i}$
    $t_{ii} = X_{:,i}^T\,\mathbf{z}$
    if $i = 1$, then
        $\mathbf{z} = \mathbf{z} - t_{ii}X_{:,i}$
    else
        $\mathbf{z} = \mathbf{z} - t_{ii}X_{:,i} - t_{i,i-1}X_{:,i-1}$
    endif
    $\mu = \|\mathbf{z}\|_2$
    if $\mu = 0$, then
        stop
    else
        $t_{i+1,i} = \mu$
        $t_{i,i+1} = \mu$
        $X_{:,i+1} = z/\mu$
    endif
endfor
compute the EVD of the tridiagonal matrix, $T(1:k,1:k) = Q\Lambda Q^T$
$U = XQ$
```

2. As $j$ increases, the largest (smallest) eigenvalues of the matrix $T_{1:j,1:j}$ converge toward some of the largest (smallest) eigenvalues of $A$ (due to the Cauchy interlace property). The algorithm can be redesigned to compute only largest or smallest eigenvalues. Also, by using shift and invert strategy, the method can be used to compute eigenvalues near some specified value. In order to obtain better approximations, $k$ should be greater than the number of required eigenvalues. On the other side, in order to obtain better accuracy and efficacy, $k$ should be as small as possible (see Facts 3 and 4 below).

3. The eigenvalues of $A$ are approximated from the matrix $T_{1:k,1:k}$; thus, the last element $\nu = t_{k+1,k}$ is not needed. However, this element provides key information about accuracy at no extra computational cost. The exact values of residuals are as follows: $\|AU - U\Lambda\|_2 = \nu$ and, in particular, $\|AU_{:,i} - \lambda_i U_{:,i}\|_2 = \nu|q_{ki}|$, $i = 1, \ldots, k$. Further, there are $k$ eigenvalues $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_k$ of $A$ such that $|\lambda_i - \tilde{\lambda}_i| \leq \nu$. For the corresponding eigenvectors, we have $\sin 2\Theta(\mathbf{u}_i, \tilde{\mathbf{u}}_i) \leq 2\nu / \min_{j \neq i} |\lambda_i - \tilde{\lambda}_j|$. In practical implementations of Algorithm 9, $\nu$ is usually used to determine the index $k$.
4. Although theoretically very elegant, the Lanczos method has inherent numerical instability in the floating point arithmetic, and so it must be implemented carefully (see, e.g., [LSY98]). Since the Krylov vectors are, in fact, generated by the power method, they converge toward an eigenvector of $A$. Thus, as $k$ increases, the Krylov vectors become more and more parallel. As a consequence, the recursion in Algorithm 9, which computes the orthogonal bases $X$ for the subspace range $K_k$, becomes numerically unstable and the computed columns of $X$ cease to be sufficiently orthogonal. This affects both the convergence and the accuracy of the algorithm. For example, it can happen that $T$ has several eigenvalues that converge toward some simple eigenvalue of $A$ (these are the so-called ghost eigenvalues).

   The loss of orthogonality is dealt with by using the full reorthogonalization procedure. In each step, the new $\mathbf{z}$ is orthogonalized against all previous columns of $X$. In Algorithm 9, the formula $\mathbf{z} = \mathbf{z} - t_{ii}X_{:,i} - t_{i,i-1}X_{:,i-1}$ is replaced by $\mathbf{z} = \mathbf{z} - \sum_{j=1}^{i-1}(\mathbf{z}^T X(:,j))X(:,j)$. To obtain better orthogonality, the latter formula is usually executed twice.

   The full reorthogonalization raises the operation count to $O(k^2 n)$. The selective reorthogonalization is the procedure in which the current $\mathbf{z}$ is orthogonalized against some selected columns of $X$. This is the way to attain sufficient numerical stability and not increase the operation count too much. The details of selective reorthogonalization procedures are very subtle and can be found in the references. (See also Chapter 57.)
5. The Lanczos method is usually used for sparse matrices. Sparse matrix $A$ is stored in the sparse format in which only values and indices of nonzero elements are stored. The number of operations required to multiply some vector by $A$ is also proportional to the number of nonzero elements. (See also Chapter 56.)

**Examples:**

1. Let $A$ be the matrix from Example 55.1.1 and let $x = [1/2 \quad 1/2 \quad 1/2 \quad 1/2]^T$. For $k = 2$, the output of Algorithm 9 is

$$\Lambda = \begin{bmatrix} -2.0062 & \\ & 5.7626 \end{bmatrix}, \qquad U = \begin{bmatrix} -0.4032 & -0.8804 \\ 0.4842 & -0.2749 \\ 0.3563 & -0.3622 \\ 0.6899 & -0.1345 \end{bmatrix},$$

   with $\nu = 1.4965$ (c.f. Fact 3). For $k = 3$, the output is

$$\Lambda = \begin{bmatrix} -2.3107 & 0 & 0 \\ 0 & 2.8641 & 0 \\ 0 & 0 & 5.9988 \end{bmatrix}, \qquad U = \begin{bmatrix} 0.3829 & -0.0244 & 0.8982 \\ -0.2739 & -0.9274 & 0.0312 \\ -0.3535 & -0.1176 & 0.3524 \\ -0.8084 & 0.3541 & 0.2607 \end{bmatrix},$$

   with $\nu = 0.6878$.
2. The Lanczos method is implemented in the ARPACK routine `DSDRV*`, where * denotes the computation mode [LSY98, App. A]. The routines from ARPACK are implemented in the

MATLAB command `eigs`. Generation of a sparse symmetric 10,000 × 10,000 matrix with 10% nonzero elements with the MATLAB command `A=sprandsym(10000,0.1)` takes 6.5 seconds on a processor described in Fact 55.9.1. The computation of 100 largest eigenvalues and the corresponding eigenvectors with `[U,Lambda]=eigs(A,100,'LM',opts)` takes approximately 100 seconds. Here, index $k = 200$ is automatically chosen by the algorithm. (See also Chapter 94.)

## 55.9 Comparison of Methods

In this section, we give timings for the LAPACK implementations of the methods described in Sections 55.2 to 55.6. The timing for the Lanczos method is given in Example 55.8.3.

**Definitions:**

A measure of a processor's **efficacy** or **speed** is the number of floating-point operations per second (flops).

**Facts:**

$A$ is an $n \times n$ real symmetric matrix and $A = U\Lambda U^T$ is its EVD. $T$ is a tridiagonal $n \times n$ real symmetric matrix and $T = Q\Lambda Q^T$ is its EVD. $T = X^T AX$ is the reduction of $A$ to a tridiagonal from Section 55.2.

1. Our tests were performed on the Intel64 Xeon Quad Core processor running at 2.33 GHz with 4 Gbytes of RAM and 4 Mbytes of cache memory. This processor performs up to 23 Gflops (23 billion operations per second). This performance is attained for the matrix multiplication with the BLAS 3 subroutine DGEMM.
2. Our test programs were compiled with the Intel `ifort` FORTRAN compiler (version 12.1.2) and linked with the Intel Math Kernel Library (version 10.3.8) in which LAPACK (version 3.3) and BLAS are implemented.
3. Timings for the methods are given in Table 55.1. The execution times for DSTEBZ (bisection) and DSTEIN (inverse iteration) are for computing one-half of the eigenvalues (the largest ones) and the corresponding eigenvectors, respectively. DSTEDC without eigenvectors, DSTEBZ, DSTEIN, and DSTEGR are implemented to use only one core.

**TABLE 55.1** Execution times(s) for LAPACK routines for various matrix dimensions $n$.

| Routine | Input | Output | Example | $n = 1000$ | $n = 2000$ | $n = 4000$ |
|---|---|---|---|---|---|---|
| DSYTRD | $A$ | $T$ | 2.3 | 0.24 | 3.11 | 32.6 |
| DSYTRD/DORGTR | | $T$, $X$ | | 0.44 | 3.92 | 40.8 |
| DSTEQR | $T$ | $\Lambda$ | 3.2 | 0.09 | 0.33 | 1.26 |
| | | $\Lambda$, $Q$ | | 0.62 | 5.03 | 37 |
| DSYEV | $A$ | $\Lambda$ | 3.4 | 0.44 | 2.91 | 15 |
| | | $\Lambda$, $U$ | | 1.04 | 7.32 | 68 |
| DSTEDC | $T$ | $\Lambda$ | 4.2 | 0.07 | 0.24 | 0.95 |
| | | $\Lambda$, $Q$ | | 0.21 | 0.24 | 0.94 |
| DSTEBZ | $T$ | $\Lambda$ | 5.1 | 0.43 | 1.66 | 6.45 |
| DSTEIN | | $Q$ | | 0.1 | 0.43 | 4.01 |
| DSTEGR | $T$ | $\Lambda$ | 6.1 | 0.13 | 0.49 | 1.78 |
| | | $\Lambda$, $Q$ | | 0.30 | 1.22 | 4.7 |

4. The performance attained for practical algorithms is lower than the peak performance from Fact 1. For example, by combining Facts 55.2.6 and 55.2.7 with Table 55.1, we see that the tridiagonalization routines DSYTRD and DORGTR attain the speed of 4 Gflops.
5. The computation times for the implicitly shifted QR routine, DSTEQR, grow approximately with $n^2$ when only eigenvalues are computed, and with $n^3$ when eigenvalues and eigenvectors are computed, as predicted in Facts 55.3.5 and 55.3.6.
6. The execution times for DSYEV are approximately equal to the sums of the timings for DSYTRD (tridiagonalization), DORGTR (computing $X$), and DSTEQR with the eigenvector option (computing the EVD of $T$).
7. The divide and conquer method, implemented in DSTEDC, is the fastest method for computing the EVD of a tridiagonal matrix.
8. DSTEBZ and DSTEIN (bisection and inverse iteration) are faster, especially for larger dimensions, than DSTEQR (tridiagonal QR iteration), but slower than DSTEDC (divide and conquer) and DSTEGR (multiple relatively robust representations).
9. Another algorithm to compute the EVD of $T$ is to use DSTEQR to compute only the eigenvalues and then use DSTEIN (inverse iteration) to compute the eigenvectors. This is usually considerably faster than computing both eigenvalues and eigenvectors by DSTEQR.
10. The execution times for DSTEGR are proportional to $O(n^2)$.

**Author Note:** This work is supported by the Croatian Ministry of Science, Education and Sports (Grant 023-0372783-1289).

## References

[ABB99] E. Anderson, Z. Bai, and C. Bischof. *LAPACK Users' Guide*, 3rd ed. SIAM, Philadelphia, 1999.

[Cup81] J.J.M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36: 177–195, 1981.

[Dem97] J.W. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[DV92] J.W. Demmel and K. Veselić. Jacobi's method is more accurate than QR. *SIAM J. Matrix Anal. Appl.*, 13: 1204–1245, 1992.

[Dhi97] I.S. Dhillon. *A New $O(N^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem.* Ph.D. thesis, University of California, Berkeley, 1997.

[DP04] I.S. Dhillon and B.N. Parlett. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25: 858–899, 2004.

[DPV06] I.S. Dhillon, B.N. Parlett, and C. Vömel. The design and implementation of the MRRR algorithm. *ACM Trans. Math. Software*, 32: 533–560, 2006.

[DHS89] J.J. Dongarra, S.J. Hammarling, and D.C. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comp. Appl. Math.*, 27: 215–227, 1989.

[DV07] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm. I. *SIAM J. Matrix Anal. Appl.*, 29:1322–1342, 2007.

[GV96] G.H. Golub and C.F. Van Loan. *Matrix Computations*, 3rd ed. The John Hopkins University Press, Baltimore, MD, 1996.

[GE95] M. Gu and S.C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16:79–92, 1995.

[Jac846] C.G.J. Jacobi, Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen, *Crelles Journal für Reine und Angew.* Math., 30: 51–95, 1846.

[JSB] N. Jakovčević Stor, I. Slapničar, and J. Barlow. Accurate eigenvalue decomposition of arrowhead matrices. Submitted.

[LSY98] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, 1998.

[Par80] B.N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Upper Saddle River, NJ, 1980.

[Rut66] H. Rutishauser. The Jacobi method for real symmetric matrices, *Numerische Mathematik*, 9: 1–10, 1966.

[Ste01] G.W. Stewart, *Matrix Algorithms, Vol. II: Eigensystems.* SIAM, Philadelphia, 2001.

[TB97] L.N. Trefethen and D. Bau, III. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[Wil65] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, U.K., 1965.

[WR71] J.H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation, Vol. II, Linear Algebra.* Springer, New York, 1971.