Lecture 30. Other Eigenvalue Algorithms

There is more to the computation of eigenvalues than the QR algorithm. In this lecture we briefly mention three famous alternatives for real symmetric eigenvalue problems: the Jacobi algorithm, for full matrices, and the bisection and divide-and-conquer algorithms, for tridiagonal matrices.

Jacobi

One of the oldest ideas for computing eigenvalues of matrices is the *Jacobi algorithm*, introduced by Jacobi in 1845. This method has attracted attention throughout the computer era, especially since the advent of parallel computing, though it has never quite managed to displace the competition.

The idea is as follows. For matrices of dimension 5 or larger, we know that eigenvalues can only be obtained by iteration (Lecture 25). However, smaller matrices than this can be handled in one step. Why not diagonalize a small submatrix of A, then another, and so on, hoping eventually to converge to a diagonalization of the full matrix?

The idea has been tried with 4×4 submatrices, but the standard approach is based on 2×2 submatrices. A 2×2 real symmetric matrix can be diagonalized in the form

$$J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{bmatrix}, \tag{30.1}$$

where J is orthogonal. Now there are several ways to choose J. One could take it to be a 2×2 Householder reflection of the form

$$F = \begin{bmatrix} -c & s \\ s & c \end{bmatrix}, \tag{30.2}$$

where $s = \sin\theta$ and $c = \cos\theta$ for some θ. Note that $\det F = -1$, the hallmark of a reflection. Alternatively, one can use not a reflection but a rotation,

$$J = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \tag{30.3}$$

with $\det J = 1$. This is the standard approach for the Jacobi algorithm. It can be shown that the diagonalization (30.1) is accomplished if θ satisfies

$$\tan(2\theta) = \frac{2d}{b-a}, \tag{30.4}$$

and the matrix J based on this choice is called a *Jacobi rotation*. (It has the same form as a Givens rotation (Exercise 10.4); the only difference is that θ is chosen to make $J^T A J$ diagonal rather than $J^T A$ triangular.)

Now let $A \in \mathbb{R}^{m \times m}$ be symmetric. The Jacobi algorithm consists of the iterative application of transformations (30.1) based on matrices defined by (30.3) and (30.4). The matrix J is now enlarged to an $m \times m$ matrix that is the identity in all but four entries, where it has the form (30.3). Applying J^T on the left modifies two rows of A, and applying J on the right modifies two columns. At each step a symmetric pair of zeros is introduced into the matrix, but previous zeros are destroyed. Just as with the QR algorithm, however, the usual effect is that the magnitudes of these nonzeros shrink steadily.

Which off-diagonal entries a_{ij} should be zeroed at each step? The approach naturally fitted to hand computation is to pick the largest off-diagonal entry at each step. Analysis of convergence then becomes a triviality, for one can show that the sum of the squares of the off-diagonal entries decreases by at least the factor $1 - 2/(m^2 - m)$ at each step (Exercise 30.3). After $O(m^2)$ steps, each requiring $O(m)$ operations, the sum of squares must drop by a constant factor, and convergence to accuracy $\epsilon_{\text{machine}}$ is assured after $O(m^3 \log(\epsilon_{\text{machine}}))$ operations. In fact, it is known that the convergence is better than this, ultimately quadratic rather than linear, so the actual operation count is $O(m^3 \log(|\log(\epsilon_{\text{machine}})|))$ (Exercise 25.2).

On a computer, the off-diagonal entries are generally eliminated in a cyclic manner that avoids the $O(m^2)$ search for the largest. For example, if the $m(m-1)/2$ superdiagonal entries are eliminated in the simplest row-wise order, beginning with $a_{12}, a_{13}, \ldots$, then rapid asymptotic convergence is again guaranteed. After one *sweep* of 2×2 operations involving all of the $m(m-1)/2$ pairs of off-diagonal entries, the accuracy has generally improved by better than a constant factor, and again, the convergence is ultimately quadratic.

The Jacobi method is attractive because it deals only with pairs of rows and columns at a time, making it easily parallelizable (Exercise 30.4). The matrix is not tridiagonalized in advance; the Jacobi rotations would destroy that structure. Convergence for matrices of dimension $m \leq 1000$ is typically achieved in fewer than ten sweeps, and the final componentwise accuracy is generally even better than can be achieved by the QR algorithm. Unfortunately, even on parallel machines, the Jacobi algorithm is not usually as fast as tridiagonalization followed by the QR or divide-and-conquer algorithm (discussed below), though it usually comes within a factor of 10 (Exercise 30.2).

Bisection

Our next eigenvalue algorithm, the method of *bisection*, is of great practical importance. After a symmetric matrix has been tridiagonalized, this is the standard next step if one does not want all of the eigenvalues but just a subset of them. For example, bisection can find the largest 10% of the eigenvalues, or the smallest thirty eigenvalues, or all the eigenvalues in the interval $[1, 2]$. Once the desired eigenvalues are found, the corresponding eigenvectors can be obtained by one step of inverse iteration (Algorithm 27.2).

The starting point is elementary. Since the eigenvalues of a real symmetric matrix are real, we can find them by searching the real line for roots of the polynomial $p(x) = \det(A - xI)$. This sounds like a bad idea, for did we not mention in Lectures 15 and 25 that polynomial rootfinding is a highly unstable procedure for finding eigenvalues? The difference is that those remarks pertained to the idea of finding roots from the polynomial *coefficients*. Now, the idea is to find the roots by evaluating $p(x)$ at various points x, without ever looking at its coefficients, and applying the usual bisection process for nonlinear functions. This could be done, for example, by Gaussian elimination with pivoting (Exercise 21.1), and the resulting algorithm would be highly stable.

This much sounds useful enough, but not very exciting. What gives the bisection method its power and its appeal are some additional properties of eigenvalues and determinants that are not immediately obvious.

Given a symmetric matrix $A \in \mathbb{R}^{m \times m}$, let $A^{(1)}, \ldots, A^{(m)}$ denote its principal (i.e., upper-left) square submatrices of dimensions $1, \ldots, m$. It can be shown that the eigenvalues of these matrices *interlace*. Before defining this property, let us first sharpen it by assuming that A is tridiagonal and *irreducible* in the sense that all of its off-diagonal entries are nonzero:

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{m-1} \\ & & & b_{m-1} & a_m \end{bmatrix}, \qquad b_j \neq 0. \tag{30.5}$$

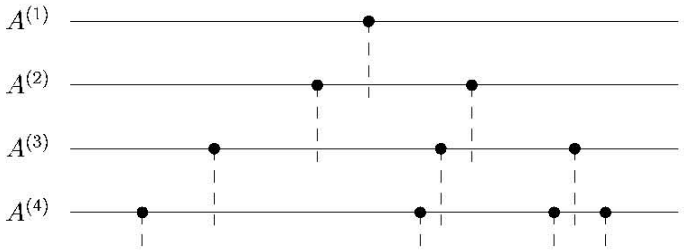


Figure 30.1. *Illustration of the strict eigenvalue interlace property* (30.6) *for the principal submatrices* $\{A^{(j)}\}$ *of an irreducible tridiagonal real symmetric matrix* A. *The eigenvalues of* $A^{(k)}$ *interlace those of* $A^{(k+1)}$. *The bisection algorithm takes advantage of this property.*

(If there are zeros on the off-diagonal, then the eigenvalue problem can be deflated, as in Algorithm 28.2.) By Exercise 25.1, the eigenvalues of $A^{(k)}$ are distinct; let them be denoted by $\lambda_1^{(k)} < \lambda_2^{(k)} < \cdots < \lambda_k^{(k)}$. The crucial property that makes bisection powerful is that these eigenvalues *strictly interlace*, satisfying the inequalities

$$\lambda_j^{(k+1)} < \lambda_j^{(k)} < \lambda_{j+1}^{(k+1)} \tag{30.6}$$

for $k = 1, 2, \ldots, m-1$ and $j = 1, 2, \ldots, k-1$. This behavior is sketched in Figure 30.1.

It is the interlacing property that makes it possible to count the exact number of eigenvalues of a matrix in a specified interval. For example, consider the 4×4 tridiagonal matrix

$$A = \begin{bmatrix} 1 & 1 & & \\ 1 & 0 & 1 & \\ & 1 & 2 & 1 \\ & & 1 & -1 \end{bmatrix}.$$

From the numbers

$$\det(A^{(1)}) = 1, \quad \det(A^{(2)}) = -1, \quad \det(A^{(3)}) = -3, \quad \det(A^{(4)}) = 4,$$

we know that $A^{(1)}$ has no negative eigenvalues, $A^{(2)}$ has one negative eigenvalue, $A^{(3)}$ has one negative eigenvalue, and $A^{(4)}$ has two negative eigenvalues. In general, for any symmetric tridiagonal $A \in \mathbb{R}^{m \times m}$, *the number of negative eigenvalues is equal to the number of sign changes in the sequence*

$$1, \ \det(A^{(1)}), \ \det(A^{(2)}), \ldots, \ \det(A^{(m)}), \tag{30.7}$$

which is known as a *Sturm sequence*. (This prescription works even if zero determinants are encountered along the way, if we define a "sign change" to

mean a transition from + or 0 to − or from − or 0 to + but not from + or − to 0.) By shifting A by a multiple of the identity, we can determine the number of eigenvalues in any interval $[a, b)$: it is the number of eigenvalues in $(-\infty, b)$ minus the number in $(-\infty, a)$.

One more observation completes the description of the bisection algorithm: for a tridiagonal matrix, the determinants of the matrices $\{A^{(k)}\}$ are related by a three-term recurrence relation. Expanding $\det(A^{(k)})$ by minors with respect to its entries b_{k-1} and a_k in row k gives, from (30.5),

$$\det(A^{(k)}) = a_k \det(A^{(k-1)}) - b_{k-1}^2 \det(A^{(k-2)}). \tag{30.8}$$

Introducing the shift by xI and writing $p^{(k)}(x) = \det(A^{(k)} - xI)$, we get

$$p^{(k)}(x) = (a_k - x)p^{(k-1)}(x) - b_{k-1}^2 p^{(k-2)}(x). \tag{30.9}$$

If we define $p^{(-1)}(x) = 0$ and $p^{(0)}(x) = 1$, then this recurrence is valid for all $k = 1, 2, \ldots, m$.

By applying (30.9) for a succession of values of x and counting sign changes along the way, the bisection algorithm locates eigenvalues in arbitrarily small intervals. The cost is $O(m)$ flops for each evaluation of the sequence, hence $O(m \log(\epsilon_{\text{machine}}))$ flops in total to find an eigenvalue to relative accuracy $\epsilon_{\text{machine}}$. If a small number of eigenvalues are needed, this is a distinct improvement over the $O(m^2)$ operation count for the QR algorithm. On a multiprocessor computer, multiple eigenvalues can be found independently on separate processors.

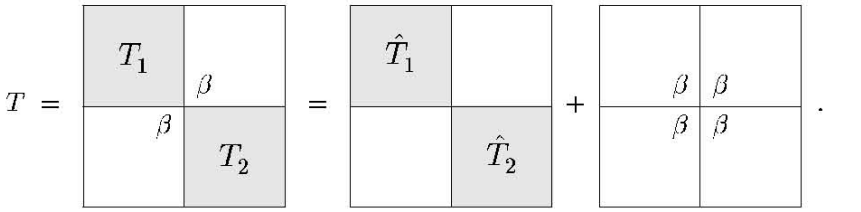
Divide-and-Conquer

The divide-and-conquer algorithm, based on a recursive subdivision of a symmetric tridiagonal eigenvalue problem into problems of smaller dimension, represents the most important advance in matrix eigenvalue algorithms since the 1960s. First introduced by Cuppen in 1981, this method is more than twice as fast as the QR algorithm if eigenvectors as well as eigenvalues are required.

We shall give just the essential idea, omitting all details. But the reader is warned that in this area, the details are particularly important, for the algorithm is not fully stable unless they are gotten right—a matter that was not well understood for a decade after Cuppen's original paper.

Let $T \in \mathbb{R}^{m \times m}$ with $m \geq 2$ be symmetric, tridiagonal, and irreducible in the sense of having only nonzeros on the off-diagonal. (Otherwise, the problem can be deflated.) Then for any n in the range $1 \leq n < m$, T can be split into

submatrices as follows:



(30.10)

Here T_1 is the upper-left $n \times n$ principal submatrix of T, T_2 is the lower-right $(m-n) \times (m-n)$ principal submatrix, and $\beta = t_{n+1,n} = t_{n,n+1} \neq 0$. The only difference between T_1 and $\hat{T}_1$ is that the lower-right entry t_{nn} has been replaced by $t_{nn} - \beta$, and the only difference between T_2 and $\hat{T}_2$ is that the upper-left entry $t_{n+1,n+1}$ has been replaced by $t_{n+1,n+1} - \beta$. These modifications of two entries are introduced to make the rightmost matrix of (30.10) have rank one.

Here is how (30.10) might be expressed in words. *A tridiagonal matrix can be written as the sum of a 2×2 block-diagonal matrix with tridiagonal blocks and a rank-one correction.*

The divide-and-conquer algorithm proceeds as follows. Split the matrix T as in (30.10) with $n \approx m/2$. Suppose the eigenvalues of $\hat{T}_1$ and $\hat{T}_2$ are known. Since the correction matrix is of rank one, a nonlinear but rapid calculation can be used to get from the eigenvalues of $\hat{T}_1$ and $\hat{T}_2$ to those of T itself. Now recurse on this idea, finding the eigenvalues of $\hat{T}_1$ and $\hat{T}_2$ by further subdivisions with rank-one corrections, and so on. In this manner an $m \times m$ eigenvalue problem is reduced to a set of 1×1 eigenvalue problems together with a collection of rank-one corrections. (In practice, for maximal efficiency, it is customary to switch to the QR algorithm when the submatrices are of sufficiently small dimension rather than to carry the recursion all the way.)

In this process there is one key mathematical point. If the eigenvalues of $\hat{T}_1$ and $\hat{T}_2$ are known, how can those of T be found? To answer this, suppose that diagonalizations

$$\hat{T}_1 = Q_1 D_1 Q_1^T, \qquad \hat{T}_2 = Q_2 D_2 Q_2^T$$

have been computed. Then from (30.10) it follows that we have

$$T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \left(\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \beta z z^T \right) \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} \tag{30.11}$$

with $z^T = (q_1^T, q_2^T)$, where q_1^T is the last row of Q_1 and q_2^T is the first row of Q_2. Since this equation is a similarity transformation, we have reduced the mathematical problem to the problem of finding the eigenvalues of a diagonal matrix plus a rank-one correction.

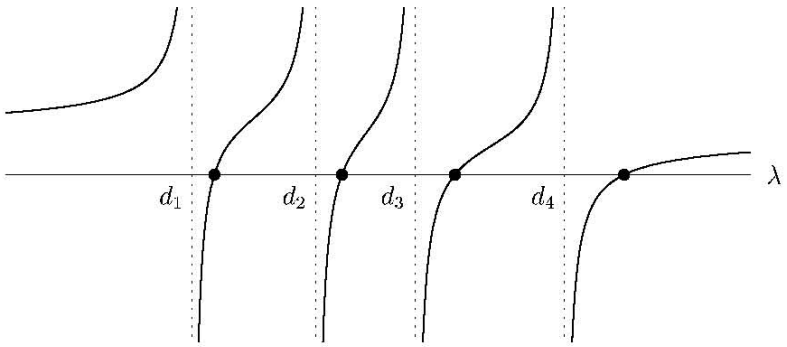


Figure 30.2. *Plot of the function $f(\lambda)$ of (30.12) for a problem of dimension 4. The poles of $f(\lambda)$ are the eigenvalues $\{d_j\}$ of D, and the roots of $f(\lambda)$ (solid dots) are the eigenvalues of $D + ww^T$. The rapid determination of these roots is the basis of each recursive step of the divide-and-conquer algorithm.*

To show how this is done, we simplify notation as follows. Suppose we wish to find the eigenvalues of $D + ww^T$, where $D \in \mathbb{R}^{m\times m}$ is a diagonal matrix with distinct diagonal entries $\{d_j\}$ and $w \in \mathbb{R}^m$ is a vector. (The choice of a plus sign corresponds to $\beta > 0$ above; for $\beta < 0$ we would consider $D - ww^T$.) We can assume $w_j \neq 0$ for all j, for otherwise, the problem is reducible. Then the eigenvalues of $D + ww^T$ are the roots of the rational function

$$f(\lambda) \;=\; 1 + \sum_{j=1}^{m} \frac{w_j^2}{d_j - \lambda}, \tag{30.12}$$

as illustrated in Figure 30.2. This assertion can be justified by noting that if $(D + ww^T)q = \lambda q$ for some $q \neq 0$, then $(D - \lambda I)q + w(w^T q) = 0$, implying $q + (D - \lambda I)^{-1}w(w^T q) = 0$, that is, $w^T q + w^T(D - \lambda I)^{-1}w(w^T q) = 0$. This amounts to the equation $f(\lambda)(w^T q) = 0$, in which $w^T q$ must be nonzero, for otherwise q would be an eigenvector of D, hence nonzero in only one position, implying $w^T q \neq 0$ after all. We conclude that if q is an eigenvector of $D + ww^T$ with eigenvalue λ, then $f(\lambda)$ must be 0, and the converse follows because the form of $f(\lambda)$ guarantees that it has exactly m zeros. The equation $f(\lambda) = 0$ is known as the *secular equation*.

At each recursive step of the divide-and-conquer algorithm, the roots of (30.12) are found by a rapid iterative process related to Newton's method. Only $O(1)$ iterations are required for each root (or $O(\log(|\log(\epsilon_{\text{machine}})|))$ iterations if $\epsilon_{\text{machine}}$ is viewed as a variable), making the operation count $O(m)$ flops per root for an $m \times m$ matrix, or $O(m^2)$ flops all together. If we imagine a recursion in which a matrix of dimension m is split exactly in half at each step, the total operation count for finding eigenvalues of a tridiagonal matrix

by the divide-and-conquer algorithm becomes

$$O\left(m^2 + 2\left(\frac{m}{2}\right)^2 + 4\left(\frac{m}{4}\right)^2 + 8\left(\frac{m}{8}\right)^2 + \cdots + m\left(\frac{m}{m}\right)^2\right), \qquad (30.13)$$

a series which converges to $O(m^2)$ (not $O(m^2 \log m)$) thanks to the squares in the denominators. Thus the operation count would appear to be of the same order $O(m^2)$ as for the QR algorithm.

So far, it is not clear why the divide-and-conquer algorithm is advantageous. Since the reduction of a full matrix to tridiagonal form ("Phase 1" in the terminology of Lecture 25) requires $4m^3/3$ flops (26.2), it would seem that any improvement in the $O(m^2)$ operation count for diagonalization of that tridiagonal matrix ("Phase 2") is hardly important. However, the economics change if one is computing eigenvectors as well as eigenvalues. Now, Phase 1 requires $8m^3/3$ flops but Phase 2 also requires $O(m^3)$ flops—for the QR algorithm, $\approx 6m^3$. The divide-and-conquer algorithm reduces this figure, ultimately because its nonlinear iterations involve just the scalar function (30.12), not the orthogonal matrices Q_j, whereas the QR algorithm must manipulate matrices Q_j at every iterative step.

An operation count reveals the following. The $O(m^3)$ part of the divide-and-conquer computation is the multiplication by Q_j and Q_j^T in (30.11). The total operation count, summed over all steps of the recursion, is $4m^3/3$ flops, a great improvement over $\approx 6m^3$ flops. Adding in the $8m^3/3$ flops for Phase 1 gives an improvement from $\approx 9m^3$ to $4m^3$.

Actually, the divide-and-conquer algorithm usually does even better than this, for a reason that is not elementary. For most matrices A, many of the vectors z and matrices Q_j that arise in (30.11) turn out to be numerically sparse in the sense that many of their entries have relative magnitudes less than machine precision. This sparsity allows a process of *numerical deflation*, whereby successive tridiagonal eigenvalue problems are reduced to uncoupled problems of smaller dimensions. In typical cases this reduces the Phase 2 operation count to an order less than m^3 flops, reducing the operation count for Phases 1 and 2 combined to $8m^3/3$. For eigenvalues alone, (30.13) becomes an overestimate and the Phase 2 operation count is reduced to an order lower than m^2 flops. The root of this fascinating phenomenon of deflation, which we shall not discuss further, is the fact that most of the eigenvectors of most tridiagonal matrices are "exponentially localized" (Exercise 30.7)—a fact that has been related by physicists to the phenomenon that glass is transparent.

We have spoken as if there is a single divide-and-conquer algorithm, but in fact, there are many variants. More complicated rank-one updates are often used for stability reasons, and rank-two updates are also sometimes used. Various methods are employed for finding the roots of $f(\lambda)$, and for large m, the fastest way to carry out the multiplications by Q_j is via multipole expansions rather than the obvious algorithm. A high-quality implementation of a divide-and-conquer algorithm can be found in the LAPACK library.

Exercises

30.1. Derive the formula (30.4), and give a precise geometric interpretation of the transformation (30.1) based on this choice of θ.

30.2. How many flops are required for one step (30.1) of the Jacobi algorithm? How many flops for $m(m-1)/2$ such steps, i.e., one sweep? How does the operation count for one sweep compare with the total operation count for tridiagonalizing a real symmetric matrix and finding its eigenvalues by the QR algorithm?

30.3. Show that if the largest off-diagonal entry is annihilated at each step of the Jacobi algorithm, then the sum of the squares of the off-diagonal entries decreases by at least the factor $1 - 2/(m^2 - m)$ at each step.

30.4. Suppose m is even and your computer has $m/2$ processors. Explain how $m/2$ transformations (30.1) can be carried out in parallel if they involve the disjoint row/column pairs $(1,2), (3,4), (5,6), \ldots, (m-1, m)$.

30.5. Write a program to find the eigenvalues of an $m \times m$ real symmetric matrix by the Jacobi algorithm with the standard row-wise ordering, plotting the sum of the squares of the off-diagonal entries on a log scale as a function of the number of sweeps. Apply your program to random matrices of dimensions 20, 40, and 80.

30.6. How many eigenvalues does

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

have in the interval $[1, 2]$? Work out the answer on paper by bisection, making use of the recurrence (30.9).

30.7. Construct a random real symmetric tridiagonal matrix T of dimension 100 and compute its eigenvalue decomposition, $T = QDQ^T$. Plot a few of the eigenvectors on a log scale (the absolute values of a few columns of Q) and observe the phenomenon of localization. What proportion of the 10,000 entries of Q are greater than 10^{-10} in magnitude? What is the answer if instead of a random matrix, T is the discrete Laplacian with entries $1, -2, 1$?