5

The Symmetric Eigenproblem and Singular Value Decomposition

5.1. Introduction

We discuss perturbation theory (in section 5.2), algorithms (in sections 5.3 and 5.4), and applications (in section 5.5 and elsewhere) of the symmetric eigenvalue problem. We also discuss its close relative, the SVD. Since the eigendecomposition of the symmetric matrix $H = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$ and the SVD of A are very simply related (see Theorem 3.3), most of the perturbation theorems and algorithms for the symmetric eigenproblem extend to the SVD.

As discussed at the beginning of Chapter 4, one can roughly divide the algorithms for the symmetric eigenproblem (and SVD) into two groups: *direct methods* and *iterative methods*. This chapter considers only direct methods, which are intended to compute all (or a selected subset) of the eigenvalues and (optionally) eigenvectors, costing $O(n^3)$ operations for dense matrices. Iterative methods are discussed in Chapter 7.

Since there has been a great deal of recent progress in algorithms and applications of symmetric eigenproblems, we will highlight three examples:

- A high-speed algorithm for the symmetric eigenproblem based on divide-and-conquer is discussed in section 5.3.3. This is the fastest available algorithm for finding all eigenvalues and all eigenvectors of a large dense or banded symmetric matrix (or the SVD of a general matrix). It is significantly faster than the previous "workhorse" algorithm, QR iteration. [16]

- High-accuracy algorithms based on the dqds and Jacobi algorithms are discussed in sections 5.2.1, 5.4.2, and 5.4.3. These algorithms can find

[16] There is yet more recent work [199, 201] on an algorithm based on inverse iteration (Algorithm 4.2), which may provide a still faster and more accurate algorithm. But as of September 1996 the theory and software were still under development.

tiny eigenvalues (or singular values) more accurately than alternative algorithms like divide-and-conquer, although sometimes more slowly.

- Section 5.5 discusses a "nonlinear" vibrating system, described by a differential equation called the *Toda flow*. Its continuous solution is closely related to the intermediate steps of the QR algorithm for the symmetric eigenproblem.

Following Chapter 4, we will continue to use a vibrating mass-spring system as a running example to illustrate features of the symmetric eigenproblem.

EXAMPLE 5.1. Symmetric eigenvalue problems often arise in analyzing *mechanical vibrations*. Example 4.1 presented one such example in detail; we will use notation from that example, so the reader is advised to review it now. To make the problem in Example 4.1 symmetric, we need to assume that there is no damping, so the differential equations of motion of the mass-spring system become $M\ddot{x}(t) = -Kx(t)$, where $M = \mathrm{diag}(m_1, \ldots, m_n)$ and

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & & & \\ -k_2 & k_2 + k_3 & -k_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -k_{n-1} & k_{n-1} + k_n & -k_n \\ & & & -k_n & k_n \end{bmatrix}.$$

Since M is nonsingular, we can rewrite this as $\ddot{x}(t) = -M^{-1}Kx(t)$. If we seek solutions of the form $x(t) = e^{\gamma t}x(0)$, then we get $e^{\gamma t}\gamma^2 x(0) = -M^{-1}Ke^{\gamma t}x(0)$, or $M^{-1}Kx(0) = -\gamma^2 x(0)$. In other words, $-\gamma^2$ is an eigenvalue and $x(0)$ is an eigenvector of $M^{-1}K$. Now $M^{-1}K$ is not generally symmetric, but we can make it symmetric as follows. Define $M^{1/2} = \mathrm{diag}(m_1^{1/2}, \ldots, m_n^{1/2})$, and multiply $M^{-1}Kx(0) = -\gamma^2 x(0)$ by $M^{1/2}$ on both sides to get

$$M^{-1/2}Kx(0) = M^{-1/2}K(M^{-1/2}M^{1/2})x(0) = -\gamma^2 M^{1/2}x(0)$$

or $\hat{K}\hat{x} = -\gamma^2\hat{x}$, where $\hat{x} = M^{1/2}x(0)$ and $\hat{K} = M^{-1/2}KM^{-1/2}$. It is easy to see that

$$\hat{K} = \begin{bmatrix} \frac{k_1+k_2}{m_1} & \frac{-k_2}{\sqrt{m_1 m_2}} & & & \\ \frac{-k_2}{\sqrt{m_1 m_2}} & \frac{k_2+k_3}{m_2} & \frac{-k_3}{\sqrt{m_2 m_3}} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{-k_{n-1}}{\sqrt{m_{n-2} m_{n-1}}} & \frac{k_{n-1}+k_n}{m_{n-1}} & \frac{-k_n}{\sqrt{m_{n-1} m_n}} \\ & & & \frac{-k_n}{\sqrt{m_{n-1} m_n}} & \frac{k_n}{m_n} \end{bmatrix}$$

is symmetric. Thus each eigenvalue $-\gamma^2$ of $\hat{K}$ is real, and each eigenvector $\hat{x} = M^{1/2}x(0)$ of $\hat{K}$ is orthogonal to the others.

In fact, $\hat{K}$ is a *tridiagonal* matrix, a special form to which any symmetric matrix can be reduced, using Algorithm 4.6, specialized to symmetric matrices as described in section 4.4.7. Most of the algorithms in section 5.3 for finding the eigenvalues and eigenvectors of a symmetric matrix assume that the matrix has initially been reduced to tridiagonal form.

There is another way to express the solution to this mechanical vibration problem, using the SVD. Define $K_D = \mathrm{diag}(k_1, \ldots, k_n)$ and $K_D^{1/2} = \mathrm{diag}(k_1^{1/2}, \ldots, k_n^{1/2})$. Then K can be factored as $K = BK_DB^T$, where

$$B = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & \ddots & -1 \\ & & & 1 \end{bmatrix},$$

as can be confirmed by a small calculation. Thus

$$\begin{aligned} \hat{K} &= M^{-1/2}KM^{-1/2} \\ &= M^{-1/2}BK_DB^TM^{-1/2} \\ &= (M^{-1/2}BK_D^{1/2}) \cdot (K_D^{1/2}B^TM^{-1/2}) \\ &= (M^{-1/2}BK_D^{1/2}) \cdot (M^{-1/2}BK_D^{1/2})^T \\ &\equiv GG^T. \end{aligned} \tag{5.1}$$

Therefore the singular values of $G = M^{-1/2}BK_D^{1/2}$ are the square roots of the eigenvalues of $\hat{K}$, and the left singular vectors of G are the eigenvectors of $\hat{K}$, as shown in Theorem 3.3. Note that G is nonzero only on the main diagonal and on the first superdiagonal. Such matrices are called *bidiagonal*, and most algorithms for the SVD begin by reducing the matrix to bidiagonal form, using the algorithm in section 4.4.7.

Note that the factorization $\hat{K} = GG^T$ implies that $\hat{K}$ is positive definite, since G is nonsingular. Therefore the eigenvalues $-\gamma^2$ of $\hat{K}$ are all positive. Thus γ is pure imaginary, and the solutions of the original differential equation $x(t) = e^{\gamma t}x(0)$ are oscillatory with frequency $|\gamma|$.

For a Matlab solution of a vibrating mass-spring system, see HOMEPAGE/Matlab/massspring.m. For a Matlab animation of the vibrations of a similar physical system, see demo/continue/fun-extras/miscellaneous/bending. $\diamond$

5.2. Perturbation Theory

Suppose that A is symmetric, with eigenvalues $\alpha_1 \geq \cdots \geq \alpha_n$ and corresponding unit eigenvectors $q_1, \ldots, q_n$. Suppose E is also symmetric, and let $\hat{A} = A + E$ have perturbed eigenvalues $\hat{\alpha}_1 \geq \cdots \geq \hat{\alpha}_n$ and corresponding perturbed eigenvectors $\hat{q}_1, \ldots, \hat{q}_n$. The major goal of this section is to bound the

differences between the eigenvalues α_i and $\hat{\alpha}_i$, and between the eigenvectors q_i and $\hat{q}_i$ in terms of the "size" of E. Most of our bounds will use $\|E\|_2$ as the size of E, except for section 5.2.1, which discusses "relative" perturbation theory.

We already derived our first perturbation bound for eigenvalues in Chapter 4, where we proved Corollary 4.1: *Let A be symmetric with eigenvalues $\alpha_1 \geq \cdots \geq \alpha_n$. Let $A + E$ be symmetric with eigenvalues $\hat{\alpha}_1 \geq \cdots \geq \hat{\alpha}_n$. If α_i is simple, then $|\alpha_i - \hat{\alpha}_i| \leq \|E\|_2 + O(\|E\|_2^2)$.*

This result is weak because it assumes α_i has multiplicity one, and it is useful only for sufficiently small $\|E\|_2$. The next theorem eliminates both weaknesses.

THEOREM 5.1. Weyl. *Let A and E be n-by-n symmetric matrices. Let $\alpha_1 \geq \cdots \geq \alpha_n$ be the eigenvalues of A and $\hat{\alpha}_1 \geq \cdots \geq \hat{\alpha}_n$ be the eigenvalues of $\hat{A} = A + E$. Then $|\alpha_i - \hat{\alpha}_i| \leq \|E\|_2$.*

COROLLARY 5.1. *Let G and F be arbitrary matrices (of the same size) where $\sigma_1 \geq \cdots \geq \sigma_n$ are the singular values of G and $\sigma'_1 \geq \cdots \geq \sigma'_n$ are the singular values of $G + F$. Then $|\sigma_i - \sigma'_i| \leq \|F\|_2$.*

We can use Weyl's theorem to get error bounds for the eigenvalues computed by any backward stable algorithm, such as QR iteration: Such an algorithm computes eigenvalues $\hat{\alpha}_i$ that are the exact eigenvalues of $\hat{A} = A + E$ where $\|E\|_2 = O(\varepsilon)\|A\|_2$. Therefore, their errors can be bounded by $|\alpha_i - \hat{\alpha}_i| \leq \|E\|_2 = O(\varepsilon)\|A\|_2 = O(\varepsilon) \max_j |\alpha_j|$. This is a very satisfactory error bound, especially for large eigenvalues (those α_i near $\|A\|_2$ in magnitude), since they will be computed with most of their digits correct. Small eigenvalues ($|\alpha_i| \ll \|A\|_2$) may have fewer correct digits (but see section 5.2.1).

We will prove Weyl's theorem using another useful classical result: the Courant–Fischer minimax theorem. To state this theorem we need to introduce the *Rayleigh quotient*, which will also play an important role in several algorithms, such as Algorithm 5.1.

DEFINITION 5.1. *The* Rayleigh quotient *of a symmetric matrix A and nonzero vector u is $\rho(u, A) \equiv (u^T A u)/(u^T u)$.*

Here are some simple but important properties of $\rho(u, A)$. First, $\rho(\gamma u, A) = \rho(u, A)$ for any nonzero scalar γ. Second, if $Aq_i = \alpha_i q_i$, then $\rho(q_i, A) = \alpha_i$. More generally, suppose $Q^T A Q = \Lambda = \mathrm{diag}(\alpha_i)$ is the eigendecomposition of A, with $Q = [q_1, \ldots, q_n]$. Expand u in the basis of eigenvectors q_i as follows: $u = Q(Q^T u) \equiv Q\xi = \sum_i q_i \xi_i$. Then we can write

$$\rho(u, A) = \frac{\xi^T Q^T A Q \xi}{\xi^T Q^T Q \xi} = \frac{\xi^T \Lambda \xi}{\xi^T \xi} = \frac{\sum_i \alpha_i \xi_i^2}{\sum_i \xi_i^2}.$$

In other words, $\rho(u, A)$ is a weighted average of the eigenvalues of A. Its largest value, $\max_{u=0} \rho(u, A)$, occurs for $u = q_1$ ($\xi = e_1$) and equals $\rho(q_1, A) = \alpha_1$.

Its smallest value, $\min_{u=0} \rho(u, A)$, occurs for $u = q_n$ ($\xi = e_n$) and equals $\rho(q_n, A) = \alpha_n$. Together, these facts imply

$$\max_{u=0} |\rho(u, A)| = \max(|\alpha_1|, |\alpha_n|) = \|A\|_2. \tag{5.2}$$

THEOREM 5.2. Courant–Fischer minimax theorem. *Let $\alpha_1 \geq \cdots \geq \alpha_n$ be eigenvalues of the symmetric matrix A and $q_1, \ldots, q_n$ be the corresponding unit eigenvectors.*

$$\max_{\mathbf{R}^j} \min_{0=r\in\mathbf{R}^j} \rho(r, A) = \alpha_j = \min_{\mathbf{S}^{n-j+1}} \max_{0=s\in\mathbf{S}^{n-j+1}} \rho(s, A).$$

The maximum in the first expression for α_j is over all j dimensional subspaces $\mathbf{R}^j$ of $\mathbb{R}^n$, and the subsequent minimum is over all nonzero vectors r in the subspace. The maximum is attained for $\mathbf{R}^j = \operatorname{span}(q_1, q_2, \ldots, q_j)$, and a minimizing r is $r = q_j$.

The minimum in the second expression for α_j is over all $(n - j + 1)$-dimensional subspaces $\mathbf{S}^{n-j+1}$ of $\mathbb{R}^n$, and the subsequent maximum is over all nonzero vectors s in the subspace. The minimum is attained for $\mathbf{S}^{n-j+1} = \operatorname{span}(q_j, q_{j+1}, \ldots, q_n)$, and a maximizing s is $s = q_j$.

EXAMPLE 5.2. Let $j = 1$, so α_j is the largest eigenvalue. Given $\mathbf{R}^1$, $\rho(r, A)$ is the same for all nonzero $r \in \mathbf{R}^1$, since all such r are scalar multiples of one another. Thus the first expression for α_1 simplifies to $\alpha_1 = \max_{r=0} \rho(r, A)$. Similarly, since $n - j + 1 = n$, the only subspace $\mathbf{S}^{n-j+1}$ is $\mathbb{R}^n$, the whole space. Then the second expression for α_1 also simplies to $\alpha_1 = \max_{s=0} \rho(s, A)$.

One can similarly show that the theorem simplifies to the following expression for the smallest eigenvalue: $\alpha_n = \min_{r=0} \rho(r, A)$. $\diamond$

Proof of the Courant–Fischer minimax theorem. Choose any subspaces $\mathbf{R}^j$ and $\mathbf{S}^{n-j+1}$ of the indicated dimensions. Since the sum of their dimensions $j + (n - j + 1) = n + 1$ exceeds n, there must be a nonzero vector $x_{\mathbf{RS}} \in \mathbf{R}^j \cap \mathbf{S}^{n-j+1}$. Thus

$$\min_{0=r\in\mathbf{R}^j} \rho(r, A) \leq \rho(x_{\mathbf{RS}}, A) \leq \max_{0=s\in\mathbf{S}^{n-j+1}} \rho(s, A).$$

Now choose $\hat{\mathbf{R}}^j$ to maximize the expression on the left, and choose $\hat{\mathbf{S}}^{n-j+1}$ to minimize the expression on the right. Then

$$\begin{aligned} \max_{\mathbf{R}^j} \min_{0=r\in\mathbf{R}^j} \rho(r, A) &= \min_{0=r\in\hat{\mathbf{R}}^j} \rho(r, A) \\ &\leq \rho(x_{\hat{\mathbf{R}}\hat{\mathbf{S}}}, A) \\ &\leq \max_{0=s\in\hat{\mathbf{S}}^{n-j+1}} \rho(s, A) \\ &= \min_{\mathbf{S}^{n-j+1}} \max_{0=s\in\mathbf{S}^{n-j+1}} \rho(s, A). \end{aligned} \tag{5.3}$$

To see that all these inequalities are actually equalities, we exhibit particular $\mathbf{R}^j$ and $\mathbf{S}^{n-j+1}$ that make the lower bound equal the upper bound. First choose $\underline{\mathbf{R}}^j = \text{span}(q_1, \ldots, q_j)$, so that

$$\begin{aligned} \max_{\mathbf{R}^j} \min_{0=r\in\mathbf{R}^j} \rho(r,A) &\geq \min_{0=r\in\underline{\mathbf{R}}^j} \rho(r,A) \\ &= \min_{0=r=\sum_{i\leq j}\xi_i q_i} \rho(r,A) \\ &= \min_{\text{some } \xi_i=0} \frac{\sum_{i\leq j}\xi_i^2\alpha_i}{\sum_{i\leq j}\xi_i^2} = \alpha_j. \end{aligned}$$

Next choose $\bar{\mathbf{S}}^{n-j+1} = \text{span}(q_j, \ldots, q_n)$ so that

$$\begin{aligned} \min_{\mathbf{S}^{n-j+1}} \max_{0=s\in\mathbf{S}^{n-j+1}} \rho(s,A) &\leq \max_{0=s\in\bar{\mathbf{S}}^{n-j+1}} \rho(s,A) \\ &= \max_{0=s=\sum_{i\geq j}\xi_i q_i} \rho(s,A) \\ &= \max_{\text{some } \xi_i=0} \frac{\sum_{i\geq j}\xi_i^2\alpha_i}{\sum_{i\geq j}\xi_i^2} = \alpha_j. \end{aligned}$$

Thus, the lower and upper bounds are sandwiched between α_j below and α_j above, so they must all equal α_j as desired. □

EXAMPLE 5.3. Figure 5.1 illustrates this theorem graphically for 3-by-3 matrices. Since $\rho(u/\|u\|_2, A) = \rho(u, A)$, we can think of $\rho(u, A)$ as a function on the unit sphere $\|u\|_2 = 1$. Figure 5.1 shows a contour plot of this function on the unit sphere for $A = \text{diag}(1, .25, 0)$. For this simple matrix $q_i = e_i$, the ith column of the identity matrix. The figure is symmetric about the origin since $\rho(u, A) = \rho(-u, A)$. The small red circles near $\pm q_1$ surround the global maximum $\rho(\pm q_1, A) = 1$, and the small green circles near $\pm q_3$ surround the global minimum $\rho(\pm q_3, A) = 0$. The two great circles are contours for $\rho(u, A) = .25$, the second eigenvalue. Within the two narrow (green) "apple slices" defined by the great circles, $\rho(u, A) < .25$, and within the wide (red) apple slices, $\rho(u, A) > .25$.

Let us interpret the minimax theorem in terms of this figure. Choosing a space $\mathbf{R}^2$ is equivalent to choosing a great circle C; every point on C lies within $\mathbf{R}^2$, and $\mathbf{R}^2$ consists of all scalar multiplicatons of the vectors in C. Thus $\min_{0=r\in\mathbf{R}^2} \rho(r, A) = \min_{r\in C} \rho(r, A)$. There are four cases to consider to compute $\min_{r\in C} \rho(r, A)$:

1. C does *not* go through the intersection points $\pm q_2$ of the two great circles in Figure 5.1. Then C clearly must intersect both a narrow green apple slice (as well as a wide red apple slice), so $\min_{r\in C} \rho(r, A) < .25$.

2. C does go through the two intersection points $\pm q_2$ and otherwise lies in the narrow green apple slices. Then $\min_{r\in C} \rho(r, A) < .25$.

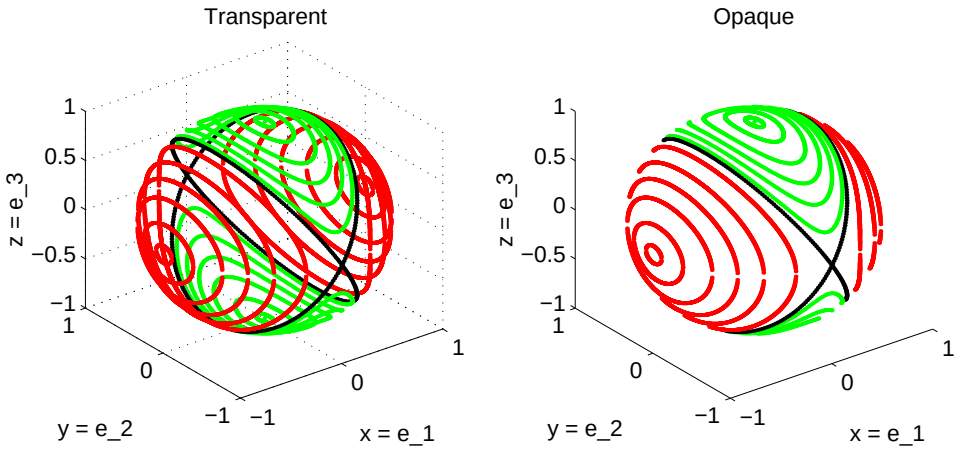


Fig. 5.1. *Contour plot of the Rayleigh quotient on the unit sphere.*

3. C does go through the two intersection points $\pm q_2$ and otherwise lies in the wide red apple slices. Then $\min_{r \in C} \rho(r, A) = .25$, attained for $r = \pm q_2$.

4. C coincides with one of the two great circles. Then $\rho(r, A) = .25$ for all $r \in C$.

The minimax theorem says that $\alpha_2 = .25$ is the maximum of $\min_{r \in C} \rho(r, A)$ over all choices of great circle C. This maximum is attained in cases 3 and 4 above. In particular, for C bisecting the wide red apple slices (case 3), $\mathbf{R}^2 = \mathrm{span}(q_1, q_2)$.

Software to draw contour plots like those in Figure 5.1 for an arbitrary 3-by-3 symmetric matrix may be found at HOMEPAGE/Matlab/RayleighContour.m. $\diamond$

Finally, we can present the *proof of Weyl's theorem.*

$$\begin{aligned}
\hat{\alpha}_i &= \min_{\mathbf{S}^{n-j+1}} \max_{0 = u \in \mathbf{S}^{n-j+1}} \frac{u^T (A+E) u}{u^T u} && \text{by the minimax theorem} \\
&= \min_{\mathbf{S}^{n-j+1}} \max_{0 = u \in \mathbf{S}^{n-j+1}} \left(\frac{u^T A u}{u^T u} + \frac{u^T E u}{u^T u} \right) \\
&\leq \min_{\mathbf{S}^{n-j+1}} \max_{0 = u \in \mathbf{S}^{n-j+1}} \left(\frac{u^T A u}{u^T u} + \|E\|_2 \right) && \text{by equation (5.2)} \\
&= \alpha_i + \|E\|_2 && \text{by the minimax theorem again.}
\end{aligned}$$

Reversing the roles of A and $A + E$, we also get $\alpha_i \leq \hat{\alpha}_i + \|E\|_2$. Together, these two inequalities complete the proof of Weyl's theorem. $\square$

A theorem closely related to the Courant–Fischer minimax theorem, one that we will need later to justify the Bisection algorithm in section 5.3.4, is Sylvester's theorem of inertia.

DEFINITION 5.2. *The* inertia *of a symmetric matrix A is the triple of integers* $\text{Inertia}(A) \equiv (\nu, \zeta, \pi)$*, where ν is the number of negative eigenvalues of A, ζ is the number of zero eigenvalues of A, and π is the number of positive eigenvalues of A.*

If X is orthogonal, then $X^T AX$ and A are similar and so have the same eigenvalues. When X is only nonsingular, we say $X^T AX$ and A are *congruent*. In this case $X^T AX$ will generally not have the same eigenvalues as A, but the next theorem tells us that the two sets of eigenvalues will at least have the same signs.

THEOREM 5.3. **(Sylvester's Inertia Theorem.)** *Let A be symmetric and X be nonsingular. Then A and $X^T AX$ have the same inertia.*

Proof. Let n be the dimension of A. Now suppose that A has ν negative eigenvalues but that $X^T AX$ has $\nu' < \nu$ negative eigenvalues; we will find a contradiction to prove that this cannot happen. Let $\mathbf{N}$ be the corresponding ν dimensional negative eigenspace of A; i.e., $\mathbf{N}$ is spanned by the eigenvectors of the ν negative eigenvalues of A. This means that for any nonzero $x \in \mathbf{N}$, $x^T Ax < 0$. Let $\mathbf{P}$ be the $(n - \nu')$-dimensional nonnegative eigenspace of $X^T AX$; this means that for any nonzero $x \in \mathbf{P}$, $x^T X^T AXx \geq 0$. Since X is nonsingular, the space $X\mathbf{P}$ is also $n - \nu'$ dimensional. Since $\dim(\mathbf{N}) + \dim(X\mathbf{P}) = \nu + n - \nu' > n$, the spaces $\mathbf{N}$ and $X\mathbf{P}$ must contain a nonzero vector x in their intersection. But then $0 > x^T Ax$ since $x \in \mathbf{N}$ and $0 \leq x^T Ax$ since $x \in X\mathbf{P}$, which is a contradiction. Therefore, $\nu = \nu'$; i.e., A and $X^T AX$ have the same number of negative eigenvalues. An analogous argument shows they have the same number of positive eigenvalues. Thus, they must also have the same number of zero eigenvalues. □

Now we consider how eigenvectors can change under perturbations of $A+E$ of A. To state our bound we need to define the *gap* in the spectrum.

DEFINITION 5.3. *Let A have eigenvalues $\alpha_1 \geq \cdots \geq \alpha_n$. Then the* gap *between an eigenvalue α_i and the rest of the spectrum is defined to be* $\text{gap}(i, A) = \min_{j=i} |\alpha_j - \alpha_i|$*. We will also write* $\text{gap}(i)$ *if A is understood from the context.*

The basic result is that the sensitivity of an eigenvector depends on the gap of its corresponding eigenvalue: a small gap implies a sensitive eigenvector.

EXAMPLE 5.4. Let $A = \begin{bmatrix} 1+g & \\ & 1 \end{bmatrix}$ and $A+E = \begin{bmatrix} 1+g & \epsilon \\ \epsilon & 1 \end{bmatrix}$, where $0 < \epsilon < g$. Thus $\text{gap}(i, A) = g \approx \text{gap}(i, A + E)$ for $i = 1, 2$. The eigenvectors of A are just $q_1 = e_1$ and $q_2 = e_2$. A small computation reveals that the eigenvectors of $A + E$ are

$$\hat{q}_1 = \beta \cdot \begin{bmatrix} 1 + \sqrt{1 + \left(\frac{2\epsilon}{g}\right)^2} \\ \frac{2\epsilon}{g} \end{bmatrix} \approx \begin{bmatrix} 1 \\ \frac{\epsilon}{g} \end{bmatrix},$$

$$\hat{q}_2 = \beta \cdot \begin{bmatrix} \frac{-\frac{2\epsilon}{g}}{1+\sqrt{1+\left(\frac{2\epsilon}{g}\right)^2}} \end{bmatrix} \approx \begin{bmatrix} -\frac{\epsilon}{g} \\ 1 \end{bmatrix},$$

where $\beta \approx 1/2$ is a normalization factor. We see that the angle between the perturbed vectors $\hat{q}_i$ and unperturbed vectors q_i equals ϵ/g to first order in ϵ. So the angle is proportional to the reciprocal of the gap g. $\diamond$

The general case is essentially the same as the 2-by-2 case just analyzed.

THEOREM 5.4. *Let $A = Q\Lambda Q^T = Q\mathrm{diag}(\alpha_i)Q^T$ be an eigendecomposition of A. Let $A + E = \hat{A} = \hat{Q}\hat{\Lambda}\hat{Q}^T$ be the perturbed eigendecomposition. Write $Q = [q_1, \ldots, q_n]$ and $\hat{Q} = [\hat{q}_1, \ldots, \hat{q}_n]$, where q_i and $\hat{q}_i$ are the unperturbed and perturbed unit eigenvectors, respectively. Let θ denote the acute angle between q_i and $\hat{q}_i$. Then*

$$\frac{1}{2}\sin 2\theta \le \frac{\|E\|_2}{\mathrm{gap}(i, A)}, \qquad \textit{provided that } \mathrm{gap}(i, A) > 0.$$

Similarly

$$\frac{1}{2}\sin 2\theta \le \frac{\|E\|_2}{\mathrm{gap}(i, A + E)}, \qquad \textit{provided that } \mathrm{gap}(i, A + E) > 0.$$

Note that when $\theta \ll 1$, then $1/2 \sin 2\theta \approx \sin\theta \approx \theta$.

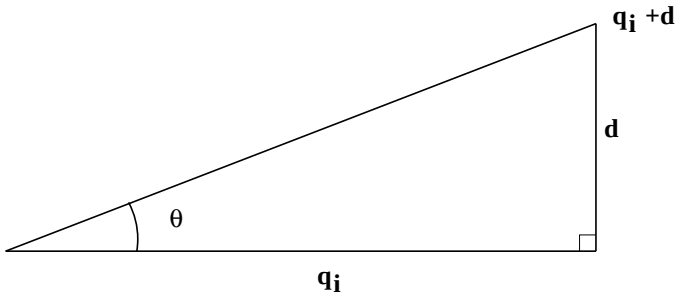
The attraction of stating the bound in terms of $\mathrm{gap}(i, A + E)$, as well as $\mathrm{gap}(i, A)$, is that frequently we know only the eigenvalues of $A + E$, since they are typically the output of the eigenvalue algorithm that we have used. In this case it is straightforward to evaluate $\mathrm{gap}(i, A + E)$, whereas we can only estimate $\mathrm{gap}(i, A)$.

When the first upper bound exceeds $1/2$, i.e., $\|E\|_2 \ge \mathrm{gap}(i, A)/2$, the bound reduces to $\sin 2\theta \le 1$, which provides no information about θ. Here is why we cannot bound θ in this situation: If E is this large, then $A + E$'s eigenvalue $\hat{\alpha}_i$ could be sufficiently far from α_i for $A + E$ to have a multiple eigenvalue at α_i. For example, consider $A = \mathrm{diag}(2, 0)$ and $A + E = I$. But such an $A + E$ does not have a unique eigenvector q_i; indeed, $A + E = I$ has any vector as an eigenvector. Thus, it makes no sense to try to bound θ. The same considerations apply when the second upper bound exceeds $1/2$.

Proof. It suffices to prove the first upper bound, because the second one follows by considering $A + E$ as the unperturbed matrix and $A = (A + E) - E$ as the perturbed matrix.

Let $q_i + d$ be an eigenvector of $A + E$. To make d unique, we impose the restriction that it be orthogonal to q_i (written $d \perp q_i$) as shown below. Note

that this means that $q_i + d$ is not a unit vector, so $\hat{q}_i = (q_i + d)/\|q_i + d\|_2$. Then $\tan\theta = \|d\|_2$ and $\sec\theta = \|q_i + d\|_2$.



Now write the ith column of $(A + E)\hat{Q} = \hat{Q}\hat{\Lambda}$ as

$$(A + E)(q_i + d) = \hat{\alpha}_i(q_i + d), \tag{5.4}$$

where we have also multiplied each side by $\|q_i + d\|_2$. Define $\eta = \hat{\alpha}_i - \alpha_i$. Subtract $Aq_i = \alpha_i q_i$ from both sides of (5.4) and rearrange to get

$$(A - \alpha_i I)d = (\eta I - E)(q_i + d). \tag{5.5}$$

Since $q_i^T(A - \alpha_i I) = 0$, both sides of (5.5) are orthogonal to q_i. This lets us write $z \equiv (\eta I - E)(q_i + d) = \sum_{j=i} \zeta_j q_j$ and $d \equiv \sum_{j=i} \delta_i q_j$. Since $(A - \alpha_i I)q_j = (\alpha_j - \alpha_i)q_j$, we can write

$$(A - \alpha_i I)d = \sum_{j=i}(\alpha_j - \alpha_i)\delta_j q_j = \sum_{j=i} \zeta_j q_j = (\eta I - E)(q_i + d)$$

or

$$d = \sum_{j=i} \delta_j q_j = \sum_{j=i} \frac{\zeta_j}{\alpha_j - \alpha_i} q_j.$$

Thus

$$\begin{aligned}
\tan\theta &= \|d\|_2 \\
&= \| \sum_{j=i} \frac{\zeta_j}{\alpha_j - \alpha_i} q_j \|_2 \\
&= \left(\sum_{j=i} \left(\frac{\zeta_j}{\alpha_j - \alpha_i} \right)^2 \right)^{1/2} && \text{since the } q_j \text{ are orthonormal} \\
&\le \frac{1}{\mathrm{gap}(i, A)} \left(\sum_{j=i} \zeta_j^2 \right)^{1/2} && \text{since } \mathrm{gap}(i, A) \text{ is the smallest denominator} \\
&= \frac{\|z\|_2}{\mathrm{gap}(i, A)} \quad .
\end{aligned}$$

If we were to use Weyl's theorem and the triangle inequality to bound $\|z\|_2 \leq (\|E\|_2 + |\eta|) \cdot \|q_i + d\|_2 \leq 2\|E\|_2 \sec\theta$, then we could conclude that $\sin\theta \leq 2\|E\|_2/\text{gap}(i, A)$.

But we can do a little better than this by bounding $\|z\|_2 = \|(\eta I - E)(q_i + d)\|_2$ more carefully: Multiply (5.4) by q_i^T on both sides, cancel terms, and rearrange to get $\eta = q_i^T E(q_i + d)$. Thus

$$\begin{aligned} z &= (q_i + d)\eta - E(q_i + d) = (q_i + d)q_i^T E(q_i + d) - E(q_i + d) \\ &= ((q_i + d)q_i^T - I)E(q_i + d), \end{aligned}$$

and so $\|z\|_2 \leq \|(q_i+d)q_i^T - I\| \cdot \|E\|_2 \cdot \|q_i+d\|$. We claim that $\|(q_i+d)q_i^T - I\|_2 = \|q_i + d\|_2$ (see Question 5.7). Thus $\|z\|_2 \leq \|q_i + d\|_2^2 \cdot \|E\|_2$, so

$$\tan\theta \leq \frac{\|z\|_2}{\text{gap}(i, A)} \leq \frac{\|q_i + d\|_2^2 \|E\|_2}{\text{gap}(i, A)} = \frac{\sec^2\theta \cdot \|E\|_2}{\text{gap}(i, A)}$$

or

$$\frac{\|E\|_2}{\text{gap}(i, A)} \geq \frac{\tan\theta}{\sec^2\theta} = \sin\theta\cos\theta = \frac{1}{2}\sin 2\theta$$

as desired. □

An analogous theorem can be proven for singular vectors (see Question 5.8).

The Rayleigh quotient has other nice properties. The next theorem tells us that the Rayleigh quotient is a "best approximation" to an eigenvalue in a natural sense. This is the basis of the Rayleigh quotient iteration in section 5.3.2 and the iterative algorithms in Chapter 7. It may also be used to evaluate the accuracy of an approximate eigenpair obtained in any way at all, not just by the algorithms discussed here.

THEOREM 5.5. *Let A be symmetric, x be a unit vector, and β be a scalar. Then A has an eigenpair $Aq_i = \alpha_i q_i$ satisfying $|\alpha_i - \beta| \leq \|Ax - \beta x\|_2$. Given x, the choice $\beta = \rho(x, A)$ minimizes $\|Ax - \beta x\|_2$.*

With a little more information about the spectrum of A, we can get tighter bounds. Let $r = Ax - \rho(x, A)x$. Let α_i be the eigenvalue of A closest to $\rho(x, A)$. Let $\text{gap}' \equiv \min_{j=i} |\alpha_j - \rho(x, A)|$; this is a variation on the gap defined earlier. Let θ be the acute angle between x and q_i. Then

$$\sin\theta \leq \frac{\|r\|_2}{\text{gap}'} \tag{5.6}$$

and

$$|\alpha_i - \rho(x, A)| \leq \frac{\|r\|_2^2}{\text{gap}'}. \tag{5.7}$$

See Theorem 7.1 for a generalization of this result to a set of eigenvalues.

Notice that in equation (5.7) the difference between the Rayleigh quotient $\rho(x, A)$ and an eigenvalue α_i is proportional to the *square* of the residual norm

$\|r\|_2$. This high accuracy is the basis of the cubic convergence of the Rayleigh quotient iteration algorithm of section 5.3.2.

Proof. We prove only the first result and leave the others for questions 5.9 and 5.10 at the end of the chapter.

If β is an eigenvalue of A, the result is immediate. So assume instead that $A - \beta I$ is nonsingular. Then $x = (A - \beta I)^{-1}(A - \beta I)x$ and

$$1 = \|x\|_2 \leq \|(A - \beta I)^{-1}\|_2 \cdot \|(A - \beta I)x\|_2.$$

Writing A's eigendecomposition as $A = Q\Lambda Q^T = Q\text{diag}(\alpha_1, \ldots, \alpha_n)Q^T$, we get

$$\|(A - \beta I)^{-1}\|_2 = \|Q(\Lambda - \beta I)^{-1}Q^T\|_2 = \|(\Lambda - \beta I)^{-1}\|_2 = 1/\min_i |\alpha_i - \beta|,$$

so $\min_i |\alpha_i - \beta| \leq \|(A - \beta I)x\|_2$ as desired.

To show that $\beta = \rho(x, A)$ minimizes $\|Ax - \beta x\|_2$ we will show that x is orthogonal to $Ax - \rho(x, A)x$ so that applying the Pythagorean theorem to the sum of orthogonal vectors

$$Ax - \beta x = [Ax - \rho(x, A)x] + [(\rho(x, A) - \beta)x]$$

yields

$$\begin{aligned} \|Ax - \beta x\|_2^2 &= \|Ax - \rho(x, A)x\|_2^2 + \|(\rho(x, A) - \beta)x\|_2^2 \\ &\geq \|Ax - \rho(x, A)x\|_2^2 \end{aligned}$$

with equality only when $\beta = \rho(x, A)$.

To confirm orthogonality of x and $Ax - \rho(x, A)x$ we need to verify that

$$x^T(Ax - \rho(x, A)x) = x^T(Ax - \frac{(x^T Ax)}{x^T x}x) = x^T Ax - x^T Ax\frac{x^T x}{x^T x} = 0$$

as desired. □

Example 5.5. We illustrate Theorem 5.5 using a matrix from Example 5.4. Let $A = \begin{bmatrix} 1+g & \epsilon \\ \epsilon & 1 \end{bmatrix}$, where $0 < \epsilon < g$. Let $x = [1, 0]^T$ and $\beta = \rho(x, A) = 1 + g$. Then $r = Ax - \beta x = [0, \epsilon]^T$ and $\|r\|_2 = \epsilon$. The eigenvalues of A are $\alpha_\pm = 1 + \frac{g}{2} \pm (1 + (\frac{2\epsilon}{g})^2)^{1/2}$, and the eigenvectors are given in Example 5.4 (where the matrix is called $A + E$ instead of A).

Theorem 5.5 predicts that $\|Ax - \beta x\|_2 = \|r\|_2 = \epsilon$ is a bound on the distance from $\beta = 1 + g$ to the nearest eigenvalue α_+ of A; this is also predicted by Weyl's theorem (Theorem '5.1). We will see below that this bound is much looser than bound (5.7).

When ϵ is much smaller than g, there will be one eigenvalue near $1 + g$ with its eigenvector near x and another eigenvalue near 1 with its eigenvector near $[0, 1]^T$. This means $\text{gap}' = |\alpha_- - \rho(x, A)| = \frac{g}{2}(1 + (1 + (\frac{2\epsilon}{g})^2)^{1/2})$, and

so bound (5.6) implies that the angle θ between x and the true eigenvector is bounded by

$$\sin\theta \le \frac{\|r\|_2}{\text{gap}'} = \frac{2\epsilon/g}{1+(1+(\frac{2\epsilon}{g})^2)^{1/2}}.$$

Comparing with the explicit eigenvectors in Example 5.4, we see that the upper bound is actually equal to $\tan\theta$, which is nearly the same as $\sin\theta$ for tiny θ. So bound (5.6) is quite accurate.

Now consider bound (5.7) on the difference $|\beta - \alpha_+|$. It turns out that for this 2-by-2 example both $|\beta - \alpha_+|$ and its bound are *exactly* equal to

$$\frac{\|r\|_2^2}{\text{gap}'} = \epsilon \cdot \frac{2\epsilon/g}{1+(1+(\frac{2\epsilon}{g})^2)^{1/2}}.$$

Let us evaluate these bounds in the special case where $g = 10^{-2}$ and $e = 10^{-5}$. Then the eigenvalues of A are approximately $\alpha_+ = 1.01000001 = 1.01 + 10^{-8}$ and $\alpha_- = .99999999 = 1 - 10^{-8}$. The first bound is $|\beta - \alpha_+| \le \|r\|_2 = 10^{-5}$, which is 10^3 times larger than the actual error 10^{-8}. In contrast, bound (5.7) is $|\beta - \alpha_+| \le \|r\|_2^2/\text{gap}' = (10^{-5})^2/(1.01 - \alpha_-) \approx 10^{-8}$, which is tight. The actual angle θ between x and the true eigenvector for α_+ is about 10^{-3}, as is the bound $\|r\|_2/\text{gap}' = 10^{-5}/(1.01 - \alpha_-) \approx 10^{-3}$. $\diamond$

Finally, we discuss what happens when one has a group of k tightly clustered eigenvalues, and wants to compute their eigenvectors. By "tightly clustered" we mean that the gap between any eigenvalue in the cluster and some other eigenvalue in the cluster is small but that eigenvalues not in the cluster are well separated. For example, one could have $k = 20$ eigenvalues in the interval [.9999,1.0001], but all other eigenvalues might be greater than 2. Then Theorems 5.4 and 5.5 indicate that we cannot hope to get the individual eigenvectors accurately. However, it is possible to compute the k-dimensional invariant subspace spanned by these vectors quite accurately. See [195] for details.

5.2.1. Relative Perturbation Theory

This section describes tighter bounds on eigenvalues and eigenvectors than in the last section. These bounds are needed to justify the high-accuracy algorithms for computing singular values and eigenvalues described in sections 5.4.2 and 5.4.3.

To contrast the bounds that we will present here to those in the previous section, let us consider the 1-by-1 case. Given a scalar α, a perturbed scalar $\hat{\alpha} = \alpha + e$ and a bound $|e| \le \epsilon$, we can obviously bound the *absolute error* in $\hat{\alpha}$ by $|\hat{\alpha} - \alpha| \le \epsilon$. This was the approach taken in the last section. Consider instead the perturbed scalar $\hat{\alpha} = x^2\alpha$ and a bound $|x^2 - 1| \le \epsilon$. This lets us

bound the *relative error* in $\hat{\alpha}$ by

$$\frac{|\hat{\alpha} - \alpha|}{|\alpha|} = |x^2 - 1| \leq \epsilon.$$

We generalize this simple idea to matrices as follows. In the last section we bounded the *absolute* difference in the eigenvalues α_i of A and $\hat{\alpha}_i$ of $\hat{A} = A + E$ by $|\hat{\alpha}_i - \alpha_i| \leq \|E\|_2$. Here we will bound the *relative* difference between the eigenvalues α_i of A and $\hat{\alpha}_i$ of $\hat{A} = X^T AX$ in terms of $\epsilon \equiv \|X^T X - I\|_2$.

THEOREM 5.6. "Relative" Weyl. *Let A have eigenvalues α_i and $\hat{A} = X^T AX$ have eigenvalues $\hat{\alpha}_i$. Let $\epsilon \equiv \|X^T X - I\|_2$. Then $|\hat{\alpha}_i - \alpha_i| \leq |\alpha_i|\epsilon$. If $\alpha_i = 0$, then we can also write*

$$\frac{|\hat{\alpha}_i - \alpha_i|}{|\alpha_i|} \leq \epsilon. \tag{5.8}$$

Proof. Since the ith eigenvalue of $A - \alpha_i I$ is zero, Sylvester's theorem of inertia tells us that the same is true of

$$X^T(A - \alpha_i I)X = (X^T AX - \alpha_i I) + \alpha_i(I - X^T X) \equiv H + F.$$

Weyl's theorem says that $|\lambda_i(H) - 0| \leq \|F\|_2$, or $|\hat{\alpha}_i - \alpha_i| \leq |\alpha_i| \cdot \|X^T X - I\|_2 = |\alpha_i|\epsilon$. □

Note that when X is orthogonal, $\epsilon = \|X^T X - I\|_2 = 0$, so the theorem confirms that $X^T AX$ and A have the same eigenvalues. If X is "nearly" orthogonal, i.e., ϵ is small, the theorem says the eigenvalue are nearly the same, in the sense of relative error.

COROLLARY 5.2. *Let G be an arbitrary matrix with singular values σ_i, and let $\hat{G} = Y^T GX$ have singular values $\hat{\sigma}_i$. Let $\epsilon \equiv \max(\|X^T X - I\|_2, \|Y^T Y - I\|_2)$. Then $|\hat{\sigma}_i - \sigma_i| \leq \epsilon\sigma_i$. If $\sigma_i = 0$, then we can write*

$$\frac{|\hat{\sigma}_i - \sigma_i|}{\sigma_i} \leq \epsilon. \tag{5.9}$$

We can similarly extend Theorem 5.4 to bound the difference between eigenvectors q_i of A and eigenvectors $\hat{q}_i$ of $\hat{A} = X^T AX$. To do so, we need to define the *relative gap* in the spectrum.

DEFINITION 5.4. *The* relative gap *between an eigenvalue α_i of A and the rest of the spectrum is defined to be* $\text{rel\_gap}(i, A) = \min_{j=i} \frac{|\alpha_j - \alpha_i|}{|\alpha_i|}$.

THEOREM 5.7. *Suppose that A has eigenvalues α_i and corresponding unit eigenvectors q_i. Suppose $\hat{A} = X^T AX$ has eigenvalues $\hat{\alpha}_i$ and corresponding unit eigenvectors $\hat{q}_i$. Let θ be the acute angle between q_i and $\hat{q}_i$. Let $\epsilon_1 = \|I - X^{-T}X^{-1}\|_2$ and $\epsilon_2 = \|X - I\|_2$. Then provided that $\epsilon_1 < 1$ and* $\text{rel\_gap}(i, X^T AX) > 0$,

$$\frac{1}{2}\sin 2\theta \leq \frac{\epsilon_1}{1 - \epsilon_1} \cdot \frac{1}{\text{rel\_gap}(i, X^T AX)} + \epsilon_2.$$

Proof. Let $\eta = \hat{\alpha}_i - \alpha_i$, $H = A - \hat{\alpha}_i I$, and $F = \hat{\alpha}_i(I - X^{-T}X^{-1})$. Note that

$$H + F = A - \hat{\alpha}_i X^{-T}X^{-1} = X^{-T}(X^T AX - \hat{\alpha}_i I)X^{-1}.$$

Thus $Hq_i = -\eta q_i$ and $(H + F)(X\hat{q}_i) = 0$ so that $X\hat{q}_i$ is an eigenvector of $H + F$ with eigenvalue 0. Let θ_1 be the acute angle between q_i and $X\hat{q}_i$. By Theorem 5.4, we can bound

$$\frac{1}{2}\sin 2\theta_1 \leq \frac{\|F\|_2}{\text{gap}(i, H + F)}. \tag{5.10}$$

We have $\|F\|_2 = |\hat{\alpha}_i|\epsilon_1$. Now $\text{gap}(i, H + F)$ is the magnitude of the smallest nonzero eigenvalue of $H + F$. Since $X^T(H + F)X = X^T AX - \hat{\alpha}_i I$ has eigenvalues $\hat{\alpha}_j - \hat{\alpha}_i$, Theorem 5.6 tells us that the eigenvalues of $H + F$ lie in intervals from $(1 - \epsilon_1)(\hat{\alpha}_j - \hat{\alpha}_i)$ to $(1 + \epsilon_1)(\hat{\alpha}_j - \hat{\alpha}_i)$. Thus $\text{gap}(i, H + F) \geq (1 - \epsilon_1)\text{gap}(i, X^T AX)$, and so substituting into (5.10) yields

$$\frac{1}{2}\sin 2\theta_1 \leq \frac{\epsilon_1|\hat{\alpha}_i|}{(1 - \epsilon_1)\text{gap}(i, X^T AX)} = \frac{\epsilon_1}{(1 - \epsilon_1)\text{rel\_gap}(i, X^T AX)}. \tag{5.11}$$

Now let θ_2 be the acute angle between $X\hat{q}_i$ and $\hat{q}_i$ so that $\theta \leq \theta_1 + \theta_2$. Using trigonometry we can bound $\sin\theta_2 \leq \|(X - I)\hat{q}_i\|_2 \leq \|X - I\|_2 = \epsilon_2$, and so by the triangle inequality (see Question 5.11)

$$\begin{aligned}
\frac{1}{2}\sin 2\theta &\leq \frac{1}{2}\sin 2\theta_1 + \frac{1}{2}\sin 2\theta_2 \\
&\leq \frac{1}{2}\sin 2\theta_1 + \sin\theta_2 \\
&\leq \frac{\epsilon_1}{(1 - \epsilon_1)\text{rel_gap}(i, X^T AX)} + \epsilon_2
\end{aligned}$$

as desired. □

An analogous theorem can be proven for singular vectors [99].

Example 5.6. We again consider the mass-spring system of Example 5.1 and use it to show that bounds on eigenvalues provided by Weyl's theorem (Theorem 5.1) can be much worse (looser) than the "relative" version of Weyl's theorem (Theorem 5.6). We will also see that the eigenvector bound of Theorem 5.7 can be much better (tighter) than the bound of Theorem 5.4.

Suppose that $M = \text{diag}(1, 100, 10000)$ and $K_D = \text{diag}(10000, 100, 1)$. Following Example 5.1, we define $K = BK_D B^T$ and $\hat{K} = M^{-1/2}KM^{-1/2}$, where

$$B = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & \ddots & -1 \\ & & & 1 \end{bmatrix}$$

and so

$$\hat{K} = M^{-1/2} K M^{-1/2} = \begin{bmatrix} 10100 & -10 & \\ -10 & 1.01 & -.001 \\ & -.001 & .0001 \end{bmatrix}.$$

To five decimal places, the eigenvalues of $\hat{K}$ are 10100, 1.0001 and .00099. Suppose we now perturb the masses (m_{ii}) and spring constants ($k_{D,ii}$) by at most 1% each. How much can the eigenvalues change? The largest matrix entry is $\hat{K}_{11}$, and changing m_{11} to .99 and $k_{D,11}$ to 10100 will change $\hat{K}_{11}$ to about 10305, a change of 205 in norm. Thus, Weyl's theorem tells us each eigenvalue could change by as much as ± 205, which would change the smaller two eigenvalues utterly. The eigenvector bound from Theorem 5.4 also indicates that the corresponding eigenvectors could change completely.

Now let us apply Theorem 5.6 to $\hat{K}$, or actually Corollary 5.2 to $G = M^{-1/2} B K_D^{1/2}$, where $\hat{K} = GG^T$ as defined in Example 5.1. Changing each mass by at most 1% is equivalent to perturbing G to XG, where X is diagonal with diagonal entries between $1/\sqrt{.99} \approx 1.005$ and $1/\sqrt{1.01} \approx .995$. Then Corollary 5.2 tells us that the singular values of G can change only by factors within the interval [.995, 1.005], so the eigenvalues of M can change only by 1% too. In other words, the smallest eigenvalue can change only in its second decimal place, just like the largest eigenvalue. Similarly, changing the spring constants by at most 1% is equivalent to changing G to GX, and again the eigenvalues cannot change by more than 1%. If we perturb both M and K_D at the same time, the eigenvalues will move by about 2%. Since the eigenvalues differ so much in magnitude, their relative gaps are all quite large, and so their eigenvectors can rotate only by about 3% in angle too.

For a different approach to relative error analysis, more suitable for matrices arising from differential ("unbounded") operators, see [159].

5.3. Algorithms for the Symmetric Eigenproblem

We discuss a variety of algorithms for the symmetric eigenproblem. As mentioned in the introduction, we will discuss only *direct methods*, leaving *iterative methods* for Chapter 7.

In Chapter 4 on the nonsymmetric eigenproblem, the only algorithm that we discussed was QR iteration, which could find all the eigenvalues and optionally all the eigenvectors. We have many more algorithms available for the symmetric eigenproblem, which offer us more flexibility and efficiency. For example, the *Bisection algorithm* described below can be used to find only the eigenvalues in a user-specified interval $[a, b]$ and can do so much faster than it could find all the eigenvalues.

All the algorithms below, except Rayleigh quotient iteration and Jacobi's method, assume that the matrix has first been reduced to tridiagonal form,

using the variation of Algorithm 4.6 in section 4.4.7. This is an initial cost of $\frac{4}{3}n^3$ flops, or $\frac{8}{3}n^3$ flops if eigenvectors are also desired.

1. *Tridiagonal QR iteration.* This algorithm finds all the eigenvalues, and optionally all the eigenvectors, of a symmetric tridiagonal matrix. Implemented efficiently, it is currently the fastest practical method to find all the eigenvalues of a symmetric tridiagonal matrix, taking $O(n^2)$ flops. Since reducing a dense matrix to tridiagonal form costs $\frac{4}{3}n^3$ flops, $O(n^2)$ is negligible for large enough n. But for finding all the eigenvectors as well, QR iteration takes a little over $6n^3$ flops on average and is only the fastest algorithm for small matrices, up to about $n = 25$. This is the algorithm underlying the Matlab command `eig`[17] and the LAPACK routines `ssyev` (for dense matrices) and `sstev` (for tridiagonal matrices).

2. *Rayleigh quotient iteration.* This algorithm underlies QR iteration, but we present it separately in order to more easily analyze its extremely rapid convergence and because it may be used as an algorithm by itself. In fact, it generally converges cubically (as does QR iteration), which means that the number of correct digits asymptotically *triples* at each step.

3. *Divide-and-conquer.* This is currently the fastest method to find all the eigenvalues and eigenvectors of symmetric tridiagonal matrices larger than $n = 25$. (The implementation in LAPACK, `sstevd`, defaults to QR iteration for smaller matrices.)

   In the worst case, divide-and-conquer requires $O(n^3)$ flops, but in practice the constant is quite small. Over a large set of random test cases, it appears to take only $O(n^{2.3})$ flops on average, and as low as $O(n^2)$ for some eigenvalue distributions.

   In theory, divide-and-conquer could be implemented to run in $O(n \cdot \log^p n)$ flops, where p is a small integer [129]. This super-fast implementation uses the fast multipole method (FMM) [122], originally invented for the completely different problem of computing the mutual forces on n electrically charged particles. But the complexity of this super-fast implementation means that QR iteration is currently the algorithm of choice for finding all eigenvalues, and divide-and-conquer without the FMM is the method of choice for finding all eigenvalues and all eigenvectors.

4. *Bisection and inverse iteration.* Bisection may be used to find just a subset of the eigenvalues of a symmetric tridiagonal matrix, say, those in an interval $[a, b]$ or $[\alpha_i, \alpha_{i-j}]$. It needs only $O(nk)$ flops, where k is the

[17] Matlab checks to see whether the argument of `eig` is symmetric or not and uses the symmetric algorithm when appropriate.

number of eigenvalues desired. Thus Bisection can be much faster than QR iteration when $k \ll n$, since QR iteration requires $O(n^2)$ flops. Inverse iteration (Algorithm 4.2) can then be used to find the corresponding eigenvectors. In the best case, when the eigenvalues are "well separated" (we explain this more fully later), inverse iteration also costs only $O(nk)$ flops. This is much less than either QR or divide-and-conquer (without the FMM), even when all eigenvalues and eigenvectors are desired ($k = n$). But in the worst case, when many eigenvalues are clustered close together, inverse iteration takes $O(nk^2)$ flops and does not even guarantee the accuracy of the computed eigenvectors (although in practice it is almost always accurate). So divide-and-conquer and QR are currently the algorithms of choice for finding all (or most) eigenvalues and eigenvectors, especially when eigenvalues may be clustered. Bisection and inverse iteration are available as options in the LAPACK routine `ssyevx`.

There is current research on inverse iteration addressing the problem of close eigenvalues, which may make it the fastest method to find all the eigenvectors eigenvectors (besides, theoretically, divide-and-conquer with the FMM) [103, 201, 199, 174, 171, 173, 267]. However, software implementing this improved version of inverse iteration is not yet available.

5. *Jacobi's method.* This method is historically the oldest method for the eigenproblem, dating to 1846. It is usually much slower than any of the above methods, taking $O(n^3)$ flops with a large constant. But the method remains interesting, because it is sometimes much more accurate than the above methods. This is because Jacobi's method is sometimes capable of attaining the relative accuracy described in section 5.2.1 and so can sometimes compute tiny eigenvalues much more accurately than the previous methods [81]. We discuss the high-accuracy property of Jacobi's method in section 5.4.3, where we show how to compute the SVD.

Subsequent sections describe these algorithms in more detail. Section 5.3.6 presents comparative performance results.

5.3.1. Tridiagonal QR Iteration

Recall that the QR algorithm for the nonsymmetric eigenproblem had two phases:

1. Given A, use Algorithm 4.6 to find an orthogonal Q so that $QAQ^T = H$ is upper Hessenberg.

2. Apply QR iteration to H (as described in section 4.4.8) to get a sequence $H = H_0, H_1, H_2, \ldots$ of upper Hessenberg matrices converging to real Schur form.

Our first algorithm for the symmetric eigenproblem is completely analogous to this:

1. Given $A = A^T$, use the variation of Algorithm 4.6 in section 4.4.7 to find an orthogonal Q so that $QAQ^T = T$ is tridiagonal.
2. Apply QR iteration to T to get a sequence $T = T_0, T_1, T_2, \ldots$ of tridiagonal matrices converging to diagonal form.

We can see that QR iteration keeps all the T_i tridiagonal by noting that since QAQ^T is symmetric and upper Hessenberg, it must also be lower Hessenberg, i.e., tridiagonal. This keeps each QR iteration very inexpensive. An operation count reveals the following:

1. Reducing A to symmetric tridiagonal form T costs $\frac{4}{3}n^3 + O(n^2)$ flops, or $\frac{8}{3}n^3 + O(n^2)$ flops if eigenvectors are also desired.
2. One tridiagonal QR iteration with a single shift ("bulge chasing") costs $6n$ flops.
3. Finding all eigenvalues of T takes only 2 QR steps per eigenvalue on average, for a total of $6n^2$ flops.
4. Finding all eigenvalues and eigenvectors of T costs $6n^3 + O(n^2)$ flops.
5. The total cost to find just the eigenvalues of A is $\frac{4}{3}n^3 + O(n^2)$ flops.
6. The total cost to find all the eigenvalues and eigenvectors of A is $8\frac{2}{3}n^3 + O(n^2)$ flops.

We must still describe how the shifts are chosen to implement each QR iteration. Denote the ith iterate by

$$T_i = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix}.$$

The simplest choice of shift would be $\sigma_i = a_n$; this is the single shift QR iteration discussed in section 4.4.8. It turns out to be cubically convergent for almost all matrices, as shown in the next section. Unfortunately, examples exist where it does not converge [195, p. 76], so to get global convergence a slightly more complicated shift strategy is needed: We let the shift σ_i be the eigenvalue of $\begin{bmatrix} a_{n-1} & b_{n-1} \\ b_{n-1} & a_n \end{bmatrix}$ that is closest to a_n. This is called *Wilkinson's shift*.

THEOREM 5.8. Wilkinson. *QR iteration with Wilkinson's shift is globally, and at least linearly, convergent. It is asymptotically cubically convergent for almost all matrices.*

A proof of this theorem can be found in [195]. In LAPACK this routine is available as `ssyev`. The inner loop of the algorithm can be organized more efficiently when eigenvalues only are desired (`ssterf`; see also [102, 198]) than when eigenvectors are also computed (`ssteqr`).

EXAMPLE 5.7. Here is an illustration of the convergence of tridiagonal QR iteration, starting with the following tridiagonal matrix (diagonals only are shown, in columns):

$$T_0 = \text{tridiag} \begin{bmatrix} & .24929 & \\ 1.263 & & 1.263 \\ & .96880 & \\ -.82812 & & -.82812 \\ & .48539 & \\ -3.1883 & & -3.1883 \\ & -.91563 & \end{bmatrix}.$$

The following table shows the last offdiagonal entry of each T_i, the last diagonal entry of each T_i, and the difference between the last diagonal entry and its ultimate value (the eigenvalue $\alpha \approx -3.54627$). The cubic convergence of the error to zero in the last column is evident.

i	$T_i(4,3)$	$T_i(4,4)$	$T_i(4,4) - \alpha$
0	-3.1883	$-.91563$	2.6306
1	$-5.7 \cdot 10^{-2}$	-3.5457	$5.4 \cdot 10^{-4}$
2	$-2.5 \cdot 10^{-7}$	-3.5463	$1.2 \cdot 10^{-14}$
3	$-6.1 \cdot 10^{-23}$	-3.5463	0

At this point

$$T_3 = \text{tridiag} \begin{bmatrix} & 1.9871 & \\ .77513 & & .77513 \\ & 1.7049 & \\ -1.7207 & & -1.7207 \\ & .64214 & \\ -6.1 \cdot 10^{-23} & & -6.1 \cdot 10^{-23} \\ & -3.5463 & \end{bmatrix},$$

and we set the very tiny (4,3) and (3,4) entries to 0. This is called *deflation* and is stable, perturbing T_3 by only $6.1 \cdot 10^{-23}$ in norm. We now apply QR iteration again to the leading 3-by-3 submatrix of T_3, repeating the process to get the other eigenvalues.

5.3.2. Rayleigh Quotient Iteration

Recall from our analysis of QR iteration in section 4.4 that we are implicitly doing inverse iteration at every step. We explore this more carefully when the shift we choose to use in the inverse iteration is the Rayleigh quotient.

ALGORITHM 5.1. *Rayleigh quotient iteration: Given* x_0 *with* $\|x_0\|_2 = 1$*, and a user-supplied stopping tolerance* tol*, we iterate*

$\rho_0 = \rho(x_0, A) = \frac{x_0^T A x_0}{x_0^T x_0}$
$i = 0$
repeat
  $y_i = (A - \rho_{i-1} I)^{-1} x_{i-1}$
  $x_i = y_i / \|y_i\|_2$
  $\rho_i = \rho(x_i, A)$
  $i = i + 1$
until convergence ($\|Ax_i - \rho_i x_i\|_2 < \text{tol}$)

When the stopping criterion is satisfied, Theorem 5.5 tells us that ρ_i is within tol of an eigenvalue of A.

If one uses the shift $\sigma_i = a_{nn}$ in QR iteration and starts Rayleigh quotient iteration with $x_0 = [0, \ldots, 0, 1]^T$, then the connection between QR and inverse iteration discussed in section 4.4 can be used to show that the sequence of σ_i and ρ_i from the two algorithms are identical (see Question 5.13). In this case we will prove that convergence is almost always cubic.

THEOREM 5.9. *Rayleigh quotient iteration is locally cubically convergent; i.e., the number of correct digits triples at each step once the error is small enough and the eigenvalue is simple.*

Proof. We claim that it is enough to analyze the case when A is diagonal. To see why, write $Q^T A Q = \Lambda$, where Q is the orthogonal matrix whose columns are eigenvectors, and $\Lambda = \text{diag}(\alpha_1, \ldots, \alpha_n)$ is the diagonal matrix of eigenvalues. Now change variables in Rayleigh quotient iteration to $\hat{x}_i \equiv Q^T x_i$ and $\hat{y}_i \equiv Q^T y_i$. Then

$$\rho_i = \rho(x_i, A) = \frac{x_i^T A x_i}{x_i^T x_i} = \frac{\hat{x}_i^T Q^T A Q \hat{x}_i}{\hat{x}_i^T Q^T Q \hat{x}_i} = \frac{\hat{x}_i^T \Lambda \hat{x}_i}{\hat{x}_i^T \hat{x}_i} = \rho(\hat{x}_i, \Lambda)$$

and $Q\hat{y}_i = (A - \rho_i I)^{-1} Q \hat{x}_i$, so

$$\hat{y}_i = Q^T (A - \rho_i I)^{-1} Q \hat{x}_i = (Q^T A Q - \rho_i I)^{-1} \hat{x}_i = (\Lambda - \rho_i I)^{-1} \hat{x}_i.$$

Therefore, running Rayleigh quotient iteration with A and x_0 is equivalent to running Rayleigh quotient iteration with Λ and $\hat{x}_0$. Thus we will assume without loss of generality that $A = \Lambda$ is already diagonal, so the eigenvectors of A are e_i, the columns of the identity matrix.

Suppose without loss of generality that x_i is converging to e_1, so we can write $x_i = e_1 + d_i$, where $\|d_i\|_2 \equiv \epsilon \ll 1$. To prove cubic convergence, we need to show that $x_{i+1} = e_1 + d_{i+1}$ with $\|d_{i+1}\|_2 = O(\epsilon^3)$.

We first note that

$$1 = x_i^T x_i = (e_1 + d_i)^T (e_1 + d_i) = e_1^T e_1 + 2e_1^T d_i + d_i^T d_i = 1 + 2d_{i1} + \epsilon^2$$

so that $d_{i1} = -\epsilon^2/2$. Therefore

$$\rho_i = x_i^T \Lambda x_i = (e_1 + d_i)^T \Lambda (e_1 + d_i) = e_1^T \Lambda e_1 + 2e_1^T \Lambda d_i + d_i^T \Lambda d_i = \alpha_1 - \eta,$$

where $\eta \equiv -2e_1^T \Lambda d_i - d_i^T \Lambda d_i = \alpha_1 \epsilon^2 - d_i^T \Lambda d_i$. We see that

$$|\eta| \le |\alpha_1|\epsilon^2 + \|\Lambda\|_2 \|d_i\|_2^2 \le 2\|\Lambda\|_2 \epsilon^2, \tag{5.12}$$

so $\rho_i = \alpha_1 - \eta = \alpha_1 + O(\epsilon^2)$ is a very good approximation to the eigenvalue α_1.

Now we can write

$$\begin{aligned}
y_{i+1} &= (\Lambda - \rho_i I)^{-1} x_i \\
&= \left[\frac{x_{i1}}{\alpha_1 - \rho_i}, \frac{x_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{x_{in}}{\alpha_n - \rho_i} \right]^T \\
&\qquad \text{because } (\Lambda - \rho_i I)^{-1} = \operatorname{diag}\left(\frac{1}{\alpha_j - \rho_i} \right) \\
&= \left[\frac{1 + d_{i1}}{\alpha_1 - \rho_i}, \frac{d_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{d_{in}}{\alpha_n - \rho_i} \right]^T \\
&\qquad \text{because } x_i = e_1 + d_i \\
&= \left[\frac{1 - \epsilon^2/2}{\eta}, \frac{d_{i2}}{\alpha_2 - \alpha_1 + \eta}, \ldots, \frac{d_{in}}{\alpha_n - \alpha_1 + \eta} \right]^T \\
&\qquad \text{because } \rho_i = \alpha_1 - \eta \text{ and } d_{i1} = -\epsilon^2/2 \\
&= \frac{1 - \epsilon^2/2}{\eta} \cdot \left[1, \frac{d_{i2}\eta}{(1 - \epsilon^2/2)(\alpha_2 - \alpha_1 + \eta)}, \ldots, \right. \\
&\qquad \left. \frac{d_{in}\eta}{(1 - \epsilon^2/2)(\alpha_n - \alpha_1 + \eta)} \right]^T \\
&\equiv \frac{1 - \epsilon^2/2}{\eta} \cdot (e_1 + \hat{d}_{i+1}).
\end{aligned}$$

To bound $\|\hat{d}_{i+1}\|_2$, we note that we can bound each denominator using $|\alpha_j - \alpha_1 + \eta| \ge \operatorname{gap}(1, \Lambda) - |\eta|$, so using (5.12) as well we get

$$\|\hat{d}_{i+1}\|_2 \le \frac{\|d_i\|_2 |\eta|}{(1 - \epsilon^2/2)(\operatorname{gap}(1, \Lambda) - |\eta|)} \le \frac{2\|\Lambda\|_2 \epsilon^3}{(1 - \epsilon^2/2)(\operatorname{gap}(1, \Lambda) - 2\|\Lambda\|\epsilon^2)}$$

or $\|\hat{d}_{i+1}\|_2 = O(\epsilon^3)$. Finally, since $x_{i+1} = e_1 + d_{i+1} = (e_1 + \hat{d}_{i+1})/\|e_1 + \hat{d}_{i+1}\|_2$, we see $\|d_{i+1}\|_2 = O(\epsilon^3)$ as well. $\square$

5.3.3. Divide-and-Conquer

This method is the fastest now available if you want all eigenvalues and eigenvectors of a tridiagonal matrix whose dimension is larger than about 25. (The exact threshold depends on the computer.) It is quite subtle to implement in a numerically stable way. Indeed, although this method was first introduced in 1981 [58], the "right" implementation was not discovered until 1992 [125, 129]). This routine is available as LAPACK routines `ssyevd` for dense matrices and `sstevd` for tridiagonal matrices. This routine uses divide-and-conquer for matrices of dimension larger than 25 and automatically switches to QR iteration for smaller matrices (or if eigenvalues only are desired).

We first discuss the overall structure of the algorithm, and leave numerical details for later. Let

$$
\begin{aligned}
T &= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m & b_m & & & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n & &
\end{array}\right] \\
&= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m - b_m & & & & & & \\
\hline
 & & & & a_{m+1} - b_m & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n & &
\end{array}\right] \\
&\quad + \left[\begin{array}{ccc|cc}
 & & & & \\
 & & & & \\
 & & & b_m & b_m \\
\hline
 & & & b_m & b_m \\
 & & & & \\
 & & & &
\end{array}\right]
\end{aligned}
$$

$$= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [0, \ldots, 0, 1, 1, 0, \ldots, 0] \equiv \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m v v^T.$$

Suppose that we have the eigendecompositions of T_1 and T_2: $T_i = Q_i \Lambda_i Q_i^T$. These will be computed recursively by this same algorithm. We relate the eigenvalues of T to those of T_1 and T_2 as follows.

$$\begin{aligned} T &= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 \Lambda_1 Q_1^T & 0 \\ 0 & Q_2 \Lambda_2 Q_2^T \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} + b_m u u^T \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}, \end{aligned}$$

where

$$u = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} \quad v = \begin{bmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{bmatrix}$$

since $v = [0, \ldots, 0, 1, 1, 0, \ldots, 0]^T$. Therefore, the eigenvalues of T are the same as those of the similar matrix $D + \rho u u^T$ where $D = \left[\begin{smallmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{smallmatrix}\right]$ is diagonal, $\rho = b_m$ is a scalar, and u is a vector. Henceforth we will assume without loss of generality that the diagonal $d_1, \ldots, d_n$ of D is sorted: $d_n \leq \cdots \leq d_1$.

To find the eigenvalues of $D + \rho u u^T$, assume first that $D - \lambda I$ is nonsingular, and compute the characteristic polynomial as follows:

$$\det(D + \rho u u^T - \lambda I) = \det((D - \lambda I)(I + \rho (D - \lambda)^{-1} u u^T)). \tag{5.13}$$

Since $D - \lambda I$ is nonsingular, $\det(I + \rho (D - \lambda)^{-1} u u^T) = 0$ whenever λ is an eigenvalue. Note that $I + \rho (D - \lambda)^{-1} u u^T$ is the identity plus a rank-1 matrix; the determinant of such a matrix is easy to compute:

LEMMA 5.1. *If x and y are vectors,* $\det(I + x y^T) = 1 + y^T x$.

The proof is left to Question 5.14.

Therefore

$$\det(I + \rho (D - \lambda)^{-1} u u^T) = 1 + \rho u^T (D - \lambda)^{-1} u = 1 + \rho \sum_{i=1}^{n} \frac{u_i^2}{d_i - \lambda} \equiv f(\lambda), \tag{5.14}$$

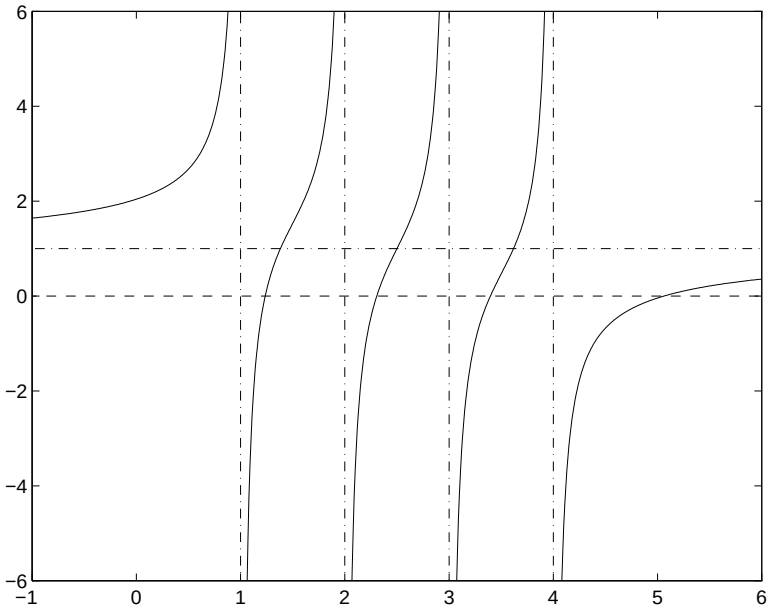


Fig. 5.2. *Graph of* $f(\lambda) = 1 + \frac{.5}{1-\lambda} + \frac{.5}{2-\lambda} + \frac{.5}{3-\lambda} + \frac{.5}{4-\lambda}$.

and the eigenvalues of T are the roots of the so-called *secular equation* $f(\lambda) = 0$. If all d_i are distinct and all $u_i = 0$ (the generic case), the function $f(\lambda)$ has the graph shown in Figure 5.2 (for $n = 4$ and $\rho > 0$).

As we can see, the line $y = 1$ is a horizontal asymptote, and the lines $\lambda = d_i$ are vertical asymptotes. Since $f'(\lambda) = \rho \sum_{i=1}^{n} \frac{u_i^2}{(d_i-\lambda)^2} > 0$, the function is strictly increasing except at $\lambda = d_i$. Thus the roots of $f(\lambda)$ are interlaced by the d_i, and there is one more root to the right of d_1 ($d_1 = 4$ in Figure 5.2). (If $\rho < 0$, then $f(\lambda)$ is decreasing and there is one more root to the left of d_n.) Since $f(\lambda)$ is monotonic and smooth on the intervals (d_i, d_{i+1}), it is possible to find a version of Newton's method that converges fast and monotonically to each root, given a starting point in (d_i, d_{i+1}). We discuss details later in this section. All we need to know here is that in practice Newton converges in a bounded number of steps per eigenvalue. Since evaluating $f(\lambda)$ and $f'(\lambda)$ costs $O(n)$ flops, finding one eigenvalue costs $O(n)$ flops, and so finding all n eigenvalues of $D + \rho uu^T$ costs $O(n^2)$ flops.

It is also easy to derive an expression for the eigenvectors of $D + uu^T$.

LEMMA 5.2. *If α is an eigenvalue of $D + \rho uu^T$, then $(D - \alpha I)^{-1}u$ is its eigenvector. Since $D - \alpha I$ is diagonal, this costs $O(n)$ flops to compute.*

Proof.

$$\begin{aligned}(D + \rho uu^T)[(D - \alpha I)^{-1}u] &= (D - \alpha I + \alpha I + \rho uu^T)(D - \alpha I)^{-1}u \\ &= u + \alpha(D - \alpha I)^{-1}u + u[\rho u^T(D - \alpha I)^{-1}u]\end{aligned}$$

$$\begin{aligned} &= u + \alpha(D-\alpha I)^{-1}u - u \\ &\quad \text{since } \rho u^T(D-\alpha I)^{-1}u + 1 = f(\alpha) = 1 \\ &= \alpha[(D-\alpha I)^{-1}u] \quad \text{as desired.} \quad \square \end{aligned}$$

Evaluating this formula for all n eigenvectors costs $O(n^2)$ flops. Unfortunately, this simple formula for the eigenvectors is not numerically stable, because two very close values of α_i can result in nonorthogonal computed eigenvectors u_i. Finding a stable alternative took over a decade from the original formulation of this algorithm. We discuss details later in this section.

The overall algorithm is recursive.

ALGORITHM 5.2. *Finding eigenvalues and eigenvectors of a symmetric tridiagonal matrix using divide-and-conquer:*

proc dc_eig (T, Q, Λ) ..... *from input* T *compute outputs* Q *and* Λ *where* $T = Q\Lambda Q^T$

if T *is* 1-*by*-1
  return $Q = 1, \Lambda = T$
else
  form $T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T$
  call dc_eig (T_1, Q_1, Λ_1)
  call dc_eig (T_2, Q_2, Λ_2)
  form $D + \rho u u^T$ *from* $\Lambda_1, \Lambda_2, Q_1, Q_2$
  find eigenvalues Λ *and eigenvectors* Q' *of* $D + \rho u u^T$
  form $Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q' =$ *eigenvectors of* T
  return Q *and* Λ
endif

We analyze the complexity of Algorithm 5.2 as follows. Let $t(n)$ be the number of flops to run dc_eig on an n-by-n matrix. Then

$$\begin{aligned} t(n) &= 2t(n/2) && \text{for the 2 recursive calls to dc\_eig}(T_i, Q_i, \Lambda_i) \\ &\quad +O(n^2) && \text{to find the eigenvalues of } D + \rho u u^T \\ &\quad +O(n^2) && \text{to find the eigenvectors of } D + \rho u u^T \\ &\quad +c \cdot n^3 && \text{to multiply } Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q'. \end{aligned}$$

If we treat Q_1, Q_2, and Q' as dense matrices and use the standard matrix multiplication algorithm, the constant in the last line is $c = 1$. Thus we see that the major cost in the algorithm is the matrix multiplication in the last line. Ignoring the $O(n^2)$ terms, we get $t(n) = 2t(n/2) + cn^3$. This geometric sum can be evaluated, yielding $t(n) \approx c\frac{4}{3}n^3$ (see Question 5.15). In practice, c

is usually much less than 1, because a phenomenon called *deflation* makes Q' quite sparse.

After discussing deflation in the next section, we discuss details of solving the secular equation, and computing the eigenvectors stably. Finally, we discuss how to accelerate the method by exploiting FMM techniques used in electrostatic particle simulation [122]. These sections may be skipped on a first reading.

Deflation

So far in our presentation we have assumed that the d_i are distinct, and the u_i nonzero. When this is not the case, the secular equation $f(\lambda) = 0$ will have $k < n$ vertical asymptotes, and so $k < n$ roots. But it turns out that the remaining $n - k$ eigenvalues are available very cheaply: If $d_i = d_{i+1}$, or if $u_i = 0$, one can easily show that d_i is also an eigenvalue of $D + \rho uu^T$ (see Question 5.16). This process is called *deflation*. In practice we use a threshold and deflate d_i either if it is close enough to d_{i+1} or if u_i is small enough.

In practice, deflation happens quite frequently: In experiments with random dense matrices with uniformly distributed eigenvalues, over 15% of the eigenvalues of the largest $D + \rho uu^T$ deflated, and in experiments with random dense matrices with eigenvalues approaching 0 geometrically, over 85% deflated! It is essential to take advantage of this behavior to make the algorithm fast [58, 208].

The payoff in deflation is not in making the solution of the secular equation faster; this costs only $O(n^2)$ anyway. The payoff is in making the matrix multiplication in the last step of the algorithm fast. For if $u_i = 0$, then the corresponding eigenvector is e_i, the ith column of the identity matrix (see Question 5.16). This means that the ith column of Q' is e_i, so no work is needed to compute the ith column of Q in the two multiplications by Q_1 and Q_2. There is a similar simplification when $d_i = d_{i+1}$. When many eigenvalues deflate, much of the work in the matrix multiplication can be eliminated. This is borne out in the numerical experiments presented in section 5.3.6.

Solving the Secular Equation

When some u_i is small but too large to deflate, a problem arises when trying to use Newton's method to solve the secular equation. Recall that the principle of Newton's method for updating an approximate solution λ_j of $f(\lambda) = 0$ is

1. to approximate the function $f(\lambda)$ near $\lambda = \lambda_j$ with a linear function $l(\lambda)$, whose graph is a straight line tangent to the graph of $f(\lambda)$ at $\lambda = \lambda_j$,
2. to let λ_{j+1} be the zero of this linear approximation: $l(\lambda_{j+1}) = 0$.

The graph in Figure 5.2 offers no apparent difficulties to Newton's method, because the function $f(\lambda)$ appears to be reasonably well approximated by

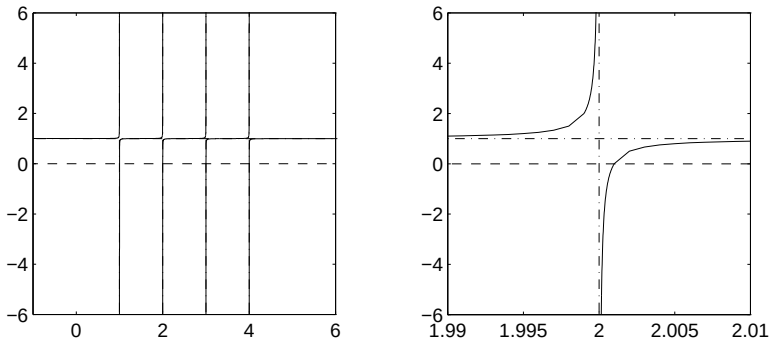


Fig. 5.3. *Graph of* $f(\lambda) = 1 + \frac{10^{-3}}{1-\lambda} + \frac{10^{-3}}{2-\lambda} + \frac{10^{-3}}{3-\lambda} + \frac{10^{-3}}{4-\lambda}$.

straight lines near each zero. But now consider the graph in Figure 5.3, which differs from Figure 5.2 only by changing u_i^2 from .5 to .001, which is not nearly small enough to deflate. The graph of $f(\lambda)$ in the left-hand figure is visually indistinguishable from its vertical and horizontal asymptotes, so in the right-hand figure we blow it up around one of the vertical asymptotes, $\lambda = 2$. We see that the graph of $f(\lambda)$ "turns the corner" very rapidly and is nearly horizontal for most values of λ. Thus, if we started Newton's method from almost any λ_0, the linear approximation $l(\lambda)$ would also be nearly horizontal with a slightly positive slope, so λ_1 would be an enormous negative number, a useless approximation to the true zero.

Newton's method can be modified to deal with this situation as follows. Since $f(\lambda)$ is not well approximated by a straight line $l(x)$, we approximate it by another simple function $h(x)$. There is nothing special about straight lines; any approximation $h(\lambda)$ that is both easy to compute and has zeros that are easy to compute can be used in place of $l(x)$ in Newton's method. Since $f(\lambda)$ has poles at d_i and d_{i+1} and these poles dominate the behavior of $f(\lambda)$ near them, it is natural when seeking the root in (d_i, d_{i+1}) to choose $h(\lambda)$ to have these poles as well, i.e.,

$$h(\lambda) = \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3.$$

There are several ways to choose the constants c_1, c_2, and c_3 so that $h(\lambda)$ approximates $f(\lambda)$; we present a slightly simplified version of the one used in the LAPACK routine `slaed4` [170, 44]. Assuming for a moment that we have chosen c_1, c_2, and c_3, we can easily solve $h(\lambda) = 0$ for λ by solving the equivalent quadratic equation

$$c_1(d_{i+1} - \lambda) + c_2(d_i - \lambda) + c_3(d_i - \lambda)(d_{i+1} - \lambda) = 0.$$

Given the approximate zero λ_j, here is how we compute c_1, c_2, and c_3 so that for λ near λ_j

$$\frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3 = h(\lambda) \approx f(\lambda) = 1 + \rho \sum_{k=1}^{n} \frac{u_k^2}{d_k - \lambda}.$$

Write

$$f(\lambda) = 1 + \sum_{k=1}^{i} \frac{u_k^2}{d_k - \lambda} + \sum_{k=i+1}^{n} \frac{u_k^2}{d_k - \lambda} \equiv 1 + \psi_1(\lambda) + \psi_2(\lambda).$$

For $\lambda \in (d_i, d_{i+1})$, $\psi_1(\lambda)$ is a sum of negative terms and $\psi_2(\lambda)$ is a sum of positive terms. Thus both $\psi_1(\lambda)$ and $\psi_2(\lambda)$ can be computed accurately, whereas adding them together would likely result in cancellation and loss of relative accuracy in the sum. We now choose c_1 and $\hat{c}_1$ so that

$$\begin{aligned} h_1(\lambda) &\equiv \hat{c}_1 + \frac{c_1}{d_i - \lambda} \quad \text{satisfies} \\ &\quad h_1(\lambda_j) = \psi_1(\lambda_j) \quad \text{and} \quad h_1'(\lambda_j) = \psi_1'(\lambda_j). \end{aligned} \tag{5.15}$$

This means that the graph of $h_1(\lambda)$ (a hyperbola) is tangent to the graph of $\psi_1(\lambda)$ at $\lambda = \lambda_j$. The two conditions in equation (5.15) are the usual conditions in Newton's method, except instead of using a straight line approximation, we use a hyperbola. It is easy to verify that $c_1 = \psi_1'(\lambda_j)(d_i - \lambda_j)^2$ and $\hat{c}_1 = \psi_1(\lambda_j) - \psi_1'(\lambda_j)(d_i - \lambda_j)$. (See Question 5.17.)

Similarly, we choose c_2 and $\hat{c}_2$ so that

$$\begin{aligned} h_2(\lambda) &\equiv \hat{c}_2 + \frac{c_2}{d_{i+1} - \lambda} \quad \text{satisfies} \\ &\quad h_2(\lambda_j) = \psi_2(\lambda_j) \quad \text{and} \quad h_2'(\lambda_j) = \psi_2'(\lambda_j). \end{aligned} \tag{5.16}$$

Finally, we set

$$\begin{aligned} h(\lambda) &= 1 + h_1(\lambda) + h_2(\lambda) \\ &= (1 + \hat{c}_1 + \hat{c}_2) + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} \\ &\equiv c_3 + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda}. \end{aligned}$$

EXAMPLE 5.8. For example, in the example in Figure 5.3, if we start with $\lambda_0 = 2.5$, then

$$h(\lambda) = \frac{1.1111 \cdot 10^{-3}}{2 - \lambda} + \frac{1.1111 \cdot 10^{-3}}{3 - \lambda} + 1,$$

and its graph is visually indistinguishable from the graph of $f(\lambda)$ in the right-hand figure. Solving $h(\lambda_1) = 0$, we get $\lambda_1 = 2.0011$, which is accurate to 4 decimal digits. Continuing, λ_2 is accurate to 11 digits, and λ_3 is accurate to all 16 digits. $\diamond$

The algorithm used in LAPACK routine `slaed4` is a slight variation on the one described here (the one here is called *the Middle Way* in [170]). The LAPACK routine averages two to three iterations per eigenvalue to converge to full machine precision, and never took more than seven steps in extensive numerical tests.

Computing the Eigenvectors Stably

Once we have solved the secular equation to get the eigenvalues α_i of $D+\rho uu^T$, Lemma 5.2 provides a simple formula for the eigenvectors: $(D - \alpha_i I)^{-1}u$. Unfortunately, the formula can be unstable [58, 88, 232], in particular when two eigenvalues α_i and α_{i+1} are very close together. Intuitively, the problem is that $(D - \alpha_i I)^{-1}u$ and $(D - \alpha_{i+1} I)^{-1}n$ are "very close" formulas yet are supposed to yield orthogonal eigenvectors. More precisely, when α_i and α_{i+1} are very close, they must also be close to the d_i between them. Therefore, there is a great deal of cancellation, either when evaluating $d_i - \alpha_i$ and $d_i - \alpha_{i+1}$ or when evaluating the secular equation during Newton iteration. Either way, $d_i - \alpha_i$ and $d_i - \alpha_{i+1}$ may contain large relative errors, so the computed eigenvectors $(D - \alpha_i)^{-1}u$ and $(D - \alpha_{i+1})^{-1}u$ are quite inaccurate and far from orthogonal.

Early attempts to address this problem [88, 232] used double precision arithmetic (when the input data was single precision) to solve the secular equation to high accuracy so that $d_i - \alpha_i$ and $d_i - \alpha_{i+1}$ could be computed to high accuracy. But when the input data is already in double precision, this means quadruple precision would be needed, and this is not available in many machines and languages, or at least not cheaply. As described in section 1.5, it is possible to simulate quadruple precision using double precision [232, 202]. This can be done portably and relatively efficiently, as long as the underlying floating point arithmetic rounds sufficiently accurately. In particular, these simulations require that $\mathrm{fl}(a \pm b) = (a \pm b)(1 + \delta)$ with $|\delta| = O(\varepsilon)$, barring overflow or underflow (see section 1.5 and Question 1.18). Unfortunately, the Cray 2, YMP, and C90 do not round accurately enough to use these efficient algorithms.

Finally, an alternative formula was found that makes simulating high precision arithmetic unnecessary. It is based on the following theorem of Löwner [127, 177].

THEOREM 5.10. Löwner. *Let* $D = \mathrm{diag}(d_1, \ldots, d_n)$ *be diagonal with* $d_n < \cdots < d_1$. *Let* $\alpha_n < \cdots < \alpha_1$ *be given, satisfying the* interlacing property

$$d_n < \alpha_n < \cdots < d_{i+1} < \alpha_{i+1} < d_i < \alpha_i < \cdots < d_1 < \alpha_1.$$

Then there is a vector $\hat{u}$ *such that the* α_i *are the exact eigenvalues of* $\hat{D} \equiv$

$D + \hat{u}\hat{u}^T$. The entries of $\hat{u}$ are given by

$$|\hat{u}_i| = \left[\frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\ j=i}^{n}(d_j - d_i)} \right]^{1/2}.$$

Proof. The characteristic polynomial of $\hat{D}$ can be written both as $\det(\hat{D} - \lambda I) = \prod_{j=1}^{n}(\alpha_j - \lambda)$ and (using equations (5.13) and (5.14)) as

$$\begin{aligned} \det(\hat{D} - \lambda I) &= \left[\prod_{j=1}^{n}(d_j - \lambda) \right] \cdot (1 + \sum_{j=1}^{n} \frac{\hat{u}_j^2}{d_j - \lambda}) \\ &= \left[\prod_{j=1}^{n}(d_j - \lambda) \right] \cdot (1 + \sum_{\substack{j=1 \\ j=i}}^{n} \frac{\hat{u}_j^2}{d_j - \lambda}) \\ &\quad + \left[\prod_{\substack{j=1 \\ j=i}}^{n} (d_j - \lambda) \right] \cdot \hat{u}_i^2. \end{aligned}$$

Setting $\lambda = d_i$ and equating both expressions for $\det(\hat{D} - \lambda I)$ yield

$$\prod_{j=1}^{n}(\alpha_j - d_i) = \hat{u}_i^2 \cdot \prod_{\substack{j=1 \\ j=i}}^{n} (d_j - d_i)$$

or

$$\hat{u}_i^2 = \frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\ j=i}^{n}(d_j - d_i)}.$$

Using the interlacing property, we can show that the fraction on the right is positive, so we can take its square root to get the desired expression for $\hat{u}_i$. □

Here is the stable algorithm for computing the eigenvalues and eigenvectors (where we assume for simplicity of presentation that $\rho = 1$).

ALGORITHM 5.3. *Compute the eigenvalues and eigenvectors of $D + uu^T$.*

Solve the secular equation $1 + \sum_{i=1}^{n} \frac{u_i^2}{d_i - \lambda} = 0$ to get the eigenvalues α_i of $D + uu^T$.
Use Löwner's theorem to compute $\hat{u}$ so that the α_i are "exact" eigenvalues of $D + \hat{u}\hat{u}^T$.
Use Lemma 5.2 to compute the eigenvectors of $D + \hat{u}\hat{u}^T$.

Here is a sketch of why this algorithm is numerically stable. By analyzing the stopping criterion in the secular equation solver,[18] one can show that $\|uu^T - \hat{u}\hat{u}^T\|_2 \leq O(\varepsilon)(\|D\|_2 + \|uu^T\|_2)$; this means that $D + uu^T$ and $D + \hat{u}\hat{u}^T$ are so close together that the eigenvalues and eigenvectors of $D + \hat{u}\hat{u}^T$ are stable approximations of the eigenvalues and eigenvectors of $D + uu^T$. Next note that the formula for $\hat{u}_i$ in Löwner's theorem requires only differences of floating point numbers $d_j - d_i$ and $\alpha_j - d_i$, products and quotients of these differences, and a square root. *Provided that the floating point arithmetic is accurate enough* that $\mathrm{fl}(a \odot b) = (a \odot b)(1 + \delta)$ for all $\odot \in \{+, -, \times, /\}$ and $\mathrm{sqrt}(a) = \sqrt{a} \cdot (1 + \delta)$ with $|\delta| = O(\varepsilon)$, this formula can be evaluated to high relative accuracy. In particular, we can easily show that

$$\mathrm{fl}\left(\left[\frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\ j=i}^{n}(d_j - d_i)}\right]^{1/2}\right) = (1 + (4n-2)\delta) \cdot \left[\frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\ j=i}^{n}(d_j - d_i)}\right]^{1/2}$$

with $|\delta| = O(\varepsilon)$, barring overflow or underflow. Similarly, the formula in Lemma 5.2 can also be evaluated to high relative accuracy, so we can compute the eigenvectors of $D + \hat{u}\hat{u}^T$ to high relative accuracy. In particular, they are very accurately orthogonal.

In summary, provided the floating point arithmetic is accurate enough, Algorithm 5.3 computes very accurate eigenvalues and eigenvectors of a matrix $D + \hat{u}\hat{u}^T$ that differs only slightly from the original matrix $D + uu^T$. This means that it is numerically stable.

The reader should note that our need for sufficiently accurate floating point arithmetic is precisely what prevented the simulation of quadruple precision proposed in [232, 202] from working on some Crays. So we have not yet succeeded in providing an algorithm that works reliably on these machines. One more trick is necessary: The only operations that fail to be accurate enough on some Crays are addition and subtraction, because of the lack of a so-called *guard digit* in the floating point hardware. This means that the bottommost bit of an operand may be treated as 0 during addition or subtraction, even if it is 1. If most higher-order bits cancel, this "lost bit" becomes significant. For example, subtracting 1 from the next smaller floating point number, in which case all leading bits cancel, results in a number twice too large on the Cray C90 and in 0 on the Cray 2. But if the bottom bit is already 0, no harm is done. So the trick is to deliberately set all the bottom bits of the d_i to 0 before applying Löwner's theorem or Lemma 5.2 in Algorithm 5.3. This modification causes only a small relative change in the d_i and α_i, and so the algorithm is still stable.[19]

[18] In more detail, the secular equation solver must solve for $\alpha_i - d_i$ or $d_{i+1} - \alpha_i$ (whichever is smaller), *not* α_i, to attain this accuracy.

[19] To set the bottom bit of a floating point number β to 0 on a Cray, one can show that it is necessary only to set $\beta := (\beta + \beta) - \beta$. This inexpensive computation does not change β at all on a machine with accurate binary arithmetic (barring overflow, which is easily avoided).

This algorithm is described in more detail in [127, 129] and implemented in LAPACK routine `slaed3`.

Accelerating Divide-and-Conquer using the FMM

The FMM [122] was originally invented for the completely different problem of computing the mutual forces on n electrically charged particles or the mutual gravitational forces on n masses. We only sketch how these problems are related to finding eigenvalues and eigenvectors, leaving details to [129].

Let d_1 through d_n be the three-dimensional position vectors of n particles with charges $z_i \cdot u_i$. Let α_1 through α_n be the position vectors of n other particles with unit positive charges. Then the inverse-square law tells us that the force on the particle at α_j due to the particles at d_1 through d_n is proportional to

$$f_j = \sum_{i=1}^{n} \frac{z_i u_i (d_i - \alpha_j)}{\|d_i - \alpha_j\|_2^3}.$$

If we are modeling electrostatics in two dimensions instead of three, the force law changes to the inverse-first-power law[20]

$$f_j = \sum_{i=1}^{n} \frac{z_i u_i (d_i - \alpha_j)}{\|d_i - \alpha_j\|_2^2}.$$

Since d_i and α_j are vectors in $\mathbb{R}^2$, we can also consider them to be complex variables. In this case

$$f_j = \sum_{i=1}^{n} \frac{z_i u_i}{\bar{d}_i - \bar{\alpha}_j},$$

where $\bar{d}_i$ and $\bar{\alpha}_j$ are the complex conjugates of d_i and α_j, respectively. If d_i and α_j happen to be real numbers, this simplifies further to

$$f_j = \sum_{i=1}^{n} \frac{z_i u_i}{d_i - \alpha_j}.$$

Now consider performing a matrix-vector multiplication $f^T = z^T Q'$, where Q' is the eigenvector matrix of $D + uu^T$. From Lemma 5.2, $Q'_{ij} = u_i s_j/(d_i - \alpha_j)$,

But on a Cray it sets the bottom bit to 0. The reader familiar with Cray arithmetic is invited to prove this. The only remaining difficulty is preventing an optimizing compiler from removing this line of code entirely, which some overzealous optimizers might do; this is accomplished (for the current generation of compilers) by computing $(\beta + \beta)$ with a function call to a function stored in a separate file from the main routine. We hope that by the time compilers become clever enough to optimize even this situation, Cray arithmetic will have died out.

[20] Technically, this means the potential function satisfies Poisson's equation in two space coordinates rather than three.

where s_j is a scale factor chosen so that column j is a unit vector. Then the jth entry of $f^T = z^T Q'$ is

$$f_j = \sum_{i=1}^{n} z_i Q'_{ij} = s_j \sum_{i=1}^{n} \frac{z_i u_i}{d_i - \alpha_j},$$

which is the same sum as for the electrostatic force, except for the scale factor s_j. Thus, the most expensive part of the divide-and-conquer algorithm, the matrix multiplication in the last line of Algorithm 5.2, is equivalent to evaluating electrostatic forces.

Evaluating this sum for $j = 1, \ldots, n$ appears to require $O(n^2)$ flops. The FMM and others like it [122, 23] can be used to approximately (but very accurately) evaluate this sum in $O(n \cdot \log n)$ time (or even $O(n)$) time instead. (See the lectures on "Fast Hierarchical Methods for the N-body Problem" at PARALLEL_HOMEPAGE for details.)

But this idea alone is not enough to reduce the cost of divide-and-conquer to $O(n \cdot \log^p n)$. After all, the output eigenvector matrix Q has n^2 entries, which appears to mean that the complexity should be at least n^2. So we must represent Q using fewer than n^2 independent numbers. This is possible, because an n-by-n tridiagonal matrix has only $2n - 1$ "degrees of freedom" (the diagonal and superdiagonal entries), of which n can be represented by the eigenvalues, leaving $n-1$ for the orthogonal matrix Q. In other words, not every orthogonal matrix can be the eigenvector matrix of a symmetric tridiagonal T; only an $(n-1)$-dimensional subset of the entire $(n(n-1)/2)$-dimensional set of orthogonal matrices can be such eigenvector matrices.

We will represent Q using the divide-and-conquer tree computed by Algorithm 5.2. Rather than accumulating $Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q'$, we will store all the Q' matrices, one at each node in the tree. And we will not store Q' explicitly but rather just store D, ρ, u, and the eigenvalues α_i of $D + \rho u u^T$. We can do this since this is all we need to use the FMM to multiply by Q'. This reduces the storage needed for Q from n^2 to $O(n \cdot \log n)$. Thus, the output of the algorithm is a "factored" form of Q consising of all the Q' factors at the nodes of the tree. This is an adequate representation of Q, because we can use the FMM to multiply any vector by Q in $O(n \cdot \log^p n)$ time.

5.3.4. Bisection and Inverse Iteration

The bisection algorithm exploits Sylvester's inertia theorem (Theorem 5.3) to find only those k eigenvalues that one wants, at cost $O(nk)$. Recall that Inertia$(A) = (\nu, \zeta, \pi)$, where ν, ζ, and π are the number of negative, zero, and positive eigenvalues of A, respectively. Suppose that X is nonsingular; Sylvester's inertia theorem asserts that Inertia$(A) =$ Inertia$(X^T A X)$.

Now suppose that one uses Gaussian elimination to factorize $A - zI = LDL^T$, where L is nonsingular and D diagonal. Then Inertia$(A - zI) =$

Inertia(D). Since D is diagonal, its inertia is trivial to compute. (In what follows, we use notation such as "# $d_{ii} < 0$" to mean "the number of values of d_{ii} that are less than zero.")

$$\begin{aligned}\text{Inertia}(A - zI) &= (\#\ d_{ii} < 0,\ \#\ d_{ii} = 0,\ \#\ d_{ii} > 0) \\ &= (\#\ \text{negative eigenvalues of } A - zI, \\ &\quad \#\ \text{zero eigenvalues of } A - zI, \\ &\quad \#\ \text{positive eigenvalues of } A - zI) \\ &= (\#\ \text{eigenvalues of } A < z, \\ &\quad \#\ \text{eigenvalues of } A = z, \\ &\quad \#\ \text{eigenvalues of } A > z).\end{aligned}$$

Suppose $z_1 < z_2$ and we compute Inertia $(A - z_1 I)$ and Inertia $(A - z_2 I)$. Then the number of eigenvalues in the interval $[z_1, z_2)$ equals (# eigenvalues of $A < z_2$) – (# eigenvalues of $A < z_1$).

To make this observation into an algorithm, define

$$\text{Negcount}(A, z) = \ \#\ \text{eigenvalues of } A < z.$$

ALGORITHM 5.4. *Bisection: Find all eigenvalues of A inside $[a, b)$ to a given error tolerance* tol:

n_a = Negcount(A, a)
n_b = Negcount(A, b)
if $n_a = n_b$, quit ... because there are no eigenvalues in $[a, b)$
put $[a, n_a, b, n_b]$ onto Worklist
    / Worklist contains a list of intervals $[a, b)$ containing*
        eigenvalues $n - n_a + 1$ through $n - n_b$, which the algorithm
        will repeatedly bisect until they are narrower than tol. **/*
while Worklist is not empty do
    remove $[\text{low}, n_{\text{low}}, \text{up}, n_{\text{up}}]$ *from Worklist*
    if (up−low < tol*) then*
        print "there are $n_{\text{up}} - n_{\text{low}}$ eigenvalues in $[\text{low}, \text{up})$*"*
    else
        mid = (low + up)/2
        n_{mid} *= Negcount (A, mid)*
        if $n_{\text{mid}} > n_{\text{low}}$ then ... there are eigenvalues in $[\text{low}, \text{mid})$
            put $[\text{low}, n_{\text{low}}, \text{mid}, n_{\text{mid}}]$ *onto Worklist*
        end if
        if $n_{\text{up}} > n_{\text{mid}}$ then ... there are eigenvalues in $[\text{mid}, \text{up})$
            put $[\text{mid}, n_{\text{mid}}, \text{up}, n_{\text{up}}]$ *onto Worklist*
        end if
    end if
end while

If $\alpha_1 \geq \cdots \geq \alpha_n$ are eigenvalues, the same idea can be used to compute α_j for $j = j_0, j_0 + 1, \ldots, j_1$. This is because we know $\alpha_{n-n_{\text{up}}+1}$ through $\alpha_{n-n_{\text{low}}}$ lie in the interval [low,up).

If A were dense, we could implement Negcount(A, z) by doing symmetric Gaussian elimination with pivoting as described in section 2.7.2. But this would cost $O(n^3)$ flops per evaluation and so not be cost effective. On the other hand, Negcount(A, z) is quite simple to compute for tridiagonal A, provided that we do not pivot:

$$A - zI \equiv \begin{bmatrix} a_1 - z & b_1 & & \\ b_1 & a_2 - z & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix} = LDL^T$$

$$\equiv \begin{bmatrix} 1 & & & \\ l_1 & \ddots & & \\ & \ddots & \ddots & \\ & & l_{n-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \cdot \begin{bmatrix} 1 & l_1 & & \\ & \ddots & \ddots & \\ & & \ddots & l_{n-1} \\ & & & 1 \end{bmatrix},$$

so $a_1 - z = d_1, d_1 l_1 = b_1$ and thereafter $l_{i-1}^2 d_{i-1} + d_i = a_i - z$, $d_i l_i = b_i$. Substituting $l_i = b_i/d_i$ into $l_{i-1}^2 d_{i-1} + d_i = a_i - z$ yields the simple recurrence

$$d_i = (a_i - z) - \frac{b_{i-1}^2}{d_{i-1}}. \tag{5.17}$$

Notice that we are not pivoting, so you might think that this is dangerously unstable, especially when d_{i-1} is small. In fact, since it is tridiagonal, it can be shown to be very stable [72, 73, 154].

LEMMA 5.3. *The d_i computed in floating point arithmetic, using equation* (5.17)*, have the same signs (and so compute the same Inertia) as the $\hat{d}_i$ computed exactly from $\hat{A}$, where $\hat{A}$ is very close to A:*

$$(\hat{A})_{ii} \equiv \hat{a}_i = a_i \;\; \textit{and} \;\; (\hat{A})_{i,i+1} \equiv \hat{b}_i = b_i(1 + \epsilon_i), \;\; \textit{where } |\epsilon_i| \le 2.5\varepsilon + O(\varepsilon^2).$$

Proof. Let $\tilde{d}_i$ denote the quantities computed using equation (5.17) including rounding errors:

$$\tilde{d}_i = \left[(a_i - z)(1 + \epsilon_{-,1,i}) - \frac{b_{i-1}^2(1 + \epsilon_{*,i})}{\tilde{d}_{i-1}} \cdot (1 + \epsilon_{/,i}) \right] (1 + \epsilon_{-,2,i}), \tag{5.18}$$

where all the ϵ's are bounded by machine roundoff ε in magnitude, and their subscripts indicate which floating point operation they come from (for example, $\epsilon_{-,2,i}$ is from the second subtraction when computing $\tilde{d}_i$). Define the new variables

$$\hat{d}_i = \frac{\tilde{d}_i}{(1 + \epsilon_{-,1,i})(1 + \epsilon_{-,2,i})} \tag{5.19}$$

$$\hat{b}_{i-1} = b_{i-1} \left[\frac{(1 + \epsilon_{*,i})(1 + \epsilon_{/,i})}{(1 + \epsilon_{-,1,i})(1 + \epsilon_{-,1,i-1})(1 + \epsilon_{-,2,i-1})} \right]^{1/2} \equiv b_{i-1}(1 + \epsilon_i).$$

Note that $\hat{d}_i$ and $\tilde{d}_i$ have the same signs, and $|\epsilon_i| \leq 2.5\varepsilon + O(\varepsilon^2)$. Substituting (5.19) into (5.18) yields

$$\hat{d}_i = a_i - z - \frac{\hat{b}_{i-1}^2}{\hat{d}_{i-1}},$$

completing the proof. □

A complete analysis must take the possibility of overflow or underflow into account. Indeed, using the exception handling facilities of IEEE arithmetic, one can safely compute even when some d_{i-1} is exactly zero! For in this case $d_i = -\infty$, $d_{i+1} = a_{i+1} - z$, and the computation continues unexceptionally [72, 80].

The cost of a single call to Negcount on a tridiagonal matrix is $4n$ flops. Therefore the overall cost to find k eigenvalues is $O(kn)$. This is implemented in LAPACK routine `sstebz`.

Note that Bisection converges linearly, with one more bit of accuracy for each bisection of an interval. There are many ways to accelerate convergence, using algorithms like Newton's method and its relatives, to find zeros of the characteristic polynomial (which may be computed by multiplying all the d_i's together) [171, 172, 173, 174, 176, 267].

To compute eigenvectors once we have computed (selected) eigenvalues, we can use inverse iteration (Algorithm 4.2); this is available in LAPACK routine `sstein`. Since we can use accurate eigenvalues as shifts, convergence usually takes one or two iterations. In this case the cost is $O(n)$ flops per eigenvector, since one step of inverse iteration requires us only to solve a tridiagonal system of equations (see section 2.7.3). When several computed eigenvalues $\hat{\alpha}_i, \ldots, \hat{\alpha}_j$ are close together, their corresponding computed eigenvectors $\hat{q}_i, \ldots, \hat{q}_j$ may not be orthogonal. In this case the algorithm *reorthogonalizes* the computed eigenvectors, computing the QR decomposition $[\hat{q}_i, \ldots, \hat{q}_j] = QR$ and replacing each $\hat{q}_k$ with the kth column of Q; this guarantees that the $\hat{q}_k$ are orthonormal. This QR decomposition is usually computed using the MGS orthogonalization process (Algorithm 3.1); i.e., each computed eigenvector has any components in the directions of previously computed eigenvectors explicitly subtracted out. When the cluster size $k = j - i + 1$ is small, the cost $O(k^2 n)$ of this reorthogonalization is small, so in principle all the eigenvalues and all the eigenvectors could be computed by Bisection followed by inverse iteration in just $O(n^2)$ flops total. This is much faster than the $O(n^3)$ cost of QR iteration or divide-and-conquer (in the worst case). The obstacle to obtaining this speedup reliably is that if the cluster size $k = j - i + 1$ is large, i.e., a sizable fraction of n, then the total cost rises to $O(n^3)$ again. Worse, there is no guarantee that the computed eigenvectors are accurate or orthogonal. (The trouble is that after reorthogonalizing a set of nearly dependent $\hat{q}_k$, cancellation may mean some computed eigenvectors consist of little more than roundoff errors.)

There has been recent progress on this problem, however [103, 199, 201], and it now appears possible that inverse iteration may be "repaired" to provide

accurate, orthogonal eigenvectors without spending more than $O(n)$ flops per eigenvector to reorthogonalize. This would make Bisection and "repaired" inverse iteration the algorithm of choice in all cases, no matter how many eigenvalues and eigenvectors are desired. We look forward to describing this algorithm in a future edition.

Note that Bisection and inverse iteration are "embarrassingly parallel," since each eigenvalue and later eigenvector may be found independently of the others. (This presumes that inverse iteration has been repaired so that reorthogonalization with many other eigenvectors is no longer necessary.) This makes these algorithms very attractive for parallel computers [75].

5.3.5. Jacobi's Method

Jacobi's method does not start by reducing A to tridiagonal from as do the previous methods but instead works on the original dense matrix. Jacobi's method is usually much slower than the previous methods and remains of interest only because it can sometimes compute tiny eigenvalues and their eigenvectors with much higher accuracy than the previous methods and can be easily parallelized. Here we describe only the basic implementation of Jacobi's method, and defer the discussion of high accuracy to section 5.4.3.

Given a symmetric matrix $A = A_0$, Jacobi's method produces a sequence $A_1, A_2, \ldots$ of orthogonally similar matrices, which eventually converge to a diagonal matrix with the eigenvalues on the diagonal. A_{i+1} is obtained from A_i by the formula $A_{i+1} = J_i^T A_i J_i$, where J_i is an orthogonal matrix called a *Jacobi rotation*. Thus

$$\begin{aligned} A_m &= J_{m-1}^T A_{m-1} J_{m-1} \\ &= J_{m-1}^T J_{m-2}^T A_{m-2} J_{m-2} J_{m-1} = \cdots \\ &= J_{m-1}^T \cdots J_0^T A_0 J_0 \cdots J_{m-1} \\ &\equiv J^T A J. \end{aligned}$$

If we choose each J_i appropriately, A_m approaches a diagonal matrix Λ for large m. Thus we can write $\Lambda \approx J^T A J$ or $J \Lambda J^T \approx A$. Therefore, the columns of J are approximate eigenvectors.

We will make $J^T A J$ nearly diagonal by iteratively choosing J_i to make *one* pair of offdiagonal entries of $A_{i+1} = J_i^T A_i J_i$ zero at a time. We will do this by

choosing J_i to be a Givens rotation,

$$J_i = R(j,k,\theta) \equiv \begin{array}{c} \\ j \\ \\ k \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & j & & k & & & \end{array} \\ \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\theta & & -\sin\theta & & & \\ & & & & \ddots & & & & \\ & & & \sin\theta & & \cos\theta & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix} \end{array},$$

where θ is chosen to zero out the j,k and k,j entries of A_{i+1}. To determine θ (or actually $\cos\theta$ and $\sin\theta$), write

$$\begin{aligned} \begin{bmatrix} a_{jj}^{(i+1)} & a_{jk}^{(i+1)} \\ a_{kj}^{(i+1)} & a_{kk}^{(i+1)} \end{bmatrix} &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T \begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \end{aligned}$$

where λ_1 and λ_2 are the eigenvalues of

$$\begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix}.$$

It is easy to compute $\cos\theta$ and $\sin\theta$: Multiplying out the last expression, using symmetry, abbreviating $c \equiv \cos\theta$ and $s \equiv \sin\theta$, and dropping the superscript (i) for simplicity yield

$$\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} a_{jj}c^2 + a_{kk}s^2 + 2sca_{jk} & sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2) \\ sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2) & a_{jj}s^2 + a_{kk}c^2 - 2sca_{jk} \end{bmatrix}.$$

Setting the offdiagonals to 0 and solving for θ we get $0 = sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)$, or

$$\frac{a_{jj} - a_{kk}}{2a_{jk}} = \frac{c^2 - s^2}{2sc} = \frac{\cos 2\theta}{\sin 2\theta} = \cot 2\theta \equiv \tau.$$

We now let $t = \frac{s}{c} = \tan\theta$ and note that $t^2 + 2\tau t - 1 = 0$ to get (via the quadratic formula) $t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$, $c = \frac{1}{\sqrt{1+t^2}}$ and $s = t \cdot c$. We summarize this derivation in the following algorithm.

ALGORITHM 5.5. *Compute and apply a Jacobi rotation to A in coordinates j,k:*

proc Jacobi-Rotation (A, j, k)
  if $|a_{jk}|$ *is not too small*
    $\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$
    $t = sign(\tau)/(|\tau| + \sqrt{1+\tau^2})$
    $c = 1/\sqrt{1+t^2}$
    $s = c \cdot t$
    $A = R^T(j,k,\theta) \cdot A \cdot R(j,k,\theta)$ ... *where* $c = \cos\theta$ *and* $s = \sin\theta$
    if eigenvectors are desired
      $J = J \cdot R(j,k,\theta)$
    end if
  end if

The cost of applying $R(j,k,\theta)$ to A (or J) is only $O(n)$ flops, because only rows and columns j and k of A (and columns j and k of J) are modified. The overall Jacobi algorithm is then as follows.

ALGORITHM 5.6. *Jacobi's method to find the eigenvalues of a symmetric matrix:*

repeat
  choose a j, k *pair*
  call Jacobi-Rotation(A, j, k)
until A *is sufficiently diagonal*

We still need to decide how to pick j, k pairs. There are several possibilities. To measure progress to convergence and describe these possibilities, we define

$$\mathrm{off}(A) \equiv \sqrt{\sum_{1 \le j < k \le n} a_{jk}^2}.$$

Thus $\mathrm{off}(A)$ is the root-sum-of-squares of the (upper) offdiagonal entries of A, so A is diagonal if and only if $\mathrm{off}(A) = 0$. Our goal is to make $\mathrm{off}(A)$ approach 0 quickly. The next lemma tells us that $\mathrm{off}(A)$ decreases monotonically with every Jacobi rotation.

LEMMA 5.4. *Let* A' *be the matrix after calling Jacobi-Rotation*(A, j, k) *for any* $j = k$. *Then* $\mathrm{off}^2(A') = \mathrm{off}^2(A) - a_{jk}^2$.

Proof. Note that $A' = A$ except in rows and columns j and k. Write

$$\mathrm{off}^2(A) = \left(\sum_{\substack{1 \le j' < k' \le n \\ j' = j \text{ or } k' = k}} a_{j'k'}^2 \right) + a_{jk}^2 \equiv S^2 + a_{jk}^2$$

and similarly $\text{off}^2(A') = S'^2 + a'^2_{jk} = S'^2$, since $a'_{jk} = 0$ after calling Jacobi-Rotation(A, j, k). Since $\|X\|_F = \|QX\|_F$ and $\|X\|_F = \|XQ\|_F$ for any X and any orthogonal Q, we can show $S^2 = S'^2$. Thus $\text{off}^2(A') = \text{off}^2(A) - a^2_{jk}$ as desired. □

The next algorithm was the original version of the algorithm (from Jacobi in 1846), and it has an attractive analysis although it is too slow to use.

ALGORITHM 5.7. *Classical Jacobi's algorithm:*

while $\text{off}(A) >$ tol *(where* tol *is the stopping criterion set by user)*
  choose j *and* k *so* a_{jk} *is the largest offdiagonal entry in magnitude*
  call Jacobi-Rotation (A, j, k)
end while

THEOREM 5.11. *After one Jacobi rotation in the classical Jacobi's algorithm, we have* $\text{off}(A') \le \sqrt{1 - \frac{1}{N}}\ \text{off}(A)$ *where* $N = \frac{n(n-1)}{2}$ *= the number of superdiagonal entries of* A*. After* k *Jacobi-Rotations* $\text{off}(\cdot)$ *is no more than* $\left(1 - \frac{1}{N}\right)^{k/2} \text{off}(A)$.

Proof. By Lemma 5.4, after one step, $\text{off}^2(A') = \text{off}^2(A) - a^2_{jk}$, where a_{jk} is the largest offdiagonal entry. Thus $\text{off}^2(A) \le \frac{n(n-1)}{2} a^2_{jk}$, or $a^2_{jk} \ge \frac{1}{n(n-1)/2} \text{off}^2(A)$, so $\text{off}^2(A) - a^2_{jk} \le \left(1 - \frac{1}{N}\right) \text{off}^2(A)$ as desired. □

So the classical Jacobi's algorithm converges at least linearly with the error (measured by $\text{off}(A)$) decreasing by a factor of at least $\sqrt{1 - \frac{1}{N}}$ at a time. In fact, it eventually converges quadratically.

THEOREM 5.12. *Jacobi's method is locally quadratically convergent after* N *steps (i.e., enough steps to choose each* a_{jk} *once). This means that for* i *large enough*

$$\text{off}(A_{i+N}) = O(\text{off}^2(A_i)).$$

In practice, we do not use the classical Jacobi's algorithm because searching for the largest entry is too slow: We would need to search $\frac{n^2-n}{2}$ entries for every Jacobi rotation, which costs only $O(n)$ flops to perform, and so for large n the search time would dominate. Instead, we use the following simple method to choose j and k.

ALGORITHM 5.8. *Cyclic-by-row-Jacobi: Sweep through the offdiagonals of* A *rowwise.*

repeat
  for $j = 1$ *to* $n - 1$
    for $k = j + 1$ *to* n

call Jacobi-Rotation(A, j, k)
end for
end for
until A *is sufficiently diagonal*

A no longer changes when Jacobi-Rotation(A, j, k) chooses only $c = 1$ and $s = 0$ for an entire pass through the inner loop. The cyclic Jacobi's algorithm is also asymptotically quadratically convergent like the classical Jacobi's algorithm [260, p. 270].

The cost of one Jacobi "sweep" (where each j, k pair is selected once) is approximately half the cost of reduction to tridiagonal form and the computation of eigenvalues and eigenvectors using QR iteration, and more than the cost using divide-and-conquer. Since Jacobi's method often takes 5–10 sweeps to converge, it is much slower than the competition.

5.3.6. Performance Comparison

In this section we analyze the performance of the three fastest algorithms for the symmetric eigenproblem: QR iteration, Bisection with inverse iteration, and divide-and-conquer. More details may be found in [10, chap. 3] or NETLIB/lapack/lug/lapack_lug.html.

We begin by discussing the fastest algorithm and later compare the others. We used the LAPACK routine `ssyevd`. The algorithm to find only eigenvalues is reduction to tridiagonal form followed by QR iteration, for an operation count of $\frac{4}{3}n^3 + O(n^2)$ flops. The algorithm to find eigenvalues and eigenvectors is tridiagonal reduction followed by divide-and-conquer. We timed `ssyevd` on an IBM RS6000/590, a workstation with a peak speed of 266 Mflops, although optimized matrix-multiplication runs at only 233 Mflops for 100-by-100 matrices and 256 Mflops for 1000-by-1000 matrices. The actual performance is given in the table below. The "Mflop rate" is the actual speed of the code in Mflops, and "Time / Time(Matmul)" is the time to solve the eigenproblem divided by the time to multiply two square matrices of the same size. We see that for large enough matrices, matrix-multiplication and finding only the eigenvalues of a symmetric matrix are about equally expensive. (In contrast, the nonsymmetric eigenproblem is least 16 times more costly [10].) Finding the eigenvectors as well is a little under three times as expensive as matrix-multiplication.

Dimension	Eigenvalues only		Eigenvalues and eigenvectors	
	Mflop rate	Time / Time(Matmul)	Mflop rate	Time / Time(Matmul)
100	72	3.1	72	9.3
1000	160	1.1	174	2.8

Now we compare the relative performance of QR iteration, Bisection with inverse iteration, and divide-and-conquer. In Figures 5.4 and 5.5 these are la-

beled QR, BZ (for the LAPACK routine `sstebz`, which implements Bisection), and DC, respectively. The horizontal axis in these plots is matrix dimension, and the vertical axis is time divided by the time for DC. Therefore, the DC curve is a horizontal line at 1, and the other curves measure how many times slower BZ and QR are than DC. Figure 5.4 shows only the time for the tridiagonal eigenproblem, whereas Figure 5.5 shows the entire time, starting from a dense matrix.

In the top graph in Figure 5.5 the matrices tested were random symmetric matrices; in Figure 5.4, the tridiagonal matrices were obtained by reducing these dense matrices to tridiagonal form. Such random matrices have well-separated eigenvalues on average, so inverse iteration requires little or no expensive reorthogonalization. Therefore BZ was comparable in performance to DC, although QR was significantly slower, up to 15 times slower in the tridiagonal phase on large matrices.

In the bottom two graphs, the dense symmetric matrices had eigenvalues 1, .5, .25, ..., $.5^{n-1}$. In other words, there were many eigenvalues clustered near zero, so inverse iteration had a lot of reorthogonalization to do. Thus the tridiagonal part of BZ was over 70 times slower than DC. QR was up to 54 times slower than DC, too, because DC actually *speeds up* when there is a large cluster of eigenvalues; this is because of deflation.

The distinction in speeds among QR, BZ, and DC is less noticeable in Figure 5.5 than in Figure 5.4, because Figure 5.5 includes the common $O(n^3)$ overhead of reduction to tridiagonal form and transforming the eigenvalues of the tridiagonal matrix to eigenvalues of the original dense matrix; this common overhead is labeled TRD. Since DC is so close to TRD in Figure 5.5, this means that any further acceleration of DC will make little difference in the overall speed of the dense algorithm.

5.4. Algorithms for the Singular Value Decomposition

In Theorem 3.3, we showed that the SVD of the general matrix G is closely related to the eigendecompositions of the symmetric matrices G^TG, GG^T and $\begin{bmatrix} 0 & G^T \\ G & 0 \end{bmatrix}$. Using these facts, the algorithms in the previous section can be transformed into algorithms for the SVD. The transformations are not straightforward, however, because the added structure of the SVD can often be exploited to make the algorithms more efficient or more accurate [118, 79, 66].

All the algorithms for the eigendecomposition of a symmetric matrix A, except Jacobi's method, have the following structure:

1. Reduce A to tridiagonal form T with an orthogonal matrix Q_1: $A = Q_1 T Q_1^T$.

2. Find the eigendecomposition of T: $T = Q_2 \Lambda Q_2^T$, where Λ is the diagonal matrix of eigenvalues and Q_2 is the orthogonal matrix whose columns

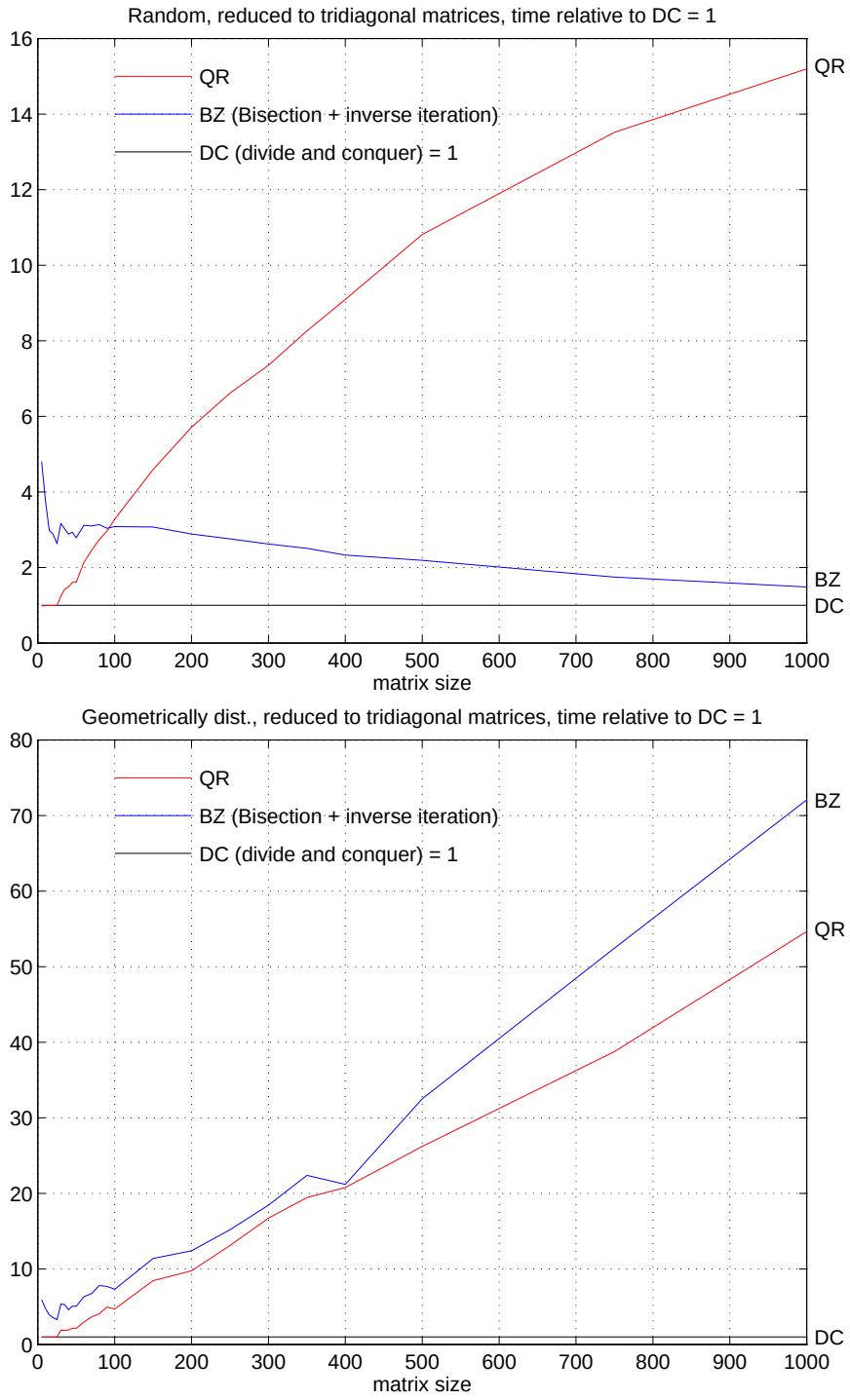


Fig. 5.4. *Speed of finding eigenvalue and eigenvectors of a symmetric tridiagonal matrix, relative to divide-and-conquer.*

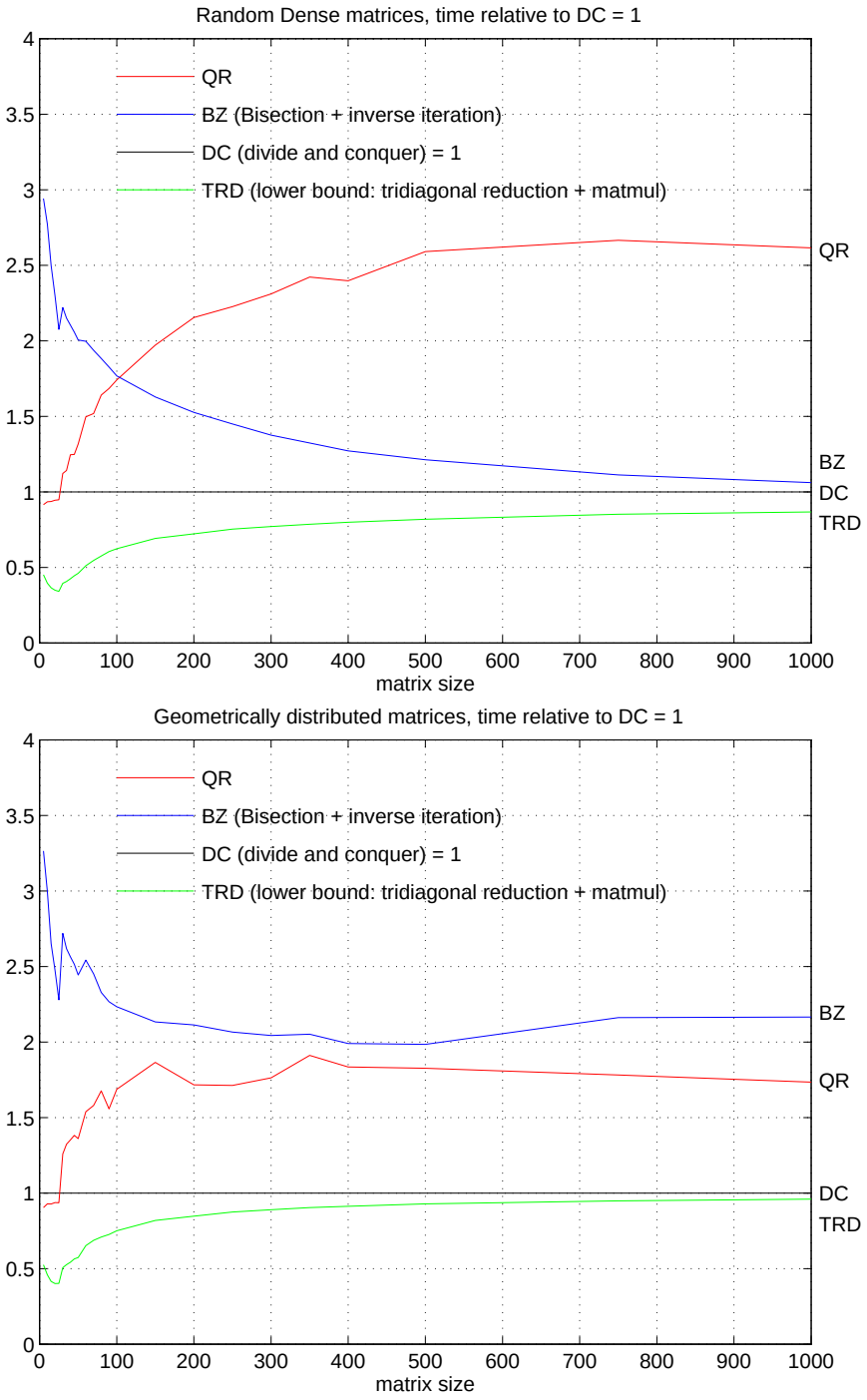


Fig. 5.5. *Speed of finding eigenvalue and eigenvectors of a symmetric dense matrix, relative to divide-and-conquer.*

are eigenvectors.

3. Combine these decompositions to get $A = (Q_1Q_2)\Lambda(Q_1Q_2)^T$. The columns of $Q = Q_1Q_2$ are the eigenvectors of A.

All the algorithms for the SVD of a general matrix G, except Jacobi's method, have an analogous structure:

1. Reduce G to bidiagonal form B with orthogonal matrices U_1 and V_1: $G = U_1BV_1^T$. This means B is nonzero only on the main diagonal and first superdiagonal.

2. Find the SVD of B: $B = U_2\Sigma V_2^T$, where Σ is the diagonal matrix of singular values, and U_2 and V_2 are orthogonal matrices whose columns are the left and right singular vectors, respectively.

3. Combine these decompositions to get $G = (U_1U_2)\Sigma(V_1V_2)^T$. The columns of $U = U_1U_2$ and $V = V_1V_2$ are the left and right singular vectors of G, respectively.

Reduction to bidiagonal form is accomplished by the algorithm in section 4.4.7. Recall from the discussion there that it costs $\frac{8}{3}n^3 + O(n^2)$ flops to compute B; this is all that is needed if only the singular values Σ are to be computed. It costs another $4n^3 + O(n^2)$ flops to compute U_1 and V_1, which are needed to compute the singular vectors as well.

The following simple lemma shows how to convert the problem of finding the SVD of the bidiagonal matrix B into the eigendecomposition of a symmetric tridiagonal matrix T.

LEMMA 5.5. *Let B be an n-by-n bidiagonal matrix, with diagonal $a_1, \ldots, a_n$ and superdiagonal $b_1, \ldots, b_{n-1}$. There are three ways to convert the problem of finding the SVD of B to finding the eigenvalues and eigenvectors of a symmetric tridiagonal matrix.*

1. *Let $A = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$. Let P be the permutation matrix $P = [e_1, e_{n+1}, e_2, e_{n+2}, \ldots, e_n, e_{2n}]$, where e_i is the ith column of the $2n$-by-$2n$ identity matrix. Then $T_{ps} \equiv P^TAP$ is symmetric tridiagonal. The subscript "ps" stands for* perfect shuffle*, because multiplying P times a vector x "shuffles" the entries of x like a deck of cards. One can show that T_{ps} has all zeros on its main diagonal, and its superdiagonal and subdiagonal is $a_1, b_1, a_2, b_2, \ldots, b_{n-1}, a_n$. If $T_{ps}x_i = \alpha_i x_i$ is an eigenpair for T_{ps}, with x_i a unit vector, then $\alpha_i = \pm\sigma_i$, where σ_i is a singular value of B, and $Px_i = \frac{1}{\sqrt{2}}\begin{bmatrix} v_i \\ \pm u_i \end{bmatrix}$, where u_i and v_i are left and right singular vectors of B, respectively.*

2. *Let $T_{BB^T} \equiv BB^T$. Then T_{BB^T} is symmetric tridiagonal with diagonal $a_1^2 + b_1^2, a_2^2 + b_2^2, \ldots, a_{n-1}^2 + b_{n-1}^2, a_n^2$, and superdiagonal and subdiagonal $a_2 b_1, a_3 b_2, \ldots, a_n b_{n-1}$. The singular values of B are the square roots of the eigenvalues of T_{BB^T}, and the left singular vectors of B are the eigenvectors of T_{BB^T}. T_{BB^T} contains no information about the right singular vectors of B.*

3. *Let $T_{B^TB} \equiv B^TB$. Then T_{B^TB} is symmetric tridiagonal with diagonal $a_1^2, a_2^2 + b_1^2, a_3^2 + b_2^2, \ldots, a_n^2 + b_{n-1}^2$, and superdigonal and subdiagonal $a_1 b_1, a_2 b_2, \ldots, a_{n_1} b_{n-1}$. The singular values of B are the square roots of the eigenvalues of T_{B^TB}, and the right singular vectors of B are the eigenvectors of T_{B^TB}. T_{B^TB} contains no information about the left singular vectors of B.*

For a proof, see Question 5.19.

Thus, we could in principle apply any of QR iteration, divide-and-conquer, or Bisection with inverse iteration to one of the tridiagonal matrices from Lemma 5.5 and then extract the singular and (perhaps only left or right) singular vectors from the resulting eigendecomposition. However, this simple approach would sacrifice both speed and accuracy by ignoring the special properties of the underlying SVD problem. We give two illustrations of this.

First, it would be inefficient to run symmetric tridiagonal QR iteration or divide-and-conquer on T_{ps}. This is because these algorithms both compute all the eigenvalues (and perhaps eigenvectors) of T_{ps}, whereas Lemma 5.5 tells us we only need the nonnegative eigenvalues (and perhaps eigenvectors). There are some accuracy difficulties with singular vectors for tiny singular values too.

Second, explicitly forming either T_{BB^T} or T_{B^TB} is numerically unstable. In fact one can lose half the accuracy in the small singular values of B. For example, let $\eta = \varepsilon/2$, so $1 + \eta$ rounds to 1 in floating point arithmetic. Let $B = \begin{bmatrix} 1 & 1 \\ 0 & \sqrt{\eta} \end{bmatrix}$, which has singular values near $\sqrt{2}$ and $\sqrt{\eta/2}$. Then $B^TB = \begin{bmatrix} 1 & 1 \\ 1 & 1+\eta \end{bmatrix}$ rounds to $T_{B^TB} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, an exactly singular matrix. Thus, rounding $1 + \eta$ to 1 changes the smaller computed singular value from its true value near $\sqrt{\eta/2} = \sqrt{\varepsilon}/2$ to 0. In contrast, a backward stable algorithm should change the singular values by no more than $O(\varepsilon)\|B\|_2 = O(\varepsilon)$. In IEEE double precision floating point arithmetic, $\varepsilon \approx 10^{-16}$ and $\sqrt{\varepsilon}/2 \approx 10^{-8}$, so the error introduced by forming B^TB is 10^8 times larger than roundoff, a much larger change. The same loss of accuracy can occur by explicitly forming T_{BB^T}.

Because of the instability caused by computing T_{BB^T} or T_{B^TB}, good SVD algorithms work directly on B or possibly T_{ps}.

In summary, we describe the practical algorithms used for computing the SVD.

1. *QR iteration and its variations.* Properly implemented [102], this is the fastest algorithm for finding all the singular values of a bidiagonal matrix.

Furthermore, it finds all the singular values to high relative accuracy, as discussed in section 5.2.1, This means that all the digits of all the singular values are correct, even the tiniest ones. In contrast, symmetric tridiagonal QR iteration may compute tiny eigenvalues with no relative accuracy at all. A different variation of QR iteration [79] is used to compute the singular vectors as well: by using QR iteration with a zero shift to compute the smallest singular vectors, this variation computes the singular values nearly as accurately, as well as getting singular vectors as accurately as described in section 5.2.1. But this is only the fastest algorithm for small matrices, up to about dimension $n = 25$. This routine is available in LAPACK subroutine `sbdsqr`.

2. *Divide-and-conquer.* This is currently the fastest method to find all singular values and singular vectors for matrices larger than $n = 25$. (The implementation in LAPACK, `sbdsdc`, defaults to `sbdsqr` for small matrices.) However, divide-and-conquer does not guarantee that the tiny singular values are computed to high relative accuracy. Instead, it guarantees only the same error bound as in the symmetric eigenproblem: the error in singular value σ_j is at most $O(\varepsilon)\sigma_1$ rather than $O(\varepsilon)\sigma_j$. This sufficiently accurate for most applications.

3. *Bisection and inverse iteration.* One can apply Bisection and inverse iteration to T_{ps} of part 1 of Lemma 5.5, to find only the singular values in a desired interval. This algorithm is guaranteed to find the singular values to high relative accuracy, although the singular vectors may occasionally suffer loss of orthogonality as described in section 5.3.4.

4. *Jacobi's method.* We may compute the SVD of a dense matrix G by applying Jacobi's method of section 5.3.5 *implicitly* to GG^T or G^TG, i.e., without explicitly forming either one and so possibly losing stability. For some classes of G, i.e., those to which we can profitably apply the relative perturbation theory of section 5.2.1, we can show that Jacobi's method computes the singular values and singular vectors to high relative accuracy, as described in section 5.2.1.

The following sections describe some of the above algorithms in more detail, notably QR iteration and its variation dqds in section 5.4.1; the proof of high accuracy of dqds and Bisection in section 5.4.2; and Jacobi's method in section 5.4.3. We omit divide-and-conquer because of its overall similarity to the algorithm discussed in section 5.3.3, and refer the reader to [128] for details.

5.4.1. QR Iteration and Its Variations for the Bidiagonal SVD

There is a long history of variations on QR iteration for the SVD, designed to be as efficient and accurate as possible; see [198] for a good survey. The

algorithm in the LAPACK routine `sbdsqr` was originally based on [79] and later updated to use the algorithm in [102] in the case when singular values only are desired. This latter algorithm, called dqds for historical reasons,[21] is elegant, fast, and accurate, so we will present it.

To derive dqds, we begin with an algorithm that predates QR iteration, called LR iteration, specialized to symmetric positive definite matrices.

ALGORITHM 5.9. *LR Iteration; Let T_0 be any symmetric positive definite matrix. The following algorithm produces a sequence of similar symmetric positive definite matrices T_i:*

$i = 0$
repeat
  Choose a shift τ_i^2 smaller than the smallest eigenvalue of T_i.
  Compute the Cholesky factorization $T_i - \tau_i^2 I = B_i^T B_i$
    (B_i is an upper triangular matrix with positive diagonal.)
  $T_{i+1} = B_i B_i^T + \tau_i^2 I$
  $i = i + 1$
until convergence

LR iteration is very similar in structure to QR iteration: We compute a factorization, and multiply the factors in reverse order to get the next iterate T_{i+1}. It is easy to see that T_{i+1} and T_i are similar: $T_{i+1} = B_i B_i^T + \tau_i^2 I = B_i^{-T} B_i^T B_i B_i^T + \tau_i^2 B_i^{-T} B_i^T = B_i^{-T} T_i B_i^T$.

In fact, when the shift $\tau_i^2 = 0$, we can show that two steps of LR iteration produce the same T_2 as one step of QR iteration.

LEMMA 5.6. *Let T_2 be the matrix produced by two steps of* Algorithm 5.9 *using $\tau_i^2 = 0$, and let T' be the matrix produced by one step of QR iteration ($QR = T_0$, $T' = RQ$). Then $T_2 = T'$.*

Proof. Since T_0 is symmetric, we can factorize T_0^2 in two ways: First, $T_0^2 = T_0^T T_0 = (QR)^T QR = R^T R$. We assume without loss of generality that $R_{ii} > 0$. This is a factorization of T_0^2 into a lower triangular matrix R^T times its transpose; since the Cholesky factorization is unique, this must in fact be the Cholesky factorization. The second factorization is $T_0^2 = B_0^T B_0 B_0^T B_0$. Now by Algorithm 5.9, $T_1 = B_0 B_0^T = B_1^T B_1$, so we can rewrite $T_0^2 = B_0^T B_0 B_0^T B_0 = B_0^T (B_1^T B_1) B_0 = (B_1 B_0)^T B_1 B_0$. This is also a factorization of T_0^2 into a lower triangular matrix $(B_1 B_0)^T$ times its transpose, so this must again be the Cholesky factorization. By uniqueness of the Cholesky factorization, we conclude $R = B_1 B_0$, thus relating two steps of LR iteration to one step of QR iteration. We exploit this relationship as follows: $T_0 = QR$ implies

$$T' = RQ = RQ(RR^{-1}) = R(QR)R^{-1} = RT_0R^{-1} \quad \text{because } T_0 = QR$$

[21] dqds is short for "differential quotient-difference algorithm with shifts" [207].

$$
\begin{aligned}
&= (B_1B_0)(B_0^TB_0)(B_1B_0)^{-1} \quad \text{because } R = B_1B_0 \text{ and } T_0 = B_0^TB_0 \\
&= B_1B_0B_0^TB_0B_0^{-1}B_1^{-1} = B_1(B_0B_0^T)B_1^{-1} \\
&= B_1(B_1^TB_1)B_1^{-1} \quad \text{because } B_0B_0^T = T_1 = B_1^TB_1 \\
&= B_1B_1^T \\
&= T_2 \text{ as desired.} \quad \square
\end{aligned}
$$

Neither Algorithm 5.9 nor Lemma 5.6 depends on T_0 being tridiagonal, just symmetric positive definite. Using the relationship between LR iteration and QR iteration in Lemma 5.6, one can show that much of the convergence analysis of QR iteration goes over to LR iteration; we will not explore this here.

Our ultimate algorithm, dqds, is mathematically equivalent to LR iteration. But it is *not* implemented as described in Algorithm 5.9, because this would involve explicitly forming $T_{i+1} = B_iB_i^T + \tau_i^2 I$, which in section 5.4 we showed could be numerically unstable. Instead, we will form B_{i+1} *directly* from B_i, without ever forming the intermediate matrix T_{i+1}.

To simplify notation, let B_i have diagonal $a_1, \ldots, a_n$ and superdiagonal $b_1, \ldots, b_{n-1}$, and B_{i+1} have diagonal $\hat{a}_1, \ldots, \hat{a}_n$ and superdiagonal $\hat{b}_1, \ldots, \hat{b}_{n-1}$. We use the convention $b_0 = \hat{b}_0 = b_n = \hat{b}_n = 0$. We relate B_i to B_{i+1} by

$$B_{i+1}^TB_{i+1} + \tau_{i+1}^2 I = T_{i+1} = B_iB_i^T + \tau_i^2 I. \tag{5.20}$$

Equating the j, j entries of the left and right sides of equation (5.20) for $j < n$ yields

$$\hat{a}_j^2 + \hat{b}_{j-1}^2 + \tau_{i+1}^2 = a_j^2 + b_j^2 + \tau_i^2 \quad \text{or} \quad \hat{a}_j^2 = a_j^2 + b_j^2 - \hat{b}_{j-1}^2 - \delta, \tag{5.21}$$

where $\delta = \tau_{i+1}^2 - \tau_i^2$. Since τ_i^2 must be chosen to approach the smallest eigenvalue of T from below (to keep T_i positive definite and the algorithm well defined), $\delta \geq 0$. Equating the squares of the $j, j+1$ entries of the left and right sides of equation (5.20) yields

$$\hat{a}_j^2\hat{b}_j^2 = a_{j+1}^2b_j^2 \quad \text{or} \quad \hat{b}_j^2 = a_{j+1}^2b_j^2/\hat{a}_j^2. \tag{5.22}$$

Combining equations (5.21) and (5.22) yields the not-yet-final algorithm

```
for j = 1 to n − 1
    â_j^2 = a_j^2 + b_j^2 − b̂_{j−1}^2 − δ
    b̂_j^2 = b_j^2 · (a_{j+1}^2/â_j^2)
end for
â_n^2 = a_n^2 − b̂_{n−1}^2 − δ
```

This version of the algorithm has only five floating point operations in the inner loop, which is quite inexpensive. It maps directly from the *squares* of the entries of B_i to the *squares* of the entries of B_{i+1}. There is no reason to take

square roots until the very end of the algorithm. Indeed, square roots, along with divisions, can take 10 to 30 times longer than additions, subtractions, or multiplications on modern computers, so we should avoid as many of them as possible. To emphasize that we are computing squares of entries, we change variables to $q_j \equiv a_j^2$ and $e_j \equiv b_j^2$, yielding the penultimate algorithm qds (again, the name is for historical reasons that do not concern us [207]).

ALGORITHM 5.10. *One step of the qds algorithm:*

for $j = 1$ *to* $n-1$
  $\hat{q}_j = q_j + e_j - \hat{e}_{j-1} - \delta$
  $\hat{e}_j = e_j \cdot (q_{j+1}/\hat{q}_j)$
end for
$\hat{q}_n = q_n - \hat{e}_{n-1} - \delta$

The final algorithm, dqds, will do about the same amount of work as qds but will be significantly more accurate, as will be shown in section 5.4.2. We take the subexpression $q_j - \hat{e}_{j-1} - \delta$ from the first line of Algorithm 5.10 and rewrite it as follows:

$$\begin{aligned} d_j &\equiv q_j - \hat{e}_{j-1} - \delta \\ &= q_j - \frac{q_j e_{j-1}}{\hat{q}_{j-1}} - \delta \quad \text{from} \quad (5.22) \\ &= q_j \cdot \left[\frac{\hat{q}_{j-1} - e_{j-1}}{\hat{q}_{j-1}} \right] - \delta \\ &= q_j \cdot \left[\frac{q_{j-1} - \hat{e}_{j-2} - \delta}{\hat{q}_{j-1}} \right] - \delta \quad \text{from} \quad (5.21) \\ &= \frac{q_j}{\hat{q}_{j-1}} \cdot d_{j-1} - \delta. \end{aligned}$$

This lets us rewrite the inner loop of Algorithm 5.10 as

$\hat{q}_j = d_j + e_j$
$\hat{e}_j = e_j \cdot (q_{j+1}/\hat{q}_j)$
$d_{j+1} = d_j \cdot (q_{j+1}/\hat{q}_j) - \delta$

Finally, we note that d_{j+1} can overwrite d_j and that $t = q_{j+1}/\hat{q}_j$ need be computed only once to get the final dqds algorithm.

ALGORITHM 5.11. *One step of the dqds algorithm:*

$d = q_1 - \delta$
for $j = 1$ *to* $n-1$
  $\hat{q}_j = d + e_j$
  $t = (q_{j+1}/\hat{q}_j)$
  $\hat{e}_j = e_j \cdot t$

$d = d \cdot t - \delta$
end for
$\hat{q}_n = d$

The dqds algorithm has the same number of floating point operations in its inner loop as qds but trades a subtraction for a multiplication. This modification pays off handsomely in guaranteed high relative accuracy, as described in the next section.

There are two important issues we have not discussed: choosing a shift $\delta = \tau_{i+1}^2 - \tau_i^2$ and detecting convergence. These are discussed in detail in [102].

5.4.2. Computing the Bidiagonal SVD to High Relative Accuracy

This section, which depends on section 5.2.1, may be skipped on a first reading.

Our ability to compute the SVD of a bidiagonal matrix B to high relative accuracy (as defined in section 5.2.1) depends on Theorem 5.13 below, which says that small relative changes in the entries of B cause only small relative changes in the singular values.

LEMMA 5.7. *Let B be a bidiagonal matrix, with diagonal entries $a_1, \ldots, a_n$ and superdiagonal entries $b_1, \ldots, b_{n-1}$. Let $\hat{B}$ be another bidiagonal matrix with diagonal entries $\hat{a}_i = a_i\chi_i$ and superdiagonal entries $\hat{b}_i = b_i\zeta_i$. Then $\hat{B} = D_1 B D_2$, where*

$$D_1 = \mathrm{diag}\left(\chi_1, \frac{\chi_2\chi_1}{\zeta_1}, \frac{\chi_3\chi_2\chi_1}{\zeta_2\zeta_1}, \ldots, \frac{\chi_n \cdots \chi_1}{\zeta_{n-1}\cdots\zeta_1}\right),$$

$$D_2 = \mathrm{diag}\left(1, \frac{\zeta_1}{\chi_1}, \frac{\zeta_2\zeta_1}{\chi_2\chi_1} \ldots, \frac{\zeta_{n-1}\cdots\zeta_1}{\chi_{n-1}\cdots\chi_1}\right).$$

The proof of this lemma is a simple computation (see Question 5.20). We can now apply Corollary 5.2 to conclude the following.

THEOREM 5.13. *Let B and $\hat{B}$ be defined as in Lemma 5.7. Suppose that there is a $\tau \geq 1$ such that $\tau^{-1} \leq \chi_i \leq \tau$ and $\tau^{-1} \leq \zeta_i \leq \tau$. In other words $\epsilon \equiv \tau - 1$ is a bound on the relative difference between each entry of B and the corresponding entry of $\hat{B}$. Let $\sigma_n \leq \cdots \leq \sigma_1$ be the singular values of B and $\hat{\sigma}_n \leq \cdots \leq \hat{\sigma}_1$ be the singular values of $\hat{B}$. Then $|\hat{\sigma}_i - \sigma_i| \leq \sigma_i(\tau^{4n-2} - 1)$. If $\sigma_i = 0$ and $\tau - 1 = \epsilon \ll 1$, then we can write*

$$\frac{|\hat{\sigma}_i - \sigma_i|}{\sigma_i} \leq \tau^{4n-2} - 1 = (4n-2)\epsilon + O(\epsilon^2).$$

Thus, the relative change in the singular values $|\hat{\sigma}_i - \sigma_i|/\sigma_i$ is bounded by $4n-2$ times the relative change ϵ in the matrix entries. With a little more work,

the factor $4n-2$ can be improved to $2n-1$ (see Question 5.21). The singular vectors can also be shown to be determined quite accurately, proportional to the reciprocal of the relative gap, as defined in section 5.2.1.

We will show that both Bisection (Algorithm 5.4 applied to T_{ps} from Lemma 5.5) and dqds (Algorithm 5.11) can be used to find the singular values of a bidiagonal matrix to high relative accuracy. First we consider Bisection. Recall that the eigenvalues of the symmetric tridiagonal matrix T_{ps} are the singular values of B and their negatives. Lemma 5.3 implies that the inertia of $T_{ps} - \lambda I$ computed using equation (5.17) is the exact inertia of some $\hat{B}$, where the relative difference of corresponding entries of $\hat{B}$ and B is at most about 2.5ε. Therefore, by Theorem 5.13, the relative difference between the computed singular values (the singular values of $\hat{B}$) and the true singular values is at most about $(10n-5)\varepsilon$.

Now we consider Algorithm 5.11. We will use Theorem 5.13 to prove that the singular values of B (the input to Algorithm 5.11) and the singular values of $\hat{B}$ (the output from Algorithm 5.11) agree to high relative accuracy. This fact implies that after many steps of dqds, when $\hat{B}$ is nearly diagonal with its singular values on the diagonal, these singular values match the singular values of the original input matrix to high relative accuracy.

The simplest situation to understand is when the shift $\delta = 0$. In this case, the only operations in dqds are additions of positive numbers, multiplications, and divisions; no cancellation occurs. Roughly speaking, any sequence of expressions built of these basic operations is guaranteed to compute each output to high relative accuracy. Therefore, $\hat{B}$ is computed to high relative accuracy, and so by Theorem 5.13, the singular values of B and $\hat{B}$ agree to high relative accuracy. The general case, where $\delta > 0$, is trickier [102].

THEOREM 5.14. *One step of Algorithm* 5.11 *in floating point arithmetic, applied to B and yielding $\hat{B}$, is equivalent to the following sequence of operations:*

1. *Make a small relative change (by at most* 1.5ε*) in each entry of B, getting $\tilde{B}$.*
2. *Apply one step of Algorithm* 5.11 *in exact arithmetic to $\tilde{B}$, getting $\check{B}$.*
3. *Make a small relative change (by at most ε) in each entry of $\check{B}$, getting $\hat{B}$.*

Steps 1 *and* 3 *above make only small relative changes in the singular values of the bidiagonal matrix, so by Theorem* 5.13 *the singular values of B and $\hat{B}$ agree to high relative accuracy.*

Proof. Let us write the inner loop of Algorithm 5.11 as follows, introducing subscripts on the d and t variables to let us keep track of them in different iterations and including subscripted $1+\epsilon$ terms for the roundoff errors:

$$\begin{aligned}
\hat{q}_j &= (d_j + e_j)(1 + \epsilon_{j,+}) \\
t_j &= (q_{j+1}/\hat{q}_j)(1 + \epsilon_{j,/}) \\
\hat{e}_j &= e_j \cdot t_j(1 + \epsilon_{j,*1}) \\
d_{j+1} &= (d_j \cdot t_j(1 + \epsilon_{j,*2}) - \delta)(1 + \epsilon_{j,-})
\end{aligned}$$

Substituting the first line into the second line yields

$$t_j = \frac{q_{j+1}}{d_j + e_j} \cdot \frac{1 + \epsilon_{j,/}}{1 + \epsilon_{j,+}}.$$

Substituting this expression for t_j into the last line of the algorithm and dividing through by $1 + \epsilon_{j,-}$ yield

$$\frac{d_{j+1}}{1 + \epsilon_{j,-}} = \frac{d_j q_{j+1}}{d_j + e_j} \cdot \frac{(1 + \epsilon_{j,/})(1 + \epsilon_{j,*2})}{1 + \epsilon_{j,+}} - \delta. \tag{5.23}$$

This tells us how to define $\tilde{B}$: Let

$$\begin{aligned}
\tilde{d}_{j+1} &= \frac{d_{j+1}}{1 + \epsilon_{j,-}}, \\
\tilde{e}_j &= \frac{e_j}{1 + \epsilon_{j-1,-}}, \\
\tilde{q}_{j+1} &= q_{j+1} \frac{(1 + \epsilon_{j,/})(1 + \epsilon_{j,*2})}{1 + \epsilon_{j,+}},
\end{aligned} \tag{5.24}$$

so (5.23) becomes

$$\tilde{d}_{j+1} = \frac{\tilde{d}_j \tilde{q}_{j+1}}{\tilde{d}_j + \tilde{e}_j} - \delta.$$

Note from (5.24) that $\tilde{B}$ differs from B by a relative change of at most 1.5ε in each entry (from the three $1 + \epsilon$ factors in $\tilde{q}_{j+1} = \tilde{b}^2_{j+1,j+1}$).

Now we can define $\check{q}_j$ and $\check{e}_j$ in $\check{B}$ by

$$\begin{aligned}
\check{q}_j &= \tilde{d}_j + \tilde{e}_j, \\
\tilde{t}_j &= (\tilde{q}_{j+1}/\check{q}_j), \\
\check{e}_j &= \tilde{e}_j \cdot \tilde{t}_j, \\
\tilde{d}_{j+1} &= \tilde{d}_j \cdot \tilde{t}_j - \delta.
\end{aligned}$$

This is one step of the dqds algorithm applied exactly to $\tilde{B}$, getting $\check{B}$. To finally show that $\check{B}$ differs from $\hat{B}$ by a relative change of at most ε in each entry, note that

$$\begin{aligned}
\check{q}_j &= \tilde{d}_j + \tilde{e}_j \\
&= \frac{d_j}{1 + \epsilon_{j-1,-}} + \frac{e_j}{1 + \epsilon_{j-1,-}} \\
&= (d_j + e_j)(1 + \epsilon_{j,+}) \cdot \frac{1}{(1 + \epsilon_{j,+})(1 + \epsilon_{j-1,-})} \\
&= \hat{q}_j \cdot \left[\frac{1}{(1 + \epsilon_{j,+})(1 + \epsilon_{j-1,-})} \right]
\end{aligned}$$

and

$$\begin{aligned}
\check{e}_j &= \tilde{e}_j \cdot \tilde{t}_j \\
&= \frac{e_j}{1+\epsilon_{j-1,-}} \cdot \frac{\tilde{q}_{j+1}}{\check{q}_j} \\
&= \frac{e_j}{1+\epsilon_{j-1,-}} \cdot t_j(1+\epsilon_{j,*2})(1+\epsilon_{j-1,-}) \\
&= e_j t_j (1+\epsilon_{j,*1}) \frac{1+\epsilon_{j,*2}}{1+\epsilon_{j,*1}} \\
&= \hat{e}_j \cdot \left[\frac{1+\epsilon_{j,*2}}{1+\epsilon_{j,*1}}\right]. \quad \square
\end{aligned}$$

5.4.3. Jacobi's Method for the SVD

In section 5.3.5 we discussed Jacobi's method for finding the eigenvalues and eigenvectors of a dense symmetric matrix A, and said it was the slowest available method for this problem. In this section we will show how to apply Jacobi's method to find the SVD of a dense matrix G by *implicitly* applying Algorithm 5.8 of section 5.3.5 to the symmetric matrix $A = G^T G$. This implies that the convergence properties of this method are nearly the same as those of Algorithm 5.8, and in particular Jacobi's method is also the slowest method available for the SVD.

Jacobi's method is still interesting, however, because for some kinds of matrices G, it can compute the singular values and singular vectors much more accurately than the other algorithms we have discussed. For these G, Jacobi's method computes the singular values and singular vectors to high relative accuracy, as described in section 5.2.1.

After describing the implicit Jacobi method for the SVD of G, we will show that it computes the SVD to high relative accuracy when G can be written in the form $G = DX$, where D is diagonal and X is well conditioned. (This means that G is ill conditioned if and only if D has both large and small diagonal entries.) More generally, we benefit as long as X is significantly better conditioned than G. We will illustrate this with a matrix where any algorithm involving reduction to bidiagonal form necessarily loses all significant digits in all but the largest singular value, whereas Jacobi computes all singular values to full machine precision. Then we survey other classes of matrices G for which Jacobi's method is also significantly more accurate than methods using bidiagonalization.

Note that if G is bidiagonal, then we showed in section 5.4.2 that we could use either Bisection or the dqds algorithm (section 5.4.1) to compute its SVD to high relative accuracy. The trouble is that reducing a matrix from dense to bidiagonal form can introduce errors that are large enough to destroy high relative accuracy, as our example will show. Since Jacobi's method operates on

the original matrix without first reducing it to bidiagonal form, it can achieve high relative accuracy in many more situations.

The implicit Jacobi method is mathematically equivalent to applying Algorithm 5.8 to $A = G^T G$. In other words, at each step we compute a Jacobi rotation J and implicitly update $G^T G$ to $J^T G^T G J$, where J is chosen so that two offdiagonal entries of $G^T G$ are set to zero in $J^T G^T G J$. But instead of computing $G^T G$ or $J^T G^T G J$ explicitly, we instead only compute GJ. For this reason, we call our algorithm *one-sided Jacobi rotation.*

ALGORITHM 5.12. *Compute and apply a one-sided Jacobi rotation to G in coordinates j, k:*

proc One-Sided-Jacobi-Rotation (G, j, k)
    Compute $a_{jj} = (G^T G)_{jj}$, $a_{jk} = (G^T G)_{jk}$, *and* $a_{kk} = (G^T G)_{kk}$
    if $|a_{jk}|$ *is not too small*
        $\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$
        $t = \mathrm{sign}(\tau)/(|\tau| + \sqrt{1 + \tau^2})$
        $c = 1/\sqrt{1 + t^2}$
        $s = c \cdot t$
        $G = G \cdot R(j, k, \theta)$ ... *where* $c = \cos\theta$ *and* $s = \sin\theta$
        if right singular vectors are desired
            $J = J \cdot R(j, k, \theta)$
        end if
    end if

Note that the jj, jk, and kk entries of $A = G^T G$ are computed by procedure One-Sided-Jacobi-Rotation, after which it computes the Jacobi rotation $R(j, k, \theta)$ in the same way as procedure Jacobi-Rotation (Algorithm 5.5).

ALGORITHM 5.13. *One-sided Jacobi: Assume that G is n-by-n. The outputs are the singular values σ_i, the left singular vector matrix U, and the right singular vector matrix V so that $G = U\Sigma V^T$, where $\Sigma = \mathrm{diag}(\sigma_i)$.*

repeat
    for $j = 1$ *to* $n - 1$
        for $k = j + 1$ *to* n
            call One-Sided-Jacobi-Rotation(G, j, k)
        end for
    end for
until $G^T G$ *is diagonal enough*
Let $\sigma_i = \|G(:, i)\|_2$ *(the 2-norm of column i of G)*
Let $U = [u_1, \ldots, u_n]$, *where* $u_i = G(:, i)/\sigma_i$
let $V = J$, *the accumulated product of Jacobi rotations*

Question 5.22 asks for a proof that the matrices Σ, U, and V computed by one-sided Jacobi do indeed form the SVD of G.

The following theorem shows that one-sided Jacobi can compute the SVD to high relative accuracy, despite roundoff, provided that we can write $G = DX$, where D is diagonal and X is well conditioned.

THEOREM 5.15. *Let $G = DX$ be an n-by-n matrix, where D is diagonal and nonsingular, and X is nonsingular. Let $\hat{G}$ be the matrix after calling One-Sided-Jacobi-Rotation(G, j, k) m times in floating point arithmetic. Let $\sigma_1 \geq \cdots \geq \sigma_n$ be the singular values of G, and let $\hat{\sigma}_1 \geq \cdots \geq \hat{\sigma}_n$ be the singular values of $\hat{G}$. Then*

$$\frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i} \leq O(m\varepsilon)\kappa(X), \tag{5.25}$$

where $\kappa(X) = \|X\| \cdot \|X^{-1}\|$ is the condition number of X. In other words, the relative error in the singular values is small if the condition number of X is small.

Proof. We first consider $m = 1$; i.e., we apply only a single Jacobi rotation and later generalize to larger m.

Examining One-Sided-Jacobi-Rotation(G, j, k), we see that $\hat{G} = \mathrm{fl}(G \cdot \tilde{R})$, where $\tilde{R}$ is a floating point Givens rotation. By construction, $\tilde{R}$ differs from some exact Givens rotation R by $O(\varepsilon)$ in norm. (It is not important or necessarily true that $\tilde{R}$ differs by $O(\varepsilon)$ from the "true" Jacobi rotation, the one that One-Sided-Jacobi-Rotation(G, j, k) would have computed in exact arithmetic. It is necessary only that that it differs from *some* rotation by $O(\varepsilon)$. This requires only that $c^2 + s^2 = 1 + O(\varepsilon)$, which is easy to verify.)

Our goal is to show that $\hat{G} = GR(I + E)$ for some E that is small in norm: $\|E\|_2 = O(\varepsilon)\kappa(X)$. If E were zero, then $\hat{G}$ and GR would have the same singular values, since R is exactly orthogonal. When E is less than one in norm, we can use Corollary 5.2 to bound the relative difference in singular values by

$$\begin{aligned} \frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i} &\leq \|(I+E)(I+E)^T - I\|_2 = \|E + E^T + EE^T\|_2 \leq 3\|E\|_2 \\ &= O(\varepsilon)\kappa(X) \end{aligned} \tag{5.26}$$

as desired.

Now we construct E. Since $\tilde{R}$ multiplies G on the right, each row of $\hat{G}$ depends only on the corresponding row of G; write this in Matlab notation as $\hat{G}(i,:) = \mathrm{fl}(G(i,:) \cdot \tilde{R})$. Let $F = \hat{G} - GR$. Then by Lemma 3.1 and the fact that $G = DX$,

$$\|F(i,:)\|_2 = \|\hat{G}(i,:) - G(i,:)R\|_2 = O(\varepsilon)\|G(i,:)\|_2 = O(\varepsilon)\|d_{ii}X(i,:)\|_2$$

and so $\|d_{ii}^{-1}F(i,:)\|_2 = O(\varepsilon)\|X(i,:)\|_2$, or $\|D^{-1}F\|_2 = O(\varepsilon)\|X\|_2$. Therefore, since $R^{-1} = R^T$ and $G^{-1} = (DX)^{-1} = X^{-1}D^{-1}$,

$$\hat{G} = GR + F = GR(I + R^T G^{-1} F) = GR(I + R^T X^{-1} D^{-1} F) \equiv GR(I + E)$$

where

$$\|E\|_2 \leq \|R^T\|_2 \|X^{-1}\|_2 \|D^{-1}F\|_2 = O(\varepsilon)\|X\|_2\|X^{-1}\|_2 = O(\varepsilon)\kappa(X)$$

as desired.

To extend this result to $m > 1$ rotations, note that in exact arithmetic we would have $\hat{G} = GR = DXR = D\hat{X}$, with $\kappa(\hat{X}) = \kappa(X)$, so that the bound (5.26) would apply at each of the m steps, yielding bound (5.25). Because of roundoff, $\kappa(\hat{X})$ could grow by as much as $\kappa(I+E) \leq (1+O(\varepsilon)\kappa(X))$ at each step, a factor very close to 1, which we absorb into the $O(m\varepsilon)$ term. □

To complete the algorithm, we need to be careful about the stopping criterion, i.e. how to implement the statement "if $|a_{jk}|$ is not too small" in Algorithm 5.12, One-Sided-Jacobi-Rotation. The appropriate criterion

$$|a_{jk}| \geq \varepsilon\sqrt{a_{jj}a_{kk}}$$

is discussed further in Question 5.24.

EXAMPLE 5.9. We consider an extreme example $G = DX$ where Jacobi's method computes all singular values to full machine precision; any method relying on bidiagonalization computes only the largest one, $\sqrt{3}$, to full machine precision; and all the others with no accuracy at all (although it still computes them with errors $\pm O(\varepsilon) \cdot \sqrt{3}$, as expected from a backward stable algorithm). In this example $\varepsilon = 2^{-53} \approx 10^{-16}$ (IEEE double precision) and $\eta = 10^{-20}$ (any value of $\eta < \varepsilon$ will do). We define

$$G \equiv \begin{bmatrix} \eta & 1 & 1 & 1 \\ \eta & \eta & 0 & 0 \\ \eta & 0 & \eta & 0 \\ \eta & 0 & 0 & \eta \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \eta & & \\ & & \eta & \\ & & & \eta \end{bmatrix} \cdot \begin{bmatrix} \eta & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \equiv D \cdot X.$$

To at least 16 digits, the singular values of G are $\sqrt{3}$, $\sqrt{3} \cdot \eta$, η, and η. To see how accuracy is lost by reducing G to bidiagonal form, we consider just the first step of the algorithm section 4.4.7: After step 1, premultiplication by a Householder transformation to zero out $G(2:4,1)$, G in exact arithmetic would be

$$\begin{bmatrix} -2\eta & -.5-\frac{\eta}{2} & -.5-\frac{\eta}{2} & -.5-\frac{\eta}{2} \\ 0 & -.5+\frac{5\eta}{6} & -.5-\frac{\eta}{6} & -.5-\frac{\eta}{6} \\ 0 & -.5-\frac{\eta}{6} & -.5+\frac{5\eta}{6} & -.5-\frac{\eta}{6} \\ 0 & -.5-\frac{\eta}{6} & -.5-\frac{\eta}{6} & -.5+\frac{5\eta}{6} \end{bmatrix},$$

but since η is so small, this rounds to

$$G_1 = \begin{bmatrix} -2\eta & -.5 & -.5 & -.5 \\ 0 & -.5 & -.5 & -.5 \\ 0 & -.5 & -.5 & -.5 \\ 0 & -.5 & -.5 & -.5 \end{bmatrix}.$$

Note that all information about η has been "lost" from the last three columns of G_1. Since the last three columns of G_1 are identical, G_1 is exactly singular and indeed of rank 2. Thus the two smallest singular values have been changed from η to 0, a complete loss of relative accuracy. If we made no further rounding errors, we would reduce G_1 to the bidiagonal form

$$B = \begin{bmatrix} -2\eta & \sqrt{.75} & & \\ & 1.5 & 0 & \\ & & 0 & 0 \\ & & & 0 \end{bmatrix}$$

with singular values $\sqrt{3}$, $\sqrt{3}\eta$, 0, and 0, the larger two of which are accurate singular values of G. But as the algorithm proceeds to reduce G_1 to bidiagonal form, roundoff introduces nonzero quantities of $O(\varepsilon)$ into the zero entries of B, making all three small singular values inaccurate. The two smallest nonzero computed singular values are accidents of roundoff and proportional to ε.

One-sided Jacobi's method has no difficult with this matrix, converging in three sweeps to $G = U\Sigma V^T$, where to machine precision

$$U = \begin{bmatrix} 0 & -\frac{\eta^2}{\sqrt{2}} & 1 & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{\eta}{3} & \frac{-1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{\eta}{3} & \frac{-1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{\eta}{\sqrt{2}} & \frac{\eta}{3} & \frac{2}{\sqrt{6}} \end{bmatrix}, \quad V = \begin{bmatrix} 1 & \frac{\eta}{\sqrt{2}} & \frac{\eta}{\sqrt{3}} & \frac{-\eta}{\sqrt{6}} \\ -\eta & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{6}} \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{6}} \\ 0 & \frac{-\eta^3}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} \end{bmatrix},$$

and $\Sigma = \mathrm{diag}(\sqrt{3}\eta, \eta, \sqrt{3}, \eta)$. (Jacobi does not automatically sort the singular values; this can be done as a postprocessing step.) $\diamond$

Here are some other examples where versions of Jacobi's method can be shown to guarantee high relative accuracy in the SVD (or symmetric eigendecomposition), whereas methods relying on bidiagonalization (or tridiagonalization) may lose all significant digits in the smallest singular value (or eigenvalues). Many other examples appear in [74].

1. If $A = LL^T$ is the Cholesky decomposition of a symmetric positive definite matrix, then the SVD of $L = U\Sigma V^T$ provides the eigendecomposition of $A = U\Sigma^2 U^T$. If $L = DX$, where X is well-conditioned and D is diagonal, then Theorem 5.15 tells us that we can use Jacobi's method to compute the singular values σ_i of L to high relative accuracy, with relative errors bounded by $O(\varepsilon)\kappa(X)$. But we also have to account for the roundoff errors in computing the Cholesky factor L: using Cholesky's backward error bound (2.16) (along with Theorem 5.6) one can bound the relative error in the singular values introduced by roundoff during Cholesky by $O(\varepsilon)\kappa^2(X)$. So if X is well-conditioned, all the eigenvalues of A will be computed to high relative accuracy (see Question 5.23 and [81, 90, 181]).

EXAMPLE 5.10. As in Example 5.9, we choose an extreme case where any algorithm relying on initially reducing A to tridiagonal form is guaranteed to lose all relative accuracy in the smallest eigenvalue, whereas Cholesky followed by one-sided Jacobi's method on the Cholesky factor computes all eigenvalues to nearly full machine precision. As in that example, let $\eta = 10^{-20}$ (any $\eta < \varepsilon/120$ will do), and let

$$A = \begin{bmatrix} 1 & \sqrt{\eta} & \sqrt{\eta} \\ \sqrt{\eta} & 1 & 10\eta \\ \sqrt{\eta} & 10\eta & 100\eta \end{bmatrix} = \begin{bmatrix} 1 & 10^{-10} & 10^{-10} \\ 10^{-10} & 1 & 10^{-19} \\ 10^{-10} & 10^{-19} & 10^{-20} \end{bmatrix}.$$

If we reduce A to tridiagonal form T exactly, then

$$T = \begin{bmatrix} 1 & \sqrt{2\eta} & \\ \sqrt{2\eta} & .5 + 60\eta & .5 - 50\eta \\ & .5 - 50\eta & .5 + 40\eta \end{bmatrix},$$

but since η is so small, this rounds to

$$\hat{T} = \begin{bmatrix} 1 & \sqrt{2\eta} & \\ \sqrt{2\eta} & .5 & .5 \\ & .5 & .5 \end{bmatrix},$$

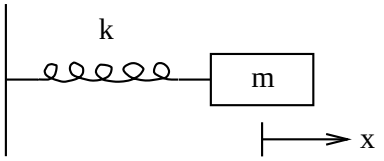
which is not even positive definite, since the bottom right 2-by-2 submatrix is exactly singular. Thus, the smallest eigenvalues of $\hat{T}$ is nonpositive, and so tridiagonal reduction has lost all relative accuracy in the smallest eigenvalue. In contrast, one-sided Jacobi's method has no trouble computing the correct square roots of eigenvalues of A, namely, $1 + \sqrt{\eta} = 1 + 10^{-10}$, $1 - \sqrt{\eta} = 1 - 10^{-10}$, and $.99\eta = .99 \cdot 10^{-20}$, to nearly full machine precision. $\diamond$

2. For extensions of the preceding result to indefinite symmetric eigenproblems, see [226, 248].

3. For extensions to the generalized symmetric eigenproblem $A - \lambda B$ and the generalized SVD, see [65, 90].

5.5. Differential Equations and Eigenvalue Problems

We seek our motivation for this section from conservation laws in physics. We consider once again the mass-spring system introduced in Example 4.1 and reexamined in Example 5.1. We start with the simplest case of one spring and one mass, without friction:



We let x denote horizontal displacement from equilibrium. Then Newton's law $F = ma$ becomes $m\ddot{x}(t) + kx(t) = 0$. Let $E(t) = \frac{1}{2}m\dot{x}^2(t) + \frac{1}{2}kx^2(t) =$ "kinetic energy" + "potential energy." Conservation of energy tells us that $\frac{d}{dt}E(t)$ should be zero. We can confirm this is true by computing $\frac{d}{dt}E(t) = m\dot{x}(t)\ddot{x}(t) + kx(t)\dot{x}(t) = \dot{x}(t)(m\ddot{x}(t) + kx(t)) = 0$ as desired.

More generally we have $M\ddot{x}(t)+Kx(t) = 0$, where M is the *mass matrix* and K is the *stiffness matrix*. The energy is defined to be $E(t) = \frac{1}{2}\dot{x}^T(t)M\dot{x}(t) + \frac{1}{2}x^T(t)Kx(t)$. That this is the correct definition is confirmed by verifying that it is conserved:

$$\begin{aligned}
\frac{d}{dt}E(t) &= \frac{d}{dt}\left(\frac{1}{2}\dot{x}^T(t)M\dot{x}(t) + \frac{1}{2}x^T(t)Kx(t)\right) \\
&= \frac{1}{2}(\ddot{x}^T(t)M\dot{x}(t) + \dot{x}^T(t)M\ddot{x}(t) + \dot{x}^T(t)Kx(t) + x^T(t)K\dot{x}(t)) \\
&= \dot{x}^T(t)M\ddot{x}(t) + \dot{x}^T(t)Kx(t) \\
&= \dot{x}^T(t)(M\ddot{x}(t) + Kx(t)) = 0,
\end{aligned}$$

where we have used the symmetry of M and K.

The differential equations $M\ddot{x}(t) + Kx(t) = 0$ are linear. It is a remarkable fact that some nonlinear differential equations also conserve quantities such as "energy."

5.5.1. The Toda Lattice

For ease of notation, we will write $\dot{x}$ instead of $\dot{x}(t)$ when the argument is clear from context.

The *Toda lattice* is also a mass-spring system, but the force from the spring is an exponentially decaying function of its stretch, instead of a linear function:

$$\ddot{x}_i = e^{-(x_i - x_{i-1})} - e^{-(x_{i+1} - x_i)}.$$

We use the boundary conditions $e^{-(x_1 - x_0)} = 0$ (i.e., $x_0 = -\infty$) and $e^{-(x_{n+1} - x_n)} = 0$ (i.e., $x_{n+1} = +\infty$). More simply, these boundary conditions mean there are no walls at the left or right (see Figure 4.1).

Now we change variables to $b_k = \frac{1}{2}e^{(x_k - x_{k+1})/2}$ and $a_k = -\frac{1}{2}\dot{x}_k$. This yields the differential equations

$$\begin{aligned}
\dot{b}_k &= \frac{1}{2}e^{(x_k - x_{k+1})/2} \cdot \frac{1}{2}(\dot{x}_k - \dot{x}_{k+1}) = b_k(a_{k+1} - a_k), \\
\dot{a}_k &= -\frac{1}{2}\ddot{x}_k = 2(b_k^2 - b_{k-1}^2)
\end{aligned} \tag{5.27}$$

with $b_0 \equiv 0$ and $b_n \equiv 0$. Now define the two tridiagonal matrices

$$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & b_1 & & \\ -b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & -b_{n-1} & 0 \end{bmatrix},$$

where $B = -B^T$. Then one can easily confirm that equation (5.27) is the same as $\frac{dT}{dt} = BT - TB$. This is called the *Toda flow*.

THEOREM 5.16. *$T(t)$ has the same eigenvalues as $T(0)$ for all t. In other words, the eigenvalues, such as "energy," are conserved by the differential equation.*

Proof. Define $\frac{d}{dt}U = BU$, $U(0) = I$. We claim that $U(t)$ is orthogonal for all t. To prove this, it suffices to show $\frac{d}{dt}U^TU = 0$ since $U^TU(0) = I$:

$$\frac{d}{dt}U^TU = \dot{U}^TU + U^T\dot{U} = U^TB^TU + U^TBU = -U^TBU + U^TBU = 0$$

since B is skew symmetric.

Now we claim that $T(t) = U(t)T(0)U^T(t)$ satisfies the Toda flow $\frac{dT}{dt} = BT - TB$, implying each $T(t)$ is orthogonally similar to $T(0)$ and so has the same eigenvalues:

$$\begin{aligned} \frac{d}{dt}T(t) &= \dot{U}(t)T(0)U^T(t) + U(t)T(0)\dot{U}^T(t) \\ &= B(t)U(t)T(0)U^T(t) + U(t)T(0)U^T(t)B^T(t) \\ &= B(t)T(t) - T(t)B(t) \end{aligned}$$

as desired. □

Note that the only property of B used was skew symmetry, so if $\frac{d}{dt}T = BT - TB$ and $B^T = -B$, then $T(t)$ has the same eigenvalues for all t.

THEOREM 5.17. *As $t \to +\infty$ or $t \to -\infty$, $T(t)$ converges to a diagonal matrix with the eigenvalues on the diagonal.*

Proof. We want to show $b_i(t) \to 0$ as $t \to \pm\infty$. We begin by showing $\int_{-\infty}^{\infty} \sum_{i=1}^{n-1} b_i^2(t)dt < \infty$. We use induction to show $\int_{-\infty}^{\infty}(b_j^2(t)+b_{n-j}^2(t))dt < \infty$ and then add these inequalities for all j. When $j = 0$, we get $\int_{-\infty}^{\infty}(b_0^2(t) + b_n^2(t))dt$, which is 0 by assumption.

Now let $\varphi(t) = a_j(t) - a_{n-j+1}(t)$. $\varphi(t)$ is bounded by $2\|T(t)\|_2 = 2\|T(0)\|_2$ for all t. Then

$$\begin{aligned} \dot{\varphi}(t) &= \dot{a}_j(t) - \dot{a}_{n-j+1}(t) \\ &= 2(b_j^2(t) - b_{j-1}^2(t)) - 2(b_{n-j+1}^2(t) - b_{n-j}^2(t)) \\ &= 2(b_j^2(t) + b_{n-j}^2(t)) - 2(b_{j-1}^2(t) - b_{n-j+1}^2(t)) \end{aligned}$$

and so

$$\begin{aligned}\varphi(\tau) - \varphi(-\tau) &= \int_{-\tau}^{\tau} \dot{\varphi}(t)dt \\ &= 2\int_{-\tau}^{\tau} (b_j^2(t) + b_{n-j}^2(t))dt - 2\int_{-\tau}^{\tau} (b_{j-1}^2(t) + b_{n-j+1}^2(t))dt.\end{aligned}$$

The last integral is bounded for all τ by the induction hypothesis, and $\varphi(\tau) - \varphi(-\tau)$ is also bounded for all τ, so $\int_{-\infty}^{\infty}(b_j^2(t) + b_{n-j}^2(t))dt$ must be bounded as desired.

Let $p(t) = \sum_{i=1}^{n-1} b_i^2(t)$. We now know that $\int_{-\infty}^{\infty} p(t)dt < \infty$, and since $p(t) \geq 0$ we want to conclude that $\lim_{t\to\pm\infty} p(t) = 0$. But we need to exclude the possibility that $p(t)$ has narrow spikes as $t \to \pm\infty$, in which case $\int_{-\infty}^{\infty} p(t)dt$ could be finite without $p(t)$ approaching 0. We show $p(t)$ has no spikes by showing its derivative is bounded:

$$|\dot{p}(t)| = \left|\sum_{i=1}^{n-1} 2\dot{b}_i(t)b_i(t)\right| = \left|\sum_{i=1}^{n-1} 2b_i^2(t)(a_{i+1}(t) - a_i(t))\right| \leq 4(n-1)\|T\|_2^2. \quad \square$$

Thus, in principle, one could use an ODE solver on the Toda flow to solve the eigenvalue problem, but this is no faster than other existing methods. The interest in the Toda flow lies in its close relationship with with QR algorithm.

DEFINITION 5.5. *Let $X_{\text{-}}$ denote the strictly lower triangle of X, and $\pi_0(X) = X_{\text{-}} - X_{\text{-}}{}^T$.*

Note that $\pi_0(X)$ is skew symmetric and that if X is already skew symmetric, then $\pi_0(X) = X$. Thus π_0 *projects* onto skew symmetric matrices.

Consider the differential equation

$$\frac{d}{dt}T = BT - TB, \tag{5.28}$$

where $B = -\pi_0(F(T))$ and F is any smooth function from the real numbers to the real numbers. Since $B = -B^T$, Theorem 5.16 shows that $T(t)$ has the same eigenvalues for all t. Choosing $F(x) = x$ corresponds to the Toda flow that we just studied, since in this case

$$-\pi_0(F(T)) = -\pi_0(T) = \begin{bmatrix} 0 & b_1 & & \\ -b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & -b_{n-1} & 0 \end{bmatrix} = B.$$

The next theorem relates the QR decomposition to the solution of differential equation (5.28).

THEOREM 5.18. *Let $F(T(0)) = F_0$. Let $e^{tF_0} = Q(t)R(t)$ be the QR decomposition. Then $T(t) = Q^T(t)T(0)Q(t)$ solves equation* (5.28).

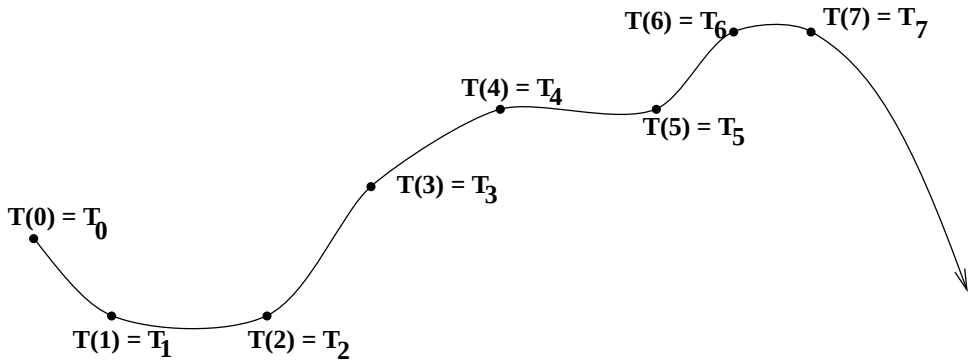
We delay the proof of the theorem until later. If we choose the function F correctly, it turns out that the iterates computed by QR iteration (Algorithm 4.4) are *identical* to the solutions of the differential equation.

DEFINITION 5.6. *Choosing $F(x) = \log x$ in equation (5.28) yields a differential equation called the* QR flow.

COROLLARY 5.3. *Let $F(x) = \log x$. Suppose that $T(0)$ is positive definite, so $\log T(0)$ is real. Let $T_0 \equiv T(0) = QR$, $T_1 = RQ$, etc. be the sequence of matrices produced by the unshifted QR iteration. Then $T(i) = T_i$. Thus the QR algorithm gives solutions to the QR flow at integer times t.*[22]

Proof of Corollary. At $t = 1$, we get $e^{t \log T_0} = T_0 = Q(1)R(1)$, the QR decomposition of T_0, and $T(1) = Q^T(1)T_0Q(1) = R(1)Q(1) = T_1$ as desired. Since the solution of the ODE is unique, this extends to show $T(i) = T_i$ for larger i. □

The following figure illustrates this corollary graphically. The curve represents the solution of the differential equation. The dots represent the solutions $T(i)$ at the integer times $t = 0, 1, 2, \ldots$, and indicates that they are equal to the QR iterates T_i.



Proof of Theorem 5.18. Differentiate $e^{tF_0} = QR$ to get

$$\begin{aligned}
F_0 e^{tF_0} &= \dot{Q}R + Q\dot{R} \\
\text{or } \dot{Q} &= F_0 e^{tF_0} R^{-1} - Q\dot{R}R^{-1} \\
\text{or } Q^T\dot{Q} &= Q^T F_0 e^{tF_0} R^{-1} - \dot{R}R^{-1} \\
&= Q^T F_0 (QR) R^{-1} - \dot{R}R^{-1} \quad \text{because } e^{tF_0} = QR \\
&= Q^T F(T(0))Q - \dot{R}R^{-1} \quad \text{because } F_0 = F(T(0)) \\
&= F(Q^T T(0) Q) - \dot{R}R^{-1} \\
&= F(T) - \dot{R}R^{-1}.
\end{aligned}$$

[22]Note that since the QR decomposition is not completely unique (Q can be replaced by QS and R can be replaced by SR, where S is a diagonal matrix with diagonal entries ± 1), T_i and $T(i)$ could actually differ by a similarity $T_i = ST(i)S^{-1}$. For simplicity we will assume here, and in Corollary 5.4, that S has been chosen so that $T_i = T(i)$.

Now $I = Q^TQ$ implies that $0 = \frac{d}{dt}Q^TQ = \dot{Q}^TQ + Q^T\dot{Q} = (Q^T\dot{Q})^T + (Q^T\dot{Q})$. This means $Q^T\dot{Q}$ is skew symmetric, and so $\pi_0(Q^T\dot{Q}) = Q^T\dot{Q} = \pi_0(F(T) - \dot{R}R^{-1})$. Since $\dot{R}R^{-1}$ is upper triangular, it doesn't affect π_0 and so finally $Q^T\dot{Q} = \pi_0(F(T))$. Now

$$\begin{aligned}
\tfrac{d}{dt}T(t) &= \tfrac{d}{dt}Q^T(t)T(0)Q(t) \\
&= \dot{Q}^TT(0)Q + Q^TT(0)\dot{Q} \\
&= \dot{Q}^T(QQ^T)T(0)Q + Q^TT(0)(QQ^T)\dot{Q} \\
&= \dot{Q}^TQT(t) + T(t)Q^T\dot{Q} \\
&= -Q^T\dot{Q}T(t) + T(t)Q^T\dot{Q} \\
&= -\pi_0(F(T(t)))T(t) + T(t)\pi_0(F(T(t)))
\end{aligned}$$

as desired. □

The next corollary explains the phenomenon observed in Question 4.15, where QR could be made to "run backward" and return to its starting matrix. See also Question 5.25.

COROLLARY 5.4. *Suppose that we obtain T_4 from the positive definite matrix T_0 by the following steps:*

1. *Do m steps of the unshifted QR algorithm on T_0 to get T_1.*
2. *Let T_2 = "flipped T_1" $= JT_1J$, where J equals the identity matrix with its columns in reverse order.*
3. *Do m steps of unshifted QR on T_2 to get T_3.*
4. *Let $T_4 = JT_3J$.*

Then $T_4 = T_0$.

Proof. If $X = X^T$, it is easy to verify that $\pi_0(JXJ) = -J\pi_0(X)J$ so $T_J(t) \equiv JT(t)J$ satisfies

$$\begin{aligned}
\tfrac{d}{dt}T_J(t) &= J\tfrac{d}{dt}T(t)J \\
&= J[-\pi_0(f(T))T + T\pi_0(F(T))]J \\
&= -J\pi_0(F(T))J(JTJ) + (JTJ)J\pi_0(F(T))J \quad \text{since } J^2 = I \\
&= \pi_0(JF(T)J)T_J - T_J\pi_0(JF(T)J) \\
&= \pi_0(F(JTJ))T_J - T_J\pi_0(F(JTJ)) \\
&= \pi_0(F(T_J))T_J - T_J\pi_0(F(T_J)).
\end{aligned}$$

This is nearly the same equation as $T(t)$. In fact, it satisfies *exactly* the same equation as $T(-t)$:

$$\frac{d}{dt}T(-t) = -\frac{d}{dt}T|_{-t} = -\left[\pi_0(F(T))T + T\pi_0(F(T))\right]_{-t}.$$

So with the same initial conditions T_2, $T_J(t)$ and $T(-t)$ must be equal. Integrating for time m, $T(-t)$ takes $T_2 = JT_1J$ back to JT_0J, the initial state, so $T_3 = JT_0J$ and $T_4 = JT_3J = T_0$ as desired. □

5.5.2. The Connection to Partial Differential Equations

This section may be skipped on a first reading.

Let $T(t) = -\frac{\partial^2}{\partial x^2} + q(x,t)$ and $B(t) = -4\frac{\partial^3}{\partial x^3} + 3(q(x,t)\frac{\partial}{\partial x} + \frac{\partial}{\partial x}q(x,t))$. Both $T(t)$ and $B(t)$ are linear operators on functions, i.e., generalizations of matrices.

Substituting into $\frac{dT}{dt} = BT - TB$ yields

$$q_t = 6qq_x - q_{xxx}, \tag{5.29}$$

provided that we choose the correct boundary conditions for q. (B must be skew symmetric and T symmetric.) Equation (5.29) is called the *Korteweg–de Vries equation* and describes water flow in a shallow channel. One can rigorously show that (5.29) preserves the eigenvalues of $T(t)$ for all t in the sense that the ODE

$$\left(-\frac{\partial^2}{\partial x^2} + q(x,t)\right) h(x) = \lambda h(x)$$

has some infinite set of eigenvalues $\lambda_1, \lambda_2, \ldots$ for all t. In other words, there is an infinite sequence of energylike quantities conserved by the Korteweg–de Vries equation. This is important for both theoretical and numerical reasons.

For more details on the Toda flow, see [142, 168, 66, 67, 237] and papers by Kruskal [164], Flaschka [104], and Moser [185] in [186].

5.6. References and Other Topics for Chapter 5

An excellent general reference for the symmetric eigenproblem is [195]. The material on relative perturbation theory can be found in [74, 81, 99]; section 5.2.1 was based on the latter of these references. Related work is found in [65, 90, 226, 248] A classical text on perturbation theory for general linear operators is [159]. For a survey of parallel algorithms for the symmetric eigenproblem, see [75]. The QR algorithm for finding the SVD of bidiagonal matrices is discussed in [79, 66, 118], and the dqds algorithm is in [102, 198, 207]. For an error analysis of the Bisection algorithm, see [72, 73, 154], and for recent attempts to accelerate Bisection see [103, 201, 199, 174, 171, 173, 267]. Current work in improving inverse iteration appears in [103, 199, 201]. The divide-and-conquer eigenroutine was introduced in [58] and further developed in [13, 88, 125, 129, 151, 170, 208, 232]. The possibility of high-accuracy eigenvalues obtained from Jacobi is discussed in [65, 74, 81, 90, 181, 226]. The Toda flow and related phenomena are discussed in [66, 67, 104, 142, 164, 168, 185, 186, 237].

5.7. Questions for Chapter 5

QUESTION 5.1. *(Easy; Z. Bai)* Show that $A = B + iC$ is Hermitian if and only if

$$M = \begin{bmatrix} B & -C \\ C & B \end{bmatrix}$$

is symmetric. Express the eigenvalues and eigenvectors of M in terms of those of A.

QUESTION 5.2. *(Medium)* Prove Corollary 5.1, using Weyl's theorem (Theorem 5.1) and part 4 of Theorem 3.3.

QUESTION 5.3. *(Medium)* Consider Figure 5.1. Consider the corresponding contour plot for an arbitrary 3-by-3 matrix A with eigenvalues $\alpha_3 \le \alpha_2 \le \alpha_1$. Let C_1 and C_2 be the two great circles along which $\rho(u, A) = \alpha_2$. At what angle do they intersect?

QUESTION 5.4. *(Hard)* Use the Courant–Fischer minimax theorem (Theorem 5.2) to prove the *Cauchy interlace theorem*:

- Suppose that $A = \begin{bmatrix} H & b \\ b^T & u \end{bmatrix}$ is an n-by-n symmetric matrix and H is $(n-1)$-by-$(n-1)$. Let $\alpha_n \le \cdots \le \alpha_1$ be the eigenvalues of A and $\theta_{n-1} \le \cdots \le \theta_1$ be the eigenvalues of H. Show that these two sets of eigenvalues *interlace*:

$$\alpha_n \le \theta_{n-1} \le \cdots \le \theta_i \le \alpha_i \le \theta_{i-1} \le \alpha_{i-1} \le \cdots \le \theta_1 \le \alpha_1.$$

- Let $A = \begin{bmatrix} H & B \\ B^T & U \end{bmatrix}$ be n-by-n and H be m-by-m, with eigenvalues $\theta_m \le \cdots \le \theta_1$. Show that the eigenvalues of A and H interlace in the sense that $\alpha_{j+(n-m)} \le \theta_j \le \alpha_j$ (or equivalently $\alpha_j \le \theta_{j-(n-m)} \le \alpha_{j-(n-m)}$).

QUESTION 5.5. *(Medium)* Let $A = A^T$ with eigenvalues $\alpha_1 \ge \cdots \ge \alpha_n$. Let $H = H^T$ with eigenvalues $\theta_1 \ge \cdots \ge \theta_n$. Let $A + H$ have eigenvalues $\lambda_1 \ge \cdots \ge \lambda_n$. Use the Courant–Fischer minimax theorem (Theorem 5.2) to show that $\alpha_j + \theta_n \le \lambda_j \le \alpha_j + \theta_1$. If H is positive definite, conclude that $\lambda_j > \alpha_j$. In other words, adding a symmetric positive definite matrix H to another symmetric matrix A can only increase its eigenvalues.

This result will be used in the proof of Theorem 7.1.

QUESTION 5.6. *(Medium)* Let $A = [A_1, A_2]$ be n-by-n, where A_1 is n-by-m and A_2 is n-by-$(n-m)$. Let $\sigma_1 \ge \cdots \ge \sigma_n$ be the singular values of A $\tau_1 \ge \cdots \ge \tau_m$ be the singular values of A_1. Use the Cauchy interlace theorem from Question 5.4 and part 4 of Theorem 3.3 to prove that $\sigma_j \ge \tau_j \ge \sigma_{j+n-m}$.

QUESTION 5.7. *(Medium)* Let q be a unit vector and d be any vector orthogonal to q. Show that $\|(q+d)q^T - I\|_2 = \|q+d\|_2$. (This result is used in the proof of Theorem 5.4.)

QUESTION 5.8. *(Hard)* Formulate and prove a theorem for singular vectors analogous to Theorem 5.4.

QUESTION 5.9. *(Hard)* Prove bound (5.6) from Theorem 5.5.

QUESTION 5.10. *(Harder)* Prove bound (5.7) from Theorem 5.5.

QUESTION 5.11. *(Easy)* Suppose $\theta = \theta_1 + \theta_2$, where all three angles lie between 0 and $\pi/2$. Prove that $\frac{1}{2}\sin 2\theta \leq \frac{1}{2}\sin 2\theta_1 + \frac{1}{2}\sin 2\theta_2$. This result is used in the proof of Theorem 5.7.

QUESTION 5.12. *(Hard)* Prove Corollary 5.2. Hint: Use part 4 of Theorem 3.3.

QUESTION 5.13. *(Medium)* Let A be a symmetric matrix. Consider running shifted QR iteration (Algorithm 4.5) with a Rayleigh quotient shift ($\sigma_i = a_{nn}$) at every iteration, yielding a sequence $\sigma_1, \sigma_2, \ldots$ of shifts. Also run Rayleigh quotient iteration (Algorithm 5.1), starting with $x_0 = [0, \ldots, 0, 1]^T$, yielding a sequence of Rayleigh quotients $\rho_1, \rho_2, \ldots$. Show that these sequences are identical: $\sigma_i = \rho_i$ for all i. This justifies the claim in section 5.3.2 that shifted QR iteration enjoys local cubic convergence.

QUESTION 5.14. *(Easy)* Prove Lemma 5.1.

QUESTION 5.15. *(Easy)* Prove that if $t(n) = 2t(n/2) + cn^3 + O(n^2)$, then $t(n) \approx c\frac{4}{3}n^3$. This justifies the complexity analysis of the divide-and-conquer algorithm (Algorithm 5.2).

QUESTION 5.16. *(Easy)* Let $A = D + \rho uu^T$, where $D = \mathrm{diag}(d_1, \ldots, d_n)$ and $u = [u_1, \ldots, u_n]^T$. Show that if $d_i = d_{i+1}$ or $u_i = 0$, then d_i is an eigenvalue of A. If $u_i = 0$, show that the eigenvector corresponding to d_i is e_i, the ith column of the identity matrix. Derive a similarly simple expression when $d_i = d_{i+1}$. This shows how to handle deflation in the divide-and-conquer algorithm, Algorithm 5.2.

QUESTION 5.17. *(Easy)* Let ψ and ψ' be given scalars. Show how to compute scalars c and $\hat{c}$ in the function definition $h(\lambda) = \hat{c} + \frac{c}{d-\lambda}$ so that at $\lambda = \xi$, $h(\xi) = \psi$, and $h'(\xi) = \psi'$. This result is needed to derive the secular equation solver in section 5.3.3.

QUESTION 5.18. *(Easy; Z. Bai)* Use the SVD to show that if A is an m-by-n real matrix with $m \geq n$, then there exists an m-by-n matrix Q with orthonormal columns ($Q^TQ = I$) and an n-by-n positive semidefinite matrix P such that $A = QP$. This decomposition is called the *polar decomposition of* A, because it is analogous to the polar form of a complex number $z = e^{i\arg(z)} \cdot |z|$.) Show that if A is nonsingular, then the polar decomposition is unique.

QUESTION 5.19. *(Easy)* Prove Lemma 5.5.

QUESTION 5.20. *(Easy)* Prove Lemma 5.7.

QUESTION 5.21. *(Hard)* Prove Theorem 5.13. Also, reduce the exponent $4n-2$ in Theorem 5.13 to $2n-1$. Hint: In Lemma 5.7, multiply D_1 and divide D_2 by an appropriately chosen constant.

QUESTION 5.22. *(Medium)* Prove that Algorithm 5.13 computes the SVD of G, assuming that G^TG converges to a diagonal matrix.

QUESTION 5.23. *(Harder)* Let A be an n-by-n symmetric positive definite matrix with Cholesky decomposition $A = LL^T$, and let $\hat{L}$ be the Cholesky factor computed in floating point arithmetic. In this question we will bound the relative error in the (squared) singular values of $\hat{L}$ as approximations of the eigenvalues of A. Show that A can be written $A = D\bar{A}D$, where $D = \mathrm{diag}(a_{11}^{1/2}, \ldots, a_{nn}^{1/2})$ and $\bar{a}_{ii} = 1$ for all i. Write $L = DX$. Show that $\kappa^2(X) = \kappa(\bar{A})$. Using bound (2.16) for the backward error δA of Cholesky $A+\delta A = \hat{L}\hat{L}^T$, show that one can write $\hat{L}^T\hat{L} = Y^TL^TLY$, where $\|Y^TY - I\|_2 \leq O(\varepsilon)\kappa(\bar{A})$. Use Theorem 5.6 to conclude that the eigenvalues of $\hat{L}^T\hat{L}$ and of L^TL differ relatively by at most $O(\varepsilon)\kappa(\bar{A})$. Then show that this is also true of the eigenvalues of $\hat{L}\hat{L}^T$ and LL^T. This means that the squares of the singular values of $\hat{L}$ differ relatively from the eigenvalues of A by at most $O(\varepsilon)\kappa(\bar{A}) = O(\varepsilon)\kappa^2(L)$.

QUESTION 5.24. *(Harder)* This question justifies the stopping criterion for one-sided Jacobi's method for the SVD (Algorithm 5.13). Let $A = G^TG$, where G and A are n-by-n. Suppose that $|a_{jk}| \leq \epsilon\sqrt{a_{jj}a_{kk}}$ for all $j = k$. Let $\sigma_n \leq \cdots \leq \sigma_1$ be the singular values of G, and $\alpha_n^2 \leq \cdots \leq \alpha_1^2$ be the sorted diagonal entries of A. Prove that $|\sigma_i - \alpha_i| \leq n\epsilon|\alpha_i|$ so that the α_i equal the singular values to high relative accuracy. Hint: Use Corollary 5.2.

QUESTION 5.25. *(Harder)* In Question 4.15, you "noticed" that running QR for m steps on a symmetric matrix, "flipping" the rows and columns, running for another m steps, and flipping again got you back to the original matrix. (Flipping X means replacing X by JXJ, where J is the identity matrix with its row in reverse order.) In this exercise we will prove this for symmetric positive definite matrices T using an approach different from Corollary 5.4.

Consider LR iteration (Algorithm 5.9) with a zero shift, applied to the symmetric positive definite matrix T (which is not necessarily tridiagonal): Let $T = T_0 = B_0^TB_0$ be the Cholesky decomposition, $T_1 = B_0B_0^T = B_1^TB_1$, and more generally $T_i = B_{i-1}B_{i-1}^T = B_i^TB_i$. Let $\hat{T}_i$ denote the matrix obtained from T_0 after i steps of unshifted QR iteration; i.e., if $\hat{T}_i = Q_iR_i$ is the QR decomposition, then $\hat{T}_{i+1} = R_iQ_i$. In Lemma 5.6 we showed that $\hat{T}_i = T_{2i}$; i.e., one step of QR is the same as two steps of LR.

1. Show that $T_i = (B_{i-1}B_{i-2}\cdots B_0)^{-T}T_0(B_{i-1}B_{i-2}\cdots B_0)^T$.
2. Show that $T_i = (B_{i-1}B_{i-2}\cdots B_0)T_0(B_{i-1}B_{i-2}\cdots B_0)^{-1}$.
3. Show that $T_0^i = (B_iB_{i-1}\cdots B_0)^T(B_iB_{i-1}\cdots B_0)$ is the Cholesky decomposition of T_0^i.
4. Show that $T_0^i = (Q_0\cdots Q_{i-2}Q_{i-1})\cdot(R_{i-1}R_{i-2}\cdots R_0)$ is the QR decomposition of T_0^i.
5. Show that $T_0^{2i} = (R_{2i-1}R_{2i-2}\cdots R_0)^T(R_{2i-1}R_{2i-2}\cdots R_0)$ is the Cholesky decomposition of T_0^{2i}.
6. Show the result after m steps of QR, flipping m steps of QR, and flipping, is the same as the original matrix. Hint: Use the fact that the Cholesky factorization is unique.

QUESTION 5.26. *(Hard; Z. Bai)* Suppose that x is an n-vector. Define the matrix C by $c_{ij} = |x_i|+|x_j|-|x_i-x_j|$. Show that $C(x)$ is positive semidefinite.

QUESTION 5.27. *(Easy; Z. Bai)* Let

$$A = \begin{pmatrix} I & B \\ B^H & I \end{pmatrix}$$

with $\|B\|_2 < 1$. Show that

$$\|A\|_2\|A^{-1}\|_2 = \frac{1+\|B\|_2}{1-\|B\|_2}.$$

QUESTION 5.28. *(Medium; Z. Bai)* A square matrix A is said to be *skew Hermitian* if $A^* = -A$. Prove that

1. the eigenvalues of a skew Hermitian are purely imaginary.
2. $I - A$ is nonsingular.
3. $C = (I-A)^{-1}(I+A)$ is unitary. C is called the *Cayley transform* of A.