4

Nonsymmetric Eigenvalue Problems

4.1. Introduction

We discuss canonical forms (in section 4.2), perturbation theory (in section 4.3), and algorithms for the eigenvalue problem for a single nonsymmetric matrix A (in section 4.4). Chapter 5 is devoted to the special case of real symmetric matrices $A = A^T$ (and the SVD). Section 4.5 discusses generalizations to eigenvalue problems involving more than one matrix, including motivating applications from the analysis of vibrating systems, the solution of linear differential equations, and computational geometry. Finally, section 4.6 summarizes all the canonical forms, algorithms, costs, applications, and available software in a list.

One can roughly divide the algorithms for the eigenproblem into two groups: *direct methods* and *iterative methods*. This chapter considers only direct methods, which are intended to compute all of the eigenvalues, and (optionally) eigenvectors. Direct methods are typically used on dense matrices and cost $O(n^3)$ operations to compute all eigenvalues and eigenvectors; this cost is relatively insensitive to the actual matrix entries.

The main direct method used in practice is *QR iteration with implicit shifts* (see section 4.4.8). It is interesting that after more than 30 years of dependable service, convergence failures of this algorithm have quite recently been observed, analyzed, and patched [25, 65]. But there is still no global convergence proof, even though the current algorithm is considered quite reliable. So the problem of devising an algorithm that is numerically stable and globally (and quickly!) convergent remains open. (Note that "direct" methods must still iterate, since finding eigenvalues is mathematically equivalent to finding zeros of polynomials, for which no noniterative methods can exist. We call a method *direct* if experience shows that it (nearly) never fails to converge in a fixed number of iterations.)

Iterative methods, which are discussed in Chapter 7, are usually applied to sparse matrices or matrices for which matrix-vector multiplication is the only convenient operation to perform. Iterative methods typically provide

approximations only to a subset of the eigenvalues and eigenvectors and are usually run only long enough to get a few adequately accurate eigenvalues rather than a large number. Their convergence properties depend strongly on the matrix entries.

4.2. Canonical Forms

DEFINITION 4.1. *The polynomial* $p(\lambda) = \det(A - \lambda I)$ *is called the* characteristic polynomial *of* A. *The roots of* $p(\lambda) = 0$ *are the* eigenvalues *of* A.

Since the degree of the characteristic polynomial $p(\lambda)$ equals n, the dimension of A, it has n roots, so A has n eigenvalues.

DEFINITION 4.2. *A nonzero vector* x *satisfying* $Ax = \lambda x$ *is a* (right) eigenvector *for the eigenvalue* λ. *A nonzero vector* y *such that* $y^*A = \lambda y^*$ *is a* left eigenvector. *(Recall that* $y^* = (\overline{y})^T$ *is the* conjugate transpose *of* y.*)*

Most of our algorithms will involve transforming the matrix A into simpler, or *canonical* forms, from which it is easy to compute its eigenvalues and eigenvectors. These transformations are called *similarity transformations* (see below). The two most common canonical forms are called the *Jordan form* and *Schur form.* The Jordan form is useful theoretically but is very hard to compute in a numerically stable fashion, which is why our algorithms will aim to compute the Schur form instead.

To motivate Jordan and Schur forms, let us ask which matrices have the property that their eigenvalues are easy to compute. The easiest case would be a *diagonal matrix*, whose eigenvalues are simply its diagonal entries. Equally easy would be a *triangular matrix*, whose eigenvalues are also its diagonal entries. Below we will see that a matrix in Jordan or Schur form is triangular. But recall that a real matrix can have complex eigenvalues, since the roots of its characteristic polynomial may be real or complex. Therefore, there is not always a *real* triangular matrix with the same eigenvalues as a *real* general matrix, since a real triangular matrix can only have real eigenvalues. Therefore, we must either use complex numbers or look beyond real triangular matrices for our canonical forms for real matrices. It will turn out to be sufficient to consider *block triangular matrices*, i.e., matrices of the form

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1b} \\ & A_{22} & \cdots & A_{2b} \\ & & \ddots & \vdots \\ & & & A_{bb} \end{bmatrix}, \tag{4.1}$$

where each A_{ii} is square and all entries below the A_{ii} blocks are zero. One can easily show that the characteristic polynomial $\det(A - \lambda I)$ of A is the product

$\prod_{i=1}^{b} \det(A_{ii} - \lambda I)$ of the characteristic polynomials of the A_{ii} and therefore that the set $\lambda(A)$ of eigenvalues of A is the union $\cup_{i=1}^{b} \lambda(A_{ii})$ of the sets of eigenvalues of the diagonal blocks A_{ii} (see Question 4.1). The canonical forms that we compute will be block triangular and will proceed computationally by breaking up large diagonal blocks into smaller ones. If we start with a complex matrix A, the final diagonal blocks will be 1-by-1, so the ultimate canonical form will be triangular. If we start with a real matrix A, the ultimate canonical form will have 1-by-1 diagonal blocks (corresponding to real eigenvalues) and 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues); such a block triangular matrix is called *quasi-triangular.*

It is also easy to find the eigenvectors of a (block) triangular matrix; see section 4.2.1.

DEFINITION 4.3. *Let S be any nonsingular matrix. Then A and $B = S^{-1}AS$ are called* similar *matrices, and S is a* similarity transformation.

PROPOSITION 4.1. *Let $B = S^{-1}AS$, so A and B are similar. Then A and B have the same eigenvalues, and x (or y) is a right (or left) eigenvector of A if and only if $S^{-1}x$ (or S^*y) is a right (or left) eigenvector of B.*

Proof. Using the fact that $\det(X \cdot Y) = \det(X) \cdot \det(Y)$ for any square matrices X and Y, we can write $\det(A - \lambda I) = \det(S^{-1}(A - \lambda I)S) = \det(B - \lambda I)$, so A and B have the same characteristic polynomials. $Ax = \lambda x$ holds if and only if $S^{-1}ASS^{-1}x = \lambda S^{-1}x$ or $B(S^{-1}x) = \lambda(S^{-1}x)$. Similarly, $y^*A = \lambda y^*$ if and only if $y^*SS^{-1}AS = \lambda y^*S$ or $(S^*y)^*B = \lambda(S^*y)^*$. □

THEOREM 4.1. Jordan canonical form. *Given A, there exists a nonsingular S such that $S^{-1}AS = J$, where J is in* Jordan canonical form. *This means that J is block diagonal, with $J = \mathrm{diag}(J_{n_1}(\lambda_1), J_{n_2}(\lambda_2), \ldots, J_{n_k}(\lambda_k))$ and*

$$J_{n_i}(\lambda_i) = \begin{bmatrix} \lambda_i & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{bmatrix}^{n_i \times n_i}.$$

J is unique, up to permutations of its diagonal blocks.

For a proof of this theorem, see a book on linear algebra such as [110] or [139].

Each $J_m(\lambda)$ is called a *Jordan block* with eigenvalue λ of *algebraic multiplicity* m. If some $n_i = 1$, and λ_i is an eigenvalue of only that one Jordan block, then λ_i is called a *simple eigenvalue.* If all $n_i = 1$, so that J is diagonal, A is called *diagonalizable*; otherwise it is called *defective.* An n-by-n defective matrix does *not* have n eigenvectors, as described in more detail in the next

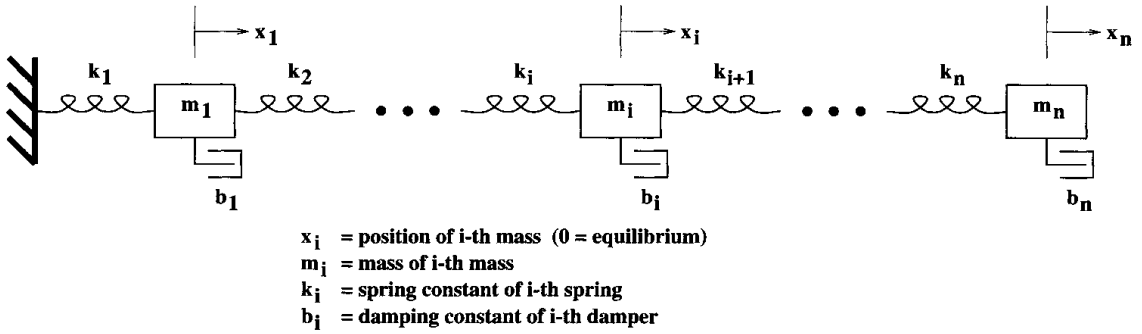


Fig. 4.1. *Damped, vibrating mass-spring system.*

proposition. Although defective matrices are "rare" in a certain well-defined sense, the fact that some matrices do not have n eigenvectors is a fundamental fact confronting anyone designing algorithms to compute eigenvectors and eigenvalues. In section 4.3, we will see some of the difficulties that such matrices cause. Symmetric matrices, discussed in Chapter 5, are never defective.

PROPOSITION 4.2. *A Jordan block has one right eigenvector,* $e_1 = [1, 0, \ldots, 0]^T$, *and one left eigenvector,* $e_n = [0, \ldots, 0, 1]^T$. *Therefore, a matrix has n eigenvectors matching its n eigenvalues if and only if it is diagonalizable. In this case,* $S^{-1}AS = \text{diag}(\lambda_i)$. *This is equivalent to* $AS = S\ \text{diag}(\lambda_i)$, *so the ith column of S is a right eigenvector for* λ_i. *It is also equivalent to* $S^{-1}A = \text{diag}(\lambda_i)S^{-1}$, *so the conjugate transpose of the ith row of* S^{-1} *is a left eigenvector for* λ_i. *If all n eigenvalues of a matrix A are distinct, then A is diagonalizable.*

Proof. Let $J = J_m(\lambda)$ for ease of notation. It is easy to see $Je_1 = \lambda e_1$ and $e_n^T J = \lambda e_n^T$, so e_1 and e_n are right and left eigenvectors of J, respectively. To see that J has only one right eigenvector (up to scalar multiples), note that any eigenvector x must satisfy $(J - \lambda I)x = 0$, so x is in the null space of

$$J - \lambda I = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}.$$

But the null space of $J - \lambda I$ is clearly $\text{span}(e_1)$, so there is just one eigenvector. If all eigenvalues of A are distinct, then all its Jordan blocks must be 1-by-1, so $J = \text{diag}(\lambda_1, \ldots, \lambda_n)$ is diagonal. □

EXAMPLE 4.1. We illustrate the concepts of eigenvalue and eigenvector with a problem of *mechanical vibrations.* We will see a defective matrix arise in a natural physical context. Consider the damped mass spring system in Figure 4.1, which we will use to illustrate a variety of eigenvalue problems.

Newton's law $F = ma$ applied to this system yields

$$\begin{aligned} m_i\ddot{x}_i(t) &= k_i(x_{i-1}(t) - x_i(t)) \\ &\qquad \text{force on mass } i \text{ from spring } i \\ &\quad +k_{i+1}(x_{i+1}(t) - x_i(t)) \\ &\qquad \text{force on mass } i \text{ from spring } i+1 \\ &\quad -b_i\dot{x}_i(t) \\ &\qquad \text{force on mass } i \text{ from damper } i \end{aligned} \tag{4.2}$$

or

$$M\ddot{x}(t) = -B\dot{x}(t) - Kx(t), \tag{4.3}$$

where $M = \text{diag}(m_1, \ldots, m_n)$, $B = \text{diag}(b_1, \ldots, b_n)$, and

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

We assume that all the masses m_i are positive. M is called the *mass matrix*, B is the *damping matrix*, and K is the *stiffness matrix*.

Electrical engineers analyzing linear circuits arrive at an analogous equation by applying Kirchoff's and related laws instead of Newton's law. In this case x represents branch currents, M represent inductances, B represents resistances, and K represents admittances (reciprocal capacitances).

We will use a standard trick to change this second-order differential equation to a first-order differential equation, changing variables to

$$y(t) = \begin{bmatrix} \dot{x}(t) \\ x(t) \end{bmatrix}.$$

This yields

$$\begin{aligned} \dot{y}(t) &= \begin{bmatrix} \ddot{x}(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} -M^{-1}B\dot{x}(t) - M^{-1}Kx(t) \\ \dot{x}(t) \end{bmatrix} \\ &= \begin{bmatrix} -M^{-1}B & -M^{-1}K \\ I & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}(t) \\ x(t) \end{bmatrix} \\ &= \begin{bmatrix} -M^{-1}B & -M^{-1}K \\ I & 0 \end{bmatrix} \cdot y(t) \equiv Ay(t). \end{aligned} \tag{4.4}$$

To solve $\dot{y}(t) = Ay(t)$, we assume that $y(0)$ is given (i.e., the initial positions $x(0)$ and velocities $\dot{x}(0)$ are given).

One way to write down the solution of this differential equation is $y(t) = e^{At}y(0)$, where e^{At} is the matrix exponential. We will give another more elementary solution in the special case where A is diagonalizable; this will be

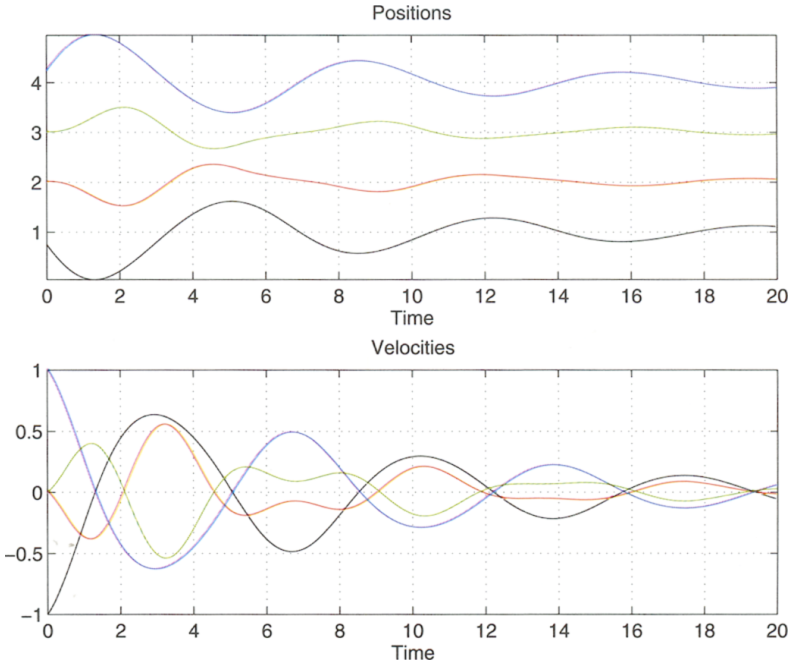


Fig. 4.2. *Positions and velocities of a mass-spring system with four masses* $m_1 = m_4 = 2$ *and* $m_2 = m_3 = 1$. *The spring constants are all* $k_i = 1$. *The damping constants are all* $b_i = .4$. *The initial displacements are* $x_1(0) = -.25$, $x_2(0) = x_3(0) = 0$, *and* $x_4(0) = .25$. *The initial velocities are* $v_1(0) = -1$, $v_2(0) = v_3(0) = 0$, *and* $v_4(0) = 1$. *The equilibrium positions are* 1, 2, 3, *and* 4. *The software for solving and plotting an arbitrary mass-spring system is HOMEPAGE/Matlab/massspring.m.*

true for almost all choices of m_i, k_i, and b_i. We will return to consider other situations later. (The general problem of computing matrix functions such as e^{At} is discussed further in section 4.5.1 and Question 4.4.)

When A is diagonalizable, we can write $A = S\Lambda S^{-1}$, where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$. Then $\dot{y}(t) = Ay(t)$ is equivalent to $\dot{y}(t) = S\Lambda S^{-1}y(t)$ or $S^{-1}\dot{y}(t) = \Lambda S^{-1}y(t)$ or $\dot{z}(t) = \Lambda z(t)$, where $z(t) \equiv S^{-1}y(t)$. This diagonal system of differential equations $\dot{z}_i(t) = \lambda_i z_i(t)$ has solutions $z_i(t) = e^{\lambda_i t}z_i(0)$, so $y(t) = S\text{diag}(e^{\lambda_1 t}, \ldots, e^{\lambda_n t})S^{-1}y(0) = Se^{\Lambda t}S^{-1}y(0)$. A sample numerical solution for four masses and springs is shown in Figure 4.2.

To see the physical significance of the nondiagonalizability of A for a mass-spring system, consider the case of a single mass, spring, and damper, whose differential equation we can simplify to $m\ddot{x}(t) = -b\dot{x}(t) - kx(t)$, and so $A = \begin{bmatrix} -b/m & -k/m \\ 1 & 0 \end{bmatrix}$. The two eigenvalues of A are $\lambda_\pm = \frac{b}{2m}(-1 \pm (1 - \frac{4km}{b^2})^{1/2})$. When $\frac{4km}{b^2} < 1$, the system is *overdamped*, and there are two negative real eigenvalues, whose mean value is $-\frac{b}{2m}$. In this case the solution eventually decays monotonically to zero. When $\frac{4km}{b^2} > 1$, the system is *underdamped*, and there are two complex conjugate eigenvalues with real part $-\frac{b}{2m}$. In this case the solution oscillates while decaying to zero. In both cases the system is diagonalizable since the eigenvalues are distinct. When $\frac{4km}{b^2} = 1$, the system

is *critically damped*, there are two real eigenvalues equal to $-\frac{b}{2m}$, and A has a single 2-by-2 Jordan block with this eigenvalue. In other words, the non-diagonalizable matrices form the "boundary" between two physical behaviors: oscillation and monotonic decay.

When A is diagonalizable but S is an ill-conditioned matrix, so that S^{-1} is difficult to evaluate accurately, the explicit solution $y(t) = Se^{\Lambda t}S^{-1}y(0)$ will be quite inaccurate and useless numerically. We will use this mechanical system as a running example because it illustrates so many eigenproblems. $\diamond$

To continue our discussion of canonical forms, it is convenient to define the following generalization of an eigenvector.

DEFINITION 4.4. *An* invariant subspace *of A is a subspace $\mathbf{X}$ of $\mathbb{R}^n$, with the property that $x \in \mathbf{X}$ implies that $Ax \in \mathbf{X}$. We also write this as $A\mathbf{X} \subseteq \mathbf{X}$.*

The simplest, one-dimensional invariant subspace is the set $\text{span}(x)$ of all scalar multiples of an eigenvector x. Here is the analogous way to build an invariant subspace of larger dimension. Let $X = [x_1, \ldots, x_m]$, where $x_1, \ldots, x_m$ are any set of independent eigenvectors with eigenvalues $\lambda_1, \ldots, \lambda_m$. Then $\mathbf{X} = \text{span}(X)$ is an invariant subspace since $x \in \mathbf{X}$ implies $x = \sum_{i=1}^m \alpha_i x_i$ for some scalars α_i, so $Ax = \sum_{i=1}^m \alpha_i Ax_i = \sum_{i=1}^m \alpha_i \lambda_i x_i \in \mathbf{X}$. $A\mathbf{X}$ will equal $\mathbf{X}$ unless some eigenvalue λ_i equals zero. The next proposition generalizes this.

PROPOSITION 4.3. *Let A be n-by-n, let $X = [x_1, \ldots, x_m]$ be any n-by-m matrix with independent columns, and let $\mathbf{X} = \text{span}(X)$, the m-dimensional space spanned by the columns of X. Then $\mathbf{X}$ is an invariant subspace if and only if there is an m-by-m matrix B such that $AX = XB$. In this case the m eigenvalues of B are also eigenvalues of A. (When $m = 1$, $X = [x_1]$ is an eigenvector and B is an eigenvalue.)*

Proof. Assume first that $\mathbf{X}$ is invariant. Then each Ax_i is also in $\mathbf{X}$, so each Ax_i must be a linear combination of a basis of $\mathbf{X}$, say, $Ax_i = \sum_{j=1}^m x_j b_{ji}$. This last equation is equivalent to $AX = XB$. Conversely, $AX = XB$ means that each Ax_i is a linear combination of columns of X, so $\mathbf{X}$ is invariant.

Now assume $AX = XB$. Choose any n-by-$(n-m)$ matrix $\tilde{X}$ such that $\hat{X} = [X, \tilde{X}]$ is nonsingular. Then A and $\hat{X}^{-1}A\hat{X}$ are similar and so have the same eigenvalues. Write $\hat{X}^{-1} = \begin{bmatrix} Y^{m\times n} \\ \tilde{Y}^{(n-m)\times n} \end{bmatrix}$, so $\hat{X}^{-1}\hat{X} = I$ implies $YX = I$ and $\tilde{Y}X = 0$. Then $\hat{X}^{-1}A\hat{X} = \begin{bmatrix} Y \\ \tilde{Y} \end{bmatrix} \cdot [AX, A\tilde{X}] = \begin{bmatrix} YAX & YA\tilde{X} \\ \tilde{Y}AX & \tilde{Y}A\tilde{X} \end{bmatrix} = \begin{bmatrix} YXB & YA\tilde{X} \\ \tilde{Y}XB & \tilde{Y}A\tilde{X} \end{bmatrix} = \begin{bmatrix} B & YA\tilde{X} \\ 0 & \tilde{Y}A\tilde{X} \end{bmatrix}$. Thus by Question 4.1 the eigenvalues of A are the union of the eigenvalues of B and the eigenvalues of $\tilde{Y}A\tilde{X}$. $\square$

For example, write the Jordan canonical form $S^{-1}AS = J = \text{diag}(J_{n_i}(\lambda_i))$ as $AS = SJ$, where $S = [S_1, S_2, \ldots, S_k]$ and S_i has n_i columns

(the same as $J_{n_i}(\lambda_i)$; see Theorem 4.1 for notation). Then $AS = SJ$ implies $AS_i = S_i J_{n_i}(\lambda_i)$, i.e., span$(S_i)$ is an invariant subspace.

The Jordan form tells everything that we might want to know about a matrix and its eigenvalues, eigenvectors, and invariant subspaces. There are also explicit formulas based on the Jordan form to compute e^A or any other function of a matrix (see section 4.5.1). But it is bad to compute the Jordan form for two numerical reasons:

First reason: It is a discontinuous function of A, so any rounding error can change it completely.

EXAMPLE 4.2. Let

$$J_n(0) = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix},$$

which is in Jordan form. For arbitrarily small ϵ, adding $i \cdot \epsilon$ to the (i,i) entry changes the eigenvalues to the n distinct values $i \cdot \epsilon$, and so the Jordan form changes from $J_n(0)$ to diag$(\epsilon, 2\epsilon, \ldots, n\epsilon)$. $\diamond$

Second reason: It cannot be computed stably in general. In other words, when we have finished computing S and J, we cannot guarantee that $S^{-1}(A + \delta A)S = J$ for some small δA.

EXAMPLE 4.3. Suppose $S^{-1}AS = J$ exactly, where S is very ill-conditioned. ($\kappa(S) = \|S\| \cdot \|S^{-1}\|$ is very large.) Suppose that we are extremely lucky and manage to compute S *exactly* and J with just a tiny error δJ with $\|\delta J\| = O(\varepsilon)\|A\|$. How big is the backward error? In other words, how big must δA be so that $S^{-1}(A + \delta A)S = J + \delta J$? We get $\delta A = S\delta J S^{-1}$, and all that we can conclude is that $\|\delta A\| \leq \|S\| \cdot \|\delta J\| \cdot \|S^{-1}\| = O(\varepsilon)\kappa(S)\|A\|$. Thus $\|\delta A\|$ may be much larger than $\varepsilon\|A\|$, which prevents backward stability. $\diamond$

So instead of computing $S^{-1}AS = J$, where S can be an arbitrarily ill-conditioned matrix, we will restrict S to be orthogonal (so $\kappa_2(S) = 1$) to guarantee stability. We cannot get a canonical form as simple as the Jordan form any more, but we do get something almost as good.

THEOREM 4.2. Schur canonical form. *Given A, there exists a unitary matrix Q and an upper triangular matrix T such that $Q^*AQ = T$. The eigenvalues of A are the diagonal entries of T.*

Proof. We use induction on n. It is obviously true if A is 1 by 1. Now let λ be any eigenvalue and u a corresponding eigenvector normalized so $\|u\|_2 = 1$. Choose $\tilde{U}$ so $U = [u, \tilde{U}]$ is a square unitary matrix. (Note that λ and u may be complex even if A is real.) Then

$$U^* \cdot A \cdot U = \begin{bmatrix} u^* \\ \tilde{U}^* \end{bmatrix} \cdot A \cdot [u, \tilde{U}] = \begin{bmatrix} u^*Au & u^*A\tilde{U} \\ \tilde{U}^*Au & \tilde{U}^*A\tilde{U} \end{bmatrix}.$$

Now as in the proof of Proposition 4.3, we can write $u^*Au = \lambda u^*u = \lambda$, and $\tilde{U}^*Au = \lambda\tilde{U}^*u = 0$ so $U^*AU \equiv \begin{bmatrix} \lambda & \tilde{a}_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix}$. By induction, there is a unitary P, so $P^*A_{22}P = \tilde{T}$ is upper triangular. Then

$$U^*AU = \begin{bmatrix} \lambda & \tilde{a}_{12} \\ 0 & P\tilde{T}P^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & P \end{bmatrix} \begin{bmatrix} \lambda & \tilde{a}_{12}P \\ 0 & \tilde{T} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & P^* \end{bmatrix},$$

so

$$\begin{bmatrix} 1 & 0 \\ 0 & P^* \end{bmatrix} U^*AU \begin{bmatrix} 1 & 0 \\ 0 & P \end{bmatrix} = \begin{bmatrix} \lambda & \tilde{a}_{12}P \\ 0 & \tilde{T} \end{bmatrix} = T$$

is upper triangular and $Q = U\begin{bmatrix} 1 & 0 \\ 0 & P \end{bmatrix}$ is unitary as desired. □

Notice that the Schur form is not unique, because the eigenvalues may appear on the diagonal of T in any order.

This introduces complex numbers even when A is real. When A is real, we prefer a canonical form that uses only real numbers, because it will be cheaper to compute. As mentioned at the beginning of this section, this means that we will have to sacrifice a *triangular* canonical form and settle for a *block-triangular* canonical form.

THEOREM 4.3. Real Schur canonical form. *If A is real, there exists a real orthogonal matrix V such that $V^TAV = T$ is quasi–upper triangular. This means that T is block upper triangular with 1-by-1 and 2-by-2 blocks on the diagonal. Its eigenvalues are the eigenvalues of its diagonal blocks. The 1-by-1 blocks correspond to real eigenvalues, and the 2-by-2 blocks to complex conjugate pairs of eigenvalues.*

Proof. We use induction as before. Let λ be an eigenvalue. If λ is real, it has a real eigenvector u and we proceed as in the last theorem. If λ is complex, let u be a (necessarily) complex eigenvector, so $Au = \lambda u$. Since $\overline{Au} = A\overline{u} = \overline{\lambda}\overline{u}$, $\overline{\lambda}$ and $\overline{u}$ are also an eigenvalue/eigenvector pair. Let $u_R = \frac{1}{2}u + \frac{1}{2}\overline{u}$ be the real part of u and $u_I = \frac{1}{2i}u - \frac{1}{2i}\overline{u}$ be the imaginary part. Then $\text{span}\{u_R, u_I\} = \text{span}\{u, \overline{u}\}$ is a two-dimensional invariant subspace. Let $\tilde{U} = [u_R, u_I]$ and $\tilde{U} = QR$ be its QR decomposition. Thus $\text{span}\{Q\} = \text{span}\{u_R, u_I\}$ is invariant. Choose $\tilde{Q}$ so that $U = [Q, \tilde{Q}]$ is real and orthogonal, and compute

$$U^T \cdot A \cdot U = \begin{bmatrix} Q^T \\ \tilde{Q}^T \end{bmatrix} \cdot A \cdot [Q, \tilde{Q}] = \begin{bmatrix} Q^TAQ & Q^TA\tilde{Q} \\ \tilde{Q}^TAQ & \tilde{Q}^TA\tilde{Q} \end{bmatrix}.$$

Since Q spans an invariant subspace, there is a 2-by-2 matrix B such that $AQ = QB$. Now as in the proof of Proposition 4.3, we can write $Q^TAQ = Q^TQB = B$ and $\tilde{Q}^TAQ = \tilde{Q}^TQB = 0$, so $U^TAU = \begin{bmatrix} B & Q^TA\tilde{Q} \\ 0 & \tilde{Q}^TA\tilde{Q} \end{bmatrix}$. Now apply induction to $\tilde{Q}^TA\tilde{Q}$. □

4.2.1. Computing Eigenvectors from the Schur Form

Let $Q^*AQ = T$ be the Schur form. Then if $Tx = \lambda x$, we have $AQx = QTx = \lambda Qx$, so Qx is an eigenvector of A. So to find eigenvectors of A, it suffices to find eigenvectors of T.

Suppose that $\lambda = t_{ii}$ has multiplicity 1 (i.e., it is simple). Write $(T-\lambda I)x = 0$ as

$$\begin{aligned} 0 &= \begin{bmatrix} T_{11} - \lambda I & T_{12} & T_{13} \\ 0 & 0 & T_{23} \\ 0 & 0 & T_{33} - \lambda I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= \begin{bmatrix} (T_{11} - \lambda I)x_1 + T_{12}x_2 + T_{13}x_3 \\ T_{23}x_3 \\ (T_{33} - \lambda I)x_3 \end{bmatrix}, \end{aligned}$$

where T_{11} is $(i-1)$-by-$(i-1)$, $T_{22} = \lambda$ is 1-by-1, T_{33} is $(n-i)$-by-$(n-i)$, and x is partitioned conformably. Since λ is simple, both $T_{11} - \lambda I$ and $T_{33} - \lambda I$ are nonsingular, so $(T_{33} - \lambda I)x_3 = 0$ implies $x_3 = 0$. Therefore $(T_{11} - \lambda I)x_1 = -T_{12}x_2$. Choosing (arbitrarily) $x_2 = 1$ means $x_1 = -(T_{11} - \lambda I)^{-1}T_{12}$, so

$$x = \begin{bmatrix} (\lambda I - T_{11})^{-1}T_{12} \\ 1 \\ 0 \end{bmatrix}.$$

In other words, we just need to solve a triangular system for x_1. To find a *real* eigenvector from real Schur form, we get a quasi-triangular system to solve. Computing complex eigenvectors from real Schur form using only real arithmetic also just involves equation solving but is a little trickier. See subroutine `strevc` in LAPACK for details.

4.3. Perturbation Theory

In this section we will concentrate on understanding when eigenvalues are ill-conditioned and thus hard to compute accurately. In addition to providing error bounds for computed eigenvalues, we will also relate eigenvalue condition numbers to related quantities, including the distance to the nearest matrix with an *infinitely* ill-conditioned eigenvalue, and the condition number of the matrix of eigenvectors.

We begin our study by asking when eigenvalues have *infinite* condition numbers. This is the case for multiple eigenvalues, as the following example illustrates.

EXAMPLE 4.4. Let

$$A = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ \epsilon & & & 0 \end{bmatrix}$$

be an n-by-n matrix. Then A has characteristic polynomial $\lambda^n - \epsilon = 0$ so $\lambda = \sqrt[n]{\epsilon}$ (n possible values). The nth root of ϵ grows much faster than any multiple of ϵ for small ϵ. More formally, the condition number is infinite because $\frac{d\lambda}{d\epsilon} = \frac{\epsilon^{\frac{1}{n}-1}}{n} = \infty$ at $\epsilon = 0$ for $n \geq 2$. For example, take $n = 16$ and $\epsilon = 10^{-16}$. Then for each eigenvalue $|\lambda| = .1$. $\diamond$

So we expect a large condition number if an eigenvalue is "close to multiple"; i.e., there is a small δA such that $A+\delta A$ has exactly a multiple eigenvalue. Having an infinite condition number does not mean that they cannot be computed with any correct digits, however.

PROPOSITION 4.4. *Eigenvalues of A are continuous functions of A, even if they are not differentiable.*

Proof. It suffices to prove the continuity of roots of polynomials, since the coefficients of the characteristic polynomial are continuous (in fact polynomial) functions of the matrix entries. We use the argument principle from complex analysis [2]: the number of roots of a polynomial p inside a simple closed curve γ is $\frac{1}{2\pi i}\oint_\gamma \frac{p'(z)}{p(z)}dz$. If p is changed just a little, $\frac{p'(z)}{p(z)}$ is changed just a little, so $\frac{1}{2\pi i}\oint_\gamma \frac{p'(z)}{p(z)}dz$ is changed just a little. But since it is an integer, it must be constant, so the number of roots inside the curve γ is constant. This means that the roots cannot pass outside the curve γ (no matter how small γ is, provided that we perturb p by little enough), so the roots must be continuous. $\square$

In what follows, we will concentrate on computing the condition number of a simple eigenvalue. If λ is a simple eigenvalue of A and δA is small, then we can identify an eigenvalue $\lambda + \delta\lambda$ of $A + \delta A$ "corresponding to" λ: it is the closest one to λ. We can easily compute the condition number of a simple eigenvalue.

THEOREM 4.4. *Let λ be a simple eigenvalue of A with right eigenvector x and left eigenvector y, normalized so that $\|x\|_2 = \|y\|_2 = 1$. Let $\lambda + \delta\lambda$ be the corresponding eigenvalue of $A + \delta A$. Then*

$$\begin{aligned} \delta\lambda &= \frac{y^*\delta A x}{y^*x} + O(\|\delta A\|^2) \text{ or} \\ |\delta\lambda| &\leq \frac{\|\delta A\|}{|y^*x|} + O(\|\delta A\|^2) = \sec\Theta(y,x)\|\delta A\| + O(\|\delta A\|^2), \end{aligned}$$

where $\Theta(y,x)$ is the acute angle between y and x. In other words, $\sec\Theta(y,x) = 1/|y^*x|$ is the condition number *of the eigenvalue λ.*

Proof. Subtract $Ax = \lambda x$ from $(A + \delta A)(x + \delta x) = (\lambda + \delta\lambda)(x + \delta x)$ to get

$$A\delta x + \delta A x + \delta A\delta x = \lambda\delta x + \delta\lambda x + \delta\lambda\delta x.$$

Ignore the second-order terms (those with two "δ terms" as factors: $\delta A\delta x$ and $\delta\lambda\delta x$) and multiply by y^* to get $y^*A\delta x + y^*\delta A x = y^*\lambda\delta x + y^*\delta\lambda x$.

Now $y^*A\delta x$ cancels $y^*\lambda\delta x$, so we can solve for $\delta\lambda = (y^*\delta Ax)/(y^*x)$ as desired. □

Note that a Jordan block has right and left eigenvectors e_1 and e_n, respectively, so the condition number of its eigenvalue is $1/|e_n^*e_1| = 1/0 = \infty$, which agrees with our earlier analysis.

At the other extreme, in the important special case of symmetric matrices, the condition number is 1, so the eigenvalues are always accurately determined by the data.

COROLLARY 4.1. *Let A be symmetric (or normal: $AA^* = A^*A$). Then $|\delta\lambda| \leq \|\delta A\| + O(\|\delta A\|^2)$.*

Proof. If A is symmetric or normal, then its eigenvectors are all orthogonal, i.e., $Q^*AQ = \Lambda$ with $QQ^* = I$. So the right eigenvectors x (columns of Q) and left eigenvectors y (conjugate transposes of the rows of Q^*) are identical, and $1/|y^*x| = 1$. □

To see a variety of numerical examples, run the Matlab code referred to in Question 4.14.

Later, in Theorem 5.1, we will prove that in fact $|\delta\lambda| \leq \|\delta A\|_2$ if $\delta A = \delta A^T$, no matter how large $\|\delta A\|_2$ is.

Theorem 4.4 is useful only for sufficiently small $\|\delta A\|$. We can remove the $O(\|\delta A\|^2)$ term and so get a simple theorem true for any size perturbation $\|\delta A\|$, at the cost of increasing the condition number by a factor of n.

THEOREM 4.5. Bauer–Fike. *Let A have all simple eigenvalues (i.e., be diagonalizable). Call them λ_i, with right and left eigenvectors x_i and y_i, normalized so $\|x_i\|_2 = \|y_i\|_2 = 1$. Then the eigenvalues of $A + \delta A$ lie in disks B_i, where B_i has center λ_i and radius $n\frac{\|\delta A\|_2}{|y_i^*x_i|}$.*

Our proof will use Gershgorin's theorem (Theorem 2.9), which we repeat here.

GERSHGORIN'S THEOREM. *Let B be an arbitrary matrix. Then the eigenvalues λ of B are located in the union of the n disks defined by $|\lambda - b_{ii}| \leq \sum_{j\neq i} |b_{ij}|$ for $i = 1$ to n.*

We will also need two simple lemmas.

LEMMA 4.1. *Let $S = [x_1, \ldots, x_n]$, the nonsingular matrix of right eigenvectors. Then*

$$S^{-1} = \begin{bmatrix} y_1^*/y_1^*x_1 \\ y_2^*/y_2^*x_2 \\ \vdots \\ y_n^*/y_n^*x_n \end{bmatrix}.$$

Proof of Lemma. We know that $AS = S\Lambda$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, since the columns x_i of S are eigenvectors. This is equivalent to $S^{-1}A = \Lambda S^{-1}$, so the rows of S^{-1} are conjugate transposes of the left eigenvectors y_i. So

$$S^{-1} = \begin{bmatrix} y_1^* \cdot c_1 \\ \vdots \\ y_n^* \cdot c_n \end{bmatrix}$$

for some constants c_i. But $I = S^{-1}S$, so $1 = (S^{-1}S)_{ii} = y_i^* x_i \cdot c_i$, and $c_i = \frac{1}{y_i^* x_i}$ as desired. □

LEMMA 4.2. *If each column of (any matrix) S has two-norm equal to* 1, $\|S\|_2 \leq \sqrt{n}$. *Similarly, if each row of a matrix has two-norm equal to* 1, *its two-norm is at most* $\sqrt{n}$.

Proof of Lemma. $\|S\|_2 = \|S^T\|_2 = \max_{\|x\|_2=1} \|S^T x\|_2$. Each component of $S^T x$ is bounded by 1 by the Cauchy–Schwartz inequality, so $\|S^T x\|_2 \leq \left\|[1, \ldots, 1]^T\right\|_2 = \sqrt{n}$. □

Proof of the Bauer–Fike theorem. We will apply Gershgorin's theorem to $S^{-1}(A + \delta A)S = \Lambda + F$, where $\Lambda = S^{-1}AS = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ and $F = S^{-1}\delta AS$. The idea is to show that the eigenvalues of $A + \delta A$ lie in balls centered at the λ_i with the given radii. To do this, we take the disks containing the eigenvalues of $\Lambda + F$ that are defined by Gershgorin's theorem,

$$|\lambda - (\lambda_i + f_{ii})| \leq \sum_{j \neq i} |f_{ij}|,$$

and enlarge them slightly to get the disks

$$\begin{aligned} |\lambda - \lambda_i| &\leq \sum_j |f_{ij}| \\ &\leq n^{1/2} \cdot \left(\sum_j |f_{ij}|^2 \right)^{1/2} \quad \text{by Cauchy–Schwarz} \\ &= n^{1/2} \cdot \|F(i,:)\|_2. \end{aligned} \tag{4.5}$$

Now we need to bound the two-norm of the ith row $F(i,:)$ of $F = S^{-1}\delta AS$:

$$\begin{aligned} \|F(i,:)\|_2 &= \|(S^{-1}\delta AS)(i,:)\|_2 \\ &\leq \|(S^{-1})(i,:)\|_2 \cdot \|\delta A\|_2 \cdot \|S\|_2 \quad \text{by Lemma 1.7} \\ &\leq \frac{n^{1/2}}{|y_i^* x_i|} \cdot \|\delta A\|_2 \quad \text{by Lemmas 4.1 and 4.2.} \end{aligned}$$

Combined with equation (4.5), this proves the theorem. □

We do not want to leave the impression that multiple eigenvalues cannot be computed with any accuracy at all just because they have infinite condition numbers. Indeed, we expect to get a *fraction* of the digits correct rather than lose a fixed number of digits. To illustrate, consider the 2-by-2 matrix with a double eigenvalue at 1: $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. If we perturb the (2,1) entry (the most sensitive) from 0 to machine epsilon ε, the eigenvalues change from 1 to $1 \pm \sqrt{\varepsilon}$. In other words the computed eigenvalues agree with the true eigenvalue to half precision. More generally, with a triple root, we expect to get about one third of the digits correct, and so on for higher multiplicities. See also Question 1.20.

We now turn to a geometric property of the condition number shared by other problems. Recall the property of the condition number $\|A\| \cdot \|A^{-1}\|$ for matrix inversion: its reciprocal measured the distance to nearest singular matrix, i.e., matrix with an infinite condition number (see Theorem 2.1). An analogous fact is true about eigenvalues. Since multiple eigenvalues have infinite condition numbers, the set of matrices with multiple eigenvalues plays the same role for computing eigenvalues as the singular matrices did for matrix inversion, where being "close to singular" implied ill-conditioning.

THEOREM 4.6. *Let λ be a simple eigenvalue of A, with unit right and left eigenvectors x and y and condition number $c = 1/|y^*x|$. Then there is a δA such that $A + \delta A$ has a multiple eigenvalue at λ, and*

$$\frac{\|\delta A\|_2}{\|A\|_2} \leq \frac{1}{\sqrt{c^2 - 1}}.$$

When $c \gg 1$, i.e., the eigenvalue is ill-conditioned, then the upper bound on the distance is $1/\sqrt{c^2 - 1} \approx 1/c$, the reciprocal of the condition number.

Proof. First we show that we can assume without loss of generality that A is upper triangular (in Schur form), with $a_{11} = \lambda$. This is because putting A in Schur form is equivalent to replacing A by $T = Q^*AQ$, where Q is unitary. If x and y are eigenvectors of A, then Q^*x and Q^*y are eigenvectors of T. Since $(Q^*y)^*(Q^*x) = y^*QQ^*x = y^*x$, changing to Schur form does not change the condition number of λ. (Another way to say this is that the condition number is the secant of the angle $\Theta(x, y)$ between x and y, and changing x to Q^*x and y to Q^*y just rotates x and y the same way without changing the angle between them.)

So without loss of generality we can assume that $A = \begin{bmatrix} \lambda & A_{12} \\ 0 & A_{22} \end{bmatrix}$. Then $x = e_1$ and y is parallel to $\tilde{y} = [1, A_{12}(\lambda I - A_{22})^{-1}]^*$, or $y = \tilde{y}/\|\tilde{y}\|_2$. Thus

$$c = \frac{1}{|y^*x|} = \frac{\|\tilde{y}\|_2}{|\tilde{y}^*x|} = \|\tilde{y}\|_2 = (1 + \|A_{12}(\lambda I - A_{22})^{-1}\|_2^2)^{1/2}$$

or

$$\begin{aligned} \sqrt{c^2 - 1} &= \|A_{12}(\lambda I - A_{22})^{-1}\|_2 \leq \|A_{12}\|_2 \cdot \|(\lambda I - A_{22})^{-1}\|_2 \\ &\leq \frac{\|A\|_2}{\sigma_{\min}(\lambda I - A_{22})}. \end{aligned}$$

By definition of the smallest singular value, there is a δA_{22} where $\|\delta A_{22}\|_2 = \sigma_{\min}(\lambda I - A_{22})$ such that $A_{22} + \delta A_{22} - \lambda I$ is singular; i.e., λ is an eigenvalue of $A_{22} + \delta A_{22}$. Thus $\begin{bmatrix} \lambda & A_{12} \\ 0 & A_{22}+\delta A_{22} \end{bmatrix}$ has a double eigenvalue at λ, where

$$\|\delta A_{22}\|_2 = \sigma_{\min}(\lambda I - A_{22}) \leq \frac{\|A\|_2}{\sqrt{c^2 - 1}}$$

as desired. □

Finally, we relate the condition numbers of the eigenvalues to the smallest possible condition number $\|S\| \cdot \|S^{-1}\|$ of any similarity S that diagonalizes A: $S^{-1}AS = \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$. The theorem says that if any eigenvalue has a large condition number, then S has to have an approximately equally large condition number. In other words, the condition numbers for finding the (worst) eigenvalue and for reducing the matrix to diagonal form are nearly the same.

THEOREM 4.7. *Let A be diagonalizable with eigenvalues λ_i and right and left eigenvectors x_i and y_i, respectively, normalized so that $\|x_i\|_2 = \|y_i\|_2 = 1$. Let us suppose that S satisfies $S^{-1}AS = \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$. Then $\|S\|_2 \cdot \|S^{-1}\|_2 \geq \max_i 1/|y_i^* x_i|$. If we choose $S = [x_1, \ldots, x_n]$, then $\|S\|_2 \cdot \|S^{-1}\|_2 \leq n \cdot \max_i 1/|y_i^* x_i|$; i.e., the condition number of S is within a factor of n of its smallest value.*

For a proof, see [69].

For an overview of condition numbers for the eigenproblem, including eigenvectors, invariant subspaces, and the eigenvalues corresponding to an invariant subspace, see chapter 4 of the LAPACK manual [10], as well as [161, 237]. Algorithms for computing these condition numbers are available in subroutines `strsna` and `strsen` of LAPACK or by calling the driver routines `sgeevx` and `sgeesx`.

4.4. Algorithms for the Nonsymmetric Eigenproblem

We will build up to our ultimate algorithm, the shifted Hessenberg QR algorithm, by starting with simpler ones. For simplicity of exposition, we assume A is real.

Our first and simplest algorithm is the *power method* (section 4.4.1), which can find only the largest eigenvalue of A in absolute value and the corresponding eigenvector. To find the other eigenvalues and eigenvectors, we apply the power method to $(A-\sigma I)^{-1}$ for some *shift* σ, an algorithm called *inverse iteration* (section 4.4.2); note that the largest eigenvalue of $(A-\sigma I)^{-1}$ is $1/(\lambda_i - \sigma)$, where λ_i is the closest eigenvalue to σ, so we can choose which eigenvalues to find by choosing σ. Our next improvement to the power method lets us compute an entire invariant subspace at a time rather than just a single eigenvector;

we call this *orthogonal iteration* (section 4.4.3). Finally, we reorganize orthogonal iteration to make it convenient to apply to $(A - \sigma I)^{-1}$ instead of A; this is called QR iteration (section 4.4.4).

Mathematically speaking, QR iteration (with a shift σ) is our ultimate algorithm. But several problems remain to be solved to make it sufficiently fast and reliable for practical use (section 4.4.5). Section 4.4.6 discusses the first transformation designed to make QR iteration fast: reducing A from dense to *upper Hessenberg form* (nonzero only on and above the first subdiagonal). Subsequent sections describe how to implement QR iteration efficiently on upper Hessenberg matrices. (Section 4.4.7 shows how upper Hessenberg form simplifies in the cases of the symmetric eigenvalue problem and SVD.)

4.4.1. Power Method

ALGORITHM 4.1. *Power method: Given x_0, we iterate*

$i = 0$
repeat
  $y_{i+1} = Ax_i$
  $x_{i+1} = y_{i+1}/\|y_{i+1}\|_2$ *(approximate eigenvector)*
  $\tilde{\lambda}_{i+1} = x_{i+1}^T A x_{i+1}$ *(approximate eigenvalue)*
  $i = i + 1$
until convergence

Let us first apply this algorithm in the very simple case when $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, with $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. In this case the eigenvectors are just the columns e_i of the identity matrix. Note that x_i can also be written $x_i = A^i x_0/\|A^i x_0\|_2$, since the factors $1/\|y_{i+1}\|_2$ only scale x_{i+1} to be a unit vector and do not change its direction. Then we get

$$A^i x_0 \equiv A^i \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix} = \begin{bmatrix} \xi_1 \lambda_1^i \\ \xi_2 \lambda_2^i \\ \vdots \\ \xi_n \lambda_n^i \end{bmatrix} = \xi_1 \lambda_1^i \begin{bmatrix} 1 \\ \frac{\xi_2}{\xi_1}\left(\frac{\lambda_2}{\lambda_1}\right)^i \\ \vdots \\ \frac{\xi_n}{\xi_1}\left(\frac{\lambda_n}{\lambda_1}\right)^i \end{bmatrix},$$

where we have assumed $\xi_1 \neq 0$. Since all the fractions λ_j/λ_1 are less than 1 in absolute value, $A^i x_0$ becomes more and more nearly parallel to e_1, so $x_i = A^i x_0/\|A^i x_0\|_2$ becomes closer and closer to $\pm e_1$, the eigenvector corresponding to the largest eigenvalue λ_1. The rate of convergence depends on how much smaller than 1 the ratios $|\lambda_2/\lambda_1| \geq \cdots \geq |\lambda_n/\lambda_1|$ are, the smaller the faster. Since x_i converges to $\pm e_1$, $\tilde{\lambda}_i = x_i^T A x_i$ converges to λ_1, the largest eigenvalue.

In showing that the power method converges, we have made several assumptions, most notably that A is diagonal. To analyze a more general case, we now assume that $A = S\Lambda S^{-1}$ is diagonalizable, with $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

and the eigenvalues sorted so that $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. Write $S = [s_1, \ldots, s_n]$, where the columns s_i are the corresponding eigenvectors and also satisfy $\|s_i\|_2 = 1$; in the last paragraph we had $S = I$. This lets us write $x_0 = S(S^{-1}x_0) \equiv S([\xi_1, \ldots, \xi_n]^T)$. Also, since $A = S\Lambda S^{-1}$, we can write

$$A^i = \underbrace{(S\Lambda S^{-1}) \cdots (S\Lambda S^{-1})}_{i \text{ times}} = S\Lambda^i S^{-1}$$

since all the $S^{-1} \cdot S$ pairs cancel. This finally lets us write

$$A^i x_0 = (S\Lambda^i S^{-1})S \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix} = S \begin{bmatrix} \xi_1 \lambda_1^i \\ \xi_2 \lambda_2^i \\ \vdots \\ \xi_n \lambda_n^i \end{bmatrix} = \xi_1 \lambda_1^i S \begin{bmatrix} 1 \\ \frac{\xi_2}{\xi_1}\left(\frac{\lambda_2}{\lambda_1}\right)^i \\ \vdots \\ \frac{\xi_n}{\xi_1}\left(\frac{\lambda_n}{\lambda_1}\right)^i \end{bmatrix}.$$

As before, the vector in brackets converges to e_1, so $A^i x_0$ gets closer and closer to a multiple of $Se_1 = s_1$, the eigenvector corresponding to λ_1. Therefore, $\tilde{\lambda}_i = x_i^T A x_i$ converges to $s_1^T A s_1 = s_1^T \lambda_1 s_1 = \lambda_1$.

A minor drawback of this method is the assumption that $\xi_1 \neq 0$, i.e., that x_0 is not the invariant subspace $\text{span}\{s_2, \ldots, s_n\}$; this is true with very high probability if x_0 is chosen at random. A major drawback is that it converges to the eigenvalue/eigenvector pair only for the eigenvalue of largest absolute magnitude, and its convergence rate depends on $|\lambda_2/\lambda_1|$, a quantity which may be close to 1 and thus cause very slow convergence. Indeed, if A is real and the largest eigenvalue is complex, there are two complex conjugate eigenvalues of largest absolute value $|\lambda_1| = |\lambda_2|$, and so the above analysis does not work at all. In the extreme case of an orthogonal matrix, *all* the eigenvalues have the same absolute value, namely, 1.

To plot the convergence of the power method, see HOMEPAGE/Matlab/powerplot.m.

4.4.2. Inverse Iteration

We will overcome the drawbacks of the power method just described by applying the power method to $(A - \sigma I)^{-1}$ instead of A, where σ is called a *shift*. This will let us converge to the eigenvalue closest to σ, rather than just λ_1. This method is called inverse iteration or the inverse power method.

ALGORITHM 4.2. *Inverse iteration: Given x_0, we iterate*

$i = 0$
repeat
  $y_{i+1} = (A - \sigma I)^{-1} x_i$
  $x_{i+1} = y_{i+1}/\|y_{i+1}\|_2$ *(approximate eigenvector)*

$\tilde{\lambda}_{i+1} = x_{i+1}^T A x_{i+1}$ *(approximate eigenvalue)*
$i = i + 1$
until convergence

To analyze the convergence, note that $A = S\Lambda S^{-1}$ implies $A - \sigma I = S(\Lambda - \sigma I)S^{-1}$ and so $(A - \sigma I)^{-1} = S(\Lambda - \sigma I)^{-1}S^{-1}$. Thus $(A - \sigma I)^{-1}$ has the same eigenvectors s_i as A with corresponding eigenvalues $((\Lambda - \sigma I)^{-1})_{jj} = (\lambda_j - \sigma)^{-1}$. The same analysis as before tells us to expect x_i to converge to the eigenvector corresponding to the largest eigenvalue in absolute value. More specifically, assume that $|\lambda_k - \sigma|$ is smaller than all the other $|\lambda_i - \sigma|$ so that $(\lambda_k - \sigma)^{-1}$ is the largest eigenvalue in absolute value. Also, write $x_0 = S[\xi_1, \ldots, \xi_n]^T$ as before, and assume $\xi_k \neq 0$. Then

$$(A - \sigma I)^{-i} x_0 = (S(\Lambda - \sigma I)^{-i} S^{-1}) S \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix} = S \begin{bmatrix} \xi_1 (\lambda_1 - \sigma)^{-i} \\ \vdots \\ \xi_n (\lambda_n - \sigma)^{-i} \end{bmatrix}$$

$$= \xi_k (\lambda_k - \sigma)^{-i} S \begin{bmatrix} \frac{\xi_1}{\xi_k} \left(\frac{\lambda_k - \sigma}{\lambda_1 - \sigma} \right)^i \\ \vdots \\ 1 \\ \vdots \\ \frac{\xi_n}{\xi_k} \left(\frac{\lambda_k - \sigma}{\lambda_n - \sigma} \right)^i \end{bmatrix},$$

where the 1 is in entry k. Since all the fractions $(\lambda_k - \sigma)/(\lambda_i - \sigma)$ are less than one in absolute value, the vector in brackets approaches e_k, so $(A - \sigma I)^{-i} x_0$ gets closer and closer to a multiple of $Se_k = s_k$, the eigenvector corresponding to λ_k. As before, $\tilde{\lambda}_i = x_i^T A x_i$ also converges to λ_k.

The advantage of inverse iteration over the power method is the ability to converge to any desired eigenvalue (the one nearest the shift σ). By choosing σ very close to a desired eigenvalue, we can converge very quickly and thus not be as limited by the proximity of nearby eigenvalues as is the original power method. The method is particularly effective when we have a good approximation to an eigenvalue and want only its corresponding eigenvector (for example, see section 5.3.4). Later we will explain how to choose such a σ without knowing the eigenvalues, which is what we are trying to compute in the first place!

4.4.3. Orthogonal Iteration

Our next improvement will permit us to converge to a ($p > 1$)-dimensional invariant subspace, rather than one eigenvector at a time. It is called *orthogonal iteration* (and sometimes *subspace iteration* or *simultaneous iteration*).

ALGORITHM 4.3. *Orthogonal iteration: Let Z_0 be an $n \times p$ orthogonal matrix. Then we iterate*

$i = 0$
repeat
  $Y_{i+1} = AZ_i$
  Factor $Y_{i+1} = Z_{i+1}R_{i+1}$ *using Algorithm 3.2 (QR decomposition)*
    (Z_{i+1} spans an approximate invariant subspace)
  $i = i + 1$
until convergence

Here is an informal analysis of this method. Assume $|\lambda_p| > |\lambda_{p+1}|$. If $p = 1$, this method and its analysis are identical to the power method. When $p > 1$, we write $\text{span}\{Z_{i+1}\} = \text{span}\{Y_{i+1}\} = \text{span}\{AZ_i\}$, so $\text{span}\{Z_i\} = \text{span}\{A^iZ_0\} = \text{span}\{S\Lambda^iS^{-1}Z_0\}$. Note that

$$\begin{aligned} S\Lambda^iS^{-1}Z_0 &= S\,\text{diag}(\lambda_1^i,\ldots,\lambda_n^i)S^{-1}Z_0 \\ &= \lambda_p^i S \begin{bmatrix} (\lambda_1/\lambda_p)^i & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & (\lambda_n/\lambda_p)^i \end{bmatrix} S^{-1}Z_0. \end{aligned}$$

Since $|\frac{\lambda_j}{\lambda_p}| \geq 1$ if $j \leq p$, and $|\frac{\lambda_j}{\lambda_p}| < 1$ if $j > p$, we get

$$\begin{bmatrix} (\lambda_1/\lambda_p)^i & & \\ & \ddots & \\ & & (\lambda_n/\lambda_p)^i \end{bmatrix} S^{-1}Z_0 = \begin{bmatrix} V_i^{p\times p} \\ W_i^{(n-p)\times p} \end{bmatrix},$$

where W_i approaches zero like $(\lambda_{p+1}/\lambda_p)^i$, and V_i does not approach zero. Indeed, if V_0 has full rank (a generalization of the assumption in section 4.4.1 that $\xi_1 \neq 0$), then V_i will have full rank too. Write the matrix of eigenvectors $S = [s_1,\ldots,s_n] \equiv [S_p^{n\times p}, \hat{S}_p^{n\times(n-p)}]$, i.e., $S_p = [s_1,\ldots,s_p]$. Then $S\Lambda^iS^{-1}Z_0 = \lambda_p^iS\begin{bmatrix} V_i \\ W_i \end{bmatrix} = \lambda_p^i(S_pV_i + \hat{S}_pW_i)$. Thus

$$\text{span}(Z_i) = \text{span}(S\Lambda^iS^{-1}Z_0) = \text{span}(S_pV_i + \hat{S}_pW_i) \Rightarrow \text{span}(S_pX_i)$$

converges to $\text{span}(S_pV_i) = \text{span}(S_p)$, the invariant subspace spanned by the first p eigenvectors, as desired.

The use of the QR decomposition keeps the vectors spanning $\text{span}\{A^iZ_0\}$ of full rank despite roundoff.

Note that if we follow only the first $\tilde{p} < p$ columns of Z_i through the iterations of the algorithm, they are *identical* to the columns that we would compute if we had started with only the first $\tilde{p}$ columns of Z_0 instead of p columns. In other words, orthogonal iteration is effectively running the algorithm for $\tilde{p} = 1, 2, \ldots, p$ all at the same time. So if *all* the eigenvalues have distinct absolute values, the same convergence analysis as before implies that the first $\tilde{p} \leq p$ columns of Z_i converge to $\text{span}\{s_1, \ldots, s_{\tilde{p}}\}$ for any $\tilde{p} \leq p$.

Thus, we can let $p = n$ and $Z_0 = I$ in the orthogonal iteration algorithm. The next theorem shows that under certain assumptions, we can use orthogonal iteration to compute the Schur form of A.

THEOREM 4.8. *Consider running orthogonal iteration on matrix A with $p = n$ and $Z_0 = I$. If all the eigenvalues of A have distinct absolute values and if all the principal submatrices $S(1 : j, 1 : j)$ have full rank, then $A_i \equiv Z_i^T A Z_i$ converges to the Schur form of A, i.e., an upper triangular matrix with the eigenvalues on the diagonal. The eigenvalues will appear in decreasing order of absolute value.*

Sketch of Proof. The assumption about nonsingularity of $S(1 : j, 1 : j)$ for all j implies that X_0 is nonsingular, as required by the earlier analysis. Geometrically, this means that no vector in the invariant subspace $\text{span}\{s_i, \ldots, s_j\}$ is orthogonal to $\text{span}\{e_i, \ldots, e_j\}$, the space spanned by the first j columns of $Z_0 I$. First note that Z_i is a square orthogonal matrix, so A and $A_i = Z_i^T A Z_i$ are similar. Write $Z_i = [Z_{1i}, Z_{2i}]$, where Z_{1i} has p columns, so

$$A_i = Z_i^T A Z_i = \begin{bmatrix} Z_{1i}^T A Z_{1i} & Z_{1i}^T A Z_{2i} \\ Z_{2i}^T A Z_{1i} & Z_{2i}^T A Z_{2i} \end{bmatrix}.$$

Since $\text{span}\{Z_{1i}\}$ converges to an invariant subspace of A, $\text{span}\{AZ_{1i}\}$ converges to the same subspace, so $Z_{2i}^T A Z_{1i}$ converges to $Z_{2i}^T Z_{1i} = 0$. Since this is true for all $p < n$, every subdiagonal entry of A_i converges to zero, so A_i converges to upper triangular form, i.e., Schur form. □

In fact, this proof shows that the submatrix $Z_{2i}^T A Z_{1i} = A_i(p + 1 : n, 1 : p)$ should converge to zero like $|\lambda_{p+1}/\lambda_p|^i$. Thus, λ_p should appear as the (p, p) entry of A_i and converge like $\max(|\lambda_{p+1}/\lambda_p|^i, |\lambda_p/\lambda_{p-1}|^i)$.

EXAMPLE 4.5. The convergence behavior of orthogonal iteration is illustrated by the following numerical experiment, where we took $\Lambda = \text{diag}(1, 2, 6, 30)$ and a random S (with condition number about 20), formed $A = S \cdot \Lambda \cdot S^{-1}$, and ran orthogonal iteration on A with $p = 4$ for 19 iterations. Figures 4.3 and 4.4 show the convergence of the algorithm. Figure 4.3 plots the actual errors $|A_i(p, p) - \lambda_p|$ in the computed eigenvalues as solid lines and the approximations $\max(|\lambda_{p+1}/\lambda_p|^i, |\lambda_p/\lambda_{p-1}|^i)$ as dotted lines. Since the graphs are (essentially) straight lines with the same slope on a semilog scale, this means that they are both graphs of functions of the form $y = c \cdot r^i$, where c and r are constants and r (the slope) is the same for both, as we predicted above.

Similarly, Figure 4.4 plots the actual values $\|A_i(p+1:n, 1:p)\|_2$ as solid lines and the approximations $|\lambda_{p+1}/\lambda_p|^i$ as dotted lines; they also match well. Here are A_0 and A_{19} for comparison:

$$A = A_0 = \begin{bmatrix} 3.5488 & 15.593 & 8.5775 & -4.0123 \\ 2.3595 & 24.526 & 14.596 & -5.8157 \\ 8.9953 \cdot 10^{-2} & 27.599 & 21.483 & -5.8415 \\ 1.9227 & 55.667 & 39.717 & -10.558 \end{bmatrix},$$

$$A_{19} = \begin{bmatrix} 30.000 & -32.557 & -70.844 & 14.984 \\ 6.7607 \cdot 10^{-13} & 6.0000 & 1.8143 & -.55754 \\ 1.5452 \cdot 10^{-23} & 1.1086 \cdot 10^{-9} & 2.0000 & -.25894 \\ 7.3360 \cdot 10^{-29} & 3.3769 \cdot 10^{-15} & 4.9533 \cdot 10^{-6} & 1.0000 \end{bmatrix}.$$

See HOMEPAGE/Matlab/qriter.m for Matlab software to run this and similar examples. ◇

EXAMPLE 4.6. To see why the assumption in Theorem 4.8 about nonsingularity of $S(1:j, 1:j)$ is necessary, suppose that A is diagonal with the eigenvalues *not* in decreasing order on the diagonal. Then orthogonal iteration yields $Z_i = \text{diag}(\pm 1)$ (a diagonal matrix with diagonal entries ± 1) and $A_i = A$ for all i, so the eigenvalues do not move into decreasing order. To see why the assumption that the eigenvalues have distinct absolute values is necessary, suppose that A is orthogonal, so all its eigenvalues have absolute value 1. Again, the algorithm leaves A_i essentially unchanged. (The rows and columns may be multiplied by -1.)

4.4.4. QR Iteration

Our next goal is to reorganize orthogonal iteration to incorporate shifting and inverting, as in section 4.4.2. This will make it more efficient and eliminate the assumption that eigenvalues differ in magnitude, which was needed in Theorem 4.8 to prove convergence.

ALGORITHM 4.4. *QR iteration: Given A_0, we iterate*

$i = 0$
repeat
  Factor $A_i = Q_i R_i$ *(the QR decomposition)*
  $A_{i+1} = R_i Q_i$
  $i = i + 1$
until convergence

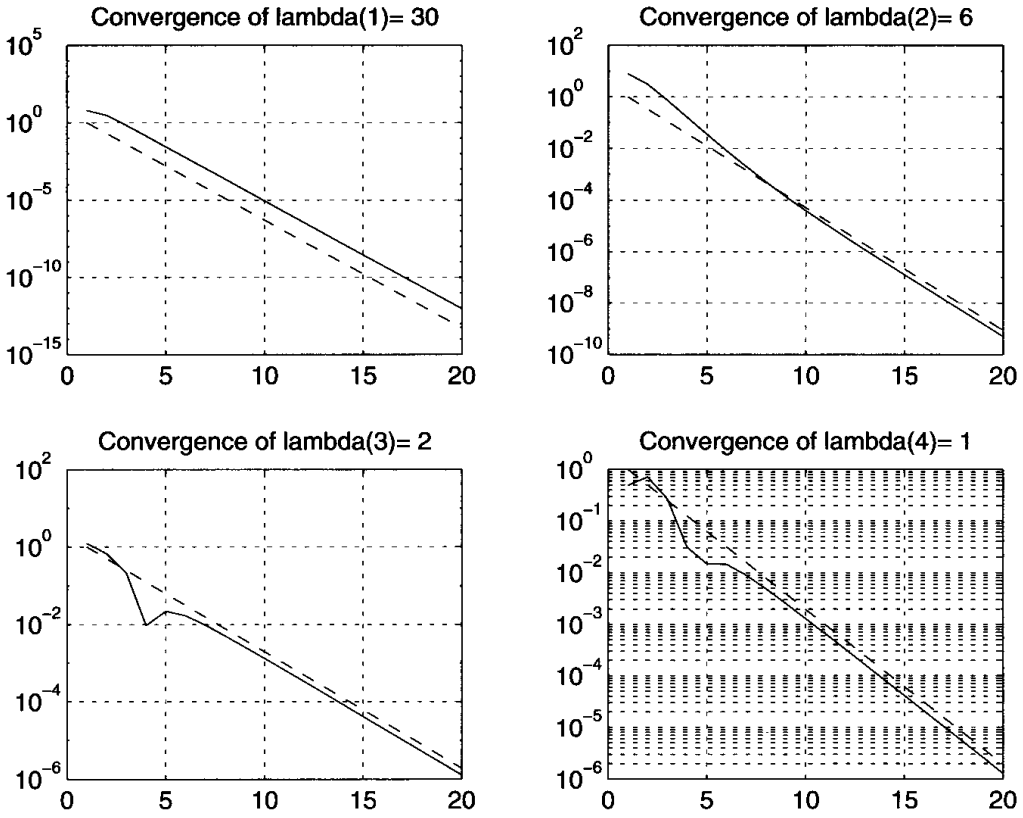


Fig. 4.3. *Convergence of diagonal entries during orthogonal iteration.*

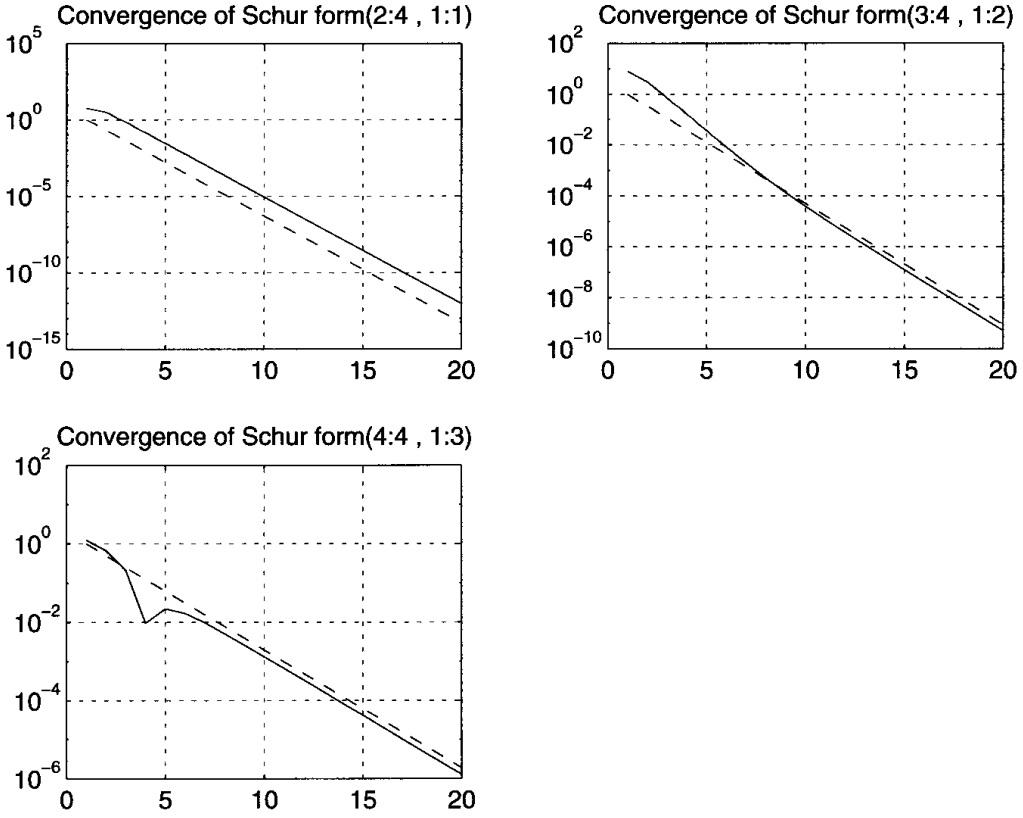


Fig. 4.4. *Convergence to Schur form during orthogonal iteration.*

Since $A_{i+1} = R_i Q_i = Q_i^T (Q_i R_i) Q_i = Q_i^T A_i Q_i$, A_{i+1} and A_i are orthogonally similar.

We claim that the A_i computed by QR iteration is identical to the matrix $Z_i^T A Z_i$ implicitly computed by orthogonal iteration.

LEMMA 4.3. *Let A_i be the matrix computed by Algorithm* 4.4. *Then $A_i = Z_i^T A Z_i$, where Z_i is the matrix computed from orthogonal iteration (Algorithm* 4.3*) starting with $Z_0 = I$. Thus A_i converges to Schur form if all the eigenvalues have different absolute values.*

Proof. We use induction. Assume $A_i = Z_i^T A Z_i$. From Algorithm 4.3, we can write $AZ_i = Z_{i+1} R_{i+1}$, where Z_{i+1} is orthogonal and R_{i+1} is upper triangular. Then $Z_i^T A Z_i = Z_i^T (Z_{i+1} R_{i+1})$ is the product of an orthogonal matrix $Q = Z_i^T Z_{i+1}$ and an upper triangular matrix $R = R_{i+1} = Z_{i+1}^T A Z_i$; this must be the QR decomposition $A_i = QR$, since the QR decomposition is unique (except for possibly multiplying each column of Q and row of R by -1). Then

$$Z_{i+1}^T A Z_{i+1} = (Z_{i+1}^T A Z_i)(Z_i^T Z_{i+1}) = R_{i+1}(Z_i^T Z_{i+1}) = RQ.$$

This is precisely how the QR iteration maps A_i to A_{i+1}, so $Z_{i+1}^T A Z_{i+1} = A_{i+1}$ as desired. □

To see a variety of numerical examples illustrating the convergence of QR iteration, run the Matlab code referred to in Question 4.15.

From earlier analysis, we know that the convergence rate depends on the ratios of eigenvalues. To speed convergence, we use shifting and inverting.

ALGORITHM 4.5. *QR iteration with a shift: Given A_0, we iterate*

$i = 0$
repeat
  Choose a shift σ_i near an eigenvalue of A
  Factor $A_i - \sigma_i I = Q_i R_i$ (QR decomposition)
  $A_{i+1} = R_i Q_i + \sigma_i I$
  $i = i + 1$
until convergence

LEMMA 4.4. *A_i and A_{i+1} are orthogonally similar.*

Proof. $A_{i+1} = R_i Q_i + \sigma_i I = Q_i^T Q_i R_i Q_i + \sigma_i Q_i^T Q_i = Q_i^T (Q_i R_i + \sigma_i I) Q_i = Q_i^T A_i Q_i$. □

If R_i is nonsingular, we may also write

$$\begin{aligned} A_{i+1} &= R_i Q_i + \sigma_i I = R_i Q_i R_i R_i^{-1} + \sigma_i R_i R_i^{-1} = R_i (Q_i R_i + \sigma_i I) R_i^{-1} \\ &= R_i A_i R_i^{-1}. \end{aligned}$$

If σ_i is an *exact* eigenvalue of A_i, then we claim that QR iteration converges in one step: since σ_i is an eigenvalue, $A_i - \sigma_i I$ is singular, so R_i is singular, and so some diagonal entry of R_i must be zero. Suppose $R_i(n,n) = 0$. This implies that the last row of $R_i Q_i$ is 0, so the last row of $A_{i+1} = R_i Q_i + \sigma_i I$ equals $\sigma_i e_n^T$, where e_n is the nth column of the n-by-n identity matrix. In other words, the last row of A_{i+1} is zero except for the eigenvalue σ_i appearing in the (n,n) entry. This means that the algorithm has *converged*, because A_{i+1} is block upper triangular, with a trailing 1-by-1 block σ_i; the leading $(n-1)$-by-$(n-1)$ block A' is a new, smaller eigenproblem to which QR iteration can be solved without ever modifying σ_i again: $A_{i+1} = \begin{bmatrix} A' & a \\ 0 & \sigma_i \end{bmatrix}$.

When σ_i is not an exact eigenvalue, then we will accept $A_{i+1}(n,n)$ as having converged when the lower left block $A_{i+1}(n, 1:n-1)$ is small enough. Recall from our earlier analysis that we expect $A_{i+1}(n, 1:n-1)$ to shrink by a factor $|\lambda_k - \sigma_i| / \min_{j \neq k} |\lambda_j - \sigma_i|$, where $|\lambda_k - \sigma_i| = \min_j |\lambda_j - \sigma_i|$. So if σ_i is a very good approximation to eigenvalue λ_k, we expect fast convergence.

Here is another way to see why convergence should be fast, by recognizing that QR iteration is implicitly doing inverse iteration. When σ_i is an exact eigenvalue, the last column $\underline{q}_i$ of Q_i will be a left eigenvector of A_i for eigenvalue σ_i, since $\underline{q}_i^* A_i = \underline{q}_i^* (Q_i R_i + \sigma_i I) = e_n^T R_i + \sigma_i \underline{q}_i^* = \sigma_i \underline{q}_i^*$. When σ_i is close to an eigenvalue, we expect $\underline{q}_i$ to be close to an eigenvector for the following reason: $\underline{q}_i$ is parallel to $((A_i - \sigma_i I)^*)^{-1} e_n$ (we explain why below). In other words $\underline{q}_i$ is the same as would be obtained from inverse iteration on $(A_i - \sigma_i I)^*$ (and so we expect it to be close to a left eigenvector).

Here is the proof that $\underline{q}_i$ is parallel to $((A_i - \sigma_i I)^*)^{-1} e_n$. $A_i - \sigma_i I = Q_i R_i$ implies $(A_i - \sigma_i I) R_i^{-1} = Q_i$. Inverting and taking the conjugate transpose of both sides leave the right-hand side Q_i unchanged and change the left-hand side to $((A_i - \sigma_i I)^*)^{-1} R_i^*$, whose last column is $((A_i - \sigma_i I)^*)^{-1} \cdot [0, \ldots, 0, R_i(n,n)]^T$, which is proportional to the last column of $((A_i - \sigma_i I)^*)^{-1}$.

How do we choose σ_i to be an accurate approximate eigenvalue, when we are trying to compute eigenvalues in the first place? We will say more about this later, but for now note that near convergence to a *real* eigenvalue $A_i(n,n)$ is close to that eigenvalue, so $\sigma_i = A_i(n,n)$ is a good choice of shift. In fact, it yields local *quadratic convergence*, which means that the number of correct digits *doubles* at every step. We explain why quadratic convergence occurs as follows: Suppose at step i that $\|A_i(n, 1:n-1)\| / \|A\| \equiv \eta \ll 1$. If we were to set $A_i(n, 1:n-1)$ to exactly 0, we would make A_i block upper triangular and so perturb a true eigenvalue λ_k to make it equal to $A_i(n,n)$. If this eigenvalue is far from the other eigenvalues, it will not be ill-conditioned, so this perturbation will be $O(\eta \|A\|)$. In other words, $|\lambda_k - A_i(n,n)| = O(\eta \|A\|)$. On the next iteration, if we choose $\sigma_i = A_i(n,n)$, we expect $A_{i+1}(n, 1:n-1)$ to shrink by a factor $|\lambda_k - \sigma_i| / \min_{j \neq k} |\lambda_j - \sigma_i| = O(\eta)$, implying that $\|A_{i+1}(n, 1:n-1)\| = O(\eta^2 \|A\|)$, or $\|A_{i+1}(n, 1:n-1)\| / \|A\| = O(\eta^2)$. Decreasing the error this way from η to $O(\eta^2)$ is quadratic convergence.

EXAMPLE 4.7. Here are some shifted QR iterations starting with the same 4-by-4 matrix A_0 as in Example 4.5, with shift $\sigma_i = A_i(4,4)$. The convergence is a bit erratic at first but eventually becomes quadratic near the end, with $\|A_i(4,1:3)\| \approx |A_i(4,3)|$ approximately squaring at each of the last three steps. Also, the number of correct digits in $A_i(4,4)$ doubles at the fourth through second-to-last steps.

$A_0(4,:) =$	$+1.9$	$+56.$	$+40.$	-10.558
$A_1(4,:) =$	$-.85$	-4.9	$+2.2 \cdot 10^{-2}$	-6.6068
$A_2(4,:) =$	$+.35$	$+.86$	$+.30$	0.74894
$A_3(4,:) =$	$-1.2 \cdot 10^{-2}$	$-.17$	$-.70$	1.4672
$A_4(4,:) =$	$-1.5 \cdot 10^{-4}$	$-1.8 \cdot 10^{-2}$	$-.38$	1.4045
$A_5(4,:) =$	$-3.0 \cdot 10^{-6}$	$-2.2 \cdot 10^{-3}$	$-.50$	1.1403
$A_6(4,:) =$	$-1.4 \cdot 10^{-8}$	$-6.3 \cdot 10^{-5}$	$-7.8 \cdot 10^{-2}$	1.0272
$A_7(4,:) =$	$-1.4 \cdot 10^{-11}$	$-3.6 \cdot 10^{-7}$	$-2.3 \cdot 10^{-3}$	0.99941
$A_8(4,:) =$	$+2.8 \cdot 10^{-16}$	$+4.2 \cdot 10^{-11}$	$+1.4 \cdot 10^{-6}$	0.9999996468853453
$A_9(4,:) =$	$-3.4 \cdot 10^{-24}$	$-3.0 \cdot 10^{-18}$	$-4.8 \cdot 10^{-13}$	0.9999999999998767
$A_{10}(4,:) =$	$+1.5 \cdot 10^{-38}$	$+7.4 \cdot 10^{-32}$	$+6.0 \cdot 10^{-26}$	1.000000000000001

By the time we reach A_{10}, the rest of the matrix has made a lot of progress toward convergence as well, so later eigenvalues will be computed very quickly, in one or two steps each:

$$A_{10} = \begin{bmatrix} 30.000 & -32.557 & -70.844 & 14.985 \\ 6.1548 \cdot 10^{-6} & 6.0000 & 1.8143 & -.55754 \\ 2.5531 \cdot 10^{-13} & 2.0120 \cdot 10^{-6} & 2.0000 & -.25894 \\ 1.4692 \cdot 10^{-38} & 7.4289 \cdot 10^{-32} & 6.0040 \cdot 10^{-26} & 1.0000 \end{bmatrix}. \quad \diamond$$

4.4.5. Making QR Iteration Practical

Here are some remaining problems we have to solve to make the algorithm more practical:

1. The iteration is too expensive. The QR decomposition costs $O(n^3)$ flops, so if we were lucky enough to do only one iteration per eigenvalue, the cost would be $O(n^4)$. But we seek an algorithm with a total cost of only $O(n^3)$.

2. How shall we choose σ_i to accelerate convergence to a complex eigenvalue? Choosing σ_i complex means all arithmetic has to be complex, increasing the cost by a factor of about 4 when A is real. We seek an algorithm that uses all real arithmetic if A is real and converges to real Schur form.

3. How do we recognize convergence?

The solutions to these problems, which we will describe in more detail later, are as follows:

1. We will initially reduce the matrix to *upper Hessenberg form*; this means that A is zero below the first subdiagonal (i.e., $a_{ij} = 0$ if $i > j + 1$) (see section 4.4.6). Then we will apply a step of QR iteration *implicitly*, i.e., without computing Q or multiplying by it explicitly (see section 4.4.8). This will reduce the cost of one QR iteration from $O(n^3)$ to $O(n^2)$ and the overall cost from $O(n^4)$ to $O(n^3)$ as desired

   When A is symmetric we will reduce it to tridiagonal form instead, reducing the cost of a single QR iteration further to $O(n)$. This is discussed in section 4.4.7 and Chapter 5.

2. Since complex eigenvalues of real matrices occur in complex conjugate pairs, we can shift by σ_i and $\bar{\sigma}_i$ simultaneously; it turns out that this will permit us to maintain real arithmetic (see section 4.4.8). If A is symmetric, all eigenvalues are real, and this is not an issue.

3. Convergence occurs when subdiagonal entries of A_i are "small enough." To help choose a practical threshold, we use the notion of backward stability: Since A_i is related to A by a similarity transformation by an orthogonal matrix, we expect A_i to have roundoff errors of size $O(\varepsilon\|A\|)$ in it anyway. Therefore, any subdiagonal entry of A_i smaller than $O(\varepsilon\|A\|)$ in magnitude may as well be zero, so we set it to zero.[16] When A is upper Hessenberg, setting $a_{p+1,p}$ to zero will make A into a block upper triangular matrix $A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$, where A_{11} is p-by-p and A_{11} and A_{22} are both Hessenberg. Then the eigenvalues of A_{11} and A_{22} may be found independently to get the eigenvalues of A. When all these diagonal blocks are 1-by-1 or 2-by-2, the algorithm has finished.

4.4.6. Hessenberg Reduction

Given a real matrix A, we seek an orthogonal Q so that QAQ^T is upper Hessenberg. The algorithm is a simple variation on the idea used for the QR decomposition.

EXAMPLE 4.8. We illustrate the general pattern of Hessenberg reduction with a 5-by-5 example. Each Q_i below is a 5-by-5 Householder reflection, chosen to zero out entries $i + 2$ through n in column i and leaving entries 1 through i unchanged.

[16] In practice, we use a slightly more stringent condition, replacing $\|A\|$ with the norm of a submatrix of A, to take into account matrices which may be "graded" with large entries in one place and small entries elsewhere. We can also set a subdiagonal entry to zero when the product $a_{p+1,p}a_{p+2,p+1}$ of two adjacent subdiagonal entries is small enough. See the LAPACK routine `slahqr` for details.

1. Choose Q_1 so

$$Q_1 A = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & x & x & x & x \\ o & x & x & x & x \end{bmatrix} \quad \text{and} \quad A_1 \equiv Q_1 A Q_1^T = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & x & x & x & x \\ o & x & x & x & x \end{bmatrix}.$$

Q_1 leaves the first row of $Q_1 A$ unchanged, and Q_1^T leaves the first column of $Q_1 A Q_1^T$ unchanged, including the zeros.

2. Choose Q_2 so

$$Q_2 A_1 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & x & x & x \end{bmatrix} \quad \text{and} \quad A_2 \equiv Q_2 A_1 Q_2^T = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & x & x & x \end{bmatrix}.$$

Q_2 changes only the last three rows of A_1, and Q_2^T leaves the first two columns of $Q_2 A_1 Q_2^T$ unchanged, including the zeros.

3. Choose Q_3 so

$$Q_3 A_2 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix} \quad \text{and} \quad A_3 = Q_3 A_2 Q_3^T = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix},$$

which is upper Hessenberg. Altogether $A_3 = (Q_3 Q_2 Q_1) \cdot A (Q_3 Q_2 Q_1)^T \equiv Q A Q^T$. $\diamond$

The general algorithm for Hessenberg reduction is as follows.

ALGORITHM 4.6. *Reduction to upper Hessenberg form:*

if Q *is desired, set* $Q = I$
for $i = 1 : n - 2$
  $u_i = \text{House}(A(i+1:n, i))$
  $P_i = I - 2u_i u_i^T$ /* $Q_i = \text{diag}(I^{i \times i}, P_i)$ */
  $A(i+1:n, i:n) = P_i \cdot A(i+1:n, i:n)$
  $A(1:n, i+1:n) = A(1:n, i+1:n) \cdot P_i$
  if Q *is desired*
    $Q(i+1:n, i:n) = P_i \cdot Q(i+1:n, i:n)$ /* $Q = Q_i \cdot Q$ */
  end if
end for

As with the QR decomposition, one does not form P_i explicitly but instead multiplies by $I - 2u_i u_i^T$ via matrix-vector operations. The u_i vectors can also be stored below the subdiagonal, similar to the QR decomposition. They can be applied using Level 3 BLAS, as described in Question 3.17. This algorithm is available as the Matlab command `hess` or the LAPACK routine `sgehrd`.

The number of floating point operations is easily counted to be $\frac{10}{3}n^3 + O(n^2)$, or $\frac{14}{3}n^3 + O(n^2)$ if the product $Q = Q_{n-1} \cdots Q_1$ is computed as well.

The advantage of Hessenberg form under QR iteration is that it costs only $6n^2 + O(n)$ flops per iteration instead of $O(n^3)$, and its form is preserved so that the matrix remains upper Hessenberg.

PROPOSITION 4.5. *Hessenberg form is preserved by QR iteration.*

Proof. It is easy to confirm that the QR decomposition of an upper Hessenberg matrix like $A_i - \sigma I$ yields an upper Hessenberg Q (since the jth column of Q is a linear combination of the leading j columns of $A_i - \sigma I$). Then it is easy to confirm that RQ remains upper Hessenberg and adding σI does not change this. □

DEFINITION 4.5. *An upper Hessenberg matrix H is* unreduced *if all subdiagonals are nonzero.*

It is easy to see that if H is reduced because $h_{i+1,i} = 0$, then its eigenvalues are those of its leading i-by-i Hessenberg submatrix and its trailing $(n-i)$-by-$(n-i)$ Hessenberg submatrix, so we need consider only unreduced matrices.

4.4.7. Tridiagonal and Bidiagonal Reduction

If A is symmetric, the Hessenberg reduction process leaves A symmetric at each step, so zeros are created in symmetric positions. This means we need work on only half the matrix, reducing the operation count to $\frac{4}{3}n^3 + O(n^2)$ or $\frac{8}{3}n^3 + O(n^2)$ to form $Q_{n-1} \ldots Q_1$ as well. We call this algorithm *tridiagonal reduction.* We will use this algorithm in Chapter 5. This routine is available as LAPACK routine `ssytrd`.

Looking ahead a bit to our discussion of computing the SVD in section 5.4, we recall from section 3.2.3 that the eigenvalues of the symmetric matrix $A^T A$ are the squares of the singular values of A. Our eventual SVD algorithm will use this fact, so we would like to find a form for A which implies that $A^T A$ is tridiagonal. We will choose A to be *upper bidiagonal*, or nonzero only on the diagonal and first superdiagonal. Thus, we want to compute orthogonal matrices Q and V such that QAV is bidiagonal. The algorithm, called *bidiagonal reduction*, is very similar to Hessenberg and tridiagonal reduction.

EXAMPLE 4.9. Here is a 4-by-4 example of bidiagonal reduction, which illustrates the general pattern:

1. Choose Q_1 so

$$Q_1 A = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & x & x & x \\ o & x & x & x \end{bmatrix} \text{ and } V_1 \text{ so } A_1 \equiv Q_1 A V_1 = \begin{bmatrix} x & x & o & o \\ o & x & x & x \\ o & x & x & x \\ o & x & x & x \end{bmatrix}.$$

Q_1 is a Householder reflection, and V_1 is a Householder reflection that leaves the first column of $Q_1 A$ unchanged.

2. Choose Q_2 so

$$Q_2 A_1 = \begin{bmatrix} x & x & o & o \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \end{bmatrix} \text{ and } V_2 \text{ so } A_2 \equiv Q_2 A_1 V_2 = \begin{bmatrix} x & x & o & o \\ o & x & x & o \\ o & o & x & x \\ o & o & x & x \end{bmatrix}.$$

Q_2 is a Householder reflection that leaves the first row of A_1 unchanged. V_2 is a Householder reflection that leaves the first two columns of $Q_2 A_1$ unchanged.

3. Choose Q_3 so

$$Q_3 A_2 = \begin{bmatrix} x & x & o & o \\ o & x & x & o \\ o & o & x & x \\ o & o & o & x \end{bmatrix} \text{ and } V_3 = I \text{ so } A_3 = Q_3 A_2.$$

Q_3 is a Householder reflection that leaves the first two rows of A_2 unchanged. $\diamond$

In general, if A is n-by-n, then we get orthogonal matrices $Q = Q_{n-1} \cdots Q_1$ and $V = V_1 \cdots V_{n-2}$ such that $QAV = A'$ is upper bidiagonal.

Note that $A'^T A' = V^T A^T Q^T Q A V = V^T A^T A V$, so $A'^T A'$ has the same eigenvalues as $A^T A$; i.e., A' has the same singular values as A.

The cost of this bidiagonal reduction is $\frac{8}{3}n^3 + O(n^2)$ flops, plus another $4n^3 + O(n^2)$ flops to compute Q and V. This routine is available as LAPACK routine `sgebrd`.

4.4.8. QR Iteration with Implicit Shifts

In this section we show how to implement QR iteration cheaply on an upper Hessenberg matrix. The implementation will be *implicit* in the sense that we do not explicitly compute the QR factorization of a matrix H but rather construct Q implicitly as a product of Givens rotations and other simple orthogonal

matrices. The *implicit Q theorem* described below shows that this implicitly constructed Q is the Q we want. Then we show how to incorporate a single shift σ, which is necessary to accelerate convergence. To retain real arithmetic in the presence of complex eigenvalues, we then show how to do a *double* shift, i.e., combine two consecutive QR iterations with complex conjugate shifts σ and $\bar{\sigma}$; the result after this double shift is again real. Finally, we discuss strategies for choosing shifts σ and $\bar{\sigma}$ to provide reliable quadratic convergence. However, there have been recent discoveries of rare situation where convergence does not occur [25, 65], so finding a completely reliable and fast implementation of QR iteration remains an open problem.

Implicit Q Theorem

Our eventual implementation of QR iteration will depend on the following theorem.

THEOREM 4.9. Implicit Q theorem. *Suppose that $Q^TAQ = H$ is unreduced upper Hessenberg. Then columns 2 through n of Q are determined uniquely (up to signs) by the first column of Q.*

This theorem implies that to compute $A_{i+1} = Q_i^T A_i Q_i$ from A_i in the QR algorithm, we will need only to

1. compute the first column of Q_i (which is parallel to the first column of $A_i - \sigma_i I$ and so can be gotten just by normalizing this column vector).
2. choose other columns of Q_i so Q_i is orthogonal and A_{i+1} is unreduced Hessenberg.

Then by the implicit Q theorem, we know that we will have computed A_{i+1} correctly because Q_i is unique up to signs, which do not matter. (Signs do not matter because changing the signs of the columns of Q_i is the same as changing $A_i - \sigma_i I = Q_i R_i$ to $(Q_i S_i)(S_i R_i)$, where $S_i = \mathrm{diag}(\pm 1, \ldots, \pm 1)$. Then $A_{i+1} = (S_i R_i)(Q_i S_i) + \sigma_i I = S_i(R_i Q_i + \sigma_i I)S_i$, which is an orthogonal similarity that just changes the signs of the columns and rows of A_{i+1}.)

Proof of the implicit Q theorem. Suppose that $Q^TAQ = H$ and $V^TAV = G$ are unreduced upper Hessenberg, Q and V are orthogonal, and the first columns of Q and V are equal. Let $(X)_i$ denote the ith column of X. We wish to show $(Q)_i = \pm(V)_i$ for all $i > 1$, or equivalently, that $W \equiv V^TQ = \mathrm{diag}(\pm 1, \ldots, \pm 1)$.

Since $W = V^TQ$, we get $GW = GV^TQ = V^TAQ = V^TQH = WH$. Now $GW = WH$ implies $G(W)_i = (GW)_i = (WH)_i = \sum_{j=1}^{i+1} h_{ji}(W)_j$, so $h_{i+1,i}(W)_{i+1} = G(W)_i - \sum_{j=1}^{i} h_{ji}(W)_j$. Since $(W)_1 = [1, 0, \ldots, 0]^T$ and G is upper Hessenberg, we can use induction on i to show that $(W)_i$ is nonzero in entries 1 to i only; i.e., W is upper triangular. Since W is also orthogonal, W is diagonal $= \mathrm{diag}(\pm 1, \ldots, \pm 1)$. □

Implicit Single Shift QR Algorithm

To see how to use the implicit Q theorem to compute A_1 from $A_0 = A$, we use a 5-by-5 example.

EXAMPLE 4.10. 1. Choose

$$Q_1^T = \begin{bmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \text{ so } A_1 \equiv Q_1^T A Q_1 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ + & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix}.$$

We discuss how to choose c_1 and s_1 below; for now they may be any Givens rotation. The + in position (3,1) is called a *bulge* and needs to be gotten rid of to restore Hessenberg form.

2. Choose

$$Q_2^T = \begin{bmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -s_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \text{ so } Q_2^T A_1 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix}$$

and

$$A_2 \equiv Q_2^T A_1 Q_2 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & + & x & x & x \\ o & o & o & x & x \end{bmatrix}.$$

Thus the bulge has been "chased" from (3,1) to (4,2).

3. Choose

$$Q_3^T = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & c_3 & s_3 & \\ & & -s_3 & c_3 & \\ & & & & 1 \end{bmatrix} \text{ so } Q_3^T A_2 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix}$$

and

$$A_3 \equiv Q_3^T A_2 Q_3 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & + & x & x \end{bmatrix}.$$

The bulge has been chased from (4,2) to (5,3).

4. Choose

$$Q_4^T = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & c_4 & s_4 \\ & & & -s_4 & c_4 \end{bmatrix} \text{ so } Q_4^T A_3 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix}$$

and

$$A_4 = Q_4^T A_3 Q_4 = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ o & x & x & x & x \\ o & o & x & x & x \\ o & o & o & x & x \end{bmatrix},$$

so we are back to upper Hessenberg form.

Altogether $Q^T A Q$ is upper Hessenberg, where

$$Q = Q_1 Q_2 Q_3 Q_4 = \begin{bmatrix} c_1 & x & x & x & x \\ s_1 & x & x & x & x \\ & s_2 & x & x & x \\ & & s_3 & x & x \\ & & & s_4 & x \end{bmatrix},$$

so the first column of Q is $[c_1, s_1, 0, \ldots, 0]^T$, which by the implicit Q theorem has uniquely determined the other columns of Q (up to signs). We now choose the first column of Q to be proportional to the first column of $A - \sigma I$, $[a_{11} - \sigma, a_{21}, 0, \ldots, 0]^T$. This means Q is the same as in the QR decomposition of $A - \sigma I$, as desired. $\diamond$

The cost of one implicit QR iteration for an n-by-n matrix is $6n^2 + O(n)$.

Implicit Double Shift QR Algorithm

This section describes how to maintain real arithmetic by shifting by σ and $\overline{\sigma}$ at the same time. This is essential for an efficient practical implementation but not for a mathematical understanding of the algorithm and may be skipped on a first reading.

The results of shifting by σ and $\bar{\sigma}$ in succession are

$$\begin{aligned} A_0 - \sigma I &= Q_1 R_1, \\ A_1 &= R_1 Q_1 + \sigma I \quad \text{so } A_1 = Q_1^T A_0 Q_1, \\ A_1 - \overline{\sigma} I &= Q_2 R_2, \\ A_2 &= R_2 Q_2 + \overline{\sigma} I \quad \text{so } A_2 = Q_2^T A_1 Q_2 = Q_2^T Q_1^T A_0 Q_1 Q_2. \end{aligned}$$

LEMMA 4.5. *We can choose Q_1 and Q_2 so*

(1) Q_1Q_2 *is real,*
(2) A_2 *is therefore real,*
(3) *the first column of* Q_1Q_2 *is easy to compute.*

Proof. Since $Q_2R_2 = A_1 - \overline{\sigma}I = R_1Q_1 + (\sigma - \overline{\sigma})I$, we get

$$\begin{aligned} Q_1Q_2R_2R_1 &= Q_1(R_1Q_1 + (\sigma - \bar{\sigma})I)R_1 \\ &= Q_1R_1Q_1R_1 + (\sigma - \overline{\sigma})Q_1R_1 \\ &= (A_0 - \sigma I)(A_0 - \sigma I) + (\sigma - \overline{\sigma})(A_0 - \sigma I) \\ &= A_0^2 - 2(\Re\sigma)A_0 + |\sigma|^2 I \equiv M. \end{aligned}$$

Thus $(Q_1Q_2)(R_2R_1)$ is the QR decomposition of the real matrix M, and therefore Q_1Q_2, as well as R_2R_1, can be chosen real. This means that $A_2 = (Q_1Q_2)^T A(Q_1Q_2)$ also is real.

The first column of Q_1Q_2 is proportional to the first column of $A_0^2 - 2\Re\sigma A_0 + |\sigma^2|I$, which is

$$\begin{bmatrix} a_{11}^2 + a_{12}a_{21} - 2(\Re\sigma)a_{11} + |\sigma|^2 \\ a_{21}(a_{11} + a_{22} - 2(\Re\sigma)) \\ a_{21}a_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad \square$$

The rest of the columns of Q_1Q_2 are computed implicitly using the implicit Q theorem. The process is still called "bulge chasing," but now the bulge is 2-by-2 instead of 1-by-1.

EXAMPLE 4.11. Here is a 6-by-6 example of bulge chasing.

1. Choose $Q_1^T = \begin{bmatrix} \tilde{Q}_1^T & 0 \\ 0 & I \end{bmatrix}$, where the first column of $\tilde{Q}_1^T$ is given as above, so

$$Q_1^T A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ + & x & x & x & x & x \\ o & o & x & x & x & x \\ o & o & o & x & x & x \\ o & o & o & o & x & x \end{bmatrix} \text{ and } A_1 \equiv Q_1^T A Q_1 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ + & x & x & x & x & x \\ + & + & x & x & x & x \\ o & o & o & x & x & x \\ o & o & o & o & x & x \end{bmatrix}.$$

We see that there is a 2-by-2 bulge, indicted by plus signs.

2. Choose a Householder reflection Q_2^T, which affects only rows 2, 3, and 4 of $Q_2^T A_1$, zeroing out entries $(3,1)$ and $(4,1)$ of A_1 (this means that Q_2^T is the identity matrix outside rows and columns 2 through 4):

$$Q_2^T A_1 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & + & x & x & x & x \\ o & o & o & x & x & x \\ o & o & o & o & x & x \end{bmatrix} \quad \text{and} \quad A_2 \equiv Q_2^T A_1 Q_2 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & + & x & x & x & x \\ o & + & + & x & x & x \\ o & o & o & o & x & x \end{bmatrix},$$

and the 2-by-2 bulge has been "chased" one column.

3. Choose a Householder reflection Q_3^T, which affects only rows 3, 4, and 5 of $Q_3^T A_2$, zeroing out entries $(4,2)$ and $(5,2)$ of A_2 (this means that Q_3^T is the identity outside rows and columns 3 through 5):

$$Q_3^T A_2 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & o & x & x & x & x \\ o & o & + & x & x & x \\ o & o & o & o & x & x \end{bmatrix} \quad \text{and} \quad A_3 \equiv Q_3^T A_2 Q_3 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & o & x & x & x & x \\ o & o & + & x & x & x \\ o & o & + & + & x & x \end{bmatrix}.$$

4. Choose a Householder reflection Q_4^T, which affects only rows 4, 5, and 6 of $Q_4^T A_3$, zeroing out entries $(5,3)$ and $(6,3)$ of A_3 (this means that Q_4^T is the identity matrix outside rows and columns 4 through 6):

$$A_4 \equiv Q_4^T A_3 Q_4 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & o & x & x & x & x \\ o & o & o & x & x & x \\ o & o & o & + & x & x \end{bmatrix}.$$

5. Choose

$$Q_5^T = \left[\begin{array}{cccc|cc} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ \hline & & & & c & s \\ & & & & -s & c \end{array}\right] \quad \text{so} \quad A_5 = Q_5^T A_4 Q_5 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ o & x & x & x & x & x \\ o & o & x & x & x & x \\ o & o & o & x & x & x \\ o & o & o & o & x & x \end{bmatrix}. \diamond$$

Choosing a Shift for the QR Algorithm

To completely specify one iteration of either single shift or double shift Hessenberg QR iteration, we need to choose the shift σ (and $\overline{\sigma}$). Recall from the end of section 4.4.4 that a reasonable choice of single shift, one that resulted in asymptotic quadratic convergence to a real eigenvalue, was $\sigma = a_{n,n}$, the bottom right entry of A_i. The generalization for double shifting is to use the *Francis shift*, which means that σ and $\overline{\sigma}$ are the eigenvalues of the bottom 2-by-2 corner of A_i: $\begin{bmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n,n-1} & a_{n,n} \end{bmatrix}$. This will let us converge to either two real eigenvalues in the bottom 2-by-2 corner or a single 2-by-2 block with complex conjugate eigenvalues. When we are close to convergence, we expect $a_{n-1,n-2}$ (and possibly $a_{n,n-1}$) to be small so that the eigenvalues of this 2-by-2 matrix are good approximations for eigenvalues of A. Indeed, one can show that this choice leads to quadratic convergence asymptotically. This means that once $a_{n-1,n-2}$ (and possibly $a_{n,n-1}$) is small enough, its magnitude will square at each step and quickly approach zero. In practice, this works so well that on average only two QR iterations per eigenvalue are needed for convergence for almost all matrices. This justifies calling QR iteration a "direct" method.

In practice, the QR iteration with the Francis shift can fail to converge (indeed, it leaves

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

unchanged). So the practical algorithm in use for decades had an "exceptional shift" every 10 shifts if convergence had not occurred. Still, tiny sets of matrices where that algorithm did not converge were discovered only recently [25, 65]; matrices in a small neighborhood of

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & h & 0 \\ 0 & -h & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

where h is a few thousand times machine epsilon, form such a set. So another "exceptional shift" was recently added to the algorithm to patch this case. But it is still an open problem to find a shift strategy that guarantees fast convergence for all matrices.

4.5. Other Nonsymmetric Eigenvalue Problems

4.5.1. Regular Matrix Pencils and Weierstrass Canonical Form

The standard eigenvalue problem asks for which scalars z the matrix $A - zI$ is singular; these scalars are the eigenvalues. This notion generalizes in several important ways.

DEFINITION 4.6. *$A - \lambda B$, where A and B are m-by-n matrices, is called a* matrix pencil, *or just a* pencil. *Here λ is an indeterminate, not a particular, numerical value.*

DEFINITION 4.7. *If A and B are square and $\det(A - \lambda B)$ is not identically zero, the pencil $A - \lambda B$ is called* regular. *Otherwise it is called* singular. *When $A - \lambda B$ is regular, $p(\lambda) \equiv \det(A - \lambda B)$ is called the* characteristic polynomial *of $A - \lambda B$ and the eigenvalues of $A - \lambda B$ are defined to be*

(1) *the roots of $p(\lambda) = 0$,*
(2) *∞ (with multiplicity $n - \deg(p)$) if $\deg(p) < n$.*

EXAMPLE 4.12. Let

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

Then $p(\lambda) = \det(A - \lambda B) = (1 - 2\lambda) \cdot (1 - 0\lambda) \cdot (0 - \lambda) = (2\lambda - 1)\lambda$, so the eigenvalues are $\lambda = \frac{1}{2}, 0$ and ∞. $\diamond$

Matrix pencils arise naturally in many mathematical models of physical systems; we give examples below. The next proposition relates the eigenvalues of a regular pencil $A - \lambda B$ to the eigenvalues of a single matrix.

PROPOSITION 4.6. *Let $A - \lambda B$ be regular. If B is nonsingular, all eigenvalues of $A - \lambda B$ are finite and the same as the eigenvalues of AB^{-1} or $B^{-1}A$. If B is singular, $A - \lambda B$ has eigenvalue ∞ with multiplicity $n - \operatorname{rank}(B)$. If A is nonsingular, the eigenvalues of $A - \lambda B$ are the same as the reciprocals of the eigenvalues of $A^{-1}B$ or BA^{-1}, where a zero eigenvalue of $A^{-1}B$ corresponds to an infinite eigenvalue of $A - \lambda B$.*

Proof. If B is nonsingular and λ' is an eigenvalue, then $0 = \det(A - \lambda' B) = \det(AB^{-1} - \lambda' I) = \det(B^{-1}A - \lambda' I)$, so λ' is also an eigenvalue of AB^{-1} and $B^{-1}A$. If B is singular, then take $p(\lambda) = \det(A - \lambda B)$, write the SVD of B as $B = U\Sigma V^T$, and substitute to get

$$p(\lambda) = \det(A - \lambda U\Sigma V^T) = \det(U(U^T AV - \lambda\Sigma)V^T) = \pm\det(U^T AV - \lambda\Sigma).$$

Since $\operatorname{rank}(B) = \operatorname{rank}(\Sigma)$, only $\operatorname{rank}(B)$ λ's appear in $U^T AV - \lambda\Sigma$, so the degree of the polynomial $\det(U^T AV - \lambda\Sigma)$ is $\operatorname{rank}(B)$.

If A is nonsingular, $\det(A - \lambda B) = 0$ if and only if $\det(I - \lambda A^{-1}B) = 0$ or $\det(I - \lambda BA^{-1}) = 0$. This equality can hold only if $\lambda \neq 0$ and $1/\lambda$ is an eigenvalue of $A^{-1}B$ or BA^{-1}. $\square$

DEFINITION 4.8. *Let λ' be a finite eigenvalue of the regular pencil $A - \lambda B$. Then $x \neq 0$ is a* right eigenvector *if $(A - \lambda' B)x = 0$, or equivalently $Ax = \lambda' Bx$. If $\lambda' = \infty$ is an eigenvalue and $Bx = 0$, then x is a right eigenvector. A left eigenvector of $A - \lambda B$ is a right eigenvector of $(A - \lambda B)^*$.*

EXAMPLE 4.13. Consider the pencil $A - \lambda B$ in Example 4.12. Since A and B are diagonal, the right and left eigenvectors are just the columns of the identity matrix. ◇

EXAMPLE 4.14. Consider the damped mass-spring system from Example 4.1. There are two matrix pencils that arise naturally from this problem. First, we can write the eigenvalue problem

$$Ax = \begin{bmatrix} -M^{-1}B & -M^{-1}K \\ I & 0 \end{bmatrix} x = \lambda x$$

as

$$\begin{bmatrix} -B & -K \\ I & 0 \end{bmatrix} x = \lambda \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} x.$$

This may be a superior formulation if M is very ill-conditioned, so that $M^{-1}B$ and $M^{-1}K$ are hard to compute accurately.

Second, it is common to consider the case $B = 0$ (no damping), so the original differential equation is $M\ddot{x}(t) + Kx(t) = 0$. Seeking solutions of the form $x_i(t) = e^{\lambda_i t} x_i(0)$, we get $\lambda_i^2 e^{\lambda_i t} M x_i(0) + e^{\lambda_i t} K x_i(0) = 0$, or $\lambda_i^2 M x_i(0) + K x_i(0) = 0$. In other words, $-\lambda_i^2$ is an eigenvalue and $x_i(0)$ is a right eigenvector of the pencil $K - \lambda M$. Since we are assuming that M is nonsingular, these are also the eigenvalue and right eigenvector of $M^{-1}K$. ◇

Infinite eigenvalues also arise naturally in practice. For example, later in this section we will show how infinite eigenvalues correspond to *impulse response* in a system described by ordinary differential equations with linear constraints, or *differential-algebraic equations* [41]. See also Question 4.16 for an application of matrix pencils to computational geometry and computer graphics.

Recall that all of our theory and algorithms for the eigenvalue problem of a single matrix A depended on finding a similarity transformation $S^{-1}AS$ of A that is in "simpler" form than A. The next definition shows how to generalize the notion of similarity to matrix pencils. Then we show how the Jordan form and Schur form generalize to pencils.

DEFINITION 4.9. *Let P_L and P_R be nonsingular matrices. Then pencils $A - \lambda B$ and $P_L A P_R - \lambda P_L B P_R$ are called* equivalent.

PROPOSITION 4.7. *The equivalent regular pencils $A - \lambda B$ and $P_L A P_R - \lambda P_L B P_R$ have the same eigenvalues. The vector x is a right eigenvector of $A - \lambda B$ if*

and only if $P_R^{-1}x$ is a right eigenvector of $P_LAP_R - \lambda P_LBP_R$. The vector y is a left eigenvector of $A - \lambda B$ if and only if $(P_L^*)^{-1}y$ is a left eigenvector of $P_LAP_R - \lambda P_LBP_R$.

Proof.

$\det(A - \lambda B) = 0$ if and only if $\det(P_L(A - \lambda B)P_R) = 0$.

$(A - \lambda B)x = 0$ if and only if $P_L(A - \lambda B)P_RP_R^{-1}x = 0$.

$(A - \lambda B)^*y = 0$ if and only if $P_R^*(A - \lambda B)^*P_L^*(P_L^*)^{-1}y = 0$. □

The following theorem generalizes the Jordan canonical form to regular matrix pencils.

THEOREM 4.10. Weierstrass canonical form. *Let $A - \lambda B$ be regular. Then there are nonsingular P_L and P_R such that*

$$P_L(A - \lambda B)P_R = \operatorname{diag}(J_{n_1}(\lambda_1) - \lambda I_{n_1}, \ldots, J_{n_k}(\lambda_{n_k}) - \lambda I_{n_k}, N_{m_1}, \ldots, N_{m_r}),$$

where $J_{n_i}(\lambda_i)$ is an n_i-by-n_i Jordan block with eigenvalue λ_i,

$$J_{n_i}(\lambda_i) = \begin{bmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix},$$

and N_{m_i} is a "Jordan block for $\lambda = \infty$ with multiplicity m_i,"

$$N_{m_i} = \begin{bmatrix} 1 & \lambda & & \\ & 1 & \ddots & \\ & & \ddots & \lambda \\ & & & 1 \end{bmatrix} = I_{m_i} - \lambda J_{m_i}(0).$$

For a proof, see [110].

Application of Jordan and Weierstrass Forms to Differential Equations

Consider the linear differential equation $\dot{x}(t) = Ax(t) + f(t)$, $x(0) = x_0$. An explicit solution is given by $x(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}f(\tau)d\tau$. If we know the Jordan form $A = SJS^{-1}$, we may change variables in the differential equation to $y(t) = S^{-1}x(t)$ to get $\dot{y}(t) = Jy(t) + S^{-1}f(t)$, with solution $y(t) = e^{Jt}y_0 + \int_0^t e^{J(t-\tau)}S^{-1}f(\tau)d\tau$. There is an explicit formula to compute e^{Jt} or any other function $f(J)$ of a matrix in Jordan form J. (We should not use this formula numerically! For the basis of a better algorithm, see Question 4.4.) Suppose that f is given by its Taylor series $f(z) = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)z^i}{i!}$ and J is a

single Jordan block $J = \lambda I + N$, where N has ones on the first superdiagonal and zeros elsewhere. Then

$$\begin{aligned} f(J) &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)(\lambda I + N)^i}{i!} \\ &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \sum_{j=0}^{i} \binom{i}{j} \lambda^{i-j} N^j \quad \text{by the binomial theorem} \\ &= \sum_{j=0}^{\infty} \sum_{i=j}^{\infty} \frac{f^{(i)}(0)}{i!} \binom{i}{j} \lambda^{i-j} N^j \quad \text{reversing the order of summation} \\ &= \sum_{j=0}^{n-1} N^j \sum_{i=j}^{\infty} \frac{f^{(i)}(0)}{i!} \binom{i}{j} \lambda^{i-j}, \end{aligned}$$

where in the last equality we used the fact that $N^j = 0$ for $j > n-1$. Note that N^j has ones on the jth superdiagonal and zeros elsewhere. Finally, note that $\sum_{i=j}^{\infty} \frac{f^{(i)}(0)}{i!} \binom{i}{j} \lambda^{i-j}$ is the Taylor expansion for $f^{(j)}(\lambda)/j!$. Thus

$$\begin{aligned} f(J) &= f\left(\begin{bmatrix} \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}^{n \times n} \right) = \sum_{j=0}^{n-1} \frac{N^j f^{(j)}(\lambda)}{j!} \\ &= \begin{bmatrix} f(\lambda) & f'(\lambda) & \frac{f''(\lambda)}{2!} & \cdots & \frac{f^{(n-1)}(\lambda)}{(n-1)!} \\ & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \frac{f''(\lambda)}{2!} \\ & & & \ddots & f'(\lambda) \\ & & & & f(\lambda) \end{bmatrix} \end{aligned} \tag{4.6}$$

so that $f(J)$ is upper triangular with $f^{(j)}(\lambda)/j!$ on the jth superdiagonal.

To solve the more general problem $B\dot{x} = Ax + f(t)$, $A - \lambda B$ regular, we use the Weierstrass form: let $P_L(A - \lambda B)P_R$ be in Weierstrass form, and rewrite the equation as $P_L B P_R P_R^{-1} \dot{x} = P_L A P_R P_R^{-1} x + P_L f(t)$. Let $P_R^{-1} x = y$ and $P_L f(t) = g(t)$. Now the problem has been decomposed into subproblems:

$$\begin{bmatrix} I_{n_1} & & & & & \\ & \ddots & & & & \\ & & I_{n_k} & & & \\ & & & J_{m_1}(0) & & \\ & & & & \ddots & \\ & & & & & J_{m_r}(0) \end{bmatrix} \dot{y}$$

$$= \begin{bmatrix} J_{n_1}(\lambda_1) & & & & & \\ & \ddots & & & & \\ & & J_{n_k}(\lambda_k) & & & \\ & & & I_{m_1} & & \\ & & & & \ddots & \\ & & & & & I_{m_r} \end{bmatrix} y + g.$$

Each subproblem $\dot{\tilde{y}} = J_{n_i}(\lambda_i)\tilde{y} + \tilde{g}(t) \equiv J\tilde{y} + \tilde{y}(t)$ is a standard linear ODE as above with solution

$$\tilde{y}(t) = \tilde{y}(0)e^{Jt} + \int_0^t e^{J(t-\tau)}\tilde{g}(\tau)d\tau$$

The solution of $J_m(0)\dot{\tilde{y}} = \tilde{y} + \tilde{g}(t)$ is gotten by back substitution starting from the last equation: write $J_m(0)\dot{\tilde{y}} = \tilde{y} + \tilde{g}(t)$ as

$$\begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix} \begin{bmatrix} \dot{\tilde{y}}_1 \\ \vdots \\ \vdots \\ \dot{\tilde{y}}_m \end{bmatrix} = \begin{bmatrix} \tilde{y}_1 \\ \vdots \\ \vdots \\ \tilde{y}_m \end{bmatrix} + \begin{bmatrix} \tilde{g}_1 \\ \vdots \\ \vdots \\ \tilde{g}_m \end{bmatrix}.$$

The mth (last) equation says $0 = \tilde{y}_m + \tilde{g}_m$ or $\tilde{y}_m = -\tilde{g}_m$. The ith equation says $\dot{\tilde{y}}_{i+1} = \tilde{y}_i + \tilde{g}_i$, so $\tilde{y}_i = \dot{\tilde{y}}_{i+1} - \tilde{g}_i$ and thus

$$\tilde{y}_i = \sum_{k=i}^{m} -\frac{d^{k-i}}{dt^{k-i}}\tilde{g}_k(t).$$

Therefore the solution depends on derivatives of $\tilde{g}$, *not* an integral of $\tilde{g}$ as in the usual ODE. Thus a continuous $\tilde{g}$ which is not differentiable can cause a discontinuity in the solution; this is sometimes called an *impulse response* and occurs only if there are infinite eigenvalues. Furthermore, to have a continuous solution $\tilde{y}$ must satisfy certain consistency conditions at $t = 0$:

$$y_i(0) = \sum_{k=m}^{i} -\frac{d^{k-i}}{dt^{k-i}}g_k(0).$$

Numerical methods, based on time-stepping, for solving such *differential algebraic equations*, or ODEs with algebraic constraints, are described in [41].

Generalized Schur Form for Regular Pencils

Just as we cannot compute the Jordan form stably, we cannot compute its generalization by Weierstrass stably. Instead, we compute the generalized Schur form.

THEOREM 4.11. Generalized Schur form. *Let $A - \lambda B$ be regular. Then there exist unitary Q_L and Q_R so that $Q_L A Q_R = T_A$ and $Q_L B Q_R = T_B$ are both upper triangular. The eigenvalues of $A - \lambda B$ are then $T_{A_{ii}}/T_{B_{ii}}$, the ratios of the diagonal entries of T_A and T_B.*

Proof. The proof is very much like that for the usual Schur form. Let λ' be an eigenvalue and x be a unit right eigenvector: $\|x\|_2 = 1$. Since $Ax - \lambda' Bx = 0$, both Ax and Bx are multiples of the same unit vector y (even if one of Ax or Bx is zero). Now let $X = [x, \tilde{X}]$ and $Y = [y, \tilde{Y}]$ be unitary matrices with first columns x and y, respectively. Then $Y^*AX = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix}$ and $Y^*BX = \begin{bmatrix} \tilde{b}_{11} & \tilde{b}_{12} \\ 0 & \tilde{B}_{22} \end{bmatrix}$ by construction. Apply this process inductively to $\tilde{A}_{22} - \lambda \tilde{B}_{22}$. □

If A and B are real, there is a generalized real Schur form too: real orthogonal Q_L and Q_R, where $Q_L A Q_R$ is quasi–upper triangular and $Q_L B Q_R$ is upper triangular.

The QR algorithm and all its refinements generalize to compute the generalized (real) Schur form; it is called the QZ algorithm and available in LAPACK subroutine `sgges`. In Matlab one uses the command `eig(A,B)`.

Definite Pencils

A simpler special case that often arises in practice is the pencil $A - \lambda B$, where $A = A^T$, $B = B^T$, and B is positive definite. Such pencils are called *definite pencils.*

THEOREM 4.12. *Let $A = A^T$, and let $B = B^T$ be positive definite. Then there is a real nonsingular matrix X so that $X^T A X = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$ and $X^T B X = \mathrm{diag}(\beta_1, \ldots, \beta_n)$. In particular, all the eigenvalues α_i/β_i are real and finite.*

Proof. The proof that we give is actually the algorithm used to solve the problem:

(1) Let $LL^T = B$ be the Cholesky decomposition.
(2) Let $H = L^{-1} A L^{-T}$; note that H is symmetric.
(3) Let $H = Q \Lambda Q^T$, with Q orthogonal, Λ real and diagonal.

Then $X = L^{-T} Q$ satisfies $X^T A X = Q^T L^{-1} A L^{-T} Q = \Lambda$ and $X^T B X = Q^T L^{-1} B L^{-T} Q = I$. □

Note that the theorem is also true if $\alpha A + \beta B$ is positive definite for some scalars α and β.

Software for this problem is available as LAPACK routine `ssygv`.

EXAMPLE 4.15. Consider the pencil $K - \lambda M$ from Example 4.14. This is a definite pencil since the stiffness matrix K is symmetric and the mass matrix

M is symmetric and positive definite. In fact, K is tridiagonal and M is diagonal in this very simple example, so M's Cholesky factor L is also diagonal, and $H = L^{-1}KL^{-T}$ is also symmetric and tridiagonal. In Chapter 5 we will consider a variety of algorithms for the symmetric tridiagonal eigenproblem. ◇

4.5.2. Singular Matrix Pencils and the Kronecker Canonical Form

Now we consider singular pencils $A - \lambda B$. Recall that $A - \lambda B$ is singular if either A and B are nonsquare or they are square and $\det(A - \lambda B) = 0$ for all values of λ. The next example shows that care is needed in extending the definition of eigenvalues to this case.

EXAMPLE 4.16. Let $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. Then by making arbitrarily small changes to get $A' = \begin{bmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 0 \end{bmatrix}$ and $B' = \begin{bmatrix} 1 & \epsilon_3 \\ \epsilon_4 & 0 \end{bmatrix}$, the eigenvalues become ϵ_1/ϵ_3 and ϵ_2/ϵ_4, which can be arbitrary complex numbers. So the eigenvalues are *infinitely* sensitive. ◇

Despite this extreme sensitivity, singular pencils are used in modeling certain physical systems, as we describe below.

We continue by showing how to generalize the Jordan and Weierstrass forms to singular pencils. In addition to Jordan and "infinite Jordan" blocks, we get two new "singular blocks" in the canonical form.

THEOREM 4.13. Kronecker canonical form. *Let A and B be arbitrary rectangular m-by-n matrices. Then there are square nonsingular matrices P_L and P_R so that $P_LAP_R - \lambda P_LBP_R$ is block diagonal with four kinds of blocks:*

$$J_m(\lambda') - \lambda I = \begin{bmatrix} \lambda' - \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda' - \lambda \end{bmatrix}, \quad m\text{-by-}m \text{ Jordan block};$$

$$N_m = \begin{bmatrix} 1 & \lambda & & \\ & \ddots & \ddots & \\ & & \ddots & \lambda \\ & & & 1 \end{bmatrix}, \quad m\text{-by-}m \text{ Jordan block for } \lambda = \infty;$$

$$L_m = \begin{bmatrix} 1 & \lambda & & \\ & \ddots & \ddots & \\ & & 1 & \lambda \end{bmatrix}, \quad \begin{array}{l} m\text{-}by\text{-}(m+1)\ right \\ singular\ block; \end{array}$$

$$L_m^T = \begin{bmatrix} 1 & & \\ \lambda & \ddots & \\ & \ddots & 1 \\ & & \lambda \end{bmatrix}, \quad \begin{array}{l} (m+1)\text{-}by\text{-}m\ left \\ singular\ block. \end{array}$$

We call L_m a right singular block since it has a right null vector $[\lambda^m, -\lambda^{m-1}, \ldots, \pm 1]$ for all λ. L_m^T has an analogous left null vector.

For a proof, see [110].

Just as Schur form generalized to regular matrix pencils in the last section, it can be generalized to arbitrary singular pencils as well. For the canonical form, perturbation theory and software, see [27, 79, 246].

Singular pencils are used to model systems arising in systems and control. We give two examples.

Application of Kronecker Form to Differential Equations

Suppose that we want to solve $B\dot{x} = Ax + f(t)$, where $A - \lambda B$ is a singular pencil. Write $P_L B P_R P_R^{-1} \dot{x} = P_L A P_R P_R^{-1} x + P_L f(t)$ to decompose the problem into independent blocks. There are four kinds, one for each kind in the Kronecker form. We have already dealt with $J_m(\lambda') - \lambda I$ and N_m blocks when we considered regular pencils and Weierstrass form, so we have to consider only L_m and L_m^T blocks. From the L_m blocks we get

$$\begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_{m+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_{m+1} \end{bmatrix} + \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix}$$

or

$$\begin{array}{lllll} \dot{y}_2 = y_1 + g_1 & \text{or} & y_2(t) = y_2(0) + \int_0^t (y_1(\tau) + g_1(\tau))d\tau, \\ \dot{y}_3 = y_2 + g_2 & \text{or} & y_3(t) = y_3(0) + \int_0^t (y_2(\tau) + g_2(\tau))d\tau, \\ \vdots & & \\ \dot{y}_{m+1} = y_m + g_m & \text{or} & y_{m+1}(t) = y_{m+1}(0) + \int_0^t (y_m(\tau) + g_m(\tau))d\tau. \end{array}$$

This means that we can choose y_1 as an arbitrary integrable function and use the above recurrence relations to get a solution. This is because we have one more unknown than equation, so the the ODE is *underdetermined.* From the L_m^T blocks we get

$$\begin{bmatrix} 0 & & \\ 1 & \ddots & \\ & \ddots & 0 \\ & & 1 \end{bmatrix} \begin{bmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_m \end{bmatrix} = \begin{bmatrix} 1 & & \\ 0 & \ddots & \\ & \ddots & 1 \\ & & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} + \begin{bmatrix} g_1 \\ \vdots \\ g_{m+1} \end{bmatrix}$$

or

$$\begin{aligned} 0 &= y_1 + g_1, \\ \dot{y}_1 &= y_2 + g_2, \\ &\vdots \\ \dot{y}_{m-1} &= y_m + g_m, \\ \dot{y}_m &= g_{m+1}. \end{aligned}$$

Starting with the first equation, we solve to get

$$\begin{aligned} y_1 &= -g_1, \\ y_2 &= -g_2 - \dot{g}_1, \\ &\vdots \\ y_m &= -g_m - \dot{g}_{m-1} - \cdots - \frac{d^{m-1}}{dt^{m-1}} g_1 \end{aligned}$$

and the *consistency condition* $g_{m+1} = -\dot{g}_m - \cdots - \frac{d^m}{dt^m} g_1$. So unless the g_i satisfy this equation, there is no solution. Here we have one more equation than unknown, and the subproblem is *overdetermined.*

Application of Kronecker Form to Systems and Control Theory

The *controllable subspace* of $\dot{x}(t) = Ax(t) + Bu(t)$ is the space in which the *system state* $x(t)$ can be "controlled" by choosing the *control input* $u(t)$ starting at $x(0) = 0$. This equation is used to model (feedback) control systems, where the $u(t)$ is chosen by the control system engineer to make $x(t)$ have certain desirable properties, such as boundedness. From

$$\begin{aligned} x(t) &= \int_0^t e^{A(t-\tau)} B u(\tau) d\tau = \int_0^t \sum_{i=0}^{\infty} \frac{(t-\tau)^i}{i!} A^i B u(\tau) d\tau \\ &= \sum_{i=0}^{\infty} A^i B \int_0^t \frac{(t-\tau)^i}{i!} u(\tau) d\tau \end{aligned}$$

one can prove the controllable space is $\text{span}\{[B, AB, A^2B, \ldots, A^{n-1}B]\}$; any components of $x(t)$ outside this space cannot be controlled by varying $u(t)$. To compute this space in practice, in order to determine whether the physical system being modeled can in fact be controlled by input $u(t)$, one applies a QR-like algorithm to the singular pencil $[B, A - \lambda I]$. For details, see [78, 246, 247].

4.5.3. Nonlinear Eigenvalue Problems

Finally, we consider the *nonlinear eigenvalue problem* or *matrix polynomial*

$$\sum_{i=0}^{d} \lambda^i A_i = \lambda^d A_d + \lambda^{d-1} A_{d-1} + \cdots + \lambda A_1 + A_0. \tag{4.7}$$

Suppose for simplicity that the A_i are n-by-n matrices and A_d is nonsingular.

DEFINITION 4.10. *The* characteristic polynomial *of the matrix polynomial* (4.7) *is* $p(\lambda) = \det(\sum_{i=0}^{d} \lambda^i A_i)$. *The roots of* $p(\lambda) = 0$ *are defined to be the* eigenvalues. *One can confirm that* $p(\lambda)$ *has degree* $d \cdot n$, *so there are* $d \cdot n$ *eigenvalues. Suppose that* γ *is an eigenvalue. A nonzero vector* x *satisfying* $\sum_{i=0}^{d} \gamma^i A_i x = 0$ *is a* right eigenvector *for* γ. *A left eigenvector* y *is defined analogously by* $\sum_{i=0}^{d} \gamma^i y^* A_i = 0$.

EXAMPLE 4.17. Consider Example 4.1 once again. The ODE arising there in equation (4.3) is $M\ddot{x}(t) + B\dot{x}(t) + Kx(t) = 0$. If we seek solutions of the form $x(t) = e^{\lambda_i t} x_i(0)$, we get $e^{\lambda_i t}(\lambda_i^2 M x_i(0) + \lambda_i B x_i(0) + K x_i(0)) = 0$, or $\lambda_i^2 M x_i(0) + \lambda_i B x_i(0) + K x_i(0) = 0$. Thus λ_i is an eigenvalue and $x_i(0)$ is an eigenvector of the matrix polynomial $\lambda^2 M + \lambda B + K$. $\diamond$

Since we are assuming that A_d is nonsingular, we can multiply through by A_d^{-1} to get the equivalent problem $\lambda^d I + A_d^{-1} A_{d-1} \lambda^{d-1} + \cdots + A_d^{-1} A_0$. Therefore, to keep the notation simple, we will assume $A_d = I$ (see section 4.6 for the general case). In the very simplest case where each A_i is 1-by-1, i.e., a scalar, the original matrix polynomial is equal to the characteristic polynomial.

We can turn the problem of finding the eigenvalues of a matrix polynomial into a standard eigenvalue problem by using a trick analogous to the one used to change a high-order ODE into a first-order ODE. Consider first the simplest case $n = 1$, where each A_i is a scalar. Suppose that γ is a root. Then the vector $x' = [\gamma^{d-1}, \gamma^{d-2}, \ldots, \gamma, 1]^T$ satisfies

$$Cx' \equiv \begin{bmatrix} -A_{d-1} & -A_{d-2} & \cdots & \cdots & \cdots & -A_0 \\ 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & 0 \end{bmatrix} x' = \begin{bmatrix} -\sum_{i=0}^{d-1} \gamma^i A_i \\ \gamma^{d-1} \\ \vdots \\ \gamma^2 \\ \gamma \end{bmatrix}$$

$$= \begin{bmatrix} \gamma^d \\ \gamma^{d-1} \\ \vdots \\ \gamma^2 \\ \gamma \end{bmatrix} = \gamma x'.$$

Thus x' is an eigenvector and γ is an eigenvalue of the matrix C, which is called the *companion matrix* of the polynomial (4.7).

(The Matlab routine `roots` for finding roots of a polynomial applies the Hessenberg QR iteration of section 4.4.8 to the companion matrix C, since this is currently one of the most reliable, if expensive, methods known [100, 117, 241]. Cheaper alternatives are under development.)

The same idea works when the A_i are matrices. C becomes an $(n \cdot d)$-by-$(n \cdot d)$ *block companion matrix*, where the 1's and 0's below the top row become n-by-n identity and zero matrices, respectively. Also, x' becomes

$$x' = \begin{bmatrix} \gamma^{d-1}x \\ \gamma^{d-2}x \\ \vdots \\ \gamma x \\ x \end{bmatrix},$$

where x is a right eigenvector of the matrix polynomial. It again turns out that $Cx' = \gamma x'$.

EXAMPLE 4.18. Returning once again to $\lambda^2 M + \lambda B + K$, we first convert it to $\lambda^2 + \lambda M^{-1}B + M^{-1}K$ and then to the companion matrix

$$C = \begin{bmatrix} -M^{-1}B & -M^{-1}K \\ I & 0 \end{bmatrix}.$$

This is the same as the matrix A in equation 4.4 of Example 4.1. ◇

Finally, Question 4.16 shows how to use matrix polynomials to solve a problem in *computational geometry.*

4.6. Summary

The following list summarizes all the canonical forms, algorithms, their costs, and applications to ODEs described in this chapter. It also includes pointers to algorithms exploiting symmetry, although these are discussed in more detail in the next chapter. Algorithms for sparse matrices are discussed in Chapter 7.

- $A - \lambda I$
  - Jordan form: For some nonsingular S,

$$A - \lambda I = S \cdot \mathrm{diag}\left(\ldots, \begin{bmatrix} \lambda_i - \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i - \lambda \end{bmatrix}^{n_i \times n_i}, \ldots \right) \cdot S^{-1}.$$

- Schur form: For some unitary Q, $A - \lambda I = Q(T - \lambda I)Q^*$, where T is triangular.
- Real Schur form of real A: For some real orthogonal Q, $A - \lambda I = Q(T - \lambda I)Q^T$, where T is real quasi-triangular.
- Application to ODEs: Provides solution of $\dot{x}(t) = Ax(t) + f(t)$.
- Algorithm: Do Hessenberg reduction (Algorithm 4.6), followed by QR iteration to get Schur form (Algorithm 4.5, implemented as described in section 4.4.8). Eigenvectors can be computed from the Schur form (as described in section 4.2.1).
- Cost: This costs $10n^3$ flops if eigenvalues only are desired, $25n^3$ if T and Q are also desired, and a little over $27n^3$ if eigenvectors are also desired. Since not all parts of the algorithm can take advantage of the Level 3 BLAS, the cost is actually higher than a comparison with the $2n^3$ cost of matrix multiply would indicate: instead of taking $(10n^3)/(2n^3) = 5$ times longer to compute eigenvalues than to multiply matrices, it takes 23 times longer for $n = 100$ and 19 times longer for $n = 1000$ on an IBM RS6000/590 [10, page 62]. Instead of taking $(27n^3)/(2n^3) = 13.5$ times longer to compute eigenvalues and eigenvectors, it takes 41 times longer for $n = 100$ and 60 times longer for $n = 1000$ on the same machine. Thus computing eigenvalues of nonsymmetric matrices is expensive. (The symmetric case is *much* cheaper; see Chapter 5.)
- LAPACK: `sgees` for Schur form or `sgeev` for eigenvalues and eigenvectors; `sgeesx` or `sgeevx` for error bounds too.
- Matlab: `schur` for Schur form or `eig` for eigenvalues and eigenvectors.
- Exploiting symmetry: When $A = A^*$, better algorithms are discussed in Chapter 5, especially section 5.3.

- Regular $A - \lambda B$ ($\det(A - \lambda B) \not\equiv 0$)
  - Weierstrass form: For some nonsingular P_L and P_R,

$$A - \lambda B = P_L \cdot \mathrm{diag}\left(\text{Jordan}, \begin{bmatrix} 1 & \lambda & & \\ & \ddots & \ddots & \\ & & \ddots & \lambda \\ & & & 1 \end{bmatrix}^{n_i \times n_i}\right) P_R^{-1}.$$

  - Generalized Schur form: For some unitary Q_L and Q_R, $A - \lambda B = Q_L(T_A - \lambda T_B)Q_R^*$, where T_A and T_B are triangular.

- Generalized real Schur form of real A and B: For some real orthogonal Q_L and Q_R, $A - \lambda B = Q_L(T_A - \lambda T_B)Q_R^T$, where T_A is real quasi-triangular and T_B is real triangular.
- Application to ODEs: Provides solution of $B\dot{x}(t) = Ax(t) + f(t)$, where the solution is uniquely determined but may depend non-smoothly on the data (*impulse response*).
- Algorithm: Hessenberg/triangular reduction followed by QZ iteration (QR applied implicitly to AB^{-1}).
- Cost: Computing T_A and T_B costs $30n^3$. Computing Q_L and Q_R in addition costs $66n^3$. Computing eigenvectors as well costs a little less than $69n^3$ in total. As before, Level 3 BLAS cannot be used in all parts of the algorithm.
- LAPACK: `sgges` for Schur form or `sggev` for eigenvalues; `sggesx` or `sggevx` for error bounds too.
- Matlab: `eig` for eigenvalues and eigenvectors.
- Exploiting symmetry: When $A = A^*$, $B = B^*$, and B is positive definite, one can convert the problem to finding the eigenvalues of a single symmetric matrix using Theorem 4.12. This is done in LAPACK routines `ssygv`, `sspgv` (for symmetric matrices in "packed storage"), and `ssbgv` (for symmetric band matrices).

- Singular $A - \lambda B$
  - Kronecker form: For some nonsingular P_L and P_R,

$$A - \lambda B = P_L \cdot \operatorname{diag}\left(\text{Weierstrass}, \begin{bmatrix} 1 & \lambda & & \\ & \ddots & \ddots & \\ & & 1 & \lambda \end{bmatrix}^{n_i \times n_i}, \begin{bmatrix} 1 & & \\ \lambda & \ddots & \\ & \ddots & 1 \\ & & \lambda \end{bmatrix}^{m_i \times m_i}\right) P_R^{-1}.$$

  - Generalized upper triangular form: For some unitary Q_L and Q_R, $A - \lambda B = Q_L(T_A - \lambda T_B)Q_R^*$, where T_A and T_B are in generalized upper triangular form, with diagonal blocks corresponding to different parts of the Kronecker form. See [79, 246] for details of the form and algorithms.
  - Cost: The most general and reliable version of the algorithm can cost as much as $O(n^4)$, depending on the details of the Kronecker Structure; this is much more than for regular $A - \lambda B$. There is also a slightly less reliable $O(n^3)$ algorithm [27].

  - Application to ODEs: Provides solution of $B\dot{x}(t) = Ax(t) + f(t)$, where the solution may be overdetermined or underdetermined.
  - Software: NETLIB/linalg/guptri.

- Matrix polynomials $\sum_{i=0}^{d} \lambda^i A_i$ [118]
  - If $A_d = I$ (or A_d is square and well-conditioned enough to replace each A_i by $A_d^{-1}A_i$), then linearize to get the standard problem

$$\begin{bmatrix} -A_{d-1} & -A_{d-2} & \cdots & \cdots & \cdots & -A_0 \\ I & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & I & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I & 0 \end{bmatrix} - \lambda I.$$

  - If A_d is ill-conditioned or singular, linearize to get the pencil

$$\begin{bmatrix} -A_{d-1} & -A_{d-2} & \cdots & \cdots & \cdots & -A_0 \\ I & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & I & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I & 0 \end{bmatrix} - \lambda \begin{bmatrix} A_d & & & & \\ & I & & & \\ & & I & & \\ & & & \ddots & \\ & & & & I \end{bmatrix}.$$

4.7. References and Other Topics for Chapter 4

For a general discussion of properties of eigenvalues and eigenvectors, see [139]. For more details about perturbation theory of eigenvalues and eigenvectors, see [161, 237, 52], and chapter 4 of [10]. For a proof of Theorem 4.7, see [69]. For a discussion of Weierstrass and Kronecker canonical forms, see [110, 118]. For their application to systems and control theory, see [246, 247, 78]. For applications to computational geometry, graphics, and mechanical CAD, see [181, 182, 165]. For a discussion of parallel algorithms for the nonsymmetric eigenproblem, see [76].

4.8. Questions for Chapter 4

QUESTION 4.1. *(Easy)* Let A be defined as in equation (4.1). Show that $\det(A) = \prod_{i=1}^{b} \det(A_{ii})$ and then that $\det(A - \lambda I) = \prod_{i=1}^{b} \det(A_{ii} - \lambda I)$. Conclude that the set of eigenvalues of A is the union of the sets of eigenvalues of A_{11} through A_{bb}.

QUESTION 4.2. *(Medium; Z. Bai)* Suppose that A is *normal*; i.e., $AA^* = A^*A$. Show that if A is also triangular, it must be diagonal. Use this to show that an n-by-n matrix is normal if and only if it has n orthonormal eigenvectors. Hint: Show that A is normal if and only if its Schur form is normal.

QUESTION 4.3. *(Easy; Z. Bai)* Let λ and μ be distinct eigenvalues of A, let x be a right eigenvector for λ, and let y be a left eigenvector for μ. Show that x and y are orthogonal.

QUESTION 4.4. *(Medium)* Suppose A has distinct eigenvalues. Let $f(z) = \sum_{i=-\infty}^{+\infty} a_i z^i$ be a function which is defined at the eigenvalues of A. Let $Q^*AQ = T$ be the Schur form of A (so Q is unitary and T upper triangular).

1. Show that $f(A) = Qf(T)Q^*$. Thus to compute $f(A)$ it suffices to be able to compute $f(T)$. In the rest of the problem you will derive a simple recurrence formula for $f(T)$.

2. Show that $(f(T))_{ii} = f(T_{ii})$ so that the diagonal of $f(T)$ can be computed from the diagonal of T.

3. Show that $Tf(T) = f(T)T$.

4. From the last result, show that the ith superdiagonal of $f(T)$ can be computed from the $(i-1)$st and earlier subdiagonals. Thus, starting at the diagonal of $f(T)$, we can compute the first superdiagonal, second superdiagonal, and so on.

QUESTION 4.5. *(Easy)* Let A be a square matrix. Apply either Question 4.4 to the Schur form of A or equation (4.6) to the Jordan form of A to conclude that the eigenvalues of $f(A)$ are $f(\lambda_i)$, where the λ_i are the eigenvalues of A. This result is called the *spectral mapping theorem.*

This question is used in the proof of Theorem 6.5 and section 6.5.6.

QUESTION 4.6. *(Medium)* In this problem we will show how to solve the *Sylvester* or *Lyapunov* equation $AX - XB = C$, where X and C are m-by-n, A is m-by-m, and B is n-by-n. This is a system of mn linear equations for the entries of X.

1. Given the Schur decompositions of A and B, show how $AX - XB = C$ can be transformed into a similar system $A'Y - YB' = C'$, where A' and B' are upper triangular.

2. Show how to solve for the entries of Y one at a time by a process analogous to back substitution. What condition on the eigenvalues of A and B guarantees that the system of equations is nonsingular?

3. Show how to transform Y to get the solution X.

QUESTION 4.7. *(Medium)* Suppose that $T = \begin{bmatrix} A & C \\ 0 & B \end{bmatrix}$ is in Schur form. We want to find a matrix S so that $S^{-1}TS = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$. It turns out we can choose S of the form $\begin{bmatrix} I & R \\ 0 & I \end{bmatrix}$. Show how to solve for R.

QUESTION 4.8. *(Medium; Z. Bai)* Let A be m-by-n and B be n-by-m. Show that the matrices

$$\begin{pmatrix} AB & 0 \\ B & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 0 \\ B & BA \end{pmatrix}$$

are similar. Conclude that the nonzero eigenvalues of AB are the same as those of BA.

QUESTION 4.9. *(Medium; Z. Bai)* Let A be n-by-n with eigenvalues $\lambda_1, \ldots, \lambda_n$. Show that

$$\sum_{i=1}^{n} |\lambda_i|^2 = \min_{\det(S)\neq 0} \|S^{-1}AS\|_F^2.$$

QUESTION 4.10. *(Medium; Z. Bai)* Let A be an n-by-n matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$.

1. Show that A can be written $A = H + S$, where $H = H^*$ is Hermitian and $S = -S^*$ is skew-Hermitian. Give explicit formulas for H and S in terms of A.
2. Show that $\sum_{i=1}^{n} |\Re\lambda_i|^2 \leq \|H\|_F^2$.
3. Show that $\sum_{i=1}^{n} |\Im\lambda_i|^2 \leq \|S\|_F^2$.
4. Show that A is normal ($AA^* = A^*A$) if and only if $\sum_{i=1}^{n} |\lambda_i|^2 = \|A\|_F^2$.

QUESTION 4.11. *(Easy)* Let λ be a simple eigenvalue, and let x and y be right and left eigenvectors. We define the *spectral projection* P corresponding to λ as $P = xy^*/(y^*x)$. Prove that P has the following properties.

1. P is uniquely defined, even though we could use any nonzero scalar multiples of x and y in its definition.
2. $P^2 = P$. (Any matrix satisfying $P^2 = P$ is called a *projection matrix.*)
3. $AP = PA = \lambda P$. (These properties motivate the name *spectral* projection, since P "contains" the left and right invariant subspaces of λ.)
4. $\|P\|_2$ is the condition number of λ.

QUESTION 4.12. *(Easy; Z. Bai)* Let $A = \begin{bmatrix} a & c \\ 0 & b \end{bmatrix}$. Show that the condition numbers of the eigenvalues of A are both equal to $(1 + (\frac{c}{a-b})^2)^{1/2}$. Thus, the condition number is large if the difference $a-b$ between the eigenvalues is small compared to c, the offdiagonal part of the matrix.

QUESTION 4.13. *(Medium; Z. Bai)* Let A be a matrix, x be a unit vector ($\|x\|_2 = 1$), μ be a scalar, and $r = Ax - \mu x$. Show that there is a matrix E with $\|E\|_F = \|r\|_2$ such that $A + E$ has eigenvalue μ and eigenvector x.

QUESTION 4.14. *(Medium; Programming)* In this question we will use a Matlab program to plot eigenvalues of a perturbed matrix and their condition numbers. (It is available at HOMEPAGE/Matlab/eigscat.m.) The input is

a = input matrix,
err = size of perturbation,
m = number of perturbed matrices to compute.

The output consists of three plots in which each symbol is the location of an eigenvalue of a perturbed matrix:

"o" marks the location of each unperturbed eigenvalue.
"x" marks the location of each perturbed eigenvalue, where a real perturbation matrix of norm err is added to a.
"." marks the location of each perturbed eigenvalue, where a complex perturbation matrix of norm err is added to a.

A table of the eigenvalues of A and their condition numbers is also printed.

Here are some interesting examples to try (for as large an m as you want to wait; the larger the m the better, and m equal to a few hundred is good).

```
(1) a = randn(5) (if a does not have complex eigenvalues,
                  try again)
       err=1e-5, 1e-4, 1e-3, 1e-2, .1, .2

(2) a = diag(ones(4,1),1);  err=1e-12, 1e-10, 1e-8

(3) a=[[1 1e6 0     0]; ...
       [0 2   1e-3  0]; ...
       [0 0   3     10]; ...
       [0 0  -1      4]]
        err=1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3

(4) [q,r]=qr(randn(4,4));a=q*diag(ones(3,1),1)*q'
       err=1e-16, 1e-14, 1e-12, 1e-10, 1e-8

(5) a = [[1 1e3 1e6];[0 1 1e3];[0 0 1]],
       err=1e-7, 1e-6, 5e-6, 8e-6, 1e-5, 1.5e-5, 2e-5

(6) a = [[1   0   0   0   0   0]; ...
         [0   2   1   0   0   0]; ...
         [0   0   2   0   0   0]; ...
         [0   0   0   3 1e2 1e4]; ...
         [0   0   0   0   3 1e2]; ...
         [0   0   0   0   0   3]]
        err= 1e-10, 1e-8, 1e-6, 1e-4, 1e-3
```

Your assignment is to try these examples and compare the regions occupied by the eigenvalues (the so-called *pseudospectrum*) with the bounds described in section 4.3. What is the difference between real perturbations and complex perturbations? What happens to the regions occupied by the eigenvalues as the perturbation err goes to zero? What is limiting size of the regions as err goes to zero (i.e., how many digits of the computed eigenvalues are correct)?

QUESTION 4.15. *(Medium; Programming)* In this question we use a Matlab program to plot the diagonal entries of a matrix undergoing unshifted QR iteration. The values of each diagonal are plotted after each QR iteration, each diagonal corresponding to one of the plotted curves. (The program is available at HOMEPAGE/Matlab/qrplt.m and also shown below.) The inputs are

a = input matrix,
m = number of QR iterations,

and the output is a plot of the diagonals.

Examples to try this code on are as follows (choose m large enough so that the curves either converge or go into cycles):

```
a = randn(6);
b = randn(6); a = b*diag([1,2,3,4,5,6])*inv(b);
a = [[1 10];[-1 1]]; m = 300
a = diag((1.5*ones(1,5)).\verb+^+(0:4)) +
    .01*(diag(ones(4,1),1)+diag(ones(4,1),-1)); m=30
```

What happens if there are complex eigenvalues?

In what order do the eigenvalues appear in the matrix after many iterations?

Perform the following experiment: Suppose that a is n-by-n and symmetric. In Matlab, let perm=(n:-1:1). This produces a list of the integers from n down to 1. Run the iteration for m iterations. Let a=a(perm,perm); we call this "flipping" a, because it reverses the order of the rows and columns of a. Run the iteration again for m iterations, and again form a=a(perm,perm). How does this value of a compare with the original value of a? You should not let m be too large (try $m = 5$) or else roundoff will obscure the relationship you should see. (See also Corollary 5.4 and Question 5.25.)

Change the code to compute the error in each diagonal from its final value (do this just for matrices with all real eigenvalues). Plot the log of this error versus the iteration number. What do you get asymptotically?

```
hold off
e=diag(a);
for i=1:m,
   [q,r]=qr(a);dd=diag(sign(diag(r)));r=dd*r;q=q*dd;a=r*q; ...
   e=[e,diag(a)];
end
clg
plot(e','w'),grid
```

QUESTION 4.16. *(Hard; Programming)* This problem describes an application of the nonlinear eigenproblem to computer graphics, computational geometry, and mechanical CAD; see also [181, 182, 165].

Let $F = [f_{ij}(x_1, x_2, x_3)]$ be a matrix whose entries are polynomials in the three variables x_i. Then $\det(F) = 0$ will (generally) define a two-dimensional surface S in 3-space. Let $x_1 = g_1(t)$, $x_2 = g_2(t)$, and $x_3 = g_3(t)$ define a (one-dimensional) curve C parameterized by t, where the g_i are also polynomials. We want to find the intersection $S \cap C$. Show how to express this as an eigenvalue problem (which can then be solved numerically). More generally, explain how to find the intersection of a surface $\det(F(x_1, \ldots, x_n)) = 0$ and curve $\{x_i = g_i(t),\ 1 \le i \le n\}$. At most how many discrete solutions can there be, as a function of n, the dimension d of F, and the maximum of the degrees of the polynomials f_{ij} and g_k?

Write a Matlab program to solve this problem, for $n = 3$ variables, by converting it to an eigenvalue problem. It should take as input a compact description of the entries of each $f_{ij}(x_k)$ and $g_i(t)$ and produce a list of the intersection points. For instance, it could take the following inputs:

- Array NumTerms(1:d,1:d), where NumTerms(i,j) is the number of terms in the polynomial $f_{ij}(x_1, x_2, x_3)$.
- Array Sterms(1:4, 1:TotalTerms), where TotalTerms is the sum of all the entries in NumTerms(.,.). Each column of Sterms represents one term in one polynomial: The first NumTerms(1,1) columns of Sterms represent the terms in f_{11}, the second Numterm(2,1) columns of Sterms represent the terms in f_{21}, and so on. The term represented by Sterms(1:4,k) is $\mathrm{Sterm}(4,k) \cdot x_1^{\mathrm{Sterm}(1,k)} \cdot x_2^{\mathrm{Sterm}(2,k)} \cdot x_3^{\mathrm{Sterm}(3,k)}$.
- Array tC(1:3) contains the degrees of polynomials g_1, g_2, and g_3 in that order.
- Array Curve(1: tC(1)+tC(2)+tC(3)+3) contains the coefficients of the polynomials g_1, g_2, and g_3, one polynomial after the other, from the constant term to the highest order coefficient of each.

Your program should also compute error bounds for the computed answers. This will be possible only when the eigenproblem can be reduced to one for which the error bounds in Theorems 4.4 or 4.5 apply. You do not have to provide error bounds when the eigenproblem is a more general one. (For a description of error bounds for more general eigenproblems, see [10, 237].

Write a second Matlab program that plots S and C for the case $n = 3$ and marks the intersection points.

Are there any limitations on the input data for your codes to work? What happens if S and C do not intersect? What happens if S lies in C?

Run your codes on at least the following examples. You should be able to solve the first five by hand to check your code.

1. $g_1 = t,\ g_2 = 1 + t,\ g_3 = 2 + t,\ F = \begin{bmatrix} x_1 + x_2 + x_3 & 0 \\ 0 & 3x_1 + 5x_2 - 7x_3 + 10 \end{bmatrix}$.

2. $g_1 = t^3,\ g_2 = 1 + t^3,\ g_3 = 2 + t^3,\ F = \begin{bmatrix} x_1 + x_2 + x_3 & 0 \\ 0 & 3x_1 + 5x_2 - 7x_3 + 10 \end{bmatrix}$.

3. $g_1 = t^2,\ g_2 = 1 + t^2,\ g_3 = 2 + t^2,\ F = \begin{bmatrix} x_1 + x_2 + x_3 & 0 \\ 0 & 3x_1 + 5x_2 - 7x_3 + 10 \end{bmatrix}$.

4. $g_1 = t^2,\ g_2 = 1 + t^2,\ g_3 = 2 + t^2,\ F = \begin{bmatrix} 1 & 0 \\ 0 & 3x_1 + 5x_2 - 7x_3 + 9 \end{bmatrix}$.

5. $g_1 = t^2,\ g_2 = 1 + t^2,\ g_3 = 2 + t^2,\ F = \begin{bmatrix} x_1 + x_2 + x_3 & 0 \\ 0 & 3x_1 + 5x_2 - 7x_3 + 8 \end{bmatrix}$.

6. $g_1 = t^2,\ g_2 = 1 + t^2,\ g_3 = 2 + t^2,\ F = \begin{bmatrix} x_1 + x_2 + x_3 & x_1 \\ x_3 & 3x_1 + 5x_2 - 7x_3 + 10 \end{bmatrix}$.

7. $g_1 = 7 - 3t + t^5,\ g_2 = 1 + t^2 + t^5,\ g_3 = 2 + t^2 - t^5,$

$$F = \begin{bmatrix} x_1x_2 + x_3^5 & 3 - x_2^2 & 5 + x_1 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 \\ x_2 - 7x_3^5 & 1 - x_1^2 + x_1x_2x_3^3 & 3 + x_1 + 3x_3 - 9x_2x_3 \\ 2 & 3x_1 + 5x_2 - 7x_3 + 8 & x_1^3 - x_2^4 + 4x_3^5 \end{bmatrix}.$$

You should turn in

- mathematical formulation of the solution in terms of an eigenproblem.
- the algorithm in at most two pages, including a road map to your code (subroutine names for each high level operation). It should be easy to see how the mathematical formulation leads to the algorithm and how the algorithm matches the code.

— At most how many discrete solutions can there be?

— Do all compute eigenvalues represent actual intersections? Which ones do?

— What limits does your code place on the input for it to work correctly?

— What happens if S and C do not intersect?

— What happens if S contains C?

- mathematical formulation of the error bounds.
- the algorithm for computing the error bounds in at most two pages, including a road map to your code (subroutine names for each high-level operation). It should be easy to see how the mathematical formulation leads to the algorithm and how the algorithm matches the code.
- program listing.

For each of the seven examples, you should turn in

- the original statement of the problem.
- the resulting eigenproblem.
- the numerical solutions.
- plots of S and C; do your numerical solutions match the plots?
- the result of substituting the computed answers in the equations defining S and C: are they satisfied (to within roundoff)?

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.[17]

- 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 tiny eigenvalues (or singular values) more accurately than alternative

[17]There is yet more recent work [201, 203] on an algorithm based on inverse iteration (Algorithm 4.2), which may provide a still faster and more accurate algorithm. But as of June 1997 the theory and software were still under development.

algorithms like divide-and-conquer, although sometimes more slowly, in the sense of Jacobi.

- 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, type demo and then click on continue/ fun-extras/miscellaneous/bending. ◇

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, $\max_{u \neq 0} \rho(u, A)$, occurs for $u = q_1$ ($\xi = e_1$) and equals $\rho(q_1, A) = \alpha_1$.

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

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

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.*

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

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 $\alpha_1 = \max_{r \neq 0} \rho(r, A)$. 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 simplifies to $\alpha_1 = \max_{s \neq 0} \rho(s, A)$.

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

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

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

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

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

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

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

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

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

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 $\min_{0 \neq r \in \mathbf{R}^2} \rho(r, A) = \min_{r \in C} \rho(r, A)$. 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 and 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 $\min_{r \in C} \rho(r, A) < .25$.

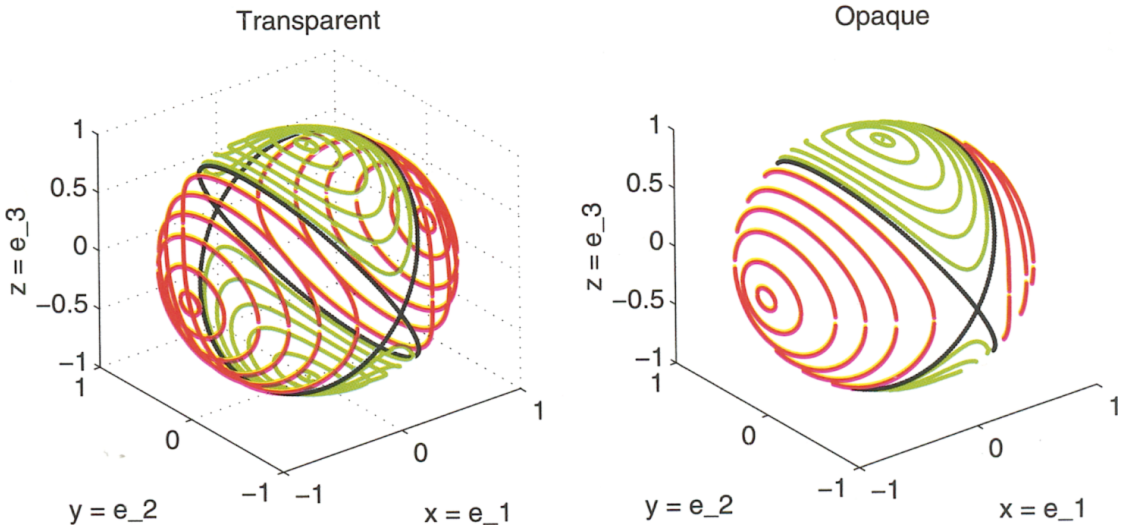


Fig. 5.1. *Contour plot of the Rayleigh quotient on the unit sphere.*

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), $\mathbf{R}^2 = \text{span}(q_1, q_2)$.

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. ◇

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

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

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

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^TAX and A are similar and so have the same eigenvalues. When X is only nonsingular, we say X^TAX and A are *congruent.* In this case X^TAX 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^TAX have the same inertia.*

Proof. Let n be the dimension of A. Now suppose that A has ν negative eigenvalues but that X^TAX 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 $x \in \mathbf{N}$, $x^TAx < 0$. Let $\mathbf{P}$ be the $(n - \nu')$-dimensional nonnegative eigenspace of X^TAX; this means that for any nonzero $x \in \mathbf{P}$, $x^TX^TAXx \geq 0$. 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 $0 > x^TAx$ since $x \in \mathbf{N}$ and $0 \leq x^TAx$ since $x \in X\mathbf{P}$, which is a contradiction. Therefore, $\nu = \nu'$. Reversing the roles of A and X^TAX, we also get $\nu' \leq \nu$; i.e., A and X^TAX 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 by perturbing A to $A + E$ 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* $\text{gap}(i, A) = \min_{j \neq i} |\alpha_j - \alpha_i|$. *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*

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

Similarly

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

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

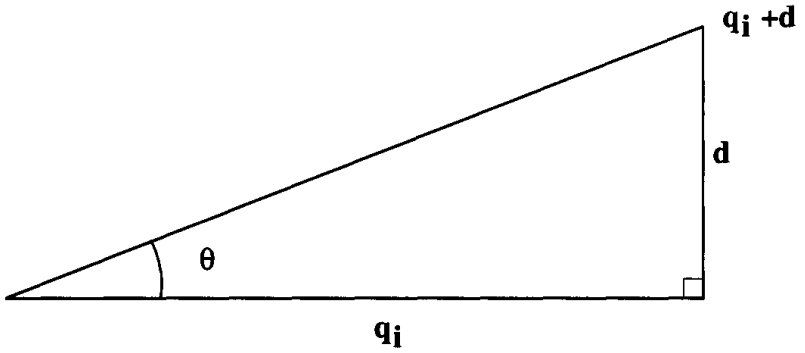
The attraction of stating the bound in terms of $\mathrm{gap}(i, A+E)$, as well as $\mathrm{gap}(i, A)$, is that frequently we know only the eigenvalues of $A+E$, since they are typically the output of the eigenvalue algorithm that we have used. In this case it is straightforward to evaluate $\mathrm{gap}(i,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

When the first upper bound exceeds $1/2$, i.e., $\|E\|_2 \geq \mathrm{gap}(i, A)/2$, the bound reduces to $\sin 2\theta \leq 1$, 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 $\hat{\alpha}_i$. 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 $z \equiv (\eta I - E)(q_i + d) = \sum_{j\neq i} \zeta_j q_j$ and $d \equiv \sum_{j\neq i} \delta_j q_j$. Since $(A - \alpha_i I)q_j = (\alpha_j - \alpha_i)q_j$, we can write

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

or

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

Thus

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

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 $\text{gap}' \equiv \min_{j \neq i} |\alpha_j - \rho(x, A)|$; 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

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

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 $\alpha_\pm = 1 + \frac{g}{2} \pm \frac{g}{2}(1 + (\frac{2\epsilon}{g})^2)^{1/2}$, 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 $\epsilon = 10^{-5}$. 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 far away. 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 [197] 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

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

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 $\alpha_i \neq 0$, 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 $\sigma_i \neq 0$, 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* $\text{rel\_gap}(i, A) = \min_{j \neq i} \frac{|\alpha_j - \alpha_i|}{|\alpha_i|}$.

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 ith 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 [101].

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 [161].

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`[18] 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 [131]. This super-fast implementation uses the fast multipole method (FMM) [124], 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

[18] 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 (besides, theoretically, divide-and-conquer with the FMM) [105, 203, 83, 201, 176, 173, 175, 269]. 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 [82]. 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 [197, 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 [197]. 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 [104, 200]) 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. See HOMEPAGE/Matlab/tridiQR.m.

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

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

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

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

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

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

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 $|\alpha_j - \alpha_1 + \eta| \geq \text{gap}(1, \Lambda) - |\eta|$, so using (5.12) as well we get

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

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. □

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 [59], the "right" implementation was not discovered until 1992 [127, 131]). 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

$$
\begin{aligned}
T &= \left[\begin{array}{cccc|ccccc}
a_1 & b_1 & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & \\
 & & b_{m-1} & a_m & b_m & & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & & \\
 & & & & b_{m+1} & \ddots & & & \\
 & & & & & & \ddots & b_{n-1} & \\
 & & & & & & b_{n-1} & a_n
\end{array}\right] \\
&= \left[\begin{array}{cccc|ccccc}
a_1 & b_1 & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & \\
 & & b_{m-1} & a_m - b_m & & & & & \\
\hline
 & & & & a_{m+1} - b_m & b_{m+1} & & & \\
 & & & & b_{m+1} & \ddots & & & \\
 & & & & & & \ddots & b_{n-1} & \\
 & & & & & & b_{n-1} & a_n
\end{array}\right] \\
&\quad + \left[\begin{array}{cccc|ccccc}
 & & & & & & & & \\
 & & & & & & & & \\
 & & & & & & & & \\
 & & & b_m & b_m & & & & \\
\hline
 & & & b_m & b_m & & & & \\
 & & & & & & & & \\
 & & & & & & & & \\
 & & & & & & & &
\end{array}\right] \\
&= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [0, \ldots, 0, 1, 1, 0, \ldots, 0] \equiv \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m v v^T.
\end{aligned}
$$

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:

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

Since $D - \lambda I$ is nonsingular, $\det(I + \rho (D - \lambda I)^{-1} u u^T) = 0$ whenever λ is an eigenvalue. Note that $I + \rho (D - \lambda I)^{-1} u u^T$ 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

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

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 $u_i \neq 0$ (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.)

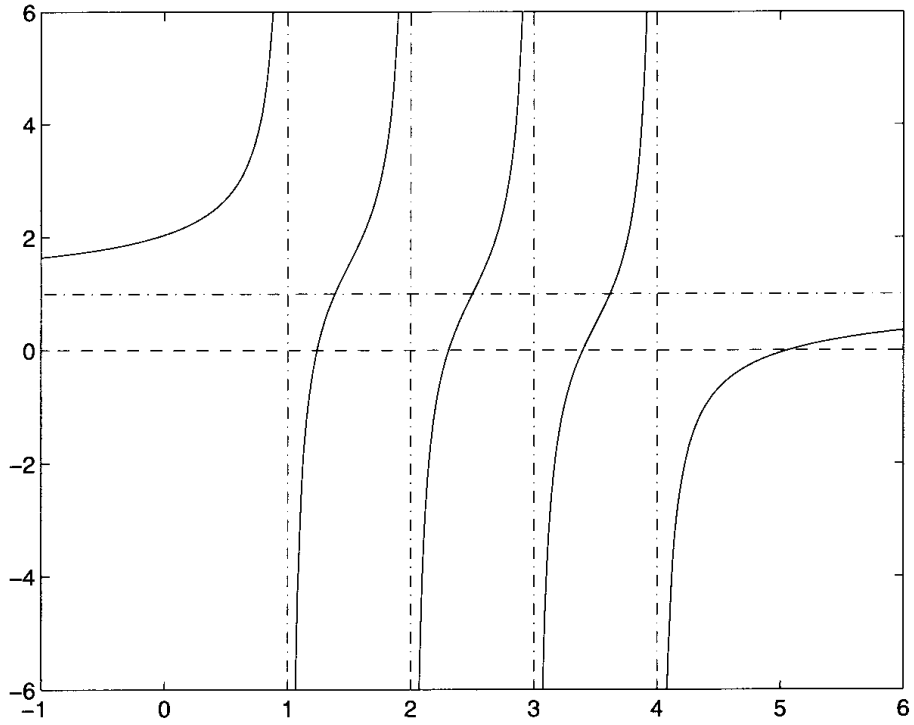


Fig. 5.2. *Graph of* $f(\lambda) = 1 + \frac{.5}{1-\lambda} + \frac{.5}{2-\lambda} + \frac{.5}{3-\lambda} + \frac{.5}{4-\lambda}$.

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.

$$\begin{aligned}
(D + \rho uu^T)[(D - \alpha I)^{-1}u] &= (D - \alpha I + \alpha I + \rho uu^T)(D - \alpha I)^{-1}u \\
&= u + \alpha(D - \alpha I)^{-1}u + u[\rho u^T(D - \alpha I)^{-1}u] \\
&= u + \alpha(D - \alpha I)^{-1}u - u \\
&\quad \text{since } \rho u^T(D - \alpha I)^{-1}u + 1 = f(\alpha) = 0 \\
&= \alpha[(D - \alpha I)^{-1}u] \quad \text{as desired.} \quad \square
\end{aligned}$$

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

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

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 [124]. 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 [59, 210].

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 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

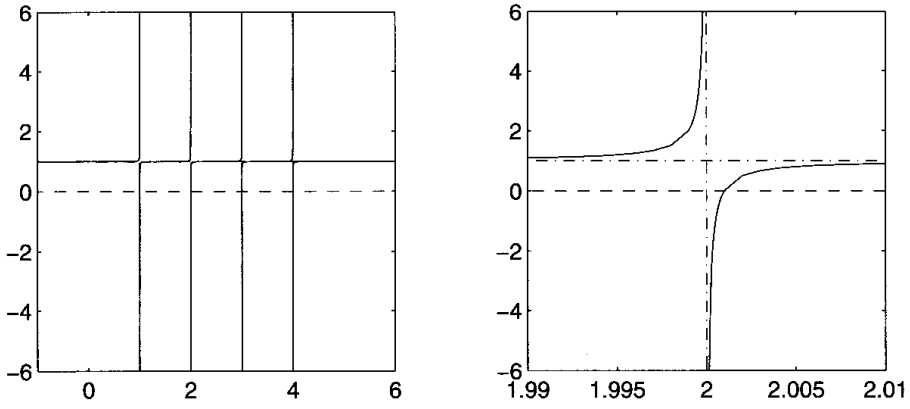


Fig. 5.3. *Graph of* $f(\lambda) = 1 + \frac{10^{-3}}{1-\lambda} + \frac{10^{-3}}{2-\lambda} + \frac{10^{-3}}{3-\lambda} + \frac{10^{-3}}{4-\lambda}$.

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` [172, 45]. 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 positive terms and $\psi_2(\lambda)$ is a sum of negative 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

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

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

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

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 [172]). The LAPACK routine averages two or 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 [59, 90, 234], 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 $(D - \alpha_{i+1} I)^{-1}u$ 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 [90, 234] 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 are 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 [234, 204]. 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 [129, 179].

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*

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

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

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

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

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

or

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

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,[19] 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

[19] 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.

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

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

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 [234, 204] from working on some Cray machines. 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 Cray machines are addition and subtraction, because of the lack of a so-called *guard digit* in the floating point hardware. This means that the bottom-most 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.[20]

This algorithm is described in more detail in [129, 131] and implemented in LAPACK routine `slaed3`.

[20]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). But on a Cray machine 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.

Accelerating Divide-and-Conquer Using the FMM

The FMM [124] 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 [131].

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[21]

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

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, $Q'_{ij} = u_i s_j / (d_i - \alpha_j)$, 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.

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

Evaluating this sum for $j = 1, \ldots, n$ appears to require $O(n^2)$ flops. The FMM and others like it [124, 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 $D + \rho uu^T$. 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 $\text{Inertia}(A) = (\nu, \zeta, \pi)$, 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 $\text{Inertia}(A) = \text{Inertia}(X^T A X)$.

Now suppose that one uses Gaussian elimination to factorize $A - zI = LDL^T$, where L is nonsingular and D diagonal. Then $\text{Inertia}(A - zI) = \text{Inertia}(D)$. 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 $n - n_a$ *through* $n - n_b + 1$, *which the algorithm*
        will repeatedly bisect until they are narrower than tol. */
while Worklist is not empty do
    remove $[low, n_{\text{low}}, up, n_{\text{up}}]$ *from Worklist*
    if (up−low < tol*) then*
        print "there are $n_{\text{up}} - n_{\text{low}}$ *eigenvalues in* $[low, up)$"
    else
        $mid = (low + up)/2$
        $n_{\text{mid}} = Negcount\ (A,\ mid)$
        if $n_{\text{mid}} > n_{\text{low}}$ *then ... there are eigenvalues in* $[low, mid)$
            put $[low, n_{\text{low}}, mid, n_{\text{mid}}]$ *onto Worklist*
        end if
        if $n_{\text{up}} > n_{\text{mid}}$ *then ... there are eigenvalues in* $[mid, up)$
            put $[mid, n_{\text{mid}}, up, n_{\text{up}}]$ *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 $\alpha_{n-n_{\text{low}}}$ through $\alpha_{n-n_{\text{up}}+1}$ 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:

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

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

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 $A - zI$ is tridiagonal, (5.17) can be shown to be very stable [73, 74, 156].

LEMMA 5.3. *The d_i computed in floating point arithmetic, using* (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:*

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

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 [73, 81].

The cost of a single call to Negcount on a tridiagonal matrix is at most $4n$ 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) [173, 174, 175, 176, 178, 269].

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 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 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 [105, 83, 201, 203], 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. 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 [76].

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

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

$$= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix},$$

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

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

Thus off(A) is the root-sum-of-squares of the (upper) offdiagonal entries of A, so A is diagonal if and only if off$(A) = 0$. Our goal is to make off(A) approach 0 quickly. The next lemma tells us that 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 $j \neq k$. Then* $\text{off}^2(A') = \text{off}^2(A) - a_{jk}^2$.

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

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

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 $\text{off}^2(A') = \text{off}^2(A) - a_{jk}^2$ 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 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* 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, $\text{off}^2(A') = \text{off}^2(A) - a_{jk}^2$, where a_{jk} is the largest offdiagonal entry. Thus $\text{off}^2(A) \leq \frac{n(n-1)}{2} a_{jk}^2$, or $a_{jk}^2 \geq \frac{1}{n(n-1)/2} \text{off}^2(A)$, so $\text{off}^2(A) - a_{jk}^2 \leq \left(1 - \frac{1}{N}\right) \text{off}^2(A)$ as desired. □

So the classical Jacobi's algorithm converges at least linearly with the error (measured by 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 [262, 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 labeled 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 [120, 80, 67].

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 are eigenvectors.
3. Combine these decompositions to get $A = (Q_1 Q_2)\Lambda(Q_1 Q_2)^T$. The columns of $Q = Q_1 Q_2$ 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: $G = U_1 B V_1^T$. 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 $G = (U_1 U_2)\Sigma(V_1 V_2)^T$. The columns of $U = U_1 U_2$ and $V = V_1 V_2$ 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;

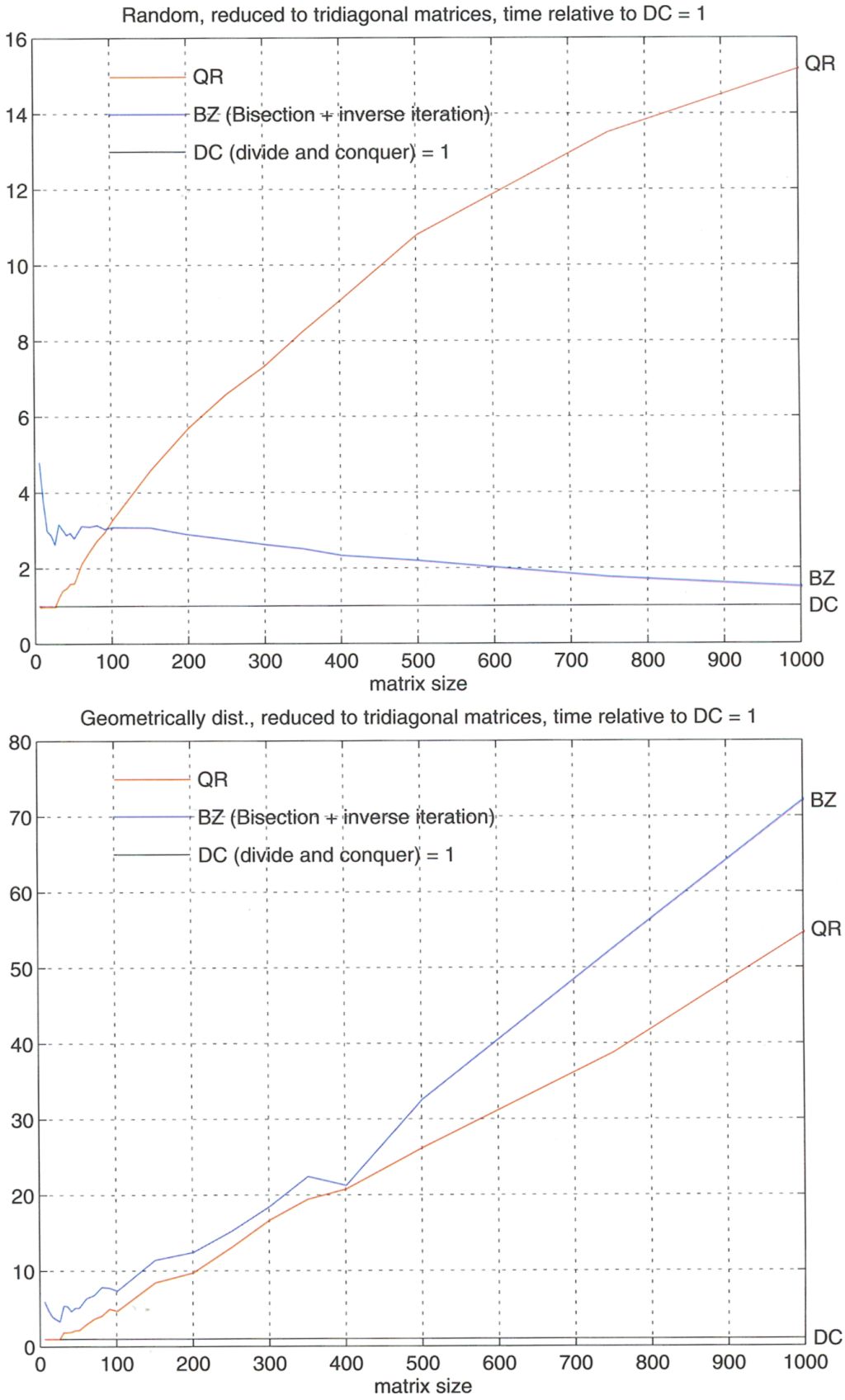


Fig. 5.4. *Speed of finding eigenvalues and eigenvectors of a symmetric tridiagonal matrix, relative to divide-and-conquer.*

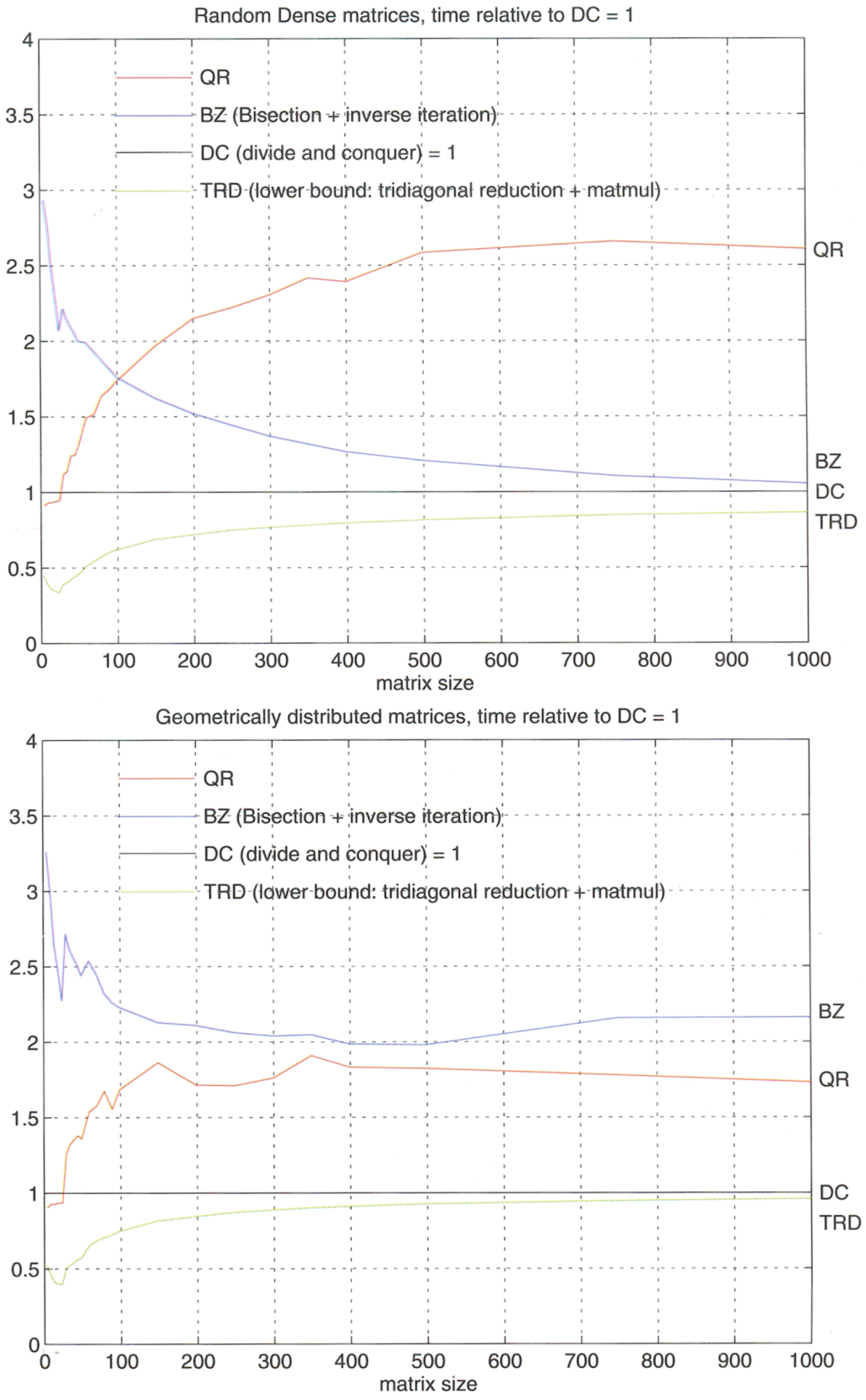


Fig. 5.5. *Speed of finding eigenvalues and eigenvectors of a symmetric dense matrix, relative to divide-and-conquer.*

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 $T_{ps} \equiv P^T A P$ 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}.*

3. *Let $T_{B^TB} \equiv B^T B$. 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 superdiagonal 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 [104], 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 [80] 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 is 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 [130] 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 [200] for a good survey. The algorithm in the LAPACK routine `sbdsqr` was originally based on [80] and later updated to use the algorithm in [104] in the case when singular values only are desired. This latter algorithm, called dqds for historical reasons,[22] 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

[22] dqds is short for "differential quotient-difference algorithm with shifts" [209].

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

$$\begin{aligned}
T' &= RQ = RQ(RR^{-1}) = R(QR)R^{-1} = RT_0R^{-1} \quad \text{because } T_0 = QR \\
&= (B_1B_0)(B_0^TB_0)(B_1B_0)^{-1} \quad \text{because } R = B_1B_0 \text{ and } T_0 = B_0^TB_0 \\
&= B_1B_0B_0^TB_0B_0^{-1}B_1^{-1} = B_1(B_0B_0^T)B_1^{-1} \\
&= B_1(B_1^TB_1)B_1^{-1} \quad \text{because } B_0B_0^T = T_1 = B_1^TB_1 \\
&= B_1B_1^T \\
&= T_2 \text{ as desired.} \quad \square
\end{aligned}$$

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 $T_{i+1} = B_i B_i^T + \tau_i^2 I$, 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

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

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

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

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

for $j = 1$ to $n-1$
  $\hat{a}_j^2 = a_j^2 + b_j^2 - \hat{b}_{j-1}^2 - \delta$
  $\hat{b}_j^2 = b_j^2 \cdot (a_{j+1}^2 / \hat{a}_j^2)$
end for
$\hat{a}_n^2 = a_n^2 - \hat{b}_{n-1}^2 - \delta$

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 [209]).

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:

$$\begin{aligned} d_j &\equiv q_j - \hat{e}_{j-1} - \delta \\ &= q_j - \frac{q_j e_{j-1}}{\hat{q}_{j-1}} - \delta \quad \text{from} \quad (5.22) \end{aligned}$$

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

This lets us rewrite the inner loop of Algorithm 5.10 as

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

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 [104].

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* $\sigma_i \neq 0$ *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 [104].

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 $\tilde{q}_{j+1} = \tilde{a}^2_{j+1,j+1}$).

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

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

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's 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's method 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's 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*
    $\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$
    $t = \text{sign}(\tau)/(|\tau| + \sqrt{1 + \tau^2})$
    $c = 1/\sqrt{1 + t^2}$
    $s = c \cdot t$
    $G = G \cdot R(j, k, \theta)$ ... *where* $c = \cos\theta$ *and* $s = \sin\theta$
    if right singular vectors are desired

```
            J = J · R(j, k, θ)
        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 σ_i = ||G(:, i)||_2 (the 2-norm of column i of G)
Let U = [u_1, ..., u_n], where u_i = G(:, i)/σ_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 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 $\eta = 10^{-20}$ (nearly 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 in 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 difficulty 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 [75].

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 [82, 92, 183]).

   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

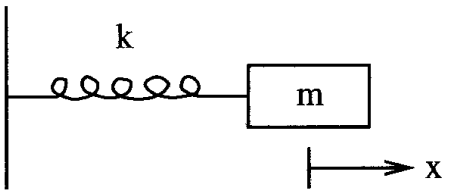
$$\hat{T} = \begin{bmatrix} 1 & \sqrt{2\eta} & \\ \sqrt{2\eta} & .5 & .5 \\ & .5 & .5 \end{bmatrix},$$

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. The most general situation in which we understand how to compute the SVD of A to high relative accuracy is when we can accurately compute *any* factorization $A = YDX$, where X and Y are well-conditioned but otherwise arbitrary and D is diagonal. In the last example we had $L = DX$; i.e., Y was the identity matrix. Gaussian elimination with complete pivoting is another source of such factorizations (with Y lower triangular and X upper triangular). For details, see [74]. For applications of this idea to indefinite symmetric eigenproblems, see [228, 250], and for generalized symmetric eigenvalue problems, see [66, 92]

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, like "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_- denote the strictly lower triangle of X, and $\pi_0(X) = X_- - X_-^T$.*

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.

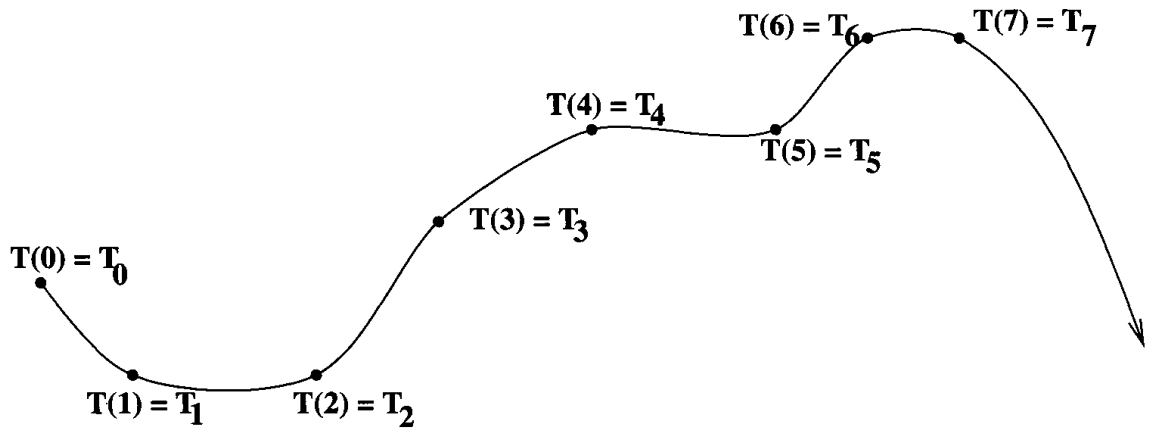
DEFINITION 5.6. *Choosing $F(x) = \log x$ in equation* (5.28) *yields a differential equation called the* QR flow.

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.*[23]

[23]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 $T_i = T(i)$.

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 indicate that they are equal to the QR iterates T_i.



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

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

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

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

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 [144, 170, 67, 68, 239] and papers by Kruskal [166], Flaschka [106], and Moser [187] in [188].

5.6. References and Other Topics for Chapter 5

An excellent general reference for the symmetric eigenproblem is [197]. The material on relative perturbation theory can be found in [75, 82, 101]; section 5.2.1 was based on the latter of these references. Related work is found in [66, 92, 228, 250]. A classical text on perturbation theory for general linear operators is [161]. For a survey of parallel algorithms for the symmetric eigenproblem, see [76]. The QR algorithm for finding the SVD of bidiagonal matrices is discussed in [80, 67, 120], and the dqds algorithm is in [104, 200, 209]. For an error analysis of the Bisection algorithm, see [73, 74, 156], and for recent attempts to accelerate Bisection see [105, 203, 201, 176, 173, 175, 269]. Current work in improving inverse iteration appears in [105, 83, 201, 203]. The divide-and-conquer eigenroutine was introduced in [59] and further developed in [13, 90, 127, 131, 153, 172, 210, 234]. The possibility of high-accuracy eigenvalues obtained from Jacobi is discussed in [66, 75, 82, 92, 183, 228]. The Toda flow and related phenomena are discussed in [67, 68, 106, 144, 166, 170, 187, 188, 239].

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 $\alpha_3 \leq \alpha_2 \leq \alpha_1$. 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 $\alpha_n \leq \cdots \leq \alpha_1$ be the eigenvalues of A and $\theta_{n-1} \leq \cdots \leq \theta_1$ be the eigenvalues of H. Show that these two sets of eigenvalues *interlace*:

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

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

QUESTION 5.5. *(Medium)* Let $A = A^T$ with eigenvalues $\alpha_1 \geq \cdots \geq \alpha_n$. Let $H = H^T$ with eigenvalues $\theta_1 \geq \cdots \geq \theta_n$. Let $A + H$ have eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$. Use the Courant–Fischer minimax theorem (Theorem 5.2) to show that $\alpha_j + \theta_n \leq \lambda_j \leq \alpha_j + \theta_1$. 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 $\sigma_1 \geq \cdots \geq \sigma_n$ be the singular values of A and $\tau_1 \geq \cdots \geq \tau_m$ 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 $\sigma_j \geq \tau_j \geq \sigma_{j+n-m}$.

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 $\hat{L}^T\hat{L} = Y^T L^T L Y$, where $\|Y^T Y - I\|_2 \leq O(\varepsilon)\kappa(\bar{A})$. Use Theorem 5.6 to conclude that the eigenvalues of $\hat{L}^T\hat{L}$ and of $L^T L$ 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 $A = G^T G$, where G and A are n-by-n. Suppose that $|a_{jk}| \leq \epsilon\sqrt{a_{jj}a_{kk}}$ for all $j \neq k$. 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 $T = T_0 = B_0^T B_0$ be the Cholesky decomposition, $T_1 = B_0 B_0^T = B_1^T B_1$, and more generally $T_i = B_{i-1}B_{i-1}^T = B_i^T B_i$. Let $\hat{T}_i$ denote the matrix obtained from T_0 after i steps of unshifted QR iteration; i.e., if $\hat{T}_i = Q_i R_i$ is the QR decomposition, then $\hat{T}_{i+1} = R_i Q_i$. 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 $T_0^i = (B_i B_{i-1}\cdots B_0)^T (B_i B_{i-1}\cdots B_0)$ 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 that 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.