Lecture Notes

Numerical Methods for Eigenvalue Problems

Davoud Mirzaei

Department of IT, Uppsala University

August 31, 2026

Contents

1 Introduction

In many scientific and engineering applications, one is interested not only in solving a linear system of equations but also in computing the eigenvalues and eigenvectors of a matrix. Given a matrix $A \in \mathbb{C}^{n\times n}$, the eigenvalue problem consists of finding a scalar $\lambda \in \mathbb{C}$ and a nonzero vector $x \in \mathbb{C}^n$ such that

$$Ax = \lambda x. \tag{1.1}$$

The scalar λ is called an *eigenvalue* of A, and the corresponding nonzero vector x is called an *eigenvector*. If $x \neq 0$ is an eigenvector corresponding to the eigenvalue λ, then αx is also an eigenvector corresponding to λ for every nonzero scalar α. Thus, eigenvectors are determined only up to a nonzero multiplicative constant. For this reason, we usually normalize eigenvectors, typically so that $\|x\| = 1$ for a specific norm.

Eigenvalue problems arise naturally in a wide variety of applications. For example, in structural engineering, eigenvalues determine the natural frequencies of vibration of a structure. In quantum mechanics, the energy levels of a physical system are obtained from eigenvalue problems involving differential operators. In fluid dynamics, stability analysis often reduces to the computation of a few dominant eigenvalues. In data science and machine learning, principal component analysis (PCA) is based on the computation of eigenvectors of covariance matrices. Similarly, in graph theory, the eigenvalues of adjacency and Laplacian matrices reveal important properties of networks.

From (1.1) we have $(A - \lambda I)x = 0$, with $x \neq 0$. Therefore, the matrix $A - \lambda I$ must be singular, which is equivalent to

$$p_n(\lambda) := \det(A - \lambda I) = 0. \tag{1.2}$$

It is straightforward to show that p_n is a polynomial of degree n. Conversely, if for a given $\lambda \in \mathbb{C}$ the relation (1.2) holds, then $A - \lambda I$ is singular, hence there exists a nonzero vector $x \in \mathbb{C}^n$ such that $(A - \lambda I)x = 0$ or equivalently $Ax = \lambda x$. Thus, the eigenvalues of A are exactly the roots of the polynomial p_n, which is called the *characteristic polynomial* of A. The equation $p_n(\lambda) = 0$ is called the *characteristic equation* of A. By the Fundamental Theorem of Algebra, every polynomial of degree n (with real or complex coefficients) has exactly n roots in $\mathbb{C}$, some of which may be repeated. Consequently, an $n \times n$ matrix A has exactly n eigenvalues. If A is a real matrix, i.e. if $A \in \mathbb{R}^{n\times n}$, then p_n has real coefficients. But the roots of real polynomials are not necessarily real, so a real matrix can have complex eigenvalues. Since eigenvalues may be complex regardless of whether A is real or complex, it makes sense to assume from the beginning that $A \in \mathbb{C}^{n\times n}$.

In some parts of this lecture, we may consider only the real matrices. Notice that if a real matrix has complex eigenvalues, they must occur in conjugate pairs, that is if $\lambda = \alpha + i\beta$ is an eigenvalue of $A \in \mathbb{R}^{n\times n}$ then so is the complex conjugate $\bar{\lambda} = \alpha - i\beta$. The reason is clear: if $p_n(\lambda) = 0$ then $p_n(\bar{\lambda}) = 0$ provided that p_n has real coefficients.

The nonlinear characteristic equation (1.2) also shows that computing eigenvalues is inherently a *nonlinear* problem for $n > 1$. Once an eigenvalue λ has been found, a corresponding eigenvector can be obtained by computing a nonzero solution of the singular linear system $(A - \lambda I)x = 0$.

Exercise 1.1. Show that if $A \in \mathbb{C}^{n\times n}$ is a (lower or upper) triangular matrix then eigenvalues of T are main diagonal entries $a_{11}, a_{22}, \ldots, a_{nn}$.

Exercise 1.2. Assume that A is an upper block triangular matrix of the form

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1s} \\ & A_{22} & \cdots & A_{2s} \\ & & \ddots & \vdots \\ & & & A_{ss} \end{bmatrix}$$

with square diagonal blocks A_{jj}, $j = 1, \ldots, s$. Show that the set of eigenvalues of A counting algebraic multiplicity, equals the union of sets of eigenvalues of diagonal blocks $A_{11}, A_{22}, \ldots, A_{ss}$. The same holds true for a lower block triangular matrix.

Exercise 1.3. Assume that $A \in \mathbb{C}^{n\times n}$. (a) Use the characteristic equation to show that A and A^T have the same eigenvalues. (b) Show that λ is en eigenvalue of A if and only if $\bar{\lambda}$ is an eigenvalue of A^H (the complex conjugate or Hermitian transpose of A, i.e. $A^H = \bar{A}^T$).

A matrix A is called *Hermitian* if $A = A^H$. If A is a real matrix then $A^H = A^T$ and the condition reduces to $A = A^T$ and the matrix is called *symmetric*.

Exercise 1.4. Prove that if $A \in \mathbb{C}^{n\times n}$ is a Hermitian matrix then eigenvalues of A are all real, and eigenvectors form an orthonormal set in $\mathbb{C}^n$ (a basis for $\mathbb{C}^n$). In a special case, if A is a real symmetric matrix, eigenvalues are all real and eigenvectors form an orthonormal set in $\mathbb{R}^n$ (a basis for $\mathbb{R}^n$).

Although the characteristic polynomial provides a complete theoretical characterization of the eigenvalues, it is rarely used in practical computations because of numerical stability issues. For a simple illustration, consider the 2×2 identity matrix

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

The characteristic equation of A is $p_2(\lambda) = \lambda^2 + 2\lambda + 1 = 0$ whose roots are $\lambda_1 = 1$ and $\lambda_2 = 1$. Now suppose that one coefficient of the characteristic polynomial is perturbed slightly and consider the equation

$$\lambda^2 + 2\lambda + (1 - \epsilon) = 0$$

for $\epsilon \ll 1$. The roots of the perturbed equation are $\tilde{\lambda}_1 = 1 + \sqrt{\epsilon}$ and $\tilde{\lambda}_2 = 1 - \sqrt{\epsilon}$ which shows that $|\lambda_j - \tilde{\lambda}_j| = \sqrt{\epsilon}$. For example, if the coefficient is perturbed by $\epsilon = \texttt{eps} \approx 10^{-16}$, then the roots are perturbed by $\sqrt{\epsilon} \approx 10^{-8}$, an amplification by a factor of one hundred million! This happens for the simple identity matrix of size 2×2. For larger matrices and more complicated polynomials, the sensitivity can be much more severe. In general, polynomial root-finding is a numerically delicate problem, especially when the degree is large, or there are roots with multiplicities, or for polynomials with tightly clustered roots. In contrast, the eigenvalue problem for identity matrix is perfectly well-conditioned. Small perturbations in

entries of A cause small perturbations in eigenvalues. This is true not only for the identity matrix but also for every symmetric matrix, regardless of how eigenvalues are distributed.

In other words, computing eigenvalues via the characteristic polynomial means reformulating a generally well-conditioned matrix problem as a potentially ill-conditioned polynomial root-finding problem. In numerical linear algebra we use iterative algorithms (because of nonlinearity) that work directly with matrix A rather than with its characteristic polynomial p_n. In fact, the opposite approach is often taken: to compute the roots of a polynomial, one constructs a matrix whose characteristic polynomial is the given polynomial (the so-called *companion matrix*) and then computes its eigenvalues using robust matrix algorithms.

For small matrices, all eigenvalues can be computed using direct methods such as the QR or Francis algorithm. However, in many applications the matrix dimension may be in the thousands or millions, making direct methods prohibitively expensive. In such situations one is usually interested in only a few eigenvalues, typically those of largest magnitude, smallest magnitude, or those closest to a prescribed target. This motivates the development of Krylov subspace methods for eigenvalue computations.

2 The power method

The simplest iterative method for computing an eigenvalue is the *power method*. It is designed to approximate the eigenvalue of largest magnitude together with its corresponding eigenvector. Suppose that A has eigenvalues $\lambda_1, \ldots, \lambda_n$ with property

$$|\lambda_1| > |\lambda_2| \geqslant \cdots \geqslant |\lambda_n|,$$

with corresponding eigenvectors $v_1, \ldots, v_n$. In this case, λ_1 is called the *dominant eigenvalue* and v_1 the *dominant eigenvector*, and both are called the *dominant eigenpair* of A. To approximate the dominant eigenpair, the power method starts from an initial vector $x_0 \neq 0$, and repeatedly multiplies by A,

$$x_m = Ax_{m-1}.$$

To avoid overflow or underflow, the iterates are normalized at each step.

Algorithm 1: Power Method

Input: Matrix $A \in \mathbb{C}^{n\times n}$, initial vector $x_0 \neq 0$, tolerance ε, maximum iteration $m_{\max}$.
Output: Approximate dominant eigenvalue λ_1 and eigenvector v_1, indicator $flag$.

Normalize $x_0 \leftarrow x_0/\|x_0\|_2$, $\ell_0 = x_0^H A x_0$, $flag = 0$.
for $m = 1, 2, \ldots, m_{\max}$ **do**
  Compute $y = Ax_{m-1}$
  Set $x_m = y/\|y\|_2$
  Compute the Rayleigh quotient $\ell_m = x_m^H A x_m$
  if $|\ell_m - \ell_{m-1}|/|\ell_{m-1}| < \varepsilon$ **and** $\|Ax_m - \ell_m x_m\|_2/\|Ax_m\|_2 < \varepsilon$ **then** Stop
end for
$v_1 = x_m$, $\lambda_1 = \ell_m$
if $m = m_{\max}$ **then** $flag = 1$.

The algorithm returns the indicator $flag = 1$ if it fails to converge in $m_{\max}$ iterations to the prescribed accuracy ε.

We note that if A is a real matrix, the assumption $|\lambda_1| > |\lambda_j|$ for all $j \geq 2$ implies that λ_1 is necessarily a real number, and hence v_1 is a real vector. Therefore, we start the power method with a real initial vector x_0 and all subsequent iterates x_m remain real. In contrast, when A is a complex matrix, λ_1 and v_1 may be either real or complex. In this case, the initial vector x_0 can be chosen either real or complex.

It is also nothing to show that if x_m is an approximation to the dominant eigenvector v_1 and is normalized so that $x_m^H x_m = 1$, then

$$\ell_m = x_m^H A x_m$$

provides an approximation to the dominant eigenvalue λ_1. The quantity ℓ_m is known as the *Rayleigh quotient* of x_m. Here, the denominator of the quotient is $x_m^H x_m = 1$, so it is removed.

The matrix–vector product appears twice in Algorithm 1. However, only one matrix–vector multiplication per iteration is actually needed. Indeed, after computing $y = Ax_m$, the Rayleigh quotient can be evaluated as $\ell_m = x_m^H y$, and the vector y is then reused in the subsequent step.

To understand why the method works, we first observe that the $(m+1)$-st iterate produced by the power method (Algorithm (1)) can be written as

$$x_{m+1} = \frac{A^m x_0}{\|A^m x_0\|_2}. \tag{2.1}$$

For simplicity, assume that A is *diagonalizable*. Then its eigenvectors $v_1, v_2, \ldots, v_n$ form a linearly independent set and hence a basis for $\mathbb{C}^n$. We can therefore expand the initial vector x_0 in this basis:

$$x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n.$$

We further assume that $c_1 \neq 0$, meaning that x_0 has a nonzero component in the direction of the dominant eigenvector. This assumption is very mild. If x_0 is chosen randomly, it is satisfied with probability almost one. Applying A^m to both sides gives

$$A^m x_0 = c_1 \lambda_1^m v_1 + c_2 \lambda_2^m v_2 + \cdots + c_n \lambda_n^m v_n = \lambda_1^m \left[c_1 v_1 + c_2 \Big(\frac{\lambda_2}{\lambda_1}\Big)^m v_2 + \cdots + c_n \Big(\frac{\lambda_n}{\lambda_1}\Big)^m v_n \right].$$

Substituting this expression into (2.1), we obtain

$$x_{m+1} = \frac{\lambda_1^m \left[c_1 v_1 + c_2 \Big(\frac{\lambda_2}{\lambda_1}\Big)^m v_2 + \cdots + c_n \Big(\frac{\lambda_n}{\lambda_1}\Big)^m v_n \right]}{|\lambda_1|^m \left\| c_1 v_1 + c_2 \Big(\frac{\lambda_2}{\lambda_1}\Big)^m v_2 + \cdots + c_n \Big(\frac{\lambda_n}{\lambda_1}\Big)^m v_n \right\|_2}.$$

Since

$$\left| \frac{\lambda_j}{\lambda_1} \right| < 1, \quad j = 2, \ldots, n,$$

all terms inside the brackets in the numerator, as well as the corresponding terms inside the norm in the denominator, decay geometrically as m increases. Consequently,

$$x_{m+1} \to \pm \frac{v_1}{|v_1|_2}.$$

This establishes the following theorem. However, the assumption that A is diagonalizable is not required and can be removed. The general case can be proved using the Jordan canonical form of A, which shows that the contributions associated with the remaining eigenvalues decay relative to that of the dominant eigenvalue. Since the proof does not provide additional insight into the main idea of the method, we omit it here.

Theorem 2.1. Let the eigenvalues of matrix $A \in \mathbb{C}^{n\times n}$ satisfy

$$|\lambda_1| > |\lambda_2| \geqslant \cdots \geqslant |\lambda_n|.$$

If the initial vector x_0 has a nonzero component in the direction of the dominant eigenvector v_1, then the normalized iterates produced by the power method converge to v_1 up to a sign. Moreover,

$$\|x_m - v_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^m\right).$$

The ratio $\rho = |\lambda_2/\lambda_1|$ therefore determines the asymptotic rate of convergence. If this ratio is close to one, the convergence can be very slow.

Definition 2.1. A sequence $\{x_m\}$, $m = 0, 1, \ldots$ that converges to x is said to converge linearly if there is a number ρ with $0 < \rho < 1$ such that

$$\lim_{m\to\infty} \frac{\|x_{m+1} - x\|}{\|x_m - x\|} = \rho,$$

in some norm $\|\cdot\|$ that we take it $\|\cdot\|_2$ here.

The linear convergence means that for sufficiently large values of m we have $\|x_{m+1} - x\| \approx \rho\,\|x_m - x\|$, the error in the current iteration is ρ times the error in the previous iteration. The number ρ is called the *convergence ratio* of the sequence. The power method, in general, exhibits the linear convergence with convergence ratio $\rho = |\lambda_2|/|\lambda_1|$. For example, if $\rho = 0.1$, the method adds approximately one decimal accuracy at each iteration. If $\rho = 1/\sqrt{10}$, one sdditional decimal accuracy is gained after two iterations.

2.1 Inverse power method

The power method computes the eigenvalue of largest magnitude. To compute eigenvalues of small magnitude, we apply the same idea to the inverse matrix A^{-1}. Indeed, if $Av_j = \lambda_j v_j$, and $\lambda_j \neq 0$ (which is the case when A is nonsingular) then

$$A^{-1}v_j = \frac{1}{\lambda_j}v_j, \quad j = 1, \ldots, n.$$

Thus the eigenvalues of A^{-1} are $1/\lambda_j$ corresponding to the same eigenvectors v_j. The eigenvalue of A having the smallest magnitude corresponds to the dominant eigenvalue of A^{-1}. We assume

$$|\lambda_1| \geqslant \cdots \geqslant |\lambda_{n-1}| > |\lambda_n|$$

so that the only strict inequality appears at the end of the sequence. We apply the power method on A^{-1} instead of A. The algorithm is named *inverse* power method. Since explicitly forming A^{-1} is not practical, each iteration requires the solution of a linear system.

Algorithm 2: Inverse Power Method

Input: Nonsingular matrix $A \in \mathbb{C}^{n\times n}$, initial vector $x_0 \neq 0$, tolerance ε, maximum iteration $m_{\max}$.
Output: Approximation of eigenvalue λ_n of smallest magnitude and eigenvector v_n, indicator $flag$.

Normalize $x_0 \leftarrow x_0/\|x_0\|_2$, Solve $Ay = x_0$, $\ell_0 = x_0^H A x_0$, $flag = 0$.
for $m = 1, 2, \ldots, m_{\max}$ **do**
  Set $x_m = y/\|y\|_2$
  Compute the Rayleigh quotient $\ell_m = x_m^H A x_m$
  if $|\ell_m - \ell_{m-1}|/|\ell_{m-1}| < \varepsilon$ **and** $\|Ax_m - \ell_m x_m\|_2/\|Ax_m\|_2 < \varepsilon$ **then** Stop
  Solve $Ay = x_m$
end for
$v_n = x_m$, $\lambda_n = \ell_m$
if $m = m_{\max}$ **then** $flag = 1$.

The convergence theory is identical to that of the power method. The convergence is linear and the convergence factor is determined by the ratio $\rho = |\lambda_n/\lambda_{n-1}|$ of the two smallest eigenvalues in magnitude.

In the linear systems $Ay = x_m$, the coefficient matrix A is fixed in all iterations. If a direct solver is used, an LU factorization (or a Cholesky factorization in the symmetric positive definite case) can be computed once before the iteration starts. The factors are then reused at every iteration, so that each step requires only forward and backward substitutions. In practice, the initial factorization usually dominates the computational cost, while the subsequent iterations are inexpensive. If A is large and sparse, a direct factorization may be too expensive. In this case, an effective alternative is to compute an incomplete LU (ILU) factorization of A before the iteration starts and use it as a preconditioner for solving the linear systems $Ay = x_m$ using an appropriate iterative method, such as GMRES or BiCG for nonsymmetric matrices, or CG when A is symmetric positive definite.

2.2 Shifted inverse iteration

In practice, one is often interested in an eigenvalue near a prescribed target value $\mu \in \mathbb{C}$. Since the eigenvalues of $(A - \mu I)^{-1}$ are

$$\frac{1}{\lambda_j - \mu},$$

the eigenvalue λ_j closest to μ produces the largest magnitude eigenvalue of $(A - \mu I)^{-1}$. Consequently, applying the inverse power method to $A - \mu I$ causes the iteration to converge to the eigenvector associated with the eigenvalue nearest to μ. The convergence factor is

$$\left|\frac{\lambda - \mu}{\lambda_* - \mu}\right|,$$

where λ denotes the eigenvalue closest to μ and λ_* denotes the second closest. If μ is chosen very close to the desired eigenvalue, this ratio becomes very small and convergence can be dramatically accelerated.

As we can easily conclude, this idea can be used to compute an eigenvector associated with an eigenvalue for which a good approximation is already available by other means. In such situations, the convergence factor is extremely small, and the desired eigenvector can often be obtained in only one or two iterations.

The algorithm of sifted inverse iteration is given in 3 in the following.

Shifts can also be used in conjunction with the direct iteration (power method), but it often results in a few acceleration. The combination of shifting and inversion is much more effective because one can choose the shift μ so that an eigenvalue of $(A-\mu I)^{-1}$ becomes much larger in magnitude than all the others. In contrast, for the shifted matrix $A-\mu I$, it is usually not possible to choose μ so that one eigenvalue dominates all the others in magnitude.

Shifted inverse iteration is one of the most important ideas in numerical eigenvalue computations. It forms the basis of several advanced algorithms, including modern Krylov subspace methods for large-scale eigenvalue problems.

Algorithm 3: Shifted Inverse Iteration

Input: Matrix $A \in \mathbb{C}^{n\times n}$, Approximate eigenvalue μ (the shift); initial vector $x_0 \neq 0$, tolerance ε, maximum iteration $m_{\max}$.

Output: More accurate eigenvalue λ (closet eigenvalue to μ); eigenvector v, indicator $flag$.

Normalize $x_0 \leftarrow x_0/\|x_0\|_2$, Solve $(A-\mu I)y = x_0$, $\ell_0 = x_0^H A x_0$, $flag = 0$.
for $m = 1, 2, \ldots, m_{\max}$ **do**
  Set $x_m = y/\|y\|_2$
  Compute the Rayleigh quotient $\ell_m = x_m^H A x_m$
  if $|\ell_m - \ell_{m-1}|/|\ell_{m-1}| < \varepsilon$ **and** $\|Ax_m - \ell_m x_m\|_2/\|Ax_m\|_2 < \varepsilon$ **then** Stop
  Solve $(A-\mu I)y = x_m$
end for
$v = x_m$, $\lambda = \ell_m$
if $m = m_{\max}$ **then** $flag = 1$.

2.3 Rayleigh quotient inverse iteration

The next question is how to choose the shift parameter μ. Ideally, μ should be close to the desired eigenvalue λ. If a sufficiently accurate approximation of λ is already available, one may simply keep μ fixed and apply Algorithm 3. If only a crude approximation is known, a common strategy is to update the shift at each iteration by setting

$$\mu_m = \ell_m = x_m^H A x_m,$$

where ℓ_m is the Rayleigh quotient associated with the current iterate. This often leads to a remarkable acceleration of convergence, Although its convergence analysis is more involved and global convergence is not guaranteed in general. However, failures are rare in practice. The resulting method is known as the *Rayleigh quotient inverse iteration*. A drawback of this approach is that a different linear system,

$$(A-\mu_m I)y = x_m,$$

must be solved at every iteration. For large sparse matrices, these systems can be solved iteratively. In such cases, a fixed preconditioner for $A-\ell_0 I$ can be used as a preconditioner for all inner linear systems.

To understand why the Rayleigh quotient inverse iteration converges very fast if it converges, we first prove the following lemma and then give an insight to the convergence.

Lemma 2.2. Assume that v is and eigenvector of $A \in \mathbb{C}^{n\times n}$ associated to eigenvalue λ. Assume further that $\|v\|_2 = 1$. Let $x \in \mathbb{C}^n$ with $\|x\|_2 = 1$, and let $\mu = x^H Ax$ be the Rayleigh quotient of x. Then we have

$$|\lambda - \mu| \leqslant 2\|A\|_2\|v - x\|_2. \tag{2.2}$$

If in addition, A is Hermitian, that is $A = A^H$, then

$$|\lambda - \mu| \leqslant 2\|A\|_2\|v - x\|_2^2. \tag{2.3}$$

Proof. In the general case, since $Av = \lambda v$ and $\|v\|_2 = 1$ we have $\lambda = v^H Av$, and we can write $\lambda - \mu = v^H Av - x^H Ax = v^H A(v-x) + (v-x)^* Ax$. This gives $|\lambda - \mu| \leqslant |v^H A(v-x)| + |(v-x)^H Ax|$. By Cauchy-Schwarz inequality we have $|v^H A(v-x)| \leqslant \|v^H\|_2\|A(v-x)\|_2 \leqslant 1 \cdot \|A\|_2\|v-x\|_2$. Similarly, we have $|(v-x)^H Ax| \leqslant \|A\|_2\|v-x\|_2$ which together give the upper bound (2.2).

If $A = A^H$, the Rayleigh quotient is stationary at an eigenvector. First note that both eigenvalue λ of A and Rayleigh quotient $\mu = x^H Ax$ are real. Let $e = x - v$. Then $x = v + e$ and

$$\mu = x^H Ax = (v+e)^H A(v+e).$$

Using $Av = \lambda v$ and $v^H A = \lambda v^H$, we obtain

$$\mu = v^H Av + v^H Ae + e^H Av + e^H Ae = \lambda + \lambda v^H e + \lambda e^H v + e^H Ae.$$

Therefore,

$$\mu - \lambda = \lambda(v^H e + e^H v) + e^H Ae.$$

Since $\|x\|_2 = \|v\|_2 = 1$, we have

$$0 = \|x\|_2^2 - \|v\|_2^2 = (v+e)^H(v+e) - v^H v = v^H e + e^H v + e^H e.$$

Hence, $v^H e + e^H v = -\|e\|_2^2$. Substituting this into the expression for $\mu - \lambda$ gives

$$\mu - \lambda = -\lambda\|e\|_2^2 + e^H Ae.$$

Thus,

$$|\mu - \lambda| \leq |\lambda|\|e\|_2^2 + \|A\|_2\|e\|_2^2.$$

Since A is Hermitian, $|\lambda| \leq \|A\|_2$. Therefore,

$$|\mu - \lambda| \leq 2\|A\|_2\|e\|_2^2,$$

which completes the proof. □

This lemma shows that if $\|v - x\|_2 = \mathcal{O}(\epsilon)$ then $|\lambda - \mu| = \mathcal{O}(\epsilon)$. In a special case, if A is a Hermitian matrix and $\|v - x\|_2 = \mathcal{O}(\epsilon)$ then $|\lambda - \mu| = \mathcal{O}(\epsilon^2)$.

Now let us take a closer look at the convergence rate of the Rayleigh quotient inverse iteration. Let x_m, with $\|x_m\|_2 = 1$, be the sequence generated by the method, and suppose that it converges to an eigenvector v of A, with $\|v\|_2 = 1$, corresponding to a simple (not repeated) eigenvalue λ. Since the m-th step of the Rayleigh quotient inverse iteration is simply the power method applied to the matrix $(A - \mu_m I)^{-1}$, the convergence theory of the power method implies that

$$\|v - x_{m+1}\|_2 \approx \rho_m \|v - x_m\|, \qquad \rho_m = \frac{|\lambda - \mu_m|}{|\lambda_* - \mu_m|}$$

where $\lambda - \mu_m$ and $\lambda_* - \mu_m$ are the last two eigenvalues of $A - \mu_m I$ with smallest magnitudes. If the method converges, then $\mu_m = \ell_m \to \lambda$, and therefore $\rho_m \to 0$. Thus, the convergence becomes faster as the iteration proceeds. From the bound (2.2), we have $|\lambda - \mu_m| \leqslant 2\|A\|_2 \|v - x_m\|_2$. Moreover, since $\mu_m \approx \lambda$, we have $|\lambda_* - \mu_m| \approx |\lambda_* - \lambda|$. Hence,

$$\rho_m \approx \frac{2\|A\|_2}{|\lambda_* - \lambda|} \|v - x_m\|_2 = C\|v - x_m\|_2,$$

where $C = 2\|A\|_2/|\lambda_* - \lambda|$. Substituting this estimate into the error relation gives

$$\|v - x_{m+1}\|_2 \approx \rho_m \|v - x_m\| \approx C \, \|v - x_m\|_2^2$$

which means that the error at iteration $m + 1$ is approximately proportional to the square of the error at iteration m. If A is Hermitian, the situation is even better. In this case, we may use the sharper bound (2.3) instead of (2.2), yielding

$$\|v - x_{m+1}\|_2 \approx C \, \|v - x_m\|_2^3.$$

Therefore, the error at iteration $m+1$ is approximately proportional to the cube of the error at iteration m.

Definition 2.2. A sequence $\{x_m\}$, $m = 0, 1, \ldots$, converging to x is said to converge with order p if there exists a constant $C > 0$ such that

$$\lim_{m \to \infty} \frac{\|x_{m+1} - x\|}{\|x_m - x\|^p} = C.$$

For the case $p = 1$ (linear convergence), the constant C must satisfy $0 < C < 1$ (see Definition 2.1). The case $p = 2$ is called *quadratic convergence*, while the case $p = 3$ is called *cubic convergence*.

The Rayleigh quotient inverse iteration *if converges*, the convergence is quadratic in general case, and it is cubic for Hermitian matrices. A quadratic convergence means that each step doubles the correct decimal digits (if $C \approx 1$) because if $\|x_m - v\| \approx 10^{-t}$ then $\|x_{m+1} - v\|_2 \approx C \cdot 10^{-2t}$ for enough large values of m. If $C > 1$ the accuracy is a bit less. Imagine how much faster a cubic convergence is.

2.4 A numerical example

Consider the tridiagonal matrix

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{20\times 20}.$$

This matrix arises from the finite difference discretization of the one-dimensional boundary value problems and is often referred to as the discrete Laplacian. It is symmetric, positive definite, and tridiagonal. Its eigenvalues are known explicitly:

$$\lambda_k = 2 - 2\cos\big(1 - \frac{k}{21}\big)\pi, \qquad k = 1, \ldots, 20,$$

while the corresponding eigenvectors are

$$v_k^{(j)} = \sin\big(1 - \frac{k}{21}\big)j\pi, \quad j = 1, \ldots, 20, \quad k = 1, \ldots, 20.$$

Numerically,

$$\lambda_1 \doteq 3.9777, \quad \lambda_2 \doteq 3.9111, \qquad \cdots \qquad \lambda_{19} \doteq 0.0889, \quad \lambda_{20} \doteq 0.0223.$$

We apply the power method to A using the initial vector

$$x_0 = [1, 0, \ldots, 0]^T.$$

Since the largest eigenvalue is $\lambda_1 \neq \lambda_2$, the iterates should converge to eigenpair with (λ_1, v_1). On the other hand the inverse power method should converge to the eigenpair (λ_{20}, v_{20}).

Figure 1 illustrates the convergence histories of the power method and the inverse power method. For each method, we monitor the errors for both eigenvalue and eigenvector. In practical computations, the exact eigenpair is unknown and the errors cannot be evaluated directly. Instead, we estimate the accuracy of the computed eigenvector by the relative residual

$$\frac{\|Ax_m - \ell_m x_m\|_2}{\|Ax_m\|_2 + 1},$$

and the accuracy of the computed eigenvalue by the relative change

$$\frac{|\ell_m - \ell_{m-1}|}{|\ell_{m-1}| + 1}.$$

Adding by 1 in the denominators avoids division by zero and automatically turns the above quantities into absolute error measures whenever the denominator is close to zero.

Since the exact eigenpairs are available for the present test problem, we can compare these practical stopping criteria with the actual errors,

$$\frac{\|v - x_m\|_2}{\|v\|_2},$$

and

$$\frac{|\lambda - \ell_m|}{|\lambda| + 1},$$

where (λ, v) denotes the exact eigenpair. This comparison allows us to assess how accurately the residual-based criteria reflect the true errors.

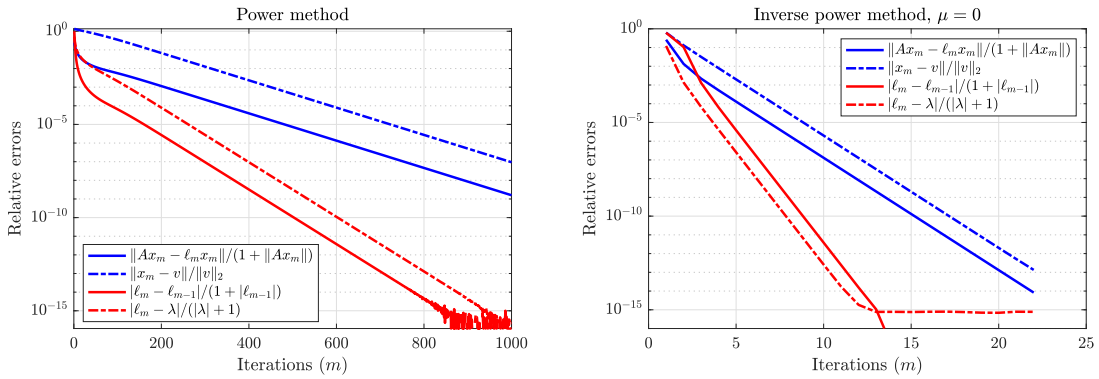

Figure 1: Convergence of the power method (left) and the inverse power method with shift $\mu = 0$ (right). Dash-dot lines denote the true errors, while solid lines correspond to the practical error indicators based on the residual and successive eigenvalue approximations.

The figures show that the stopping criteria provide reliable estimates of the actual errors. In all experiments, the practical error indicators and the true errors are nearly parallel on the logarithmic scale and remain almost close throughout the iteration.

Another observation is that the eigenvalue converges significantly faster than the eigenvector. This is a consequence of the symmetry of the matrix. For Hermitian matrices, once the eigenvector approximation is sufficiently accurate, the Rayleigh quotient satisfies

$$|\lambda - \ell_m| = \mathcal{O}\left(\|x_m - v\|_2^2\right),$$

that is, the eigenvalue error is proportional to the square of the eigenvector error. This quadratic relationship generally does not hold for non-Hermitian matrices.

Comparing the left and right panels of Figure 1, we observe that the inverse power method converges much faster than the power method. The power method requires approximately 900 iterations to achieve the prescribed tolerance, whereas the inverse power method converges in only about 13 iterations. The difference is explained by the respective convergence factors,

$$\rho_{\text{power}} = \frac{|\lambda_{19}|}{|\lambda_{20}|} \approx 0.98, \qquad \rho_{\text{inv}} = \frac{|\lambda_1|}{|\lambda_2|} \approx 0.25.$$

Since ρ_{power} is very close to 1, the error decreases very slowly. In contrast, the inverse power method has a convergence factor of approximately 0.25, so its error behaves roughly like $\left(\frac{1}{4}\right)^m$, which explains its rapid convergence.

Next, we apply the inverse power method with different choices of the shift to compute the dominant eigenvalue λ_1. To illustrate the influence of the shift, we perform the computations with the fixed values $\mu = 5$ and $\mu = 4.5$, and finally with the adaptive choice $\mu = \ell_m$, which yields the Rayleigh quotient iteration.

The convergence histories are shown in Figure 2. For the fixed shifts, the method converges linearly with asymptotic convergence ratios

$$\frac{|\lambda_1 - 5|}{|\lambda_2 - 5|} \approx 0.94, \qquad \frac{|\lambda_1 - 4.5|}{|\lambda_2 - 4.5|} \approx 0.89.$$

As predicted by the convergence theory, the smaller convergence factor leads to a noticeably faster convergence. This is visible in Figure 2 where approximately 250 iterations are required when $\mu = 5$, whereas only about 130 iterations are needed for $\mu = 4.5$. Finally, we consider the adaptive shift strategy $\mu = \ell_m$, that is, the Rayleigh quotient iteration. In this case, the shift is updated at every iteration using the current eigenvalue approximation. As soon as the iterates enter a sufficiently small neighborhood of the exact eigenpair, the convergence becomes cubic. Consequently, the dominant eigenvalue is computed to machine precision in only about 10 iterations, a dramatic improvement over the inverse iteration with a fixed shift.

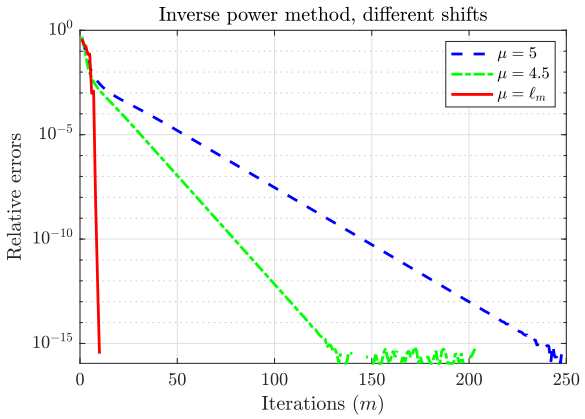

Figure 2: Convergence of inverse iteration with the fixed shifts $\mu = 5$ and $\mu = 4.5$, and of Rayleigh quotient iteration with the adaptive shift $\mu = \ell_m$. The fixed-shift methods exhibit linear convergence, while the Rayleigh quotient iteration converges cubically once sufficiently close to the desired eigenpair.

3 Similarity transformations

Similarity transformations play a central role in eigenvalue computations. The basic idea is to replace a given matrix by another matrix that has the same eigenvalues but a simpler structure. Many eigenvalue algorithms, including the QR algorithm that we will discuss soon, are based on sequences of similarity transformations.

Definition 3.1. Two matrices $A, B \in \mathbb{C}^{n\times n}$ are said to be *similar* if there exists a nonsingular matrix $S \in \mathbb{C}^{n\times n}$ such that

$$B = S^{-1}AS.$$

The matrix B is called a *similarity transformation* of A.

Similarity transformations preserve the eigenvalue problem.

Theorem 3.1. Assume that $B = S^{-1}AS$, where S is nonsingular. Then A and B have the same eigenvalues. Moreover, if w is an eigenvector of B corresponding to an eigenvalue λ, then

$$v = Sw$$

is an eigenvector of A corresponding to the same eigenvalue.

Proof. Suppose that $Bw = \lambda w$. Substituting $B = S^{-1}AS$ gives $S^{-1}ASw = \lambda w$. Multiplying by S yields $A(Sw) = \lambda(Sw)$. Since S is nonsingular and $w \neq 0$, we have $Sw \neq 0$. Therefore, Sw is an eigenvector of A corresponding to λ. This shows that every eigenvalue of B is also an eigenvalue of A. Reversing the roles of A and B proves the converse inclusion. □

An alternative proof follows from the characteristic polynomial:

$$\det(B - \lambda I) = \det(S^{-1}(A - \lambda I)S)\det(S^{-1})\det(A - \lambda I)\det(S)\det(A - \lambda I).$$

Hence A and B have the same characteristic polynomial and therefore the same eigenvalues.

The importance of similarity transformations comes from the fact that many matrices can be transformed into simpler forms while preserving their eigenvalues. For example, if A is *diagonalizable*, then there exists a nonsingular matrix S such that

$$S^{-1}AS = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}.$$

More generally, every matrix can be transformed into its Jordan canonical form. In practical computations, however, diagonal and Jordan forms are rarely computed directly because they are numerically sensitive.

A particularly important class of similarity transformations is obtained when the transformation matrix is unitary.

Definition 3.2. A matrix $Q \in \mathbb{C}^{n\times n}$ is called *unitary* if

$$Q^H Q = QQ^H = I.$$

A similarity transformation of the form

$$B = Q^H AQ$$

is called a *unitary similarity transformation.*

Unitary similarity transformations have several desirable properties. Since

$$\|Qx\|_2 = \|x\|_2$$

for every vector $x \in \mathbb{C}^n$, they preserve lengths, angles, and orthogonality. Consequently, they are numerically stable and do not magnify rounding errors. The following result is fundamental.

Theorem 3.2 (Schur decomposition)**.** Every matrix $A \in \mathbb{C}^{n\times n}$ is unitarily similar to an upper triangular matrix. That is, there exists a unitary matrix Q and an upper triangular matrix $T \in \mathbb{C}^{n\times n}$ such that

$$Q^H A Q = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ & t_{22} & \cdots & t_{2n} \\ & & \ddots & \vdots \\ & & & t_{nn} \end{bmatrix} =: T,$$

where the diagonals $t_{kk} = \lambda_k$, $k = 1, \ldots, n$ are the eigenvalues of A.

Proof. We prove the result by induction on n. For $n = 1$, the statement is trivial. Assume that the result holds for matrices of size $(n-1)\times(n-1)$, and let $A \in \mathbb{C}^{n\times n}$. Let λ_1 be an eigenvalue of A and let q_1 be a corresponding eigenvector normalized so that $\|q_1\|_2 = 1$. Extend q_1 to an orthonormal basis of $\mathbb{C}^n$ and form the unitary matrix

$$Q_1 = [q_1, q_2, \ldots, q_n].$$

Then

$$Q_1^H A Q_1 = \begin{bmatrix} q_1^H A q_1 & q_1^H A \tilde{Q}_1 \\ \tilde{Q}_1^H A q_1 & \tilde{Q}_1^H A \tilde{Q}_1 \end{bmatrix},$$

where $\tilde{Q}_1 = [q_2, \ldots, q_n]$. Since $Aq_1 = \lambda_1 q_1$, we have $q_1^H A q_1 = \lambda_1$ and $\tilde{Q}_1^H A q_1 = \lambda_1 \tilde{Q}_1^H q_1 = 0$. Hence

$$Q_1^H A Q_1 = \begin{bmatrix} \lambda_1 & w^H \\ 0 & A_1 \end{bmatrix},$$

where $A_1 \in \mathbb{C}^{(n-1)\times(n-1)}$. By the induction hypothesis, there exists a unitary matrix $Q_2' \in \mathbb{C}^{(n-1)\times(n-1)}$ such that $(Q_2')^H A_1 Q_2' = T_1$, where T_1 is upper triangular. Define

$$Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & Q_2' \end{bmatrix}.$$

Then Q_2 is unitary, and

$$Q_2^H \begin{bmatrix} \lambda_1 & w^H \\ 0 & A_1 \end{bmatrix} Q_2 = \begin{bmatrix} \lambda_1 & w^H Q_2' \\ 0 & T_1 \end{bmatrix}.$$

The matrix on the right is upper triangular. Finally, set $Q = Q_1 Q_2$. Since Q_1 and Q_2 are unitary, Q is unitary. Moreover,

$$Q^H A Q = Q_2^H Q_1^H A Q_1 Q_2$$

is upper triangular. Thus A is unitarily similar to an upper triangular matrix. □

The matrix T is called the (complex) *Schur form* of A. Even when A is real, the Schur form T and unitary matrix Q might be complex. It is predictable because real matrices may have complex eigenvalues and eigenvectors.

Exercise 3.1. Show that in the Schur decomposition $Q^H AQ = T$, the first column of Q is an eigenvector of A associated to eigenvalue $\lambda_1 = t_{11}$. But other columns of Q are not eigenvectors of A. Given the eigenvectors of T obtain the eigenvectors of A.

Exercise 3.2. Show that if A is Hermitian, then the Schur form T is diagonal and we obtain the spectral decomposition

$$A = Q\Lambda Q^H,$$

where Q is unitary and Λ is a real diagonal matrix containing the eigenvalues of A. In this case, columns of Q are eigenvalues of A.

The above Exercise again proves that eigenvalues of a Hermitian matrix are real and its eigenvectors form an orthonormal basis for $\mathbb{C}^n$.

Hermitian matrices are not the only class of matrices which are unitarily similar to a diagonal matrix. In Exercises 8.6, 8.7, and 8.8 we prove that:

Theorem 3.3. A matrix $A \in \mathbb{C}^{n\times n}$ is unitarily similar to a diagonal matrix if and only if A is a *normal* matrix.

Note that, a matrix $A \in \mathbb{C}^{n\times n}$ is called normal if $A^H A = AA^H$. Examples of normal matrices are Hermitian, skew-Hermitian, and unitary matrices.

We are not happy with the fact that the Schur decomposition of a real matrix falls into the complex field. From a computational point of view, complex arithmetic is considerably more expensive than real arithmetic. This raises the following question: can we relax some conditions and instead have a similar real decompotion for real matrices? The answer is yes.

Theorem 3.4 (Real Schur Decomposition)**.** Let $A \in \mathbb{R}^{n\times n}$. Then there exists an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$, such that

$$Q^T AQ = \begin{pmatrix} T_{11} & * & \cdots & * \\ & T_{22} & \ddots & \vdots \\ & & \ddots & * \\ & & & T_{pp} \end{pmatrix} =: T,$$

where each diagonal block T_{kk} is either a 1×1 block containing a real eigenvalue of A, or a 2×2 block whose eigenvalues form a complex conjugate pair, i.e. two eigenvalues of A.

Proof. The proof proceeds by induction on n. If A has a real eigenvalue λ, let v be a corresponding real eigenvector. Extend the normalized vector $q_1 = v/\|v\|_2$ to an orthonormal basis of $\mathbb{R}^n$, and let $Q_1 = [q_1, \ldots, q_n]$. Then

$$Q_1^T AQ_1 = \begin{pmatrix} \lambda & * \\ 0 & A_1 \end{pmatrix},$$

where $A_1 \in \mathbb{R}^{(n-1)\times(n-1)}$. Applying the induction hypothesis to A_1 yields the desired decom-

position.

If A has no real eigenvalues, then it possesses a pair of complex conjugate eigenvalues $\lambda, \overline{\lambda}$. Let $v = u + iw$ be an eigenvector corresponding to λ, where $u, w \in \mathbb{R}^n$. Since λ is nonreal, the vectors u and w are linearly independent and span a two-dimensional A-invariant subspace. Choosing an orthonormal basis of this subspace and extending it to an orthonormal basis of $\mathbb{R}^n$, we obtain an orthogonal matrix Q_1 such that

$$Q_1^T A Q_1 = \begin{pmatrix} B & * \\ 0 & A_1 \end{pmatrix},$$

where B is a 2×2 real matrix with eigenvalues λ and $\overline{\lambda}$. Applying the induction hypothesis to A_1 completes the proof. □

4 The QR algorithm

We now discuss one of the most important algorithms for computing eigenvalues of a matrix $A \in \mathbb{C}^{n \times n}$: the *QR algorithm* introduced in [2, 1961] and [3, 1962] by John Francis[1]. In this section we present the so-called *explicit* form of the algorithm, which is conceptually simpler but not the most efficient implementation in practice. In Section 5, we introduce the *implicit* or *Francis* algorithm, which is the practical implementation used in all eigensolvers.

The main idea of QR algorithm is to construct a sequence of matrices unitarily similar to A that converges, under suitable assumptions, to an upper triangular matrix. Since similar matrices have the same eigenvalues, and the eigenvalues of an upper triangular matrix are its diagonal entries, this gives a practical method for computing the eigenvalues of A.

Let $A_0 = A$. At the m-th step of the algorithm, we compute the QR factorization

$$A_m = Q_m R_m,$$

where Q_m is unitary and R_m is upper triangular. We then flip the order and define

$$A_{m+1} = R_m Q_m.$$

Substituting $R_m = Q_m^H A_m$ into the second equation yields

$$A_{m+1} = Q_m^H A_m Q_m, \tag{4.1}$$

which shows that matrices A_m and A_{m+1} are unitarily similar. Hence all matrices in the sequence have the same eigenvalues as A. One step the the algorithm which moves us from

[1]Like many ideas in mathematics, the QR algorithm was discovered by more than one researcher. At about the same time, Vera Kublanovskaya independently introduced the method in [4, 1961]. Her work is more theoretical, whereas the papers of Francis, especially the second one, is more implementation focused. Both inventors were influenced by Heinz Rutishauser, who introduced the LR algorithm for the eigenvalue problem a few years earlier [6, 1958]. The LR algorithm is based on the LU decomposition instead of the QR decomposition. Although effective in some cases, it is difficult to extend into a robust algorithm for general matrices.

A_m to A_{m+1} is called an *explicit QR step*. Each step contains one matrix factorization and one matrix-matrix multiplication.

We will see in Subsection **??** that QR algorithm can be interpreted as a clever implementation of an algorithm known as *simultaneous iteration*, which itself is a natural extension of the power method. As a consequence, under suitable assumptions, the sequence of matrices $\{A_m\}$ converges to an upper triangular matrix $T \in \mathbb{C}^{n\times n}$ of the form

$$T = \begin{bmatrix} \lambda_1 & * & \cdots & * \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda_n \end{bmatrix}$$

where eigenvalues of A are sorted on its diagonal in order of decreasing magnitude. A natural convergence criterion is based on the relative norm of the strictly lower triangular part of A_{m+1}. The algorithm needs to be improved in many directions but a naive version is given below.

Algorithm 4: QR algorithm (a naive version)

Input: Matrix $A \in \mathbb{C}^{n\times n}$; maximum iteration $m_{\max}$.
Output: Eigenvalues $\lambda_1, \ldots, \lambda_n$ of A.

Set $A_0 = A$ and $m = 0$
while $m < m_{\max}$ **and** *convergence criterion does not hold* **do**
    Compute the QR factorization $[Q_m, R_m] = \mathrm{qr}(A_m)$
    Compute $A_{m+1} = R_m Q_m$
    Set $m \leftarrow m + 1$
end while
Set $\lambda_j = a_{jj}^{(m)}$ for $j = 1, \ldots, n$.

We test the above algorithm on some simple and real matrices with real eigenvalues.

Example 4.1. Consider

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

This matrix has eigenvalues 3 and 1. We set $A_0 = A$ and compute the QR factorization $A_0 = Q_0 R_0$:

$$Q_0 = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}, \quad R_0 = \frac{1}{\sqrt{5}} \begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}.$$

Therefore,

$$A_1 = R_0 Q_0 = \begin{bmatrix} 2.8 & 0.6 \\ 0.6 & 1.2 \end{bmatrix}.$$

The off-diagonal entries have already decreased from 1 to 0.6 and the diagonal entries have approached the eigenvalues. Repeating the process gives matrices that converge to a diagonal matrix with diagonal entries 3 and 1:

$$A_m \to \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}.$$

Thus, the eigenvalues appear on the diagonal. Since A is symmetric, the algorithm converges to a diagonal matrix (a symmetric triangular matrix!).

Example 4.2. Consider the matrix

$$A = \begin{bmatrix} 2 & \sqrt{2.5} & 0 \\ \sqrt{2.5} & 4 & \sqrt{2.5} \\ 0 & \sqrt{2.5} & 6 \end{bmatrix}.$$

This matrix has eigenvalues 7, 4, and 1. We apply the QR algorithm and we wire two iterations A_3 and A_4 below (numbers are rounded to four decimal digits):

$$A_3 \doteq \begin{bmatrix} 6.2244 & 1.3145 & 0 \\ 1.3145 & 4.7670 & -0.1717 \\ 0 & -0.1717 & 1.0086 \end{bmatrix}, \quad A_4 \doteq \begin{bmatrix} 6.6937 & 0.9083 & 0 \\ 0.9083 & 4.3058 & 0.0391 \\ 0 & 0.0391 & 1.0005 \end{bmatrix}$$

We observe that the diagonal entries are converging to the eigenvalues of A. The convergence is faster for the smaller eigenvalue so that after only five iterations, the approximation of the eigenvalue 1 is already accurate to about four decimal places. In contrast, the approximations of the eigenvalues 4 and 7 are improving at a slower rate. At the same time, the subdiagonal entries are converging to zero and the matrix is approaching an upper triangular form (here a diagonal form because the original matrix is symmetric). Again, the entry in the last subdiagonal position converges more rapidly than the others.

The fact that the QR algorithm tends to converge first to the eigenvalues of smallest magnitude is reminiscent of the *inverse* power method. This connection is not accidental and will become clear when we analyze the mechanism behind the QR algorithm. But let us first discuss two important drawbacks of its basic form.

1. Each QR step requires a QR factorization of a dense $n \times n$ matrix and a matrix-matrix product of two matrices (one triangular and the other dense) which cost about $\mathcal{O}(n^3)$ flops. If many iterations are required, this becomes expensive.

2. The convergence may be slow. The rate depends on the separation of the eigenvalues. If two eigenvalues are close in magnitude, then many QR steps may be needed before the subdiagonal entries become small. In some cases, the method does not converge.

The first drawback can be addressed by reducing A to an upper *Hessenberg* form before applying the QR algorithm. The second drawback, is addressed by introducing *shifts*, as we did also for the inverse power method.

4.1 Reducing to Hessenberg form

Definition 4.1. A matrix $H \in \mathbb{C}^{n\times n}$ is called an *upper Hessenberg matrix* if

$$h_{ij} = 0, \qquad i > j + 1.$$

Thus, H has zeros below the first subdiagonal.

Compared to an upper triangular matrix, an upper Hessenberg matrix contains an additional nonzero subdiagonal right below its main diagonal. A schematic is given below:

$$H = \begin{bmatrix} * & * & \cdots & * & * \\ * & * & \cdots & * & * \\ & * & \ddots & \vdots & \vdots \\ & & \ddots & * & * \\ & & & * & * \end{bmatrix}.$$

Every matrix $A \in \mathbb{C}^{n\times n}$ can be reduced by unitary similarity transformations to upper Hessenberg form

$$H = U^H A U. \tag{4.2}$$

The eigenvalues of A and H are then the same. The algorithm for reducing A to H is similar but not identical to the algorithm used for computing the QR factorization via Householder reflectors or Givens rotations. The first difference is that, at step k, the orthogonal transformation is chosen to eliminate the entries below the first subdiagonal rather than the entries below the diagonal. The second difference is that the transformations are applied from both the left and the right, so each step preserves similarity and finally we obtain the similarity transformation (4.2). Let us explain the algorithm in details.

Starting with a dense matrix $A =: A^{(0)} \in \mathbb{C}^{n\times n}$, our goal is to annihilate the entries below the first subdiagonal, one column at a time, using Householder similarity transformations. At the first step, we eliminate the entries below the subdiagonal in the first column. We consider the vector

$$x = A_{2:n,1} \in \mathbb{C}^{n-1},$$

and construct the Householder vector

$$u_1 = x + \alpha \widehat{e}_1, \qquad \alpha = e^{i\theta}\|x\|_2, \qquad \theta = \operatorname{angle}(x_1),$$

where $\widehat{e}_1$ denotes the first coordinate vector in $\mathbb{C}^{n-1}$. The corresponding Householder reflector is

$$\tilde{P}_1 = I - \frac{2}{u_1^H u_1} u_1 u_1^H \in \mathbb{C}^{(n-1)\times(n-1)}.$$

By construction, $\tilde{P}_1 x = -\alpha \widehat{e}_1$, so that all entries of x except the first are annihilated. To apply this reflector to the whole matrix, we embed it into $\mathbb{C}^{n\times n}$ by defining

$$P_1 = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1 \end{bmatrix}.$$

Premultiplication by P_1 transforms $A^{(0)}$ to

$$A^{(\frac{1}{2})} = P_1 A^{(0)} = \begin{bmatrix} * & * & * & \cdots & * \\ \times & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \times & \times & \cdots & \times \end{bmatrix}.$$

Notice that the first row remains unchanged because of the block structure of P_1, while the entries below the first subdiagonal in the first column have been eliminated. Since we seek a similarity transformation, we must also postmultiply by P_1. This affects only the last $n-1$ columns and therefore does not destroy the zeros introduced in the first column. Consequently,

$$A^{(1)} = A^{(\frac{1}{2})}P_1 = \tilde{P}_1 A^{(0)} P_1 = \begin{bmatrix} * & * & * & \cdots & * \\ \times & + & + & \cdots & + \\ 0 & + & + & \cdots & + \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & + & + & \cdots & + \end{bmatrix},$$

Thus, after the first step, the first column already has the desired Hessenberg structure and the remaining work is confined to the $(n-1) \times (n-1)$ submatrix marked by $+$. The subsequent steps proceed in exactly the same manner. At step k, where $1 \leqslant k \leqslant n-2$, we consider the vector

$$x = A^{(k-1)}_{k+1:n,\,k},$$

construct the corresponding Householder vector u_k, and define the reflector

$$\tilde{P}_k = I - \frac{2}{u_k^H u_k} u_k u_k^H, \qquad P_k = \begin{bmatrix} I_k & 0 \\ 0 & \tilde{P}_k \end{bmatrix},$$

where I_k denotes the $k \times k$ identity matrix. The matrix is then updated by the similarity transformation

$$A^{(k)} = P_k A^{(k-1)} P_k.$$

Each transformation annihilates the entries below the first subdiagonal in column k while leaving the previously constructed Hessenberg structure unchanged. After completing the $n-2$ steps, we obtain

$$A^{(n-2)} = (P_{n-2} \cdots P_2 P_1) A (P_1 P_2 \cdots P_{n-2}).$$

Since every Householder reflector is Hermitian and unitary, $P_k^H = P_k = P_k^{-1}$, we may define $U = P_1 P_2 \cdots P_{n-2}$, so that

$$H := A^{(n-2)} = U^H A U.$$

For practical implementation, since P_k acts only on rows and columns $k+1, \ldots, n$, we do not actually need to form it and we can work only with $\tilde{P}_k$, and the update can be performed as

$$A_{k+1:n,k:n} \leftarrow \tilde{P}_k A_{k+1:n,k:n},$$

followed by

$$A_{1:n,k+1:n} \leftarrow A_{1:n,k+1:n} \tilde{P}_k.$$

After this step, the entries $A_{k+2:n,k}$ are zero. The algorithm is given below.

Algorithm 5: Transforming a matrix to an upper Hessenberg form

Input: Matrix $A \in \mathbb{C}^{n\times n}$
Output: Matrix $A \leftarrow U^H AU$, Householder vectors $u_1, \ldots, u_{n-2}$.

for $k = 1$ **to** $n-2$ **do**
  Set $x = A_{k+1:n,\,k}$
  Construct Householder vector u_k and reflector $\tilde{P}_k$ such that $\tilde{P}_k x = \alpha \widehat{e}_1$
  Update $A_{k+1:n,\,k:n} \leftarrow \tilde{P}_k\, A_{k+1:n,\,k:n}$
  Update $A_{1:n,\,k+1:n} \leftarrow A_{1:n,\,k+1:n}\tilde{P}_k$
end for

An additional optimization is that the Householder matrices $\tilde{P}_k$ never need to be formed explicitly. Instead, we work directly with the Householder vectors u_k, which allows the similarity transformations to be carried out at a much lower computational cost. To see this, let $B \in \mathbb{C}^{n\times m}$ and let $P = I - \beta uu^H$, where $\beta = 2/u^H u$. Then

$$PB = (I - \beta uu^H)B = B - \beta u(u^H B).$$

If we first compute $w = B^H u$, then

$$PB = B - \beta uw^H.$$

Consequently, the reflector is applied without ever forming the matrix P. The computation consists of one matrix-vector product $w = B^H u$, one inner product $u^H u$ to determine β, one outer product $(\beta u)w^H$, and one matrix subtraction. The total cost is approximately $2mn + 2n + mn + mn \approx 4mn$ flops. Compare it with explicit product PB which requires approximately $2n^2 m$ flops. The right multiplication is treated in exactly the same way. If $B \in \mathbb{C}^{m\times n}$, then

$$BP = B(I - \beta uu^H) = B - \beta wu^H, \quad w = Bu.$$

We also note that, during the Hessenberg reduction, only the Householder vectors $u_1, u_2, \ldots, u_{n-2}$ are stored. The unitary matrix

$$U = P_1 P_2 \cdots P_{n-2}$$

is never formed explicitly and whenever a product of a matrix $B \in \mathbb{C}^{n\times n}$ with U or U^H is needed, it is computed by successively applying the stored Householder reflectors at cost $4n^2$ instead of $2n^3$.

An additional memory optimization is commonly employed in practical implementations. At the kth step, the entries below the first subdiagonal in column k become zero after the transformation. These locations are therefore no longer needed by the Hessenberg matrix and can be reused to store the Householder vector u_k (except for its leading entry, which is implicitly equal to 1 or stored separately together with the scalar β). Consequently, the Hessenberg matrix and the Householder vectors occupy the same storage, requiring only a single matrix together with a vector of scalar coefficients.

Exercise 4.1. Show that for a matrix A of size $n \times n$, the computational cost for transforming it to upper Hessenberg matrix H is $O(n^3)$. Derive the leading constant as well.

Exercise 4.2. Assume that A is Hermitian. Show that the Hessenberg matrix produced by Algorithm 5 is necessarily tridiagonal. Then modify the update lines of Algorithm 5 to exploit the Hermitian structure and show that the overall computational cost can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

Reducing a general matrix A to a Hessenberg form is useful for QR algorithm because the Hessenberg structure is preserved by a QR step: if H_m is upper Hessenberg and

$$H_m = Q_m R_m,$$

then

$$H_{m+1} = R_m Q_m$$

is again upper Hessenberg. Therefore, after the initial reduction, each QR factorization can be performed in $\mathcal{O}(n^2)$ flops instead of $\mathcal{O}(n^3)$, because the QR factorization of an upper Hessenberg matrix can be computed using Givens rotations to eliminate $n-1$ subdiagonal entries $a^{(m)}_{j+1,j}$, $j = 1, \ldots, n-1$. This process costs only $\mathcal{O}(n^2)$ flops per step. In addition, the unitary matrix Q_m, obtained from QR factorization of upper Hessenberg matrix H_m, is not a dense matrix but it is an upper Hessenberg matrix. However, the cost of the matrix product $R_m Q_m$ is about $n^3/6$. It remains at order $\mathcal{O}(n^3)$ although 12 times less than the product of two dense matrices. We will see in Section 5, how we can bypass these costs more efficiently.

4.2 Shifted QR algorithm

Since we can always reduce a general matrix $A \in \mathbb{C}^{n\times n}$ to an upper Hessenberg matrix in an initial step, from here on we assume that A is an upper Hessenberg matrix.

Instead of applying the QR algorithm to A_m, we choose a proper shift μ and compute the QR factorization

$$A_m - \mu I = Q_m R_m.$$

Then we define

$$A_{m+1} = R_m Q_m + \mu I.$$

Again, this is a similarity transformation because

$$A_{m+1} = Q_m^H A_m Q_m. \tag{4.3}$$

Thus, the shifted QR iteration preserves the eigenvalues of the original matrix A. This is one QR step, also called *explicit QR step*.

Algorithm 6: Explicit QR step

Input: Matrix $A \in \mathbb{C}^{n\times n}$; shift μ
Output: Matrix $A \leftarrow Q^H A Q$.

Compute the QR factorization $[Q, R] = \mathrm{qr}(A - \mu I)$
Set $A = RQ + \mu I$

The form of similarity transformation (4.3) is similar to (4.1), except that Q_m (and R_m) now are the QR factors of the shifted matrix $A_m - \mu I$. It does not matter where Q_m come from. As long as we have a similarity transformation of the form (4.3), the matrices A_{m+1} and

A_m have the same eigenvalues. However, if they come from the shifted matrix $A_m - \mu I$ with a suitable shift parameter μ, then the convergence is very fast for the closest eigenvalue to μ. This eigenvalue starts to appear in the bottom-right corner of matrix A_m as m increases.

Example 4.3. Consider again the matrix A from Example 4.2, and choose the shift $\mu = 4.5$. Then we compute the QR factorization of $A_m - 4.5I$, say $A_m - 4.5I = Q_m R_m$, and then form $A_m = R_m Q_m + 4.5I$. Iterations A_3 and A_4 are given below:

$$A_3 = \begin{bmatrix} 1.0317 & -0.4350 & 0 \\ -0.4350 & 6.9678 & 0.0380 \\ 0 & 0.0380 & 4.0005 \end{bmatrix}, \quad A_4 = \begin{bmatrix} 1.0162 & 0.3115 & 0 \\ 0.3115 & 6.9838 & 0.0076 \\ 0 & 0.0076 & 4.0000 \end{bmatrix}$$

The bottom-right entry converges rapidly to the eigenvalue 4, which is the eigenvalue closest to the chosen shift $\mu = 4.5$. An even faster convergence could be achieved with a shift that is closer to the desired eigenvalue. The other diagonal entries are also moving toward the exact eigenvalues 1 and 7, although at slower rates. Moreover, the subdiagonal entry a_{32} is already much smaller in magnitude than a_{21}. The idea of shifted QR algorithm is to exclude the eigenvalue 4 when it is accurate enough, and go after other eigenvalues through the top-left block of the latest matrix.

Once the size of the last subdiagonal entry, $|a^{(m)}_{n,n-1}|$, becomes sufficiently small, we set it to zero. The matrix then takes the block form

$$A_m \equiv \begin{bmatrix} * & \cdots & * & * & * \\ * & \cdots & * & * & * \\ & \ddots & & \vdots & \vdots \\ & & * & * & * \\ & & & 0 & a^{(m)}_{nn} \end{bmatrix} = \begin{bmatrix} \tilde{A}_m & w^H \\ 0 & a^{(m)}_{nn} \end{bmatrix}$$

where $\tilde{A}^{(m)}$ is still an upper Hessenberg matrix of size $n-1$ (typically with subdiagonal entries of smaller magnitude than those of the original matrix A). The eigenvalues of A_m are then the eigenvalues of $\tilde{A}_m$ together with $a^{(m)}_{nn}$ (up to a prescribed tolerance). At this stage, we accept $a^{(m)}_{nn}$ as an approximate eigenvalue and continue the QR algorithm on the smaller matrix $\tilde{A}_m$. In other words, we set $A \leftarrow \tilde{A}_m$ and $n \leftarrow n-1$, and then repeat the process to compute the next eigenvalue.

This procedure is called *deflation* and is repeated until all eigenvalues of A have been computed. The computation becomes cheaper after each deflation step for two reasons. First, the size of the active matrix decreases. Second, the new Hessenberg matrix typically has smaller subdiagonal entries than the previous one, so fewer QR steps are required to compute the desired eigenvalue and achieve the next deflation. In practice, the last few eigenvalues are often obtained after only a few iterations, typically one or two.

A stopping criterion to accept the bottom-right corner $a^{(m)}_{jj}$ as an approximation for eigenvalue λ_j is to use a tolerance ε and stop iteration on m when

$$|a^{(m)}_{j,j-1}| \leqslant \varepsilon\big(|a^{(m)}_{j-1,j-1}| + |a^{(m)}_{j,j}|\big).$$

Another important question is how to choose shift parameter μ so that the method does not break down and also gives a fast convergence. For computing each eigenvalue, experi-

ments and theoretical insights suggest that it is highly recommended to choose an adaptive shift μ_m similar to what we use in Rayleigh quotient iteration. Let $A_0 = A$. A natural choice is to choose

$$\mu_m = a_{nn}^{(m)}, \tag{4.4}$$

at step m. This is called the *Rayleigh quotient shift* because $a_{nn}^{(m)}$ has actually a Rayleigh quotient form. The motivation is that the bottom-right entry often becomes a good approximation to the eigenvalue we are looking for as the iteration proceeds. For instance, in Example 4.3, suppose that we accept A_4 as sufficiently converged and extract the eigenvalue 4. After deflation, we obtain the leading 2×2 block

$$A = \begin{bmatrix} 1.0162 & 0.3115 \\ 0.3115 & 6.9838 \end{bmatrix}.$$

If we now choose the shift $\mu_0 = a_{22}^{(0)} = 6.9838$, then only one QR step is required to find an approximation of the eigenvalue 7 with very high accuracy.

This choice of shift works for many cases and gives a quadratic (and for Hermitian matrices a cubic) convergence rate. However, it may fail in some situations. For example, consider the matrix

$$A_0 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

from Example 4.1. Choosing $\mu_0 = a_{22}^{(0)} = 2$ gives

$$A_0 - 2I = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \overset{\text{QR}}{=} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then

$$A_1 = R_0 Q_0 + 2I = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

which is exactly the original matrix A. Hence the iteration falls into a loop, and the algorithm stagnates.

A more robust choice is the *Wilkinson shift*. The Wilkinson shift is defined as that eigenvalue of the 2×2 bottom-right block

$$W_m = \begin{bmatrix} a_{n-1,n-1}^{(m)} & a_{n-1,n}^{(m)} \\ a_{n,n-1}^{(m)} & a_{nn}^{(m)} \end{bmatrix}$$

that is closer to $a_{nn}^{(m)}$. This shift usually gives much faster and more reliable convergence. One can compute the eigenvalues of 2×2 matrices by computing the roots of its quadratic characteristic polynomial.

Remark 4.1. The difficulty with computing the roots of quadratic equation

$$a\lambda^2 + b\lambda + c = 0, \qquad a \neq 0,$$

is *catastrophic cancellation* when $b^2 \gg 4ac$. In that case one of the roots is computed as the difference of two nearly equal numbers. To avoid this problem, we first compute the

discriminant $d = b^2 - 4ac$, and then define

$$q = -\frac{1}{2}\Big(b + \text{sign}(b)\sqrt{d}\Big),$$

where $\text{sign}(0) = 1$. The roots are then computed as $\lambda_1 = q/a$, and $\lambda_2 = c/q$. The second formula follows from Vieta's relation $\lambda_1\lambda_2 = c/a$. The advantage of this approach is that the quantity $b + \text{sign}(b)\sqrt{d}$ never involves subtracting two nearly equal numbers. Consequently, the larger root is computed accurately, while the smaller root is recovered through Vieta's formula, which avoids cancellation altogether.

Algorithm 7: Shifted QR algorithm with explicit QR steps

Input: Matrix $A \in \mathbb{C}^{n\times n}$; tolerance ε; maximum iteration $m_{\max}$.
Output: Eigenvalues $\lambda_1, \ldots, \lambda_n$ of A.

Transform A to upper Hessenberg form A_0 using Algorithm 5,
for $j = n$ **downto** 2 **do**
  Set $\varepsilon_{loc} = \varepsilon\big(|a^{(0)}_{j-1,j-1}| + |a^{(0)}_{j,j}|\big)$ and set $e = \varepsilon_{loc} + 1$, and set $m = 0$.
  while $m < m_{\max}$ **and** $e > \varepsilon_{loc}$ **do**
    Compute the Wilkinson shift μ_m
    if $m = m_{\max} - 1$ **then** change μ_m to a random shift and set $m = 0$
    Given A_m and μ_m, compute A_{m+1} from Algorithm 6 % explicit QR step
    Set $e = |a^{(m+1)}_{j,j-1}|$
    Set $m \leftarrow m + 1$
  end while
  Set $\lambda_j = a^{(m)}_{jj}$
  Set $A_0 = A_m(1 : j-1, 1 : j-1)$
end for
Set $\lambda_1 = a^{(0)}_{11}$.

This algorithm calls Algorithm 6 for performing an explicit QR step at each iteration. To achieve better computational complexity, the QR factorization at each step must exploit the upper Hessenberg structure of the shifted matrix $A_m - \mu_m I$. Likewise, the product $R_m Q_m$ should take advantage of the structures of the factors: R_m is upper triangular and Q_m is upper Hessenberg.

For real symmetric Hessenberg matrices (which are necessarily tridiagonal and all eigenvalues are real), it has been shown that the shifted QR algorithm with Wilkinson shifts always converges and the convergence is at least cubic [5, 1980]. For general matrices, however, there still remain some special and rare cases for which the Wilkinson shift fails. See the following exercise.

Exercise 4.3. Assume that A is a unitary matrix, that is, $A^{-1} = A^H$. Show that the standard QR algorithm (without shifts) makes no progress in this case; specifically, prove that

$$A_m = A_0 = A$$

for all m. How can this phenomenon be explained in terms of the eigenvalues of a unitary matrix? *Hint:* The eigenvalues of a unitary matrix lie on the unit circle in the complex

plane.
Now consider the following real proper upper Hessenberg matrix:

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ & 1 & \ddots & \vdots & \vdots \\ & & \ddots & 0 & 0 \\ & & & 1 & 0 \end{bmatrix}.$$

Confirm that A is a unitary (orthogonal in this real case) matrix, and show why both Rayleigh quotient and Wilkinson shifts fail for this matrix.

Exceptional shifts: To handle the rare situations, in which the QR iteration stagnates or converges extremely slowly, practical implementations of the QR algorithm for nonsymmetric matrices incorporate an *exceptional shift* strategy. When the algorithm detects a lack of progress, it temporarily replaces the usual shift by a specially chosen (random) shift whose magnitude is comparable to that of the matrix entries. Such a shift breaks any unfavorable symmetries or cyclic behavior that may be hindering convergence. After that the algorithm resumes its standard shifting strategy.

4.3 Eigenvalues of real nonsymmetric matrices

If A is a real symmetric matrix, then every iterate produced by the QR algorithm with Wilkinson shifts remains real and symmetric (Wilkinson shifts are all real in this case), so all computations can be carried out in real arithmetic. As discussed earlier, the algorithm is guaranteed to converge in this case [5, 1980].

If A is real but nonsymmetric, even if all eigenvalues of A are real, Wilkinson shifts computed from the trailing 2×2 block may be complex. This pushes all computations to complex arithmetic. The same issue arises when A has complex eigenvalues. In this case, since A is real, complex eigenvalues always occur in conjugate pairs: if $\lambda = \alpha + i\beta$ is an eigenvalue, then $\bar{\lambda} = \alpha - i\beta$ is also an eigenvalue. Thus, once one eigenvalue of the pair has been identified, the other is known automatically. Nevertheless, the above shifted QR algorithm does not care about this fact. Furthermore, because the original matrix is real, it is desirable to perform the entire computation using only real arithmetic. We prefer real arithmetic because it is much cheaper. For this reason, the QR algorithm for real nonsymmetric matrices is modified so that complex conjugate eigenvalue pairs are treated simultaneously and in real arithmetic.

First, let us consider the application of the QR algorithm *without shift* on a real nonsymmetric matrix A. In this case, rather than converging to an upper triangular matrix, the iteration converges to a *real Schur form*, also called a *quasi-triangular matrix*. Such a matrix is block upper triangular with diagonal blocks of size either 1×1 or 2×2. The 1×1 blocks correspond to real eigenvalues, while the 2×2 blocks correspond to complex conjugate pairs. We illustrate this in the following example.

Example 4.4. Consider the upper Hessenberg matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 15 \\ 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 4 \end{bmatrix},$$

which is the companion matrix of the polynomial

$$p(t) = t^4 - 4t^3 + 6t^2 - 4t - 15 = (t-3)(t+1)(t^2 - 2t + 5).$$

The roots of p are the eigenvalues of A, namely $\lambda_1 = 3$, $\lambda_2 = 1 + 2i$, $\lambda_3 = 1 - 2i$, and $\lambda_4 = -1$. We now apply the unshifted QR algorithm to A in order to see how the method behaves in the presence of complex conjugate eigenvalues. Many iterations are required before the limiting structure becomes visible. After 40 iterations, we obtain

$$A_{40} \doteq \begin{bmatrix} 3.0000 & -9.2810 & 10.3803 & -7.6666 \\ -0.0000 & 2.8046 & -3.7117 & 2.4721 \\ 0 & 1.9551 & -0.8046 & 1.0542 \\ 0 & 0 & -0.0000 & -1.0000 \end{bmatrix}.$$

The iteration is approaching a quasi-triangular matrix. The isolated diagonal entries a_{11} and a_{44} approximate the real eigenvalues 3 and -1, respectively. The complex conjugate pair is represented by the middle 2×2 diagonal block. Indeed, the eigenvalues of this block are approximately $1 + 2i$ and $1 - 2i$.

From the above example, we observe that for real nonsymmetric matrices all computations can be carried out in real arithmetic. Complex conjugate eigenvalue pairs are hiding in 2×2 diagonal blocks, and once the QR algorithm has converged, they can be obtained by solving the characteristic equations of these small blocks. The drawback is that the convergence may be slow, as it is the case in the above example. It is also not difficult to construct examples for which the unshifted QR algorithm fails to converge at all. This makes the use of shifts essential. One possibility is to employ real shifts, such as the Rayleigh quotient shift. In this case all computations remain in real arithmetic. However, a real shift cannot approximate a complex eigenvalue, and therefore may not accelerate the convergence toward a complex conjugate pair. To achieve rapid convergence, one would like to use complex shifts, such as Wilkinson shifts. Unfortunately, applying a complex shift directly would move the entire computation into complex arithmetic, even when the original matrix is real.

However, there exists a solution for this problem. One can show that if we start with a real matrix and perform one QR step with a complex shift μ, the resulting matrix is, in general, complex. If this step is immediately followed by another QR step with the conjugate shift $\bar{\mu}$, the resulting matrix is real again. This observation suggests the *double-shift* strategy: instead of explicitly computing the intermediate complex matrix, we combine the two QR steps into a single real step that produces the same final matrix while carrying out all computations entirely in real arithmetic.

Lemma 4.1. Let $A_0 \in \mathbb{R}^{n\times n}$ be a real matrix. Consider a pair of QR steps with shifts μ and $\bar{\mu}$:

$$\begin{aligned} A_0 - \mu I &= Q_0 R_0, \quad A_1 = R_0 Q_0 + \mu I \\ A_1 - \bar{\mu} I &= Q_1 R_1, \quad A_2 = R_1 Q_1 + \bar{\mu} I \end{aligned} \tag{4.5}$$

Then the matrix $(A_0 - \bar{\mu}I)(A_0 - \mu I)$ is real and we have

$$(A_0 - \bar{\mu}I)(A_0 - \mu I) = QR, \quad \text{where} \quad Q = Q_0 Q_1, \quad R = R_1 R_0.$$

In addition, if μ is not an eigenvalue of A_0 then matrices Q, R and A_2 are all real matrices.

Proof. First we have

$$(A_0 - \bar{\mu}I)(A_0 - \mu I) = A_0^2 - (\mu + \bar{\mu})A_0 + (\mu\bar{\mu})I = A_0^2 - 2\,\mathrm{Re}(\mu)A_0 + |\mu|^2 I,$$

which is a real matrix because A_0 is real and all coefficients are also real. Then from the first relation in (4.5), we have $A_1 = Q_0^H A_0 Q_0$. This gives $(A_0 - \bar{\mu}I)Q_0 = Q_0(A_1 - \bar{\mu}I)$. Thus we can use again (4.5) and write

$$(A_0 - \bar{\mu}I)(A_0 - \mu I) = (A_0 - \bar{\mu}I)Q_0 R_0 = Q_0(A_1 - \bar{\mu}I)R_0 = Q_0 Q_1 R_1 R_0 = QR.$$

Finally, if μ (and thus $\bar{\mu}$) are not eigenvalues of A_0 (and thus of A_1) then both $A_0 - \mu I$ and $A_1 - \bar{\mu}I$ are nonsingular and thus QR decompositions in (4.5) are unique provided that R_0 and R_1 are taken to have real positive entries on their main diagonals. This means that $Q = Q_0 Q_1$ is unitary and $R_1 R_0$ is upper triangular with real and positive entries on its main diagonal. Therefore, Q and R are unique QR factors of real matrix $(A_0 - \bar{\mu}I)(A_0 - \mu I)$. So, Q and R must be real matrices. On the other hand, from the first relation in (4.5) we have $A_1 = Q_0^H A_0 Q_0$ and from the second relation we have $A_2 = Q_1^H A_1 Q_1$ which together give $A_2 = (Q_0 Q_1)^H A_0 (Q_0 Q_1) = Q^H A_0 Q$. Since Q and A_0 are real, so is A_2. □

The assumption that μ is not an eigenvalue of A_0 is, in fact, not a necessary condition. It was made only to simplify the proof. Indeed, since the matrix $(A_0 - \bar{\mu}I)(A_0 - \mu I)$ is real, it is always possible to find a real Q factor for it. Then, $A_2 = Q^T A_0 Q$ is also a real matrix.

The *double-shift* strategy can now be described as follows. We start with the real upper Hessenberg matrix $A_0 = A$. At each iteration, if the computed shift μ_m is real, we perform the usual single-shift QR step:

$$A_m - \mu_m I = Q_m R_m, \qquad A_{m+1} = R_m Q_m + \mu_m I.$$

If μ_m is complex, we instead form the real matrix

$$B = (A_m - \bar{\mu}_m I)(A_m - \mu_m I) = A_m^2 - 2\,\mathrm{Re}(\mu_m)A_m + |\mu_m|^2 I,$$

compute its real QR factorization

$$B = Q_m R_m,$$

and then update

$$A_{m+1} = Q_m^T A_m Q_m.$$

Since all matrices involved are real, the entire computation is carried out in real arithmetic.

Algorithm 8: Explicit double-shift QR step

Input: Real matrix A; complex shift μ
Output: Matrix $A \leftarrow Q^T A Q$.

Compute $B = A^2 - 2\,\mathrm{Re}(\mu)A + |\mu|^2 I$
Compute factorization $[Q, R] = \mathrm{qr}(B)$
Update $A \leftarrow Q^T A Q$

An important difference is that the matrix B is no longer upper Hessenberg. Since B contains the product A_m^2 (product of two Hessenberg matrices), it has one additional nonzero subdiagonal. The pattern is schematically

$$B = \begin{bmatrix} * & * & \cdots & * & * & * \\ * & * & \cdots & * & * & * \\ * & * & \ddots & & \vdots & \vdots \\ & * & \ddots & \ddots & \vdots & \vdots \\ & & \ddots & * & * & * \\ & & & * & * & * \end{bmatrix}.$$

This does not pose a significant difficulty. The QR factorization of B can be computed using $2n-3$ Givens rotations. The computational cost is about 1.5 times more than that of the QR factorization of an upper Hessenberg matrix and therefore remains of order $\mathcal{O}(n^2)$. Noting that, in either case, A_{m+1} always remains upper Hessenberg.

Continuing this process, the shifts stabilize and converge either to a real eigenvalue or to a pair of complex conjugate eigenvalues of A. In the latter case, the QR iteration converges to a matrix of the form

$$A_m \equiv \left[\begin{array}{ccc|cc} & & & * & * \\ & \tilde{A}_m & & \vdots & \vdots \\ & & & * & * \\ \hline & & 0 & a & b \\ & & & c & d \end{array}\right],$$

where $\tilde{A}^{(m)}$ is an upper Hessenberg matrix of size $n-2$. The bottom-right 2×2 block has eigenvalues that approximate the complex conjugate pair λ and $\bar{\lambda}$ associated with the shifts μ_m and $\bar{\mu}_m$. We then accept this pair as converged, remove the last two rows and columns (perform a two-dimensional deflation), and continue the QR algorithm on the smaller Hessenberg matrix $\tilde{A}_m$.

The stopping criterion depends on whether the current shift is real or complex. For a real shift, convergence is monitored via the last subdiagonal entry $a_{n,n-1}^{(m)}$, whereas for a complex conjugate pair of shifts it is monitored via the entry $a_{n-1,n-2}^{(m)}$ (because $a_{n,n-1}^{(m)} \equiv c$ does not go to zero in this case). In both cases, the criterion can be written as

$$|a_{j,j-1}^{(m)}| \le \varepsilon\big(|a_{j-1,j-1}^{(m)}| + |a_{j,j}^{(m)}|\big),$$

where $j = n$ in the real shift case and $j = n-1$ in the complex shift case. Once this condition is satisfied, a one-dimensional or two-dimensional deflation is performed, respectively.

Algorithm 9: QR double-shift algorithm

Input: Matrix $A \in \mathbb{R}^{n\times n}$; tolerance ε; maximum iteration $m_{\max}$.
Output: Eigenvalues $\lambda_1, \ldots, \lambda_n$ of A.

Transform A to upper Hessenberg form A_0 using Algorithm 5,
Set $j = n$.
while $j > 2$ **do**
  Set $\varepsilon_{loc} = \varepsilon\big(|a^{(0)}_{j-1,j-1}| + |a^{(0)}_{j,j}|\big)$ and set $e = \varepsilon_{loc} + 1$, and set $m = 0$.
  while $m < m_{\max}$ **and** $e > \varepsilon_{loc}$ **do**
    Compute the Wilkinson shift μ_m
    if $m = m_{\max} - 1$ **then** change μ_m to a random shift and set $m = 0$
    if μ_m *is real* **then**
      Given A_m and μ_m, compute A_{m+1} form Algorithm 6 % Explicit single-shift QR step
      Set $deflation = 1$, and $e = |a^{(m+1)}_{j,j-1}|$
    end if
    if μ_m *is complex* **then**
      Given A_m and μ_m, compute A_{m+1} form Algorithm 8 % Explicit double-shift QR step
      Set $deflation = 2$, and $e = |a^{(m+1)}_{j-1,j-2}|$.
    end if
    Set $m \leftarrow m + 1$
  end while
  if *deflation = 1* **then**
    Set $\lambda_j = a^{(m)}_{j,j}$
  end if
  if *deflation = 2* **then**
    Find eigenvalues σ and $\bar{\sigma}$ of matrix $\begin{bmatrix} a^{(m)}_{j-1,j-1} & a^{(m)}_{j-1,j} \\ a^{(m)}_{j,j-1} & a^{(m)}_{j,j} \end{bmatrix}$
    Set $\lambda_j = \sigma$ and $\lambda_{j+1} = \bar{\sigma}$
  end if
  Set $j \leftarrow j - deflation$
  Set $A_0 = A_m(1 : j, 1 : j)$
end while
if $j = 1$ **then** Set $\lambda_1 = a^{(0)}_{11}$
if $j = 2$ **then** Compute eigenvalues of 2×2 matrix A_0 for λ_1 and $\lambda_2 = \bar{\lambda}_1$.

To make this algorithm computationally more efficient in terms of execution time and the memory usage, the QR factorizations and matrix-matrix products should exploit the structure of the matrices involved. We do not pursue these implementation details further, because in practice a more efficient *implicit* single- or double-shift QR algorithm (Francis algorithm) is used. We invite you now to the next section.

5 The Francis algorithm

The explicit shifted QR algorithm presented in the previous sections is useful for understanding the mechanism of the QR algorithm. However, it is not the algorithm used in practice. An explicit QR step computes the QR factorization of the shifted matrix $A_m - \mu_m I$ (at $\mathcal{O}(n^2)$ flops) and then forms the product $R_m Q_m$ (at $\mathcal{O}(n^3)$ flops). In addition, forming the shifted matrix $A_m - \mu_m I$ may introduce cancellation errors and lead to numerical instability. In the

double-shift case, it even requires the explicit construction of the matrix

$$B = (A_m - \bar{\mu}_m I)(A_m - \mu_m I),$$

which introduces additional work and may amplify rounding errors due to cancellation.

Francis observed that none of these intermediate matrices are actually needed. Since the QR algorithm is merely a sequence of unitary similarity transformations, it is sufficient to construct the final orthogonal (or unitary) similarity transformation directly. This leads to the *implicit QR algorithm*, also known as the *Francis algorithm*. It produces exactly the same iterates as the explicit algorithm (up to harmless diagonal sign changes), while avoids the explicit QR factorizations and matrix-matrix products altogether. The resulting algorithm is simpler and considerably faster in practice.

5.1 Francis single-shift algorithm

Assume that we are in the m-th step of the QR algorithm. Given an upper Hessenberg matrix A_m we aim to compute the next iteration A_{m+1}. For notational simplicity, we use A, A_+, and μ instead of A_m, A_{m+1}, and μ_m, respectively.

We start with upper Hessenberg matrix $A \in \mathbb{C}^{n\times n}$. The explicit single-shift QR algorithm computes the QR factorization

$$A - \mu I = QR$$

and then forms

$$A_+ = RQ + \mu I,$$

thereby we have

$$A_+ = Q^H A Q. \tag{5.1}$$

This process explicitly forms the shifted matrix, computes its QR factorization, and multiplies the factors in reverse order. Francis observed that none of these intermediate quantities are actually needed. Instead, one can construct the similarity transformation (5.1) directly while preserving the Hessenberg structure throughout the computation.

We have already assumed that A is an upper Hessenberg matrix. A matrix is called a *proper upper Hessenberg* or unreduced upper Hessenberg if all its subdiagonal entries are nonzero,

$$a_{j+1,j} \neq 0, \quad j = 1, 2, \ldots, n-1.$$

For the Francis algorithm, we assume the matrix A and all the subsequent matrices produced by the algorithm are properly upper Hessenberg. This assumption is not a real restriction, because if at some step we obtain $a^+_{j+1,j} = 0$ for some j then the matrix has the block upper Hessenberg form

$$A_+ = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where $A_{11} \in \mathbb{C}^{j\times j}$ and $A_{22} \in \mathbb{C}^{(n-j)\times(n-j)}$. The eigenvalues of A_+ are then the union of the eigenvalues of A_{11} and A_{22}. Therefore, the eigenvalue problem splits into two smaller independent eigenvalue problems and the algorithm continues separately on the diagonal blocks A_{11} and A_{22}. The process becomes even faster.

Now we present the Francis algorithm. The starting point is the observation that the first Givens rotator (or Householder reflector) in the explicit QR factorization depends only on the first column of the shifted matrix $A - \mu I$. Since A is upper Hessenberg, the first column has the form

$$(A - \mu I)\widehat{e}_1 = \begin{bmatrix} a_{11} - \mu \\ a_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where $\widehat{e}_1 = [1, 0, \ldots, 0]^T$. Thus, the first Givens rotation is completely determined by the vector

$$x = (A - \mu I)\widehat{e}_1.$$

Since $a_{21} \neq 0$ we can always construct a transformation Q_1^H based on first and second entries of x to annihilate a_{21}:

$$Q_1^H x = \begin{bmatrix} * \\ 0 \end{bmatrix}.$$

Refer to section A.1 to see how such transformation can be constructed by Householder reflections or Givens rotations. Instead of continuing the QR factorization, Francis immediately applies the corresponding similarity transformation

$$A^{(1)} = Q_1^H A Q_1.$$

The action of Q_1^H from the left affects only the first two rows of A, and the action of Q_1 from the right affects only the first two columns. Thus the Hessenberg structure is almost preserved except for a single nonzero entry created below the first subdiagonal,

$$A^{(1)} = \begin{bmatrix} * & * & * & * & \cdots \\ * & * & * & * & \cdots \\ \boxed{*} & * & * & * & \cdots \\ & * & * & * & \cdots \\ & & \ddots & \ddots & \end{bmatrix}.$$

Noting that we apply Q_1 on both sides of A and not $A - \mu I$. The boxed entry is called the *bulge*. It is the only entry violating the upper Hessenberg structure. The next step is to eliminate the bulge. Since it occupies position $(3, 1)$, we construct a second transformation Q_2 based on the second and third entries of the first column. Then

$$Q_2^H A^{(1)}$$

has a zero in position $(3, 1)$. We then apply Q_2 from the right to obtain the similarity transformation

$$A^{(2)} = Q_2^H A^{(1)} Q_2.$$

The entry $(3,1)$ disappears, but since the action of Q_2 from both sides affects rows 2 and 3 and columns 2 and 3, a new bulge is created one row down and one column to the right:

$$A^{(2)} = \begin{bmatrix} * & * & * & * & \cdots \\ * & * & * & * & \cdots \\ 0 & * & * & * & \cdots \\ & \boxed{*} & * & * & \cdots \\ & & \ddots & \ddots & \end{bmatrix}.$$

Repeating this procedure moves the bulge down-right at a time. At the k-th stage, the bulge occupies position $(k+2, k)$. A new transformation acting on rows and columns $k+1$ and $k+2$ removes it and creates a new bulge one row down and one column to the right. Eventually the bulge reaches the last position $(n-2, n)$. After applying the last transformation, the bulge disappears and the matrix is again upper Hessenberg. The complete similarity transformation is therefore

$$A_+ = Q^H A Q, \qquad Q = Q_1 Q_2 \cdots Q_{n-1},$$

without ever explicitly computing either the QR factorization of $A - \mu I$ or the product RQ. This process is also called *bulge-chasing* algorithm, because it creates a bulge and chases it from the matrix.

Exercise 5.1. Assume that A is upper Hessenberg but not properly upper Hessenberg. Show that if $a_{j+1,j} = 0$, the bulge that is supposed to appear at position $(j+1, j-1)$ at step $j-2$ of the budge chase does not in fact appear, and the budge chase ends permanently.

Another remarkable fact is that if A is proper upper Hessenberg, the matrix A_+ produced by this procedure is the same as the matrix obtained from one explicit shifted QR step. This is a consequence of the Implicit-Q Theorem [7, 1]. We state the theorem without proof here.

Theorem 5.1 (Implicit-Q Theorem)**.** Let $A, \hat{A}, \hat{Q}, \tilde{A}$ and $\tilde{Q} \in \mathbb{C}^{n\times n}$. Suppose that $\hat{A}$ is properly upper Hessenberg, $\tilde{A}$ is upper Hessenberg, and $\hat{Q}$ and $\tilde{Q}$ are unitary matrices, and

$$\hat{A} = \hat{Q}^H A \hat{Q}, \quad \tilde{A} = \tilde{Q}^H A \tilde{Q}.$$

Suppose further that the first columns of $\hat{Q}$ and $\tilde{Q}$ are proportional so that $\hat{q}_1 = d_1 \tilde{q}_2$ where $|d_1| = 1$. Then $\tilde{A}$ is also properly upper Hessenberg and there is a unitary diagonal matrix D such that

$$\hat{Q} = \tilde{Q} D, \quad \hat{A} = D^H \tilde{A} D.$$

This theorem states that a proper Hessenberg matrix is *uniquely* determined by its first column together with a unitary similarity transformation. Since both algorithms start by annihilating the same vector

$$(A - \mu I)\widehat{e}_1,$$

they generate the same similarity transformation (up to multiplication by diagonal matrices).

We started by A and perform one step to obtain $A_+ = Q^H A Q$. We call it a *Francis step* rather than a QR step because we do not form any explicit QR factorization [7, 8]. The algorithm of one Francis single-shift step is given in Algorithm 10 below.

Algorithm 10: Francis single-shift step

Input: Upper Hessenberg matrix $A \in \mathbb{C}^{n\times n}$; shift μ, matrix $Z \in \mathbb{C}^{n\times q}$, indicator $flag$.
Output: Upper Hessenberg matrix $A \leftarrow Q^H AQ$, updated matrix $Z \leftarrow ZQ$ if $flag = 1$.

% Generating the bulge
Set $x_1 = a_{11} - \mu$ and $x_2 = a_{21}$
Compute the transformation G so that $G^H \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$
Update $A_{1:2,1:\text{end}} \leftarrow G^H A_{1:2,1:\text{end}}$ % update rows 1 and 2
Update $A_{1:3,1:2} \leftarrow A_{1:3,1:2}\, G$ % update columns 1 and 2
if $flag = 1$ **then** $Z_{1:\text{end},\,1:2} \leftarrow Z_{1:\text{end},\,1:2}\, G$

% Chasing the bulge
for $k = 1 : n-2$ **do**
  Set $x_1 = a_{k+1,k}$ and $x_2 = a_{k+2,k}$
  Compute the transformation G so that $G^H \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$
  Set $p = \min(n, k+3)$
  Update $A_{k+1:k+2,\,k:\text{end}} \leftarrow G^H A_{k+1:k+2,\,k:\text{end}}$ % update rows $k+1$ and $k+2$
  Update $A_{1:p,\,k+1:k+2} \leftarrow A_{1:p,\,k+1:k+2}\, G$ % update columns $k+1$ and $k+2$
  if $flag = 1$ **then** $Z_{1:\text{end},\,k+1:k+2} \leftarrow Z_{1:\text{end},\,k+1:k+2}\, G$
end for

The algorithm includes also the update of a matrix Z, whose role will be discussed in Section 6 when we consider the computation of eigenvectors. Since Z is not needed for computing the eigenvalues alone, this update can be skipped here by setting $flag = 0$.

Cost of one step: Assume that the input matrix A is of size $n \times n$. At each step k of the above algorithm we need the product of a transformation G^H of size 2×2 by a submatrix of A of size $(n-k)\times 2$ to update rows $k+1$ and $k+2$. The cost of this update is $6(n-k)$ (why?). To update two columns $k+1$ and $k+2$, we must multiply a submatrix of size $(k+3) \times 2$ by a transformation G of size 2×2. This update costs for $6(k+3)$. The cost for initial updates for creating the bulge has the same formula with $k = 0$. Therefore, the overall cost for one Francis single-shift step is

$$6\sum_{k=0}^{n-2}[(n-k)+(k+3)] = 6\sum_{k=0}^{n-2}(n+3) \approx 6n^2.$$

We ignored the cost for computing all Givens rotations which is in total of order $\mathcal{O}(n)$.

After completing one Francis step, we check the magnitude of the last subdiagonal entry $a^+_{n,n-1}$ of A_+ and decide whether another iteration is needed. If the criterion

$$|a^+_{n,n-1}| \leqslant \varepsilon(|a^+_{n-1,n-1}| + |a^+_{n,n}|)$$

is satisfied, we accept

$$\lambda_n = a^+_{nn}$$

as an approximation of the eigenvalue, perform a one-dimensional deflation by removing the last row and column, and continue the algorithm on the remaining $(n-1)\times(n-1)$ Hessenberg matrix. A new shift is then computed for the smaller problem, and the process is repeated until all eigenvalues have been determined.

The algorithm of Francis method: The Francis single-shift algorithm for computing all eigenvalues of a matrix A (real or complex) is identical to Algorithm 7, except that the line performing the explicit QR step inside the loop is replaced by the Francis step given in Algorithm 10.

Cost of computing all eigenvalues: Assume that approximating the eigenvalue λ_{n-j} requires m_j Francis single-shift steps applied to a matrix of size $(n-j)\times(n-j)$, for $j = 0, 1, \ldots, n-1$. The total computational cost then is

$$6\sum_{j=0}^{n-1} m_j(n-j)^2 \leqslant 6M\sum_{j=0}^{n-1}(n-j)^2 \approx Mn^3,$$

where $M = \max_j m_j$. In practice, however, this estimate is pessimistic. After each deflation considerably fewer iterations are needed to compute the next eigenvalue. Typically, $m_0 \gg m_1 \gg \cdots \gg m_{n-1}$, so that the last few eigenvalues are often obtained in only one or two Francis steps.

Exercise 5.2. Assume that $A \in \mathbb{C}^{n\times n}$ is upper Hessenberg and Hermitian (that is, A is tridiagonal). Algorithm 10 is still applicable to such a matrix. However, the tridiagonal structure allows modifications in update lines in the algorithm to speed it up substantially. Modify Algorithm 10 to exploit the tridiagonal structure of A, and show that one Francis single-shift step can be carried out in $\mathcal{O}(n)$ flops instead of $\mathcal{O}(n^2)$. Then estimate the total computational cost of computing all eigenvalues of a Hermitian matrix by this algorithm.

5.2 Francis double-shift algorithm

As discussed earlier, a double-shift strategy is better in some situations. For example, when A is real and nonsymmetric, we apply a double-shift method to perform all computations in real arithmetic even though complex shifts are needed to compute complex eigenvalues. Francis double-shift algorithm is an efficient implementation of the QR double-shift algorithm given before. Rather than explicitly forming

$$B = (A - \bar{\mu}I)(A - \mu I),$$

and computing its QR factorization, the algorithm constructs the required similarity transformation directly using a sequence of orthogonal transformations. The starting vector is now

$$x = (A - \bar{\mu}I)(A - \mu I)\widehat{e}_1.$$

Since A is upper Hessenberg, the matrix product B has only one additional subdiagonal, so only the first three entries of x are nonzero. Indeed, x has the form

$$x = \begin{bmatrix} a_{11}^2 - 2\,\mathrm{Re}(\mu) + |\mu|^2 + a_{12}a_{21} \\ a_{21}(a_{11} + a_{22} - 2\,\mathrm{Re}(\mu)) \\ a_{32}a_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix} =: \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Consequently, it is sufficient to construct an orthogonal transformation acting on the first three entries. One can simply use a Householder reflection Q_1 of size 3×3. We can also use two consecutive Givens rotations G_1 and G_2 where the first rotation G_1^H annihilates the third entry x_3, and the second rotation G_2^H annihilates the resulting second entry. Then we define

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & G_1 \end{bmatrix} \begin{bmatrix} G_2 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{C}^{3\times 3}$$

In either Householder or Givens case, since $a_{21} \neq 0$ we can always construct such transformation Q_1. Then we have

$$Q_1^H x = \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Instead of continuing with the QR factorization of $(A - \bar{\mu}I)(A - \mu I)$, we immediately apply the similarity transformation

$$A^{(1)} = Q_1^T A Q_1.$$

Again note that, Q_1 acts on upper Hessenberg matrix A and not on $(A - \bar{\mu}I)(A - \mu I)$. Since Q_1 acts only on the first three rows and columns of A, the Hessenberg structure is preserved except for a triangle bulge containing three entries. In this case the bulge occupies positions $(1, 3)$, $(1, 4)$, and $(2, 4)$:

$$A^{(1)} = \begin{bmatrix} * & * & * & * & * & \cdots \\ * & * & * & * & * & \cdots \\ \boxed{*} & * & * & * & * & \cdots \\ \boxed{*} & \boxed{*} & * & * & * & \cdots \\ & & & * & * & \cdots \\ & & & & \ddots & \end{bmatrix}.$$

The next orthogonal transformation Q_2 acts to remove two boxed entries in the first column. If we then operate it on both sides of $A^{(1)}$ to get $A^{(2)} = Q_2^H A^{(1)} Q_2$, it affects rows and columns 2, 3, and 4 and moves the bulge one row down and one column to the right:

$$A^{(2)} = \begin{bmatrix} * & * & * & * & * & \cdots \\ * & * & * & * & * & \cdots \\ 0 & * & * & * & * & \cdots \\ 0 & \boxed{*} & * & * & * & \cdots \\ & \boxed{*} & \boxed{*} & * & * & \\ & & & & \ddots & \end{bmatrix}.$$

Repeating this procedure moves the bulges downward through the matrix until it finally disappears at the bottom of the matrix by applying the last transformation. The last transformation Q_{n-1} involves only a 2×2 transformation, while each of the others involves transformations of size 3×3. After the bulge has been chased out of the matrix, we obtain a new

upper Hessenberg matrix

$$A_+ = Q^T A Q,$$

where $Q = Q_1 Q_2 \cdots Q_{n-1}$. By the Implicit-$Q$ Theorem, this matrix is exactly the same as the one produced by the explicit double-shift QR step based on $(A-\bar{\mu}I)(A-\mu I)$, because A is assumed to be proper upper Hessenberg and both algorithms start from annihilating the same vector x. We summarize the Francis double-shift step in the algorithm below.

Algorithm 11: Francis double-shift step

Input: Upper Hessenberg matrix $A \in \mathbb{C}^{n\times n}$; shifts μ_1 and μ_2 (or μ and $\bar{\mu}$); matrix Z; indicator $flag$.
Output: Upper Hessenberg matrix $A \leftarrow Q^H A Q$, update matrix $Z \leftarrow ZQ$ if $flag = 1$.

% Generating the bulge
Set $x_1 = (a_{11}-\mu_1)(a_{11}-\mu_2) + a_{12}a_{21}$, $x_2 = a_{21}(a_{11}+a_{12}-(\mu_1+\mu_2))$, and $x_3 = a_{32}a_{21}$
Compute transformation G such that $G^H \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}$
Update $A_{1:3,1:\text{end}} \leftarrow G^H A_{1:3,1:\text{end}}$ % update rows 1, 2, 3
Update $A_{1:4,1:3} \leftarrow A_{1:4,1:3} G$ % update columns 1, 2 and 3
if $flag = 1$ **then** $Z_{1:\text{end},\,1:3} \leftarrow Z_{1:\text{end},\,1:3} G$

% Chasing the bulge
for $k = 1 : n-3$ **do**
  Set $x_1 = a_{k+1,k}$, $x_2 = a_{k+2,k}$, $x_3 = a_{k+3,k}$
  Compute the transformation G so that $G^H \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}$
  Set $p = \min(n, k+4)$
  Update $A_{k+1:k+3,\,k:\text{end}} \leftarrow G^H A_{k+1:k+3,\,k:\text{end}}$ % update rows $k+1$, $k+2$, and $k+3$
  Update $A_{1:p,\,k+1:k+3} \leftarrow A_{1:p,\,k+1:k+3} G$ % update columns $k+1$, $k+2$ and $k+3$
  if $flag = 1$ **then** $Z_{1:\text{end},\,k+1:k+3} \leftarrow Z_{1:\text{end},\,k+1:k+3} G$
end for
Set $x_1 = A_{n-1,n-2}$ and $x_2 = A_{n,n-2}$
Compute transformation G such that $G^H \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$
Update $A_{n-1:n,\,n-1:\text{end}} \leftarrow G^H A_{n-1:n,\,n-1:\text{end}}$ % update rows $n-1$ and n
Update $A_{1:n,\,n-2:n} \leftarrow A_{1:n,\,n-2:n} G$ % update columns $n-1$ and n
if $flag = 1$ **then** $Z_{1:\text{end},\,n-1:n} \leftarrow Z_{1:\text{end},\,n-1:n} G$.

This algorithm also computes the update of a matrix Z, whose role will be discussed in Section 6. We set $flag = 0$ if we do not need it.

Cost of one step: Each Francis double-shift step, if is applied on a matrix of size $n \times n$, requires $n-1$ transformations all of size 3×3 except the last one which is of size 2×2. At each iterate of the loop three rows of size $(n-k)$ and a three columns of size $k+4$ are updated. We can simply show that the total cost is $9n^2$ flops, ignoring the costs of order $\mathcal{O}(n)$. Although a double-shift step costs approximately 1.5 times more than a single-shift step, it simultaneously computes two eigenvalues and is therefore more efficient. In addition, for real nonsymmetric matrices, it performs all computations in real arithmetic.

We now update the shifts using the bottom-right 2×2 block of A_+ and perform another Francis double-shift step. As this process continues, the subdiagonal entry $a^+_{n-1,n-2}$ goes to

zero. Once the stopping criterion

$$|a^+_{n-1,n-2}| \leqslant \varepsilon\big(|a^+_{n-2,n-2}| + |a^+_{n-1,n-1}|\big),$$

is satisfies we set $a^+_{n-1,n-2}$ to zero and obtain the block form

$$A_+ = \left[\begin{array}{ccc|cc} & & & * & * \\ & \tilde{A}_+ & & \vdots & \vdots \\ & & & * & * \\ \hline & & 0 & a & b \\ & & & c & d \end{array}\right],$$

The eigenvalues of the trailing 2×2 block are then computed and accepted as the converged eigenvalues (in the case $\mu_1 = \mu$ and $\mu_2 = \bar{\mu}$ they are a complex conjugate pair). We perform a two-dimensional deflation by removing the last two rows and columns, and continue the iteration on the smaller Hessenberg matrix $\tilde{A}_+$ of size $(n-2) \times (n-2)$.

The algorithm of Francis method for real matrices: For a real matrix $A \in \mathbb{R}^{n\times n}$, the Francis algorithm is identical to Algorithm 9, except the line performing the explicit single-shift QR step is replaced by the Francis single-shift Algorithm 10, and the line performing the explicit double-shift QR step is replaced by the Francis double-shift Algorithm 11.

Remark 5.1. The double-shift strategy is not restricted to the computation of complex conjugate eigenvalue pairs. It can also be applied to compute any two eigenvalues of A, provided that two suitable shifts μ_1 and μ_2 are available. Algorithm 11 is indeed written for two shifts μ_1 and μ_2 where the case $\mu_1 = \mu$ and $\mu_2 = \bar{\mu}$ is a special case. The same idea extends naturally to a *multiple-shift* strategy, in which several eigenvalues are computed simultaneously. The main difference is that the created bulge becomes larger and requires transformations of bigger sizes to be chased through the matrix. Modern implementations of the Francis algorithm are based on such multiple-shift techniques, but we do not pursue this topic further.

6 Computing Eigenvectors

So far we have discussed how the Francis algorithm computes the eigenvalues of a matrix. We now briefly explain how the corresponding eigenvectors can be obtained. First we note that if $B = Q^H AQ$ for a unitary matrix Q, and w is an eigenvector of B associated to eigenvalue λ then $v = Qw$ is an eigenvector of A associated to the same eigenvalue λ.

6.1 Hermitian matrices

Suppose that A is Hermitian. We first reduce A to a Hessenberg (tridiagonal) matrix

$$A_0 = U^H AU, \tag{6.1}$$

using the Hermitian version of Algorithm 5 (see Exercise 4.2), where U is the unitary matrix. The cost is $\mathcal{O}(n^2)$. Also, instead of storing the matrix U explicitly we store Householder

vectors $u_1, \ldots, u_{n-2}$. Then any matrix product with U can be carried out with $4n^2$ flops instead of $2n^3$.

Each Francis step then is a unitary similarity transformation of the form

$$A_{m+1} = Q_m^H A_m Q_m, \quad Q_m = G_1 G_2 \cdots G_{n-1},$$

where G_k are transformations constructed to chase the budge and eliminate it. Assume that after m_0 Francis steps the last subdiagonal entry $a_{n,n-1}^{(m_0)}$ becomes negligible, so that $\lambda_n = a_{nn}^{(m_0)}$ is accepted as an eigenvalue. The accumulated similarity transformation is

$$A_{m_0} = Z_0^H A_0 Z_0, \quad Z_0 = Q_0 Q_1 \cdots Q_{m_0-1},$$

Since the last row and column are now decoupled, the last coordinate vector $\widehat{e}_n$ is an eigenvector of A_{m_0} corresponding to the eigenvalue $\lambda_n = a_{nn}^{(m_0)}$. Hence, the corresponding eigenvector of A_0 is

$$z_n = Z_0 \widehat{e}_n \quad \text{(last column of } Z_0\text{)}$$

and the eigenvector of the original matrix A is

$$v_n = U z_n.$$

Cost of computing v_n: To accumulate the unitary matrix Z_0, we initialize it by setting $Z_0 = I$ before the first Francis step. During each Francis step, we apply $n-1$ transformations. After computing each transformation G_k, we update $Z_0 \leftarrow Z_0 G_k$. Since G_k acts only on two columns of Z_0, only $2n$ entries are modified. Each modified entry requires three flops, so applying one transformation costs about $6n$ flops. Consequently, accumulating Z_0 during one Francis step requires $6n(n-1) \approx 6n^2$ flops. For the next Francis step, we do not restart with $Z = I$; instead, we continue accumulating the new transformations into the current matrix Z_0. After m_0 Francis steps, the total cost of accumulating Z_0 is approximately $6m_0n^2$. On the other hand, Exercise 5.2 shows that computing the eigenvalue λ_n itself requires only $m_0\mathcal{O}(n)$ flops for Hermitian matrices, which is one order of magnitude smaller (Z_0 is not tridiagonal). To reduce the cost, one usually does not accumulate Z_0 while computing λ_n. Instead, after λ_n has converged, it is used as an exact shift and a final Francis step is performed on matrix A_0 (not A_{m_0}) to compute the corresponding eigenvector. In exact arithmetic, a single Francis step is sufficient; in finite precision arithmetic, one additional steps are usually enough. This approach is known as the *ultimate shift strategy*. With this strategy, the cost of computing the eigenvector z_n is reduced to about $6n^2$ flops (or $12n^2$ flops if two Francis steps are required), which is much more reasonable. We need additional matrix-vector product Uz_n to compute v_n which costs $4n$ flops, if we use Householder vectors implicitly.

After doing the ultimate shift step (assume that $m_0 = 1$) we set

$$A_1 = Z_0^H A_0 Z_0. \tag{6.2}$$

and go after the next eigenvector. Assume that A_1 has the form (up to the prescribed tolerance)

$$A_1 = \begin{bmatrix} \tilde{A}_1 & 0 \\ 0 & \lambda_n \end{bmatrix}.$$

After deflation, the Francis algorithm continues on the leading $(n-1)\times(n-1)$ submatrix $\tilde{A}_1$. The eigenvalue λ_{n-1} is computed and used as an ultimate shift to compute the matrix $\tilde{Z}_1$. The last column of $\tilde{Z}_1$, say $\tilde{z}_{n-1}$, is an eigenvector of $\tilde{A}_1$ associated to eigenvalue λ_{n-1}. Since the eigenvector z_{n-1} of A_0 is of size n, the transformation $\tilde{Z}_1$ must therefore be embedded as

$$Z_1 = \begin{bmatrix} \tilde{Z}_1 & 0 \\ 0 & 1 \end{bmatrix}.$$

We can easily see that (up to the prescribed tolerance)

$$A_2 = Z_1^H A_1 Z_1 = \begin{bmatrix} \tilde{Z}_1^H \tilde{A}_1 \tilde{Z}_1 & 0 \\ 0 & \lambda_n \end{bmatrix} =: \begin{bmatrix} \tilde{A}_2 & 0 & 0 \\ 0 & \lambda_{n-1} & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}.$$

This relation shows that $\widehat{e}_{n-1}$ is the eigenvector of A_2 associated to eigenvalue λ_{n-1}. Consequently, $Z_1\widehat{e}_{n-1}$ is the eigenvector of A_1 associated to eigenvalue λ_{n-1}. Then, according to similarity relation (6.2),

$$z_{n-1} := Z_0 Z_1 \widehat{e}_{n-1} \tag{6.3}$$

is the eigenvector of A_0 associated to eigenvalue λ_{n-1}, and finally according to (6.1),

$$v_{n-1} = U z_{n-1}$$

is the eigenvector of the original matrix A associated to eigenvalue λ_{n-1}.

A direct computation of the matrix product $Z_0 Z_1 =: Z$ requires approximately $2n^3$ flops, which is prohibitively expensive for computing just one eigenvector. We fix this problem as follows. Suppose that Z_0 is partitioned as

$$Z_0 = \begin{bmatrix} \hat{Z}_0 & z_n \end{bmatrix},$$

where $\hat{Z}_0 \in \mathbb{C}^{n\times(n-1)}$, and z_n is its last column (the eigenvector of A_0). Then we can write $Z = Z_0 Z_1 = [\hat{Z}_0\tilde{Z}_1 \;\; z_n]$. Since $\tilde{Z}_1$ is represented as the product of local transformations in the second cycle of the algorithm, we easily input the matrix $\hat{Z}_0$ instead of the identity matrix as an initial transformation to this cycle to get the product $\hat{Z}_0\tilde{Z}_1$. According to (6.3), the last columns of this product is just z_{n-1}. Thus we can write

$$Z = Z_0 Z_1 = [Z_{:,1:n-2} \;\; z_{n-1} \;\; z_n]$$

where $Z_{:,1:n-2}$ is $\hat{Z}_0\tilde{Z}_1$ except its last column.

We repeat this process until all converged eigenvalues have been used to compute the corresponding eigenvectors of A. In programming, we start with the identity matrix I and after the first cycle we obtain the matrix $Z\,(= Z_0)$ where its last columns is the eigenvector z_n. At the subsequent cycles, for $j = n-1, n-2, \ldots, 2$, we compute the eigenvector z_j by applying the Francis algorithm with shift λ_j to the leading $j \times j$ submatrix, while updating only the first j columns of Z. Consequently, the cost of computing each successive eigenvector decreases as the size of the active submatrix becomes smaller. At the end of the process, we obtain

$$Z = \begin{bmatrix} z_1 & \cdots & z_{n-1} & z_n \end{bmatrix},$$

whose columns are the eigenvectors of the tridiagonal matrix A_0. Finally, we recover the eigenvectors of the original matrix by setting

$$V = UZ.$$

The columns of V are the eigenvectors of A. If we define $D = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, then we obtain the spectral decomposition

$$A = VDV^H.$$

We have already included the accumulation of the matrix Z in Algorithm 10, provided that the indicator $flag$ is set to 1. Using this option, the following algorithm computes all eigenvectors of a Hermitian matrix A.

Algorithm 12: Eigenvectors of Hermitian matrices

Input: Hermitian matrix $A \in \mathbb{C}^{n\times n}$; Eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ of A.
Output: Eigenvector matrix $V = [v_1, \ldots, v_n] \in \mathbb{C}^{n\times n}$.

Compute the tridiagonal transformation $A \leftarrow U^H AU$
Set $Z = I$
for $j = n$ **downto** 2 **do**
    Pass A, $\mu = \lambda_j$, $Z_{:,\,1:j}$, and $flag = 1$ to Algorithm 10 and receive the updates A and $Z_{:,\,1:j}$
    Update $A \leftarrow A_{1:j-1,\,1:j-1}$
end for
Compute $V = UZ$

For improved accuracy, we may repeat the Francis step by making one additional call to Algorithm 10.

Cost of computing all eigenvectors: If only one Francis step is performed for each eigenvector (as in the algorithm), then the cost of computing z_n is approximately $6n^2$ flops, the cost of computing z_{n-1} is about $6n(n-1)$ flops, and so on, down to $6n$ flops for computing z_2. The first eigenvector z_1 comes for free. Hence, the total cost of computing the matrix Z is $6n\big(n + (n-1) + \cdots + 1\big) = 3n^2(n+1) \approx 3n^3$ flops. Finally, forming $V = UZ$ requires approximately $4n^2$ flops. Therefore, the total cost of computing all eigenvectors is about $3n^3$ flops.

6.2 Non-Hermitian matrices

In the Hermitian case, the Francis algorithm reduces the tridiagonal matrix A_0 to the diagonal matrix $A_n = D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, whose eigenvector matrix is simply the identity matrix $I = [\widehat{e}_1\ \widehat{e}_2\ \cdots\ \widehat{e}_n]$. Consequently, the accumulated unitary matrix Z is the eigenvector matrix of A_0.

For a non-Hermitian matrix the situation is slightly different. After reducing A to upper Hessenberg form, $A_0 = U^H AU$, the Francis algorithm converges to an upper triangular matrix $A_n = T$. The eigenvectors of T are not known a priori and must be computed separately. If V_T denotes the eigenvector matrix of T, then $V_0 = ZV_T$ is the eigenvector matrix of A_0 because of similarity transformation $T = Z^H A_0 Z$. Finally, $V = UV_0$ is the eigenvector matrix of the original matrix A.

The matrix Z is accumulated in the same way as in the Hermitian case. There is, however, one important difference for updates of submatrices of A. After the first deflation we obtain

$$A_1 = \begin{bmatrix} \tilde{A}_1 & w \\ 0 & \lambda_n \end{bmatrix}.$$

For Hermitian matrices the vector w is zero, whereas in the non-Hermitian case $w \neq 0$. Therefore, after deflation the subsequent Francis steps applied to the leading block $\tilde{A}_1$ must also update the vector w. Otherwise, the final triangular matrix T would be incorrect. This should be continued in subsequent cycles and upper-right block should also be updated.

This modification is straightforward to implement. Instead of passing only the leading principal submatrix

$$A_{1:j-1,\,1:j-1}$$

to Algorithm 10 during the jth cycle, we pass the rectangular matrix

$$A_{1:j-1,\,1:\mathrm{end}},$$

so that every row update is applied to all remaining columns. This is the reason why the row-update lines of Algorithm 10 uses the notation "1 : end", so this algorithm works for this general case as well.

Algorithm 13: Eigenvectors of non-Hermitian matrices

Input: Non-Hermitian matrix $A \in \mathbb{C}^{n\times n}$; Eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ of A.
Output: Eigenvector matrix $V = [v_1, \ldots, v_n] \in \mathbb{C}^{n\times n}$.

Compute the Hessenberg transformation $A \leftarrow U^H A U$ using Householder reflections
Set $Z = I$ and $T = 0$
for $j = n$ **downto** 2 **do**
    Pass A, $\mu = \lambda_j$, $Z_{:,\,1:j}$, and $flag = 1$ to Algorithm 10 and receive the updates A and $Z_{:,\,1:j}$
    Set $T(j,:) = A(j,:)$
    Update $A \leftarrow A_{1:j-1,\,1:\mathrm{end}}$
end for
Set $T(1,:) = A(1,:)$
Compute the eigenvector matrix V_T for triangular matrix T
Compute $V = UV_TZ$

It remains to compute the eigenvectors of the triangular matrix T. For simplicity, assume that the diagonal entries of T (eigenvalues) are distinct. To compute an eigenvector corresponding to the eigenvalue t_{kk}, we solve the singular system

$$(T - t_{kk}I)v = 0,$$

for a nonzero vector v. Partition

$$T - t_{kk}I = \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix}, \qquad v = \begin{bmatrix} \tilde{v} \\ \hat{v} \end{bmatrix},$$

where S_{11} is of size $(k-1)\times(k-1)$ and it is nonsingular, and S_{22} is of size $(n-k+1)\times(n-k+1)$ and it is singular with a zero top-left entry. We therefore choose $\hat{v} = \widehat{e}_1 = [1, 0, \ldots, 1]^T$ for which we have $T_{22}\hat{v} = 0$, and then we determine $\tilde{v}$ from the triangular system

$$S_{11}\tilde{v} = -S_{12}e_1.$$

Thus, each eigenvector of T is obtained by solving one upper triangular system, requiring $\mathcal{O}(n^2)$ flops.

The situation becomes more complicated when some eigenvalues are repeated. We do not pursue this case further here.

Exercise 6.1. Given all eigenvalues, derive the computational cost of computing all eigenvectors of a non-Hermitian matrix using the Francis single-shift algorithm.

6.3 Eigenvectors of real nonsymmeric matrices

For real nonsymmetric matrices, we preferably use the Francis double-shift algorithm so that the whole process is carried out in real arithmetic for computing eigenvalues. The algorithm converges to a quasi-upper triangular matrix

$$T = Z^H A_0 Z,$$

whose diagonal blocks are of size either 1×1 or 2×2. The unitary matrix Z is accumulated exactly as in the Hermitian case. The only difference is that, after each deflation, either one or two columns of Z are removed from further updates, depending on whether the converged diagonal block is of size 1×1 or 2×2. Algorithm 11 already supports the accumulation of Z when the input parameter $flag$ is set to 1.

After the algorithm has converged, we compute the eigenvector matrix V_T of the quasi-upper triangular matrix T. The eigenvector matrix of A_0 is then $V_0 = ZV_T$, and the eigenvector matrix of the original matrix A is obtained as $V = UZV_T$. Although the matrices U and Z are real, the matrix V_T is generally complex because of the 2×2 diagonal blocks. Writing $V_T = \tilde{V}_T + i\hat{V}_T$, where $\tilde{V}_T$ and $\hat{V}_T$ are real matrices, yields

$$V = (UZ)\,\tilde{V}_T + i\,(UZ)\,\hat{V}_T.$$

Hence, all matrix-matrix products can still be performed in real arithmetic. Complex arithmetic is required only when computing the eigenvectors of the quasi-upper triangular matrix T.

Algorithm 14: Eigenvectors of non-symmetric real matrices

Input: Non-symmetric $A \in \mathbb{R}^{n\times n}$; Eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ of A.
Output: Eigenvector matrix $V = [v_1, \ldots, v_n] \in \mathbb{C}^{n\times n}$.

Compute the Hessenberg transformation $A \leftarrow U^H AU$
Set $Z = I$ and $T = 0$ and $j = n$
while $j \geqslant 2$ **do**
  Pass A, $\mu = \lambda_j$, $Z_{:,\,1:j}$, and $flag = 1$ to Algorithm 11 and receive the updates A and $Z_{:,\,1:j}$
  if λ_j *is real* **then** $d = 1$ **else** $d = 2$
  Set $T(j-d+1:j,:) = A(j-d+1:j,:)$
  Update $A \leftarrow A_{1:j-d,\,1:\text{end}}$
  Update $j \leftarrow j - d$
end while
if $j = 1$ **then** $T(1,:) = A(1,:)$
Compute the eigenvector matrix V_T for quasi-triangular matrix T
Compute $W = UZ$ and set $\tilde{V}_T = \mathrm{Re}(V_T)$ and $\hat{V}_T = \mathrm{Im}(V_T)$
Set $V = W\tilde{V}_T + i\,W\hat{V}_T$

It remains to see how the eigenvectors of quasi-triangular form

$$T = \begin{bmatrix} T_{11} & * & \cdots & * \\ & T_{22} & \ddots & \vdots \\ & & \ddots & * \\ & & & T_{pp} \end{bmatrix},$$

where each diagonal block T_{kk} is either of size 1×1 or 2×2, can be computed. We assume that the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ are distinct and are given in the order of the diagonal blocks. Thus, a 1×1 block contributes one eigenvalue, while a 2×2 block contributes its two (complex conjugate) eigenvalues.

To compute an eigenvector corresponding to an eigenvalue λ, let T_{kk} be the diagonal block containing λ. Partition $T - \lambda I$ conformably as

$$T - \lambda I = \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix}, \qquad v = \begin{bmatrix} \tilde{v} \\ \hat{v} \end{bmatrix},$$

where the leading diagonal block of S_{22} is $T_{kk} - \lambda I$, with T_{kk} being the diagonal block associated with the eigenvalue λ. Since λ is an eigenvalue of T_{kk}, the matrix S_{22} is singular. We first compute a nonzero vector v_2 satisfying

$$S_{22}\hat{v} = 0.$$

If T_{kk} is a 1×1 block, then simply set $\hat{v} = [1, 0, \ldots, 0]^T$. If T_{kk} is a 2×2 block, then $\hat{v} = [w, 0, \ldots, 0]^T$ where w is obtained by solving the singular 2×2 system

$$(T_{kk} - \lambda I)w = 0.$$

Since the system has rank one, any nonzero vector in its nullspace may be chosen. The remaining part of the eigenvector is obtained by solving

$$S_{11}\tilde{v} = -S_{12}\hat{v}.$$

The matrix S_{11} is block upper triangular with diagonal blocks of size at most 2×2. Assuming that λ is a simple eigenvalue, S_{11} is nonsingular, and the above system can be solved by block back substitution. Starting from the last diagonal block of S_{11} and proceeding upward, each step requires solving either a 1×1 or a 2×2 linear system.

Computing one eigenvector requires one block back substitution, whose cost is $\mathcal{O}(n^2)$ flops. Consequently, all eigenvectors of T can be computed in $\mathcal{O}(n^3)$ operations.

7 Why does QR algorithm work?

This section will be added ...

8 Additional Exercises

Exercise 8.1. Let A be a Hermitian matrix with eigenvalues -5, -1, 2, 2.5, 7. Shifted inverse iteration is used with a fixed shift μ.

1. Which eigenvalue is obtained for each of the shifts $\mu = 0$, $\mu = 2.2$, $\mu = 4$, $\mu = 10$?
2. For each case, compute the asymptotic convergence factor.
3. Find the interval of values of μ for which the iteration converges to the eigenvalue 2.
4. Explain what happens if μ is exactly equal to an eigenvalue.

Exercise 8.2. Let $A = A^H$ and let x be normalized, $\|x\|_2 = 1$. Define

$$\mu = x^H A x, \qquad r = Ax - \mu x.$$

1. Prove that $x^H r = 0$.
2. Show that the Rayleigh quotient is the value of α that minimizes $\|Ax - \alpha x\|_2$ over all $\alpha \in \mathbb{C}$.
3. Let $\lambda_1, \dots, \lambda_n$ be the eigenvalues of A. Prove that

$$\min_j |\lambda_j - \mu| \le \|r\|_2.$$

   Thus, a small residual guarantees that the Rayleigh quotient is close to at least one eigenvalue of A.

Exercise 8.3. Suppose that $A \in \mathbb{R}^{n\times n}$ is diagonalizable and has eigenvalues

$$\lambda_1 = -\lambda_2, \qquad |\lambda_1| = |\lambda_2| > |\lambda_3| \ge \cdots \ge |\lambda_n|.$$

Assume that initial vector x_0 has nonzero components in the directions of both v_1 and v_2.

1. Show that the normalized power iteration does not, in general, converge to a single eigenvector.
2. Describe the asymptotic behavior of the even and odd subsequences $\{x_{2m}\}$ and $\{x_{2m+1}\}$.
3. Show that the two-dimensional subspace $\operatorname{span}\{v_1, v_2\}$ is nevertheless identified asymptotically by the iteration.

Exercise 8.4. Consider

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}, \qquad x_0 = \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix}, \qquad 0 < |\theta_0| \ll 1.$$

Apply one step of Rayleigh quotient inverse iteration.

1. Compute the Rayleigh quotient μ_0 explicitly.
2. Write the direction of the new iterate x_1 in the form
$$x_1 = \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix}.$$
3. Show that $\tan\theta_1 = -\tan^3\theta_0$.
4. Explain how this relation demonstrates the cubic convergence of Rayleigh quotient iteration for Hermitian matrices.

Exercise 8.5. Prove that if the Hermitian matrix A is positive definite and B is unitarily similar to A then B is also positive definite.

In the following exercises we introduce other classes of matrices for which the Schur decomposition reduces to a spectral decomposition.

Exercise 8.6. A matrix A is called *skew Hermitian* if $A^H = -A$.

(a) Prove that if A is skew Hermitian and B is unitarily similar to A then B is skew Hermitian.

(b) Prove that the Schur decomposition of a skew Hermitian matrix A reduces to spectral decomposition $A = Q\Lambda Q^H$ for a diagonal matrix Λ and a unitary matrix Q.

(c) Show that eigenvalues of A are pure imaginary (i.e. with zero real part, or $\bar{\lambda} = -\lambda$).

Exercise 8.7. Recall that a nonsingular matrix A is called *unitary* if $A^H = A^{-1}$. For real matrices A is called *orthogonal* if $A^T = A^{-1}$.

(a) Prove that if A is unitary and B is unitarily similar to A then B is also unitary.

(b) Prove that a matrix T that is both upper triangular and unitary, must be a diagonal matrix.

(c) Prove that the Schur decomposition of a unitary matrix A reduces to spectral decomposition $A = Q\Lambda Q^H$ for a diagonal matrix Λ and a unitary matrix Q.

(d) Prove that the eigenvalues of a unitary matrix satisfy $\bar{\lambda} = \lambda^{-1}$. Equivalently $\bar{\lambda}\lambda = 1$ or $|\lambda| = 1$, that is eigenvalues lie on the unit circle in the complex plane.

Hermitian, skew Hermitian, and unitary matrices belong to a bigger class of matrices called *normal matrices*, where $A^H A = AA^H$.

Exercise 8.8. Assume that $A \in \mathbb{C}^{n\times n}$ is a normal matrix.

(a) Prove that if A is normal and B is unitarily similar to A then B is also normal.

(b) Prove that a matrix T that is both upper triangular and normal, must be a diagonal matrix. *Hint:* Use induction on n. Write T in partitioned form

$$T = \begin{bmatrix} t_{11} & w^* \\ & T_1 \end{bmatrix}$$

and show that if T is normal then $w = 0$.

(c) Prove that every diagonal matrix is normal.

(d) Prove that the Schur decomposition of a normal matrix A reduces to spectral decomposition $A = Q\Lambda Q^H$ for a diagonal matrix Λ and a unitary matrix Q.

(e) Prove the other side of item (d): If A is unitarily similar to a diagonal matrix then A is normal.

The above exercise proves a known result in linear algebra: a matrix $A \in \mathbb{C}^{n\times n}$ is normal if and only if A is unitarily similar to a diagonal matrix.

Exercise 8.9. Let $A \in \mathbb{C}^{n\times n}$ is a non-diagonalizable matrix. Using the Schur decomposition Theorem 3.2 prove that given any $\epsilon > 0$ there exists a diagonalizable matrix A_ϵ such that $\|A - A_\epsilon\|_2 \leqslant \epsilon$. This shows that the set of diagonalizable matrices is dense in $\mathbb{C}^{n\times n}$.

Exercise 8.10. Let H be an upper Hessenberg matrix and suppose $H = QR$ is its QR factorization, where Q is unitary and R is upper triangular.

1. Show that Q is upper Hessenberg.
2. Prove that $H_+ = RQ$ is again upper Hessenberg.
3. Show that the same conclusion holds for a shifted QR step

$$H - \mu I = QR, \qquad H_+ = RQ + \mu I.$$

4. Explain why this structural property is essential for making the QR algorithm computationally practical.

Exercise 8.11. Let $H \in \mathbb{C}^{n\times n}$ be upper Hessenberg and suppose that, for some k, we have $h_{k+1,k} = 0$.

1. Show that H has the block upper triangular form

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix},$$

where $H_{11} \in \mathbb{C}^{k\times k}$.

2. Prove that the eigenvalues of H are the union of the eigenvalues of H_{11} and H_{22}.
3. Explain why a sufficiently small subdiagonal entry can be set equal to zero in the QR algorithm and why this reduces the remaining computational work.

Exercise 8.12. Let $A \in \mathbb{R}^{n\times n}$ and let $\mu = \alpha + i\beta$, and $\beta \neq 0$.

1. Show directly that $B := (A - \mu I)(A - \overline{\mu} I)$ is real.
2. Show that this matrix commutes with A, i.e. $AB = BA$.
3. If $Av = \lambda v$, show that
$$(A - \mu I)(A - \overline{\mu} I)v = (\lambda - \mu)(\lambda - \overline{\mu})v.$$
4. Explain why treating the conjugate shifts simultaneously allows the QR double-shift algorithm (or Francis double-shift algorithm) to work entirely in real arithmetic.

Exercise 8.13. Let H be an upper Hessenberg matrix and let μ be a shift. In an explicit shifted QR step,
$$H - \mu I = QR.$$

1. Show that the first column q_1 of Q is determined, up to a factor of modulus one, by the first column of $H - \mu I$.
2. Show that this first column has only its first two entries possibly nonzero.
3. Hence explain why a single Givens rotation acting on the first two coordinates can start an implicit Francis single-shift step.
4. After this initial transformation, explain why a "bulge" appears below the Hessenberg band and why it must be chased through the matrix.

Exercise 8.14. Consider
$$T = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & 3 & 4 & 1 \\ 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 7 \end{bmatrix}.$$

1. Without computing a characteristic polynomial, determine all eigenvalues of T.
2. Compute an eigenvector associated with $\lambda = 5$ using the triangular procedure described in Section 6.
3. Compute an eigenvector associated with $\lambda = 7$.
4. Why is back substitution possible without pivoting in these two computations?
5. Explain what difficulty may arise in this procedure when an eigenvalue is repeated.

Exercise 8.15. Suppose that a dense Hermitian matrix $A \in \mathbb{C}^{n\times n}$ is reduced to tridiagonal form before applying the Francis algorithm.

1. Explain why one Francis step on the tridiagonal matrix can be performed in $O(n)$ operations, whereas a step on a general Hessenberg matrix requires $O(n^2)$ operations.
2. Suppose that at most M Francis steps are required before each deflation. Estimate the total cost of computing all eigenvalues after the tridiagonal reduction.
3. Compare this cost with the cost of the initial reduction from dense to tridiagonal form.
4. The lecture notes show that computing all eigenvectors requires $O(n^3)$ additional work. Explain why computing eigenvectors can therefore be considerably more expensive than computing only the eigenvalues once the tridiagonal form has been obtained.

A Appendix

A.1 Householder and Givens transformations

Suppose that we are given a nonzero vector $x \in \mathbb{C}^n$ and wish to construct a unitary transformation that annihilates all entries of x below the first. Two classical techniques for accomplishing this are Householder reflections and Givens rotations.

Householder reflections. Householder's idea is to construct a reflector that maps the vector x to a multiple of the first coordinate vector. To this end, define

$$u = x + \alpha \widehat{e}_1, \qquad \alpha = e^{i\theta}\|x\|_2, \qquad \theta = \operatorname{angle}(x_1),$$

where $\widehat{e}_1 = [1, 0, \ldots, 0]^T$ is the first coordinate vector. The corresponding Householder reflector is

$$P = I - \frac{2}{u^H u} u u^H.$$

The matrix P is both Hermitian and unitary,

$$P^H = P, \qquad P^H P = I,$$

and satisfies

$$Px = -\alpha \widehat{e}_1.$$

Consequently, a single Householder reflection annihilates every entry of x except the first.

Givens rotations. An alternative approach is based on Givens rotations, which eliminate one entry at a time. Consider first the vector

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \neq 0.$$

We seek a 2×2 unitary matrix of the form

$$G = \begin{bmatrix} c & s \\ -\bar{s} & c \end{bmatrix},$$

where $c \in \mathbb{R}$, $s \in \mathbb{C}$, and

$$c^2 + |s|^2 = 1,$$

such that

$$G \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}.$$

If $x_2 = 0$, we simply choose $c = 1$, $s = 0$, $\alpha = x_1$. If $x_1 = 0$, we take $c = 0$, $s = 1$, and $\alpha = x_2$. Otherwise, let

$$\rho = \sqrt{|x_1|^2 + |x_2|^2}, \qquad \beta = \frac{x_1}{|x_1|},$$

and define

$$c = \frac{|x_1|}{\rho}, \qquad s = \frac{\beta \bar{x}_2}{\rho}, \qquad \alpha = \beta\rho.$$

Then c is real and nonnegative,

$$c^2 + |s|^2 = 1,$$

and a straightforward computation verifies that

$$G \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}.$$

For a vector $x \in \mathbb{C}^n$, a sequence of Givens rotations is applied successively. Starting with the last two components, we construct a rotation that annihilates the last entry:

$$G_{n-1} \begin{bmatrix} x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \alpha_{n-1} \\ 0 \end{bmatrix}.$$

Next, we eliminate the $(n-1)$st entry by constructing a rotation satisfying

$$G_{n-2} \begin{bmatrix} x_{n-2} \\ \alpha_{n-1} \end{bmatrix} = \begin{bmatrix} \alpha_{n-2} \\ 0 \end{bmatrix}.$$

Proceeding in this manner, we continue upward until the first component is reached. After $n-1$ rotations, all entries below the first have been eliminated.

If $G_k\circ$ denotes the rotation acting on rows k and $k+1$ only, then the resulting transformation is

$$G = G_1 \circ G_2 \circ \cdots \circ G_{n-1} \circ I,$$

and

$$Gx = \begin{bmatrix} \alpha_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Householder reflections and Givens rotations each have their advantages. A Householder reflector annihilates an entire subvector in a single transformation and requires approximately $4n$ flops. In contrast, eliminating the same subvector with Givens rotations requires $n-1$ successive rotations and approximately $6n$ flops. Consequently, Householder reflections are generally preferred for reducing dense matrices. On the other hand, Givens rotations affect only two rows (or columns) at a time. This locality makes them particularly well suited for sparse matrices and for applications in which individual nonzero entries must be eliminated without significantly altering the remaining matrix structure.

References

[1] Å Björck. *Numerical Methods in Matrix Computations*. Springer, 2013.

[2] J. G. F. Francis. The QR transformation: A unitary analogue to the LR transformation. part I. *The Computer Journal*, 4(3):265–271, 1961.

[3] J. G. F. Francis. The QR transformation. part II. *The Computer Journal*, 4(4):332–345, 1962.

[4] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 1(4):555–570, 1961. In Russian.

[5] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice–Hall, 1980. Reprinted by SIAM, 1998.

[6] H. Rutishauser. Solution of eigenvalue problems with the LR-transformation. volume 49 of *National Bureau of Standards Applied Mathematics Series*, pages 47–81. U.S. Government Printing Office, Washington, DC, 1958.

[7] D. S. Watkins. *Fundamentals of Matrix Computations*. John Wiley & Sons, Hoboken, NJ, 2nd edition 2002, 3rd edition, 2010.

[8] D. S. Watkins. Francis's algorithm. *Amer. Math. Monthly*, 118(5):387–403, 2011.