Matrix Analysis and Algorithms

Andrew Stuart Jochen Voss

4th August 2009

Introduction

The three basic problems we will address in this book are as follows. In all cases we are given as data a matrix $A \in \mathbb{C}^{m\times n}$, with $m \geq n$ and, for the first two problems, the vector $b \in \mathbb{C}^m$. (SLE) denotes *simultaneous linear equations*, (LSQ) denotes *least squares* and (EVP) denotes *eigenvalue problem.*

(SLE) $(m = n)$	(LSQ) $(m \geq n)$	(EVP) $(m = n)$
find $x \in \mathbb{C}^n$:	find $x \in \mathbb{C}^n$:	find $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$:
$Ax = b$	$\min_{x\in\mathbb{C}^n} \|Ax - b\|_2^2$	$Ax = \lambda x, \quad \|x\|_2^2 = 1$

The book contains an introduction to matrix analysis, and to the basic algorithms of numerical linear algebra. Further results can be found in many text books. The book of Horn and Johnson [HJ85] is an excellent reference for theoretical results about matrix analysis; see also [Bha97]. The subject of linear algebra, and matrix analysis in particular, is treated in an original and illuminating fashion in [Lax97]. For a general introduction to the subject of numerical linear algebra we recommend the book by Trefethen and Bau [TB97]; more theoretical treatments of the subject can be found in Demmel [Dem97], Golub and Van Loan [GL96] and in Stoer and Bulirsch [SB02]. Higham's book [Hig02] contains a wealth of information about stability and the effect of rounding errors in numerical algorithms; it is this source that we used for almost all theorems we state concerning backward error analysis. The book of Saad [Saa97] covers the subject of iterative methods for linear systems. The symmetric eigenvalue problem is analysed in Parlett [Par80].

Acknowledgement

We are grateful to Menelaos Karavelas, Ian Mitchell and Stuart Price for assistance in the typesetting of this material. We are grateful to a variety of students at Stanford University (CS237A) and at Warwick University (MA398) for many helpful comments which have significantly improved the notes.

Contents

Chapter 1

Vector and Matrix Analysis

The purpose of this chapter is to summarise the fundamental theoretical results from linear algebra to which we will frequently refer, and to provide some basic theoretical tools which we will use in our analysis. We study vector and matrix norms, inner-products, the eigenvalue problem, orthogonal projections and a variety of special matrices which arise frequently in computational linear algebra.

1.1 Vector Norms and Inner Products

Definition 1.1. A *vector norm* on $\mathbb{C}^n$ is a mapping $\|\cdot\|\colon \mathbb{C}^n \to \mathbb{R}$ satisfying

a) $\|x\| \geq 0$ for all $x \in \mathbb{C}^n$ and $\|x\| = 0$ iff $x = 0$,

b) $\|\alpha x\| = |\alpha|\|x\|$ for all $\alpha \in \mathbb{C}, x \in \mathbb{C}^n$, and

c) $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{C}^n$.

Remark. The definition of a norm on $\mathbb{R}^n$ is identical, but with $\mathbb{C}^n$ replaced by $\mathbb{R}^n$ and $\mathbb{C}$ replaced by $\mathbb{R}$.

Examples.

- the p-norm for $1 \leq p < \infty$:

$$\|x\|_p = \Big(\sum_{j=1}^n |x_j|^p\Big)^{1/p} \quad \forall x \in \mathbb{C}^n;$$

- for $p = 2$ we get the Euclidean norm:

$$\|x\|_2 = \sqrt{\sum_{j=1}^n |x_j|^2} \quad \forall x \in \mathbb{C}^n;$$

- for $p = 1$ we get

$$\|x\|_1 = \sum_{j=1}^n |x_j| \quad \forall x \in \mathbb{C}^n;$$

- Infinity norm: $\|x\|_\infty = \max_{1\leq j\leq n} |x_j|$.

Theorem 1.2. *All norms on $\mathbb{C}^n$ are* equivalent*: for each pair of norms $\|\cdot\|_{\mathrm{a}}$ and $\|\cdot\|_{\mathrm{b}}$ on $\mathbb{C}^n$ there are constants $0 < c_1 \leq c_2 < \infty$ with*

$$c_1\|x\|_{\mathrm{a}} \leq \|x\|_{\mathrm{b}} \leq c_2\|x\|_{\mathrm{a}} \quad \forall x \in \mathbb{C}^n.$$

Proof. Using property b) from the defintion of a vector norm it suffices to consider vectors $x \in S = \{x \in \mathbb{C}^n \mid \|x\|_2 = 1\}$. Since $\|\cdot\|_{\mathrm{a}}$ is non-zero on all of S we can define $f\colon S \to \mathbb{R}$ by $f(x) = \|x\|_{\mathrm{b}}/\|x\|_{\mathrm{a}}$. Because the function f is continuous and the set S is compact there are $x_1, x_2 \in S$ with $f(x_1) \leq f(x) \leq f(x_2)$ for all $x \in S$. Setting $c_1 = f(x_1) > 0$ and $c_2 = f(x_2)$ completes the proof. $\square$

Remarks.

1. The same result holds for norms on $\mathbb{R}^n$. The proof transfers to this situation without change.

2. We remark that, if $A \in \mathbb{C}^{n\times n}$ is an invertible matrix and $\|\cdot\|$ a norm on $\mathbb{C}^n$ then $\|\cdot\|_A := \|A\cdot\|$ is also a norm.

Definition 1.3. An *inner-product* on $\mathbb{C}^n$ is a mapping $\langle\cdot,\cdot\rangle\colon \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ satisfying:

a) $\langle x, x\rangle \in \mathbb{R}^+$ for all $x \in \mathbb{C}^n$ and $\langle x, x\rangle = 0$ iff $x = 0$;

b) $\langle x, y\rangle = \overline{\langle y, x\rangle}$ for all $x, y \in \mathbb{C}^n$;

c) $\langle x, \alpha y\rangle = \alpha\langle x, y\rangle$ for all $\alpha \in \mathbb{C}$, $x, y \in \mathbb{C}^n$;

d) $\langle x, y + z\rangle = \langle x, y\rangle + \langle x, z\rangle$ for all $x, y, z \in \mathbb{C}^n$;

Remark. Conditions c) and d) above state that $\langle\cdot,\cdot\rangle$ is linear in the second component. Using the rules for inner products we get

$$\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle \quad \text{for all } x, y, z \in \mathbb{C}^n$$

and

$$\langle \alpha x, y\rangle = \overline{\alpha}\langle x, y\rangle \quad \text{for all } \alpha \in \mathbb{C},\ x, y \in \mathbb{C}^n.$$

The inner product is said to be *anti-linear* in the first component.

Example. The *standard inner product* on $\mathbb{C}^n$ is given by

$$\langle x, y\rangle = \sum_{j=1}^{n} \overline{x_j} y_j \qquad \forall x, y \in \mathbb{C}^n. \tag{1.1}$$

Definition 1.4. Two vectors x, y are *orthogonal* with respect to an inner product $\langle\cdot,\cdot\rangle$ iff $\langle x, y\rangle = 0$.

Lemma 1.5 (Cauchy-Schwarz inequality). *Let $\langle\cdot,\cdot\rangle\colon \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ be an inner product. Then*

$$\big|\langle x, y\rangle\big|^2 \leq \langle x, x\rangle \cdot \langle y, y\rangle \tag{1.2}$$

for every $x, y \in \mathbb{C}^n$ and equality holds if and only if x and y are linearly dependent.

Proof. For every $\lambda \in \mathbb{C}$ we have

$$0 \leq \langle x - \lambda y, x - \lambda y\rangle = \langle x, x\rangle - \overline{\lambda}\langle y, x\rangle - \lambda\langle x, y\rangle + \lambda\overline{\lambda}\langle y, y\rangle. \tag{1.3}$$

For $\lambda = \langle y, x\rangle/\langle y, y\rangle$ this becomes

$$0 \leq \langle x, x\rangle - \frac{\langle x, y\rangle\langle y, x\rangle}{\langle y, y\rangle} - \frac{\langle y, x\rangle\langle x, y\rangle}{\langle y, y\rangle} + \frac{\langle y, x\rangle\langle x, y\rangle}{\langle y, y\rangle} = \langle x, x\rangle - \frac{\big|\langle x, y\rangle\big|^2}{\langle y, y\rangle}$$

and multiplying the result by $\langle y, y\rangle$ gives (1.2).

If equality holds in (1.2) then $x - \lambda y$ in (1.3) must be 0 and thus x and y are linearly dependent. If on the other hand x and y are linearly dependent, say $x = \alpha y$, then $\lambda = \langle y, \alpha y\rangle/\langle y, y\rangle = \alpha$ and $x - \lambda y = 0$ giving equality in (1.3) and thus in (1.2). $\square$

Lemma 1.6. *Let $\langle \cdot\,, \cdot \rangle \colon \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ be an inner product. Then $\|\cdot\| \colon \mathbb{C}^n \to \mathbb{R}$ defined by*

$$\|x\| = \sqrt{\langle x, x \rangle} \qquad \forall x \in \mathbb{C}^n$$

is a vector norm.

Proof. a) Since $\langle \cdot\,, \cdot \rangle$ is an inner product we have $\langle x, x \rangle \geq 0$ for all $x \in \mathbb{C}^n$, i.e. $\sqrt{\langle x, x \rangle}$ is real and positive. Also we get

$$\|x\| = 0 \iff \langle x, x \rangle = 0 \iff x = 0.$$

b) We have

$$\|\alpha x\| = \sqrt{\langle \alpha x, \alpha x \rangle} = \sqrt{\overline{\alpha}\alpha \langle x, x \rangle} = |\alpha| \cdot \|x\|.$$

c) Using the Cauchy-Schwarz inequality

$$\big|\langle x, y \rangle\big| \leq \|x\| \|y\| \qquad \forall x, y \in \mathbb{C}^n$$

from Lemma 1.5 we get

$$\begin{aligned} \|x + y\|^2 &= \langle x + y, x + y \rangle \\ &= \langle x, x \rangle + \langle x, y \rangle + \langle y, x \rangle + \langle y, y \rangle \\ &\leq \|x\|^2 + 2\big|\langle x, y \rangle\big| + \|y\|^2 \\ &\leq \|x\|^2 + 2\|x\|\|y\| + \|y\|^2 \\ &= \big(\|x\| + \|y\|\big)^2 \qquad \forall x, y \in \mathbb{C}^n. \end{aligned}$$

This completes the proof. $\square$

Remark. The *angle* between two vectors x and y is the unique value $\varphi \in [0, \pi]$ with

$$\cos(\varphi)\|x\|\|y\| = \langle x, y \rangle.$$

When considering the Euclidean norm and inner product on $\mathbb{R}^n$, this definition of angle coincides with the usual, geometric meaning of angles. In any case, two vectors are orthogonal, if and only if they have angle $\pi/2$.

We write matrices $A \in \mathbb{C}^{m \times n}$ as

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix};$$

we write $(A)_{ij} = a_{ij}$ for the ij^{th} entry of A.

Definition 1.7. Given $A \in \mathbb{C}^{m \times n}$ we define the *adjoint* $A^* \in \mathbb{C}^{n \times m}$ by $\big(A^*\big)_{ij} = \overline{a_{ji}}$. (For $A \in \mathbb{R}^{m \times n}$ we write A^T instead of A^*.)

By identifying the space $\mathbb{C}^n$ of vectors with the space $\mathbb{C}^{n \times 1}$ of $n \times 1$-matrices, we can take the adjoint of a vector. Then we can write the standard inner product as

$$\langle x, y \rangle = x^* y.$$

Thus, the standard inner product satisfies

$$\langle Ax, y \rangle = (Ax)^* y = x^* A^* y = \langle x, A^* y \rangle$$

for all $x \in \mathbb{C}^n$, $y \in \mathbb{C}^m$ and all $A \in \mathbb{C}^{m \times n}$. Unless otherwise specified, we will use $\langle \cdot\,, \cdot \rangle$ to denote the standard inner product (1.1) and $\|\cdot\|_2$ to denote the corresponding Euclidean norm.

The following families of special matrices will be central in what follows:

Definition 1.8.
1. $Q \in \mathbb{C}^{m\times n}$ is *unitary* if $Q^*Q = I$. (If Q is real then $Q^TQ = I$ and we say Q is *orthogonal*.)
2. $A \in \mathbb{C}^{n\times n}$ is *Hermitian* if $A^* = A$. (If A is real, we say A is *symmetric*.)
3. A Hermitian matrix $A \in \mathbb{C}^{n\times n}$ is *positive-definite* (resp. *positive semi-definite*) if $x^*Ax = \langle Ax, x\rangle > 0$ (resp. ≥ 0) for all $x \in \mathbb{C}^n \setminus \{0\}$. In this text, whenever we use the terminology *positive-definite* or *positive semi-definite* we are necessarily refering to Hermitian matrices.

Remarks. Unitary matrices have the following properties:

- A matrix Q is unitary, if and only if the columns of Q are orthonormal with respect to the standard inner-product. In particular unitary matrices cannot have more columns than rows.
- If Q is a square matrix, $Q^{-1} = Q^*$ and thus $QQ^* = I$.
- A square matrix Q is unitary, if and only if Q^* is unitary.

The standard inner product and norm are invariant under multiplication by a unitary matrix:

Theorem 1.9. *Let $\langle\cdot,\cdot\rangle$ denote the standard inner product. Then for any unitary $Q \in \mathbb{C}^{m\times n}$ and any $x, y \in \mathbb{C}^n$ we have $\langle Qx, Qy\rangle = \langle x, y\rangle$ and $\|Qx\|_2 = \|x\|_2$.*

Proof. The first claim follows from $\langle Qx, Qy\rangle = \langle x, Q^*Qy\rangle = \langle x, y\rangle$ and using the relation $\|x\|^2 = \langle x, x\rangle$ gives the second claim. □

Other inner products with appropriate properties can give rise to other norms; for example, for matrices A which are Hermitian and positive-definite,

$$\langle x, y\rangle_A = \langle x, Ay\rangle \tag{1.4}$$

is an inner product and

$$\|x\|_A = \sqrt{\langle x, x\rangle_A}. \tag{1.5}$$

defines a norm (see Exercise 1-2).

1.2 Eigenvalues and Eigenvectors

Definition 1.10. Given a matrix $A \in \mathbb{C}^{n\times n}$, a vector $x \in \mathbb{C}^n$ is an *eigenvector* and $\lambda \in \mathbb{C}$ is an *eigenvalue* (also called a *right eigenvalue*) of A if

$$Ax = \lambda x \quad \text{and} \quad x \neq 0. \tag{1.6}$$

When x is an eigenvector of A, then for every $\alpha \neq 0$ the vector αx is an eigenvector for the same eigenvalue, since both sides of (1.6) are linear in x. Sometimes it is convenient to normalise x by choosing $\|x\|_2 = 1$. Then the eigenvalue problem is to find $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$ satisfying

$$Ax = \lambda x \quad \text{and} \quad \|x\|_2 = 1.$$

Definition 1.11. Given a matrix $A \in \mathbb{C}^{n\times n}$ we define the *characteristic polynomial* of A as

$$\rho_A(z) := \det(A - zI).$$

Theorem 1.12. *A value $\lambda \in \mathbb{C}$ is an eigenvalue of the matrix A, if and only if $\rho_A(\lambda) = 0$.*

Proof. λ is an eigenvalue of A, if and only if there is an $x \neq 0$ with $(A - \lambda I)x = 0$. This is equivalent to the condition that $A - \lambda I$ is singular which in turn is equivalent to $\det(A - \lambda I) = 0$. □

Since ρ_A is a polynomial of degree n, there will be n (some possibly repeated) eigenvalues, denoted by $\lambda_1, \dots, \lambda_n$ and determined by $\rho_A(\lambda_k) = 0$.

Definition 1.13. An eigenvalue λ has *algebraic multiplicity* q if q is the largest integer such that $(z-\lambda)^q$ is a factor of the characteristic polynomial $\rho_A(z)$. The *geometric multiplicity*, r, is the dimension of the null space of $A - \lambda I$. An eigenvalue is *simple* if $q = r = 1$.

If λ is an eigenvalue of $A \in \mathbb{C}^{n\times n}$ then $\det(A - \lambda I) = 0$ which implies that $\det(A^* - \bar{\lambda} I) = 0$ and so $(A^* - \bar{\lambda} I)$ has non-trivial null space. Thus there is a vector y (the eigenvector of A^* corresponding to the eigenvalue $\bar{\lambda}$) with $y^* A = \lambda y^*$ and $y \neq 0$.

Definition 1.14. A vector $y \in \mathbb{C}^n$ with

$$y^* A = \lambda y^* \quad \text{and} \quad y \neq 0$$

is known as a *left eigenvector* of $A \in \mathbb{C}^{n\times n}$ corresponding to the eigenvalue λ.

Note that, even though the corresponding eigenvalues are the same, the right and left eigenvectors of a matrix are usually different.

Definition 1.15. Matrices $A, B \in \mathbb{C}^{n\times n}$ are *similar*, if $B = S^{-1}AS$ with $S \in \mathbb{C}^{n\times n}$ invertible. The matrix S is a *similarity transform.*

Remarks. If a matrix $A \in \mathbb{C}^{n\times n}$ has n linearly independent eigenvectors x_i and we arrange them as columns of the matrix X, then X is invertible. If we let Λ denote a diagonal matrix with eigenvalues of A on the diagonal, then we may write

$$AX = X\Lambda.$$

By invertibility of X we have

$$A = X\Lambda X^{-1}. \tag{1.7}$$

Thus Λ is a similarity transform of A. It reveals the eigenstructure of A and is hence very useful in many situations. However, in general a matrix does not have n linearly independent eigenvalues and hence generalizations of this factorization are important. Two which will arise in the next chapter are:

- Jordan Canonical Form: $A = SJS^{-1}$ (see Theorem 2.7)
- Schur Factorization: $A = QTQ^*$ (see Theorem 2.2)

These are both similarity transformations which reveal the eigenvalues of A on the diagonals of J and T, respectively. The Jordan Canonical Form is not stable to perturbations, but the Schur Factorization is. Hence Schur Factorization will form the basis of good algorithms while the Jordan Canonical Form is useful for more theoretical purposes, such as defining the matrix exponential e^{At}.

Theorem 1.16 (Similarity and Eigenvalues)**.** *If B is similar to A, then B and A have the same eigenvalues with the same algebraic and geometric multiplicities.*

Proof. Exercise 2-4. □

Lemma 1.17. *For a simple eigenvalue μ,*

$$\dim\big(\ker(A - \mu I)^2\big) = 1.$$

Proof. (Sketch) We prove this in the case where A has n linearly independent eigenvalues x_i all of which correspond to simple eigenvalues λ_i. Then A may be factorized as in (1.7) with $\Lambda = \operatorname{diag}\{\lambda_1, \lambda_2, \dots, \lambda_n\}$. Hence

$$(A - \mu I)^2 = X\Omega X^{-1},$$

where $\Omega = \operatorname{diag}\{(\lambda_1-\mu)^2, (\lambda_2-\mu)^2, \dots, (\lambda_n-\mu)^2\}$.

Without loss of generality let $\mu = \lambda_1$, noting that $\lambda_j \neq \mu$ by simplicity. $\ker(\Omega)$ is one dimensional, spanned by e_1. Hence $\ker(A-\mu I)^2$ is one dimensional by Theorem 1.16

The general case can be established by use of the Jordan form (see Theorem 2.7), using the fact that the Jordan block corresponding to a simple eigenvalue is diagonal. ☐

Theorem 1.18 (Eigenvalue Multiplicities). *For any eigenvalue of $A \in \mathbb{R}^{n\times n}$ the algebraic and geometric multiplicities q and r respectively satisfy*

$$1 \le r \le q.$$

Proof. Let μ be the eigenvalue. Since $(A-\mu I)$ is non-invertible, its null-space $\mathcal{U}$ has dimension $r \ge 1$. Let $\hat{V} \in \mathbb{C}^{n\times r}$ have r columns comprising an orthonormal basis for $\mathcal{U}$; then extend $\hat{V}$ to $V \in \mathbb{C}^{n\times n}$ by adding orthonormal columns so that V is unitary. Now

$$B = V^*AV = \begin{pmatrix} \mu I & C \\ 0 & D \end{pmatrix},$$

where I is the $r \times r$ identity, C is $r\times(n-r)$ and D is $(n-r)\times(n-r)$, and B and A are similar. Then

$$\begin{aligned} \det(B-zI) &= \det(\mu I - zI)\det(D-zI) \\ &= (\mu - z)^r \det(D-zI). \end{aligned}$$

Thus B has algebraic multiplicity $\ge r$ for B and hence A has algebraic multiplicity $\ge r$, by Theorem 1.16. ☐

Definition 1.19. The *spectral radius* of a matrix $A \in \mathbb{C}^{n\times n}$ is defined by

$$\rho(A) = \max\bigl\{|\lambda| \bigm| \lambda \text{ is eigenvalue of } A\bigr\}.$$

By considering the eigenvectors of a matrix, we can define an important class of matrices

Definition 1.20. A matrix $A \in \mathbb{C}^{n\times n}$ is *normal* iff it has n orthogonal eigenvectors.

The importance of this concept lies in the fact, that normal matrices can always be diagonalised: if $Q \in \mathbb{C}^{n\times n}$ is a matrix where the columns form an orthonormal system of eigenvectors and $\Lambda \in \mathbb{C}^{n\times n}$ is the diagonal matrix with the corresponding eigenvalues on the diagonal, then we have

$$A = Q\Lambda Q^*.$$

Using this relation, we see that every normal matrix satisfies $A^*A = Q\Lambda^*\Lambda Q^* = Q\Lambda\Lambda^*Q^* = AA^*$. In Theorem 2.3 we will see that the condition $A^*A = AA^*$ is actually equivalent to A being normal in the sense of the definition above. Sometimes this alternative condition is used to define when a matrix is normal. As a consequence of this equivalence, every Hermitian matrix is also normal.

Let now A be Hermitian and positive definite. Then $Ax = \lambda x$ implies

$$\lambda\langle x, x\rangle = \langle x, \lambda x\rangle = \langle x, Ax\rangle = \langle Ax, x\rangle = \langle \lambda x, x\rangle = \bar{\lambda}\langle x, x\rangle.$$

Thus, all eigenvalues of A are real and we can arrange them in increasing order $\lambda_{\min} = \lambda_1 \le \dots \le \lambda_n = \lambda_{\max}$. The following lemma uses $\lambda_{\min}$ and $\lambda_{\max}$ to estimate the values of the norm $\|\cdot\|_A$ from (1.5) by $\|\cdot\|$.

Lemma 1.21. *Let $\lambda_{\min}$ and $\lambda_{\max}$ be the smallest and largest eigenvalues of a Hermitian, positive definite matrix $A \in \mathbb{C}^{n\times n}$. Then*

$$\lambda_{\min}\|x\|^2 \le \|x\|_A^2 \le \lambda_{\max}\|x\|^2 \quad \forall x \in \mathbb{C}^n.$$

Proof. Let $\varphi_1, \ldots, \varphi_n$ be an orthonormal system of eigenvectors of A with corresponding eigenvalues $\lambda_{\min} = \lambda_1 \leq \cdots \leq \lambda_n = \lambda_{\max}$. By writing x as $x = \sum_{i=1}^n \xi_i \varphi_i$ we get

$$\|x\|^2 = \sum_{i=1}^n |\xi_i|^2$$

and

$$\|x\|_A^2 = \sum_{i=1}^n \lambda_i \xi_i^2.$$

This gives the upper bound

$$\|x\|_A^2 \leq \sum_{i=1}^n \lambda_{\max} \xi_i^2 \leq \lambda_{\max} \|x\|^2$$

and similarly we get the lower bound. □

1.3 Dual Spaces

Let $\langle \cdot, \cdot \rangle$ denote the standard inner-product.

Definition 1.22. Given a norm $\|\cdot\|$ on $\mathbb{C}^n$, the pair $(\mathbb{C}^n, \|\cdot\|)$ is a Banach space B. The Banach space B', the *dual* of B, is the pair $(\mathbb{C}^n, \|\cdot\|_{B'})$, where

$$\|x\|_{B'} = \max_{\|y\|=1} |\langle x, y \rangle|.$$

See Exercise 1-5 to deduce that the preceeding definition satisfies the norm axioms.

Theorem 1.23. *The spaces* $(\mathbb{C}^n, \|\cdot\|_1)$ *and* $(\mathbb{C}^n, \|\cdot\|_\infty)$ *are the duals of one another.*

Proof. Firstly, we must show

$$\|x\|_1 = \max_{\|y\|_\infty = 1} |\langle x, y \rangle|.$$

This is clearly true for $x = 0$ and so we consider only $x \neq 0$. Now

$$|\langle x, y \rangle| \leq \max_i |y_i| \sum_{j=1}^n |x_j| = \|y\|_\infty \|x\|_1,$$

and therefore

$$\max_{\|y\|_\infty = 1} |\langle x, y \rangle| \leq \|x\|_1.$$

We need to show that this upper-bound is achieved. If $y_j = x_j/|x_j|$ (with the convention that this is 0 when x_j is 0) then $\|y\|_\infty = 1$ (since $x \neq 0$) and

$$\langle x, y \rangle = \sum_{j=1}^n |x_j|^2/|x_j| = \sum_{j=1}^n |x_j| = \|x\|_1.$$

Hence $\max_{\|y\|_\infty = 1} |\langle x, y \rangle| = \|x\|_1$.

Secondly, it remains to show that

$$\|x\|_\infty = \max_{\|y\|_1 = 1} |\langle x, y \rangle|.$$

We have

$$\begin{aligned} & |\langle x, y \rangle| \leq \|y\|_1 \|x\|_\infty \\ \Rightarrow \quad & \max_{\|y\|_1 = 1} |\langle x, y \rangle| \leq \|x\|_\infty. \end{aligned}$$

If $x = 0$ we have equality; if not then, for some k such that $|x_k| = \|x\|_\infty > 0$, choose $y_j = \delta_{jk} x_k / |x_k|$. Then

$$\langle x, y \rangle = |x_k| = \|x\|_\infty \quad \text{and} \quad \|y\|_1 = 1.$$

Thus $\max_{\|y\|_1 = 1} |\langle x, y \rangle| = \|x\|_\infty$. □

Theorem 1.24. *If $p, q \in (1, \infty)$ with $p^{-1} + q^{-1} = 1$ then the Banach spaces $(\mathbb{C}^n, \|\cdot\|_p)$ and $(\mathbb{C}^n, \|\cdot\|_q)$ are the duals of one another.*

Proof. See Exercise 1-6. $\square$

1.4 Matrix Norms

Since we can consider the space $\mathbb{C}^{m\times n}$ of all $m\times n$-matrices to be a copy of the $m\times n$-dimensional vector space $\mathbb{C}^{mn}$, we can use all vector norms on $\mathbb{C}^{mn}$ as vector norms on the matrices $\mathbb{C}^{m\times n}$. Examples of vector norms on the space of matrices include

- maximum norm: $\|A\|_{\max} = \max_{i,j} |a_{ij}|$
- Frobenius norm: $\|A\|_{\mathrm{F}} = \left(\sum_{i,j=1}^{m,n} |a_{ij}|^2\right)^{\frac{1}{2}}$
- operator norm $\mathbb{C}^n \to \mathbb{C}^m$: if $A \in \mathbb{C}^{m\times n}$,

  $$\|A\|_{(\hat{m},\hat{n})} = \max_{\|x\|_{\hat{n}}=1} \|Ax\|_{\hat{m}}$$

  where $\|\cdot\|_{\hat{m}}$ is a norm on $\mathbb{C}^m$, and $\|\cdot\|_{\hat{n}}$ is a norm on $\mathbb{C}^n$. Note that, for any operator norm,

  $$\|A\|_{(\hat{m},\hat{n})} = \max_{\|x\|_{\hat{n}}\le 1} \|Ax\|_{\hat{m}} = \max_{\|x\|_{\hat{n}}=1} \|Ax\|_{\hat{m}} = \max_{x\in\mathbb{C}^n\setminus\{0\}} \frac{\|Ax\|_{\hat{m}}}{\|x\|_{\hat{n}}}.$$

Sometimes it is helpful to consider special vector norms on a space of matrices, which are compatible with the matrix-matrix multiplication.

Definition 1.25. A *matrix norm* on $\mathbb{C}^{n\times n}$ is a mapping $\|\cdot\| \colon \mathbb{C}^{n\times n} \to \mathbb{R}$ with

a) $\|A\| \ge 0$ for all $A \in \mathbb{C}^{n\times n}$ and $\|A\| = 0$ iff $A = 0$,

b) $\|\alpha A\| = |\alpha|\|A\|$ for all $\alpha \in \mathbb{C}, A \in \mathbb{C}^{n\times n}$,

c) $\|A + B\| \le \|A\| + \|B\|$ for all $A, B \in \mathbb{C}^{n\times n}$.

d) $\|AB\| \le \|A\|\|B\|$ for all $A, B \in \mathbb{C}^{n\times n}$.

Remark. Conditions a), b) and c) state that $\|\cdot\|$ is a vector norm on the vector space $\mathbb{C}^{n\times n}$. Condition d) only makes sense for matrices, since general vectors spaces are not equipped with a product.

Examples of matrix norms include

- p-operator norm $\mathbb{C}^n \to \mathbb{C}^n$: if $A \in \mathbb{C}^{n\times n}$,

  $$\|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p, \qquad 1 \le p \le \infty$$

  The vector operator norm from $\mathbb{C}^n$ into $\mathbb{C}^m$ reduces to the p-operator norm if $n = m$ and the p-norm is chosen in the range and image spaces.

Definition 1.26. Given a vector norm $\|\cdot\|_{\mathrm{v}}$ on $\mathbb{C}^n$ we define the *induced norm* $\|\cdot\|_{\mathrm{m}}$ on $\mathbb{C}^{n\times n}$ by

$$\|A\|_{\mathrm{m}} = \max_{x\ne 0} \frac{\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}}$$

for all $A \in \mathbb{C}^{n\times n}$.

We now show that the induced norm is indeed a norm.

Theorem 1.27. *The induced norm $\|\cdot\|_{\mathrm{m}}$ of a vector norm $\|\cdot\|_{\mathrm{v}}$ is a matrix norm with*

$$\|I\|_{\mathrm{m}} = 1$$

and

$$\|Ax\|_{\mathrm{v}} \le \|A\|_{\mathrm{m}}\|x\|_{\mathrm{v}}$$

for all $A \in \mathbb{C}^{n\times n}$ and $x \in \mathbb{C}^n$.

Proof. a) $\|A\|_{\mathrm{m}} \in \mathbb{R}$ and $\|A\|_{\mathrm{m}} \ge 0$ for all $A \in \mathbb{C}^{n\times n}$ is obvious from the definition. Also from the definition we get

$$\begin{aligned}\|A\|_{\mathrm{m}} = 0 \iff & \frac{\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = 0\ \forall x \neq 0 \\ \iff & \|Ax\|_{\mathrm{v}} = 0\ \forall x \neq 0 \iff Ax = 0\ \forall x \neq 0 \iff A = 0.\end{aligned}$$

b) For $\alpha \in \mathbb{C}$ and $A \in \mathbb{C}^{n\times n}$ we get

$$\|\alpha A\|_{\mathrm{m}} = \max_{x\neq 0} \frac{\|\alpha Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = \max_{x\neq 0} \frac{|\alpha|\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = |\alpha|\|A\|_{\mathrm{m}}$$

c) For $A, B \in \mathbb{C}^{n\times n}$ we get

$$\begin{aligned}\|A+B\|_{\mathrm{m}} &= \max_{x\neq 0} \frac{\|Ax+Bx\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} \\ &\le \max_{x\neq 0} \frac{\|Ax\|_{\mathrm{v}} + \|Bx\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} \\ &\le \max_{x\neq 0} \frac{\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} + \max_{x\neq 0} \frac{\|Bx\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = \|A\|_{\mathrm{m}} + \|B\|_{\mathrm{m}}.\end{aligned}$$

Before we check condition d) from the definition of a matrix norm we verify

$$\|I\|_m = \max_{x\neq 0} \frac{\|Ix\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = \max_{x\neq 0} \frac{\|x\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = 1$$

and

$$\|A\|_{\mathrm{m}} = \max_{y\neq 0} \frac{\|Ay\|_{\mathrm{v}}}{\|y\|_{\mathrm{v}}} \ge \frac{\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} \qquad \forall x \in \mathbb{C}^n \setminus \{0\}$$

which gives

$$\|Ax\|_{\mathrm{v}} \le \|A\|_{\mathrm{m}}\|x\|_{\mathrm{v}} \qquad \forall x \in \mathbb{C}^n.$$

d) Using this estimate we find

$$\begin{aligned}\|AB\|_{\mathrm{m}} &= \max_{x\neq 0} \frac{\|ABx\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} \\ &\le \max_{x\neq 0} \frac{\|A\|_{\mathrm{m}}\|Bx\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}} = \|A\|_{\mathrm{m}}\|B\|_{\mathrm{m}}.\end{aligned}$$

□

Remarks.

1. Usually one denotes the induced matrix norm with the same symbol as the corresponding vector norm. For the remainder of this text we will follow this convention.

2. As a consequence of theorem 1.27 we can see that not every matrix norm is an induced norm: If $\|\cdot\|_{\mathrm{m}}$ is a matrix norm, then it is easy to check that $\|\cdot\|'_{\mathrm{m}} = 2\|\cdot\|_{\mathrm{m}}$ is a matrix norm, too. But at most one of these two norms can equal 1 for the identity matrix, and thus the other one cannot be an induced matrix norm.

3. Recall that $\|\cdot\|_A := \|A\cdot\|$ is a vector norm on $\mathbb{C}^n$ whenever $\|\cdot\|$ is, provided that A is invertible. The inequality from Theorem 1.27 gives the following upper and lower bounds for the norm $\|\cdot\|$ in terms of the original norm:

$$\frac{1}{\|A^{-1}\|}\|x\| \leq \|x\|_A \leq \|A\|\|x\|.$$

Theorem 1.28. *The matrix norm induced by the infinity norm is the* maximum row sum*:*

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|.$$

Proof. For $x \in \mathbb{C}^n$ we get

$$\|Ax\|_\infty = \max_{1\leq i\leq n} |(Ax)_i| = \max_{1\leq i\leq n} |\sum_{j=1}^n a_{ij}x_j| \leq \max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|\|x\|_\infty$$

which gives

$$\frac{\|Ax\|_\infty}{\|x\|_\infty} \leq \max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|$$

for all $x \in \mathbb{C}^n$ and thus $\|A\|_\infty \leq \max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|$.

For the lower bound choose $k \in \{1, 2, \ldots, n\}$ such that

$$\max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}| = \sum_{j=1}^n |a_{kj}|$$

and define $x \in \mathbb{C}^n$ by $x_j = \overline{a_{kj}}/|a_{kj}|$ for all $j = 1, \ldots, n$ (with the convention that this is 0 when a_{kj} is 0). Then we have $\|x\|_\infty = 1$ and

$$\begin{aligned}
\|A\|_\infty &\geq \frac{\|Ax\|_\infty}{\|x\|_\infty} \\
&= \max_{1\leq i\leq n} |\sum_{j=1}^n a_{ij}\frac{\overline{a_{kj}}}{|a_{kj}|}| \\
&\geq |\sum_{j=1}^n a_{kj}\frac{\overline{a_{kj}}}{|a_{kj}|}| \\
&= \sum_{j=1}^n |a_{kj}| \\
&= \max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|.
\end{aligned}$$

This is the required result. $\square$

Theorem 1.29. *Let $A \in \mathbb{C}^{n\times n}$. Then $\|A^*\|_1 = \|A\|_\infty$ and so*

$$\|A\|_1 = \|A^*\|_\infty = \max_{1\leq j\leq n} \sum_{i=1}^n |a_{ij}|.$$

This expression is known as the maximum column sum*.*

Proof. (Sketch)

$$\begin{aligned}
\|A\|_\infty &= \max_{\|x\|_\infty=1} \|Ax\|_\infty \\
&= \max_{\|x\|_\infty=1} \left(\max_{\|y\|_1=1} |\langle Ax, y\rangle| \right) && \text{(Dual definition)} \\
&= \max_{\|y\|_1=1} \left(\max_{\|x\|_\infty=1} |\langle Ax, y\rangle| \right) && \text{(needs careful justification)} \\
&= \max_{\|y\|_1=1} \left(\max_{\|x\|_\infty=1} |\langle x, A^* y\rangle| \right) && \text{(definition of } A^*\text{)} \\
&= \max_{\|y\|_1=1} \|A^* y\|_1 && \text{(Dual definition)} \\
&= \|A^*\|_1.
\end{aligned}$$

Since $(A^*)_{ij} = \overline{a_{ji}}$, the result follows. □

Remark. This result is readily extended to non-square matrices $A \in \mathbb{C}^{m\times n}$.

Recall that the spectral radius of a matrix is defined as

$$\rho(A) = \max\big\{|\lambda| \;\big|\; \lambda \text{ is eigenvalue of } A\big\}.$$

Theorem 1.30. *For any matrix norm* $\|\cdot\|$*, any matrix* $A \in \mathbb{C}^{n\times n}$ *and any* $k \in \mathbb{N}$ *we have*

$$\rho(A)^k \le \rho(A^k) \le \|A^k\| \le \|A\|^k.$$

Proof. Let $B = A^k$. The first inequality is a consequence of the fact that, whenever x is an eigenvector of A with eigenvalue λ, the vector x is also an eigenvector of B, but with eigenvalue λ^k. By definition of the spectral radius $\rho(B)$ we can find an eigenvector x with $Bx = \lambda x$ and $\rho(B) = |\lambda|$. Let $X \in \mathbb{C}^{n\times n}$ be the matrix where all n columns are equal to x. Then we have $BX = \lambda X$ and thus

$$\|B\|\|X\| \ge \|BX\| = \|\lambda X\| = |\lambda|\|X\| = \rho(B)\|X\|.$$

Dividing by $\|X\|$ gives $\rho(B) \le \|B\|$. The final inequality follows from property d) in the definition of a matrix norm. □

Theorem 1.31. *If* $A \in \mathbb{C}^{n\times n}$ *is normal, then*

$$\rho(A)^\ell = \|A^\ell\|_2 = \|A\|_2^\ell \qquad \forall \ell \in \mathbb{N}.$$

Proof. Let $x_1, \dots, x_n$ be an orthonormal basis composed of eigenvectors of A with corresponding eigenvalues $\lambda_1, \dots, \lambda_n$. Without loss of generality we have $\rho(A) = |\lambda_1|$.

Let $x \in \mathbb{C}^n$. Then we can write

$$x = \sum_{j=1}^n \alpha_j x_j$$

and get

$$\|x\|_2^2 = \sum_{j=1}^n |\alpha_j|^2.$$

Similarly we find

$$Ax = \sum_{j=1}^n \alpha_j \lambda_j x_j \quad \text{and} \quad \|Ax\|_2^2 = \sum_{j=1}^n |\alpha_j \lambda_j|^2.$$

This shows

$$\begin{aligned}\frac{\|Ax\|_2}{\|x\|_2} &= \frac{\left(\sum_{j=1}^n |\alpha_j\lambda_j|^2\right)^{1/2}}{\left(\sum_{j=1}^n |\alpha_j|^2\right)^{1/2}} \\ &\le \left(\frac{\sum_{j=1}^n |\alpha_j|^2|\lambda_1|^2}{\sum_{j=1}^n |\alpha_j|^2}\right)^{1/2} \\ &= |\lambda_1| = \rho(A) \qquad \forall x \in \mathbb{C}^n\end{aligned}$$

and consequently $\|A\|_2 \le \rho(A)$.

Using Theorem 1.30 we get

$$\rho(A)^\ell \le \|A^\ell\|_2 \le \|A\|_2^\ell \le \rho(A)^\ell$$

for all $\ell \in \mathbb{N}$. This completes the proof. □

Similar methods to those used in the proof of the previous result yield the following theorem.

Theorem 1.32. *For all matrices* $A \in \mathbb{C}^{m\times n}$

$$\|A\|_2^2 = \rho(A^*A).$$

Proof. See Exercise 1-9. □

The matrix 2-norm has the special property that it is invariant under multplication by a unitary matrix. This is the analog of Theorem 1.9 for vector norms.

Theorem 1.33. *For all matrices* $A \in \mathbb{C}^{m\times n}$ *and unitary matrices* $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$

$$\|UA\|_2 = \|A\|_2, \quad \|AV\|_2 = \|A\|_2.$$

Proof. The first result follows from the previous theorem, after noting that $(UA)^*(UA) = A^*U^*UA = A^*A$. Because $(AV)^*(AV) = V^*(A^*A)V$ and because $V^*(A^*A)V$ is a similarity transformation of A^*A the second result also follows from the previous theorem. □

Let $A, B \in \mathbb{C}^{m\times n}$. In the following it will be useful to employ the notation $|A|$ to denote the matrix with entries $(|A|)_{ij} = |a_{ij}|$ and the notation $|A| \le |B|$ as shorthand for $|a_{ij}| \le |b_{ij}|$ for all i, j.

Lemma 1.34. *If two matrices* $A, B \in \mathbb{C}^{m\times n}$ *satisfy* $|A| \le |B|$ *then* $\|A\|_\infty \le \|B\|_\infty$ *and* $\|A\|_1 \le \|B\|_1$. *Furthermore* $\||AB|\|_\infty \le \||A||B|\|_\infty$.

Proof. For the first two observations, it suffices to prove the first result since $\|A\|_1 = \|A^*\|_\infty$ and $|A| \le |B|$ implies that $|A^*| \le |B^*|$. The first result is a direct consequence of the representation of the ∞-norm and 1-norm from theorems 1.28 and 1.29. To prove the last result note that

$$\begin{aligned}(|AB|)_{ij} &= |\sum_k A_{ik}B_{kj}| \\ &\le \sum_k |A_{ik}||B_{kj}| \\ &= (|A||B|)_{ij}.\end{aligned}$$

The first result completes the proof. □

Lemma 1.35. *Let* $A, B \in \mathbb{C}^{n\times n}$. *Then*

$$\begin{aligned}\|A\|_{\max} &\le \|A\|_\infty \le n\|A\|_{\max}, \\ \|AB\|_{\max} &\le \|A\|_\infty \|B\|_{\max}, \\ \|AB\|_{\max} &\le \|A\|_{\max}\|B\|_1.\end{aligned}$$

Proof. Exercise 1-8. □

Definition 1.36. The *outer product* of two vectors $a, b \in \mathbb{C}^n$ is the matrix $a \otimes b \in \mathbb{C}^{n\times n}$ defined by

$$(a \otimes b)c = (b^* c)a = \langle b, c\rangle a \qquad \forall c \in \mathbb{C}^n.$$

We sometimes write $a \otimes b = ab^*$. The ij^{th} entry of the outer product is $(a \otimes b)_{ij} = a_i \bar{b}_j$.

Definition 1.37. Let S be a subspace of $\mathbb{C}^n$. Then the *orthogonal complement* of S is defined by

$$S^\perp = \{x \in \mathbb{C}^n \mid \langle x, y\rangle = 0\ \forall y \in S\}.$$

The *orthogonal projection* onto S, P, can be defined as follows: let $\{y_i\}_{i=1}^k$ be an orthonormal basis for S, then

$$Px = \sum_{j=1}^k \langle y_j, x\rangle y_j = \Big(\sum_{j=1}^k y_j y_j^*\Big)x = \Big(\sum_{j=1}^k y_j \otimes y_j\Big)x.$$

Theorem 1.38. *P is a projection, that is $P^2 = P$. Furthermore, if $P^\perp = I - P$, then $P^\perp$ is the orthogonal projection onto $S^\perp$.*

Proof. Extend $\{y_i\}_{i=1}^k$ to a basis for $\mathbb{C}^n$, denoted $\{y_i\}_{i=1}^n$, noting that $S^\perp = \operatorname{span}\{y_{k+1}, \dots, y_n\}$. Any $x \in \mathbb{C}^n$ can be written uniquely as

$$x = \sum_{j=1}^n \langle y_j, x\rangle y_j,$$

and so

$$Px = \sum_{j=1}^k \langle y_j, x\rangle y_j,$$

found by truncating to k terms. Clearly truncating again leaves the expression unchanged:

$$P^2 x = \sum_{j=1}^k \langle y_j, x\rangle y_j = Px, \quad \forall x \in \mathbb{C}^n.$$

Now $(I - P)x = P^\perp x = \sum_{j=k+1}^n \langle y_j, x\rangle y_j$, proving the second result. □

1.5 Structured Matrices

Definition 1.39. A matrix $A \in \mathbb{C}^{n\times n}$ is

$$\begin{array}{rrlll}
 & \textit{diagonal} & \text{if } a_{ij} = 0 & i \neq j & \\
\text{(strictly)} & \textit{upper-triangular} & \text{if } a_{ij} = 0 & i > j & (\geq) \\
\text{(strictly)} & \textit{lower-triangular} & \text{if } a_{ij} = 0 & i < j & (\leq) \\
 & \textit{upper Hessenberg} & \text{if } a_{ij} = 0 & i > j+1 & \\
 & \textit{upper bidiagonal} & \text{if } a_{ij} = 0 & i > j \,\&\, i < j-1 & \\
 & \textit{tridiagonal} & \text{if } a_{ij} = 0 & i > j+1 \,\&\, i < j-1 &
\end{array}$$

Definition 1.40. A matrix $P \in \mathbb{R}^{n\times n}$ is called a *permutation matrix* if every row and every column contains $n-1$ zeros and 1 one.

Remarks.

1. If P is a permutation matrix, then we have

$$(P^TP)_{ij} = \sum_{k=1}^{n} p_{ki}p_{kj} = \delta_{ij}$$

and thus $P^TP = I$. This shows that permutation matrices are orthogonal.

2. If $\pi\colon \{1,\dots,n\} \to \{1,\dots,n\}$ is a permutation, then the matrix $P = (p_{ij})$ with

$$p_{ij} = \begin{cases} 1 & \text{if } j = \pi(i) \text{ and} \\ 0 & \text{else} \end{cases}$$

is a permutation matrix. Indeed every permutation matrix is of this form. In particular the identity matrix is a permutation matrix.

3. If P is the permutation matrix corresponding to the permutation π, then $(P^{-1})_{ij} = 1$ if and only if $j = \pi^{-1}(i)$. Thus the permutation matrix P^{-1} corresponds to the permutation π^{-1}.

4. We get

$$(PA)_{ij} = \sum_{k=1}^{n} p_{ik}a_{kj} = a_{\pi(i),j}$$

for all $i, j \in \{1,\dots,n\}$. This shows that multiplying a permutation matrix from the left reorders the rows of A. Furthermore we have

$$(AP)_{ij} = \sum_{k=1}^{n} a_{ik}p_{kj} = a_{i,\pi^{-1}(j)}$$

and hence multiplying a permutation matrix from the right reorders the columns of A.

5. If P is a permutation matrix, then P^T is also a permutation matrix.

Bibliography

Excellent treatments of matrix analysis may be found in [Bha97] and [HJ85]. More advanced treatment of the subject includes [Lax97]. Theorem 1.16 and Lemma 1.17 are proved in [Ner71]. The proof of Theorem 1.24 may be found in [Rob01]. Theorem 1.32 is proved in the solutions for instructors.

Exercises

Exercise 1-1. Show that the following relations hold for all $x \in \mathbb{C}^n$:
a) $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$,
b) $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$ and
c) $\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$.

Exercise 1-2. Prove that, for Hermitian positive definite A, equations (1.4) and (1.5) define an inner-product and norm, respectively.

Exercise 1-3. For matrices in $\mathbb{R}^{m\times n}$ prove that

$$\|A\|_{\max} \leq \|A\|_F \leq \sqrt{mn}\|A\|_{\max}.$$

Exercise 1-4. Define an inner product $\langle \cdot , \cdot \rangle$ on matrices in $\mathbb{R}^{n\times n}$ such that

$$\|A\|_F^2 = \langle A, A\rangle.$$

Exercise 1-5. Prove that the norm $\|\cdot\|_{B'}$ appearing in Definition 1.22 is indeed a norm.

Exercise 1-6. Prove Theorem 1.24.

Exercise 1-7. Show that $\|A\|_{\max} = \max_{i,j} |a_{ij}|$ for all $A \in \mathbb{C}^{n\times n}$ defines a vector norm on the space of $n \times n$-matrices, but not a matrix norm.

Exercise 1-8. Prove Lemma 1.35.

Exercise 1-9. Show that $\|A\|_2^2 = \rho(A^T A)$ for every matrix $A \in \mathbb{R}^{n\times n}$ (this is the real version of Theorem 1.32).

Exercise 1-10. For $A \in \mathbb{R}^{n\times n}$ recall the definition of $|A|$, namely $(|A|)_{ij} = |a_{ij}|$. Show that

$$\||A|\| = \|A\|$$

holds in the Frobenius, infinity and 1-norms. Is the result true in the Euclidean norm? Justify your assertion.

Exercise 1-11. Let $\|\cdot\|$ be an operator norm. Prove that if $\|X\| < 1$, then

- $I - X$ is invertible,
- the series $\sum_{i=0}^{\infty} X^i$ converges, and
- $(I - X)^{-1} = \sum_{i=0}^{\infty} X^i$.

Moreover, prove that in the same norm

$$\|(I - X)^{-1}\| \leq (1 - \|X\|)^{-1}.$$

Exercise 1-12. Let K be a matrix in $\mathbb{R}^{n\times n}$ with non-negative entries and let f, g be two vectors in $\mathbb{R}^n$ with strictly positive entries which satisfy

$$(Kf)_i/g_i < \lambda,\ (K^T g)_i/f_i < \mu \quad \forall i \in \{1, \dots, n\}.$$

Prove that $\|K\|_2^2 \leq \lambda\mu$.

Exercise 1-13. Let $A \in \mathbb{R}^{k\times l}, B \in \mathbb{R}^{l\times m}$ and $C \in \mathbb{R}^{m\times m}$. Here $k \geq l$. If A and C are orthogonal, that is if $C^T C = I$ (the identity on $\mathbb{R}^m$) and $A^T A = I$ (the identity on $\mathbb{R}^l$) then show that $\|ABC\|_2 = \|B\|_2$.

Exercise 1-14. Prove that the Frobenius norm of a matrix is unchanged by multplication by unitary matrices. This is an analogue of Theorem 1.33.

Exercise 1-15. Show that for every vector norm $\|\cdot\|$ on $\mathbb{R}^{n\times n}$ there is a number $\lambda > 0$ such that

$$\|A\|_\lambda = \lambda\|A\| \qquad A \in \mathbb{R}^{n\times n}$$

defines a matrix norm.

Chapter 2

Matrix Factorisations

In this chapter we present various matrix factorisations. These are of interest in their own right, and also because they form the basis for many useful numerical algorithms.

There are two groups of results presented in this chapter. The first kind of results factorises a matrix $A \in \mathbb{C}^{n\times n}$ as

$$A = S\tilde{A}S^{-1}$$

where, typically, $\tilde{A}$ is of a simpler form than the original matrix A. Results of this type are matrix diagonalisations and the Jordan canonical form. These factorisations are useful, because properties of $\tilde{A}$ are often easier to understand than properties of A and often questions about A can be reduced to questions about $\tilde{A}$. For example, since $Ax = \lambda x$ implies $\tilde{A}(S^{-1}x) = \lambda(S^{-1}x)$, the matrices A and $\tilde{A}$ have the same eigenvalues (but different eigenvectors). These factorisations are typically used in proofs.

The second group of results, including the QR factorisation and LU factorisation, just splits a matrix A into two, simpler parts:

$$A = BC$$

where B and C are matrices of a simpler form than A is, for example triangular matrices. These factorisations typically form the basis of numerical algorithms, because they allow to split one complicated problem into two simpler ones. This strategy will be used extensively in the later chapters of this text.

2.1 Diagonalisation

A matrix $A \in \mathbb{C}^{n\times n}$ is diagonalised by finding a unitary matrix $Q \in \mathbb{C}^{n\times n}$ and a diagonal matrix $D \in \mathbb{C}^{n\times n}$ such that

$$A = QDQ^*.$$

Since this implies $AQ = DQ$ we see, by considering the individual columns of this matrix equation, that the diagonal elements of D are the eigenvalues of A and the (orthonormal) columns of Q are the corresponding eigenvectors. This insight has several consequences: Firstly, a matrix can be diagonalised if and only if it has a complete, orthonormal system of eigenvectors. And, secondly, there can be no direct algoritms to diagonalise a matrix, since the eigenvalues of a matrix in general cannot be found exactly in finite time (see the discussion around Theorem 8.2). Thus, diagonalisation will be mostly useful as a tool in our proofs and not as part of an algorithm.

The basic result in this section is the Schur triangularisation of a matrix; diagonalisation will follow from this. The next lemma is key in proving the Schur factorisation.

Lemma 2.1. *For all $A \in \mathbb{C}^{n\times n}$ satisfying $\dim(\operatorname{range}(A)) = k \leq n$, there is an orthonormal set $\{y_1 \dots, y_k\} \subseteq \operatorname{range}(A)$ with the property that $Ay_l \in \operatorname{span}\{y_1, \dots, y_l\}$ for $l = 1, \dots, k$.*

Proof. If $k = 1$, then there is a $y_1 \in \mathbb{C}^n$ with $\|y_1\|_2 = 1$ which spans $\operatorname{range}(A)$. Clearly $Ay_1 \in \operatorname{range}(A) = \operatorname{span}\{y_1\}$.

For induction assume that we have the result for some $k < n$. Let A satisfy $\dim(\mathrm{range}(A)) = k+1$. Choose y_1 to be an eigenvector of A with $\|y_1\|_2 = 1$. Let P denote the orthogonal projection onto $\mathrm{span}\{y_1\}$ and $P^\perp = I - P$. Define $A^\perp = P^\perp A$ and note that $\dim(\mathrm{range}(A^\perp)) = k$. By the inductive hypothesis, there is an orthonormal set $\{y_2, \dots, y_{k+1}\} \subseteq \mathrm{range}(A^\perp)$ with the property

$$A^\perp y_l \in \mathrm{span}\{y_2, \dots, y_l\} \qquad l = 2, \dots, k+1.$$

Furthermore we have that y_1 is orthogonal to $\mathrm{span}\{y_2, \dots, y_k\}$. Consider the set $\{y_1, \dots, y_{k+1}\}$. Note that $Ay_1 = \lambda y_1$. Also

$$Ay_l = (PA + P^\perp A)y_l = PAy_l + A^\perp y_l.$$

Since $PAy_l \in \mathrm{span}\{y_1\}$ and $A^\perp y_l \in \mathrm{span}\{y_2, \dots, y_l\}$ we obtain

$$Ay_l \in \mathrm{span}\{y_1, \dots, y_l\},$$

as required. $\square$

Theorem 2.2 (Schur Factorisation)**.** *For any $A \in \mathbb{C}^{n\times n}$, there is a unitary $Q \in \mathbb{C}^{n\times n}$ and an upper triangular $T \in \mathbb{C}^{n\times n}$ such that*

$$A = QTQ^*.$$

Proof. Let $k = \dim(\mathrm{range}(A))$, and construct orthonormal vectors $\{y_1, \dots, y_k\}$ as in Lemma 2.1. Since $\dim\big(\mathrm{range}(A)^\perp\big) = n - k$, we can find an orthonormal basis $\{y_{k+1}, \dots, y_n\}$ of $\mathrm{range}(A)^\perp$. Then $\{y_q, \dots, y_n\}$ is an orthonormal basis of $\mathbb{C}^n$ and

$$Ay_l \in \mathrm{range}(A) = \mathrm{span}\{y_1, \dots, y_k\} \subseteq \mathrm{span}\{y_1, \dots, y_l\}$$

for $l = k+1, \dots, n$. We also have $Ay_l \in \mathrm{span}\{y_1, \dots, y_l\}$ for $l = 1, \dots, k$ and thus

$$Ay_l = \sum_{j=1}^{l} t_{jl} y_j, \qquad l = 1, \dots, n.$$

Letting $Q = \big(y_1 \cdots y_n\big)$ and defining T by $(T)_{ij} = t_{ij}$ for $i \le j$ and $(T)_{ij} = 0$ for $i > j$ we obtain $AQ = QT$ as required. $\square$

Theorem 2.3 (Normal Diagonalisation)**.** *If $A \in \mathbb{C}^{n\times n}$ satisfies $A^*A = AA^*$, then there is unitary $Q \in \mathbb{C}^{n\times n}$ and diagonal $D \in \mathbb{C}^{n\times n}$ such that $A = QDQ^*$.*

Proof. By Schur factorisation, there is T upper triangular and Q unitary such that

$$A = QTQ^*,$$

and it suffices to show that T is diagonal.

We have

$$A^*A = QT^*TQ^* \quad \text{and} \quad QTT^*Q^* = AA^*,$$

and since A is normal we deduce that $T^*T = TT^*$. Now

$$(T^*T)_{ij} = \sum_k (T^*)_{ik}(T)_{kj} = \sum_k (\bar{T})_{ki}(T)_{kj},$$

so that

$$(T^*T)_{ii} = \sum_k |t_{ki}|^2 = \sum_{k=1}^{i} |t_{ki}|^2.$$

Similarly,

$$(TT^*)_{ii} = \sum_{k=i}^{n} |t_{ik}|^2.$$

We now prove that T is diagonal by equating these expressions and using induction.

$$i = 1: \qquad |t_{11}|^2 = \sum_{k=1}^{n} |t_{1k}|^2,$$

and so $t_{1k} = 0$ for $k = 2, \ldots, n$.

Assume for induction in m that $t_{lk} = 0$ for $l = 1, \ldots, m-1$ and all $k \neq l$. Note that we have proved this for $m = 2$. Then

$$(T^*T)_{mm} = \sum_{k=1}^{m} |t_{km}|^2 = |t_{mm}|^2 \qquad \text{(by induction hyp.)}$$

$$(TT^*)_{mm} = \sum_{k=m}^{n} |t_{mk}|^2 = |t_{mm}|^2 + \sum_{k=m+1}^{n} |t_{mk}|^2,$$

and so $t_{mk} = 0$ for $k = m+1, \ldots, n$. Also, $t_{mk} = 0$ for $k = 1, \ldots, m-1$ since T is upper triangular. Thus

$$\begin{aligned} t_{mk} &= 0 \qquad k \neq m \\ \text{and} \quad t_{lk} &= 0 \qquad l = 1, \ldots, m, \ k \neq l, \end{aligned}$$

and the induction is complete. ☐

Remark. In the situation of the preceeding theorem, the diagonal elements of D are the eigenvalues of A and the columns of Q are corresponding eigenvectors. Since Q is unitary, the eigenvectors are orthogonal and thus the matrix A is normal. When combined with the discussion after definition 1.20 this shows that a matrix A is normal if and only if $A^*A = AA^*$.

Theorem 2.4 (Hermitian Diagonalisation). *If $A \in \mathbb{C}^{n\times n}$ is Hermitian, then there exists a unitary matrix $Q \in \mathbb{C}^{n\times n}$ and diagonal $\Lambda \in \mathbb{R}^{n\times n}$ such that $A = Q\Lambda Q^*$.*

Proof. Since Hermitian matrices are normal, A can be factorised in the required form with $\Lambda \in \mathbb{C}^{n\times n}$ diagonal by Theorem 2.3. It remains to show that Λ is real. We have

$$AQ = Q\Lambda,$$

and hence, if $q_1, \ldots, q_n$ are the columns of Q, we get $Aq_i = \lambda_i q_i$ and $\|q_i\| = 1$. This implies

$$\lambda_i = \langle q_i, \lambda_i q_i \rangle = \langle q_i, Aq_i \rangle = \langle Aq_i, q_i \rangle = \langle \lambda_i q_i, q_i \rangle = \overline{\lambda_i}$$

for $i = 1, \ldots, n$ as required. ☐

To illustrate the usefulness of Hermitian diagonalisation, we consider the following application.

Lemma 2.5. *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian and positive definite. Then there is a Hermitian, positive definite matrix $A^{1/2} \in \mathbb{C}^{n\times n}$, the* square root *of A, such that $A = A^{1/2}A^{1/2}$.*

Proof. Since $A \in \mathbb{C}^{n\times n}$ is positive-definite, we have

$$\lambda_i \|x_i\|_2^2 = \langle x_i, Ax_i \rangle > 0$$

for all eigenpairs (x_i, λ_i) and thus all eigenvalues are positive.

By Theorem 2.4 we have

$$A = Q\Lambda Q^*$$

with $\lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Since all $\lambda_i \geq 0$, we may define $\Lambda^{1/2} = \operatorname{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$ and this is real. Now define

$$A^{1/2} = Q\Lambda^{1/2}Q^*. \tag{2.1}$$

Then $A^{1/2}A^{1/2} = Q\Lambda^{1/2}\Lambda^{1/2}Q^* = Q\Lambda Q^* = A$ as required and, since $\sqrt{\lambda_i} > 0$ for all $i = 1, \ldots, n$, the matrix $A^{1/2}$ is Hermitian, positive definite. ☐

Remarks.

- A real, positive number λ has two distinct square roots, $\sqrt{\lambda}$ and $-\sqrt{\lambda}$. Similarly, a Hermitian, positive definite matrix $A \in \mathbb{C}^{n\times n}$ has 2^n distinct square roots, obtained by choosing all possible combination of signs in front of the square roots on the diagonal of $\Lambda^{1/2}$ in (2.1). The square root constructed in the lemma is the only positive one.

- The same principle used to construct the square root of a matrix here, can be used to construct many different functions of a Hermitian matrix: one diagonalises the matrix and applies the function to the eigenvalues on the diagonal.

2.2 Jordan Canonical Form

Definition 2.6. A *Jordan block* $J_n(\lambda) \in \mathbb{C}^{n\times n}$ for $\lambda \in \mathbb{C}$ is the matrix satisfying $J_k(\lambda)_{ii} = \lambda$, $J_k(\lambda)_{i,i+1} = 1$, and $J_k(\lambda)_{ij} = 0$ else, for $i, j = 1, \dots, n$, i.e. a matrix of the form

$$J_k(\lambda) = \begin{pmatrix} \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{pmatrix}.$$

A *Jordan matrix* is a block diagonal matrix $J \in \mathbb{C}^{n\times n}$ of the form

$$J = \begin{pmatrix} J_{n_1}(\lambda_1) & & & \\ & J_{n_2}(\lambda_2) & & \\ & & \ddots & \\ & & & J_{n_k}(\lambda_k) \end{pmatrix}$$

where $\sum_{j=1}^{k} n_j = n$.

The following factorisation is of central theoretical importance.

Theorem 2.7 (Jordan Canonical Form)**.** *For any $A \in \mathbb{C}^{n\times n}$ there is an invertible $S \in \mathbb{C}^{n\times n}$ and a Jordan matrix $J \in \mathbb{C}^{n\times n}$ satisfying*

$$A = SJS^{-1}$$

where the diagonal elements $\lambda_1, \dots, \lambda_k$ of the Jordan blocks are the eigenvalues of A.

Remarks.

1. Clearly both the normal and Hermitian diagonalisation results reveal the eigenvalues of A: they are simply the diagonal entries of D and Λ. This is also true of the Jordan and Schur factorisations. The following lemma shows that triangular matrices reveal their eigenvalues as diagonal entries. Since both the Jordan Canonical Form and the Schur Factorisation provide similarity transformations of A which reduce it to triangular form, and since similarity transformations leave the eigenvalues unchanged, this establishes the desired properties. Thus all the preceding factorisations are *eigenvalue revealing factorisations.*

2. An eigenvalue revealing factorisation cannot be achieved in a finite number of arithmetic steps, in dimension $n \geq 5$, since it implies factorisation of a polynomial equation of degree n. See Chapter 8.

Lemma 2.8. *Let $T \in \mathbb{C}^{n\times n}$ be triangular. Then*

$$\det(T) = \prod_{i=1}^{n} T_{ii}.$$

Hence the eigenvalues of T are its diagonal entries.

Proof. Let $T_j \in \mathbb{C}^{j\times j}$ be upper triangular:

$$T_j = \begin{pmatrix} a & b^* \\ 0 & T_{j-1} \end{pmatrix}, \qquad \begin{array}{c} a \in \mathbb{C}, \quad b, 0 \in \mathbb{C}^{j-1}, \\ T_{j-1} \in \mathbb{C}^{(j-1)\times(j-1)} \text{ upper triangular.} \end{array}$$

Then $\det T_j = a \det(T_{j-1})$. By induction,

$$\det(T) = \prod_{i=1}^{n} T_{ii}.$$

Eigenvalues of T are λ such that $\det(T - \lambda I) = 0$. Now $T - \lambda I$ is triangular with diagonal entries $T_{ii} - \lambda$, therefore

$$\det(T - \lambda I) = \prod_{i=1}^{n} (T_{ii} - \lambda).$$

Hence $\det(T - \lambda_i) = 0$ if and only if $\lambda_i = T_{ii}$ for some $i = 1, \dots, n$. $\square$

As an example of the central theoretical importance of the Jordan normal form we now prove a useful lemma showing that a matrix norm can be constructed which, for a given matrix A, has norm arbitrarily close to the spectral radius.

Definition 2.9. A *δ-Jordan block* $J_n^\delta(\lambda) \in \mathbb{C}^{n\times n}$ for $\lambda \in \mathbb{C}$ is the matrix satisfying $J_k^\delta(\lambda)_{ii} = \lambda$, $J_k^\delta(\lambda)_{i,i+1} = \delta$, and $J_k^\delta(\lambda)_{ij} = 0$ else, for $i, j = 1, \dots, n$. A *δ-Jordan matrix* is a block diagonal matrix $J^\delta \in \mathbb{C}^{n\times n}$ of the form

$$J = \begin{pmatrix} J_{n_1}^\delta(\lambda_1) & & & \\ & J_{n_2}^\delta(\lambda_2) & & \\ & & \ddots & \\ & & & J_{n_k}^\delta(\lambda_k) \end{pmatrix}$$

where $\sum_{j=1}^{k} n_j = n$.

Lemma 2.10. *Let $A \in \mathbb{C}^{n\times n}$ and $\delta > 0$. Then there is a vector norm $\|\cdot\|_S$ on $\mathbb{C}^n$ such that the induced matrix norm satisfies $\rho(A) \le \|A\|_S \le \rho(A) + \delta$.*

Proof. From Theorem 1.30 we already know $\rho(A) \le \|A\|$ for every matrix norm $\|\cdot\|$. Thus we only have to show the second inequality of the claim.

Let $J = S^{-1}AS$ be the Jordan Canonical Form of A and $D_\delta = \operatorname{diag}(1, \delta, \delta^2, \dots, \delta^{n-1})$. Then

$$(SD_\delta)^{-1}A(SD_\delta) = D_\delta^{-1} J D_\delta = J^\delta.$$

Define a vector norm $\|\cdot\|_S$ on $\mathbb{C}^n$ by

$$\|x\|_S = \left\|(SD_\delta)^{-1}x\right\|_\infty$$

for all $x \in \mathbb{C}^n$. Then the induced matrix norm satisfies

$$\begin{aligned} \|A\|_S &= \max_{x\neq 0} \frac{\|Ax\|_S}{\|x\|_S} \\ &= \max_{x\neq 0} \frac{\|(SD_\delta)^{-1}Ax\|_\infty}{\|(SD_\delta)^{-1}x\|_\infty} \\ &= \max_{y\neq 0} \frac{\|(SD_\delta)^{-1}A(SD_\delta)y\|_\infty}{\|y\|_\infty} \\ &= \left\|(SD_\delta)^{-1}A(SD_\delta)\right\|_\infty \\ &= \|J^\delta\|_\infty. \end{aligned}$$

Since we know the $\|\cdot\|_\infty$-matrix norm from Theorem 1.28 and we have calculated the explicit form of the matrix $(SD_\delta)^{-1}A(SD_\delta)$ above, this is easy to evaluate. We get $\|A\| \le \max_i |\lambda_i| + \delta = \rho(A) + \delta$. This completes the proof. $\square$

Remark. In general, $\rho(\,\cdot\,)$ is not a norm. But note that if the Jordan matrix J is diagonal then $\delta = 0$ and we can deduce the existence of a norm in which $\|A\|_{\mathcal{S}} = \rho(A)$. This situation arises whenever A is diagonalisable.

2.3 Singular Value Decomposition

The singular value decomposition is based on the fact that, for any matrix A, it is possible to find a set of real positive σ_i and vectors u_i, v_i such that

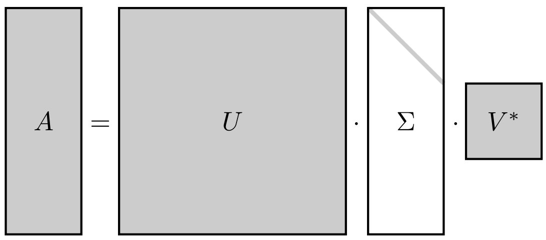
$$Av_i = \sigma_i u_i.$$

The σ_i are known as singular values and, in some applications, are more useful than eigenvalues. This is because the singular values exist even for non-square matrices, because they are always real, and because the $\{u_i\}$ and $\{v_i\}$ always can be chosen orthogonal. Furthermore, the singular value decomposition is robust to perturbations, unlike the Jordan canonical form.

Definition 2.11. Let $A \in \mathbb{C}^{m\times n}$ with $m, n \in \mathbb{N}$. A factorisation

$$A = U\Sigma V^*$$

is called *singular value decomposition* (SVD) of A, if $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary, $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal, and the diagonal entries of Σ are $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = min(m, n)$. The values $\sigma_1, \ldots, \sigma_p$ are called *singular values* of A. The columns of U are called *left singular vectors* of A, the columns of V are *right singular vectors* of A.



Theorem 2.12 (SVD)**.** *Every matrix has a singular value decomposition and the singular values are uniquely determined.*

Proof. Let $A \in \mathbb{C}^{m\times n}$. We prove existence of the SVD by induction over $p = \min(m, n)$. If $p = 0$ the matrices U, V, and Σ are just the appropriately shaped empty matrices (one dimension is zero) and there is nothing to show.

Assume $p > 0$ and that the existence of the SVD is already known for matrices where one dimension is smaller than $\min(m, n)$. Let $\sigma_1 = \|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2$. Since the map $v \mapsto Av$ is continuous and the set $\{\, x \mid \|x\|_2 = 1 \,\} \subseteq \mathbb{C}^n$ is compact, the image $\{\, Ax \mid \|x\|_2 = 1 \,\} \subseteq \mathbb{C}^m$ is also compact. Since $\|\cdot\|_2 \colon \mathbb{C}^n \to \mathbb{R}$ is continuous there is a $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$ and

$$\|Av_1\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \sigma_1.$$

Defining $u_1 = Av_1/\sigma_1$ we get $\|u_1\|_2 = 1$.

Extend $\{v_1\}$ to an orthonormal basis $\{v_1, \ldots, v_n\}$ of $\mathbb{C}^n$ and $\{u_1\}$ to an orthonormal basis $\{u_1, \ldots, u_m\}$ of $\mathbb{C}^m$. Consider the matrices

$$U_1 = (u_1, \ldots, u_m) \in \mathbb{C}^{m\times m}$$

and

$$V_1 = (v_1, \ldots, v_n) \in \mathbb{C}^{n\times n}.$$

Then the product $U_1^* A V_1$ is of the form

$$S = U_1^* A V_1 = \begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix}$$

with $w \in \mathbb{C}^{n-1}, 0 \in \mathbb{C}^{m-1}$ and $B \in \mathbb{C}^{(m-1)\times(n-1)}$.

For unitary matrices U we have $\|Ux\|_2 = \|x\|_2$ and thus

$$\|S\|_2 = \max_{x\neq 0} \frac{\|U_1^* A V_1 x\|_2}{\|x\|_2} = \max_{x\neq 0} \frac{\|AV_1 x\|_2}{\|V_1 x\|_2} = \|A\|_2 = \sigma_1.$$

On the other hand we get

$$\left\| S \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \sigma_1^2 + w^* w \\ Bw \end{pmatrix} \right\|_2 \geq \sigma_1^2 + w^* w = \left(\sigma_1^2 + w^* w\right)^{1/2} \left\| \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2$$

and thus $\|S\|_2 \geq (\sigma_1^2 + w^* w)^{1/2}$. Thus we conclude that $w = 0$ and thus

$$A = U_1 S V_1^* = U_1 \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} V_1^*.$$

By the induction hypothesis the $(m-1)\times(n-1)$-matrix B has a singular value decomposition

$$B = U_2 \Sigma_2 V_2^*.$$

Then

$$A = U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & V_2^* \end{pmatrix} V_1^*$$

is a SVD of A and existence of the SVD is proved.

Uniqueness of the largest singular value σ_1 holds, since σ_1 is uniquely determined by the relation

$$\|A\|_2 = \max_{x\neq 0} \frac{\|U\Sigma V^* x\|_2}{\|x\|_2} = \max_{x\neq 0} \frac{\|\Sigma x\|_2}{\|x\|_2} = \sigma_1.$$

Uniqueness of $\sigma_2, \ldots, \sigma_n$ follows by induction as above. □

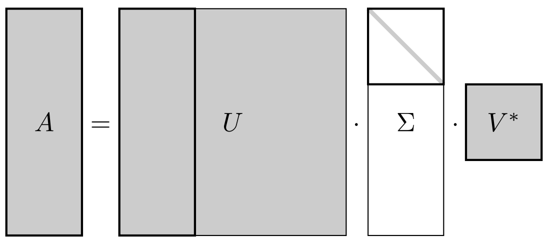
The penultimate line of the proof shows that, with the ordering of singular values as defined, we have the following:

Corollary 2.13. *For any matrix $A \in \mathbb{C}^{m\times n}$ we have $\|A\|_2 = \sigma_1$.*

Remarks.

1. Inspection of the above proof reveals that for real matrices A the matrices U and V are also real.

2. If $m > n$ then the last $m-n$ columns of U do not contribute to the factorisation $A = U\Sigma V^*$:



   Hence we can also write A as $A = \hat{U}\hat{\Sigma}V^*$ where $\hat{U} \in \mathbb{C}^{m\times n}$ consists of the first n columns of U and $\hat{\Sigma} \in \mathbb{C}^{n\times n}$ consists of the first n rows of Σ. This factorisation is called the *reduced singular value decomposition* (reduced SVD) of A.

3. Since we have $A^*A = V\Sigma^* U^* U \Sigma V^* = V\Sigma^*\Sigma V^*$ and thus $A^*A \cdot V = V \cdot \Sigma^*\Sigma$, we find $A^*Av_j = \sigma_j^2 v_j$ for the columns $v_1, \ldots, v_n$ of V. This shows that the vectors v_j are eigenvectors of A^*A with eigenvalues σ_j^2.

4. From the proof we see that we can get the $\|\cdot\|_2$-norm of a matrix from its SVD: we have $\|A\|_2 = \sigma_1$.

Theorem 2.14. *For $m \geq n$ the SVD has the following properties:*

1. *If $A \in \mathbb{R}^{n\times n}$ is Hermitian then $A = Q\Lambda Q^*$ with $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ and $Q = \begin{pmatrix} q_1 & \cdots & q_n \end{pmatrix}$. An SVD of A may be found in the form $A = U\Sigma V^T$ with $U = Q$, $\Sigma = |\Lambda|$, and*
$$V = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}, \qquad v_i = \operatorname{sgn}(\lambda_i) q_i.$$
2. *The eigenvalues of A^*A are σ_i^2 and the eigenvectors of A^*A are the right singular vectors v_i.*
3. *The eigenvalues of AA^* are σ_i^2 and $(m-n)$ zeros. The (right) eigenvectors of AA^* corresponding to eigenvalues σ_i^2 are the left singular vectors u_i corresponding to the singular values σ_i.*

Proof. 1. By definition.

2. We have, from the reduced SVD,
$$A = \hat{U}\hat{\Sigma}V^* \implies A^*A = V\hat{\Sigma}^2V^* \in \mathbb{R}^{n\times n}$$
Since V is orthogonal and $\hat{\Sigma}^2 = \operatorname{diag}(\sigma_1^2, \dots, \sigma_n^2)$, the result follows.

3. We have
$$A = U\Sigma V^* \implies AA^* = U\Sigma\Sigma^*U^*$$
where $\tilde{U} \in \mathbb{R}^{m\times(m-n)}$ is any matrix such that $[U \quad \tilde{U}] \in \mathbb{R}^{m\times m}$ is orthogonal. The result then follows since
$$\Sigma\Sigma^* = \begin{pmatrix} \hat{\Sigma}^2 & 0 \\ 0 & 0 \end{pmatrix}.$$

□

For the rest of this section let $A \in \mathbb{C}^{m\times n}$ be a matrix with singular value decomposition $A = U\Sigma V^*$ and singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0 = \cdots = 0$. To illustrate the usefulness of the SVD we prove several fundamental results about it.

Theorem 2.15. *The rank of A is equal to r.*

Proof. Since U and V^* are invertible we have $\operatorname{rank}(A) = \operatorname{rank}(\Sigma) = r$. □

Theorem 2.16. *We have* $\operatorname{range}(A) = \operatorname{span}\{u_1, \dots, u_r\}$ *and* $\ker(A) = \operatorname{span}\{v_{r+1}, \dots, v_n\}$.

Proof. Since Σ is diagonal and V^* is invertible we have
$$\operatorname{range}(\Sigma V^*) = \operatorname{range}(\Sigma) = \operatorname{span}\{e_1, \dots, e_r\} \subseteq \mathbb{C}^m.$$
This shows
$$\operatorname{range}(A) = \operatorname{range}(U\Sigma V^*) = \operatorname{span}\{u_1, \dots, u_r\} \subseteq \mathbb{C}^m.$$

We also have
$$\ker(A) = \ker(U\Sigma V^*) = \ker(\Sigma V^*).$$
Since V is orthogonal we can conclude
$$\ker(A) = \operatorname{span}\{v_{r+1}, \dots, v_n\} \subseteq \mathbb{C}^n.$$

□

Theorem 2.17 (The SVD and Eigenvalues). *Let $A \in \mathbb{R}^{n\times n}$ be invertible with SVD $A = U\Sigma V^T$. If*

$$H = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix} \in \mathbb{R}^{2n\times 2n}$$

and

$$\begin{aligned} U &= (u_1 \;\cdots\; u_n) \\ V &= (v_1 \;\cdots\; v_n) \\ \Sigma &= \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \end{aligned}$$

then H has

- *$2n$ eigenvalues $\{\pm\sigma_i\}_{i=1}^n$*
- *eigenvectors $\{\frac{1}{\sqrt{2}}\begin{pmatrix} v_i \\ \pm u_i \end{pmatrix} \mid i = 1, \ldots, n\}$*

Proof. If $Hx = \lambda x$ with $x = (y^T, z^T)^T$ then

$$\begin{aligned} A^T z &= \lambda y \\ Ay &= \lambda z. \end{aligned}$$

Hence

$$\begin{aligned} A^T(\lambda z) &= \lambda^2 y \\ \text{and} \quad A^T A y &= \lambda^2 y. \end{aligned}$$

Thus $\lambda^2 \in \{\sigma_1^2, \ldots, \sigma_n^2\}$ and so the $2n$ eigenvalues of H are drawn from the set

$$\{\pm\sigma_1, \ldots, \pm\sigma_n\}.$$

Note that

$$A^T U = V\Sigma, \; AV = U\Sigma$$

and so

$$Av_i = \sigma_i u_i, \; A^T u_i = \sigma_i v_i.$$

The eigenvalue problem for H may be written as

$$Ay = \lambda z, \; A^T z = \lambda y.$$

Hence, taking $\lambda = \pm\sigma_i$, we obtain $2n$ solutions of the eigenvalue problem given by

$$(y^T, z^T) = \frac{1}{\sqrt{2}}(v_i, \pm u_i).$$

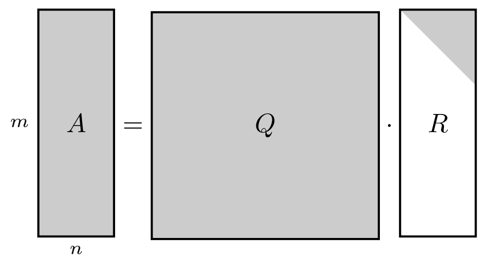
This exhibits a complete set of $2n$ eigenvectors for H. □

2.4 QR Factorisation

The SVD factorisation, like the four preceding it, reveals eigenvalues; hence it cannot be achieved in a finite number of steps. The next three factorisations, QR, LU and Cholesky do not reveal eigenvalues and, as we will show in later chapters, can be achieved in a polynomial number of operations, with respect to dimension n. Recall the following classical algorithm for the construction of an orthonormal basis from the columns of a matrix A.

Algorithm (Gram-Schmidt orthonormalisation).
input: $A \in \mathbb{C}^{m\times n}$ with $m \geq n$
output: $Q \in \mathbb{C}^{m\times m}$ unitary, $R \in \mathbb{C}^{m\times n}$ upper triangular with $A = QR$
let $a_1, \ldots, a_n \in \mathbb{C}^m$ be the columns of A.

$R = 0$
for j=1,...,n **do**
  $\hat{q}_j = a_j - \sum_{k=1}^{j-1} r_{kj} q_k$ with $r_{kj} = \langle q_k, a_j\rangle$
  $r_{jj} = \|\hat{q}_j\|_2$
  if $r_{jj} > 0$ **then**
    $q_j = \hat{q}_j / r_{jj}$
  else
    let q_j be an arbitrary normalised vector orthogonal to $q_1, \ldots, q_{j-1}$
  end if
end for
choose $q_{n+1}, \ldots, q_m$ to make $q_1, \ldots, q_m$ an orthonormal basis.
let $q_1, \ldots, q_m \in \mathbb{C}^m$ be the columns of Q; let $(R)_{ij} = r_{ij}$, $i \leq j$, $(R)_{ij} = 0$ otherwise.



From this algorithm we prove:

Theorem 2.18 (QR factorisation)**.** *Every matrix $A \in \mathbb{C}^{m\times n}$ with $m \geq n$ can be written as $A = QR$ where $Q \in \mathbb{C}^{m\times m}$ is unitary and $R \in \mathbb{C}^{m\times n}$ is upper triangular.*

Proof. The Gram-Schmidt algorithm calculates matrices Q and R with

$$(QR)_{ij} = \Big(\sum_{k=1}^{j} q_k r_{kj}\Big)_i = \Big(\sum_{k=1}^{j-1} q_k r_{kj} + \hat{q}_j\Big)_i = (a_j)_i$$

and thus we get $A = QR$.

By construction we have $\|q_j\|_2 = 1$ for $j = 1, \ldots, m$. We use induction to show that the columns $q_1, \ldots, q_j$ are orthogonal for all $j \in \{1, \ldots, m\}$. For $j = 1$ there is nothing to show. Now let $j > 1$ and assume that $q_1, \ldots, q_{j-1}$ are orthogonal. We have to prove $\langle q_i, q_j\rangle = 0$ for $i = 1, \ldots, j-1$. If $r_{jj} = 0$, this holds by definition of q_j. Otherwise we have

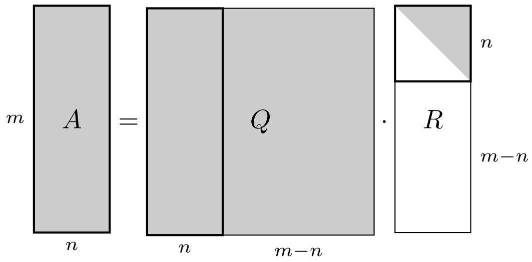
$$\begin{aligned}
\langle q_i, q_j\rangle &= \frac{1}{r_{jj}} \langle q_i, \hat{q}_j\rangle \\
&= \frac{1}{r_{jj}} \Big(\langle q_i, a_j\rangle - \sum_{k=1}^{j-1} r_{kj} \langle q_i, q_k\rangle\Big) \\
&= \frac{1}{r_{jj}} \big(\langle q_i, a_j\rangle - r_{ij}\big) \\
&= 0.
\end{aligned}$$

Thus induction shows that the columns of Q are orthonormal and hence that Q is unitary. □

Remarks.

1. The factorisation in the theorem is called *full QR factorisation*. Since all entries below the diagonal of R are 0, the columns $n+1, \ldots, m$ of Q do not contribute to the product QR.

Let $\hat{Q} \in \mathbb{C}^{m \times n}$ consist of the first n columns of Q and $\hat{R} \in \mathbb{C}^{n \times n}$ consist of the first n rows of R. Then we have $A = \tilde{Q}\tilde{R}$. This is called the *reduced QR factorisation* of A. The following picture illustrates the situation.



2. For $m = n$ we get square matrices $Q, R \in \mathbb{C}^{n \times n}$. Since

$$\det(A) = \det(QR) = \det(Q)\det(R)$$

and $\det(Q) \in \{+1, -1\}$ the matrix R is invertible if and only if A is invertible.

3. The Gram-Schmidt orthonormalisation algorithm is numerically unstable and should not be used to calculate a QR factorisation in practice.

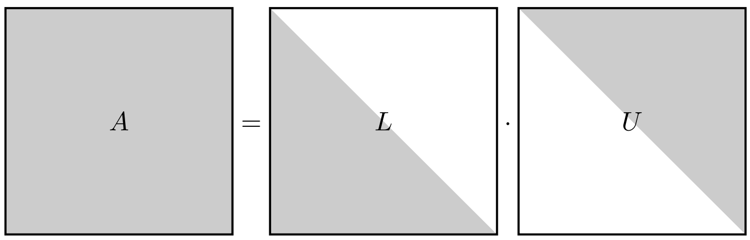
2.5 LU Factorisation

Definition 2.19. A triangular matrix is said to be *unit* if all diagonal entries are equal to 1.

Definition 2.20. The j^{th} *principal sub-matrix* of a matrix $A \in \mathbb{C}^{n \times n}$ is the matrix $A_j \in \mathbb{C}^{j \times j}$ with $(A_j)_{kl} = a_{kl}$ for $1 \leq k, l \leq j$.

Theorem 2.21 (LU Factorisation). *a) Let $A \in \mathbb{C}^{n \times n}$ be a matrix such that A_j is invertible for $j = 1, \ldots, n$. Then there is a unique factorisation $A = LU$ where $L \in \mathbb{C}^{n \times n}$ is unit lower triangular and $U \in \mathbb{C}^{n \times n}$ is non-singular upper triangular. b) If A_j is singular for one $j \in \{1, \ldots, n\}$ then there is no such factorisation.*

The following picture gives a graphical representation of the LU factorisation.



Proof. a) We use a proof by induction: If $n = 1$ we have $a_1 \neq 0$ by assumption and can set $L = (1) \in \mathbb{C}^{1 \times 1}$ and $U = (a_{11}) \in \mathbb{C}^{1 \times 1}$ to get $A = LU$. Since L is the only unit lower triangular 1×1-matrix the factorisation is unique.

Now let $n > 1$ and assume that any matrix $A \in \mathbb{C}^{(n-1)\times(n-1)}$ can be uniquely factorised in the required form $A = LU$ if all its principal sub-matrices are invertible. We write $A \in \mathbb{C}^{n \times n}$ as

$$A = \begin{pmatrix} A_{n-1} & b \\ c^* & a_{nn} \end{pmatrix} \tag{2.2}$$

where A_{n-1} is the $(n-1)^{\text{th}}$ principal sub-matrix of A, and $b, c \in \mathbb{C}^{(n-1)}$ and $a_{nn} \in \mathbb{C}$ are the remaining blocks. We are looking for a factorisation of the form

$$A = \begin{pmatrix} L & 0 \\ \ell^* & 1 \end{pmatrix} \begin{pmatrix} U & u \\ 0 & \eta \end{pmatrix} = \begin{pmatrix} LU & Lu \\ \ell^* U & \ell^* u + \eta \end{pmatrix} \tag{2.3}$$

with $L \in \mathbb{C}^{(n-1)\times(n-1)}$ unit lower triangular, $U \in \mathbb{C}^{(n-1)\times(n-1)}$ invertible upper triangular, $\ell, u \in \mathbb{C}^{n-1}$ and $\eta \in \mathbb{C}\backslash\{0\}$. We compare the blocks of (2.2) and (2.3).

By the induction hypothesis L and U with $A_{n-1} = LU$ exist and are unique. Since the matrix L is invertible the condition $Lu = b$ determines a unique vector u. Since U is invertible there is an uniquely determined ℓ with $U^*\ell = c$ and thus $\ell^*U = c^*$. Finally the condition $\ell^*u + \eta = a_{nn}$ uniquely determines $\eta \in \mathbb{C}$. This shows that the required factorisation for A exists and is unique. Since $0 \neq \det(A) = 1 \cdot \eta \cdot \det U$ the upper triangular matrix is non-singular and $\eta \neq 0$.

b) Assume that A has an LU factorisation and let $j \in \{1, \ldots, n\}$. Then we can write $A = LU$ in block form as

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix}$$

where $A_{11}, L_{11}, U_{11} \in \mathbb{C}^{j\times j}$. We get

$$\det(A_j) = \det(A_{11}) = \det(L_{11}U_{11}) = \det(L_{11})\det(U_{11}) = 1 \cdot \det(U_{11}) \neq 0$$

and thus A_j is non-singular. □

To illustrate the failure of LU factorisation when the principal submatrices are not invertible, consider the following matrix:

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

This matrix is clearly non-singular: $\det(A) = 2$. However, both principal sub-matrices are singular:

$$A_1 = (0)$$
$$A_2 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$

and therefore the factorisation $A = LU$ is not possible. In contrast,

$$A' = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

(which is a permutation of the rows of A) has non-singular principal sub-matrices

$$A'_1 = (1)$$
$$A'_2 = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$$

and hence A' has an LU factorisation.

Because a non-singular matrix may not have an LU factorisation, while that same matrix with its rows interchanged may be factorisable, it is of interest to study the effect of *permutations* on LU factorisation.

Theorem 2.22 (LU Factorisation with Permutations). *If $A \in \mathbb{C}^{n\times n}$ is invertible, then there exists a permutation matrix $P \in \mathbb{C}^{n\times n}$, unit lower triangular $L \in \mathbb{C}^{n\times n}$, and non-singular upper triangular $U \in \mathbb{C}^{n\times n}$ such that $PA = LU$.*

Proof. We prove the result by induction. Note that the base case $n = 1$ is straightforward: $P = L = 1$ and $U = A \neq 0$. Now assume the theorem is true for the $(n-1)\times(n-1)$ case. Let $\tilde{A} \in \mathbb{C}^{n\times n}$ be invertible. Choose a permutation P_1 such that

$$(P_1\tilde{A})_{11} := a \neq 0.$$

This is possible since $\tilde{A}$ invertible implies that the first column of $\tilde{A}$ has a non-zero entry and P_1 permutes rows. Now we can factorise $P_1\tilde{A}$ as follows:

$$P_1\tilde{A} := \begin{pmatrix} a & u^* \\ l & B \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l/a & I \end{pmatrix} \begin{pmatrix} a & u^* \\ 0 & A \end{pmatrix}$$

which is possible if $A = B - lu^*/a = B - \frac{1}{a} l \otimes u$. Now

$$0 \neq \det(\tilde{A}) = \pm \det(P_1\tilde{A}) = a \det(A)$$

and so $\det(A) \neq 0$ since $a \neq 0$. Thus A is invertible and $A = P_2LU$. Hence

$$\begin{aligned} P_1\tilde{A} &= \begin{pmatrix} 1 & 0 \\ l/a & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & P_2L \end{pmatrix} \begin{pmatrix} a & u^* \\ 0 & U \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ l/a & P_2L \end{pmatrix} \begin{pmatrix} a & u^* \\ 0 & U \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & P_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ P_2^T l/a & L \end{pmatrix} \begin{pmatrix} a & u^* \\ 0 & U \end{pmatrix} \\ &= P_3\tilde{L}\tilde{U}, \end{aligned}$$

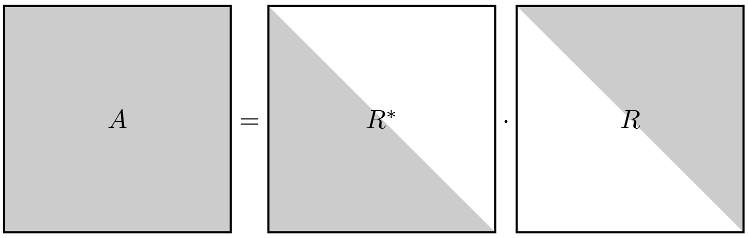
and therefore

$$P_3^T P_1 \tilde{A} = \tilde{L}\tilde{U}.$$

Note that $\tilde{L}$ is unit lower triangular and that $\det(\tilde{U}) = a\det(U) \neq 0$ so that $\tilde{U}$ is non-singular upper trangular. Since $P = P_3^T P_1$ is a permutation the result follows. □

2.6 Cholesky Factorisation

Theorem 2.23 (Cholesky Factorisation). *If $A \in \mathbb{C}^{n\times n}$ is positive definite then there exists a unique upper triangular $R \in \mathbb{C}^{n\times n}$ with positive diagonal elements such that $A = R^*R$.*



Proof. We use induction. The claim is clearly true if $n = 1$: $A = \alpha \in \mathbb{R}^+$, $R = \sqrt{\alpha}$.

Assume the claim holds true for $A_{n-1} \in \mathbb{C}^{(n-1)\times(n-1)}$:

$$A_{n-1} = R_{n-1}^* R_{n-1}, \qquad (R_{n-1})_{ii} > 0 \quad i = 1, \ldots, n-1.$$

Then write $A \in \mathbb{C}^{n\times n}$ as

$$A = \begin{pmatrix} A_{n-1} & c \\ c^* & \alpha \end{pmatrix}.$$

It is straightforward to show that A_{n-1} is Hermitian and positive definite, and that α is real and positive, because A is Hermitian and positive definite.

We can now attempt to factor A as follows:

$$\begin{aligned} A = R^*R &:= \begin{pmatrix} R_{n-1}^* & 0 \\ r^* & \beta \end{pmatrix} \begin{pmatrix} R_{n-1} & r \\ 0 & \beta \end{pmatrix} \\ &= \begin{pmatrix} A_{n-1} & R_{n-1}^* r \\ r^* R_{n-1} & \|r\|_2^2 + \beta^2 \end{pmatrix}. \end{aligned}$$

For this factorisation to work we require $r \in \mathbb{C}^{n-1}$ and $\beta \in \mathbb{R}^+$ such that

$$\begin{aligned} R_{n-1}^* r &= c \\ \beta^2 &= \alpha - \|r\|_2^2. \end{aligned}$$

Since R_{n-1} is non-singular (positive diagonal elements), r and β are uniquely defined.

It remains to show $\beta \in \mathbb{R}^+$. Note that, since A has positive eigenvalues $\det(A) > 0$ and so

$$\begin{aligned} 0 < \det(A) &= \det(R^* R) = \det(R^*)\det(R) \\ &= \beta^2 \det(R_{n-1}^*)\det(R_{n-1} = \beta^2 \det(R_{n-1})^2. \end{aligned}$$

Here we have used the fact that $\det(R_{n-1}^*) = \det(R_{n-1})$ because R_{n-1} is triangular with real positive entries. But $\det(R_{n-1})^2 \in \mathbb{R}^+$ since R_{n-1} has diagonals in $\mathbb{R}^+$. Thus $\beta^2 > 0$ and we can choose $\beta \in \mathbb{R}^+$. $\square$

Bibliography

The books [Bha97], [HJ85] and [Ner71] all present matrix factorisations in a theoretical context. The books [Dem97], [GL96] and [TB97] all present related material, focussed on applications to computational algorithms. Theorem 2.7 is proved in [Ner71].

Exercises

Exercise 2-1. By following the proof of the existence of a singular value decomposition, find an SVD for the following matrices:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} -2 & 0 & 1 \\ 2 & 0 & 1 \\ 0 & \sqrt{18} & 0 \end{pmatrix}.$$

Exercise 2-2. Show that a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ can be written in the form

$$A = \begin{pmatrix} a & z^T \\ z & A_1 \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ \frac{1}{\alpha} z & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & A_1 - \frac{1}{a} z z^T \end{pmatrix} \begin{pmatrix} \alpha & \frac{1}{\alpha} z^T \\ 0 & I \end{pmatrix}.$$

where $\alpha = \sqrt{a}$. Based on this observation find an alternative proof of Theorem 2.23, for the Cholesky factorisation of real symmetric positive definite matrices.

Exercise 2-3. Let $A \in \mathbb{R}^{m\times n}$ have singular values $(\sigma_i \mid i = 1, \ldots, n)$. Show that $\|A\|_2 = \sigma_{max}$ and, if $m = n$ and A is invertible, $\|A^{-1}\|_2^{-1} = \sigma_{min}$.

Exercise 2-4. Prove Theorem 1.16 concerning properties of similar matrices.

Chapter 3

Stability and Conditioning

Rounding errors lead to computational results which are different from the theoretical ones. The methods from this chapter will help us to answer the following question: how close is the calculated result to the correct one?

It is common to view the computed solution of a problem in linear algebra as the exact solution to a perturbed problem. This approach to the study of error is known as *backward error analysis*. If the perturbation is small (when measured in terms of machine precision) the algorithm is termed *backward stable*; when the perturbation is not small it is referred to as *unstable*. Thus the effect of computing the solution in floating point arithmetic, using a particular algorithm, is converted into a perturbation of the original problem. Once this perturbed problem is known, the central issue in studying error is to estimate the difference between the solution of the original problem and a perturbed problem: the issue of *conditioning*. Conditioning is a property of the problem at hand, whilst the size of the perturbation arising in the backward error analysis framework is a property of the algorithm.

The three problems (SLE), (LSQ) and (EVP) can all be viewed as solving a linear or nonlinear equation of the form

$$G(y, \eta) = 0, \tag{3.1}$$

for some $G\colon \mathbb{C}^p \times \mathbb{C}^q \to \mathbb{C}^p$. Here y is the solution that we want to find, and η the data which defines the problem. For (SLE) and (LSQ) the problem is linear and $y = x$ (see below for a proof of this for (LSQ)) and for (EVP) it is nonlinear and $y = (x, \lambda)$. The parameter η is thus defined by the matrix A and, for (SLE) and (LSQ), the matrix A and the vector b. Backward error analysis will enable us to show that the computed solution $\hat{y}$ solves

$$G(\hat{y}, \hat{\eta}) = 0.$$

Conditioning is concerned with estimating $y - \hat{y}$ in terms of $\eta - \hat{\eta}$. Since conditioning of a given problem will affect the error analysis for all algorithms applied to it, we start with its study, for each of our three problems in turn. The following definition will be useful in this chapter.

Definition 3.1. Let $f = f(\eta)\colon \mathbb{C}^q \to \mathbb{C}^p$. We write $f = \mathcal{O}(\|\eta\|^\alpha)$ for $\alpha > 0$ if there is a constant $C > 0$ such that $\|f\| \leq C\|\eta\|^\alpha$ uniformly as $\eta \to 0$.

3.1 Conditioning of SLE

A problem is called *well conditioned* if small changes in the problem only lead to small changes in the solution and *badly conditioned* if small changes in the problem can lead to large changes in the solution. In this context the following definition is central.

Definition 3.2. The *condition number* $\kappa(A)$ of a matrix $A \in \mathbb{C}^{n\times n}$ in the norm $\|\cdot\|$ is the number

$$\kappa(A) = \begin{cases} \|A\| \cdot \|A^{-1}\| & \text{if } A \text{ is invertible and} \\ +\infty & \text{else.} \end{cases}$$

For this section fix a vector norm $\|\cdot\|$ and an induced matrix norm $\|\cdot\|$. By Theorem 1.27 these norms then satisfy

$$\begin{aligned} \|Ax\| &\leq \|A\| \cdot \|x\| \qquad \text{for all } x \in \mathbb{C}^n, A \in \mathbb{C}^{n\times n} \\ \|I\| &= 1. \end{aligned}$$

Remark. We always have $\|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \kappa(A)$. For induced matrix norms this implies $\kappa(A) \geq 1$ for every $A \in \mathbb{C}^{n\times n}$.

Example. Let A be real, symmetric and positive-definite with eigenvalues

$$\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = \lambda_{\min} > 0.$$

Then we have $\|A\|_2 = \lambda_{\max}$. Since the matrix A^{-1} has eigenvalues $1/\lambda_1, \ldots, 1/\lambda_n$ we find $\|A^{-1}\|_2 = \lambda_{\min}^{-1}$ and thus the condition number of A in the 2-norm is

$$\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{3.2}$$

Proposition 3.3. *Let $Ax = b$ and $A(x + \Delta x) = b + \Delta b$. Assume $b \neq 0$. Then*

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\Delta b\|}{\|b\|}.$$

Proof. If A is not invertible the right hand side of the inequality is $+\infty$ and the result holds. Thus we assume that A is invertible and we have

$$\|b\| = \|Ax\| \leq \|A\|\|x\|. \tag{3.3}$$

Since $A^{-1}\Delta b = \Delta x$ we get

$$\frac{\|\Delta x\|}{\|x\|} = \frac{\|A^{-1}\Delta b\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\Delta b\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\Delta b\|}{\|b\|}$$

where the last inequality is a consequence of (3.3). □

The previous proposition gave an upper bound on how much the solution of the equation $Ax = b$ can change if the right hand side is perturbed. The result shows that the problem is well conditioned if the condition number $\kappa(A)$ is small. Proposition 3.5 below gives a similar result for perturbation of the matrix A instead of the vector b. For the proof we will need the following lemma.

Lemma 3.4. *Assume that $A \in \mathbb{C}^{n\times n}$ satisfies $\|A\| < 1$ in some induced matrix norm. Then $I + A$ is invertible and*

$$\|(I + A)^{-1}\| \leq (1 - \|A\|)^{-1}.$$

Proof. With the triangle inequality we get

$$\begin{aligned} \|x\| &= \|(I + A)x - Ax\| \\ &\leq \|(I + A)x\| + \|-Ax\| \\ &\leq \|(I + A)x\| + \|A\|\|x\| \end{aligned}$$

and thus

$$\|(I + A)x\| \geq \big(1 - \|A\|\big)\|x\|$$

for every $x \in \mathbb{C}^n$. Since this implies $(I + A)x \neq 0$ for every $x \neq 0$, and thus the matrix $I + A$ is invertible.

Now let $b \neq 0$ and $x = (I + A)^{-1}b$. Then

$$\frac{\|(I + A)^{-1}b\|}{\|b\|} = \frac{\|x\|}{\|(I + A)x\|} \leq \frac{1}{1 - \|A\|}.$$

Since this is true for all $b \neq 0$, we have

$$\|(I+A)^{-1}\| = \sup_{b\neq 0} \frac{\|(I+A)^{-1}b\|}{\|b\|} \leq \frac{1}{1-\|A\|}.$$

This completes the proof. $\square$

Proposition 3.5 (Conditioning of SLE). *Let x solve the equations*

$$Ax = b \qquad \textit{and} \qquad (A+\Delta A)(x+\Delta x) = b.$$

Assume that A is invertible with condition number $\kappa(A)$ in some induced matrix norm $\|\cdot\|$. Then we have, for $\kappa(A)\|\Delta A\| < \|A\|$,

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1-\kappa(A)\frac{\|\Delta A\|}{\|A\|}} \cdot \frac{\|\Delta A\|}{\|A\|}.$$

Proof. Note that

$$\|A^{-1}\Delta A\| \leq \|A^{-1}\|\|\Delta A\| = \kappa(A)\frac{\|\Delta A\|}{\|A\|} < 1. \tag{3.4}$$

Here $I + A^{-1}\Delta A$ is invertible by Lemma 3.4.

We have

$$(A+\Delta A)\Delta x = -\Delta A\, x$$

and thus

$$(I + A^{-1}\Delta A)\Delta x = -A^{-1}\Delta A\, x.$$

Using Lemma 3.4 we can write

$$\Delta x = -(I + A^{-1}\Delta A)^{-1}A^{-1}\Delta A\, x$$

and we get

$$\|\Delta x\| \leq \|(I + A^{-1}\Delta A)^{-1}\|\|A^{-1}\Delta A\|\|x\| \leq \frac{\|A^{-1}\Delta A\|}{1-\|A^{-1}\Delta A\|}\|x\|.$$

Using (3.4) and the fact that the map $x \mapsto x/(1-x)$ is increasing on the interval $[0,1)$ we get

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\kappa(A)\frac{\|\Delta A\|}{\|A\|}}{1-\kappa(A)\frac{\|\Delta A\|}{\|A\|}}.$$

This is the required result. $\square$

Refer to Exercise 3-2 for a result which combines the statements of propositions 3.3 and 3.5.

3.2 Conditioning of LSQ

In this section we study the conditioning of the following problem: given $A \in \mathbb{C}^{m\times n}$ with $m \geq n$, $\operatorname{rank}(A) = n$ and $b \in \mathbb{C}^m$, find $x \in \mathbb{C}^n$ which minimizes $\|Ax - b\|_2$.

For expository purposes consider the case where A and b are real. Notice that then x solving (LSQ) minimizes $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ given by

$$\varphi(x) := \frac{1}{2}\langle Ax, Ax\rangle - \langle Ax, b\rangle + \frac{1}{2}\|b\|_2^2.$$

This is equivalent to minimizing

$$\frac{1}{2}\langle x, A^*Ax\rangle - \langle x, A^*b\rangle + \frac{1}{2}\|b\|_2^2.$$

(Note that $A^* = A^T$ in this real case). This quadratic form is positive-definite since A^*A is Hermitian and has positive eigenvalues under (3.5). The quadratic form is minimized where the gradient is zero, namely where

$$A^*Ax = A^*b.$$

Although we have derived this in the case of real A, b, the final result holds as stated in the complex case; see Chapter 7.

Consider the reduced SVD $A = \hat{U}\hat{\Sigma}V^*$ where $\hat{U} \in \mathbb{C}^{m\times n}$, $\hat{\Sigma} \in \mathbb{R}^{n\times n}$ with $\Sigma_{ii} = \sigma_i$, and $V \in \mathbb{C}^{n\times n}$. The assumption on rank$(A)$ implies, by Theorem 2.15,

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0. \tag{3.5}$$

In particular, $A^*A = V\Sigma^2V^*$ is invertible. The solution of (LSQ) is hence unique and given by solution of the *normal equations*

$$x = (A^*A)^{-1}A^*b.$$

Definition 3.6. For $A \in \mathbb{C}^{m\times n}$, the matrix $A^\dagger = (A^*A)^{-1}A^* \in \mathbb{C}^{n\times m}$ is called the *pseudo-inverse* (or *Moore-Penrose inverse*) of A.

With this notation the solution of (LSQ) is

$$x = A^\dagger b.$$

Definition 3.7. For $A \in \mathbb{C}^{m\times n}$ with $m \geq n$ and n positive singular values, define the *condition number* of A in the norm $\|\cdot\|$ to be

$$\kappa(A) = \begin{cases} \|A\| \cdot \|A^\dagger\| & \text{if rank}(A) = n \text{ and} \\ +\infty & \text{else.} \end{cases}$$

Remark. Since for square, invertible matrices A we have $A^\dagger = A^{-1}$, the definition of $\kappa(A)$ is consistent with the previous one for square matrices. As before the condition number depends on the chosen matrix norm.

Lemma 3.8. *Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$ have n positive singular values satisfying* (3.5)*. Then the condition number in the $\|\cdot\|_2$-norm is*

$$\kappa(A) = \frac{\sigma_1}{\sigma_n}.$$

Proof. Let $A = \hat{U}\hat{\Sigma}V^*$ be a reduced SVD of A. Then $A^*A = V\hat{\Sigma}^2V^*$ and thus

$$A^\dagger = V\hat{\Sigma}^{-1}\hat{U}^*. \tag{3.6}$$

This equation is a reduced SVD for $A^\dagger$ (modulo ordering of the singular values) and it can be extended to a full SVD by adding $m - n$ orthogonal columns to $\hat{U}$ and zeros to $\hat{\Sigma}^{-1}$. Doing this we find, by Corollary 2.13,

$$\kappa(A) = \|A\|_2\|A^\dagger\|_2 = \sigma_1 \cdot \frac{1}{\sigma_n}.$$

□

Notice that (3.6) implies

$$Ax = AA^\dagger b = \hat{U}\hat{U}^*b$$

and hence that

$$\|Ax\|_2 \leq \|UU^*b\|_2 = \|U^*b\|_2 = \|b\|_2$$

by the properties of orthogonal matrices. Thus we may define $\theta \in [0, \pi/2]$ by $\cos(\theta) = \frac{\|Ax\|_2}{\|b\|_2}$.

Theorem 3.9. *Assume that x solves (LSQ) for (A, b) and $x + \Delta x$ solves (LSQ) for $(A, b + \Delta b)$. Define $\eta = \|A\|_2 \|x\|_2 / \|Ax\|_2 \geq 1$. Then we have*

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \frac{\kappa(A)}{\eta \cos(\theta)} \cdot \frac{\|\Delta b\|_2}{\|b\|_2}$$

where $\kappa(A)$ is the condition number of A in $\|\cdot\|_2$-norm.

Proof. We have $x = A^\dagger b$ and $x + \Delta x = A^\dagger (b + \Delta b)$. Linearity then gives $\Delta x = A^\dagger \Delta b$ and we get

$$\begin{aligned}\frac{\|\Delta x\|_2}{\|x\|_2} &\leq \frac{\|A^\dagger\|_2 \|\Delta b\|_2}{\|x\|_2} = \frac{\kappa(A) \|\Delta b\|_2}{\|A\|_2 \|x\|_2} \\ &= \frac{\kappa(A) \|\Delta b\|_2}{\eta \|Ax\|_2} = \frac{\kappa(A)}{\eta \cos(\theta)} \cdot \frac{\|\Delta b\|_2}{\|b\|_2}.\end{aligned}$$

$\square$

Remark. The constant $\kappa(A)/\eta \cos(\theta)$ becomes large if either $\kappa(A)$ is large or $\theta \approx \pi/2$. In either of these cases the problem is badly conditioned. The first case involves only the singular values of A. The second case involves a relationship between A and b. In particular $\cos(\theta)$ is small if the range of A is nearly orthogonal to b. Then

$$\varphi(x) \approx \frac{1}{2} \|Ax\|_2^2 + \frac{1}{2} \|b\|_2^2$$

so that simply setting $x = 0$ gives a good approximation to the minimizer; in this case small changes in b can induce large relative changes in x.

Proof of the following result is left as Exercise 3-4.

Theorem 3.10. *Let θ and η as above. Assume that x solves (LSQ) for (A, b) and $x + \Delta x$ solves (LSQ) for $(A + \Delta A, b)$. Then*

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \Big(\kappa(A) + \frac{\kappa(A)^2 \tan(\theta)}{\eta}\Big) \cdot \frac{\|\Delta A\|_2}{\|A\|_2}$$

where $\kappa(A)$ is the condition number of A in $\|\cdot\|_2$-norm.

More generally, when both the matrix A and vector b are perturbed one has

Theorem 3.11 (Conditioning of LSQ)**.** *Assume that x solves (LSQ) for (A, b) and $x + \Delta x$ solves (LSQ) for $(A + \Delta A, b + \Delta b)$. Let $r = b - Ax$ and*

$$\delta = \max\Big(\frac{\|\Delta A\|_2}{\|A\|_2}, \frac{\|\Delta b\|_2}{\|b\|_2}\Big).$$

Then

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \frac{\kappa(A)\delta}{1 - \kappa(A)\delta} \Big(2 + (\kappa(A) + 1) \frac{\|r\|_2}{\cos(\theta) \eta \|b\|_2}\Big)$$

where $\kappa(A)$ is the condition number of A in $\|\cdot\|_2$-norm.

3.3 Conditioning of EVP

In this section we will discuss the conditioning of the eigenvalue problem (EVP), i.e. we will discuss how much the eigenvalues of a matrix can change, if the matrix is changed by a small amount. A preliminiary result is given in the following theorem: the eigenvalues change continuously when the matrix is changed. A more detailed analysis is presented in the rest of the section.

Theorem 3.12 (Continuity of Eigenvalues)**.** *Let $\lambda(A)$ denote the vector of eigenvalues of the matrix $A \in \mathbb{C}^{n\times n}$, ordered in decreasing absolute value, and repeated according to algebraic multiplicities. Then $\lambda\colon \mathbb{C}^{n\times n} \to \mathbb{C}^n$ is continuous.*

Proof. This follows since the eigenvalues are the roots of the characteristic polynomial of A; this has coefficients continuous in A. $\square$

Before discussing the conditioning of eigenvalue problems we make a brief diversion to state a version of the implicit function theorem (IFT) which we will then use.

Theorem 3.13 (IFT)**.** *Let $G\colon \mathbb{C}^l \times \mathbb{C}^m \to \mathbb{C}^m$ be two times differentiable and assume that $G(x, y) = 0$ for some $(x, y) \in \mathbb{C}^l \times \mathbb{C}^m$. If $D_y G(x, y)$ is invertible then, for all sufficiently small Δx, there is a unique solution Δy of the equation*

$$G(x + \Delta x, y + \Delta y) = 0$$

in a small ball at the origin. Furthermore the solution Δy satisfies

$$D_x G(x, y)\, \Delta x + D_y G(x, y)\, \Delta y = \mathcal{O}\big(\|\Delta x\|^2\big).$$

Definition 3.14. For $A \in \mathbb{C}^{n\times n}$, the *eigenvalue condition number* for an eigenvalue $\lambda \in \mathbb{C}$ of a matrix $A \in \mathbb{C}^{n\times n}$ is

$$\kappa_A(\lambda) = \begin{cases} |\langle x, y\rangle|^{-1}, & \text{if } \langle x, y\rangle \neq 0 \text{ and} \\ \infty, & \text{else,} \end{cases}$$

where x and y are the normalised right and left eigenvectors for eigenvalue λ.

Theorem 3.15 (Conditioning of EVP)**.** *Let λ be a simple eigenvalue of A corresponding to right and left normalized eigenvectors x and y. Assume $\langle y, x\rangle \neq 0$. Then for all sufficiently small $\Delta A \in \mathbb{C}^{n\times n}$ the matrix $A + \Delta A$ has an eigenvalue $\lambda + \Delta\lambda$ with*

$$\Delta\lambda = \frac{1}{\langle y, x\rangle}\Big(\langle y, \Delta A\, x\rangle + \mathcal{O}\big(\|\Delta A\|_2^2\big)\Big).$$

Proof. Define $G\colon \mathbb{C}^{n\times n} \times \mathbb{C}^n \times \mathbb{C} \to \mathbb{C}^n \times \mathbb{C}$ by

$$G(A, x, \lambda) = \begin{pmatrix} Ax - \lambda x \\ \frac{1}{2}\|x\|_2^2 - \frac{1}{2} \end{pmatrix}.$$

Thus we have $G(A, x, \lambda) = 0$ if and only if x is a normalized eigenvector of A with eigenvalue λ and clearly $G \in C^\infty$.

We will apply the IFT with $l = n^2$ and $m = n+1$ to write (x, λ) as a function of A: Provided the invertibility condition holds, the equation

$$G(A + \Delta A, x + \Delta x, \lambda + \Delta\lambda) = 0$$

has, for sufficiently small ΔA, a solution $(\Delta x, \Delta\lambda)$ satisfying

$$D_A G(A, x, \lambda)\ \Delta A + D_{(x,\lambda)} G(A, x, \lambda)\ \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = \mathcal{O}\big(\|\Delta A\|^2\big).$$

Now computing the derivatives of G gives

$$D_{(x,\lambda)} G(A, x, \lambda) = \begin{pmatrix} A - \lambda I & -x \\ x^* & 0 \end{pmatrix} \in \mathbb{C}^{(n+1)\times(n+1)}$$

and, for every $C \in \mathbb{C}^{n\times n}$,

$$D_A G(A, x, \lambda)\ C = \begin{pmatrix} Cx \\ 0 \end{pmatrix} \in \mathbb{C}^{n+1}$$

To show that $D_{(x,\lambda)}G(A,x,\lambda)$ is invertible assume $D_{(x,\lambda)}G(A,x,\lambda)\ (y,\mu)=0$ for $y\in\mathbb{C}^n$ and $\mu\in\mathbb{C}$. It suffices to show that this implies $(y,\mu)=0$. From

$$\begin{aligned}(A-\lambda I)y - x\mu &= 0\\ x^*y &= 0\end{aligned}$$

we get

$$(A-\lambda I)^2 y = 0 \quad \text{and} \quad \langle x,y\rangle = 0.$$

This implies $(A-\lambda I)$ has a two dimensional null-space, contradicting simplicity by Lemma 1.17, unless $y=0$. Hence it follows that $y=0$, $\mu=0$, and so $D_{(x,\lambda)}G$ is invertible.

Finally,

$$\begin{pmatrix}\Delta\\ \delta\end{pmatrix} = \begin{pmatrix}\Delta A\, x\\ 0\end{pmatrix} + \begin{pmatrix}A-\lambda I & -x\\ x^* & 0\end{pmatrix}\begin{pmatrix}\Delta x\\ \Delta\lambda\end{pmatrix}.$$

satisfies

$$\|\Delta\|_2 + |\delta| = \mathcal{O}\big(\|\Delta A\|_2^2\big).$$

by Theorem 3.13. Using

$$\Delta = \Delta A\, x + (A-\lambda I)\Delta x - x\Delta\lambda.$$

we find

$$-y^* x\Delta\lambda + y^*\Delta A\, x = y^*\Delta = \mathcal{O}\big(\|\Delta A\|_2^2\big)$$

and the result follows. □

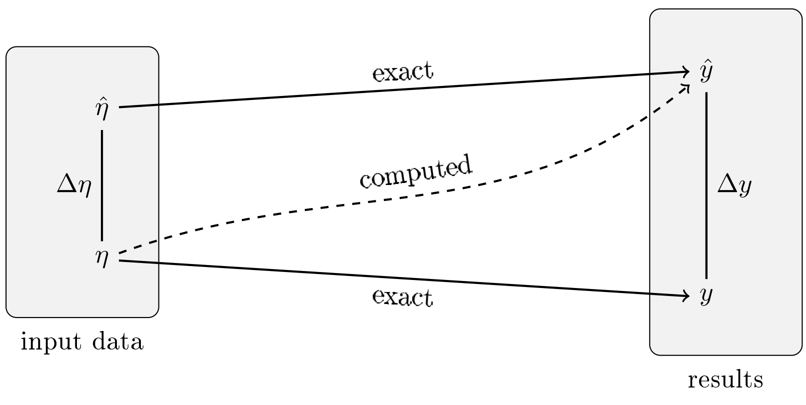
Corollary 3.16. *Let λ be a simple eigenvalue of A corresponding to right and left normalized eigenvectors x and y. Then for all sufficiently small $\Delta A\in\mathbb{C}^{n\times n}$ the matrix $A+\Delta A$ has an eigenvalue $\lambda+\Delta\lambda$ with*

$$|\Delta\lambda| \le \kappa_A(\lambda)\Big(\|\Delta A\|_2 + \mathcal{O}\big(\|\Delta A\|_2^2\big)\Big).$$

3.4 Stability of Algorithms

The stability of an algorithm measures how susceptible the result is to rounding errors occurring during the computation. Consider the general framework for all our problems, namely equation (3.1), where y denotes the solution we wish to compute and η the input data. Then we may view a locally unique family of solutions to the problem as a mapping from the input data to the solution: $y=g(\eta)$. For example, for (SLE) we have $y=x$, $\eta=(A,b)$ and $g(\eta)=A^{-1}b$. We assume that the algorithm returns the computed result $\hat{y}\neq y$ which can be represented as the exact image $\hat{y}=g(\hat{\eta})$ of a different input value $\hat{\eta}$.

Definition 3.17. The quantity $\Delta y=\hat{y}-y$ is called the *forward error* and $\Delta\eta=\hat{\eta}-\eta$ is called a *backward error*. If $\hat{\eta}$ is not uniquely defined then we make the choice of η which results in minimal $\|\Delta\eta\|$. The *relative backward error* is $\|\Delta\eta\|/\|\eta\|$ and the *relative forward error* is $\|\Delta y\|/\|y\|$.



As an illustration of the concept, imagine that we solve (SLE) and find $\hat{x}$ satisfying $A\hat{x} - b = \Delta b$. Then $\Delta x := x - \hat{x}$ is the forward error and Δb the backward error. The following theorem is a direct consequence of Proposition 3.3.

Theorem 3.18. *For the problem (SLE) we have*

$$\text{rel. forward error} \leq \kappa(A) \cdot \text{rel. backward error.}$$

Internally computers represent real number using only a finite number of bits. Thus they can only represent finitely many numbers and when dealing with general real numbers rounding errors will occur. Let $\mathbb{F} \subset \mathbb{R}$ be the set of representable numbers and let $\mathrm{fl}\colon \mathbb{R} \to \mathbb{F}$ be the rounding to the closest element of $\mathbb{F}$.

In this book we will use a simplified model for computer arithmetic which is described by the following two assumptions. The main simplification is that we ignore the problems of numbers which are unrepresentable because they are very large (overflows) or very close to zero (underflows). We simply use a model in which every bounded interval of $\mathbb{R}$ is approximated by a finite set of numbers from $\mathbb{F}$ in such a way that the following assumption holds:

Assumption 3.19. There is a parameter $\varepsilon_{\mathrm{m}} > 0$ (*machine epsilon*) such that the following conditions hold.

A1: For all $x \in \mathbb{R}$ there is an $\varepsilon \in (-\varepsilon_{\mathrm{m}}, +\varepsilon_{\mathrm{m}})$ with

$$\mathrm{fl}(x) = x \cdot (1 + \varepsilon).$$

A2: For each operation $* \in \{+, -, \cdot, /\}$ and every $x, y \in \mathbb{F}$ there is a $\delta \in (-\varepsilon_{\mathrm{m}}, +\varepsilon_{\mathrm{m}})$ with

$$x \circledast y = (x * y) \cdot (1 + \delta)$$

where $\circledast$ denotes the computed version of $*$.

Definition 3.20. An algorithm, is called *backward stable* if the relative backward error satisfies

$$\frac{\|\Delta\eta\|}{\|\eta\|} = \mathcal{O}(\varepsilon_{\mathrm{m}}).$$

Remark. Some algorithms which are backward stable by the definition are termed unstable in practice. This occurs when the constant in the $\mathcal{O}(\varepsilon_{\mathrm{m}})$ term depends badly on some measure of the problem dimension – for instance grows exponentially in n for the (SLE). For this reason, rather than just stating a backward stability result, we will often try to identify a norm $\|\cdot\|$ and constant $C(n)$ such that

$$\frac{\|\Delta\eta\|}{\|\eta\|} \leq C(n)\varepsilon_{\mathrm{m}}.$$

This, of course, establishes backward stability. But it gives more detailed information on exactly when the backward error is small, as a function of problem size. Note, however, that statements about the dependence of the constants in the backward error are norm-dependent. The definition of backward stable itself is not.

The typical way to use backward stability is in a two-step procedure as follows. In the first step, *backward error analysis*, one shows that the algorithm in question is backward stable, i.e. that the influence of rounding errors can be represented as a small perturbation Δx of the original problem. (This is a property of the algorithm used.) In the second step one uses results like Theorem 3.18 about the conditioning of the problem (which does not depend on the algorithm used) to show that the forward error is also small. Together these steps show that the calculated result is close the the exact result.

We give a simple example of a backward stability calculation.

Lemma 3.21. *The calculated subtraction $\ominus$ is backward stable.*

Proof. The exact result of a subtraction is given by $g(x_1, x_2) = x_1 - x_2$, the computed result is $\tilde{g}(x_1, x_2) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2)$. Using Assumption A1 we get

$$\mathrm{fl}(x_1) = x_1(1 + \varepsilon_1) \quad \text{and} \quad \mathrm{fl}(x_2) = x_2(1 + \varepsilon_2)$$

with $|\varepsilon_1|, |\varepsilon_2| < \varepsilon_{\mathrm{m}}$. Using Assumption A2 we get

$$\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = \bigl(\mathrm{fl}(x_1) - \mathrm{fl}(x_2)\bigr)(1 + \varepsilon_3)$$

where $|\varepsilon_3| < \varepsilon_{\mathrm{m}}$. This gives

$$\begin{aligned}\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) &= \bigl(x_1(1 + \varepsilon_1) - x_2(1 + \varepsilon_2)\bigr)(1 + \varepsilon_3) \\ &= x_1(1 + \varepsilon_1)(1 + \varepsilon_3) - x_2(1 + \varepsilon_2)(1 + \varepsilon_3) \\ &= x_1(1 + \varepsilon_4) - x_2(1 + \varepsilon_5)\end{aligned}$$

where $\varepsilon_4 = \varepsilon_1 + \varepsilon_3 + \varepsilon_1\varepsilon_3$ and $\varepsilon_5 = \varepsilon_2 + \varepsilon_3 + \varepsilon_2\varepsilon_3$ satisfy $|\varepsilon_4|, |\varepsilon_5| \leq 2\varepsilon_{\mathrm{m}} + \mathcal{O}(\varepsilon_{\mathrm{m}}^2) = \mathcal{O}(\varepsilon_{\mathrm{m}})$ for $\varepsilon_{\mathrm{m}} \to 0$.

Thus for the input error we can choose

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \hat{x} = \begin{pmatrix} x_1(1 + \varepsilon_4) \\ x_2(1 + \varepsilon_5) \end{pmatrix}, \quad \Delta x = \hat{x} - x$$

and we get $\|\Delta x\|_2 = \sqrt{\varepsilon_4^2 x_1^2 + \varepsilon_5^2 x_2^2} \leq \mathcal{O}(\varepsilon_{\mathrm{m}})\|x\|_2$. This completes the proof. □

Remarks.

1. The above proof is a case where the $\hat{x}$ from the definition of the backward error is not uniquely determined. But since we are only interested in the $\hat{x}$ which minimizes the backward error, we can choose any $\hat{x}$ which gives the result $\|\Delta x\|_2 \leq \mathcal{O}(\varepsilon_{\mathrm{m}})\|x\|_2$. The optimal $\hat{x}$ can only be better.

2. Similar proofs show that the operations $\oplus$, $\odot$ and $\oslash$ are also backward stable.

3. Proofs of backward stability have to analyse the influence of rounding errors and thus are typically based on our Assumptions A1 and A2 about computer arithmetic. Since they tend to be long and involved we omit most of these proofs in this book. We frequently study the statements of backward stability results, however, as they give useful practical insight into the efficacy of algorithms.

Bibliography

Conditioning for the basic problems of numerical linear algebra is a central part of the presentation in [Dem97]. Numerical stability is introduced, and thoroughly studied, in a range of contexts arising in numerical linear algebra, in [Hig02]. Theorem 3.10 may be found in [Dem97] and Theorem 3.11 in [Hig02]. The Implicit Function Theorem 3.13 is proved in [CH96].

Exercises

Exercise 3-1. Choose a matrix norm and calculate the condition number of the matrix

$$A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$$

in this norm.

Exercise 3-2. Assume that A is invertible with $\kappa(A)\|\Delta A\|/\|A\| < 1$ in some induced matrix norm. Let x be a solution of $Ax = b$ and let $\hat{x}$ be a solution of $(A + \Delta A)\hat{x} = b + \Delta b$. Show that

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}} \cdot \Bigl(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}\Bigr).$$

Exercise 3-3. Let $A = U\Sigma V^T$ be an SVD factorization of $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and rank$=n$. Prove that there exists a matrix ΔA with $\|\Delta A\|_2 = \sigma_{min}$ such that $A + \Delta A$ does not have rank n.

Exercise 3-4. Prove Theorem 3.10.

Exercise 3-5. Show that the calculated arithmetic operations $\oplus$, $\odot$ and $\oslash$ are backward stable.

Chapter 4

Complexity of Algorithms

In this chapter we study methods which quantify how long it takes to solve a numerical problem on a computer. Specifically we focus on the question how the run time, as measured by the number of algebraic operations, depends on some measure of the problem size.

4.1 Computational Cost

The computational cost of an algorithm is the amount of resources it takes to perform this algorithm on a computer. For simplicity here we just count the number of floating point operations (additions, subtractions, multiplications, divisions) performed during one run of the algorithm, together with a count of the number of square roots taken; on one occasion we also count the number of comparisons between real numbers. A more detailed analysis would also take factors like memory usage into account.

Definition 4.1. The *cost* of an algorithm is

$$C(n) = \text{number of additions, subtractions, multiplications, divisions and square roots}$$

where n is the size of the input data (e.g. the number of equations etc.).

The following definition provides the notation we will use to describe the asyptotic computation cost of an algorithm, this is the behaviour of the cost $C(n)$ for $n \to \infty$. (In the case of the notation $\mathcal{O}$ it is closely related to the notation used in Definition 3.1 from the last chapter, except that there we consider the order of magnitude of small quantities. Here our concern with cost leads to consideration of the order of magnitude of large quantities. However, the definitions below can be adapted to the case $x \to 0$ and we will sometimes use this without comment.)

Definition 4.2. For $f, g \colon \mathbb{N} \to \mathbb{N}$ or $f, g \colon \mathbb{R}^+ \to \mathbb{R}^+$ we write

$$\begin{aligned}
g(x) &= \mathcal{O}\big(f(x)\big) \quad \text{if} \quad \limsup_{x\to\infty} \frac{g(x)}{f(x)} < \infty, \\
g(x) &= \Omega\big(f(x)\big) \quad \text{if} \quad \liminf_{x\to\infty} \frac{g(x)}{f(x)} > 0, \\
g(x) &= \Theta\big(f(x)\big) \quad \text{if} \quad g(x) = \Omega(f(x)) \quad \text{and} \quad g(x) = \mathcal{O}(f(x)), \\
g(x) &\sim f(x) \quad \text{if} \quad \lim_{x\to\infty} \frac{g(x)}{f(x)} = 1.
\end{aligned}$$

Example. From the definition we can see that the property $g(x) \sim f(x)$ implies $g(x) = \Theta\big(f(x)\big)$.

Example. Using this notation we can write $5n^2 + 2n - 3 \sim 5n^2 = \Theta(n^2)$, $n^2 = \mathcal{O}(n^3)$ and $n^2 = \Omega(n)$.

Algorithm (standard inner product algorithm).
input: $x, y \in \mathbb{C}^n$
output: $s = \langle x, y \rangle$

$s = \bar{x}_1 y_1$
for $i = 2, \ldots, n$ **do**
    $s = s + \bar{x}_i y_i$
end for
return s

Theorem 4.3. *The standard inner-product algorithm on $\mathbb{C}^n$ has computational cost $C(n) = \Theta(n)$. Any algorithm for the inner product has $C(n) = \Omega(n)$.*

Proof. The standard inner-product algorithm above has n multiplications and $n-1$ additions, i.e. $C(n) = 2n - 1 = \Theta(n)$. Sketch of the proof for $C(n) = \Omega(n)$: since each of the products $x_i y_i$ is "independent" of the others, we have to calculate all n of them. □

Remark. Giving a real proof for the lower bound in the above theorem would require a detailed model about what an algorithm actually is. One would for example need to be able to work in a framework where "guessing" the result in just one operation and returning it is not a proper algorithm. We avoid these difficulties here by only giving "sketches" for lower bounds.

Algorithm (standard matrix-vector multiplication).
input: $A \in \mathbb{C}^{m \times n}$, $x \in \mathbb{C}^n$
output: $y = Ax \in \mathbb{C}^m$
let $a_1^*, \ldots, a_m^*$ denote the rows of A.

for $i = 1, \ldots, m$ **do**
    let $y_i = \langle a_i, x \rangle$ using the standard inner product algorithm
end for
return y

Theorem 4.4. *The computational complexity of the standard method for $\mathbb{C}^{n \times n}$-matrix-vector multiplication satisfies $C(n) = \Theta(n^2)$. Any method has $C(n) = \Omega(n)$.*

Proof. See Exercise 4-2. □

Algorithm (standard matrix-matrix multiplication).
input: $A \in \mathbb{C}^{l \times m}$ with rows $a_1^*, \ldots, a_l^*$, $B \in \mathbb{C}^{m \times n}$ with columns $b_1, \ldots, b_n$
output: $C = AB \in \mathbb{C}^{l \times n}$

for $i = 1, \ldots, l$ **do**
    for $j = 1, \ldots, n$ **do**
        let $c_{ij} = \langle a_i, b_j \rangle$ using the standard inner product algorithm
    end for
end for
return C

Theorem 4.5. *The standard method for $\mathbb{C}^{n \times n}$-matrix-matrix multiplication satisfies $C(n) = \Theta(n^3)$. Any method has $C(n) = \Omega(n^2)$.*

Proof. We have to calculate n^2 inner products. Thus the asymptotic computational cost is $C(n) = n^2 \Theta(n) = \Theta(n^3)$. Sketch for the lower bound: we have to calculate n^2 entries of the resulting matrix and thus $C(n) \geq n^2$. □

4.2 Matrix-Matrix Multiplication

In Theorem 4.5 there is a gap between the order $\Theta(n^3)$ of the standard method for multiplying matrices and the lower bound $\Omega(n^2)$. The purpose of this section is to show that there are actually algorithms with an asymptotic order which is better than $\Theta(n^3)$.

For $A, B \in \mathbb{C}^{n\times n}$, n even and $D = AB$ write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, D = \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix}$$

where $A_{ij}, B_{ij}, D_{ij} \in \mathbb{C}^{n/2\times n/2}$. Then we have

$$\begin{aligned} D_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ D_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ D_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ D_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

The above method calculates the product of two $n \times n$-matrices using eight multiplications of $(n/2) \times (n/2)$-matrices. Using this idea recursively leads to an algorithm for $n \times n$ matrix multplication which builds the answer from the multplication of small matrices. The cost then satisfies

$$C(n) = 8C(n/2) + n^2$$

with the n^2 contribution coming from the 4 additions of $(n/2) \times (n/2)$ matrices. Exercise 4-4 shows that this leads to a cost of $\Theta(n^3)$ and is thus no better than the standard algorithm.

There is, however, another way to calculate the entries of the matrix D, which is algebraically more complicated, but, crucially, only uses seven multiplications of $(n/2) \times (n/2)$-matrices. It will transpire that this fact can be utilised to get an asymptotically faster method of multiplying matrices. Using

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

we can write

$$\begin{aligned} D_{11} &= P_1 + P_4 - P_5 + P_7 \\ D_{12} &= P_3 + P_5 \\ D_{21} &= P_2 + P_4 \\ D_{22} &= P_1 + P_3 - P_2 + P_6. \end{aligned}$$

Algorithm (Strassen Multiplication).
input: $A, B \in \mathbb{C}^{n\times n}$ with $n = 2^k$ for some $k \in \mathbb{N}_0$
output: $D = AB \in \mathbb{C}^{n\times n}$

```
if n = 1 then
    return AB
else
    calculate P_1, ..., P_7 (using recursion)
    calculate D_11, D_12, D_21 and D_22
    return D
end if
```

Remark. Recursive algorithms of this kind are called *divide and conquer* algorithms.

Using the Strassen-multiplication we can calculate D with 7 multiplications of $\frac{n}{2} \times \frac{n}{2}$-matrices and 18 additions of $\frac{n}{2} \times \frac{n}{2}$-matrices. Thus we find

$$C(n) = 7C(n/2) + 18n^2/4.$$

Lemma 4.6. *The Strassen-multiplication has computational cost $C(2^k) = 7 \cdot 7^k - 6 \cdot 4^k$ for all $k \in \mathbb{N}_0$.*

Proof. For $k = 0$ we get $C(2^0) = C(1) = 1$.

Assume the claim is true for $k \in \mathbb{N}_0$. Then

$$\begin{aligned} C(2^{k+1}) &= 7C(2^k) + 18\,(2^k)^2 \\ &= 7(7 \cdot 7^k - 6 \cdot 4^k) + 18 \cdot 4^k \\ &= 7 \cdot 7^{k+1} - (7 \cdot 6 - 18)4^k \\ &= 7 \cdot 7^{k+1} - 6 \cdot 4^{k+1}. \end{aligned}$$

Now the claim follows by induction. $\square$

Theorem 4.7 (Strassen Multiplication)**.** *Using the Strassen-algorithm it is possible to construct an algorithm for matrix-matrix multiplication with asymptotic computational cost $C(n) = \Theta(n^{\log_2 7})$.*

Proof. We first prove the theorem for $n = 2^k$, $k \in \mathbb{N}_0$. Then we have

$$7^k = 2^{\log_2(7^k)} = 2^{k \log_2 7} = (2^k)^{\log_2 7} = n^{\log_2 7}$$

and

$$4^k = (2^k)^2 = n^2.$$

Using the lemma we get

$$C(n) = 7 \cdot 7^k - 6 \cdot 4^k = 7n^{\log_2 7} - 6 \cdot n^2 = \Theta(n^{\log_2 7}).$$

This finishes the proof.

If n is not a power of 2 then we argue as follows. We can extend the matrices: choose $k \in \mathbb{N}_0$ with $2^k \geq n > 2^{k-1}$ and define $\tilde{A}, \tilde{B} \in \mathbb{C}^{2^k \times 2^k}$ by

$$\tilde{A} = \begin{pmatrix} A & 0_{12} \\ 0_{21} & 0_{22} \end{pmatrix}, \qquad \tilde{B} = \begin{pmatrix} B & 0_{12} \\ 0_{21} & 0_{22} \end{pmatrix}$$

where $0_{12} \in \mathbb{C}^{n \times (2^k - n)}$, $0_{21} \in \mathbb{C}^{(2^k - n) \times n}$ and $0_{22} \in \mathbb{C}^{(2^k - n) \times (2^k - n)}$ are zero-matrices of appropriate size. The product of $\tilde{A}$ and $\tilde{B}$ may again be written in block form:

$$\tilde{A}\tilde{B} = \begin{pmatrix} AB & 0_{12} \\ 0_{21} & 0_{22} \end{pmatrix}$$

Thus we can find the product of the $n \times n$-matrices A and B by multiplying the $2^k \times 2^k$-matrices $\tilde{A}$ and $\tilde{B}$ with the Strassen-algorithm. Since we have $n \leq 2^k \leq 2n$, the extended matrices are at most double the size of the original ones, and because $(2n)^\alpha = \Theta(n^\alpha)$ for every $\alpha > 0$ the result for $n = 2^k$ implies $C(n) = \Theta(n^{\log_2 7})$ as required. $\square$

Remark. By ideas similar to (but more involved than) those used by Strassen it is possible to construct an algorithm which multiplies matrices in $\mathcal{O}(n^{2.376\ldots})$. It remains an open question whether the exponent can be made arbitrarily close to 2.

Often the first method encountered for matrix inversion is Cramer's rule:

$$A^{-1} = \frac{\operatorname{adj}(A)}{\det(A)}.$$

Using the naive way to calculate $\det(A)$ takes $\Theta(n!)$ operations, and so for Cramer's rule we would get asymptotic computational cost

$$C(n) = \Theta(n!) = \Theta(e^{-n/2} n^n).$$

Although of theoretical importance, this is a totally inefficient method to invert a matrix. The next result indicates why.

Theorem 4.8 (Complexity of Matrix Inversion). *If there is a method to multiply $n \times n$-matrices with asymptotic computational cost $\mathcal{O}(n^\alpha)$ for some $\alpha \geq 2$, then it is also possible to invert $n \times n$-matrices with cost $\mathcal{O}(n^\alpha)$.*

Proof. Let $A \in \mathbb{C}^{n\times n}$ be invertible. The proof consists of three steps.

1) As in the previous theorem we can restrict ourselves to the case $n = 2^k$ for some $k \in \mathbb{N}_0$. If n is not of this form we extend the matrix: choose $k \in \mathbb{N}_0$ with $2^k \geq n > 2^{k-1}$ and define the matrix $\tilde{A} \in \mathbb{C}^{2^k \times 2^k}$ by

$$\tilde{A} = \begin{pmatrix} A & 0_{12} \\ 0_{21} & I_{22} \end{pmatrix}$$

where $0_{12} \in \mathbb{C}^{n\times(2^k-n)}$ and $0_{21} \in \mathbb{C}^{(2^k-n)\times n}$ are zero-matrices and I_{22} is the $(2^k-n)\times(2^k-n)$-identity matrix. Then the inverse of this matrix is

$$\tilde{A}^{-1} = \begin{pmatrix} A^{-1} & 0_{12} \\ 0_{21} & I_{22} \end{pmatrix}$$

and we can invert A by inverting the $2^k \times 2^k$-matrix $\tilde{A}$. Since $\Theta\big((2n)^\alpha\big) = \Theta\big(n^\alpha\big)$ the asymptotic cost is unchanged.

2) Since A is invertible we have

$$x^*(A^*A)x = (Ax)^*Ax = \|Ax\|_2^2 > 0$$

for every $x \neq 0$ and thus A^*A is positive definite and therefore invertible. We can write

$$A^{-1} = (A^*A)^{-1}A^*.$$

This allows us to invert A with cost $C(n) = D(n) + \mathcal{O}(n^\alpha)$ where D is the cost of inverting a Hermitian, positive definite matrix and $\mathcal{O}(n^\alpha)$ is the cost for matrix-matrix multiplication. So we can restrict ourselves to the case of Hermitian, positive definite matrices.

3) To determine the cost function D let B be Hermitian and positive definite:

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{12}^* & B_{22} \end{pmatrix}$$

where the B_{jk} are $\frac{n}{2} \times \frac{n}{2}$-matrices. Let $S = B_{22} - B_{12}^* B_{11}^{-1} B_{12}$, known as the Schur complement. A direct calculation shows that

$$B^{-1} = \begin{pmatrix} B_{11}^{-1} + B_{11}^{-1}B_{12}S^{-1}B_{12}^*B_{11}^{-1} & -B_{11}^{-1}B_{12}S^{-1} \\ -S^{-1}B_{12}^*B_{11}^{-1} & S^{-1} \end{pmatrix}.$$

From exercise 4-3 we know that the matrices B_{11} and S are invertible.

This method to calculate B^{-1} needs 2 inversions of $\frac{n}{2} \times \frac{n}{2}$-matrices (namely of B_{11} and of S), a products of $\frac{n}{2} \times \frac{n}{2}$-matrices and b sums/subtractions of $\frac{n}{2} \times \frac{n}{2}$-matrices where a and b are independent of n. This shows that

$$D(n) \leq 2D(n/2) + \mathcal{O}(n^\alpha) + \mathcal{O}(n^2)$$

where $\mathcal{O}(n^\alpha)$ is the cost for the multiplications and $\mathcal{O}(n^2)$ is the cost for the additions and subtractions.

From Theorem 4.5 we already know $\alpha \geq 2$, so we can simplify the above estimate to

$$D(n) \leq 2D(n/2) + cn^\alpha$$

for some constant $c > 0$. With an induction argument (see Exercise 4-5 below) one can conclude

$$D(2^k) \leq \frac{c}{1 - 2^{1-\alpha}} (2^k)^\alpha$$

and thus we get

$$D(2^k) = O\big((2^k)^\alpha\big)$$

for $k \to \infty$. This finishes the proof. □

4.3 Fast Fourier Transform

The previous section shows that the inner-product based algorithm for matrix-matrix multiplication can be substantially improved by use of a recursive divide and conquer approach. We now attempt a similar improvement of matrix-vector multiplication, over the inner-product based algorithm. However, in contrast to the matrix-matrix case, the idea we present here works only for a restricted class of matrices — the discrete Fourier transform matrices. However other, similar, ideas have also been developed for other discrete transforms. The technique we present here is prototypical of a whole range of results available for matrix-vector multiplications of structured matrices.

As an abbreviation write

$$\omega_n = e^{2\pi i/n}$$

and define the vectors $\varphi_l \in \mathbb{C}^n$ for $l = 0, \dots, n-1$ by

$$\begin{aligned} \varphi_l &= \frac{1}{\sqrt{n}} \big((e^{-2\pi i l/n})^0, \, (e^{-2\pi i l/n})^1, \dots, \, (e^{-2\pi i l/n})^{n-1} \big) \\ &= \frac{1}{\sqrt{n}} \big(1, \, \omega_n^{-l}, \, \omega_n^{-2l}, \dots, \, \omega_n^{-(n-1)l} \big). \end{aligned}$$

The following lemma shows that these vectors form a basis of $\mathbb{C}^n$.

Lemma 4.9. $\langle \varphi_l, \varphi_m \rangle = \delta_{lm}$.

Proof.

$$\langle \varphi_l, \varphi_m \rangle = \frac{1}{n} \sum_{j=0}^{n-1} \omega_n^{(l-m)j}.$$

If $l = m$, clearly $\langle \varphi_l, \varphi_m \rangle = 1$. If $l \neq m$ the geometric sum gives

$$\frac{1}{n} \frac{\omega_n^{(l-m)n} - 1}{\omega_n^{(l-m)} - 1}.$$

But $\omega_n^{(l-m)n} = e^{2\pi i (l-m)} = 1$ implying $\langle \varphi_l, \varphi_n \rangle = 0$. □

Hence the $\{\varphi_l\}_{l=0}^{n-1}$ form an orthonormal basis for $\mathbb{C}^n$. Thus any vector $u \in \mathbb{C}^n$ can be expanded as

$$u = \sum_{l=0}^{n-1} a_l \varphi_l,$$

where $u, \varphi_l \in \mathbb{C}^n$ and the $a_l \in \mathbb{C}$.

The a_l can be found using orthogonality:

$$\langle \varphi_m, u \rangle = \sum_{l=0}^{n-1} a_l \langle \varphi_m, \varphi_l \rangle = \sum_{l=0}^{n-1} a_l \delta_{ml} = a_m.$$

Thus $a_m = \langle \varphi_m, u \rangle$.

Writing this out, with ω_n denoted by ω for simplicity, gives

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_{n-1} \end{pmatrix} = \underbrace{\frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & & & \ddots & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix}}_{:=A_n} \begin{pmatrix} \\ \\ u \\ \\ \\ \end{pmatrix}.$$

Thus to expand arbitrary u in the basis $\{\varphi_k\}$ we need to be able to perform matrix vector multiplication by $A_n \in \mathbb{C}^{n \times n}$.

Theorem 4.10 (Complexity of FFT Matrix-Vector Multiply)**.** *Let $n = 2^k$. Matrix-vector multiplication by $A_n \in \mathbb{C}^{n \times n}$ can be performed in $\mathcal{O}(n \log n)$ time.*

Proof. We let

$$F_k = \sqrt{n} A_n \quad \text{with } n = 2^k.$$

Note that $F_{k-1} \in \mathbb{C}^{n/2 \times n/2}$ and is built from $\omega_{n/2}$. We prove that multiplication by F_k can be effected in $\mathcal{O}(k2^k)$ time, which gives the desired result.

Let

$$\begin{aligned} x &= (x_0, x_1, \dots, x_{n-1})^T \in \mathbb{C}^{2^k} \\ x^{\mathrm{e}} &= (x_0, x_2, \dots, x_{n-2})^T \in \mathbb{C}^{2^{k-1}} \\ x^{\mathrm{o}} &= (x_1, x_3, \dots, x_{n-1})^T \in \mathbb{C}^{2^{k-1}}. \end{aligned}$$

Knowledge of $x^{\mathrm{o}}, x^{\mathrm{e}}$ gives x. Let

$$y = F_k x, \qquad y^* = ((y^t)^*, (y^b)^*)^*.$$

Then, for $j = 0, \dots, 2^k - 1$,

$$\begin{aligned} y_j &= \sum_{l=0}^{2^k-1} \omega_{2^k}^{jl} x_l \\ &= \sum_{l=0}^{2^{k-1}-1} \omega_{2^k}^{2jl} x_{2l} + \sum_{l=0}^{2^{k-1}-1} \omega_{2^k}^{j(2l+1)} x_{2l+1}. \end{aligned}$$

But $\omega_{2^k}^{2jl} = e^{2\pi i(2jl)/2^k} = e^{2\pi i(jl)/2^{k-1}} = \omega_{2^{k-1}}^{jl}$, and so $\omega_{2^k}^{j(2l+1)} = \omega_{2^k}^{j}\, \omega_{2^k}^{2jl} = \omega_{2^k}^{j}\, \omega_{2^{k-1}}^{jl}$. Hence,

$$y_j = \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl}\, x_l^{\mathrm{e}} + \omega_{2^k}^{j} \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl} x_l^{\mathrm{o}}.$$

Now $\omega_M^{jl} = \omega_M^{(j+M)l}$ and $\omega_{2M}^{j} = -\omega_{2M}^{j+M}$, and so

$$\begin{aligned} y_j &= \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl} x_l^{\mathrm{e}} + \omega_{2^k}^{j} \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl} x_l^{\mathrm{o}}, \quad j = 0, \dots, 2^{k-1} - 1 \\ y_{2^{k-1}+j} &= \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl} x_l^{\mathrm{e}} - \omega_{2^k}^{j} \sum_{l=0}^{2^{k-1}-1} \omega_{2^{k-1}}^{jl} x_l^{\mathrm{o}}, \quad j = 0, \dots, 2^{k-1} - 1. \end{aligned}$$

Then, in matrix notation,

$$\begin{aligned} y^t =& F_{k-1}x^{\mathrm{e}} + DF_{k-1}x^{\mathrm{o}} = y^{\mathrm{e}} + Dy^{\mathrm{o}} \\ y^b =& F_{k-1}x^{\mathrm{e}} - DF_{k-1}x^{\mathrm{o}} = y^{\mathrm{e}} - Dy^{\mathrm{o}}. \end{aligned}$$

where

$$D = \mathrm{diag}\{\omega_{2^k}^0, \omega_{2^k}^1, \ldots, \omega_{2^k}^{2^{k-1}-1}\}.$$

Thus, if $y^{\mathrm{e}} = F_{k-1}x^{\mathrm{e}}$ and $y^{\mathrm{o}} = F_{k-1}x^{\mathrm{o}}$ are known, then y is found through $\mathcal{O}(2^k)$ multiplications and additions (since D is diagonal). If $L_k = C(2^k)$, $C(n)$ being cost of multiplication by A_n, then

$$L_k \le 2L_{k-1} + a2^k.$$

For induction, assume $L_k \le (ak + L_0)2^k$. For the inductive step $l-1 \mapsto l$ assume $L_{l-1} \le (a(l-1) + L_0)2^{l-1}$, then

$$L_l \le (a(l-1) + L_0)2^l + a2^l = (al + L_0)2^l,$$

completing the proof. □

4.4 Bidiagonal and Hessenberg Forms

We conclude with the statement of a pair of matrix factorization results. Their importance is intricately bound up with the complexity of the algorithms to effect them. It is for this reason that they appear in this chapter.

Lemma 4.11. *For any $A \in \mathbb{C}^{n\times n}$ there exist unitary U, V and upper bidiagonal B such that*

$$A = UBV^*.$$

Furthermore this factorization can be achieved in $\mathcal{O}(n^3)$ operations.

This result may be proved by means of Householder reflections, which are introduced in Chapter 5. Similar techniques enable proof of the second result.

Lemma 4.12. *For any $A \in \mathbb{C}^{n\times n}$ there exists unitary Z and upper Hessenberg T such that*

$$A = ZTZ^*.$$

Furthermore this factorization can be achieved in $\mathcal{O}(n^3)$ operations.

If we remove the stipulation that the factorizations be achieved in $\mathcal{O}(n^3)$ then, by using the SVD and the Schur factorisation respectively, we may obtain the desired results; indeed B is then diagonal and T upper triangular. However, both the SVD and the Schur factorizations reveal eigenvales (of A^TA and A respectively). Hence they cannot be achieved, for arbitary n, in a finite number of arithmetic operations (see Chapter 8). The point, then, of these two lemmas, is that the factorizations can be achieved at $\mathcal{O}(n^3)$ cost. The first result may be proved by means of Householder reflections, which are introduced in Chapter 5. Similar techniques enable proof of the second result.

Bibliography

A pedagogical introduction to the study of complexity of algorithms is the book [CLRS01]. Although aimed at a wide range of problems in computer science in general, it includes a wealth of information concerning numerical algorithms. The topic of computing lower bounds, as touched upon in Theorem 4.3, is well-explained in this book. Lemmas 4.11 and 4.12 are proved in [TB97].

Exercises

Exercise 4-1. Let (i) $f(n) = n^2\big(1+\sin(n)\big)$ and (ii) $f(n) = n + n^2$. In each case, which of the following are true:

- $f(n) = \mathcal{O}(1)$;
- $f(n) = \mathcal{O}(n)$;
- $f(n) = \mathcal{O}(n^2)$;
- $f(n) = \Omega(1)$;
- $f(n) = \Omega(n)$;
- $f(n) = \Theta(n^2)$.

Exercise 4-2. Show that the standard method for matrix-vector multiplication has asymptotic computational cost $C(n) = \Theta(n^2)$.

Exercise 4-3. Show that the matrices S and B_{11} in the proof of Theorem 4.8 are invertible.

Exercise 4-4. Prove that if $C(n) = 8C(n/2) + \Theta(n^2)$ then $C(n) = \Theta(n^3)$. This shows that a divide and conquer approach to matrix multiplication which requires 8 matrix multiplications of the smaller sub-matrices cannot beat the complexity of standard matrix-matrix multiplication; in contrast, the Strassen method, using 7 multiplications, does.

Exercise 4-5. Show by induction that $D(1) = 1$ and

$$D(n) \le 2D(n/2) + cn^\alpha$$

for some constant $c > 0$ and $\alpha \ge 2$ implies

$$D(2^k) \le \frac{c}{1-2^{1-\alpha}}(2^k)^\alpha$$

for all $k \in \mathbb{N}_0$.

Exercise 4-6. Let $A \in \mathbb{R}^{n\times n}$ where $n = 2^k$. Noting that the LU factorisation of a matrix A can be written as

$$A = LU = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

design a divide and conquer strategy which results in LU factorisation of A in $\mathcal{O}(n^a)$ operations where $\mathcal{O}(n^a)$ is the cost of matrix multiplication.

Exercise 4-7. Prove Lemma 4.11.

Exercise 4-8. Prove Lemma 4.12.

Chapter 5

Systems of Linear Equations

In this chapter we analyse algorithms for (SLE): given $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$, find an $x \in \mathbb{C}^n$ such that $Ax = b$.

We present several methods to solve this problem. The common idea is to factorise the matrix A into simpler matrices M and N as $A = MN$ and to solve first the system $My = b$ and then $Nx = y$. The result is a vector x with $Ax = MNx = My = b$ and thus we have solved (SLE). In each section of the chapter we present a method, discuss its computational complexity, and study its backward error properties.

5.1 Gaussian Elimination

Gaussian elimination is the most commonly used method to solve systems of linear equations. The method is based on a systemisation of the elimination procedure used to solve simultaneous linear equations by hand. The mathematical background of the algorithm is the LU factorisation described in Theorems 2.21 and 2.22.

The Method

Under the conditions of Theorem 2.21 the matrix A can be converted into upper triangular shape by multiplying lower triangular matrices from the left. We introduce the subsequent theoretical developments with an example.

Example. Consider the matrix

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix}.$$

Then we can create zeros in the first column below the diagonal by subtracting multiples of the first row from the other rows. In matrix notation this can be written as

$$L_1 A = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix}.$$

Repeating this for the second column gives

$$L_2 L_1 A = \begin{pmatrix} 1 & & \\ & 1 & \\ & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

We have found lower triangular matrices L_1 and L_2 and an upper triangular matrix U with $A = (L_2 L_1)^{-1} U$. It turns out that $(L_2 L_1)^{-1}$ itself is unit lower triangular with inverse that is easily calculable. Hence we have found an LU factorisation of A during the process of elimination.

The ideas in this example generalise to arbitrary dimension. The following lemma helps to calculate $(L_k \cdots L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} \cdots L_k^{-1}$ in general.

Lemma 5.1. *a) Let $L = (\ell_{ij})$ be unit lower triangular with non-zero entries below the diagonal only in column k. Then L^{-1} is also unit lower triangular with non-zero entries below the diagonal only in column k and we have $(L^{-1})_{ik} = -\ell_{ik}$ for all $i > k$.*

b) Let $A = (a_{ij})$ and $B = (b_{ij})$ be unit lower triangular $n \times n$ matrices where A has non-zero entries below the diagonal only in columns $1, \ldots, k$ and B has non-zero entries below the diagonal only in columns $k+1, \ldots, n$. Then AB is unit lower triangular with $(AB)_{ij} = a_{ij}$ for $j \in \{1, \ldots, k\}$ and $(AB)_{ij} = b_{ij}$ for $j \in \{k+1, \ldots, n\}$.

Proof. a) Multiplying L with the suggested inverse gives the identity. b) Direct calculation. □

Example. For the matrices L_1 and L_2 from the previous example we get

$$(L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 1 & \\ & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix}.$$

Thus we have found the LU factorisation

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & & 2 \end{pmatrix}.$$

Recall the notation for principal submatrix from Definition 2.20. The technique to convert A into an upper triangular matrix by multiplying lower triangular matrices leads to the following algorithm:

Algorithm LU (LU factorisation).
input: $A \in \mathbb{C}^{n \times n}$ with $\det(A_j) \neq 0$ for $j = 1, \ldots, n$
output: $L, U \in \mathbb{C}^{n \times n}$ where $A = LU$ is the LU factorisation of A

1: $L = I$, $U = A$
2: **for** $k = 1, \ldots, n-1$ **do**
3:     **for** $j = k+1, \ldots, n$ **do**
4:         $l_{jk} = u_{jk}/u_{kk}$
5:         $(u_{j,k}, \ldots, u_{j,n}) = (u_{j,k}, \ldots, u_{j,n}) - l_{j,k}(u_{k,k}, \ldots, u_{k,n})$
6:     **end for**
7: **end for**

Remarks. Line 5 of the algorithm subtracts a multiple of row k from row j, causing $u_{jk} = 0$ without changing columns $1, \ldots, k-1$. This corresponds to multiplication with a lower triangular matrix L_k as in the example above. Thus after the loop ending in line 6 is finished, the current value of the matrix U is $L_k \cdots L_1 A$ and it has zeros below the diagonal in columns $1, \ldots, k$.

Since the principal sub-matrices A_j are non-singular and the matrices L_j are unit triangular we get, for $\tilde{A} = L_k \cdots L_1 A$,

$$\det \tilde{A}_{k+1} = \det A_{k+1} \neq 0$$

and thus, since $\tilde{A}$ has zeros below the diagonal in column k, we must have $u_{kk} \neq 0$ in line 4. Lemma 5.1 shows that the algorithm calculates the correct entry l_{jk} for the matrix $L = (L_n \cdots L_1)^{-1}$.

The last missing building block for the Gaussian elimination method is the following algorithm to solve systems of linear equations when the coefficient matrix is triangular.

Algorithm BS (back substitution).
input: $U \in \mathbb{C}^{n \times n}$ non-singular, upper triangular and $b \in \mathbb{C}^n$
output: $x \in \mathbb{C}^n$ with $Ux = b$

for $j = n, \ldots, 1$ **do**

$\quad x_j = \dfrac{1}{u_{jj}}\Big(b_j - \sum\limits_{k=j+1}^{n} u_{jk} x_k\Big)$

end for

Remarks.

1. Since U is triangular we get

$$(Ux)_i = \sum_{j=i}^{n} u_{ij} x_j = u_{ii}\Big(\frac{1}{u_{ii}}\big(b_i - \sum_{k=i+1}^{n} u_{ik} x_k\big)\Big) + \sum_{j=i+1}^{n} u_{ij} x_j = b_i.$$

   Thus the algorithm is correct.

2. The corresponding algorithm to solve $Lx = b$ where L is a lower triangular matrix is called *forward substitution.*

Combining all our preparations we get the Gaussian elimination algorithm to solve the problem (SLE):

Algorithm GE (Gaussian elimination).
input: $A \in \mathbb{C}^{n\times n}$ with $\det(A_j) \neq 0$ for $j = 1, \ldots, n$ and $b \in \mathbb{C}^n$
output: $x \in \mathbb{C}^n$ with $Ax = b$

find the LU factorisation of A
solve $Ly = b$ using forward substitution
solve $Ux = y$ using back substitution

Remarks.

1. The result of this algorithm is an $x \in \mathbb{C}^n$ with $Ax = LUx = Ly = b$ and thus the algorithm gives the correct result.

2. The vector y may be calculated at the same time as the LU factorisation of A, by adding b as an additional column in A.

Computational Complexity

Lemma 5.2. *The LU factorisation algorithm has computational cost*

$$C(n) = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n.$$

Proof. We have to count the number of floating point operations in the LU factorisation algorithm. Line 5 requires $(n-k+1)$ multiplications and $(n-k+1)$ subtractions, i.e. $2(n-k+1)$ operations. Line 4 contributes one division. Thus the loop starting at line 3 needs $(n-k)\big(1 + 2(n-k+1)\big)$ operations. Considering the outer loop the total number of operations is

$$C(n) = \sum_{k=1}^{n-1}(n-k)\big(1 + 2(n-k+1)\big) = \sum_{k=1}^{n-1} 2(n-k)^2 + 3(n-k) = 2\sum_{l=1}^{n-1} l^2 + 3\sum_{l=1}^{n-1} l.$$

The claim follows now by induction. □

Remark. Using the $\sim$-notation the claim of the lemma becomes $C(n) \sim \frac{2}{3}n^3$.

Lemma 5.3. *Forward substitution and back substitution have computational cost* $C(n) \sim n^2$.

Proof. Calculating x_j in the back substitution algorithm needs $2(n-j)+1$ operations. Thus the total cost is

$$C(n) = \sum_{j=1}^{n}\big(2(n-j)+1\big) = 2\sum_{k=0}^{n-1} k + n = n(n-1) + n = n^2.$$

A similar argument applies to the situation of forward substitution. $\square$

Theorem 5.4 (Computational Complexity of Gaussian Elimination)**.** *The asymptotic computational cost of the GE algorithm is of order*

$$C(n) \sim \frac{2}{3}n^3.$$

Proof. This is an immediate consequence of the previous two lemmas. $\square$

Error Analysis

The following theorem is proved by using the Assumptions (A1) and (A2) concerning computer arithmetic. The proof is somewhat involved and we omit it here. We include the statement, however, because it is an important building block in the overall understanding of backward error analysis for Gaussian elimination. We use the notation $|\cdot|$ for matrices, introduced just prior to Lemma 1.34.

Theorem 5.5. *The forward and back substitution algorithms are backward stable: the computed solution $\hat{x}$ of a triangular system of equations $Tx = b$ satisfies $(T+\Delta T)\hat{x} = b$ for some triangular matrix $\Delta T \in \mathbb{C}^{n\times n}$ with, for ε_{m} sufficiently small,*

$$|\Delta T| \le \frac{n\varepsilon_{\mathrm{m}}}{1-n\varepsilon_{\mathrm{m}}}|T|.$$

Note that this theorem implies that ΔT has the same triangular structure as T itself. The result shows that, for $n\varepsilon_{\mathrm{m}}$ sufficiently small, there is a constant c independent of n such that, by Lemma 1.34,

$$\|\Delta T\|_\infty \le cn\varepsilon_{\mathrm{m}}\|T\|_\infty.$$

Applying this result to backsubstitution we find that the computed solution of $Ux = y$, $\hat{x}$, solves $(U+\Delta U)\hat{x} = y$ and that

$$\|\Delta U\|_\infty \le cn\varepsilon_{\mathrm{m}}\|U\|_\infty.$$

Using Proposition 3.5 about the conditioning of (SLE) we get an upper bound on the error in the computed result of back substitution:

$$\frac{\|\hat{x}-x\|_\infty}{\|x\|_\infty} \le c\kappa(U)n\varepsilon_{\mathrm{m}}$$

for $\kappa(U)n\varepsilon_{\mathrm{m}}$ sufficiently small.

Thus the backward substitution step of Gaussian elimination is numerically stable and does not introduces large errors, provided $\kappa(U)n\varepsilon_{\mathrm{m}}$ remains small. The same holds for forward substitution. The LU factorisation algorithm, however, is not backward stable. To explain this fact we describe two more results from backward error analysis.

Theorem 5.6. *The computed LU-factors in a Gaussian elimination of $A \in \mathbb{R}^{n\times n}$, $\hat{L}$ and $\hat{U}$, satisfy*

$$\hat{L}\hat{U} = A + \Delta A$$

where, for ε_{m} sufficiently small,

$$|\Delta A| \le \frac{n\varepsilon_{\mathrm{m}}}{1-n\varepsilon_{\mathrm{m}}}|\hat{L}||\hat{U}|.$$

The proof of this result is similar to that of the previous theorem, and we omit it too. Combining the two previous theorems gives the following backward error analysis result for the solution of (SLE) by Gaussian elimnation.

Theorem 5.7. *Let $\hat{x}$ be the approximate solution of (SLE) as computed by algorithm GE. Then*

$$(A + \Delta A)\hat{x} = b$$

where, for ε_{m} sufficiently small,

$$|\Delta A| \leq \frac{3n\varepsilon_{\mathrm{m}}}{1 - 3n\varepsilon_{\mathrm{m}}}|\hat{L}||\hat{U}|.$$

Proof. See Exercise 5-10. □

This would give a backward stability result if it were possible to bound $\||\hat{L}||\hat{U}|\|$ in terms of $\|A\|$. Unfortunately this is not possible as we illustrate in the following example.

Example. [Effect of Rounding] Let

$$A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$$

for some $\varepsilon > 0$. Then A has a LU factorisation $A = LU$ with

$$L = \begin{pmatrix} 1 & 0 \\ \varepsilon^{-1} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} \varepsilon & 1 \\ 0 & 1 - \varepsilon^{-1} \end{pmatrix}.$$

Now assume $\varepsilon \ll 1$. Then ε^{-1} is a huge number and the representation of these matrices stored in a computer will be rounded. The matrices might be represented as

$$\hat{L} = \begin{pmatrix} 1 & 0 \\ \varepsilon^{-1} & 1 \end{pmatrix}, \qquad \hat{U} = \begin{pmatrix} \varepsilon & 1 \\ 0 & -\varepsilon^{-1} \end{pmatrix},$$

which is compatible with Assumption (A1) on rounding errors. We have $\hat{L} = L$ and $\hat{U} \approx U$. But multiplying the two rounded matrices gives

$$\hat{L}\hat{U} = \begin{pmatrix} \varepsilon & 1 \\ 1 & 0 \end{pmatrix} = A + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix}}_{\Delta A}.$$

A small rounding error leads to a large backward error ΔA. The example shows that for Gaussian elimination a backward error analysis will, in general, lead to the conclusion that the perturbed problem is not close to the original one: the algorithm is not backward stable.

Discussing the result in terms of the previous theorem, note that $\|A\|_\infty = 2$ and that $\||\hat{L}||\hat{U}|\|_\infty = \Theta(\varepsilon^{-1})$. Hence it is not possible to bound $\||\hat{L}||\hat{U}|\|$ in terms of $\|A\|$, independently of ε.

Note that this problem is not related to the conditioning of the matrix A. We have

$$A^{-1} = (1 - \varepsilon)^{-1} \begin{pmatrix} -1 & 1 \\ 1 & -\varepsilon \end{pmatrix}$$

and thus $\kappa(A) = \|A\|_\infty \|A^{-1}\|_\infty \approx 4$ for small $\varepsilon > 0$ and the matrix A is well conditioned.

5.2 Gaussian Elimination with Partial Pivoting

Because of the instability illustrated in the previous example, the classical Gaussian elimination method is not used in black box numerical software for (SLE). However, the continuation of the example illustrates the resolution: permutations can cure this problem.

Example. [Effect of Rounding Continued] Let

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{and} \quad A' = PA = \begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix},$$

then

$$L' = \begin{pmatrix} 1 & 0 \\ \varepsilon & 1 \end{pmatrix} \qquad U' = \begin{pmatrix} 1 & 1 \\ 0 & 1-\varepsilon \end{pmatrix}$$
$$\hat{L}' = \begin{pmatrix} 1 & 0 \\ \varepsilon & 1 \end{pmatrix} \qquad \hat{U}' = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Thus

$$\hat{L}'\hat{U}' = \begin{pmatrix} 1 & 1 \\ \varepsilon & 1+\varepsilon \end{pmatrix} = A' + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & \varepsilon \end{pmatrix}}_{\Delta A'},$$

and the backward error $\Delta A'$ is small.

Thus permuting is important. There are two primary methods of permutation:

GEPP Swap rows only to maximise $|a_{ll}^{(l)}|$ over all choices from $l, \dots, n$: $(n-l+1)$ of them.

GECP Swap rows and columns to maximise $|a_{ll}^{(l)}|$ over all choices from $l, \dots, n$: $(n-l+1)^2$ of them.

Here GEPP denotes *Gaussian Elimination with partial pivoting* whilst GECP denotes *Gaussian Elimination with complete pivoting*. GEPP is the most widely used in practice and so we devote most of our analysis to this algorithm.

The Method

The instability of Gaussian elimination highlighted above is caused by the fact that we divide by the tiny number $u_{kk} = \varepsilon$ in step 4 of the LU factorisation algorithm. Motivated by the resolution in the example above, we will avoid this problem in the improved version of the algorithm by rearranging rows $k, \dots, n$ by multiplication with a permutation matrix at the beginning of the kth iteration in order to maximise the element u_{kk}. The following argument shows that the modified algorithm still works correctly.

We want to calculate
$$U = L_{n-1}P_{n-1} \cdots L_1 P_1 A.$$

Multiplying the permutation matrix P_k from the left exchanges rows k and i_k where i_k is chosen from $\{k, \dots, n\}$ to maximise the element $u_{i_k k}$. We can rewrite this as

$$U = L'_{n-1} \cdots L'_1 \, P_{n-1} \cdots P_1 \, A$$

where $L'_{n-1} = L_{n-1}$ and
$$L'_k = P_{n-1} \cdots P_{k+1} L_k P_{k+1}^{-1} \cdots P_{n-1}^{-1}$$

for $k = 1, \dots, n-2$. Since $P_{n-1} \cdots P_{k+1}$ exchanges rows $k+1, \dots, n$ and $P_{k+1}^{-1} \cdots P_{n-1}^{-1}$ performs the corresponding permutation on the columns $k+1, \dots, n$ the shape of L'_k is the same as the shape of L_k: it is unit lower triangular and the only non-vanishing entries below the diagonal are in column k. Hence we can still use Lemma 5.1 to calculate $L = (L'_{n-1} \cdots L'_1)^{-1}$. The above arguments lead to the following algorithm.

Algorithm LUPP (LU factorisation with partial pivoting).
input: $A \in \mathbb{C}^{n \times n}$ non-singular
output: $L, U, P \in \mathbb{C}^{n \times n}$ where $PA = LU$ with L unit lower triangular,
U non-singular upper triangular and P a permutation matrix

1: $U = A$, $L = I$, $P = I$
2: **for** $k = 1, \ldots, n-1$ **do**
3: choose $i \in \{k, \ldots, n\}$ which maximises $|u_{ik}|$
4: exchange row $(u_{k,k}, \ldots, u_{k,n})$ with $(u_{i,k}, \ldots, u_{i,n})$
5: exchange row $(l_{k,1}, \ldots, l_{k,k-1})$ with $(l_{i,1}, \ldots, l_{i,k-1})$
6: exchange row $(p_{k,1}, \ldots, p_{k,n})$ with $(p_{i,1}, \ldots, p_{i,n})$
7: **for** $j = k+1, \ldots, n$ **do**
8: $l_{jk} = u_{jk}/u_{kk}$
9: $(u_{j,k}, \ldots, u_{j,n}) = (u_{j,k}, \ldots, u_{j,n}) - l_{j,k}(u_{k,k}, \ldots, u_{k,n})$
10: **end for**
11: **end for**

Remark. One can show that, whenever A is non-singular, the element u_{kk} is always non-zero after the row-exchange. Thus we do no longer require the condition $\det(A_j) \neq 0$ to avoid a division by zero here.

Lemma 5.8. *The resulting matrices L and L_k satisfy $\|L\|_\infty \leq n$ and $\|L_k\|_\infty \leq 2$.*

Proof. Each matrix L_k has entries $|(L_k)_{ij}| \leq 1$ for all $i, j \in \{1, \ldots, n\}$. Consequently the same is true of L. The result for $\|L\|_\infty$ follows directly. Furthermore, as each row of L_k contains at most two non-zero entries, we have $\|L_k\|_\infty \leq 2$. □

Gaussian elimination with partial pivoting now works as follows:

Algorithm GEPP (Gaussian elimination with partial pivoting).
input: $A \in \mathbb{C}^{n\times n}$ non-singular, $b \in \mathbb{C}^n$
output: $x \in \mathbb{C}^n$ with $Ax = b$
find $PA = LU$ using algorithm LUPP
solve $Ly = Pb$ using forward substitution
solve $Ux = y$ using back substitution

Remarks.

1. The result of this algorithm is an $x \in \mathbb{C}^n$ with $Ax = P^{-1}LUx = P^{-1}Ly = P^{-1}Pb = b$ and thus the algorithm is correct.

2. Again it is possible to compute y by appending b as an extra column in the matrix A.

Computational Complexity

The computational complexity is the same as for the LU algorithm given in Theorem 5.4. This follows from the fact that we only added steps to exchange rows in the LU algorithm and such exchanges do not incur computational cost in our model. If we assign unit cost to the comparisons between number, made to maximise $|u_{ik}|$ in step 3 of the algorithm, then this added additional cost is of order $\Theta(n^2)$; thus it can be neglected for a $\Theta(n^3)$ algorithm. For GECP the additional cost is of order $\Theta(n^3)$ and hence cannot be neglected.

Error Analysis

To derive a backward error analysis for GEPP is nontrivial. We limit ourselves to an heuristic understanding of the analysis, based on the following two assumptions:

- The computed permutation matrix $\hat{P}$ is the correct permutation P;

- The computed L, U factors $\hat{L}$, $\hat{U}$ are well approximated by L, U.

Then applying Theorem 5.7 to $PA = \hat{P}A$ we obtain a computed solution $\hat{x}$ which satisfies

$$P(A + \Delta A)\hat{x} = Pb$$

where

$$\|\Delta A\|_\infty \leq \frac{3n\varepsilon_{\mathrm{m}}}{1 - 3n\varepsilon_{\mathrm{m}}} \|\hat{L}\|_\infty \|\hat{U}\|_\infty.$$

To bound the right-hand side of this expression we use the following definition:

Definition 5.9. Define $g_n(A)$, the *growth factor* for matrix $A \in \mathbb{C}^{n \times n}$, by

$$g_n(A) = \frac{\|U\|_{\max}}{\|A\|_{\max}}. \tag{5.1}$$

From this it follows, by Lemma 1.35, that

$$\|U\|_\infty \leq n g_n(A) \|A\|_\infty.$$

We also have $\|L\|_\infty \leq n$ and hence, using the approximation $\hat{L} \approx L, \hat{U} \approx U$,

$$\|\Delta A\|_\infty \lesssim \frac{3n^3 g_n(A)\varepsilon_{\mathrm{m}}}{1 - 3n\varepsilon_{\mathrm{m}}} \|A\|_\infty. \tag{5.2}$$

(Here $\lesssim$ indicates that approximations have been made in deriving the inequality). To understand the implications of this bound we study the behaviour of the growth factor in further detail.

Lemma 5.10. *The growth factor $g_n(A)$ satisfies $g_n(A) \leq 2^{n-1}$ for every $A \in \mathbb{C}^{n \times n}$.*

Proof. We have $U = L'_{n-1} \cdots L'_1 PA$. For any $k \in \{1, \ldots, n-1\}$ and any matrix $A \in \mathbb{C}^{n \times n}$ we find

$$\begin{aligned}
\|L'_k A\|_{\max} &= \max_{i,j} \big|(L'_k A)_{ij}\big| \\
&= \max_{i,j=1,\ldots,n} \Big|\sum_{l=1}^{n} (L'_k)_{il} a_{lj}\Big| \\
&\leq \max_{i=1,\ldots,n} \Big|\sum_{l=1}^{n} (L'_k)_{il}\Big| \max_{i,j} |a_{ij}| \\
&= \|L'_k\|_\infty \|A\|_{\max} \\
&\leq 2\|A\|_{\max}
\end{aligned}$$

and thus by applying the matrices $L'_1, \ldots L'_{n-1}$ in order we get, since $\|PA\|_{\max} = \|A\|_{\max}$, $\|U\|_{\max} \leq 2^{n-1} \max_{i,j} \|A\|_{\max}$. This completes the proof. ☐

Placing this upper-bound on $g_n(A)$ in (5.2) shows a worryingly large n-dependence in the backward error bound for GEPP:

$$\|\Delta A\|_\infty \lesssim \frac{3n^3 2^{n-1} \varepsilon_{\mathrm{m}}}{1 - 3n\varepsilon_{\mathrm{m}}} \|A\|_\infty.$$

Furthermore, the following example shows that the upper bound on $g_n(A)$ from the lemma can be attained.

Example. Let

$$A = \begin{pmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ -1 & -1 & \cdots & -1 & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Since all relevant elements have modulus 1 we do not need to use pivoting and LU factorisation gives

$$U = \begin{pmatrix} 1 & & & & & 1 \\ & 1 & & & & 2 \\ & & 1 & & & 4 \\ & & & \ddots & & \vdots \\ & & & & 1 & 2^{n-2} \\ & & & & & 2^{n-1} \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Thus for the matrix A as defined above we get

$$g_n(A) = \frac{\|U\|_{\max}}{\|A\|_{\max}} = 2^{n-1}.$$

Despite this worst case analysis, GEPP is widely used in practice; we outline why this is the case.

Remarks.

1. The implied backward error analysis for GEPP given by (5.2) is indeed very weak: although the algorithm is technically backward stable, for certain problems it is necessary that $2^n\varepsilon_{\mathrm{m}}$ be small to ensure a small backward error.
2. The preceding analysis may also be carried out for the GECP algorithm and results in a bound for the growth factor which scales like $n^{\frac{1}{4}\log n+\frac{1}{2}}$; whilst still faster than polynomial in n, this bound is considerably slower than 2^n. Thus GECP has, in a worst case analysis, far better stability properties than GEPP.
3. If comparisons between numbers are assigned unit cost, then GECP is still an $\Theta(n^3)$ algorithm, but the constant is larger than for GEPP.
4. Practical experience, and some statistical analysis, has led to a belief that matrices resulting in large backward error for GEPP do not arise commonly in practice. Thus the worst case analysis is thought, typically, to be misleading.
5. For these reasons GEPP is commonly used in practice, because it is cheaper than GECP if comparisons are assigned a unit computational cost, and because it appears to produce reasonable backward error on most problems.

5.3 The QR Factorisation

The Householder QR factorisation is another method to solve (SLE). QR factorisation avoids the poor dependence of the stability constant on n that arises for the LU factorisation with partial pivoting. However the computation takes roughly twice the number of operations.

The Method

The following algorithm solves the problem (SLE) using the QR factorisation from Theorem 2.18. In order to apply the algorithm we will need a numerically stable method to calculate the QR factorisation. To avoid a minor technical complication, we restrict ourselves to the case of real-valued matrices in this section.

Algorithm (solving (SLE) by QR factorisation).
input: $A \in \mathbb{R}^{n\times n}$ non-singular, $b \in \mathbb{R}^n$
output: $x \in \mathbb{R}^n$ with $Ax = b$

1: find the QR factorisation $A = QR$
2: calculate $y = Q^*b$
3: solve $Rx = y$ using back substitution

The result of this algorithm is an $x \in \mathbb{R}^n$ with $Ax = QRx = Qy = QQ^*b = b$ and thus the algorithm is correct. To calculate the QR factorisation we will use the following algorithm. We present the full algorithm first and then analyse it to see how it works. The algorithm uses the sign function, which is defined as follows.

$$\operatorname{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0, \text{ and} \\ -1 & \text{else.} \end{cases}$$

Algorithm QR (Householder QR factorisation).
input: $A \in \mathbb{R}^{m \times n}$ with $m \geq n$
output: $Q \in \mathbb{R}^{m \times m}$ orthogonal, $R \in \mathbb{R}^{m \times n}$ upper triangular with $A = QR$

1: $Q = I$, $R = A$
2: **for** $k = 1, \ldots, n$ **do**
3: $\quad u = (r_{kk}, \ldots, r_{mk}) \in \mathbb{R}^{m-k+1}$
4: $\quad \bar{v} = \operatorname{sign}(u_1)\|u\|_2 e_1 + u$ where $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^{m-k+1}$
5: $\quad v = \bar{v}/\|\bar{v}\|_2$
6: $\quad H_k = (I_{m-k+1} - 2vv^*) \in \mathbb{R}^{(m-k+1)\times(m-k+1)}$
7: $\quad Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & H_k \end{pmatrix}$
8: $\quad R = Q_k \cdot R$
9: $\quad Q = Q \cdot Q_k$
10: **end for**

Remarks.

1. This method of calculating a QR factorisation has very desirable numerical stability properties.

2. The algorithm calculates matrices Q_k with $Q_k = Q_k^*$ for $k = 1, \ldots, n-1$ as well as $R = Q_{n-1} \cdots Q_1 A$ and $Q = Q_1 \cdots Q_{n-1}$.

3. We will see that $Q_k \cdots Q_1 A$ has zeros below the diagonal in columns $1, \ldots, k$ and thus the final result $R = Q_{n-1} \cdots Q_1 A$ is upper triangular.

4. The only use we make of the matrix Q when solving (SLE) by QR factorisation is to calculate Q^*b. Thus for solving (SLE) we can omit the explicit calculation of Q by replacing line 9 of algorithm QR with the statement $b = Q_k b$. The final result in the variable b will then be
$$Q_{n-1} \cdots Q_1 b = (Q_1 \cdots Q_{n-1})^* b = Q^* b.$$

Householder Reflections

In step 8 of algorithm QR we calculate a product of the form

$$Q_k \cdot \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & H_k R_{22} \end{pmatrix} \tag{5.3}$$

where $R_{11} \in \mathbb{R}^{(k-1)\times(k-1)}$ and $H_k, R_{22} \in \mathbb{R}^{(m-k+1)\times(m-k+1)}$. The purpose of this subsection is to understand this step of the algorithm.

If H_k as calculated in step 6 of algorithm QR is applied to a vector $x \in \mathbb{R}^{m-k+1}$ the result is

$$H_k x = x - 2vv^*x = x - 2v\langle v, x\rangle.$$

Since the vector $v\langle v, x\rangle$ is the projection of x onto v, the value $x - v\langle v, x\rangle$ is the projection of x onto the plane which is orthogonal to v and $x - 2v\langle v, x\rangle$ is the reflection of x at that plane. Reflecting twice at the same plane gives back the original vector and thus we find

$$H_k^* H_k = H_k H_k = I.$$

This shows that the matrices H_k, and then also Q_k, are orthogonal for every $k \in \{1, \ldots, n-1\}$.

The vector which defines the reflection plane is either $\bar{v} = u - \|u\|_2 e_1$ or $\bar{v} = u - (-\|u\|_2 e_1)$, depending on the sign of u_1. The corresponding reflection maps the vector u to $H_k u = \|u\|_2 e_1$ or $H_k u = -\|u\|_2 e_1$ respectively. In either case the image is a multiple of e_1 and since u is the first column of the matrix block R_{22} the product $H_k R_{22}$ has zeros below the diagonal in the first column. The first column of R_{22} is the kth column of R and thus $Q_k R$ has zeros below the diagonal in columns $1, \ldots, k$. For $k = n-1$ we find that $R = Q_{n-1} \cdots Q_1 A$ is an upper triangular matrix as required.

Remarks.

1. The matrices H_k and sometimes also Q_k are called *Householder reflections.*
2. The choice of sign in the definition of u helps to increase the stability of the algorithm in the cases $u \approx \|u\|_2 e_1$ and $u \approx -\|u\|_2 e_1$. In the complex case, $\operatorname{sign}(x)$ will be replaced by a complex number $\sigma \in \mathbb{C}$ with $|\sigma| = 1$.

Computational Complexity

We considered two variants of algorithm QR, either calculating the full matrix Q as formulated in line 9 of the algorithm or only calculating $Q^* b$ by replacing line 9 with the statement $b = Q_k b$. We handle the different cases by first analysing the operation count for the algorithm with line 9 omitted.

Lemma 5.11. *The computational cost $C(m,n)$ for algorithm QR applied to an $m \times n$-matrix without calculating Q or $Q^* b$ is asymptotically*

$$C(m,n) \sim 2mn^2 - \frac{2}{3}n^3 \quad \text{for } m, n \to \infty \text{ with } m = \Theta(n).$$

Proof. We count the number of operations for the individual steps of algorithm QR. From equation (5.3) we can see that for calculating the product $Q_k R$ in step 8 we only have to calculate $H_k R_{22} = R_{22} - 2vv^* R_{22}$. Since $v = \bar{v}/\|\bar{v}\|_2$ and $\|\bar{v}\|_2 = \sqrt{\bar{v}^* \bar{v}}$ we can calculate this as

$$H_k R_{22} = R_{22} - \frac{\bar{v}}{\bar{v}^* \bar{v}/2} \bar{v}^* R_{22}.$$

Using this formula we get the following operations count:

- construction of $\bar{v}$: $2(m-k+1)+1$ operations (counting $\sqrt{\cdot}$ as 1).
- computing $\bar{v}^* R_{22}$: Since each of the $n-k+1$ components of the matrix-vector product $\bar{v}^* R_{22}$ needs $m-k+1$ multiplications and $m-k$ additions, the computation of $\bar{v}^* R_{22}$ requires $(n-k+1)\big((m-k+1)+(m-k)\big)$ operations.
- Calculating $\bar{v}^* \bar{v}/2$ needs $2(m-k+1)$ operations and dividing $\bar{v}$ by the result requires another $(m-k+1)$ divisions.
- Calculating $(\cdots)(\bar{v}^* R_{22})$ from this needs $(m-k+1)(n-k+1)$ multiplications.
- Calculating $R_{22} - (\cdots)$ requires $(m-k+1)(n-k+1)$ subtractions.

Thus the total operation count is

$$\begin{aligned}
C(m,n) &= \sum_{k=1}^{n} 5(m-k+1)+1 \;+\; (n-k+1)\big(4(m-k+1)-1\big) \\
&= \sum_{l=m-n+1}^{m} 5l+1 \;+\; (n-m+l)(4l-1) \\
&= 2mn^2 - \frac{2}{3}n^3 + \text{terms with at most two factors } m, n \\
&\sim 2mn^2 - \frac{2}{3}n^3
\end{aligned}$$

for $m, n \to \infty$ with $m = \Theta(n)$. $\square$

If we need to calculate the full matrix Q we have to perform an $(m-k+1) \times (m-k+1)$ matrix-matrix multiplication in step 9. Assuming that we use the standard matrix multiplication algorithm this contributes asymptotic cost $\Theta(m^3)$ and so the asymptotic cost of algorithm QR will be increased by this step. But if we apply algorithm QR only to solve (SLE), we just have to calculate Q^*b instead of Q. Algorithmically this is the same as appending the vector b as an additional column to the matrix A. Thus the computational cost for this algorithm is $C(m, n+1)$ and since

$$2m(n+1)^2 - \frac{2}{3}(n+1)^3 \sim 2mn^2 - \frac{2}{3}n^3$$

for $m, n \to \infty$ with $m = \Theta(n)$ the asymptotic cost does not change. For solving (SLE) we also have $m = n$ and thus we find that the asymptotic computational cost of solving (SLE) using Householder QR factorisation is

$$C(n) \sim 2nn^2 - \frac{2}{3}n^3 = \frac{4}{3}n^3 \qquad \text{for } n \to \infty.$$

This analysis shows that solving (SLE) using Householder QR factorisation requires asymptotically double the number of steps than algorithm GEPP does. This is the price we have to pay for the better stability properties of the QR algorithm.

Error Analysis

The backward error analysis for Householder QR is best expressed in terms of the norms of matrix columns. For any matrix $D \in \mathbb{R}^{n \times n}$ we write d_j to denote the j^{th} column of D. A similar convention for upper and lower case letters is used in what follows.

The first result concerns the form of the triangular factor in the Householder QR factorisation.

Theorem 5.12. *Let $\hat{R}$ denote the computed upper triangular factor in a Householder QR factorisation of $A \in \mathbb{R}^{n \times n}$. Then there exists orthogonal $Q \in \mathbb{R}^{n \times n}$ and constant $c \in \mathbb{R}^+$ such that*

$$A + \Delta A = Q\hat{R}$$

where, for ε_{m} sufficiently small,

$$\|\Delta a_j\|_2 \leq \frac{cn^2\varepsilon_{\mathrm{m}}}{1 - cn^2\varepsilon_{\mathrm{m}}}\|a_j\|_2, j = 1, \ldots, n.$$

This result can be used to show the following.

Theorem 5.13. *Let x solve $Ax = b$ and let $\hat{x}$ be the solution computed through Householder QR. Then*

$$(A + \Delta A)\hat{x} = b + \Delta b,$$

where, for some constant $c \in \mathbb{R}^+$ and for ε_{m} sufficiently small,

$$\|\Delta a_j\|_2 \leq \frac{cn^2\varepsilon_{\mathrm{m}}}{1 - cn^2\varepsilon_{\mathrm{m}}}\|a_j\|_2,$$

$$\|\Delta b\|_2 \leq \frac{cn^2\varepsilon_{\mathrm{m}}}{1 - cn^2\varepsilon_{\mathrm{m}}}\|b\|_2.$$

This is a considerable improvement over the worst case analysis for GEPP. However, as we have seen, the worst case analysis does not give a true picture of the stability of GEPP in practice; furthermore, Householder QR costs twice as much as GEPP. Consequently GEPP is favoured in practice.

Bibliography

The book [Hig02] is an excellent source of material concerning backward error analysis for numerical algorithms. In particular, proofs of Theorems 5.5, 5.6, 5.12 and 5.13 may all be found in this book. The fact that the worst case analysis of GEPP does not seem to arise often in practice is discussed in [TB97].

Exercises

Exercise 5-1. The purpose of this question is to illustrate some of the content of the chapter by means of some Matlab experiments. We consider the linear system

$$Ax = b \tag{5.4}$$

where $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$ are given and $x \in \mathbb{R}^n$ is to be found. In MATLAB we start by creating a random matrix A and vector b. Type

```
>> A=randn(5)
>> b=randn(5,1)
```

These two commands will produce a random 5×5 matrix and 5-vector where each entry is independent of all others and is distributed normally with mean 0 and variance 1. Now type

```
>> x=A\b
```

This x is the (computer approximation) to the solution of (5.4). To verify this type

```
>> r=A*x-b; rn=norm(r)
```

which will produce the norm of the *residual* r between Ax (with the computed x) and b. How big is `rn`? What does this tell you?

Now we try and explain how MATLAB solved (5.4). Type

```
>> [L,U,P]=lu(A)
```

and describe what special properties L, U and P have. In particular, what effect does P have when applied to an arbitrary 5×5 matrix? Why is $P^{-1} = P^T$? If you type

```
>> P*A
```

and then

```
>> L*U
```

you see that $PA = LU$ (approximately – use `norm` to measure the residual). Thus, applying P to (5.4), we see that x solves

$$LUx = Pb.$$

Thus

$$x = U^{-1}L^{-1}Pb.$$

What special features do U and L have which make this a good way to find x? (Think about what is involved in solving an arbitrary linear system and compare it to what is involved if the matrix to be inverted has the form of U or L).

The LU factorisation described above is what MATLAB uses to solve (5.4) (unless A is symmetric with positive diagonal entries, in which case *Cholesky factorisation* is used). But an alternative is to use the QR factorisation which we now outline. Type

```
>> [Q,R]=qr(A)
```

and then type

```
>> rn=norm(Q*R-A)
```

noting that this shows that QR is a (approximate) factorisation of A. R has a property you have seen on another matrix in this question (what is it?) and is hence easy to invert. What about Q? Let us extract the columns from Q. Type

```
>> q1=Q(1:5,1); q2=Q(1:5,2); q3=Q(1:5,3); q4=Q(1:5,4); q5=Q(1:5,5)
```

What do these vectors q represent? Now, using the Matlab command `dot(a,b)`, which calculates dot products of vectors, experiment with the properties of inner-products amongst the q vectors. What do you find? Why is Q easy to invert?

Using this QR factorisation it is possible to find x solving (5.4) as $x = R^{-1}Q^{-1}b$. Although this method is not often used it has some nice mathematical properties which we will explain in the lectures.

Exercise 5-2. In analogy to the back substitution algorithm formulate the forward substitution algorithm to solve $Ly = b$ where $L \in \mathbb{C}^{n\times n}$ is a lower triangular, invertible matrix and $b \in \mathbb{C}^n$ is a vector. Show that your algorithm computes the correct result and that it has asymptotic computational cost of order $\Theta(n^2)$.

Exercise 5-3. a) Find the LU factorisation of

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}.$$

What is the growth factor $g_n(A)$ in this case? b) Find a QR factorisation of the same matrix, using both Gram-Schmidt and Householder. c) Find the condition number of A in the $\|\cdot\|_\infty$, $\|\cdot\|_2$ and $\|\cdot\|_1$ norms.

Exercise 5-4. a) Let $A = a \otimes b$. Find all eigenvalues and eigenvectors of A. When is A a normal matrix? b) Let $H \in \mathbb{R}^{n\times n}$ be a Householder reflection. Show that H has a single eigenvalue at -1 and an eigenvalue $+1$ of multiplicity $(n-1)$. What is the value of $\det(H)$?

Exercise 5-5. Determine the asymptotic computational cost of algorithm QR when calculating the full matrix Q.

Exercise 5-6. Write a Matlab algorithms to solve

$$\begin{aligned} Ux &= y \\ Ly &= b \end{aligned}$$

where $U \in \mathbb{R}^{n\times n}$ (resp. $L \in \mathbb{R}^{n\times n}$) is an upper (resp. lower) triangular matrix. Construct examples U and y where you know the exact solution x (for some large n). Compute $\hat{x}$ using your algorithm and study the error

$$e = \frac{\|x - \hat{x}\|}{\|x\|},$$

discussing your results.

Now generate a random upper triangular $U \in \mathbb{R}^{50\times 50}$ with non-zero entries uniformly distributed in $[-1, 1]$. Let $x = (1, \ldots, 1)^T \in \mathbb{R}^{50}$ and compute $y = Ux$ by use of Matlab's matrix multiply. Solve $U\hat{x} = y$ (with the just computed y) using the algorithm you designed above and compute the relative error

$$e = \frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty}.$$

Do this several hundred times and discuss the resulting statistics of e.

Exercise 5-7. Using the construction from Exercise 2-2 as the basis for an algorithm, and making maximum use of symmetry, show that Cholesky factorisation of a symmetric positive-definite matrix can be achieved in approximately half the flops of standard Gaussian Elimination.

Exercise 5-8. Let A be a symmetric positive definite matrix that is also strongly (row) diagonally dominant. Let $g_n(A)$ be the growth factor from (5.1). Show that when Gaussian Elmination is applied to A we get $g_n(A) \leq 1$.

Exercise 5-9. Banded matrices arise frequently in discretisations of PDEs and it is thus important to understand them both from a theoretical and a practical point of view.

We consider here $m \times m$ tridiagonal matrices. These have the form

$$\begin{pmatrix} d_1 & c_1 & & & \\ a_2 & d_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{m-1} & d_{m-1} & c_{m-1} \\ & & & a_m & d_m \end{pmatrix} \tag{5.5}$$

First of all it will be useful to be able to invert tridiagonal matrices. Consider solving

$$Ax = b, \tag{5.6}$$

where A is defined by (5.5) and b is an $m \times 1$ vector of the form $b = [b_1, b_2, \ldots, b_m]^T$.

Given the coefficients a_i, d_i, c_i of the tridiagonal matrix A and the right-hand side b_i, the solution x_i of (5.6) may be found as follows.

Algorithm (TriDiagonal Solver).
input: the coefficients a_i, d_i, c_i of the tridiagonal matrix A
output: $x \in \mathbb{R}^n$

for $k = 2, \ldots, m$ **do**
    if $d_{k-1} = 0$ **then**
        signal failure and stop
    else
        $rm = \frac{a_k}{d_{k-1}}$
    end if
    $d_k = d_k - rm * c_{k-1}$
    $b_k = b_k - rm * b_{k-1}$
end for
if $d_n = 0$ **then**
    signal failure and stop
else
    $x_m = \frac{b_m}{d_m}$
end if
for $k = m - 1, \ldots, 1$ **do**
    $x_k = \frac{b_k - c_k x_{k+1}}{d_k}$
end for

Note that this algorithm does not include any pivoting strategies. When and why is this satisfactory?

Exercise 5-10. Prove Theorem 5.7 about the backward stability of algorithm GE.

Chapter 6

Iterative Methods

In the previous chapter we considered methods which directly solve (SLE) using $\Theta(n^3)$ operations. In this chapter we will introduce iterative methods. These methods construct a sequence $(x_k)_{k\in\mathbb{N}}$ with

$$x_k \to x \qquad \text{for } k \to \infty,$$

where x solves (SLE). These methods approximate the exact solution x by the computed value x_k for large k and thus, even in the absence of rounding errors, the resulting algorithms only compute approximate solutions to (SLE). In practical applications this is no problem, since one can choose k large enough such that the approximation error is of the same order of magnitude as the deviation caused by rounding errors in the "exact" methods.

The primary motivation for the use of such iterative methods arises for problems where $A \in \mathbb{C}^{n\times n}$ with $n \gg 1$, so that solving the problem by the direct methods of Chapter 5, with cost $\Theta(n^3)$, is prohibitively expensive. If A has some special structure, for example if matrix-vector multiplication with A is cheap to calculate, then iterative methods can have a substantial advantage over direct methods.

For the methods considered in this chapter, the sequence (x_k) is constructed iteratively, i.e. starting with a value x_0 the value x_k, for each k, is computed from x_{k-1}. The map $x_{k-1} \mapsto x_k$ may be either linear or nonlinear; we discuss methods of both types. The two competing criteria for the iteration are:

- The calculation of x_k from x_{k-1} should be *cheap*, relative to the cost of solving (SLE).
- The convergence of x_k to x should be *fast*.

These two criteria are often contradictory to one another and the trade-off between them is very much dependent upon the matrix A itself. To illustrate this trade off it is useful to have the following two extreme iterative methods in mind:

- $x_k = A^{-1}b$; and
- $x_k = x_{k-1}$.

The first is not cheap to implement — each step involves solving (SLE) by a direct method, and our assumption is that this is prohibitively expensive — but converges fast: in one step. The second is cheap to implement but does not converge at all. Successful iterative methods lie somewhere between these extremes.

In many cases, the cost of computing x_k from x_{k-1} is dominated by the cost of matrix-vector multiplication for a matrix N which is derived from A. The resulting methods will be efficient if these multiplications can be computed efficiently. The most common structure leading to cheap matrix-vector multiplication is ***sparsity***: where A has only a small number of non-zero entries. Then matrix-vector multiplication by A, A^* or any matrix N where N inherits the sparsity of A will be cheap; the matrix N might, for example, be formed from the off-diagonals of A.

In this chapter we will focus only on errors incurred through using a finite number of iterations but we will not discuss the effect of rounding errors. For each section describing a computational approach there will be three subsections, one defining the method, the second studying error as a function of the number of iterations k, and the third discussing computational cost. We will consider the cost of achieving an error of size ε. We will show that, for some important problems, it is possible to compute an approximate solution in time $\Theta(n^a)$ with $a < 3$ as the dimension n of the problem increases, thus beating the cost of the direct methods from the previous chapter.

6.1 Linear Methods

Iterative methods where x_k is computed from x_{k-1} by application of a linear map are called *linear methods.* In this section we describe results about linear methods in general. Sections 6.2 and 6.3 below introduce two specific instances of linear methods, namely the Jacobi method and the successive over relaxation scheme (SOR).

The Method

The basic idea of the methods described in this section is to write the matrix A as $A = M + N$ where the matrix M is easier to invert than A. Then, given x_{k-1}, we can define x_k by

$$Mx_k = b - Nx_{k-1}. \tag{6.1}$$

If we assume for the moment that $\lim_{k\to\infty} x_k = x$, then the limit x satisfies

$$Mx = \lim_{k\to\infty} Mx_k = \lim_{k\to\infty} (b - Nx_{k-1}) = b - Nx$$

and thus we get $Ax = b$. This shows that the only possible limit for this sequence is the solution of (SLE). Thus, once we know that the generated sequence converges for a given matrix A and a given initial value x_0, we can iteratively compute x_k for a "big" value of k to get an approximation for x. In the following sections we will give sufficient criteria for convergence of the sequence $(x_k)_{k\in\mathbb{N}}$.

This leads to the following algorithm.

Algorithm LI (linear iterative methods).
input: $A = M + N \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$, $x_0 \in \mathbb{C}^n$
output: $x_k \in \mathbb{C}^n$ with $Ax_k \approx b$

```
for k = 1, 2, 3, ... do
    compute y_k = b - N x_{k-1}
    solve M x_k = y_k
end for
```

To actually implement this method one needs to choose M and N such that the method converges (we discuss different choices below). One also needs a stopping criterion to decide when to quit iterating and to return the resulting approximation x_k.

Thus, the remaining problem is to choose a stopping criterion for the iteration. We consider the *error*

$$e_k = x - x_k,$$

and the *residual vector*

$$r_k = b - Ax_k = Ae_k$$

where x is the exact solution of (SLE) and x_k the approximation at step k. We would like to choose k big enough such that $\|e_k\| \leq \varepsilon$ for a given $\varepsilon > 0$ but, since e_k cannot be computed without knowledge of the exact solution x, this is not a practical stopping criterion. Instead, we will stop the iteration once $\|r_k\| \leq \varepsilon_\mathrm{r}$ for some $\varepsilon_\mathrm{r} > 0$. The residual error r_k can easily be computed during the iteration and from the estimate $\|e_k\| = \|A^{-1}r_k\| \leq \|A^{-1}\|\|r_k\| \leq \|A^{-1}\|\varepsilon_\mathrm{r}$ we can get estimates for the error if needed.

Error Analysis

Since $Mx = b - Nx$ we get the relation

$$e_k = -M^{-1}Ne_{k-1}.$$

The method converges if $e_k \to 0$ for $k \to \infty$. The following lemma characterises convergence with the help of the spectral radius of the matrix $R = -M^{-1}N$.

Lemma 6.1. *Let $R \in \mathbb{C}^{n\times n}$, $e_0 \in \mathbb{C}^n$ and $e_k = R^k e_0$ for all $k \in \mathbb{N}$. Then $e_k \to 0$ for all $e_0 \in \mathbb{C}^n$ if and only if $\rho(R) < 1$.*

Proof. Assume first $\rho(R) < 1$. Then by Lemma 2.10 we find an induced matrix norm $\|\cdot\|_{\mathcal{S}}$ with $\|R\|_{\mathcal{S}} < 1$ and we get

$$\|e_k\|_{\mathcal{S}} = \|R^k e_0\|_{\mathcal{S}} \le \|R\|_{\mathcal{S}}^k \|e_0\|_{\mathcal{S}} \longrightarrow 0$$

for $k \to \infty$.

On the other hand, if $\rho(R) \ge 1$, then there is an $e_0 \in \mathbb{C}^n \setminus \{0\}$ with $Re_0 = \lambda e_0$ for some $\lambda \in \mathbb{C}$ with $|\lambda| \ge 1$. For this vector e_0 we get, in any norm $\|\cdot\|$,

$$\|e_k\| = \|R^k e_0\| = |\lambda|^k \|e_0\|$$

and thus e_k does not converge to 0 as $k \to \infty$. □

Remarks.

1. The lemma shows that the linear iterative method defined by (6.1) converges for every initial condition $x_0 \in \mathbb{C}^n$ if and only if $\rho(R) < 1$ for $R = -M^{-1}N$. The convergence is fast if $\rho(R)$ is small.

2. Since $\|\cdot\| \ge \rho(\cdot)$ for every matrix norm $\|\cdot\|$, a sufficient criterion for convergence of an iterative method is $\|R\| < 1$ for any matrix norm $\|\cdot\|$. On the other hand, whenever the method converges for every initial condition, there is an induced matrix norm $\|\cdot\|$ with $\|R\| < 1$ by Lemma 2.10.

Computational Complexity

When measuring computational cost for iterative methods it is of interest to know how many iterations are required to reduce the error to size ε. As before we will consider the relative error $\|x_k - x\|/\|x\| = \|e_k\|/\|x\|$. By Proposition 3.3 we have

$$\frac{\|e_k\|}{\|x\|} \le \kappa(A)\frac{\|r_k\|}{\|b\|}. \tag{6.2}$$

Finding k to achieve

$$\frac{\|r_k\|}{\|b\|} \le \varepsilon, \tag{6.3}$$

and thus

$$\frac{\|e_k\|}{\|x\|} \le \kappa(A)\varepsilon, \tag{6.4}$$

will lead to an estimate of the computational cost of using the iterative method.

Using the relation $r_k = Ae_k = AR^k e_0$ we can estimate the left hand side of (6.3) to get, from (6.2),

$$\frac{\|e_k\|}{\|x\|} \le \kappa(A)\frac{\|A\|\|R\|^k\|e_0\|}{\|b\|}$$

for any vector norm $\|\cdot\|$. If $\|\cdot\|$ is chosen so that $\|R\| < 1$ then choosing k to be the smallest integer greater than

$$k^\sharp = \frac{\ln \varepsilon^{-1} + \ln \|A\| + \ln \|e_0\| - \ln \|b\|}{\ln \|R\|^{-1}} \tag{6.5}$$

will ensure (6.3).

In order to study the dependence of the cost on the dimension n of the problem we consider a family of matrices $A = M + N \in \mathbb{C}^{n\times n}$ and vectors $b \in \mathbb{C}^n$ and also a family of vector norms on the spaces $\mathbb{C}^n$. The number $k^\sharp$ depends on both ε and n; the n dependence can enter through $\|R\|^{-1}$, $\|A\|, \|b\|$ and $\|e_0\|$, but the largest contribution is often through $\|R\|^{-1}$, for families of problems in which $\|R\| = 1-\Theta(n^{-\beta})$. We incorporate this into the following assumptions, which lead to a quantification of computational cost.

Assumption 6.2. 1. Calculation of Nx and $M^{-1}x$ together costs $\Theta(n^\alpha)$ for some $\alpha > 0$, uniformly for all $x \in \mathbb{C}^n$.

2. $\|R\| = 1 - \Theta(n^{-\beta})$ for some $\beta > 0$.

3. $\kappa(A)$, $\|A\|$, $\|b\|$ and $\|e_0\|$ are bounded uniformly in n.

Theorem 6.3. *Under Assumption 6.2 the computational cost to achieve* (6.4) *by using the linear iterative method is bounded above by $cn^{\alpha+\beta}\ln\varepsilon^{-1}$, for some constant c independent of n and ε.*

Proof. The first item of the assumption ensures that each step of the iteration costs $\Theta(n^\alpha)$. The second item of the assumption implies that $\big(\ln\|R\|^{-1}\big)^{-1} = \Theta(n^\beta)$. Hence combining the second and third items and using (6.5) gives the desired result. □

Remarks.

1. For the theorem to be practically useful, the norm in (6.3) must be readily computable. In some cases $\|R\| < 1$ in some computable norm such as $\|\cdot\|_\infty, \|\cdot\|_2$ or $\|\cdot\|_1$. Then the preceding theorem applies directly.

   In other cases we only know that $\|R\|_{\mathcal{S}} < 1$; as the construction of $\|\cdot\|_{\mathcal{S}}$ requires knowledge of the Jordan Canonical form, this is not a readily computable norm. However, by norm equivalence, we know that there is a constant $c_2 \in (0,\infty)$ such that

$$\frac{\|a\|}{\|b\|} \le c_2\frac{\|a\|_{\mathcal{S}}}{\|b\|_{\mathcal{S}}}.$$

   The constant c_2 may depend on n, but in many important applications this occurs only in a polynomial fashion.

   Iterating until we find a k so that

$$\frac{\|r_k\|_{\mathcal{S}}}{\|b\|_{\mathcal{S}}} \le \frac{\varepsilon}{c_2} \tag{6.6}$$

   will ensure

$$\frac{\|r_k\|}{\|b\|} \le \varepsilon$$

   and hence (6.4). Finding k to achieve (6.6) will incur cost

$$kn^{\alpha+\beta}\Big(\ln\varepsilon^{-1} + \ln c_2(n)\Big)$$

   by the methods given in the preceding theorem, if Assumption 6.2 holds in the norm $\|\cdot\|_{\mathcal{S}}$. Thus there is an increase in cost by only a logarithmic factor in n when $c_2(n)$ is polynomial in n.

2. In many applications $\kappa(A)$, $\|A\|$ and $\|b\|$ do depend upon n, but only polynomially. Again this leads to an increase in computational cost over the above, but only by a factor which is logarithmic in n.

6.2 The Jacobi Method

In this and the next section we will consider specific methods which are obtained by choosing the matrices M and N in (6.1). For a matrix $A \in \mathbb{C}^{n\times n}$ define the three $n \times n$-matrices

$$L = \begin{pmatrix} & & & & \\ a_{21} & & & & \\ a_{31} & a_{32} & & & \\ \vdots & \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & \end{pmatrix}, \quad D = \begin{pmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & a_{33} & & \\ & & & \ddots & \\ & & & & a_{nn} \end{pmatrix},$$

$$U = \begin{pmatrix} & a_{12} & a_{13} & \dots & a_{1n} \\ & & a_{23} & \dots & a_{2n} \\ & & & \ddots & \vdots \\ & & & & a_{n-1,n} \\ & & & & \end{pmatrix}. \tag{6.7}$$

Then we have $A = L + D + U$. The matrices L and U are, respectively, strictly lower and upper triangular.

The Method

The iterative method obtained by choosing $M = D$ and $N = L + U$ in (6.1) is called *Jacobi method*. This choice of M and N leads to the iteration

$$x_k = D^{-1}\big(b - (L+U)x_{k-1}\big)$$

for all $k \in \mathbb{N}$. Since D is diagonal, the inverse D^{-1} is trivial to compute.

Error Analysis

In order to study convergence of the Jacobi method using Lemma 6.1 we have to consider the matrix $R = -M^{-1}N = -D^{-1}(L+U)$. The method converges if and only if $\rho(R) < 1$. The following theorems give sufficient criteria for this to happen.

Theorem 6.4. *a) The Jacobi method is convergent for all matrices A with*

$$|a_{ii}| > \sum_{j\neq i} |a_{ij}| \tag{6.8}$$

for $i = 1, \dots, n$. (This condition is called the strong row sum criterion.*)*

b) The Jacobi method is convergent for all matrices A with

$$|a_{jj}| > \sum_{i\neq j} |a_{ij}| \tag{6.9}$$

for $j = 1, \dots, n$. (This condition is called the strong column sum criterion.*)*

Proof. a) The matrix $R = -D^{-1}(L+U)$ has entries

$$r_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & \text{if } i \neq j, \text{ and} \\ 0 & \text{else.} \end{cases}$$

Using Theorem 1.28 we find

$$\|R\|_\infty = \max_{i=1,\dots,n} \frac{1}{|a_{ii}|} \sum_{j\neq i} |a_{ij}|.$$

Thus the strong row sum criterion gives $\|R\|_\infty < 1$ which implies $\rho(R) < 1$ and the method converges.

b) If the strong column sum criterion (6.9) holds for A, then the strong row sum criterion (6.8) holds for A^* and thus the method converges for A^*. From Lemma 6.1 we know then $\rho\big(-(D^*)^{-1}(L^*+U^*)\big) < 1$. Since for every matrix R the matrices R, R^* and also $D^{-1}RD$ have the same eigenvalues we get

$$\begin{aligned}\rho\big(-D^{-1}(L+U)\big) &= \rho\big(-D^{-1}(L+U)D^{-1}D\big)\\ &= \rho(-(L+U)D^{-1})\\ &= \rho\big(-(D^*)^{-1}(L^*+U^*)\big) < 1\end{aligned}$$

and so the method converges for A. □

Remark. As well as showing convergence, the proof of the theorem estimates $\|R\|_\infty < 1$. This estimate can be used in Theorem 6.3 to analyse computational complexity.

Definition 6.5. A matrix $A \in \mathbb{C}^{n\times n}$ is called *irreducible* if there is no permutation matrix P such that

$$P^TAP = \begin{pmatrix}\tilde{A}_{11} & \tilde{A}_{12}\\ 0 & \tilde{A}_{22}\end{pmatrix}$$

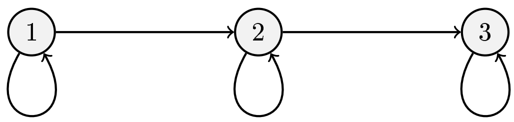
where $\tilde{A}_{11} \in \mathbb{C}^{p\times p}$ and $\tilde{A}_{22} \in \mathbb{C}^{q\times q}$ are square matrices with $p,q > 0$ and $p+q = n$, $\tilde{A}_{12} \in \mathbb{C}^{p\times q}$, and 0 is the $q \times p$ zero-matrix.

There is an alternative description of irreducibility, which is often easier to check than the definition given. To the matrix A we associate the oriented graph $G(A)$ with vertices $1,\dots,n$ and edges $i \longrightarrow j$ for all $i,j \in \{1,\dots,n\}$ with $a_{ij} \neq 0$. Then the matrix A is irreducible if and only if the graph $G(A)$ is connected, i.e. if you can reach any vertex j from any vertex i by following edges.

Example. Consider the matrix

$$A = \begin{pmatrix}1 & -1 & \\ & 1 & -1\\ & & 1\end{pmatrix}.$$

The associated graph is

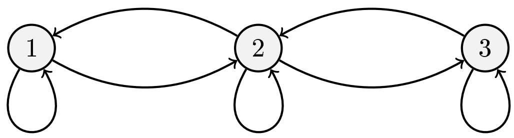


The matrix A is not irreducible (indeed $P = I$ in the definition is enough to see this) and since there is no path from 3 to 1, the graph $G(A)$ is not connected.

Example. In continuation of the previous example consider the modified matrix

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

The associated graph is



Now the graph $G(A)$ is connected and the matrix is thus irreducible.

Remarks.

The proof of equivalence of the two characterisations of irreducibility is based on the following observations.

1. For any permutation P the graphs $G(A)$ and $G(P^TAP)$ are isomorphic, only the vertices are numbered in a different way.
2. The block $\tilde{A}_{22}$ in the definition of irreducibility corresponds to a set of states from where there is no path into the states corresponding to the block $\tilde{A}_{11}$.

The characterisation of irreducibility through $G(A)$ shows that the diagonal entries of A are irrelevant to determining irreducibility. Any matrix which has non-zero off-diagonal entries if and only if A does so, will have the same graph. Hence $R = -D^{-1}(L+U)$ is irreducible if and only if A is.

Theorem 6.6. *If A is irreducible and satisfies*

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \tag{6.10}$$

for $i = 1, \ldots, n$ but

$$|a_{kk}| > \sum_{j \neq k} |a_{kj}| \tag{6.11}$$

for one index k, then the Jacobi method converges. (This condition is called the weak row sum criterion.*)*

Proof. We have to show that $\rho(R) < 1$ where $R = -D^{-1}(L+U)$. Define the matrix $|R| = (|r_{ij}|)_{ij} \in \mathbb{R}^{n\times n}$ and the vector $e = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$. Then we have

$$(|R|e)_i = \Big(\sum_{j=1}^{n} |r_{ij}| \cdot 1 \Big)_i = \Big(\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} \Big) \leq 1 = e_i.$$

Thus $|R|e \leq e$ where this and some of the following inequalities between vectors are to be read componentwise. Therefore we get

$$|R|^{l+1}e \leq |R|^l e \leq \cdots \leq e$$

for all $l \in \mathbb{N}$.

Let $t^{(l)} = e - |R|^l e \geq 0$. Then the vectors $t^{(l)}$ are componentwise increasing. Let τ_l be the number of non-vanishing components of $t^{(l)}$. We will show that τ_l is strictly increasing until it reaches the value n. Assume, for contradiction, that $\tau_{l+1} = \tau_l = k < n$. Since one row of A satisfies the strict inequality (6.11) we have $|R|e \neq e$ and thus $k > 0$. Then without loss of generality (since we can reorder the rows and columns of A) we have

$$t^{(l)} = \begin{pmatrix} a \\ 0 \end{pmatrix}, \quad t^{(l+1)} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

where $a, b \in \mathbb{R}^k$ and $a, b > 0$. We can conclude

$$\begin{aligned}
\begin{pmatrix} b \\ 0 \end{pmatrix} &= t^{(l+1)} = e - |R|^{l+1}e \\
&\geq |R|e - |R|^{l+1}e = |R|t^{(l)} \\
&= \begin{pmatrix} |R_{11}| & |R_{12}| \\ |R_{21}| & |R_{22}| \end{pmatrix} \begin{pmatrix} a \\ 0 \end{pmatrix}.
\end{aligned}$$

From $a > 0$ we have $|R_{21}| = 0$. Hence R is not irreducible. This implies that A is not irreducible, and we have found the required contradiction. Thus we can conclude that $\tau_{l+1} > \tau_l$ whenever $\tau_l < n$. This proves $t^{(n)} > 0$ componentwise.

Hence $e > |R|^n e$ and we get, using Lemma 1.34,

$$\rho(R)^n \leq \rho(R^n) \leq \|R^n\|_\infty = \big\||R^n|\big\|_\infty \leq \big\||R|^n\big\|_\infty = \max_{i=1,\ldots,n} \big(|R|^n e\big)_i < 1$$

and thus $\rho(R) < 1$. This completes the proof. □

Computational Complexity

The sequence $(x_k)_{k\in\mathbb{N}}$ from the Jacobi method is defined by the relation

$$x_k = -D^{-1}\big(b - (L+U)x_{k-1}\big).$$

For each iteration we have to calculate
1) the product $(L+U)x_{k-1}$
2) the difference $b - \cdots$ (needs n subtractions)
3) the product $D^{-1}\cdots$ (needs n divisions because D is diagonal)

For general matrices the first step needs $\Theta(n^2)$ operations and thus dominates the computational cost. If the matrix A is sparse, i.e. if it contains many zeros, the matrix-vector multiplication can be performed using fewer operations.

Example. Assume that each row of A contains at most $\ell > 0$ non-zero entries outside the diagonal. Then step 1) requires $\Theta(n)$ operations and performing one step of Jacobi method has cost $\Theta(n)$ as $n \to \infty$. In this case we get $\alpha = 1$ in Theorem 6.3.

6.3 The Gauss-Seidel and SOR Methods

The Method

The method obtained by using $M = L+\omega D$ and $N = U+(1-\omega)D$ in (6.1), for the matrices L, D and U from (6.7) and some $\omega \in \mathbb{R}$, is called the *Successive Over Relaxation* (SOR) scheme. The parameter ω can be chosen to accelerate convergence. The special case $\omega = 1$, i.e. $M = L + D$ and $N = U$, is called the *Gauss-Seidel method.* For the SOR scheme we get the iteration

$$(L+\omega D)x_k = b - Ux_{k-1} - (1-\omega)Dx_{k-1}.$$

Since $(L+\omega D)$ is lower triangular we can use forward substitution to calculate x_k in each step.

Error Analysis

The error analysis of the SOR and Gauss-Seidel method is similar to the analysis for the Jacobi method from the previous section. The convergence properties of the SOR scheme are determined by the matrix

$$R = -(L+\omega D)^{-1}\big(U + (1-\omega)D\big).$$

The following summarises some sufficient criteria for convergence of the Gauss-Seidel method.

Theorem 6.7. *Assume either*
a) A satisfies the strong row sum criterion or
b) A is irreducible and satisfies the weak row sum criterion.

Then the Gauss-Seidel ($\omega = 1$) method converges.

Computational Complexity

The speed of convergence and thus the cost of the method depends on the choice of the parameter ω. For linear systems arising from discretisation of certain elliptic PDEs, optimising can lead to an order of magnitude improvement in efficiency, when compared with Gauss-Seidel and Jacobi methods. This arises though decreasing β in Theorem 6.3 for appropriate choice of ω. In the same context Gauss-Seidel improves over Jacobi, but not by an order of magnitude.

6.4 Nonlinear Methods

In the remaining part of this chapter we will consider *nonlinear methods* for solving (SLE), i.e. methods where the map from x_k to x_{k+1} is no longer linear. We restrict ourselves to the case where the matrix A is Hermitian and positive definite. This section explains the general approach. The remaining two sections of the chapter examines two specific instances of nonlinear methods in detail.

The fundamental observation for the methods we consider here is that the vector equation $Ax = b$ is equivalent to $\|Ax - b\| = 0$ for any vector norm $\|\cdot\|$. Since we assume that A is Hermitian and positive definite, $\langle x, y\rangle_A = \langle x, Ay\rangle$ defines an inner product and $\|x\|_A^2 = \langle x, Ax\rangle$ defines the associated vector norm. The matrix A^{-1} is again Hermitian and positive definite and $\|x\|_{A^{-1}}^2 = \langle x, A^{-1}x\rangle$ defines the associated vector norm.

Lemma 6.8. *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian and positive definite and $b \in \mathbb{C}^n$. Then $x \in \mathbb{C}^n$ solves $Ax = b$ if and only if x is the minimiser of the function*

$$g(y) = \frac{1}{2}\|Ay - b\|_{A^{-1}}^2 \quad \forall y \in \mathbb{C}^n. \tag{6.12}$$

In this section we will consider iterative methods which solve (SLE) by constructing sequences $(x_k)_{k\in\mathbb{N}}$ which converge to the unique minimum of the function g. Using the residual $r_k = b - Ax_k$ and the error $e_k = x - x_k$, where x is the exact solution of $Ax = b$, we can write

$$g(x_k) = \frac{1}{2}\|r_k\|_{A^{-1}}^2 = \frac{1}{2}\|e_k\|_A^2.$$

Thus minimising g corresponds to minimising the length of the residual vector in the $\|\cdot\|_{A^{-1}}$-norm or, equivalently, to minimising the error for the exact solution in the $\|\cdot\|_A$-norm.

The Method

The two methods that we study below both take the form

$$x_k = x_{k-1} + \alpha_{k-1}d_{k-1}, \tag{6.13}$$

where $d_{k-1} \in \mathbb{C}^n \setminus \{0\}$ is called the *search direction* and the scalar $\alpha_{k-1} \in \mathbb{C}$ is called the *step length*. The step length is chosen so that, given x_{k-1} and d_{k-1},

$$\alpha_{k-1} = \operatorname*{argmin}_{\alpha\in\mathbb{C}} g(x_{k-1} + \alpha d_{k-1}).$$

Since g is a convex, quadratic function, the value α_{k-1} is uniquely determined.

The minimum can be found using the fact the g satisfies

$$g(y + \varepsilon z) = g(y) + \varepsilon \operatorname{Re}\langle z, Ay - b\rangle + \frac{\varepsilon^2}{2}\|z\|_A^2 \tag{6.14}$$

for all $\varepsilon \in \mathbb{R}$ and $y, z \in \mathbb{C}^n$, where Re denotes the real part of a complex number. Taking the derivative w.r.t. ε (along the real line) gives

$$\frac{\partial}{\partial\varepsilon} g(y + \varepsilon z)\big|_{\varepsilon=0} = \operatorname{Re}\langle z, Ay - b\rangle. \tag{6.15}$$

At the minimum we find

$$\begin{aligned}
0 &= \frac{\partial}{\partial\varepsilon} g(x_{k-1} + \alpha_{k-1}d_{k-1} + \varepsilon\bar{\beta}d_{k-1})\big|_{\varepsilon=0} \\
&= \operatorname{Re}\beta\langle d_{k-1}, A(x_{k-1} + \alpha_{k-1}d_{k-1}) - b\rangle \\
&= \operatorname{Re}\beta\langle d_{k-1}, -r_{k-1} + \alpha_{k-1}Ad_{k-1}\rangle
\end{aligned}$$

for all $\beta \in \mathbb{C}$ and thus

$$\alpha_{k-1} = \frac{\langle d_{k-1}, r_{k-1}\rangle}{\|d_{k-1}\|_A^2}.$$

Hence we get the following algorithm.

Algorithm NI (nonlinear iterative methods).
input: $A \in \mathbb{C}^{n\times n}$ Hermitian and positive definite, $b \in \mathbb{C}^n$, $x_0 \in \mathbb{C}^n$
output: $x_k \in \mathbb{C}^n$ with $Ax_k \approx b$

1: **for** $k = 1, 2, 3, \dots$ **do**
2: compute d_{k-1}
3: $\alpha_{k-1} = \dfrac{\langle d_{k-1}, r_{k-1}\rangle}{\|d_{k-1}\|_A^2}$
4: $x_k = x_{k-1} + \alpha_{k-1} d_{k-1}$
5: **end for**

The steepest descent method and the conjugate gradient method, discussed below, are instances of this algorithm, employing different choices for the search directions d_k.

Error Analysis

The error analysis for nonlinear methods is done in two steps. We first study how fast the value $\|e_k\|_A^2 = 2g(x_k)$ decays. This analysis depends on the specific choice of search directions d_k and is presented for the two methods below, separately. In a second step we can use this result to estimate the error of the approximate solution x_k in the Euclidean norm:

Lemma 6.9. *Assume $\|e_k\|_A \leq cq^k \|e_0\|_A$ for all $k \in \mathbb{N}$ for some constants $q, c > 0$. Then*

$$\|e_k\| \leq \sqrt{\kappa(A)}c\, q^k \|e_0\| \quad \forall k \in \mathbb{N}$$

where $\kappa(A) = \|A\|_2 \|A^{-1}\|_2$ is the condition number of A in the 2-norm.

Proof. Let $\lambda_{\min}$ and $\lambda_{\max}$ be the minimal and maximal eigenvalue of A. Then we have $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ by (3.2) and from Lemma 1.21 we find

$$\|e_k\|^2 \leq \frac{1}{\lambda_{\min}} \|e_k\|_A^2 \leq \frac{2}{\lambda_{\min}} cq^k \|e_0\|_A^2 \leq \frac{\lambda_{\max}}{\lambda_{\min}} cq^k \|e_0\|^2.$$

This completes the proof. □

For future reference we state the following relation, which describes how the residual error changes in each step:

$$r_k = r_{k-1} - \alpha_{k-1} A d_{k-1} \quad \forall k \in \mathbb{N}. \tag{6.16}$$

This is a direct consequence of (6.13).

6.5 The Steepest Descent Method

The Method

The steepest descent method chooses a search direction which makes a step in the direction of the (locally) steepest descent for g. From equation (6.14) we get

$$g(x_k + \varepsilon d_k) = g(x_k) - \varepsilon \operatorname{Re}\langle d_k, r_k\rangle + \frac{\varepsilon^2}{2} \|d_k\|_A^2 \tag{6.17}$$

for all $\varepsilon \in \mathbb{R}$. Thus, the direction of steepest descent corresponds to the direction d_k which maximises $\operatorname{Re}\langle d_k, r_k\rangle$ while keeping the length of d_k fixed. Since d_k is only determined up to a scalar factor, we can choose

$$d_k = r_k \quad \forall k \in \mathbb{N}_0$$

in (6.13). With this choice, the step length becomes

$$\alpha_k = \frac{\|r_k\|^2}{\|r_k\|_A^2}$$

and we get the following algorithm.

Algorithm SD (steepest descent).
input: $A \in \mathbb{C}^{n\times n}$ Hermitian, positive definite, $b \in \mathbb{C}^n$, $x_0 \in \mathbb{C}^n$, $\varepsilon_r > 0$
output: $x_k \in \mathbb{C}^n$ with $Ax_k \approx b$

for $k = 1, 2, \ldots$ **do**
  $r_{k-1} = b - Ax_{k-1}$
  if $\|r_{k-1}\| \leq \varepsilon_r$ **then**
    return x_{k-1}
  end if
  $\alpha_{k-1} = \|r_{k-1}\|^2 / \|r_{k-1}\|_A^2$
  $x_k = x_{k-1} + \alpha_{k-1} r_{k-1}$
end for

Error Analysis

The following result establishes the convergence of the method.

Theorem 6.10. *Let A be Hermitian and positive definite. Then the rate of convergence of the steepest descent algorithm for solving (SLE) is given by*

$$\|x_k - x\|_A \leq \sqrt{1 - 1/\kappa(A)}^{\,k} \|x_0 - x\|_A \quad \forall k \in \mathbb{N},$$

where $\kappa(A)$ is the condition number of A in the 2-norm.

Proof. By substituting $\varepsilon = \alpha_{k-1}$ into equation (6.17) we get

$$g(x_k) = g(x_{k-1}) - \frac{1}{2}\frac{\langle d_{k-1}, r_{k-1}\rangle^2}{\|d_{k-1}\|_A^2} = \Big(1 - \frac{\langle d_{k-1}, r_{k-1}\rangle^2}{\|d_{k-1}\|_A^2 \|r_{k-1}\|_{A^{-1}}^2}\Big) g(x_{k-1})$$

for all $k \in \mathbb{N}$. Together with the estimates from Lemma 1.21 we get

$$g(x_k) \leq \Big(1 - \frac{\|r_{k-1}\|^4}{\lambda_{\max}\|r_{k-1}\|^2 \frac{1}{\lambda_{\min}}\|r_{k-1}\|^2}\Big) g(x_{k-1}) = \Big(1 - \frac{1}{\kappa(A)}\Big) g(x_{k-1})$$

for all $k \in \mathbb{N}$ and thus

$$g(x_k) \leq \Big(1 - \frac{1}{\kappa(A)}\Big)^k g(x_0).$$

The result follows then with $\|e_k\|_A^2 = 2g(x_k)$. □

Remarks.

1. If the matrix A is ill-conditioned then the method converges slowly. Ill-conditioning occurs in the 2-norm when the eigenvalues of A are very different: $\lambda_{\max} \gg \lambda_{\min}$. In this case the steepest descent direction is almost orthogonal to the direction in which the solution lies.

2. Since the steepest descent method uses $d_k = r_k$, the search directions satisfy $r_k = r_{k-1} - \alpha_{k-1} A r_{k-1}$ by (6.16). Using induction we get

$$x_k \in x_0 + \operatorname{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\} \tag{6.18}$$

   for all $k \in \mathbb{N}$.

Computational Complexity

We can achieve

$$\|x_k - x\|_A \leq \varepsilon \tag{6.19}$$

by choosing k to be the smallest integer greater than

$$k^\# = \frac{\ln \varepsilon^{-2} + \ln \|x_0 - x\|_A^2}{-\ln\Big(1 - \kappa(A)^{-1}\Big)}. \tag{6.20}$$

Assumption 6.11. 1. Calculation of Ax costs $\mathcal{O}(n^\alpha)$ uniformly in $x \in \mathbb{C}^n$.

2. The condition number of A in the 2-norm satisfies $\kappa(A) = \mathcal{O}(n^\beta)$.

3. $\|x - x_0\|_A$ is bounded uniformly in n.

Theorem 6.12. *Under Assumption 6.11 the computational cost of using the steepest descent method to achieve* (6.19) *is bounded above by* $cn^{\max\{\alpha,1\}+\beta} \ln \varepsilon^{-1}$, *for some constant* c *independent of* n *and* ε.

Proof. The first item of the assumption ensures that each step of the iteration costs $\Theta(n^{\max\{\alpha,1\}})$. Hence, combining the second and third items and using (6.20) gives the desired result. □

6.6 The Conjugate Gradient Method

Theorem 6.10 shows that the steepest descent method can behave quite poorly on ill-conditioned problems. The conjugate gradient method, and its pre-conditioned variants, go a long way to ameliorating this difficulty, at little extra cost when compared to the method of steepest descents.

Again, we assume that A is Hermitian and positive definite. The conjugate gradient method is similar to the steepest descent method but instead of making steps in direction of the steepest descent it restricts the increments $x_k - x_{k-1}$ to lie in the *Krylov subspace*

$$\mathcal{K}_k(A, r_0) = \operatorname{span}\{r_0, Ar_0, A^2 r_0, \dots, A^{k-1} r_0\}.$$

Since the spaces $\mathcal{K}_k(A, r_0)$ are increasing, this leads to $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ for all k and, as we will see, the algorithm even chooses x_k to minimise the function g from (6.12) on this space.

An underlying, general approach to the design of iterative methods for systems of linear equations (and also for the eigenvalue problem) is as follows: seek approximate solutions which at the k^{th} step of the iteration use only increments in $\mathcal{K}_k(A, r_0)$. Such methods are natural when matrix-vector multiplication by A is cheap to compute. Equation (6.18) shows that the steepest descent method follows this principle. The conjugate gradient method can be implemented almost as easily as the method of steepest descents, but since it finds the *optimal* approximation to x from $\mathcal{K}_k(A, r_0)$ we expect it to perform better. Indeed, the CG method converges considerably faster on ill-conditioned problems. Many other iterative methods also use the idea of finding approximations from the Krylov subspace.

The Method

Definition 6.13. Given A Hermitian positive-definite we say that the set $\{d_0, \cdots, d_k\}$ is *A-orthogonal* (or *conjugate*) if

$$\langle d_i, d_j \rangle_A = 0 \quad \forall i \neq j.$$

The following Lemma shows that choosing the search directions d_k to be A-orthogonal is advantageous for non-linear iterative methods.

Lemma 6.14. *Assume that the search directions* $\{d_0, \dots, d_{k-1}\}$ *form an A-orthogonal set. Then* x_k, *given by* (6.13), *minimises* g *over* $x_0 + \operatorname{span}\{d_0, \cdots, d_{k-1}\}$.

Proof. Define $G\colon \mathbb{C}^n \to \mathbb{R}$ by

$$G(\gamma) = g(x_0 + \sum_{l=0}^{k-1} \gamma_l d_l) \quad \forall \gamma \in \mathbb{C}^n.$$

Then G is a convex quadratic function and has a unique minimum where all directional derivatives vanish. Using (6.15) with $y = x_0 + \sum_{l=0}^{k-1} \gamma_l d_l$ and $z = \bar{\beta} d_m$ we find

$$\operatorname{Re} \beta \langle d_m, Ax_0 + \sum_{l=0}^{k-1} \gamma_l A d_l - b \rangle = 0$$

for all $\beta \in \mathbb{C}$ and thus

$$\gamma_m = \frac{\langle d_m, r_0 \rangle}{\|d_m\|_A^2}$$

for $m = 0, \ldots, k-1$.

Using (6.16) and the conjugacy of the d_l we get

$$\langle d_m, r_k \rangle = \langle d_m, r_{k-1} \rangle - \alpha_{k-1} \langle d_M, A d_{k-1} \rangle = \langle d_m, r_{k-1} \rangle$$

for $k = 1, \ldots, m$ and thus $\langle d_m, r_0 \rangle = \langle d_m, r_m \rangle$. This shows that the minimum is, indeed, attained for $\gamma = \alpha$. $\square$

The steepest descent method also chooses $x_k \in x_0 + \operatorname{span}\{d_0, \cdots, d_{k-1}\}$, but minimises g only along a line instead of over all of the affine sub-space. Thus, any method with A-orthogonal search directions will be at least as good as the steepest descent method.

In general it is expensive to construct orthogonal sets – something like a Gram-Schmidt orthogonalisation is required. Thus to make a viable method it remains to try and construct an iteration which iteratively generates an A-orthogonal set in a cheap fashion. The idea of the conjugate gradient method is to construct the A-orthogonal set by using the iteration

$$d_k = r_k + \beta_k d_{k-1}$$

where the factor β_k is chosen to enforce $\langle d_{k-1}, d_k \rangle_A = 0$, giving

$$\beta_k = -\frac{\langle d_{k-1}, r_k \rangle_A}{\|d_{k-1}\|_A^2}.$$

This clearly ensures that d_k is orthogonal to d_{k-1}; we will see that it is possible to choose d_0 such that the entire set $\{d_0, \ldots, d_k\}$ is an A-orthogonal set.

A preliminary form of the resulting algorithm is given by the following procedure: choose $x_0, d_0 \in \mathbb{C}^n$ and let $r_0 = b - Ax_0$. Then, for $k = 1, 2, \ldots$ define, in the given order,

$$\alpha_{k-1} = \frac{\langle d_{k-1}, r_{k-1} \rangle}{\|d_{k-1}\|_A^2}, \quad x_k = x_{k-1} + \alpha_{k-1} d_{k-1},$$
$$r_k = r_{k-1} - \alpha_{k-1} A d_{k-1}, \quad \beta_k = -\frac{\langle d_{k-1}, r_k \rangle_A}{\|d_{k-1}\|_A^2}, \quad d_k = r_k + \beta_k d_{k-1}. \tag{6.21}$$

Here we have used (6.16) to compute r_k inductively. This allows to compute all required quantities with only one matrix-vector multiplication per step (to calculate Ad_{k-1}).

Lemma 6.15. *Let $x_1, \ldots, x_k$ be given by the conjugate gradient method* (6.21) *with $d_0 = r_0$. Assume $r_0, \ldots, r_k \neq 0$ and $d_0, \ldots, d_{k-1} \neq 0$. Then $d_k \neq 0$ and $\langle d_{k-1}, r_k \rangle = \langle r_{k-1}, r_k \rangle = 0$. Furthermore*

$$\alpha_{k-1} = \frac{\|r_{k-1}\|^2}{\|d_{k-1}\|_A^2} > 0, \quad \beta_k = \frac{\|r_k\|^2}{\|r_{k-1}\|^2} > 0.$$

Proof. From the definition of r_k and the definition of α_{k-1} we get

$$\langle d_{k-1}, r_k \rangle = \langle d_{k-1}, r_{k-1} \rangle - \alpha_{k-1} \|d_{k-1}\|_A^2 = \langle d_{k-1}, r_{k-1} \rangle - \langle d_{k-1}, r_{k-1} \rangle = 0$$

and thus, by Pythagoras' theorem,

$$\|d_k\|^2 = \|r_k\|^2 + |\beta_k|^2 \|d_{k-1}\|^2 \geq \|r_k\|^2 > 0$$

and $d_k \neq 0$. For $k = 1$ we have $\langle r_{k-1}, d_{k-1} \rangle_A = \langle d_0, d_0 \rangle_A = \|d_{k-1}\|_A^2$. For $k > 1$ we can solve the equation defining d_{k-1} for r_{k-1} to get $\langle r_{k-1}, d_{k-1} \rangle_A = \langle d_{k-1} - \beta_{k-1} d_{k-2}, d_{k-1} \rangle_A = \|d_{k-1}\|_A^2$. Hence

$$\langle r_{k-1}, r_k \rangle = \langle r_{k-1}, r_{k-1} \rangle - \alpha_{k-1} \langle r_{k-1}, A d_{k-1} \rangle = 0$$

which is the second orthogonality relation from the claim.

For $k = 1$ we have $\langle d_{k-1}, r_{k-1}\rangle = \langle r_0, r_0\rangle = \|r_{k-1}\|^2$. For $k > 1$ we can use the the definition of d_{k-1} to get $\langle d_{k-1}, r_{k-1}\rangle = \langle r_{k-1} + \beta_{k-1}d_{k-2}, r_{k-2}\rangle = \|r_{k-1}\|^2$. Thus we get

$$\alpha_{k-1} = \frac{\langle d_{k-1}, r_{k-1}\rangle}{\|d_{k-1}\|_A^2} = \frac{\|r_{k-1}\|^2}{\|d_{k-1}\|_A^2} > 0.$$

Finally, since $\alpha_{k-1} > 0$, we can solve the definition of r_k for Ad_{k-1} to get

$$\beta_k = \frac{\langle -Ad_{k-1}, r_k\rangle}{\|d_{k-1}\|_A^2} = \frac{\langle r_k - r_{k-1}, r_k\rangle}{\alpha_{k-1}\|d_{k-1}\|_A^2} = \frac{\|r_k\|^2}{\|r_{k-1}\|^2} > 0.$$

This completes the proof. □

A consequence of Lemma 6.15 is that, assuming we have not already found the exact solution, we will have $d_k \neq 0$ and thus all the expressions in (6.21) make sense. We can now prove that the conjugate gradient method does indeed produce an A-orthogonal set of search directions.

Lemma 6.16. *Let $x_1, \ldots, x_k$ be given by the conjugate gradient method* (6.21) *with $d_0 = r_0$. Then $\{d_0, \cdots, d_{k-1}\} \subseteq \mathcal{K}_k(A, r_0)$ is an A-orthogonal set.*

Proof. We first prove by induction that

$$\operatorname{span}\{r_0, r_1, \ldots, r_l\} = \operatorname{span}\{d_0, d_1, \ldots, d_l\} = \operatorname{span}\{r_0, Ar_0, \ldots, A^l r_0\} \tag{6.22}$$

for $l = 0, \ldots, k-1$. Since $d_0 = r_0$, the statement holds for $l = 0$. Now let $l > 0$ and assume that the statement holds for $l - 1$. Then, since $\alpha_{l-1} \neq 0$, we can solve the definition of r_l for Ad_{l-1} to get $Ad_{l-1} \in \operatorname{span}\{r_{l-1}, r_l\}$ and thus

$$A^l r_0 = A(A^{l-1}r_0) \in \operatorname{span}\{Ad_0, \ldots, Ad_{l-1}\} \subseteq \operatorname{span}\{r_0, \ldots, r_l\}.$$

This shows $\operatorname{span}\{r_0, Ar_0, \ldots, A^l r_0\} \subseteq \operatorname{span}\{r_0, r_1, \ldots, r_l\}$. From the definition of d_k we get

$$r_l = d_l - \beta_l d_{l-1} \in \operatorname{span}\{d_{l-1}, d_l\} \subseteq \operatorname{span}\{d_0, d_1, \ldots, d_l\}$$

and thus $\operatorname{span}\{r_0, r_1, \ldots, r_l\} \subseteq \operatorname{span}\{d_0, d_1, \ldots, d_l\}$. Finally we have

$$d_l = r_l + \beta_l d_{l-1} = -\alpha_{l-1}Ad_{l-1} + r_{l-1} + \beta_l d_{l-1} \in \operatorname{span}\{r_0, \ldots, A^l r_0\}$$

and thus $\operatorname{span}\{d_0, d_1, \ldots, d_l\} \subseteq \operatorname{span}\{r_0, Ar_0, \ldots, A^l r_0\}$. This completes the proof of (6.22).

Assume for induction that $\{d_0, \ldots, d_{k-1}\}$ is an A-orthogonal set. This is true for $k = 1$. Now let $k > 1$. From the definition of r_{j+1} we get

$$\langle d_i, r_{j+1}\rangle = \langle d_i, r_j\rangle - \alpha_m\langle d_i, d_j\rangle_A = \langle d_i, r_j\rangle$$

for $j = i+1, \ldots, k-1$ and thus, by induction, $\langle d_i, r_j\rangle = \langle d_i, r_{i+1}\rangle = 0$ for all $0 \leq i < j \leq k$ where the last identity comes from Lemma 6.15. Since $Ad_i \in \operatorname{span}\{r_i, r_{i+1}\} \subseteq \operatorname{span}\{d_0, \ldots, d_{i+1}\}$, we get

$$\langle d_i, r_k\rangle_A = \langle Ad_i, r_k\rangle = 0$$

for $i = 0, \ldots, k-2$. Using this result we can prove that d_k is A-orthogonal to $d_0, \ldots, d_{k-1}$: for $i < k-1$ we get

$$\langle d_i, d_k\rangle_A = \langle d_i, r_k\rangle_A + \beta_k\langle d_i, d_{k-1}\rangle_A = \langle d_i, r_k\rangle_A = 0$$

and $\langle d_{k-1}, d_k\rangle = 0$ by the construction of β_k. This completes the proof. □

The CG-algorithm is normally implemented in the following form which exploits the expressions for α_k, β_k as derived in Lemma 6.15. This allows the required number of arithmetic operations to be kept small.

Algorithm CG (conjugate gradient method).
input: $A \in \mathbb{C}^{n\times n}$ Hermitian, positive definite, $b \in \mathbb{C}^n$, $x_0 \in \mathbb{C}^n$, $\varepsilon_r > 0$
output: $x_k \in \mathbb{C}^n$ with $x_k \approx A^{-1}b$

$r_0 = b - Ax_0$, $d_0 = r_0$
for $k = 1, 2, 3, \dots$ **do**
  $\alpha_{k-1} = \dfrac{\|r_{k-1}\|^2}{\|d_{k-1}\|_A^2} > 0$
  $x_k = x_{k-1} + \alpha_{k-1} d_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1} A d_{k-1}$
  if $\|r_k\| \le \varepsilon_r$ **then**
    return x_k
  end if
  $\beta_k = \dfrac{\|r_k\|^2}{\|r_{k-1}\|^2} > 0$
  $d_k = r_k + \beta_k d_{k-1}$
end for

Error Analysis

We have already seen, in Lemma 6.14, that any method with A-orthogonal search directions will have smaller errors than the steepest descent method. The following theorem shows a surprising result: these methods, even though iterative by construction, obtain the exact solution of (SLE) after at most n iterations.

Theorem 6.17. *If $\{d_0, \dots, d_{n-1}\}$ form an A-orthogonal set in $\mathbb{C}^n$ then, for any starting vector x_0, the sequence (x_k) given by* (6.13) *reaches the exact solution x of (SLE) in at most n steps.*

Proof. The set $\{d_0, \cdots d_{n-1}\}$ forms an A-orthogonal basis for $\mathbb{C}^n$ and so we may write

$$x - x_0 = \sum_{k=0}^{n-1} \gamma_k d_k, \quad \gamma_k = \frac{\langle d_k, x - x_0\rangle_A}{\|d_k\|_A^2}.$$

Also, by construction of the x_k,

$$x_k - x_0 = \sum_{i=0}^{k-1} \alpha_i d_i.$$

Conjugacy gives

$$\langle d_k, x_k - x_0\rangle_A = 0$$

and thus

$$\langle d_k, x - x_0\rangle_A = \langle d_k, x - x_0\rangle_A + \langle d_k, x_0 - x_k\rangle_A = \langle d_k, x - x_k\rangle_A = \langle d_k, r_k\rangle$$

for $k = 0, \dots, n-1$. Comparing the definitions of α_k and γ_k we find $\alpha_k = \gamma_k$ which implies

$$x - x_0 = \sum_{k=0}^{n-1} \alpha_k d_k = x_n - x_0$$

so that $x = x_n$ as required. □

Corollary 6.18. *The conjugate gradient algorithm, for any starting vector x_0, reaches the exact solution of (SLE) in at most n steps.*

Proof. Assume that the algorithm has not converged in $n-1$ steps; otherwise the proof is complete. Then the set $\{d_0, \cdots, d_{n-1}\}$ is A-orthogonal by Lemma 6.16. Hence the result follows from Theorem 6.17. □

Remark. When the computation is performed on a computer, rounding errors will cause $\hat{r}_n \neq 0$ where $\hat{r}_n$ is the calculated value for the residual error r_n. In practice one just treats the method as an iterative method and continues the iteration until the residual error $\|\hat{r}_k\|$ is small enough. Typically the exit criterion will even be reached for $k < n$.

In order to understand how the conjugate gradient method performs iteratively, we describe an error analysis based around the theory of polynomial approximation. In the following we let

$$\Lambda(A) := \{\, \lambda \in \mathbb{C} \mid \lambda \text{ is an eigenvalue of } A \,\}$$

and we let $\mathcal{P}_k$ denote the set of all polynomials of degree k with $p(0) = 1$.

Theorem 6.19. *If the conjugate gradient algorithm has not converged at step k, then*

$$\begin{aligned} \|x_k - x\|_A &= \inf_{p \in \mathcal{P}_k} \|p(A)(x_0 - x)\|_A \\ &\leq \Big(\inf_{p \in \mathcal{P}_k} \max_{\lambda \in \Lambda(A)} |p(\lambda)| \Big) \|x_0 - x\|_A. \end{aligned}$$

Proof. Since $x_k \in x_0 + \operatorname{span}\{r_0, Ar_0, \dots, A^{k-1} r_0\}$ we can find $\gamma_j \in \mathbb{C}$ with

$$e_k = x - x_k = x - x_0 + \sum_{j=0}^{k-1} \gamma_j A^j r_0 = e_0 + \sum_{j=0}^{k-1} \gamma_j A^{j+1} e_0.$$

Defining $p \in \mathcal{P}_k$ by

$$p(\lambda) = 1 + \sum_{j=1}^{k} \gamma_{j-1} \lambda^j$$

we obtain $e_k = p(A) e_0$ and thus

$$\|x_k - x\|_A = \|p(A)(x_0 - x)\|_A.$$

The value x_k minimises $\|x_k - x\|_A$ over $x_k \in \mathcal{K}_k(A, b)$, which is equivalent to minimising over all choices of γ_j. Hence

$$\|x_k - x\|_A = \inf_{p \in \mathcal{P}_k} \|p(A)(x_0 - x)\|_A.$$

To complete the proof let (ψ_j) be a basis of $\mathbb{C}^n$ composed of eigenvectors of A with corresponding eigenvalues λ_j. If we let $e_0 = \sum_{j=1}^n a_j \psi_j$, then

$$e_k = p(A) e_0 = \sum_{j=1}^{n} a_j p(\lambda_j) \psi_j$$

and thus we get

$$\|e_0\|_A^2 = \sum_{j=1}^{n} \lambda_j |a_j|^2, \qquad \|e_k\|_A^2 = \sum_{j=1}^{n} \lambda_j |a_j p(\lambda_j)|^2.$$

Therefore

$$\|e_k\|_A^2 \leq \max_{\lambda \in \Lambda(A)} |p(\lambda)|^2 \sum_{j=1}^{n} |a_j|^2 \lambda_j = \max_{\lambda \in \Lambda(A)} |p(\lambda)|^2 \|e_0\|_A^2$$

and the result follows. □

By choosing an appropriately scaled and shifted Chebyshev polynomial and applying the previous result, the following may be shown:

Theorem 6.20. *Let κ be the condition number of a symmetric positive definite matrix A in the Euclidean norm. Then the error in step k of the conjugate gradient algorithm for solving (SLE) satisfies*

$$\|x_k - x\|_A \leq 2 \left[\left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1} \right)^k + \left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1} \right)^{-k} \right]^{-1} \|x_0 - x\|_A.$$

Proof. See Exercise 6-6. □

Computational Complexity

We make the same assumptions as for the steepest descent method, and show the decrease in computational cost that can be obtained by using the conjugate gradient method.

Theorem 6.21. *Under Assumption 6.11 the computational cost of using the conjugate gradient method to achieve* (6.19) *is bounded above by* $cn^{\max\{\alpha,1\}+\frac{1}{2}\beta}\ln\varepsilon^{-1}$*, for some constant* c *independent of* n *and* ε*.*

Proof. The first item of the assumption ensures that each step of the iteration costs $\Theta(n^{\max\{\alpha,1\}})$. If $\sqrt{\kappa}\gg 1$ then the Theorem 6.20 gives

$$\|x_k - x\|_A \lesssim 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \|x_0 - x\|_A.$$

Combining the second and third items and using this expression gives the desired result. □

Remark. In an applied context the most important form of the conjugate gradient method is its *preconditioned form.* Roughly speaking the method is applied to the linear system

$$MAx = Mb$$

where the preconditioner M is chosen to try and improve the conditioning of MA over A, yet have M easy to invert. In practical terms the algorithm differs from standard conjugate gradient only by the addition of one multiplication with M per step.

Bibliography

Linear iterative methods for (SLE) are described in the book [Saa97]. Theorem 6.7 is proved in [SB02]. For discussion of Theorem 6.20 see [TB97]; for related results see [NW06]. The SOR method is analysed in detail in [LT88].

The nonlinear iterative methods that we discuss are constructed through minimisation or *optimisation* techniques. As such they are strongly related to the topic of optimisation in general. This topic is well overviewed in [NW06]; the exposition of the conjugate gradient method there forms the basis for ours. A good discussion of preconditioned versions of the conjugate gradient method may also be found in this book.

Exercises

Exercise 6-1. The purpose of this exercise is to introduce you to the idea of *iteration.* Many problems are simply too large to solve by the methods we have looked at so far — too much computer time or memory is required if the dimension n of the matrix is large. For linear systems

$$Ax = b, \quad A\in\mathbb{R}^{n\times n}, \quad b\in\mathbb{R}^n \tag{6.23}$$

this will lead to the idea of generating approximating sequences $\{x_k\}$ which *hopefully* converge to x as $k\to\infty$. For eigenvalue problems which are large we will study methods which generate approximations to a *subset* of the eigenvalues, and not to all of them. (Note that for eigenvalue problems it is necessary to use iterative methods in all cases, provided the matrix has dimension bigger than 4; the main difference when the problem is large is that we solve for only a subset of the eigenvalues).

(i) To show why iterative methods are necessary construct a matrix A with $n = 1000$ as follows: type

```
>> A=0.0005*rand(1000);
>> b=rand(1000,1);
>> for i=1:1000; A(i,i)=1; end;
```

This creates a matrix A with 1 on the diagonal and random entries uniformly distributed in $[0, 5 \times 10^{-4}]$ on the off-diagonals. Now try solving (6.23). Type

```
>> x=A\b;
```

Almost certainly your computer cannot handle a problem of this size. So type `ctrl c` to stop it from trying. (If you are lucky enough to be working on a machine where this problem is solved easily then repeat the exercise replacing 1000 by 10000 or by some matrix sufficiently large that MATLAB fails.) Keep the (original 1000×1000) matrix A for use in (iii) where we will use a more successful approach to solve this problem.
(ii) We look at the simplest iterative method, known as Jacobi. Any matrix A can be written uniquely as

$$A = D + L + U$$

where D is diagonal, L is zero on and above the diagonal and U is zero on and below the diagonal. Assuming that D is invertible, consider a sequence $\{x_k\}_{k=1}^{\infty}$ satisfying

$$x_{k+1} = D^{-1}[b - Lx_k - Ux_k].$$

If $x_k \to x^*$ as $k \to \infty$ then what equation does x^* satisfy and why?
(iii) Implement the idea in (ii) as follows. Type

```
>> L=tril(A,-1);
>> U=triu(A,1);
>> O=L+U;
```

This creates L and U as defined in (ii) (a MATLAB question: why is this so?). Type

```
>> j=1;
>> y=ones(1000,1);
>> n(j)=2*j;
>> for i=1:n(j); y=b-O*y; end
>> rn(j)=norm(A*y-b)
```

What does this tell you? Repeat the above 5 lines of code with $j = 2, 3, \ldots, 5$ and type:

```
>> plot(n,rn)
```

What do you observe? Also type

```
>> plot(n,log(rn))
```

(iv) Under what conditions on A will the Jacobi iteration converge to the correct solution of (6.23)?

Exercise 6-2. Prove that the Jacobi iteration that you implemented in Exercise 6-1 converges for the matrix A given there.

Exercise 6-3. Which of the following matrices are irreducible? For which of these matrices does the Jacobi-method converge?

$$\begin{pmatrix} 4 & 2 & 1 \\ & 2 & 1 \\ & & 1 \end{pmatrix}, \qquad \begin{pmatrix} 1 & 3 & \\ 3 & 25 & 4 \\ & 4 & 1 \end{pmatrix}, \qquad \begin{pmatrix} 8 & & \\ 4 & 2 & \\ 2 & 1 & 1 \end{pmatrix}$$

Exercise 6-4. Show that the Jacobi method for the solution of

$$\begin{pmatrix} 1 & 1 & \\ 1 & 10 & 2 \\ & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

converges and, for the iteration starting with $x_0 = 0$, give an upper bound on the number of steps required to get the relative error of the result below 10^{-6}.

Exercise 6-5. Consider the linear system $Ax = b$ where

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \varepsilon \end{pmatrix},$$

where $b = (1, \varepsilon)^T$. Calculate x_1 from the steepest descents algorithm, starting from $x_0 = 0$. Which component of the solution is better approximated after one step? Why is this?

Exercise 6-6. Prove Theorem 6.20 by choosing a Chebyshev polynomial scaled and shifted so that its min/max properties hold on the interval $[\lambda_{\min}(A), \lambda_{\max}(A)]$. Then prove that

$$\|x_k - x\|_A \leq 2\delta^k \|x_0 - x\|_A$$

where $\delta \in (0,1)$ and

$$\delta \approx 1 - 2\sqrt{\frac{\mu_{\min}}{\mu_{\max}}}$$

for $\kappa \gg 1$.

Exercise 6-7. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric positive-definite matrix. If $A = QMQ^T$ is the eigenvalue/eigenvector decomposition of A, then we define a square root of A (one of many possible square roots) to be $A^{1/2} = QDQ^T$, where D is a diagonal matrix whose elements are the square roots of the corresponding elements of M.

1. If $B \in \mathbb{R}^{n\times n}$ is a matrix such that B and $A^{1/2}$ commute, prove that

$$\|Bx\|_A \leq \|B\|_2 \|x\|_A, \qquad \forall x \in \mathbb{R}^n$$

2. Using your proof above, prove that

$$\inf_{p\in\mathcal{P}_k} \|p(A)y\|_A \leq \inf_{p\in\mathcal{P}_k} \max_{\lambda\in\Lambda(A)} |p(\lambda)| \|y\|_A$$

   where $\mathcal{P}_k$ is the set of all polynomials p of degree less than or equal to k with $p(0) = 1$, $y \in \mathbb{R}^n$, and $\Lambda(A)$ is the set of eigenvalues of A.

Exercise 6-8. Define

$$\varphi(x) := \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle$$

with A symmetric positive definite. Show that minimising $\varphi(x)$ is equivalent to solving $Ax = b$.

Consider the algorithm

$$x_{k+1} = x_k + \alpha_k r_k$$

to minimise $\varphi(x)$. Find α_k which minimises $\varphi(x_{k+1})$ over all $\alpha_k \in \mathbb{R}$ given x_k. This choice of α_k defines the *gradient* or *steepest descent* algorithm.

For CG we have

$$x_{k+1} = x_k + \alpha_k d_k.$$

Show that the choice of α_k made in Algorithm CG minimises $\varphi(x_{k+1})$ over all $\alpha \in \mathbb{R}$, given x_k and d_k.

Chapter 7

Least Squares Problems

In this chapter we introduce algorithms for the (LSQ) problem: given $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$ with $m \geq n$, find $x \in \mathbb{C}^n$ which minimises $\|Ax-b\|_2$. We will describe and discuss three different algorithms to solve (LSQ). The following theorem underpins all of the algorithms described.

Theorem 7.1. *A vector $x \in \mathbb{C}^n$ minimises $\|Ax-b\|_2$ for $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$, if and only if $Ax - b$ is orthogonal to* $\operatorname{range}(A)$.

Proof. Let $g(x) = \frac{1}{2}\|Ax-b\|_2^2$. Then minimising $\|Ax-b\|_2$ is equivalent to minimising the function g.

Assume that $Ax-b$ is orthogonal to $\operatorname{range}(A)$ and let $y \in \mathbb{C}^n$. Then

$$Ay - Ax = A(y-x) \perp Ax - b$$

and Pythagoras' theorem gives

$$\|Ay-b\|_2^2 = \|Ay-Ax\|_2^2 + \|Ax-b\|_2^2 \geq \|Ax-b\|_2^2$$

for all $y \in \mathbb{C}^n$. Thus x minimises g.

Now assume that the vector x minimises g. Then for every $y \in \mathbb{C}^n$ we have

$$0 = \frac{\partial}{\partial\lambda} g(x+\lambda y) = \frac{1}{2}\Big(\langle Ay, Ax-b\rangle + \langle Ax-b, Ay\rangle\Big) = \operatorname{Re}\langle Ax-b, Ay\rangle$$

and

$$0 = \frac{\partial}{\partial\lambda} g(x+\lambda i y) = \frac{1}{2}\Big(-i\langle Ay, Ax-b\rangle + i\langle Ax-b, Ay\rangle\Big) = -\operatorname{Im}\langle Ax-b, Ay\rangle.$$

This shows $\langle Ax-b, Ay\rangle = 0$ and thus $Ax - b \perp Ay$ for all $y \in \mathbb{C}^n$. □

Corollary 7.2. *A vector $x \in \mathbb{C}^n$ solves (LSQ) if and only if*

$$A^*Ax = A^*b. \tag{7.1}$$

Proof. From the theorem we know that x solves (LSQ) if and only if $Ax - b \perp \operatorname{range}(A)$. This in turn is equivalent to $Ax - b \perp a_i$ for every column a_i of A, i.e. to $A^*(Ax-b) = 0$. □

It is hence of interest to know whether A^*A is invertible. We assume throughout the remainder of this chapter that the singular values of A satisfy

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$$

and in this case A^*A is indeed invertible. Under this assumption, (LSQ) is solved by

$$x = A^\dagger b$$

where the pseudo-inverse $A^\dagger = (A^*A)^{-1}A^*$ is defined in Definition 3.6. The system (7.1) is called the *normal equations* for (LSQ).

Recall the definition of the 2-norm condition number for (LSQ) from Chapter 3. Note that this is large when A is close to being rank-deficient, so that σ_n is small. In many practical applications of (LSQ), such as arise naturally in statistics, A is indeed close to being rank-deficient. We will comment on this issue when we discuss the stability of the algorithms.

7.1 LSQ via Normal Equations

The Method

A natural approach to solution of the normal equations is as follows:

Algorithm (LSQ via normal equations).
input: $A \in \mathbb{C}^{m \times n}$ with $m \geq n$ and $\text{rank}(A) = n$, $b \in \mathbb{C}^m$
output: $x \in \mathbb{C}^n$ with minimal $\|Ax - b\|_2$
1: calculate A^*A and A^*b
2: solve $(A^*A)x = A^*b$

Computational Complexity

The method is usually implemented via standard algorithms for the matrix-matrix multiplication and matrix-vector multiplication, together with Cholesky factorisation for solution of the symmetric linear system.

Theorem 7.3 (Cost of Normal Equations for LSQ)**.** *The cost of solving (LSQ) by the Normal Equations approach is, in the usual implementation described above,*

$$C(m, n) \sim mn^2 + \frac{1}{3}n^3$$

as $m, n \to \infty$.

Proof. Calculating A^*A and A^*b requires asymptotically $\sim 2mn^2$ operations for $m, n \to \infty$. Since A^*A is symmetric we only need to calculate half of the entries, which can be done in $\sim mn^2$ operations. Calculation of A^*b costs $\Theta(mn)$. Solving $(A^*A)x = A^*b$ with Cholesky-factorisation requires $\sim \frac{1}{3}n^3$ operations. Combining these terms gives the result. $\square$

Of particular interest is the case $m \sim an$ as $n \to \infty$ for some constant $a \gg 1$, which arises frequently in practice. If $a \gg 1$ the cost is dominated by mn^2.

Error Analysis

One way that the error analysis proceeds is by proving that the computed x, $\hat{x}$, solves

$$(A^*A + \Delta A)\hat{x} = A^*b + c$$

and then using the backward error analysis for (SLE). Because $\kappa(A^*A) = \kappa(A)^2$ in the 2-norm it follows that

$$\frac{\|\Delta x\|_2}{\|x\|_2} = O\big(\kappa(A)^2 \epsilon_m\big).$$

In practice this method can perform very badly when the matrix A is close to being rank-deficient (so that σ_n is small by Lemma 3.8) and the 2-norm condition number of A is large.

7.2 LSQ via QR factorisation

The Method

To increase stability, at an increase in computational cost, the following QR based algorithm is often used in practice.

Algorithm (LSQ via QR factorisation).
input: $A \in \mathbb{C}^{m \times n}$ with $m \geq n$ and $\text{rank}(A) = n$, $b \in \mathbb{C}^m$
output: $x \in \mathbb{C}^n$ with minimal $\|Ax - b\|_2$
1: compute the reduced QR factorisation $A = \hat{Q}\hat{R}$
2: compute $\hat{Q}^*b \in \mathbb{C}^n$
3: solve $\hat{R}x = \hat{Q}^*b$ using back substitution

The result of the algorithm satisfies

$$\begin{aligned} A^*Ax &= A^*\hat{Q}\hat{R}x \\ &= A^*\hat{Q}\hat{Q}^*b \\ &= \hat{R}^*\hat{Q}^*\hat{Q}\hat{Q}^*b \\ &= \hat{R}^*\hat{Q}^*b \\ &= A^*b \end{aligned}$$

and thus solves (LSQ).

Computational Complexity

The method is usually implemented by means of Householder based QR factorisation, standard matrix-vector multiplication for $\hat{Q}^*b$ and standard triangular inversion of R. As for the normal equations approach, we assume that there is positive a, independent of n, such that $m \sim an$ as $n \to \infty$.

Theorem 7.4 (Cost of QR for LSQ)**.** *Let $s_n(A) > 0$. The cost of solving (LSQ) by the QR approach is, in the usual implementation,*

$$2mn^2 - \frac{2}{3}n^3 + \Theta(mn + n^2).$$

Proof. Steps 1 and 2 together have asymptotic cost $C_1(m,n) \sim 2mn^2 - \frac{2}{3}n^3 + \Theta(mn)$, and step 3 has asymptotic cost $C_2(m,n) = \Theta(n^2)$. Thus we get the desired result. □

If $a \gg 1$ then the cost is dominated by $2mn^2$ and is roughly twice that of the normal equations approach.

Error Analysis

The Householder QR approach has considerably improved backward stability properties when compared with the normal equations based solution method. In particular, the backward error is small in the following sense.

Theorem 7.5. *Let x solve the (LSQ) problem for (A,b) and let $\hat{x}$ be the computer solution via Householder QR factorisation. Then $\hat{x}$ minimises $\|(A+\Delta A)\hat{x} - (b+\Delta b)\|_2$ for matrix ΔA with columns Δa_j and vector Δb satisfying, for ε_{m} sufficiently small,*

$$\begin{aligned} \|\Delta a_j\|_2 &\le \frac{Cmn\varepsilon_{\mathrm{m}}}{1 - Cmn\varepsilon_{\mathrm{m}}}\|a_j\|_2, \quad j = 1, \dots, n \\ \|\Delta b\|_2 &\le \frac{Cmn\varepsilon_{\mathrm{m}}}{1 - Cmn\varepsilon_{\mathrm{m}}}\|b\|_2. \end{aligned}$$

In contrast to the normal equations method, this bound on the backward error is independent of the condition number of A. However, practitioners are often interested in the implied error in estimating the residual $r = b - Ax$. This can still behave poorly for the QR method, particularly when the matrix A is ill-conditioned.

7.3 LSQ via SVD

The Method

The most stable solution of (LSQ), especially useful for problems where A is close to being rank-deficient, and σ_n close to zero, is SVD based.

Algorithm (LSQ via SVD).
input: $A \in \mathbb{C}^{m\times n}$ with $m \geq n$ and $\operatorname{rank}(A) = n$, $b \in \mathbb{C}^m$
output: $x \in \mathbb{C}^n$ with minimal $\|Ax - b\|_2$

1: compute the reduced SVD $A = \hat{U}\hat{\Sigma}V^*$
2: compute $\hat{U}^*b \in \mathbb{C}^n$
3: solve $\hat{\Sigma}y = \hat{U}^*b$
4: return $x = Vy$

The result x of the calculation satisfies

$$A^*Ax = A^*\hat{U}\hat{\Sigma}V^*Vy = A^*\hat{U}\hat{U}^*b = A^*b$$

and thus it is the solution of (LSQ).

Computing the SVD

Assume that we are given $A \in \mathbb{C}^{m\times n}$. A common approach to computing the SVD is as follows:

- Step 1. Find the reduced QR factorisation $A = \hat{Q}\hat{R}$ with $\hat{Q} \in \mathbb{C}^{m\times n}$ and $\hat{R} \in \mathbb{C}^{n\times n}$.
- Step 2. Perform a bidiagonalisation of R, via orthogonal transformations as in Lemma 4.11, so that $B = U_1^*RV_1$ where $B \in \mathbb{C}^{n\times n}$ is upper-bidiagonal: has non-zeros only on the diagonal and on the first upper off-diagonal.
- Step 3. Find an SVD $B = U_2\Sigma V_2^*$ by use of Theorem 2.17.
- Step 4. Note that $A = (QU_1U_2)\Sigma(V_1V_2)^*$ and that hence $A = U\Sigma V^*$ with orthogonal U, V given by $U = QU_1U_2, V = V_1V_2$. (Note that the composition of orthogonal matrices is itself orthogonal).

It may be shown that the cost of this algorithm is $2mn^2 + 11n^3 + \Theta(mn + n^2)$.

Computational Complexity

Using these ideas is the basis for the standard implementation of SVD based approaches to (LSQ). As for the normal equations approach, we assume that there is a positive a, independent of n, such that $m \sim an$ as $n \to \infty$.

Theorem 7.6 (Cost of SVD for LSQ)**.** *Let $s_n(A) > 0$. The cost of solving (LSQ) by the SVD approach is, in the usual implementation,*

$$2mn^2 + 11n^3 + \Theta(mn + n^2).$$

Proof. (Sketch) SVD is $2mn^2 + 11n^3 + \Theta(mn + n^2)$.

$$\begin{aligned} \text{form } c = \hat{U}^*b : &\quad \Theta(mn) \\ \text{solve } \hat{\Sigma}w = c : &\quad \Theta(n) \\ \text{form } x = Vy : &\quad \Theta(n^2). \end{aligned}$$

□

If $a \gg 1$ the cost is similar to that of of the QR based method. But, for moderate values of a, the additional $11n^3$ term means that this method can be considerably more time-consuming than the QR method.

Error Analysis

The SVD approach is by far the most stable of the three methods for (LSQ). This is because it reveals the singular values which are close to zero, and hence possibly reflect rank-deficiency of A. Exploiting this fact, enables the method to perform well even in situations which are close to rank-deficient and, in particular, where σ_n is very close to zero.

Bibliography

The books [TB97] and [Dem97] both contain clear discussions of algorithms for (LSQ); the statements about complexity of the SVD, and SVD based methods for (LSQ) may be found in [TB97] for example. More details concerning the error analysis, and Theorem 7.5 in particular, can be found in [Hig02].

Exercises

Exercise 7-1. The purpose of this question is to introduce you to the content of this chapter. It is broken into three parts: (i) formulate a least squares problem; (ii) solve it blindly using Matlab; (iii) introduce the SVD, a major tool used in solving least squares problems.
(i) Let $\{(u^{(i)}, v^{(i)})\}_{i=1}^m$ be a set of points in the (u, v) plane. We wish to find the "best" straight line fit

$$v = \alpha u + \beta$$

to this data. The least squares approach to this problem is to find $(\alpha, \beta) \in \mathbb{R}^2$ minimising

$$G(\alpha, \beta) = \sum_{i=1}^{m} |\alpha u^{(i)} + \beta - v^{(i)}|^2.$$

Show that G may be written as

$$G(\alpha, \beta) = \|Ay - b\|_2^2$$

for $y = (\alpha, \beta)^T$ and some choice of $A \in \mathbb{R}^{m\times 2}$ and $b \in \mathbb{R}^m$ which you should specify.
(ii) Generate some data $\{(u^{(i)}, v^{(i)})\}_{i=1}^{100}$ in the following way: specify some `delta` $\in \mathbb{R}$ and type

```
>> u=rand(100,1);
>> v=2*u+ones(100,1)+delta*rand(100,1);
```

Construct A and b as defined in (i) and then form

```
>> x=A\b
```

This is the least squares solution. What would you expect the solution to be if `delta` is very small? Test your hypothesis by repeating the previous construction of A and b for a variety of small `delta`.
(iii) As we shall see, there are several competing methods to solve a least squares problem. One is based on the QR decomposition. A second is based on the SVD (*singular value decomposition*). To make the matrices shorter type `A=A(1:5,:)`. We will perform operations on this new shortened version of A (corresponding to 5 data points rather than 100.) Type

```
>> [U,X,V]=svd(A)
```

and discuss the properties of U, X and V (think about columns!). Now type

```
>> rn=norm(U*X*V'-A)
```

What does this tell you? Now type

```
>> [W,S,Z]=svd(A,0)
```

and compare W, S and Z with U, X and V. Again type

```
>> rn=norm(W*S*Z'-A)
```

What does this tell you?

The two factorisations are known (respectively) as the *full SVD* and the *reduced SVD* In our theoretical development we will mainly use the reduced SVD, and will refer to this as *the* SVD for short.

Exercise 7-2. You are given m data points $(u^{(i)}, v^{(i)})$ where $u^{(i)} \in \mathbb{R}^{n-1}$ and $v^{(i)} \in \mathbb{R}$ for $i = 1, \ldots, m$. We would like to find $\alpha \in \mathbb{R}^{n-1}$ and $\beta \in \mathbb{R}$ to minimise

$$\sum_{i=1}^{m} |\alpha^T u^{(i)} + \beta - v^{(i)}|^2.$$

Show that this problem may be reformulated as a standard least squares problem by specifying appropriate choices of $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Now let $n = 2$ and generate $m = 50$ random data points of the form

$$v^{(i)} = u^{(i)} + 10^{-2} W^{(i)}$$

where the $u^{(i)}$ are chosen at random uniformly from $[0, 100]$ and the $W^{(i)}$ are chosen at random from the standard normal distribution (with mean zero and variance one). Solve the resulting least squares problem by the QR method and by using the SVD (make use of Matlab's `qr` and `svd` routines). What do you expect the answer to be? Compare the efficiency and accuracy of the two different methods.

Exercise 7-3. Vandermonde matrices again: remember the command `vander` in Matlab.

a) Consider the *polynomial interpolation* problem. For each of the cases $l = 2, 4, 6, 8, 10$, generate a square Vandermonde matrix $B^{(l)}$ with the data points $\{u_j\}_{j=1}^{l}$ chosen from the set $\{1, 2, \ldots, l\}$. Estimate the condition numbers of the $B^{(l)}$ with the Matlab routine `cond`, and plot it as a function of l. What do you deduce from this exercise?

b) Now consider the problem of *polynomial data fitting*. Write down a least squares problem $B\alpha = v$ which determines the coefficients $\{\alpha_j\}_{j=1}^{n}$ in a polynomial

$$v(x) = \sum_{j=1}^{n} \alpha_j u^j$$

which best fits the data points $\{(u_j, v_j)\}_{j=1}^{n}$ in the sense of minimising

$$\sum_{i=1}^{m} |v(u_i) - v_i|^2.$$

The matrix B is a rectangular Vandermonde matrix. For each of the cases $l = 2, 4, 6, 8, 10$, generate the rectangular $B^{(l)}$ with $m = l$ and $n = l/2$. Choose the data points $\{u_j\}_{j=1}^{l}$ from the set $\{1, 2, \ldots, l\}$. Estimate the condition numbers of the $B^{(l)}$ with the Matlab routine `cond`, and plot it as a function of l. Discuss what you observe in relation to the first part of the question.

c) Now consider the data points

$$\begin{aligned} u_j &= j, \qquad j = 1, \ldots, 9 \\ u_{10} &= 9.0000001 \\ v_j &= 1 + u_j + u_j^4 + 10^{-2} W^{(j)} \end{aligned}$$

where the $W^{(j)}$ are independently chosen at random from the standard normal distribution (mean zero and variance one). Find the tenth degree polynomial interpolant to this data set, using QR factorisation. Then find the best fit polynomial of degree four, using QR to solve the resulting least squares problem. Compare the results you obtain from interpolation and best fit. Use graphical output to illustrate your discussion.

Exercise 7-4. Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite with eigenvalues

$$0 < \mu_1 \leq \mu_2 \leq \ldots \leq \mu_n.$$

1. Show that
$$\|A\|_2 = \mu_n \text{ and } \|A^{-1}\|_2 = \mu_1^{-1}.$$
What is the condition number of A in the Euclidean norm in this case?

2. Explicitly find δA solving

$$\min\left\{\frac{\|\delta A\|_2}{\|A\|_2} : A + \delta A \text{ is singular}\right\}.$$

   How big is $\|\delta A\|_2/\|A\|_2$ for this δA?

3. Let $B \in \mathbb{R}^{m\times n}$ with $m > n$. State and derive the *normal equations* for the solution of the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Bx - b\|_2$$

   where $b \in \mathbb{R}^m$.

4. If B has singular values $\{\sigma_i\}_{i=1}^n$ then what is the Euclidean norm condition number of the matrix which must be inverted to solve the normal equations?

Exercise 7-5. Show that the least squares solution will yield $Ax = b$ if and only if $b = AA^\dagger b$ where $A^\dagger$ is the Moore-Penrose inverse of A.

Chapter 8

Eigenvalue Problems

In this chapter we will study methods for solution of the eigenvalue problem (EVP): given $A \in \mathbb{C}^{n\times n}$, compute the eigenvalues and eigenvectors of A.

A first insight is that finding eigenvalues and finding eigenvectors are equivalent problems. To find an eigenvalue from an eigenvector we proceed as follows: Given an eigenvector $x \in \mathbb{C}^n$ we try to find the $\alpha \in \mathbb{C}$ which minimises $\|\alpha x - Ax\|_2$. Since x is an eigenvector, the minimum of 0 is attained for the corresponding eigenvalue. It is easy to check, for example using Corollary 7.2, that the optimal choice of α is

$$\alpha = (x^*x)^{-1}x^*(Ax) = \frac{\langle x, Ax\rangle}{\langle x, x\rangle}.$$

Note that solvability of the normal equations in this case is guaranteed, provided that $x \neq 0$. The preceding considerations show that the following definition provides a natural way to estimate an eigenvalue from an estimate of an eigenvector.

Definition 8.1. The *Rayleigh quotient* of a matrix $A \in \mathbb{C}^{n\times n}$ is defined by

$$r_A(x) = \frac{\langle x, Ax\rangle}{\langle x, x\rangle}$$

for all $x \in \mathbb{C}^n$.

Using this notation, we get that

$$Ax = r_A(x)x$$

for all eigenvectors x of A.

Calculating an eigenvector x from an eigenvalue λ can be achieved by fixing one component of the vector x, say x_l, to be 1 and then solving $n-1$ of the linear equations $Ax = \lambda x$, with λ and x_l given. If the eigenvectors corresponding to λ have $x_l = 0$ then this system of linear equations will have no solution but, since $x \neq 0$, there will be an $l \in \{1, \ldots, n\}$ such that the system can be solved and the solution will be an eigenvector with eigenvalue λ.

Theorem 1.12 shows that, if we can find the roots of arbitrary polynomials, we can also find the eigenvalues of arbitrary matrices. In fact the two problems are equivalent: for every polynomial we can find a matrix with this polynomial as the characteristic polynomial. To see this let $p(z) = (-1)^n(c_0 + c_1 z + \cdots + c_{n-1}z^{n-1} + z^n)$ be given. Define the *companion matrix* $C \in \mathbb{C}^{n\times n}$ of the polynomial p by

$$C = \begin{pmatrix} 0 & & & & -c_0 \\ 1 & & & & -c_1 \\ & \ddots & & & \vdots \\ & & \ddots & & -c_{n-2} \\ & & & 1 & -c_{n-1} \end{pmatrix}. \tag{8.1}$$

One can show that $\rho_C(z) = \det(C - zI) = p(z)$ (see Exercise 8-8). Hence finding eigenvalues of C is equivalent to the problem of finding the roots of the polynomial p. This establishes equivalence between the eigenvalue problem and root finding for polynomials. Regarding this latter problem we know the following:

Theorem 8.2 (Abel, 1824)**.** *For every $n \geq 5$ there is a polynomial p of degree n with rational coefficients that has a real root which cannot be expressed by only using rational numbers, addition, subtraction, multiplication, division and taking kth roots.*

Any possible algorithm will be based on the operations mentioned in the theorem and hence it is not possible to find an algorithm which calculates eigenvalues exactly after a finite number of steps. The conclusion to be drawn from this is that any eigenvalue solver must be iterative: aiming to approximate the solution of the eigenvalue problem in a finite number of steps. We describe a variety of iterative methods for (EVP) in the following sections.

In the remainder of this chapter we will consider the Hermitian eigenvalue problem. As we have seen, if $A \in \mathbb{C}^{n\times n}$ is Hermitian, the matrix has an orthonormal system of eigenvectors and the corresponding eigenvalues are real. In this chapter $x_1, \ldots, x_n \in \mathbb{C}^n$ will always denote an orthonormal system of eigenvectors of A and $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ will be the corresponding eigenvalues. We order the eigenvalues such that

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0.$$

Since our methods will be iterative, the algorithms will only compute approximate eigenvectors and eigenvalues. The following theorem shows, that the Rayleigh quotient of an approximate eigenvector is a good approximation for the corresponding eigenvalue.

Theorem 8.3. *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian and $x \in \mathbb{R}^n \backslash \{0\}$.*
a) x is an eigenvector of A with eigenvalue λ if and only if $\nabla r_A(x) = 0$ and $r_A(x) = \lambda$.
b) If x is an eigenvector of A, then

$$\big|r_A(x) - r_A(y)\big| = \mathcal{O}\big(\|x - y\|_2^2\big)$$

as $y \to x$.

Proof. a) The gradient of r_A is

$$\nabla r_A(x) = \frac{2}{\|x\|_2}\Big(Ax - r_A(x) \cdot x\Big).$$

Assume $Ax = \lambda x$. Then $r_A(x) = \langle x, \lambda x\rangle / \langle x, x\rangle = \lambda$ and

$$\nabla r_A(x) = \frac{2}{\|x\|_2^2}\big(\lambda x - \lambda x\big) = 0.$$

Note that $\nabla r_A(x) \to \infty$ as $x \to 0$. Thus, if $\nabla r_A(x) = 0$, then $Ax - r_A(x)x = 0$ and $x \neq 0$ so that $(x, r_A(x))$ is an eigenpair for A.

b) From part a) we know $\nabla r_A(x) = 0$. Taylor expansion of r_A around x gives

$$\begin{aligned} r_A(y) &= r_A(x) + \langle \nabla r_A(x), y - x\rangle + \mathcal{O}\big(\|y - x\|_2^2\big) \\ &= r_A(x) + \mathcal{O}\big(\|y - x\|_2^2\big). \end{aligned}$$

□

Many of the methods for finding eigenvalues rely on repeated matrix-vector multiplication by A. Thus they will be considerably cheaper if A is preprocessed so that it has as many zeros as possible, but is similar to the original matrix. This can be achieved by reducing A to Hessenberg form using Lemma 4.12. When A is Hermitian, the Hessenberg form is necessarily tridiagonal. Hence tridiagonal matrices play an important role in what follows.

8.1 The Power Method

The Method

The key idea of the power method is that, assuming some minor technical conditions, repeated application of A to an arbitrary starting vector will lead to a sequence of vectors which eventually align with the eigenvector associated with the largest eigenvalue.

Algorithm PI (power iteration).
input: $A \in \mathbb{C}^{n\times n}$ Hermitian
output: $z^{(k)} \in \mathbb{C}^n$, $\lambda^{(k)} \in \mathbb{R}$ with $z^{(k)} \approx x_1$ and $\lambda^{(k)} \approx \lambda_1$

choose $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 = 1$
for $k = 1, 2, 3, \ldots$ **do**
  $w^{(k)} = Az^{(k-1)}$
  $z^{(k)} = \dfrac{w^{(k)}}{\|w^{(k)}\|_2}$
end for
$\lambda^{(k)} = r_A(z^{(k)})$

Remarks.

1. As with all iterative methods, the method must be stopped at some point where the result is "close enough" to the real one.

2. The algorithm calculates an approximate eigenvector as $z^{(k)} = A^k z^{(0)}/\|A^k z^{(0)}\|_2$. To avoid overflow/underflow errors the vector $z^{(k)}$ is normalised in every step of the iteration.

3. The method is based on the following idea: if we express $z^{(0)}$ in the basis $x_1, \ldots, x_n$ we get
$$z^{(0)} = \sum_{i=1}^{n} \alpha_i x_i$$
and
$$A^k z^{(0)} = \sum_{i=1}^{n} \alpha_i A^k x_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k x_i.$$
For large k this expression is dominated by the term corresponding to the eigenvalue with the largest modulus.

Error Analysis

Since eigenvectors are only determined up to scalar factors, we cannot expect the approximation $z^{(k)}$ to be close to the vector x_1 we chose. Instead, we would expect $z^{(k)}$ to be close to σx_1 where σ is a factor which might depend on k. We get the following result.

Theorem 8.4. *Let $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ and $\langle x_1, z^{(0)}\rangle \neq 0$. Then there is a sequence $(\sigma^{(k)})_{k\in\mathbb{N}}$ in $\mathbb{C}$ with $|\sigma^{(k)}| = 1$ for all $k \in \mathbb{N}$ such that the sequences $(z^{(k)})$ and $(\lambda^{(k)})$ from the power iteration algorithm satisfy*
$$\big\|z^{(k)} - \sigma^{(k)} x_1\big\|_2 = \mathcal{O}\Big(\Big|\frac{\lambda_2}{\lambda_1}\Big|^k\Big) \tag{8.2}$$
and
$$\big|\lambda^{(k)} - \lambda_1\big| = \mathcal{O}\Big(\Big|\frac{\lambda_2}{\lambda_1}\Big|^{2k}\Big) \tag{8.3}$$
as $k \to \infty$.

Proof. a) We can write $z^{(0)}$ as

$$z^{(0)} = \sum_{i=1}^{n} \alpha_i x_i, \quad \alpha_i = \langle x_i, z^{(0)} \rangle.$$

Since $\alpha_1 = \langle x_1, z^{(0)} \rangle \neq 0$, we get

$$A^k z^{(0)} = \sum_{i=1}^{n} \alpha_i \lambda_i^k x_i = \alpha_1 \lambda_1^k \Big(x_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \Big(\frac{\lambda_i}{\lambda_1}\Big)^k x_i \Big)$$

and Pythagoras' Theorem gives

$$\big\| A^k z^{(0)} \big\|_2^2 = |\alpha_1 \lambda_1^k|^2 \Big(1 + \sum_{i=2}^{n} \Big|\frac{\alpha_i}{\alpha_1}\Big|^2 \Big|\frac{\lambda_i}{\lambda_1}\Big|^{2k} \Big) = |\alpha_1 \lambda_1^k|^2 (1 + \gamma_k)$$

where

$$\gamma_k = \sum_{i=2}^{n} \Big|\frac{\alpha_i}{\alpha_1}\Big|^2 \Big|\frac{\lambda_i}{\lambda_1}\Big|^{2k} \leq \frac{\|z^{(0)}\|_2^2}{|\alpha_1|^2} \Big|\frac{\lambda_2}{\lambda_1}\Big|^{2k}.$$

Thus we can conclude

$$z^{(k)} = \frac{A^k z^{(0)}}{\|A^k z^{(0)}\|_2} = \frac{\alpha_1 \lambda_1^k}{|\alpha_1 \lambda_1^k|} \Big(x_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \Big(\frac{\lambda_i}{\lambda_1}\Big)^k x_i \Big) \frac{1}{\sqrt{1+\gamma_k}}.$$

Now define $\sigma^{(k)} = \alpha_1 \lambda_1^k / |\alpha_1 \lambda_1^k|$. Then

$$\Big\| \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} - \sigma^{(k)} x_1 \Big\|_2^2 = \Big\| \frac{A^k z^{(0)}}{\alpha_1 \lambda_1^k} - x_1 \Big\|_2^2 = \sum_{i=2}^{n} \big(\frac{\alpha_i}{\alpha_1}\big)^2 \big(\frac{\lambda_i}{\lambda_1}\big)^{2k} = \gamma_k$$

and thus

$$\begin{aligned}
\Big\| z^{(k)} - \sigma^{(k)} x_1 \Big\|_2 &\leq \Big\| \frac{A^k z^{(0)}}{\|A^k z^{(0)}\|_2} - \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} \Big\|_2 + \Big\| \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} - \sigma^{(k)} x_1 \Big\|_2 \\
&= \sqrt{1+\gamma_k} - 1 + \sqrt{\gamma_k} \\
&\leq 2\sqrt{\gamma_k} \\
&\leq 2 \frac{\|z^{(0)}\|_2}{|\alpha_1|} \Big|\frac{\lambda_2}{\lambda_1}\Big|^k.
\end{aligned}$$

This completes the proof of (8.2).

b) Using part b) of Theorem 8.3 and (8.2) we get

$$\big|\lambda^{(k)} - \lambda_1\big| = \big|r_A(z^{(k)}) - r_A(\sigma^{(k)} x_1)\big| = \mathcal{O}\big(\|z^{(k)} - \sigma^{(k)} x_1\|_2^2\big) = \mathcal{O}\Big(\Big|\frac{\lambda_2}{\lambda_1}\Big|^{2k}\Big).$$

This completes the proof of (8.3). ◻

Computational Complexity

For simplicity assume that our primary interest is in the accuracy of eigenvectors. We wish to iterate until

$$\|z^{(k)} - \sigma^{(k)} x_1\|_2 \leq \varepsilon. \tag{8.4}$$

Inspection of the preceding proof shows that this inequality will be attained provided

$$2\Big(\frac{\|z^{(0)}\|_2}{\alpha_1}\Big) \Big|\frac{\lambda_2}{\lambda_1}\Big|^k \leq \varepsilon. \tag{8.5}$$

Hence it suffices to choose k to be the smallest integer greater than

$$k^{\#} = \frac{\ln \varepsilon^{-1} + \ln \|z^{(0)}\|_2 + \ln 2 - \ln \alpha_1}{\ln(\lambda_1/\lambda_2)}.$$

Assumption 8.5. The first and second eigenvalues of A satisfy $\frac{\lambda_1}{\lambda_2} = 1 + \Theta(n^{-\beta})$.

Theorem 8.6. *Under Assumption 8.5 and the assumptions of Theorem 8.4, the computational cost of using the power method to achieve* (8.4) *is bounded above by* $c\Big(n^{1+\beta}\ln\varepsilon^{-1} + n^3\Big)$, *for some constant* c *independent of* n *and* ε.

Proof. Because of $\|z^{(0)}\|_2 = 1$, the quantities $\|z^{(0)}\|$ and α_1 are independent of n. The cost of reducing A to Hessenberg (tridiagonal) form is $\Theta(n^3)$ (see Lemma 4.12). Each step of the iteration then costs $\Theta(n)$, since matrix vector-multiplication and inner-products are both $\Theta(n)$ when the matrix tridiagonal. Combining the second and third items gives a bound of $\Theta(n^\beta \ln\varepsilon^{-1})$ for the number of iterations. Using (8.5) gives the desired result for the largest eigenvalue and corresponding eigenvector of the tridiagonal matrix. Transforming back to the original matrix A costs $\Theta(n^2)$ and is hence negligible relative to the cost of transforming to Hessenberg form. The transformation is an orthogonal one and hence does not change the 2-norm of the error in the eigenvector. □

8.2 Inverse Iteration

The Method

The power iteration algorithm helps to find the eigenvalue with the largest modulus and the corresponding eigenvector. The following simple idea leads to a modification of the power iteration method to find different eigenvalues of A.

Given $\lambda \in \mathbb{R}$, the matrix $(A - \lambda I)^{-1}$ has eigenvalues $\mu_i = 1/(\lambda_i - \lambda)$ for $i = 1, \ldots, n$. The power iteration method applied to this matrix will thus return an approximation for the eigenvector corresponding to the μ_j with the biggest modulus. This is the μ_j where λ_j is the eigenvalue closest to λ. The resulting algorithm is called the *inverse iteration* method.

Algorithm II (inverse iteration).
input: $A \in \mathbb{C}^{n\times n}$ Hermitian, $\lambda \in \mathbb{R}$
output: $z^{(k)} \in \mathbb{C}^n$, $\lambda^{(k)} \in \mathbb{R}$ with $z^{(k)} \approx x_j$ and $\lambda^{(k)} \approx \lambda_j$
    where λ_j is the eigenvalue closest to λ

choose $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 = 1$
for $k = 1, 2, 3, \ldots$ **do**
  solve $(A - \lambda I)w^{(k)} = z^{(k-1)}$
  $z^{(k)} = \dfrac{w^{(k)}}{\|w^{(k)}\|_2}$
end for
return $\lambda^{(k)} = r_A(z^{(k)})$.

Remark. For practical usage the method is stopped at some point where the result is "close enough" to the real one.

Error Analysis

The following convergence result is a direct consequence of Theorem 8.4.

Theorem 8.7 (Convergence of Inverse Iteration)**.** *Let* $A \in \mathbb{C}^{n\times n}$ *be Hermitian and* $\lambda \in \mathbb{R}$. *Suppose* $a, b \in \{1, \ldots, n\}$ *are such that* $|\lambda_a - \lambda| < |\lambda_b - \lambda| \le |\lambda_j - \lambda|$ *for all* $j \in \{1, \ldots, n\} \setminus \{a, b\}$. *Then, if* $\langle z^{(0)}, x_a\rangle \ne 0$, *there exists a sequence* $(\sigma^{(k)})_{k\in\mathbb{N}}$ *with* $|\sigma^{(k)}| = 1$ *such that*

$$\|z^{(k)} - \sigma^{(k)} x_a\|_2^2 + |\lambda^{(k)} - \lambda_a| = \mathcal{O}\left(\left|\frac{\lambda - \lambda_a}{\lambda - \lambda_b}\right|^{2k}\right).$$

Computational Complexity

The computational cost is analogous to that of the power iteration, under appropriate assumptions, found by translating the theorem for the power method, where we iterate A, to the inverse iteration situation where we iterate $(A - \lambda I)^{-1}$. Because, after preprocessing, A is tridiagonal the cost of each step of inverse iteration is the same as for the power method.

8.3 Rayleigh Quotient Iteration

The Method

For inverse iteration it is necessary to estimate the eigenvalue in advance. An alternative is to estimate the eigenvalue during the course of the algorithm:

Algorithm RQI (Rayleigh quotient iteration).
input: $A \in \mathbb{C}^{n\times n}$ symmetric
output: $z^{(k)} \in \mathbb{R}^n$, $\lambda^{(k)} \in \mathbb{R}$ with $z^{(k)} \approx x_j$ and $\lambda^{(k)} \approx \lambda_j$ for some $j \in \{1, \cdots, n\}$

1: choose $z^{(0)}$ such that $\|z^{(0)}\|_2 = 1$, $\lambda^{(0)} \in \mathbb{R}$
2: **for** $k = 1, 2, \dots$ **do**
3: $\quad [A - \lambda^{(k-1)} I] w^{(k)} = z^{(k-1)}$
4: $\quad z^{(k)} = w^{(k)} / \|w^{(k)}\|_2$
5: $\quad \lambda^{(k)} = \dfrac{\langle z^{(k-1)}, w^{(k)}\rangle + \lambda^{(k-1)} \|w^{(k)}\|_2^2}{\|w^{(k)}\|_2^2}$
6: **end for**

Notice that $\lambda^{(k)}$ in step 5 is simply $r_A(w^{(k)})$, computed without application of A.

Error Analysis

The previous two algorithms both converged *linearly*: asymptotically, the error decreases by a constant factor at each step. In contrast, when the Rayleigh quotient iteration converges to an eigenvalue, say λ_j, it does so *cubically*:

$$|\lambda^{(k+1)} - \lambda_j| = \mathcal{O}(|\lambda^{(k)} - \lambda_j|^3).$$

A similar cubic rate of convergence is also obtained for the eigenvector. Such methods can hence be very efficient.

Computational Complexity

However, with iterations of this type, where the convergence is not linear (as with the Newton method for example), it is difficult to predict when it will converge, and to which eigenvalue/eigenvector pair it will converge. Much of the computational complexity of such algorithms stems from choosing starting points which will lead to convergence to the desired pair.

8.4 Simultaneous Iteration

The Method

Now imagine that we wish to calculate all the eigenvectors. A natural idea is to take

$$z^{(0)} \in \mathbb{R}^{n\times n}$$

orthonormal, and generate

$$Z^{(k+1)} = AZ^{(k)}.$$

Then each column of $Z^{(k)}$, $z_i^{(k)}$, is calculated as

$$z_i^{(k+1)} = Az_i^{(k)}.$$

Let

$$z_i^{(0)} = \sum_{j=1}^{n} \alpha_{j,i} x_j$$

and assume that

$$\alpha_{i,i} = \langle z_i^{(0)}, x_i \rangle \neq 0, \quad i = 1, \dots, n \tag{8.6}$$

Now

$$z_i^{(k)} = \sum_{j=1}^{n} \alpha_{j,i} \lambda_j^k x_j.$$

If

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_{n-1}| > |\lambda_n| \tag{8.7}$$

then

$$\begin{aligned}
\operatorname{span}\{z_1^{(k)}\} &= \operatorname{span}\{y_1\}, \quad \|y_1\|_2 = 1, \quad y_1 = x_1 + e_1 \\
\operatorname{span}\{z_1^{(k)}, z_2^{(k)}\} &= \operatorname{span}\{y_1, y_2\}, \quad \|y_2\|_2 = 1, \quad y_2 = x_2 + e_2 \\
&\vdots \\
\operatorname{span}\{z_1^{(k)}, z_2^{(k)}, \dots, z_n^{(k)}\} &= \operatorname{span}\{y_1, y_2, \dots, y_n\}, \quad \|y_n\|_2 = 1, \quad y_n = x_n + e_n,
\end{aligned}$$

where

$$e_j = \sum_{l=1}^{\min\{j,n-1\}} \mathcal{O}\left(\left| \frac{\lambda_{l+1}}{\lambda_l} \right|^k \right) = \mathcal{O}(\xi)$$

for

$$\xi := \max_{1 \le l \le n-1} \frac{\lambda_{l+1}}{\lambda_l}.$$

However, in practice this is a poor algorithm, not only because of overflow / underflow if $|\lambda_1| \neq 1$, but also because the approximate basis $\{z_1^{(k)}, z_2^{(k)}, \dots, z_n^{(k)}\}$ for $\{x_1, \dots, x_n\}$ is highly *ill-conditioned*, here meaning that the vectors nearly all point in the same direction, x_1.

To overcome these problems the algorithm used in practice forms an orthonormal basis from the approximate basis for $\{x_1, \dots, x_n\}$, by means of QR.

Algorithm (simultaneous iteration).
input: $A \in \mathbb{R}^{n \times n}$ symmetric
output: $Z^{(k)}, \Lambda^{(k)} \in \mathbb{R}^{n \times n}$ with $Z^{(k)} \approx (x_1, x_2, \dots, x_n)$ and $\Lambda_{ii}^{(k)} \approx \lambda_i$ for $i = 1, \dots, n$

```
choose an orthogonal matrix Z^(0) ∈ R^{n×n}
for k = 1, 2, 3, ... do
    W^(k) = A Z^(k-1)
    calculate the QR factorisation W^(k) = Z^(k) R^(k)
    Λ^(k) = (Z^(k-1))^T W^(k)
end for
```

Here $Z^{(k)}R^{(k)}$ is the QR factorisation of $W^{(k)}$. In the following we write $Z^{(k)}$ and $W^{(k)}$ in terms of columns:

$$Z^{(k)} = (z_1^{(k)}, \dots, z_n^{(k)}), \qquad W^{(k)} = (w_1^{(k)}, \dots, w_n^{(k)}).$$

We show below that the diagonal entries of $\Lambda^{(k)}$ are $r_A(z_i^{(k-1)})$.

Error Analysis

Theorem 8.8 (Convergence of Simultaneous Iteration)**.** *Let $A \in \mathbb{R}^{n\times n}$ be symmetric, satisfying* (8.7) *so that $\xi < 1$. Under assumption* (8.6) *there are, for $j = 1, \ldots, n$, sequences $\{s_j^{(k)}\}_{k\in\mathbb{Z}^+}$, $s_j^{(k)} \in \{-1, +1\}$, such that*

$$\|z_j^{(k)} - s_j^{(k)} x_j\|_2^2 + |\Lambda_{jj}^{(k)} - \lambda_j| = \mathcal{O}(|\xi|^{2k}).$$

Proof. By the discussion preceding the algorithm the result follows for the approximate eigenvectors. Since

$$\begin{aligned}\Lambda_{jj}^{(k)} &= \langle z_j^{(k-1)}, w_j^{(k)} \rangle = \langle z_j^{(k-1)}, A z_j^{(k-1)} \rangle \\ &= r_A(z_j^{(k-1)}),\end{aligned}$$

the convergence to the eigenvalues follows by Theorem 8.3. □

Computational Complexity

In practice the simultaneous iteration is usually implemented in the form known as the QR algorithm, which we discuss in the next section. Hence we discuss complexity in that context.

8.5 The QR Algorithm for Eigenvalues

The Method

In practice the eigenvalues of A can be found directly by a method related to simultaneous iteration, and which we now derive. The resulting algorithm appears almost magical upon first sight, but when related to simultaneous iteration, its mechanism is clear. To establish this relation let $Z^{(k-1)}$, $Z^{(k)}$, $R^{(k)}$, $W^{(k)}$ and $\Lambda^{(k)}$ be as in the simultaneous iteration algorithm and define

$$Q^{(k)} := (Z^{(k-1)})^T Z^{(k)}.$$

This is an orthogonal matrix by construction.

Recall that

$$\Lambda^{(k)} = (Z^{(k-1)})^T A Z^{(k-1)}.$$

Thus

$$\begin{aligned}Q^{(k)} R^{(k)} &= (Z^{(k-1)})^T Z^{(k)} (Z^{(k)})^T W^{(k)} \\ &= (Z^{(k-1)})^T A Z^{(k-1)} \\ &= \Lambda^{(k)}.\end{aligned}$$

Furthermore

$$\begin{aligned}R^{(k)} Q^{(k)} &= (Z^{(k)})^T W^{(k)} (Z^{(k-1)})^T Z^{(k)} \\ &= (Z^{(k)})^T A Z^{(k)} \\ &= \Lambda^{(k+1)}.\end{aligned}$$

Thus reversing the order of Q and R results in an improved estimate of the eigenvalues. This suggests the following algorithm for locating eigenvalues of A.

QR for eigenvalues input: $A \in \mathbb{R}^{n\times n}$ symmetric
output: $\Lambda \in \mathbb{R}^{n\times n}$ with diagonal entries approximating the eigenvalues

Set $\Lambda^{(0)} = A$
for $k = 1, 2, \ldots$ **do**
  calculate the QR factorisation $\Lambda^{(k-1)} = Q^{(k-1)} R^{(k-1)}$
  $\Lambda^{(k)} = R^{(k-1)} Q^{(k-1)}$
end for

Error Analysis

Theorem 8.9 (Convergence of QR)**.** *Under the assumptions of Theorem 8.8, the QR eigenvalue algorithm satisfies (possibly after reordering of eigenvalues)*

$$\|\Lambda_{ii}^{(k)} - \lambda_i\|_2 = \mathcal{O}\big(|\xi|^{2k}\big), \quad \forall i = 1, \cdots, n.$$

Proof. The algorithm produces the same $\Lambda^{(k)}$ as simultaneous iteration. So the diagonal behaviour follows from Theorem 8.8. $\square$

For the off-diagonals, note that by orthogonality of the x_l

$$\begin{aligned}\Lambda_{jl}^{(k)} = \langle z_j^{(k)}, Az_l^{(k)}\rangle &= \langle x_j, Ax_l\rangle + \mathcal{O}(|\xi|^k)\\ &= \langle x_j, \lambda_l x_l\rangle + \mathcal{O}(|\xi|^k)\\ &= \lambda_l\langle x_j, x_l\rangle + \mathcal{O}(|\xi|^k)\\ &= \mathcal{O}(|\xi|^k).\end{aligned}$$

Thus the matrix Λ becomes diagonal as $k \to \infty$.

Computational Complexity

Making the error in the QR method of size ε gives the number of iterations $k = \mathcal{O}(\frac{\ln\varepsilon}{\ln\xi})$. Under the assumptions of Theorem 8.8, the QR algorithm as stated would cost

$$\mathcal{O}\left(\frac{\ln\varepsilon}{\ln\xi}n^3\right)$$

work to obtain estimates of all eigenvalues to within $\mathcal{O}(\varepsilon)$. However, this can be improved to

$$\mathcal{O}\left(\frac{\ln\varepsilon}{\ln\xi}n^2\right) + \mathcal{O}(n^3),$$

with second term independent of ε, ξ as follows.

Recall from Definition 1.8 that a matrix $A \in \mathbb{C}^{n\times n}$ is upper Hessenberg if

$$a_{ij} = 0 \quad i > j+1.$$

Lemma 8.10. *If $\Lambda^{(0)}$ is upper Hessenberg, the QR Algorithm generates $\Lambda^{(k)}$ which are upper Hessenberg.*

Proof. We have

$$\begin{aligned}\Lambda^{(k)} &= R^{(k-1)}Q^{(k-1)}\\ &= R^{(k-1)}\Lambda^{(k-1)}(R^{(k-1)})^{-1}.\end{aligned}$$

But $R^{(k-1)}$ and its inverse are upper triangular and pre- or post-multiplication by upper triangular matrices preserves the upper Hessenberg form. $\square$

Theorem 8.11 (Cost of QR)**.** *Under the assumptions of Theorem 8.8 the QR algorithm can be implemented to give eigenvalue approximations of size ε with cost*

$$C(n,\varepsilon) \le k_1\frac{\ln\varepsilon}{\ln\xi}n^2 + k_2 n^3,$$

where k_1, k_2 are independent of n, ε and ξ.

Proof. (Sketch) First find T as in Lemma 4.12, at cost $\Theta(n^3)$. Then apply the QR algorithm to T. Since each QR factorisation is on an upper Hessenberg matrix, the cost reduces to $\Theta(n^2)$. The number of iterations required to reduce the error to size ε is bounded by $\mathcal{O}(\ln\varepsilon/\ln\xi)$ and the result follows. $\square$

8.6 Divide and Conquer for Symmetric Problems

We conclude the chapter with a somewhat different method for eigenvalue calculation, specifically constructed for symmetric tridiagonal systems. It uses a divide and conquer approach, as introduced for the Strassen algorithm.

The Method

If $A \in \mathbb{R}^{n\times n}$ is symmetric then the reduction to upper Hessenberg form of Lemma 4.12 yields tridiagonal T in $\Theta(n^3)$ operations. Thus we consider finding the eigenvalues of a symmetric tridiagonal T. By expressing T in terms of two smaller tridiagonal matrices of half the size, we create the basis for a divide and conquer approach for the symmetric eigenvalue problem. Consider a tridiagonal T in the form:

$$T = \left(\begin{array}{cccc|cccc}
a_1 & b_1 & & & & & & \\
b_1 & \ddots & \ddots & & & & & \\
 & \ddots & \ddots & b_{n/2-1} & & & & \\
 & & b_{n/2-1} & a_{n/2} & b_{n/2} & & & \\
\hline
 & & & b_{n/2} & a_{n/2+1} & b_{n/2+1} & & \\
 & & & & b_{n/2+1} & \ddots & \ddots & \\
 & & & & & \ddots & \ddots & b_{n-1} \\
 & & & & & & b_{n-1} & a_n
\end{array}\right).$$

The basis of the divide and conquer approach is to split T into two separate tridiagonal matrices, each of whose eigenvalues and eigenvectors determine those of T. This can be done exactly if b is zero, and the following splitting of T tries to build on this fact.

Let $n = 2^k$, $b = b_{n/2}$ and write

$$T = \left(\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right) + b(v \otimes v), \tag{8.8}$$

where $v = (e_{n/2}^T | e_1^T)^T$, with $e_1, e_{n/2} \in \mathbb{R}^{n/2}$, and T_1, T_2 tridiagonal. In fact,

$$\begin{aligned}
(T_1)_{ij} &= T_{ij} \quad \text{provided } (i,j) \neq (n/2, n/2) \\
(T_1)_{n/2\, n/2} &= T_{n/2\, n/2} - b \\
(T_2)_{ij} &= T_{i+n/2, j+n/2} \quad \text{provided } (i,j) \neq (1,1) \\
(T_2)_{11} &= T_{1+n/2, 1+n/2} - b.
\end{aligned}$$

How do we find the eigenvalues of T from those of T_1 and T_2? Since T_1, T_2 are real symmetric,

$$T_i = Q_i \Lambda_i Q_i^T, \qquad i = 1, 2.$$

Let

$$\begin{aligned}
D &= \begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} = \operatorname{diag}(d_1, \dots, d_n) \\
u &= \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} v = \begin{pmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{pmatrix}.
\end{aligned}$$

Definition 8.12. Let $f\colon \mathbb{R} \to \mathbb{R}$ be defined by

$$f(\lambda) := 1 + b\langle u, (D - \lambda I)^{-1} u \rangle.$$

The equation $f(\lambda) = 0$ is called the *secular equation*.

The following theorem shows how the eigenvalues of T can be calculated from those of T_1 and T_2.

Theorem 8.13. *Assume that the $\{d_i\}$ are distinct and that $u_i \neq 0$ for all i. Then the eigenvalues of T are the real roots of the secular equation.*

Proof.

$$\begin{aligned} T &= \begin{pmatrix} Q_1\Lambda_1 Q_1^T & 0 \\ 0 & Q_2\Lambda_2 Q_2^T \end{pmatrix} + bv \otimes v \\ &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left(\begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} + bu \otimes u \right) \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix}. \end{aligned}$$

Thus T has the same eigenvalues as $D + bu \otimes u$ by similarity.

Now,

$$\det(D + bu \otimes u - \lambda I) = \det((D - \lambda I)(I + b(D - \lambda I)^{-1} u \otimes u)).$$

If

$$\det(D + bu \otimes u - \lambda I) = 0$$

and λ is not an eigenvalue of D then

$$\det(I + b(D - \lambda I)^{-1} u \otimes u) = 0.$$

Now $\det(I + x \otimes y) = 1 + \langle y, x\rangle$ (see Exercise 8-4) and so

$$1 + b\langle u, (D - \lambda I)^{-1} u\rangle = f(\lambda) = 0.$$

Recall that we assume the $\{d_i\}$ are distinct and that $u_i \neq 0$. Let $d_0 = -\infty$ and $d_{n+1} = \infty$. Then

- if $b < 0$ then $f(\lambda)$ has n roots $\lambda_i \in (d_i, d_{i+1})$, $i = 0, \ldots, n-1$;
- if $b > 0$ then $f(\lambda)$ has n roots $\lambda_i \in (d_i, d_{i+1})$, $i = 1, \ldots, n$.

This follows from the fact that

$$f(\lambda) = 1 + b \sum_{i=1}^{n} \frac{u_i^2}{(d_i - \lambda)}.$$

Hence all the eigenvalues are accounted for and it follows that the assumption that λ cannot be an eigenvalue of D holds. □

Theorem 8.13 is the basis of a divide and conquer algorithm for the eigenvalue problem.

Error Analysis

The primary source of error, in real arithmetic, arises from solution of the secular equation. However, because the roots which we seek are real, and because they lie in known intervals, root-finding methods with great accuracy and efficiency can be applied (a simple example being bisection). In practice the errors arising from this step of the algorithm are negligible compared with those arising from finite precision arithmetic.

Computational Complexity

Let $L_k = C(2^k)$, where $C(n)$ is the cost of finding eigenvalues and eigenvectors of symmetric tridiagonal matrices in $\mathbb{R}^{n\times n}$. If we use a recursive algorithm, then the cost L_k needs to be expressed in terms of the cost L_{k-1} of finding the eigenvalues and eigenvectors of symmetric tridiagonal matrices in $\mathbb{R}^{n/2\times n/2}$.

Finding the eigenvalues and eigenvectors in $\mathbb{R}^{n\times n}$ has cost L_k where

$$L_k = 2L_{k-1} + m_k,$$

where m_k is the cost of constructing, and solving, the secular equation for $T \in \mathbb{R}^{n\times n}$. This cost comprises:

- finding eigenvectors from these eigenvalues: this costs $\Theta(n)$ for each eigenvalue since the matrix is tridiagonal, giving a total of $\Theta(n^2)$;
- constructing u: $\Theta(n)$ to find last column of Q_1^T and first column of Q_2^T;
- Solving the secular equation: $\Theta(n)$, assuming unit cost for each root using, for example, bisection.

To state a theorem about the cost of this iteration we need the following definition.

Definition 8.14. The decomposition (8.8) of a tridiagonal matrix T is *non-degenerate* if the $\{d_i\}$ are distinct and $u_i \neq 0$ for all i.

Theorem 8.15 (Cost of Eigenvalue Divide and Conquer)**.** *Let T be symmetric, tridiagonal. If all decompositions are non-degenerate at each level of recursion and if root finding for the secular equation is assigned unit cost, then the algorithm generates all eigenvalues and eigenvectors of T at cost $\mathcal{O}(n^2)$.*

Proof. We have $m_k = \Theta(n^2) = \Theta(4^k)$. Thus, for some $a > 0$,

$$L_k \leq 2L_{k-1} + a4^k.$$

Induction based on the assumption

$$L_k \leq b4^k$$

gives the desired result, provided $b \geq \min\{1, 2a\}$, since finding the eigenvalues of a 1×1 matrix has unit cost. □

Bibliography

The book [Par80] contains a very complete overview of the symmetric eigenvalue problem, the subject that dominates this chapter. The book [TB97] has a good introduction to the subject.

Exercises

Exercise 8-1. The purpose of this question is to introduce you to ideas associated with eigenvalue problems and their solution in MATLAB. The basic problem is, given $A \in \mathbb{R}^{n\times n}$, to find $x \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$ such that

$$Ax = \lambda x \tag{8.9}$$

and, of course, x should be non-zero.
(i) What can you say about the eigenvalues λ if A is symmetric (just state, no proof). What can you say about the eigenvectors?
(ii) Write down the definition of the characteristic polynomial satisfied by the eigenvalues λ.
(iii) What is the polynomial in the case `>> A=[3,2; 1,2]` Hence find the eigenvalues and eigenvectors.
(iv) If

```
>> A=[3,2,1; 2,2,2; 1,2,1]
```

write the polynomial for λ in the form

$$p_1\lambda^3 + p_2\lambda^2 + p_3\lambda + p_4 = 0$$

and find $p = (p_1, p_2, p_3, p_4)$. In principle you can solve this cubic exactly so there is no need to resort to numerical solution. (In general we obtain

$$\sum_{i=0}^{n} \lambda^{n-i} p_{i+1} = 0.$$

What does Galois tell us about our need for applying numerical solutions to this problem?)
(v) One way to find the eigenvalues of A is to use the MATLAB command `roots`. Type

```
>> roots(p)
```

to find the roots of our cubic, and hence the eigenvalues of A. We can do this more generally. Type

```
>> A=randn(20);
>> A=0.5*[A+A'];
>> p=poly(A)'
```

This creates a 20×20 matrix with normally distributed random entries, and then makes a symmetric matrix from A. The vector p contains the co-efficients in the characteristic polynomial of A so that, if you type `sort(roots(p))` you will get the eigenvalues of A (`sort` just orders the roots). Do this.

(vi) Thus one way to find eigenvalues of A in MATLAB is to form the characteristic polynomial and finds its roots. But MATLAB *does not do this* as more efficient methods are available. Type

```
>> sort(eig(A))
```

and you should get the same as you did in (v). But the method employed was very different. Here we show you the two key ideas used in `eig`. Set

```
>> A=[-1,2,3,0,4;2,1,6,2,2;3,6,3,3,0;0,2,3,8,2;4,2,0,2,-1]
```

and now type

```
>> [Q,H]=hess(A)
```

What properties do Q and H have? Type

```
>> rn=norm(Q*H*Q'-A)
```

What can you say about the relationship between the eigenvalues of H and those of A?

Now type

```
>> [W,D]=schur(H)
```

What properties do W and D have? Type

```
>> rn=norm(W*D*W'-H)
```

What can you say about the relationship between the eigenvalues of D and those of H?

Finally, what can you say about the relationship between the eigenvalues of A and the diagonal elements of D? Test your hypothesis by typing `eig(A)` and comparing with D.

Exercise 8-2. Give a proof by induction which shows that the matrix A from (8.1) really has determinant $(-1)^n p(z)$ where $p(z) = a_0 + a_1 z + \cdots + a_{n-1} z^{n-1} + z^n$.

Exercise 8-3. Let B be a constant matrix and let $T(t)$ solve the matrix differential equation

$$\frac{dT}{dt} = BT - TB, \qquad T(0) = T_0.$$

Assuming that the solution of this equation exists for all $t \in \mathbb{R}$, show that $T(t)$ has the same eigenvalues as T_0, for all $t \in \mathbb{R}$. If B is skew-symmetric, show that $\|T(t)\|_2 = \|T_0\|_2$ for all $t \in \mathbb{R}$.

Exercise 8-4. Prove that $\det(I + xy^T) = 1 + y^T x$.

Exercise 8-5. Construct an example showing that the Jordan canonical form of a matrix can be discontinuous with respect to small perturbations.

Exercise 8-6. Newton's Method

1. Let $G\colon \mathbb{R}^m \to \mathbb{R}^m$ be differentiable. Newton's method for the solution of the equation $G(y) = 0$ is to generate iterates y_k according to

$$dG(y_k)\delta y_k = -G(y_k), \qquad y_{k+1} = y_k + \delta y_k$$

where $dG(y)$ is the Jacobian of G evaluated at y. Find the explicit formula for y_{k+1} in terms of y_k when $m = 1$. What is the order of convergence of the method?

2. The eigenvalue problem for $A \in \mathbb{R}$ can be regarded as a nonlinear system of equations for $y = y(x, \lambda)$ satisfying

$$\begin{aligned} Ax - \lambda x &= 0 \\ \frac{1}{2}(x^T x - 1) &= 0 \end{aligned}$$

Apply Newton's method to this system and show that it gives the iteration

$$\begin{aligned} \lambda_{k+1} &= \lambda_k + \frac{\frac{1}{2}(\|x_k\|_2^2 + 1)}{x_k^T (A - \lambda_k I)^{-1} x_k} \\ x_{k+1} &= (\lambda_{k+1} - \lambda_k)(A - \lambda_k I)^{-1} x_k. \end{aligned}$$

If $x_0 = \alpha_0 \varphi$ ($\alpha_0^2 \neq 1$), $A\varphi = \mu\varphi$ ($\|\varphi\|_2 = 1$) and $\lambda_0 \neq \mu$, show that $x_k = \alpha_k \varphi$ and find the iteration governing the α_k, i.e. find $h\colon \mathbb{R} \to \mathbb{R}$ such that

$$\alpha_{k+1} = h(\alpha_k).$$

Show that this iteration is Newton's method applied to a scalar equation and identify that equation. Using this fact, prove that the full method on $\mathbb{R}^m$ is second order convergent for this specific choice of initial data.

Exercise 8-7. Prove the following theorem concerning eigenvalues of tridiagonal matrices.

Theorem 8.16. *Consider a matrix A of the form (5.5). Let the diagonal entries be real, constant with equal off-diagonal entries: $a_i = a, c_i = a$ and $d_i = d$ for all $i = 1, \dots, m$ where a and d are real. Then the eigenvalues λ_k of the matrix are given by*

$$\lambda_k = d + 2a \cos\left(\frac{k\pi}{m+1}\right),$$

for $k = 1, \dots, m$. The corresponding eigenvector $x = [x_1, \dots, x_m]^T$ has j^{th} component given by

$$x_j = \sin(k\pi j/(m+1)) \tag{8.10}$$

Exercise 8-8. Consider the companion matrix C from (8.1). Prove that the eigenvalues λ_k of C are the roots of the polynomial

$$\sum_{j=0}^{m} \alpha_j \lambda^j = 0$$

where $\alpha_m = 1$. The corresponding eigenvector is

$$x = [x_1, \dots, x_m]^T$$

where $x_j = \lambda_k^{j-1}$.

Bibliography

[Bha97] R. Bhatia. *Matrix Analysis.* Springer, 1997.

[CH96] S.-N. Chow and J.K. Hale. *Methods of Bifurcation Theory.* Springer, 1996.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* McGraw Hill, 2001.

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

[GL96] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins, third edition, 1996.

[Hig02] N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 2002.

[HJ85] R.A. Horn and JC.R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

[Lax97] P. Lax. *Linear Algebra.* Wiley, 1997.

[LT88] Randall J. Leveque and Lloyd N. Trefethen. Fourier analysis of the SOR iteration. *IMA Journal of Numerical Analysis*, 8(3):273–279, 1988.

[Ner71] E.D. Nering. *Linear Algebra and Matrix Theory.* John Wiley, 1971.

[NW06] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer, 2006.

[Par80] B. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, 1980.

[Rob01] J.C. Robinson. *Infinite Dimensional Dynamical Systems.* CUP, 2001.

[Saa97] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS, 1997.

[SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer, third edition, 2002.

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