

Numerical Linear Algebra

Jochen Voss
University of Warwick

December 2004

I tried to keep the text as free of errors as possible. Please report any remaining mistakes to Jochen Voss (voss@seehuhn.de). The current version of the text can always be found on my home page at <http://seehuhn.de/mathe/numlinalg.html>.

Copyright © 2004 Jochen Voss and Andrew Stuart

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Contents

Introduction	3
1 Linear Algebra	5
1.1 Vector Norms and Inner Products	5
1.2 Matrix Norms	7
2 Complexity of Algorithms	11
2.1 Computational Cost	11
2.2 Analysis of Matrix-Matrix Multiplication	12
3 Stability and Conditioning	16
3.1 Conditioning	16
3.2 Stability	18
4 Systems of Linear Equations	21
4.1 Gaussian Elimination	21
4.2 Gaussian Elimination with Partial Pivoting	25
4.3 The QR-Factorisation	28
5 Iterative Methods	34
5.1 Linear Methods	34
5.2 The Conjugate-Gradient Method	40
6 Least Square Problems	43
6.1 Singular Value Decomposition	43
6.2 The Normal Equations	45
6.3 Conditioning of LSQ	47
7 The Eigenvalue Problem	49

Introduction

These lecture notes cover the course “Numerical Linear Algebra” (MA398) given in the autumn term 2004 at the University of Warwick. The notes are partially based on lecture notes written by Andrew Stuart for earlier courses.

What is numerical linear algebra? Of course, we consider the same problems which are considered in a linear algebra course. But this time the focus is different. We are interested in

- solving large problems by using a computer
- considering speed and stability of the algorithms

These large problems occur for example when continuous problems are discretised or in image analysis. Throughout the lecture we will address three main problems.

SLE (simultaneous linear equations): given a matrix $A \in \mathbb{C}^{n \times n}$ and a vector $b \in \mathbb{C}^n$, find $x \in \mathbb{C}^n$ with

$$Ax = b.$$

LSQ (least square problem): given a matrix $A \in \mathbb{C}^{m \times n}$ and a vector $b \in \mathbb{C}^m$ ($m \geq n$), find $x \in \mathbb{C}^n$ which minimises the distance

$$\|Ax - b\|.$$

EVP (eigenvalue problem): given a matrix $A \in \mathbb{C}^{n \times n}$ find $x \in \mathbb{C}^n$, $\lambda \in \mathbb{C}$ with

$$Ax = \lambda x \quad \text{with} \quad x \neq 0.$$

The lecture is structured as follows.

Chapter 1 (Linear Algebra) will start by summarising a few results from linear algebra and provide some basic theoretical tools which we will need for our analysis.

Chapter 2 (Complexity of Algorithms) introduces measures for the cost to perform a given algorithm and illustrates this by analysing a few simple problems.

Chapter 3 (Stability and Conditioning) examines how much the solution of a linear problem can change when the problem is slightly perturbed, for example in the presence of rounding errors.

Chapter 4 (SLE) compares several algorithms to solve systems of linear equations. We will consider both the stability and the cost of these algorithms.

Chapter 5 (LSQ) introduces the least squares problem and discusses several algorithms to solve it.

Chapter 6 (Iterative Methods) introduces another set of algorithms to solve systems of linear equations: iterative methods only give approximate solutions but can be both faster and more stable than the methods presented in chapter 4.

Chapter 7 (EVP) explains algorithms to find eigenvalues of large matrices.

All of this is meant to deal with really big matrices. Where do these matrices come from? As mentioned above these occur e.g. in the area of discretised continuous problems. This is illustrated by the following example.

Example. We want to numerically solve the following problem: given $a, b \in \mathbb{R}$, find a function $f: [0, 1] \rightarrow \mathbb{R}$ with $f''(x) = 0$ for all $x \in [0, 1]$, $f(0) = a$ and $f(1) = b$.

Of course this problem only serves as an illustration. It can easily be solved analytically (question to the reader: what is the result?). But there are similar problems (for example in the two-dimensional case) where direct solution is no longer feasible and the numerical approach becomes practical. The methods used there are exactly the same as the ones presented in this example.

The idea is to discretise the problem: for $k = 0, \dots, N$ and $N \in \mathbb{N}$ let $x_k = k/N$. We consider the finite set $\{x_0, x_1, \dots, x_N\}$ instead of the interval $[0, 1]$ and we consider the vector $(f(x_0), f(x_1), \dots, f(x_n))$ instead of the function f .

What should we do about the derivative f'' ? Using the Taylor formula we find that for large N the approximation

$$f''(x_k) \approx \frac{f(x_{k-1}) - 2f(x_k) + f(x_{k+1}))}{(1/N)^2} \quad \text{for all } k = 1, \dots, N-1 \quad (0.1)$$

holds. This leads to the following system of $N+1$ linear equations:

$$\begin{aligned} f(x_0) &= a \\ N^2 f(x_{k-1}) - 2N^2 f(x_k) + N^2 f(x_{k+1}) &= 0 \quad \text{for } k = 1, \dots, N-1 \\ f(x_N) &= b. \end{aligned}$$

Since we used the approximation (0.1) the result will not be exact, but if we choose N large enough the approximation gets better and we can hope that the result is close to the exact result.

To solve this problem we need to be able to deal with large systems of linear equations.

As mentioned above this example is very simple and only serves as an illustration, but more interesting examples lead to the same kind of linear equations. You can learn more about this in courses about numerical solution of partial differential equations.

The course gives only an introduction in the topics of numerical linear algebra. Further results can be found in many text books. The course is based on the following books: the books of Lloyd N. Trefethen [TB97] and James W. Demmel [Dem97] give a good introductions into the Topic. J. Stoer and R. Bulirsch [SB02] give a more theoretical presentation of numerical analysis, which also contains results about numerical linear Algebra. The book of Roger A. Horn and Charles R. Johnson [HJ85] is a good reference for theoretical results about matrix analysis. Nicholas J. Higham's book [Hig02] contains a lot of information about stability and the effect of rounding errors in numerical algorithms.

Chapter 1

Linear Algebra

The purpose of this chapter is to summarise a few results from linear algebra and to provide some basic theoretical tools which we will later need for our analysis.

1.1 Vector Norms and Inner Products

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

- $\|x\| \geq 0$ for all $x \in \mathbb{C}^n$ and $\|x\| = 0$ iff $x = 0$,
- $\|\alpha x\| = |\alpha| \|x\|$ for all $\alpha \in \mathbb{C}, x \in \mathbb{C}^n$, and
- $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{C}^n$.

Examples:

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

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

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

Definition 1.2. An *inner-product* on \mathbb{C}^n is a mapping $\langle \cdot, \cdot \rangle: \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}$ satisfying:

- $\langle x, x \rangle \in \mathbb{R}^+$ for all $x \in \mathbb{C}^n$ and $\langle x, x \rangle = 0$ iff $x = 0$
- $\langle x, y \rangle = \overline{\langle y, x \rangle}$ for all $x, y \in \mathbb{C}^n$
- $\langle x, \alpha y \rangle = \alpha \langle x, y \rangle$ for all $\alpha \in \mathbb{C}, x, y \in \mathbb{C}^n$.
- $\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle$ for all $x, y, z \in \mathbb{C}^n$

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

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

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

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

i.e. the inner product is *anti-linear* in the first component.

Definition 1.3. Two vectors x, y are *orthogonal* with respect to the inner product $\langle \cdot, \cdot \rangle$ iff $\langle x, y \rangle = 0$.

Lemma 1.4. Let $\langle \cdot, \cdot \rangle: \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}$ be an inner product. Then $\|\cdot\|: \mathbb{C}^n \rightarrow \mathbb{R}$ defined by

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

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 defined without problems 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) Note that the proof of the Cauchy-Schwarz inequality

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

only uses properties of the inner product. So we can use it here even before we know that $\|\cdot\|$ is a norm. We get

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

This finishes the proof. □

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

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} = (a_{ij})_{ij}. \quad (1.2)$$

Definition 1.5. Given $A \in \mathbb{C}^{m \times n}$ we define the *adjoint* $A^* \in \mathbb{C}^{n \times m}$ by $A^* = (\overline{a_{ji}})_{ij}$. (For $A \in \mathbb{R}^{m \times n}$ we get $A^* = A^T$.)

Using this definition we can write the standard inner product as

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

Definition 1.6. A matrix $Q \in \mathbb{C}^{m \times n}$, $m \geq n$, is *unitary* if $Q^* Q = I$, i.e. if the columns of Q are orthogonal with respect to the standard inner-product. (If $Q \in \mathbb{R}^{m \times n}$ satisfies $Q^T Q = I$, we say that the matrix Q is *orthogonal*.)

A matrix $A \in \mathbb{C}^{n \times n}$ is *Hermitian* if $A^* = A$ (for real matrices: *symmetric*).

A Hermitian matrix $A \in \mathbb{C}^{n \times n}$ is *positive-definite* if $x^* A x > 0$ for all $x \in \mathbb{C}^n \setminus \{0\}$ (*positive semi-definite* for ≥ 0).

Remarks. 1) Unless otherwise specified, $\langle \cdot, \cdot \rangle$ will denote the standard inner-product (1.1). The standard inner product satisfies

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

for all $x, y \in \mathbb{C}^n$.

2) If $A \in \mathbb{C}^{n \times n}$ is Hermitian and positive-definite, then

$$\langle x, y \rangle_A = \langle x, Ay \rangle \quad \forall x, y \in \mathbb{C}^n$$

defines an inner product and

$$\|x\|_A = \sqrt{\langle x, x \rangle_A} \quad \forall x \in \mathbb{C}^n$$

defines a norm on \mathbb{C}^n .

1.2 Matrix Norms

Definition 1.7. A *matrix norm* on $\mathbb{C}^{n \times n}$ is a mapping $\|\cdot\|: \mathbb{C}^{n \times n} \rightarrow \mathbb{R}$ with

- a) $\|A\| \geq 0$ 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) $\|A + B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{C}^{n \times n}$.
- d) $\|AB\| \leq \|A\| \|B\|$ 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.

Definition 1.8. Given a vector norm $\|\cdot\|_v$ on \mathbb{C}^n we define the *induced norm* $\|\cdot\|_m$ on $\mathbb{C}^{n \times n}$ by

$$\|A\|_m = \max_{x \neq 0} \frac{\|Ax\|_v}{\|x\|_v}$$

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

Theorem 1.9. The induced norm $\|\cdot\|_m$ of a vector norm $\|\cdot\|_v$ is a matrix norm with

$$\|I\|_m = 1$$

and

$$\|Ax\|_v \leq \|A\|_m \|x\|_v$$

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

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

$$\begin{aligned} \|A\|_m = 0 &\iff \frac{\|Ax\|_v}{\|x\|_v} = 0 \quad \forall x \neq 0 \\ &\iff \|Ax\|_v = 0 \quad \forall x \neq 0 \iff Ax = 0 \quad \forall x \neq 0 \iff A = 0. \end{aligned}$$

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

$$\|\alpha A\|_m = \max_{x \neq 0} \frac{\|\alpha Ax\|_v}{\|x\|_v} = \max_{x \neq 0} \frac{|\alpha| \|Ax\|_v}{\|x\|_v} = |\alpha| \|A\|_m$$

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

$$\begin{aligned} \|A + B\|_m &= \max_{x \neq 0} \frac{\|Ax + Bx\|_v}{\|x\|_v} \\ &\leq \max_{x \neq 0} \frac{\|Ax\|_v + \|Bx\|_v}{\|x\|_v} \\ &\leq \max_{x \neq 0} \frac{\|Ax\|_v}{\|x\|_v} + \max_{x \neq 0} \frac{\|Bx\|_v}{\|x\|_v} = \|A\|_m + \|B\|_m. \end{aligned}$$

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

$$\|I\| = \max_{x \neq 0} \frac{\|Ix\|_v}{\|x\|_v} = \max_{x \neq 0} \frac{\|x\|_v}{\|x\|_v} = 1$$

and

$$\|A\|_m = \max_{y \neq 0} \frac{\|Ay\|_v}{\|y\|_v} \geq \frac{\|Ax\|_v}{\|x\|_v} \quad \forall x \in \mathbb{C}^n \setminus \{0\}$$

which gives

$$\|Ax\|_v \leq \|A\|_m \|x\|_v \quad \forall x \in \mathbb{C}^n.$$

d) Using this estimate we find

$$\begin{aligned} \|AB\|_m &= \max_{x \neq 0} \frac{\|ABx\|_v}{\|x\|_v} \\ &\leq \max_{x \neq 0} \frac{\|A\|_m \|Bx\|_v}{\|x\|_v} = \|A\|_m \|B\|_m. \end{aligned}$$

□

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

Recall that $x \in \mathbb{C}^n$ is an *eigenvector* with *eigenvalue* $\lambda \in \mathbb{C}$ of a matrix $A \in \mathbb{C}^{n \times n}$ if

$$Ax = \lambda x \quad \text{and} \quad x \neq 0. \tag{1.3}$$

Definition 1.10. The *spectral radius* of a matrix $A \in \mathbb{C}^{n \times n}$ is defined by

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

Theorem 1.11. For any matrix norm $\|\cdot\|$, any matrix $A \in \mathbb{C}^{n \times n}$ and any $\ell \in \mathbb{N}$ we have

$$\rho(A)^\ell \leq \|A^\ell\| \leq \|A\|^\ell.$$

Proof. By definition of the spectral radius $\rho(A)$ we can find an eigenvector x with $Ax = \lambda x$ and $\rho(A) = |\lambda|$. Let $X \in \mathbb{C}^{n \times n}$ be the matrix where all n columns are equal to x . Then we have $A^\ell X = \lambda^\ell X$ and thus

$$\|A^\ell\| \|X\| \geq \|A^\ell X\| = \|\lambda^\ell X\| = |\lambda|^\ell \|X\| = \rho(A)^\ell \|X\|.$$

Dividing by $\|X\|$ gives $\rho(A)^\ell \leq \|A^\ell\|$. The second inequality follows from property d) in the definition of a matrix norm. \square

Definition 1.12. A matrix $A \in \mathbb{C}^{n \times n}$ is *normal* if $A^*A = AA^*$.

Lemma 1.13. $A \in \mathbb{C}^{n \times n}$ is normal iff it has n orthonormal eigenvectors, i.e. eigenvectors x_1, \dots, x_n with $\langle x_i, x_j \rangle = \delta_{ij}$ for all $i, j = 1, \dots, n$.

Theorem 1.14. If $A \in \mathbb{C}^{n \times n}$ is normal, then

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

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

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

This shows

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

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

Using theorem 1.11 we get

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

for all $\ell \in \mathbb{N}$. This finishes the proof. \square

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

Theorem 1.15. For all matrices $A \in \mathbb{C}^{n \times n}$ we have

$$\|A\|_2 = \sqrt{\rho(A^*A)}.$$

With the previous theorems we have among other things identified the matrix norm which is induced by the 2-vector norm. The following theorem explicitly identifies the matrix norm associated to the infinity-norm.

Theorem 1.16. *The matrix norm induced by the infinity norm is the maximum row sum norm:*

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

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

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

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 $k \in \{1, 2, \dots, n\}$ 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 $j = 1, \dots, n$. Then we have $\|x\|_\infty = 1$ and

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

This is the required result. □

Exercises

1) Show that the matrix norm induced by the $\|\cdot\|_1$ -norm on \mathbb{C}^n is the *maximum column sum norm*

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

2) 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.

3) Let $A, B, C \in \mathbb{C}^{n \times n}$ with $A = BC$. Show that

$$\|A\|_{\max} \leq \|B\|_\infty \|C\|_{\max}$$

and

$$\|A\|_{\max} \leq \|B\|_{\max} \|C\|_1.$$

Chapter 2

Complexity of Algorithms

In this chapter we learn how to analyse how long it takes to solve a numerical problem on a computer. Specifically we are interested in the question how the cost to perform an algorithm depends on the input size.

2.1 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. A more detailed analysis would also take factors like memory usage into account.

Definition 2.1. The *cost* of an algorithm is

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

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 asymptotic computational cost of an algorithm, this is the behaviour of the cost $C(n)$ for $n \rightarrow \infty$.

Definition 2.2. For $f, g: \mathbb{N} \rightarrow \mathbb{N}$ or $f, g: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ we write

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

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

Standard inner product algorithm.

input: $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$

output: $s = \langle x, y \rangle$

- 1: $s = 0$
- 2: **for** $i = 1, \dots, n$ **do**
- 3: $s = s + x_i y_i$
- 4: **end for**
- 5: return s

Theorem 2.3. 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 additions, i.e. $C(n) = 2n = \Theta(n)$.

Sketch for the proof of 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. \square

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 prove that “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.

Theorem 2.4. *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. Denote the rows of $A \in \mathbb{C}^{n \times n}$ by a_1^*, \dots, a_n^* and the columns of $B \in \mathbb{C}^{n \times n}$ by b_1, \dots, b_n . Since

$$(AB)_{ij} = a_i^* b_j \quad \text{for all } i, j = 1, \dots, n$$

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 $C(n) \geq n^2$. \square

2.2 Analysis of Matrix-Matrix Multiplication

In theorem 2.4 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

$$D_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$D_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$D_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$D_{22} = A_{21}B_{12} + A_{22}B_{22}.$$

The above method calculates the product of two $n \times n$ -matrices using eight multiplications of $(n/2) \times (n/2)$ -matrices. There is another way to calculate the entries of the matrix D , which looks more complicated at first but 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

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})B_{11}$$

$$P_3 = A_{11}(B_{12} - B_{22})$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{12})B_{22}$$

$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

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: $AB \in \mathbb{C}^{n \times n}$

- 1: **if** $n = 1$ **then**
- 2: return AB
- 3: **else**
- 4: calculate P_1, \dots, P_7 (using recursion)
- 5: calculate D_{11}, D_{12}, D_{21} and D_{22}
- 6: return D
- 7: **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

$$C(n) = 7C(n/2) + 18n^2/4. \quad (2.1)$$

Lemma 2.5. *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

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

Now the claim follows by induction. □

Theorem 2.6. *The Strassen-algorithm for matrix-matrix multiplication has asymptotic computational cost $C(n) = \Theta(n^{\log_2 7})$.*

Remark. We will prove the theorem for $n = 2^k$, $k \in \mathbb{N}_0$. If n is not of this form 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

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

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 $C(n) = \Theta(n^{\log_2 7})$ for every $n \in \mathbb{N}$.

Proof. (of the theorem) Let $n = 2^k$. 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. \square

Theorem 2.7. *If there is a method to multiply $n \times n$ -matrices with asymptotic computational cost $\mathcal{O}(n^\alpha)$ for some $\alpha > 0$, 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 $\Theta((2n)^\alpha) = \Theta(n^\alpha)$ 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 $S = B_{22} - B_{12}^*B_{11}^{-1}B_{12}$. A direct calculation shows that

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

(This formula for B^{-1} is called *Schur complement*.)

Exercise: show that 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 2.4 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 below) one can conclude

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

and thus we get

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

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

Exercises

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

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

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

3) Show that the matrices S and B_{11} in the proof of theorem 2.7 are invertible.

4) Show by induction that $D(1) = 1$ and

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

for some constant $c > 0$ implies

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

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

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?

3.1 Conditioning

Definition 3.1. 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.

For this chapter fix a vector norm $\|\cdot\|$ and a matrix norm $\|\cdot\|$ which is *compatible* with the vector norm, that is which satisfies

$$\|Ax\| \leq \|A\| \cdot \|x\| \quad \text{for all } x \in \mathbb{C}^n, A \in \mathbb{C}^{n \times n}.$$

This condition is for example satisfied when the matrix norm is induced by the vector norm.

Definition 3.2. The *condition number* $\kappa(A)$ of a matrix is the number

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

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

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

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

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

Lemma 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 everything is clear. Otherwise we have

$$\|b\| = \|Ax\| \leq \|A\| \|x\| \tag{3.1}$$

and

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

Therefore 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 (3.1). \square

The previous lemma gave an upper bound on how much the solution of the equation $Ax = b$ can change if the right hand side is slightly perturbed. The result shows that the problem is well conditioned if the condition number $\kappa(A)$ is small. Theorem 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. *If $A \in \mathbb{C}^{n \times n}$ satisfies $\|A\| < 1$ in any 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

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

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

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

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

Theorem 3.5 (conditioning of SLE). *Let x solve the equations*

$$Ax = b \quad \text{and} \quad (A + \Delta A)(x + \Delta x) = b.$$

Assume that A is invertible with $\|A^{-1}\|\|A\| < 1$ in some induced matrix norm. Then we have

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

Proof. We find

$$(A + \Delta A)\Delta x = b - (A + \Delta A)x = \Delta Ax$$

and thus $(I + A^{-1}\Delta A)\Delta x = A^{-1}\Delta Ax$. Using lemma 3.4 we can write

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

and we get

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

Since

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

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

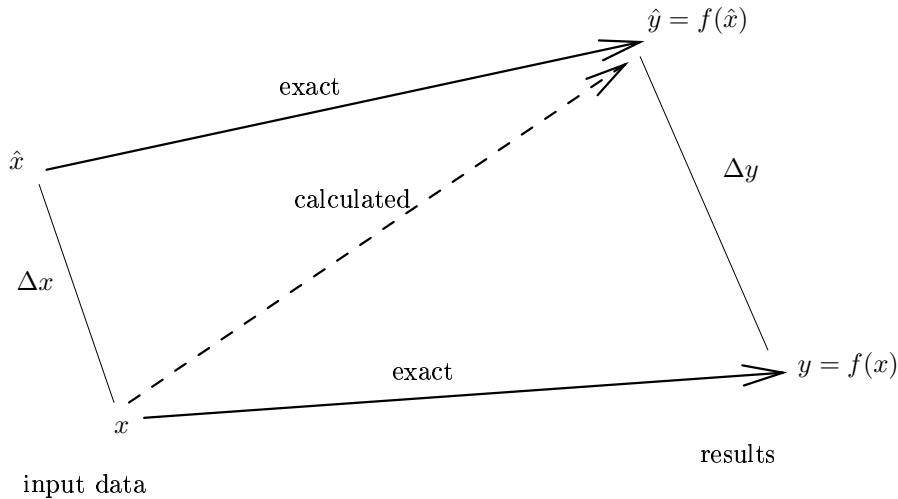
$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\kappa(A) \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

This is the required result. □

3.2 Stability

Stability of an algorithm measures how susceptible it is to rounding errors occurring during the computation. In this section we will for the first time distinguish between the *computed result* as returned by a computer and the *exact result* which would be the mathematically correct solution of the problem.

Definition 3.6. Assume we want to numerically calculate a value $y = f(x)$, but the algorithm returns the computed result $\hat{y} \neq y$ which can be represented as the exact image $\hat{y} = f(\hat{x})$ of a different input value \hat{x} . Then $\Delta y = \hat{y} - y$ is called the *forward error* and $\Delta x = \hat{x} - x$ is called the *backward error*.



If \hat{x} is not unique then we choose the one which results in minimal $\|\Delta x\|$. Typically we consider the *relative backward error* $\|\Delta x\|/\|x\|$ and the *relative forward error* $\|\Delta y\|/\|y\|$.

Theorem 3.7. For the problem SLE we have

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

Proof. Note that in the notation of SLE the relative forward error is $\|\Delta x\|/\|x\|$ and the relative backward error is $\|\Delta b\|/\|b\|$. Therefore the theorem is an immediate consequence of lemma 3.3. □

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 $\mathcal{F} \subset \mathbb{R}$ be the set of representable numbers and let $\text{fl}: \mathbb{R} \rightarrow \mathcal{F}$ be the rounding to the closest element of \mathcal{F} .

In this course 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).

Assumptions. There is a parameter $\varepsilon_m > 0$ (the *machine epsilon*) such that the following conditions hold.

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

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

A2: For each operation $* \in \{+, -, \cdot, /\}$ and every $x, y \in \mathcal{F}$ there is an $\varepsilon \in (-\varepsilon_m, +\varepsilon_m)$ with

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

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

Definition 3.8. An algorithm, is called *backward stable* if the relative backward error satisfies

$$\frac{\|\Delta x\|}{\|x\|} = \mathcal{O}(\varepsilon_m).$$

The typical way to use this concept is in a two-step procedure called *backward error analysis*. In a first step 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. In the second step one uses results like theorem 3.7 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 to the exact result.

Lemma 3.9. *The calculated subtraction \ominus is backward stable.*

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

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

with $|\varepsilon_1|, |\varepsilon_2| < \varepsilon_m$. Using assumption A2 we get

$$\text{fl}(x_1) \ominus \text{fl}(x_2) = (\text{fl}(x_1) - \text{fl}(x_2))(1 + \varepsilon_3)$$

where $|\varepsilon_3| < \varepsilon_m$. This gives

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

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

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 $\|\Delta x\|_2 = \sqrt{\varepsilon_4^2 x_1^2 + \varepsilon_5^2 x_2^2} \leq \mathcal{O}(\varepsilon_m)\|x\|_2$. This completes the proof. \square

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 minimises the backward error, we can choose any \hat{x} which gives the result $\|\Delta x\|_2 \leq \mathcal{O}(\varepsilon_m)\|x\|_2$. The “real” \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 always based on our assumptions A1 and A2 about computer arithmetic. Since they tend to be long but not difficult we omit most of these proofs in the course.

Exercises

- 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.

- 2) 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

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

- 3) Show that the calculated arithmetic operations \oplus , \odot and \otimes are backward stable.

Chapter 4

Systems of Linear Equations

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

We present several methods to solve this problem. The common idea is to split 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.

4.1 Gaussian Elimination

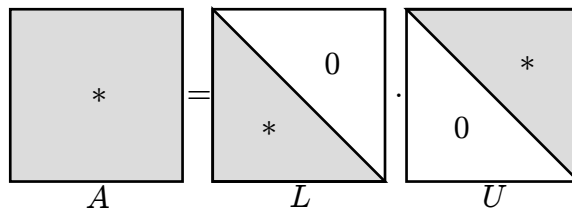
Gaussian elimination is the most commonly known method to solve systems of linear equations. The mathematical background of the algorithm is the following theorem.

Definition 4.1. A matrix $A \in \mathbb{C}^{m \times n}$ is said to be *upper triangular* if $a_{ij} = 0$ for all $i > j$ and *lower triangular* if $a_{ij} = 0$ for all $i < j$. A triangular matrix is said to be *unit triangular* if all diagonal entries are equal to 1.

Definition 4.2. 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 4.3 (LU Factorisation). a) Let $A \in \mathbb{C}^{n \times n}$ be a matrix such that A^j is invertible for $j = 1, \dots, n$. Then there is a unique factorisation $A = LU$ where L is unit lower triangular and U is non-singular upper triangular. b) If A^j is singular for one $j \in \{1, \dots, n\}$ 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 can set $L = (1) \in \mathbb{C}^{1 \times 1}$ and $R = (a_{11}) \in \mathbb{C}^{1 \times 1}$ 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 principle sub-matrices are invertible. We write $A \in \mathbb{C}^{n \times n}$ as

$$A = \begin{pmatrix} A^{n-1} & b \\ c^* & a_{nn} \end{pmatrix} \tag{4.1}$$

where A^{n-1} is the $(n-1)^{\text{th}}$ principle 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

$$A = \begin{pmatrix} L & 0 \\ \ell^* & 1 \end{pmatrix} \begin{pmatrix} U & u \\ 0 & \eta \end{pmatrix} = \begin{pmatrix} LU & Lu \\ \ell^*U & \ell^*u + \eta \end{pmatrix} \quad (4.2)$$

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 $\eta \in \mathbb{C}$. We compare the blocks of (4.1) and (4.2).

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 a 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.

b) Assume that A has an LU-factorisation and let $j \in \{1, \dots, n\}$. 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. □

Example. The matrix A can be converted into upper triangular shape by multiplying lower triangular matrices from the left. Let for example

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

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

$$L_1A = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix}.$$

Repeating this for the second column gives

$$L_2L_1A = \begin{pmatrix} 1 & & \\ & 1 & \\ & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

We have found lower triangular matrices L_1 and L_2 and an upper triangular matrix R with $A = (L_2L_1)^{-1}R$. The following lemma helps to calculate $(L_2L_1)^{-1} = L_1^{-1}L_2^{-1}$.

Lemma 4.4. 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 $1, \dots, k$ and B has non-zero entries below the diagonal only in columns $k+1, \dots, n$. Then AB is unit lower triangular with $(AB)_{ij} = a_{ij}$ for $j \in \{1, \dots, k\}$ and $(AB)_{ij} = b_{ij}$ for $j \in \{k+1, \dots, n\}$.

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

$$(L_2L_1)^{-1} = L_1^{-1}L_2^{-1} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 1 & \\ & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix}.$$

Thus we found the LU -factorisation

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

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 $j = 1, \dots, n$

output: $L, U \in \mathbb{C}^{n \times n}$ where $A = LU$ is the LU-factorisation of A

- 1: $U = A, L = I$
- 2: **for** $k = 1, \dots, n - 1$ **do**
- 3: **for** $j = k + 1, \dots, n$ **do**
- 4: $l_{jk} = u_{jk}/u_{kk}$
- 5: $(u_{j,k}, \dots, u_{j,n}) = (u_{j,k}, \dots, u_{j,n}) - l_{j,k}(u_{k,k}, \dots, u_{k,n})$
- 6: **end for**
- 7: **end for**

Remarks. Line 5 of the algorithm subtracts a multiple of line k from line j , causing $u_{jk} = 0$ without changing columns $1, \dots, k - 1$. 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 $1, \dots, k$.

Since the principal sub-matrices A^j are non-singular and the matrices L_j are unit triangular we get

$$\det(L_k \cdots L_1 A)^{k+1} = \det A^{k+1} \neq 0$$

and thus we have $u_{kk} \neq 0$ in line 4. Lemma 4.4 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 $Ux = b$

- 1: **for** $j = n, \dots, 1$ **do**
- 2: $x_j = \frac{1}{u_{jj}} \left(b_j - \sum_{k=j+1}^n u_{jk} x_k \right)$
- 3: **end for**

Remarks. 1) Since U is triangular we get

$$\begin{aligned} (Ux)_i &= \sum_{j=i}^n u_{ij} x_j \\ &= u_{ii} \left(\frac{1}{u_{ii}} (b_i - \sum_{k=i+1}^n u_{ik} x_k) \right) + \sum_{j=i+1}^n u_{ij} x_j \\ &= b_i. \end{aligned}$$

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 $j = 1, \dots, n$

output: $x \in \mathbb{C}^n$ with $Ax = b$

- 1: find the LU-factorisation of A
- 2: solve $Ly = b$ using forward substitution
- 3: solve $Ux = y$ using back substitution

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.

Computational Complexity of Gaussian Elimination

Lemma 4.5. *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 $(n - k)(1 + 2(n - k + 1))$ operations. Considering the outer loop the total number of operations is

$$C(n) = \sum_{k=1}^{n-1} (n - k)(1 + 2(n - k + 1)) = \sum_{k=1}^{n-1} 2(n - k)^3 + 3(n - k).$$

The claim follows now by induction. □

Notation. For $f, g: \mathbb{N} \rightarrow \mathbb{N}$ or $f, g: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ we write

$$f(x) \sim g(x) \quad \text{for } x \rightarrow \infty$$

if $\lim_{x \rightarrow \infty} f(x)/g(x) = 1$. This implies $f(x) = \Theta(g(x))$ but is a stronger concept since we also compare the leading constants. Using this notation the claim of the lemma becomes $C(n) \sim \frac{2}{3}n^3$.

Lemma 4.6. *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

$$C(n) = \sum_{j=1}^n (2(n - j) + 1) = 2 \sum_{k=0}^{n-1} k + n = n(n - 1) + n = n^2.$$

A similar argument applies to the situation of forward substitution. □

Theorem 4.7 (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. □

Error Analysis of Gaussian Elimination

Theorem 4.8. *The back substitution algorithm is backward stable: the computed solution \hat{x} satisfies $(U + \Delta U)\hat{x} = b$ for some upper triangular matrix $\Delta U \in \mathbb{C}^{n \times n}$ with*

$$\frac{\|\Delta U\|}{\|U\|} = \mathcal{O}(\varepsilon_m).$$

The proof makes extensive use of assumptions (A1) and (A2) about computer arithmetic and we omit it here. Using theorem 3.5 about the conditioning of SLE we get an upper bound on the error in the computed result of back substitution:

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \frac{\kappa(U)}{1 - \kappa(U) \frac{\|\Delta U\|}{\|U\|}} \cdot \frac{\|\Delta U\|}{\|U\|} = \kappa(U) \mathcal{O}(\varepsilon_m).$$

Thus the backward substitution step of Gaussian elimination is numerically stable and introduces no problem. The same holds for forward substitution.

Problem. LU-factorisation is not backward stable! The effect is illustrated in the following example. Let

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

for some $\varepsilon > 0$. Then A has a LU-factorisation $A = LU$ with

$$L = \begin{pmatrix} 1 & 0 \\ \varepsilon^{-1} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} \varepsilon & 1 \\ 0 & 1 - \varepsilon^{-1} \end{pmatrix}.$$

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

$$\tilde{L} = \begin{pmatrix} 1 & 0 \\ \varepsilon^{-1} & 1 \end{pmatrix}, \quad \tilde{U} = \begin{pmatrix} \varepsilon & 1 \\ 0 & -\varepsilon^{-1} \end{pmatrix},$$

which is compatible with assumption (A1) on rounding errors. We have $\tilde{L} = L$ and $\tilde{U} \approx U$. But multiplying the two rounded matrices gives

$$\tilde{L}\tilde{U} = \begin{pmatrix} \varepsilon & 1 \\ 1 & 0 \end{pmatrix} = A + \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix}.$$

A small rounding error led to a large difference in the result! 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.

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.

Because of this instability the classical Gaussian elimination method is not used in numerical methods. The next section introduces a modification of Gaussian elimination, which cures this problem.

4.2 Gaussian Elimination with Partial Pivoting

Definition 4.9. 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 one one.

Example. If $\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ 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. (Every permutation matrix is of this form.) In particular the identity matrix is a permutation matrix.

Remarks. 1) If P is a permutation matrix then we have

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

and thus $P^* P = I$. This shows that permutation matrices are orthogonal and have $P^{-1} = P^T$.

2) 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} .

3) 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 .

The problem in our example for the instability of Gaussian elimination was caused by the fact that we had to divide by the tiny number $u_{kk} = \varepsilon$ in step 4 of the LU-factorisation algorithm. We will avoid this problem in the improved version of the algorithm by rearranging rows k, \dots, n at the beginning of the k th 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 P_k from the left exchanges rows k and i_k where i_k is chosen 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

$$L'_k = P_{n-1} \cdots P_{k+1} L_k P_{k+1}^{-1} \cdots P_{n-1}^{-1}$$

for $k = 1, \dots, n-1$. 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 4.4 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, R non-singular upper triangular and P a permutation matrix

- 1: $U = A, L = I, P = I$
- 2: **for** $k = 1, \dots, n-1$ **do**
- 3: choose $i \in \{k, \dots, n\}$ which maximises $|u_{ik}|$

```

4:  exchange row  $(u_{k,k}, \dots, u_{k,n})$  with  $(u_{i,k}, \dots, u_{i,n})$ 
5:  exchange row  $(l_{k,1}, \dots, l_{k,k-1})$  with  $(l_{i,1}, \dots, l_{i,k-1})$ 
6:  exchange row  $(p_{k,1}, \dots, p_{k,n})$  with  $(p_{i,1}, \dots, p_{i,n})$ 
7:  for  $j = k + 1, \dots, n$  do
8:     $l_{jk} = u_{jk}/u_{kk}$ 
9:     $(u_{j,k}, \dots, u_{j,n}) = (u_{j,k}, \dots, u_{j,n}) - l_{j,k}(u_{k,k}, \dots, u_{k,n})$ 
10: end for
11: end for

```

Remarks. 1) The resulting matrix L has $l_{ij} \leq 1$ for all $i, j \in \{1, \dots, n\}$.

2) The computational complexity is the same as for the LU-algorithm. This is trivial in our simplified analysis since we only added steps to exchange rows to the LU-algorithm and we do not count these operations. But the result still holds for a more detailed complexity analysis: the number of additional assignments is of order $\mathcal{O}(n^2)$ and can be neglected for a $\Theta(n^3)$ algorithm.

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

output: $x \in \mathbb{C}^n$ with $Ax = b$

- 1: find $PA = LU$ using algorithm LUPP
- 2: solve $Ly = Pb$ using forward substitution
- 3: solve $Ux = y$ using back substitution

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.

Error Analysis of GEPP

Theorem 4.10 (Backward Error Analysis for LUPP). *The computed result of algorithm LUPP satisfies*

$$\tilde{L}\tilde{U} = \tilde{P}(A + \Delta A)$$

with

$$\frac{\|\Delta A\|}{\|A\|} \leq \varepsilon_m p(n) g_n(A)$$

where p is a polynomial and $g_n(A)$ (the growth factor) is defined as

$$g_n(A) = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}.$$

Lemma 4.11. *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 $k \in \{1, \dots, n-1\}$ and any matrix $A \in \mathbb{C}^{n \times n}$ we find

$$\begin{aligned} \max_{i,j} |(L'_k A)_{ij}| &= \max_{i,j=1,\dots,n} \left| \sum_{l=1}^n (L'_k)_{il} a_{lj} \right| \\ &\leq \max_{i=1,\dots,n} \left| \sum_{l=1}^n (L'_k)_{il} \right| \max_{i,j} |a_{ij}| \\ &\leq 2 \max_{i,j} |a_{ij}| \end{aligned}$$

and thus by applying the matrices L'_1, \dots, L'_{n-1} in order we get

$$\max_{i,j} |u_{ij}| \leq 2^{n-1} \max_{i,j} |(PA)_{ij}| \leq 2^{n-1} \max_{i,j} |a_{ij}|.$$

This completes the proof. □

Remarks. Theorem 4.10 and lemma 4.11 together show that algorithm LUPP and hence algorithm GEPP is backward stable.

Example. Let

$$A = \begin{pmatrix} 1 & & & 1 \\ -1 & 1 & & 1 \\ -1 & -1 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & -1 & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Since all relevant elements have modulus 1 we do not need to use pivoting and LU factorisation gives

$$U = \begin{pmatrix} 1 & & & 1 \\ & 1 & & 2 \\ & & 1 & 4 \\ & & & \ddots \\ & & & & 1 & 2^{n-2} \\ & & & & & 2^{n-1} \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Thus for the matrix A as defined above we get

$$g_n(A) = \frac{\max_{ij} |u_{ij}|}{\max_{ij} |a_{ij}|} = 2^{n-1}.$$

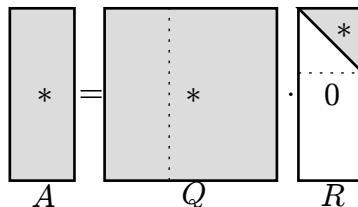
The example shows that the bound from lemma 4.11 is sharp. Thus the constant for the backward error in theorem 4.10 can grow exponentially fast in n . We showed that GEPP is stable but there are matrices where algorithm GEPP does not work very well! Since experience shows that these matrices “never” occur in practice the GEPP method is nevertheless commonly used.

4.3 The QR-Factorisation

The Householder QR-factorisation is another method to solve SLE. QR-factorisation avoids the issues of the LU factorisation presented at the end of the previous section, but the computation takes about double the number of operations. The method is based on the following theorem.

Theorem 4.12 (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.*

Remarks. The factorisation in the theorem is called *full QR-factorisation*. Since all entries below the diagonal of R are 0, the columns $n+1, \dots, m$ of Q do not contribute to the product QR . Let $\tilde{Q} \in \mathbb{C}^{m \times n}$ consist of the first n columns of Q and $\tilde{R} \in \mathbb{C}^{n \times n}$ 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.



Proof. Let $a_1, \dots, a_n \in \mathbb{C}^m$ be the columns of A and $q_1, \dots, q_m \in \mathbb{C}^m$ be the columns of Q . The proof is based on the Gram-Schmidt orthonormalisation method to construct Q and R :

1: **for** $j=1, \dots, n$ **do**

```

2:   $r_{kj} = \langle q_k, a_j \rangle$ 
3:   $\hat{q}_j = a_j - \sum_{k=1}^{j-1} r_{kj} q_k$ 
4:   $r_{jj} = \|\hat{q}_j\|_2$ 
5:  if  $r_{jj} > 0$  then
6:     $q_j = \hat{q}_j / r_{jj}$ 
7:  else
8:    let  $q_j$  be an arbitrary normalised vector orthogonal to  $q_1, \dots, q_{j-1}$ 
9:  end if
10: end for
11: choose  $q_{n+1}, \dots, q_m$  to make  $q_1, \dots, q_m$  an orthonormal system.

```

This algorithm calculates the columns q_1, \dots, q_m of the matrix Q and the entries of R which are on or above the diagonal. The entries of R below the diagonal are 0. For the matrices Q and R we get

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

and thus $A = QR$.

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

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

Thus induction shows that the columns of Q are orthonormal and that Q is unitary. \square

Remarks. 1) The Gram-Schmidt orthonormalisation used in the proof is numerically unstable and should not be used to calculate a QR-factorisation in practice.

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.

The following algorithm solves the problem SLE using QR-factorisation. In order to apply it we will need a numerically stable method to calculate the QR-factorisation and we need to calculate the matrix-vector product Q^*b .

Algorithm (solving SLE by QR-factorisation).

input: $A \in \mathbb{C}^{n \times n}$ non-singular

output: $x \in \mathbb{C}^n$ 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 $x \in \mathbb{C}^n$ 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.

$$\text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0, \text{ and} \\ -1 & \text{else.} \end{cases}$$

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** $k = 1, \dots, n - 1$ **do**
- 3: $u = (r_{kk}, \dots, r_{mk}) \in \mathbb{R}^{m-k+1}$
- 4: $\bar{v} = \text{sign}(u_1) \|u\|_2 e_1 + u$ where $e_1 = (1, 0, \dots, 0) \in \mathbb{R}^{m-k+1}$
- 5: $v = \bar{v} / \|\bar{v}\|_2$
- 6: $H_k = (I_{m-k+1} - 2vv^*) \in \mathbb{R}^{(m-k+1) \times (m-k+1)}$
- 7: $Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & H_k \end{pmatrix}$
- 8: $R = Q_k \cdot R$
- 9: $Q = Q \cdot Q_k$
- 10: **end for**

Remarks. 1) The algorithm calculates matrices Q_k with $Q_k = Q_k^*$ for $k = 1, \dots, n - 1$ as well as $R = Q_{n-1} \cdots Q_1 A$ and $Q = Q_1 \cdots Q_{n-1}$.

2) We will see that $Q_k \cdots Q_1 A$ has zeros below the diagonal in columns $1, \dots, k$ and thus the final result $R = Q_{n-1} \cdots Q_1 A$ is upper triangular.

3) 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

$$Q_k \cdot \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & H_k R_{22} \end{pmatrix} \quad (4.3)$$

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 the current section 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 $k \in \{1, \dots, n - 1\}$.

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 k th column of R and thus $Q_k R$ has zeros below the diagonal in columns $1, \dots, k$. 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 $u \approx -\|u\|_2 e_1$.

Computational Cost

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 4.13. *The computational cost $C(m, n)$ for algorithm QR applied to an $m \times n$ -matrix without calculating Q or Q^*b is asymptotically*

$$C(m, n) \sim 2mn^2 - \frac{2}{3}n^3 \quad \text{for } m, n \rightarrow \infty \text{ with } m = \Theta(n).$$

Proof. We count the number of operations for the individual steps of algorithm QR. From equation (4.3) 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

$$H_k R_{22} = R_{22} - \frac{\bar{v}}{\bar{v}^* \bar{v}/2} \bar{v}^* R_{22}. \quad (4.4)$$

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 $(n - k + 1)((m - k + 1) + (m - k))$ 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 $(\dots)(\bar{v}^* R_{22})$ from this needs $(m - k + 1)(n - k + 1)$ multiplications.
- Calculating $R_{22} - (\dots)$ requires $(m - k + 1)(n - k + 1)$ subtractions.

Thus the total operation count is

$$\begin{aligned} C(m, n) &= \sum_{k=1}^{n-1} 5(m - k + 1) + 1 + (n - k + 1)(4(m - k + 1) - 1) \\ &= \sum_{l=m-n+2}^m 5l + 1 + (n - m + l)(4l - 1) \\ &= 2mn^2 - \frac{2}{3}n^3 + \text{terms with at most two factors } m, n \\ &\sim 2mn^2 - \frac{2}{3}n^3 \end{aligned}$$

for $m, n \rightarrow \infty$ with $m = \Theta(n)$. □

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 $m, n \rightarrow \infty$ 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

$$C(n) \sim 2nn^2 - \frac{2}{3}n^3 = \frac{4}{3}n^3 \quad \text{for } n \rightarrow \infty.$$

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

Theorem 4.14. a) For $A \in \mathbb{R}^{m \times n}$ let $\tilde{R}, \tilde{v}_1, \dots, \tilde{v}_{n-1}$ be the computed result of the Householder QR-algorithm. Let

$$\tilde{Q}_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & I_{m-k+1} - 2\tilde{v}_k\tilde{v}_k^* \end{pmatrix}$$

and $\tilde{Q} = \tilde{Q}_1 \cdots \tilde{Q}_{n-1}$. Then we have $\tilde{Q}\tilde{R} = A + \Delta A$ for some $\Delta A \in \mathbb{R}^{m \times n}$ with

$$\frac{\|\Delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_m).$$

b) Let \tilde{y} be the computed value for \tilde{Q}^*b . Then

$$(\tilde{Q}^* + \Delta Q)\tilde{y} = b, \quad \|\Delta Q\| = \mathcal{O}(\varepsilon_m).$$

Remarks. 1) The values $\tilde{v}_1, \dots, \tilde{v}_{n-1}$ in the theorem denote the value calculated for the vector v in step 5 during iteration k . Since the matrices Q_k are not explicitly calculated (we use formula (4.4) instead), $\tilde{v}_1, \dots, \tilde{v}_{n-1}$ are the relevant computed values. For our analysis we consider the matrices \tilde{Q}_k which are exactly calculated from the vectors v_k and thus are exactly orthogonal.

2) Note that part b) of the theorem only gives an absolute error for \tilde{Q}^* , which seems not very useful at first sight. The following theorem about backward stability of the Householder algorithm shows how we can use this nevertheless.

Theorem 4.15. The Householder algorithm for solving SLE is backward stable: the computed solution \tilde{x} for $Ax = b$ satisfies

$$(A + \Delta \bar{A})\tilde{x} = b, \quad \frac{\|\Delta \bar{A}\|}{\|A\|} = \mathcal{O}(\varepsilon_m).$$

Proof. Let \tilde{Q}, \tilde{R} and \tilde{y} be as in theorem 4.14. The final result \tilde{x} is computed from $\tilde{R}\tilde{x} = \tilde{y}$ using back substitution. From the backward stability of the back substitution algorithm we get that the calculated \tilde{x} satisfies

$$(\tilde{R} + \Delta R)\tilde{x} = \tilde{y}, \quad \frac{\|\Delta R\|}{\|\tilde{R}\|} = \mathcal{O}(\varepsilon_m).$$

This gives

$$\begin{aligned} b &= (\tilde{Q} + \Delta Q)\tilde{y} \\ &= (\tilde{Q} + \Delta Q)(\tilde{R} + \Delta R)\tilde{x} \\ &= (\tilde{Q}\tilde{R} + \Delta Q\tilde{R} + \tilde{Q}\Delta R + \Delta Q\Delta R)\tilde{x} \\ &= (A + \Delta \bar{A})\tilde{x} \end{aligned}$$

where $\Delta \bar{A} = \Delta A + \Delta Q\tilde{R} + \tilde{Q}\Delta R + \Delta Q\Delta R$ and ΔA is the error of the computed QR-factorisation from part a) of theorem 4.14.

We study the four terms in the definition of $\Delta \bar{A}$ one by one. From theorem 4.14 we know $\|\Delta A\|/\|A\| = \mathcal{O}(\varepsilon_m)$. Since \tilde{Q} is orthogonal we have $\tilde{R} = \tilde{Q}^*(A + \Delta A)$ and thus

$$\frac{\|\tilde{R}\|}{\|A\|} \leq \|\tilde{Q}^*\| \cdot \frac{\|A + \Delta A\|}{\|A\|} = \mathcal{O}(1).$$

Therefore we also get

$$\frac{\|\Delta Q \tilde{R}\|}{\|A\|} \leq \|\Delta Q\| \cdot \mathcal{O}(1) = \mathcal{O}(\varepsilon_m).$$

Similarly we find

$$\frac{\|\tilde{Q} \Delta R\|}{\|A\|} \leq \|\tilde{Q}\| \frac{\|\Delta R\|}{\|\tilde{R}\|} \frac{\|\tilde{R}\|}{\|A\|} = \mathcal{O}(\varepsilon_m)$$

and

$$\frac{\|\Delta Q \Delta R\|}{\|A\|} \leq \|\Delta Q\| \frac{\|\Delta R\|}{\|\tilde{R}\|} \frac{\|\tilde{R}\|}{\|A\|} = \mathcal{O}(\varepsilon_m^2).$$

Together this shows $\|\Delta \bar{A}\|/\|A\| = \mathcal{O}(\varepsilon_m)$. □

The theorem shows that the Householder method for solving SLE is backward stable. For simplicity we did not consider the constants in the stability estimate $\|\Delta \bar{A}\|/\|A\| = \mathcal{O}(\varepsilon_m)$. A more detailed analysis shows that the constant here does not suffer from the exponential growth problem described at the end of section 4.2

Exercises

1) 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 $\Theta(n^2)$.

2) a) Find the LU factorisation of

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}.$$

What is $g_n(A)$ 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.

3) 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 $\det(H)$?

4) Determine the asymptotic computational cost of algorithm QR when calculating the full matrix Q .

5) 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.

Chapter 5

Iterative Methods

Until now we considered methods which directly calculate $x = A^{-1}b$ using $\Theta(n^3)$ operations. In this chapter we will introduce the idea of iterative methods. These methods construct a sequence $(x_k)_{k \in \mathbb{N}}$ with

$$x_k \rightarrow x \quad \text{for } k \rightarrow \infty.$$

For special matrices A the error $\|x_k - x\|$ becomes reasonably small after less than $\Theta(n^3)$ operations.

5.1 Linear Methods

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} for $k \in \mathbb{N}$, we can define x_k by

$$Mx_k = b - Nx_{k-1}. \quad (5.1)$$

If we assume for the moment that $\lim_{k \rightarrow \infty} x_k = x$, then the limit x satisfies

$$Mx = \lim_{k \rightarrow \infty} Mx_k = \lim_{k \rightarrow \infty} b - Nx_{k-1} = b - Nx$$

and thus for the limit we get $Ax = b$. This shows that the only possible limit for this sequence is the solution of SLE.

To study convergence for the method we consider the error $e_k = x_k - x$ where x is the exact solution of SLE. We get the recursive relation

$$Me_k = Mx_k - Mx = (b - Nx_{k-1}) - (A - N)x = -Ne_{k-1}$$

and thus $e_k = -M^{-1}Ne_{k-1}$. The method converges if $e_k \rightarrow 0$ for $k \rightarrow \infty$.

Before we settle the question of convergence in theorem 5.4 we need a few theoretical tools. The first of these is the Jordan canonical form of a matrix. We state the result without a proof.

Definition 5.1. 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 $i, j = 1, \dots, n$. 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 $\sum_{j=1}^k n_j = n$.

Proof. Assume first $\rho(C) < 1$. Then by lemma 5.3 we find an induced matrix norm with $\|C\| < 1$ and we get

$$\|e_k\| = \|C^k e_0\| \leq \|C\|^k \|e_0\| \longrightarrow 0$$

for $k \rightarrow \infty$.

On the other hand, if $\rho(C) \geq 1$, then there is an $e_0 \in \mathbb{C}^n$ with $Ce_0 = \lambda e_0$ for some $\lambda \in \mathbb{C}$ with $|\lambda| \geq 1$. For this vector e_0 we get

$$\|e_k\| = \|C^k e_0\| = |\lambda|^k \|e_0\|$$

and thus e_k does not converge to 0. □

Remarks. 1) The theorem shows that the linear iterative method defined by (5.1) converges if and only if $\rho(C) < 1$ for $C = -M^{-1}N$. The convergence is fast if $\rho(C)$ is small.

2) Since $\|A\| \geq \rho(A)$ for every matrix norm $\|\cdot\|$, a sufficient criterion for convergence of an iterative method is $\|C\| < 1$ for any matrix norm $\|\cdot\|$.

In the remaining part of the section we will consider specific methods which are obtained by choosing the matrices M and N in (5.1). For a matrix $A \in \mathbb{C}^{n \times n}$ define the three $n \times n$ -matrices

$$L = \begin{pmatrix} a_{21} & & & & \\ a_{31} & a_{32} & & & \\ \vdots & \cdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & \end{pmatrix}, \quad D = \begin{pmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & a_{33} & & \\ & & & \ddots & \\ & & & & a_{nn} \end{pmatrix},$$

$$U = \begin{pmatrix} & a_{12} & a_{13} & \cdots & a_{1n} \\ & & a_{23} & \cdots & a_{2n} \\ & & & \ddots & \vdots \\ & & & & a_{n-1,n} \end{pmatrix}. \quad (5.3)$$

Then we have $A = L + D + U$.

The Jacobi Method

The iterative method obtained by choosing $M = D$ and $N = L + U$ in (5.1) is called *Jacobi method*. This choice of M and N leads to the iteration

$$x_k = D^{-1}(b - (L + U)x_{k-1})$$

for all $k \in \mathbb{N}$ and the convergence properties are determined by the matrix $C = -M^{-1}N = -D^{-1}(L + U)$. Since D is diagonal, the inverse D^{-1} is trivial to compute.

Theorem 5.5. a) *The Jacobi method is convergent for all matrices A with*

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad (5.4)$$

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*

$$|a_{jj}| > \sum_{i \neq j} |a_{ij}| \quad (5.5)$$

for $j = 1, \dots, n$. (This condition is called the strong column sum criterion.)

Proof. a) The matrix $C = -D^{-1}(L + U)$ has entries

$$c_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & \text{if } i \neq j, \text{ and} \\ 0 & \text{else.} \end{cases}$$

Using theorem 1.16 we find

$$\|C\|_{\infty} = \max_{i=1, \dots, n} \frac{1}{|a_{ii}|} \sum_{j=1}^n |a_{ij}|.$$

Thus the strong row sum criterion gives $\|C\|_{\infty} < 1$ which implies $\rho(C) < 1$ and the method converges.

b) If the strong column sum criterion (5.5) holds for A , then the strong row sum criterion (5.4) holds for A^* and thus the method converges for A^* . From theorem 5.4 we know then $\rho(-D^{-1}(U^* + L^*)) < 1$. Since for every matrix C the matrices C , C^* and also $D^{-1}C^*D$ have the same eigenvalues we get $\rho(-D^{-1}(U + L)) = \rho(-D^{-1}(U^* + L^*)) < 1$ and the method converges for A . \square

Definition 5.6. A matrix $A \in \mathbb{C}^{n \times n}$ is called *irreducible* if there is no permutation matrix P such that

$$P^T A P = \begin{pmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ 0 & \tilde{A}_{22} \end{pmatrix}$$

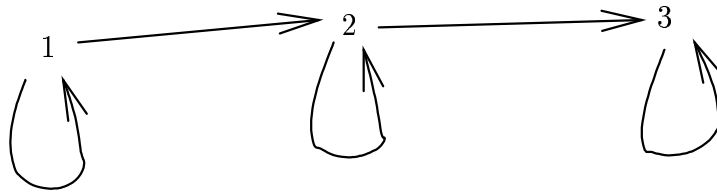
where $\tilde{A}_{11} \in \mathbb{C}^{p \times p}$, $\tilde{A}_{12} \in \mathbb{C}^{p \times q}$, $\tilde{A}_{22} \in \mathbb{C}^{q \times q}$ and 0 is a $q \times p$ zero-matrix with $p + q = n$ and $p, q > 0$.

There is an alternative description of irreducibility, which is often easier to check than the definition. To A associate the oriented graph $G(A)$ with vertices $1, \dots, n$ and edges $i \rightarrow j$ 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

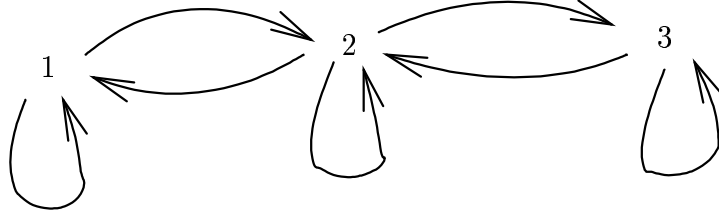


The matrix A is not irreducible (indeed $P = I$ in the definition is enough to see this) and since there is no path from 3 to 1, the graph $G(A)$ is not connected.

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) The graphs $G(A)$ and $G(P^T AP)$ are isomorphic, only the vertices are numbered in a different way. 2) The block \hat{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 \hat{A}_{11} .

Theorem 5.7. *If A is irreducible and satisfies*

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad (5.6)$$

for $i = 1, \dots, n$ but

$$|a_{kk}| > \sum_{j \neq k} |a_{kj}| \quad (5.7)$$

for one index k , then the Jacobi method converges. (This condition is called the weak row sum criterion.)

Proof. We have to show that $\rho(C) < 1$ where $C = -D^{-1}(L + U)$. Define the matrix $|C| = (|c_{ij}|)_{ij} \in \mathbb{R}^{n \times n}$ and the vector $e = (1, 1, \dots, 1) \in \mathbb{R}^n$. Then we have

$$|C|e = \left(\sum_{j=1}^n |c_{ij}| \cdot 1 \right)_i = \left(\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} \right)_i \leq e$$

where this and some of the following \leq -signs are to be read componentwise. Therefore we get

$$|C|^{l+1}e \leq |C|^l e \leq \dots \leq e$$

for all $l \in \mathbb{N}$.

Let $t^{(l)} = e - |C|^l e \geq 0$. 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 $\tau_{l+1} = \tau_l = k < n$. Since one row of A satisfies the strict inequality (5.7) we have $|C|e \neq e$ 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

$$\begin{aligned} \begin{pmatrix} b \\ 0 \end{pmatrix} &= t^{(l+1)} = e - |C|^{l+1}e \\ &\geq |C|e - |C|^{l+1}e = |C|t^{(l)} \\ &= \begin{pmatrix} |C_{11}| & |C_{12}| \\ |C_{21}| & |C_{22}| \end{pmatrix} \begin{pmatrix} a \\ 0 \end{pmatrix}. \end{aligned}$$

From $a > 0$ we have $|C_{21}| = 0$ and thus $C_{21} = 0$ and C would not be irreducible. Since this implies that A would not be irreducible, we have found the required contradiction and can conclude $\tau_{l+1} > \tau_l$ whenever $\tau_l < n$. This proves $t^{(n)} > 0$ componentwise.

Because of $e > |C|^n e$ we get

$$\rho(C^n) \leq \|C^n\|_\infty = \||C^n|\|_\infty \leq \||C|^n\|_\infty = \max_{i=1,\dots,n} (|C|^n e)_i < 1$$

and thus $\rho(C) < 1$. This completes the proof. \square

Computational Complexity

The sequence $(x_k)_{k \in \mathbb{N}}$ from the Jacobi method is defined by the relation

$$x_k = -D^{-1}(b - (L + U)x_{k-1}).$$

For each iteration we have to calculate

- 1) the product $(L + U)x_{k-1}$
- 2) the difference $b - \dots$ (needs n subtractions)
- 3) the product $D^{-1} \dots$ (needs n divisions because D is diagonal)

For general matrices step 1) needs $\Theta(n^2)$ operations and thus dominates the computational cost. Iterative methods are most powerful if the matrix A is sparse, i.e. if it contains many zeros, since then the matrix-vector multiplication can be performed using fewer operations.

Example. Assume that each row of A contains at most ℓ non-zero entries outside the diagonal. Then step 1) requires $\mathcal{O}(\ell n)$ operations and performing one step of Jacobi method has cost $\mathcal{O}(\ell n)$.

The next question is of course how many steps of the method we have to perform. Given $\varepsilon > 0$ we can calculate x_1, \dots, x_k until

$$\frac{\|Ax_k - b\|}{\|b\|} \leq \varepsilon.$$

From the conditioning of SLE (lemma 3.3) we then know

$$\frac{\|x_k - x\|}{\|x\|} \leq \kappa(A)\varepsilon \tag{5.8}$$

and by choosing ε small enough we can make the relative error of the solution arbitrarily small.

Also we know that the absolute error satisfies $x_k - x = C^k(x_0 - x)$. If the matrix A is symmetric, then $C = -D^{-1}(L + U)$ is normal and from theorem 1.14 we get

$$\|x_k - x\|_2 \leq \|C^k\| \|x_0 - x\|_2 = \rho(C)^k \|x_0 - x\|_2. \tag{5.9}$$

Thus the error converges exponentially fast to 0 as $k \rightarrow \infty$. If we know both the spectral radius $\rho(C)$ and the condition number $\kappa(A)$ we can combine estimate (5.9) with the conditioning of the problem to get an upper bound on the number of remaining iteration steps until the relative error of the result will satisfy (5.8).

Remarks. 1) If the matrix A is known explicitly, the above arguments can be used to derive more specific results about the computational complexity of the Jacobi method.

2) For large sparse matrices A the intermediate results in the LU or QR-factorisation will often not be sparse. If there is not enough memory in the system to store a full matrix, then it might still be possible to use iterative methods whereas direct methods will fail.

The Gauss-Seidel Method

If we use $M = L + D$ and $N = U$ for the matrices L , D and U from (5.3) then we still have $A = M + N$ and we get the *Gauss-Seidel method*. In this situation the iterative scheme (5.1) has the form

$$(L + D)x_k = b - Ux_{k-1}.$$

Since $(L + D)$ is lower triangular we can use forward substitution to calculate x_k in each step. The convergence properties of this method are determined by the matrix $C = -(L + D)^{-1}U$.

Theorem 5.8. *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 method converges.

5.2 The Conjugate-Gradient Method

Let $A \in \mathbb{R}^{n \times n}$ be positive definite, $c \in \mathbb{R}^n$ and $x = A^{-1}b$. Consider the norm $\|z\|_A = \sqrt{z^T A z}$ and define $F: \mathbb{R}^n \rightarrow \mathbb{R}$ by

$$F(z) = \frac{1}{2} \|z - x\|_A^2 \tag{5.10}$$

for all $z \in \mathbb{R}^n$. The function F can also be expressed as a function for the residuals $r = b - Az$: for every $z \in \mathbb{R}^n$ we get

$$\begin{aligned} F(z) &= \frac{1}{2} (z - x)^T A (z - x) \\ &= \frac{1}{2} z^T A z - b^T z + \frac{1}{2} b^T A b \\ &= \frac{1}{2} (b - Az)^T A (b - Az). \end{aligned}$$

Since x is the unique minimum of F we can solve SLE by minimising F . We will convert this idea into an iterative method by taking this minimum over an increasing sequence of subspaces.

Idea. Given x_k , find x_{k+1} with

$$F(x_{k+1}) = \min_{u_0, \dots, u_k} F(x_k + u_0 r_0 + \dots + u_k r_k)$$

where $F(z) = \frac{1}{2} \|z - x\|_A^2$ and $r_k = b - Ax_k$.

Algorithm CG (Conjugate-Gradient method).

input: $A \in \mathbb{R}^{n \times n}$ positive definite, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$

output: $x_k \in \mathbb{R}^n$ with $x_k = A^{-1}b$

- 1: $p_0 = b - Ax_0$, $r_0 = b - Ax_0$
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: **if** $p_k = 0$ **then**
- 4: return x_k
- 5: **else**
- 6: $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
- 7: $x_{k+1} = x_k + \alpha_k p_k$
- 8: $r_{k+1} = r_k - \alpha_k A p_k$
- 9: $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
- 10: $p_{k+1} = r_{k+1} + \beta_k p_k$
- 11: **end if**
- 12: **end for**

Remarks. At first sight this seems not to be a proper algorithm, since it is not clear whether it terminates after a finite number of steps. We will later see that (in the absence of rounding errors) we get $p_k = 0$ for some $k < n$ and thus the algorithm really terminates.

Lemma 5.9. *Consider the algorithm CG. Let $m = \min\{k \mid p_k = 0\}$. Then for all $0 \leq k \leq m$ the following statement holds.*

$$(A_k) \left\{ \begin{array}{l} (1) \ r_j \perp p_i \text{ for all } 0 \leq i < j \leq k \\ (2a) \ r_i^T r_i > 0 \text{ for all } 0 \leq i < k \\ (2b) \ r_i^T p_i = r_i^T r_i \text{ for all } 0 \leq i \leq k \\ (3) \ p_i^T A p_j = 0 \text{ for all } 0 \leq i < j \leq k \\ (4) \ r_i^T r_j = 0 \text{ for all } 0 \leq i < j \leq k \\ (5) \ r_i = b - A x_i \text{ for all } 0 \leq i \leq k \end{array} \right.$$

Proof. We use induction over k . First note that the claim (A_0) is trivially true. Assume that (A_k) holds for some $k < m$. We show properties (1) to (5) for (A_{k+1}) .

(1) For $i = k$ we get

$$r_{k+1}^T p_k = (r_k - \alpha_k A p_k)^T p_k = r_k^T p_k - \frac{r_k^T r_k}{p_k^T A p_k} p_k^T A p_k = r_k^T p_k - r_k^T r_k = 0 \quad (5.11)$$

where the last equality comes from (2b) of (A_k) . For $i < k$ we can use (1) and (3) to get

$$r_{k+1}^T p_i = (r_k - \alpha_k A p_k)^T p_i = 0.$$

(2a) If we had $r_k = 0$, then we had $p_k = \beta_{k-1} p_{k-1}$ and using property (3) we would get the contradiction

$$0 < p_k^T A p_k = \beta_{k-1} p_{k-1}^T A p_{k-1} = 0.$$

Thus we have $r_k \neq 0$ and consequently $r_k^T r_k > 0$.

(2b) Using (5.11) we get $r_{k+1}^T p_{k+1} = r_{k+1}^T (r_{k+1} + \beta_k p_k) = r_{k+1}^T r_{k+1}$.

(3) Applying the definitions of r_{k+1} and p_{k+1} from the algorithm we find

$$\begin{aligned} p_i^T A p_{k+1} &= p_i^T A (r_{k+1} + \beta_k p_k) \\ &= \frac{1}{\alpha_i} (r_i - r_{i+1})^T A r_{k+1} + \beta_k p_i^T A p_k \\ &= \frac{1}{\alpha_i} (p_i - \beta_{i-1} p_{i-1} - p_{i+1} + \beta_i p_i)^T r_{k+1} + \beta_k p_i^T A p_k. \end{aligned}$$

For $i < k$ the right-hand side is zero because of properties (1) and (3). For $i = k$ we can use the definitions of α_i and β_k as well as properties (1) and (2) to see that the right hand side still vanishes. Thus we get $p_i^T A p_{k+1}$ for every $i < k + 1$.

(4) From the definition of p_{k+1} and property (1) we find $r_i^T r_{k+1} = (p_i - \beta_{i-1} p_{i-1})^T r_{k+1} = 0$.

(5) By definition of x_{k+1} and r_{k+1} we get $b - A x_{k+1} = b - A (x_k + \alpha_k p_k) = r_k - \alpha_k A p_k = r_{k+1}$.

The claim of the lemma follows by induction. \square

Remarks. 1) Since the lemma shows that the vectors $r_0, \dots, r_{m-1} \in \mathbb{R}^n$ are non-zero and orthogonal, we have $m < n$. From $p_m = 0$ we can conclude $r_m^T r_m = r_m^T p_m = 0$ and thus $r_m = 0$. Therefore the algorithm terminates after at most n steps and on termination we have $A x_m = b$.

2) When the computation is performed on a computer, rounding errors will cause $\hat{p}_m \neq 0$ where \hat{p}_m is the calculated value for p_m . In practice one just treats the method as an iterative method and continues the iteration until the residual error $\|r_k\|$ is small enough.

3) From the construction of the vectors r_k and p_k in the algorithm we find $\text{span}(r_0, \dots, r_k) = \text{span}(p_0, \dots, p_k)$. Define

$$\Phi(u_0, \dots, u_k) = F\left(x_k + \sum_{i=0}^k u_i p_i\right)$$

for all $u_0, \dots, u_k \in \mathbb{R}$, where F is given by (5.10) and x_k is the value constructed in step k of the algorithm. Since Φ is positive and grows to the outside, it has only one minimum and there the gradient is zero. We show that x_{k+1} is the value which minimises Φ . We get

$$\frac{\partial}{\partial u_j} \Phi(u_0, \dots, u_k) = \left(x_k + \sum_{i=1}^k u_i p_i - x\right)^T A p_j = -r^T p_j$$

where $r = A(x - x_k - \sum_{i=1}^k u_i p_i) = b - A(x_k + \sum_{i=1}^k u_i p_i)$. For $(u_0, u_1, \dots, u_k) = (0, 0, \dots, \alpha_k)$ we get

$$x_k + \sum_{i=1}^k u_i p_i = x_k + \alpha_k p_k = x_{k+1}$$

and

$$r = b - A x_{k+1} = r_{k+1}.$$

Using this value of r and conditions (2b) and (1) from the lemma we also find

$$\frac{\partial}{\partial u_j} \Phi(u_0, \dots, u_k) = r_{k+1}^T p_j = 0$$

for $j = 1, \dots, k$. Together this shows that we really have

$$F(x_{k+1}) = \min_{u_0, \dots, u_k} \Phi(u_0, \dots, u_k).$$

4) From the algorithm we see $p_k \in \text{span}(r_0, A r_0, \dots, A^k r_0)$ for every $k < m$ and thus we get

$$\text{span}(p_0, \dots, p_k) = \text{span}(r_0, A r_0, \dots, A^k r_0) =: \mathcal{K}_{k+1}(r_0, A).$$

The space $\mathcal{K}_{k+1}(r_0, A)$ is called a *Krylov subspace* of \mathbb{R}^n for the matrix A .

Theorem 5.10 (speed of convergence). *The CG-method satisfies*

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \cdot \|e_0\|_A$$

where $e_k = x_k - x$ and $\kappa(A)$ is the condition number of A in the $\|\cdot\|_2$ -norm.

Exercises

1) Which of the following matrices are irreducible? For which of these matrices does the Jacobi-method converge?

$$\begin{pmatrix} 4 & 2 & 1 \\ & 2 & 1 \\ & & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 3 \\ 3 & 25 & 4 \\ & 4 & 1 \end{pmatrix}, \quad \begin{pmatrix} 8 & & \\ 4 & 2 & \\ 2 & 1 & 1 \end{pmatrix}$$

2) 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} .

Chapter 6

Least Square Problems

In this chapter we want to solve the following 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 $\|Ax - b\|_2$.

6.1 Singular Value Decomposition

Definition 6.1. 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, $\Sigma \in \mathbb{C}^{m \times n}$ is diagonal, and the diagonal entries of Σ are $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ where $p = \min(m, n)$. The values $\sigma_1, \dots, \sigma_p$ are called *singular values* of A .

Theorem 6.2. *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 $\min(m, n)$. If $m = 0$ or $n = 0$ the matrices U , V , and Σ are just the appropriately shaped empty matrices (one dimension is zero) and nothing is to show.

Assume $\min(m, n) > 0$ 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 $\|\cdot\|_2: \mathbb{C}^n \rightarrow \mathbb{R}$ is continuous it realises its supremum: there is an $v_1 \in \mathbb{C}^n$ 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 $\{v_1, \dots, v_n\}$ of \mathbb{C}^n and $\{u_1\}$ to an orthonormal basis $\{u_1, \dots, u_m\}$ of \mathbb{C}^m . Consider the matrices

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

and

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

Then the product $U_1^*AV_1$ is of the form

$$S = U_1^*AV_1 = \begin{pmatrix} \sigma_1 & w^* \\ 0 & \\ \vdots & B \\ 0 & \end{pmatrix}$$

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

For unitary matrices U we have

$$\|Ux\|_2^2 = \langle Ux, Ux \rangle = \langle x, U^*Ux \rangle = \langle x, x \rangle = \|x\|_2^2$$

and thus

$$\|S\|_2 = \max_{x \neq 0} \frac{\|U_1^*AV_1x\|_2}{\|x\|_2} = \max_{x \neq 0} \frac{\|AV_1x\|_2}{\|V_1x\|_2} = \|A\|_2 = \sigma_1.$$

On the other hand we get

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

and thus $\|S\|_2 \geq (\sigma_1^2 + w^*w)^{1/2}$. Together this allows us to conclude $w = 0$ and thus

$$A = U_1SV_1^* = U_1 \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} V_1^*.$$

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

$$B = U_2\Sigma_2V_2^*.$$

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 $\sigma_2, \dots, \sigma_n$ follows by induction as above. \square

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 $A = U\Sigma V^*$:

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, \dots, 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 $\|A\|_2 = \sigma_1$.

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 $\sigma_1 \geq \dots \geq \sigma_r > 0 = \dots = 0$. To illustrate the usefulness of the SVD we prove a few basic results.

Theorem 6.3. *The rank of A is equal to r .*

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

Theorem 6.4. *We have $\text{range}(A) = \text{span}(u_1, \dots, u_r)$ and $\text{ker}(A) = \text{span}(v_{r+1}, \dots, v_n)$.*

Proof. a) Since Σ is diagonal and V^* is invertible we have

$$\text{range}(\Sigma V^*) = \text{range}(\Sigma) = \text{span}(e_1, \dots, e_r) \subseteq \mathbb{C}^m.$$

This shows

$$\text{range}(A) = \text{range}(U\Sigma V^*) = \text{span}(u_1, \dots, u_r) \subseteq \mathbb{C}^m.$$

b) We have

$$\text{ker}(A) = \text{ker}(U\Sigma V^*) = \text{ker}(\Sigma V^*).$$

Since V is orthogonal we can conclude

$$\text{ker}(A) = \text{span}(v_{r+1}, \dots, v_n) \subseteq \mathbb{C}^n.$$

□

Theorem 6.5. *If $A = A^*$, then the singular values of A are $|\lambda_1|, \dots, |\lambda_n|$ where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of A .*

6.2 The Normal Equations

In this section we will first derive a theoretical criterion to find the solution of LSQ and then we will derive three different algorithms to solve LSQ.

Theorem 6.6. *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 $\text{range}(A)$.*

Proof. Let $g(x) = \frac{1}{2}\|Ax - b\|_2^2$. Then minimising $\|Ax - b\|_2$ is equivalent to minimising the function g .

(1) Assume that $Ax - b$ is orthogonal to $\text{range}(A)$ 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 .

(2) Now assume that the vector x minimises g . Then for every $y \in \mathbb{C}^n$ we have

$$0 = \frac{\partial}{\partial \lambda} g(x + \lambda y) = \frac{1}{2} \left(\langle Ay, Ax - b \rangle + \langle Ax - b, Ay \rangle \right) = \text{Re} \langle Ax - b, Ay \rangle$$

and

$$0 = \frac{\partial}{\partial \lambda} g(x + \lambda iy) = \frac{1}{2} \left(-i \langle Ay, Ax - b \rangle + i \langle Ax - b, Ay \rangle \right) = i \text{Im} \langle Ax - b, Ay \rangle.$$

This shows $\langle Ax - b, Ay \rangle = 0$ and thus $Ax - b \perp Ay$ for all $y \in \mathbb{C}^n$. □

Corollary 6.7. *A vector $x \in \mathbb{C}^n$ solves LSQ if and only if*

$$A^* Ax = A^* b. \tag{6.1}$$

Proof. From the theorem we know that x solves LSQ if and only if $Ax - b \perp \text{range}(A)$. This in turn is equivalent to $Ax - b \perp a_i$ for every column a_i of A , i.e. to $A^*(Ax - b) = 0$. \square

Definition 6.8. The system (6.1) of linear equations is called *the normal equations* for LSQ.

We will consider different algorithms to solve LSQ. The first one is directly based on the normal equations.

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 $(A^*A)x = A^*b$

Remarks. Calculating A^*A and A^*b requires asymptotically $\sim 2mn^2$ operations for $m, n \rightarrow \infty$. Since A^*A is symmetric we only need to calculate half of the entries, which can be done in $\sim mn^2$ operations. Solving $(A^*A)x = A^*b$ with QR-factorisation requires $\frac{4}{3}n^3$ operations. Together this gives a total asymptotic cost of order

$$C(m, n) \sim mn^2 + \frac{4}{3}n^3.$$

There are methods to solve SLE for symmetric matrices with fewer steps. Using Cholesky factorisation one gets a total asymptotic cost of order $C(m, n) \sim mn^2 + \frac{1}{3}n^3$.

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

$$A^*Ax = A^*\hat{Q}\hat{R}x = A^*\hat{Q}\hat{Q}^*b = A^*b$$

and thus solves LSQ. Steps 1 and 2 together have asymptotic cost $C_1(m, n) \sim 2mn^2 - \frac{2}{3}n^3$, and step 3 has asymptotic cost $C_2(m, n) = \mathcal{O}(n^2)$. Thus we get total asymptotic cost

$$C(m, n) \sim 2mn^2 - \frac{2}{3}n^3$$

for $m, n \rightarrow \infty$ with $m = \Theta(n)$.

Algorithm (LSQ via SVD).

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 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.

Remarks. 1) The three algorithms were presented in order of increasing stability.

2) The ratio of computational cost between the first two algorithms depends on the ratio between m and n . The algorithm using SVD is more expensive than the other two.

6.3 Conditioning of LSQ

As before let $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, $\text{rank}(A) = n$ and $b \in \mathbb{C}^m$. From corollary 6.7 we know that the solution of LSQ can be calculated as

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

Definition 6.9. The matrix $A^+ = (A^*A)^{-1}A^*$ is called the *pseudo-inverse* of $A \in \mathbb{C}^{m \times n}$.

Definition 6.10. For $A \in \mathbb{C}^{m \times n}$ define the *condition number* of A to be

$$\kappa(A) = \|A\| \|A^+\|.$$

Remarks. If $\text{rank}(A) < n$ then A^*A is not invertible and we set $\kappa(A) = +\infty$. Since for square, invertible matrices A we have $A^+ = A^{-1}$, the definition of $\kappa(A)$ is consistent with the previous one. As before the condition number depends on the chosen matrix norm.

Lemma 6.11. The condition number in $\|\cdot\|_2$ -norm of a matrix $A \in \mathbb{C}^{m \times n}$ is

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

Proof. Using the singular value decomposition of A we find

$$A^+ = (A^*A)^{-1}A^* = (V\Sigma^*\Sigma V^*)^{-1}V\Sigma^*U^* = V(\Sigma^*\Sigma)^{-1}\Sigma^*U^*. \quad (6.2)$$

The matrix $(\Sigma^*\Sigma)^{-1}\Sigma^* \in \mathbb{C}^{n \times m}$ is diagonal with diagonal elements

$$\frac{1}{\sigma_i^2}\sigma_i = \frac{1}{\sigma_i}$$

for $i = 1, \dots, n$. Thus equation (6.2) is (modulo ordering of the singular values) a singular value decomposition of A^+ and we find

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

□

Theorem 6.12. Assume that x solves LSQ for b and $x + \Delta x$ solves LSQ for $b + \Delta b$. Define $\vartheta \in [0, \pi/2]$ by $\cos(\vartheta) = \frac{\|Ax\|_2}{\|b\|_2}$ and let $\eta = \|A\| \|x\| / \|Ax\| \geq 1$. Then we have

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

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

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

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

□

Remarks. The constant $\kappa(A)/\eta \cos(\vartheta)$ becomes large if either $\kappa(A)$ is large or $\vartheta \approx \pi/2$. In either of these cases the problem is badly conditioned.

Theorem 6.13. *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*

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

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

Theorem 6.14. *The algorithm to solve LSQ via Householder QR-factorisation is backward stable: the computed solution \hat{x} minimises $\|(A + \Delta A)\hat{x} - b\|_2$ for some matrix ΔA with*

$$\frac{\|\Delta A\|_2}{\|A\|_2} = \mathcal{O}(\varepsilon_m).$$

Theorems 6.13 and 6.14 together give estimates for the accuracy of the computed result x , given that ϑ is bounded away from $\pi/2$.

Exercises

1) By following the proof of the existence of a singular value decomposition, find the SVD for the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}.$$

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 $i = 1, \dots, m$. 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 $b \in \mathbb{R}^m$.

Chapter 7

The Eigenvalue Problem

A vector $x \in \mathbb{C}^n$ is an eigenvector of a matrix $A \in \mathbb{C}^{n \times n}$ with eigenvalue $\lambda \in \mathbb{C}$ if

$$Ax = \lambda x, \quad x \neq 0.$$

In this chapter we will learn to solve the following problem: given A , compute the eigenvalues and eigenvectors.

Definition 7.1. 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 7.2. 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 $\det(A - \lambda I) = 0$. \square

This shows that, if we can find the roots of arbitrary polynomials, we can also find the eigenvalues of arbitrary matrices. We will now see that the two problems are even equivalent, i.e. that for every polynomial we can find a matrix with this polynomial as the characteristic polynomial. Let $p(z) = a_0 + a_1z + \cdots + a_{n-1}z^{n-1} + z^n$ be given. Define $A \in \mathbb{C}^{n \times n}$ by

$$A = \begin{pmatrix} 0 & & & -a_0 \\ 1 & & & -a_1 \\ & \ddots & & \vdots \\ & & \ddots & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{pmatrix}. \quad (7.1)$$

By induction one can show that $\rho_A(z) = \det(A - zI) = (-1)^n p(z)$. This shows that the problem of finding eigenvalues is equivalent to the problem of finding roots of a polynomial.

Theorem 7.3 (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 only using rational numbers, addition, subtraction, multiplication, division and taking k th roots.

As the computed result of a computer program will always be based on the operations mentioned in the theorem, we find that it is not possible to find an algorithm which calculates eigenvalues exactly after a finite number of steps. Conclusion: any eigenvalue solver must be iterative.

Theorem 7.4. 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 $\Lambda \in \mathbb{R}^{n \times n}$ with

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

Remarks. 1) The orthonormal columns of Q are the eigenvectors of A and the diagonal entries of Λ are the corresponding eigenvalues.

2) The theorem shows that Hermitian matrices have real eigenvalues.

The iterative methods presented in this chapter will calculate approximations to eigenvectors. The following consideration will help to get an approximation for the corresponding eigenvalue from this. Given $A \in \mathbb{C}^{n \times n}$ and $x \in \mathbb{C}^n$ we try to find the $\alpha \in \mathbb{C}$ which minimises $\|Ax - \alpha x\|_2$. If x is an eigenvector then the minimum is attained for the corresponding eigenvalue. Otherwise we consider the normal equations: in the distance

$$\|Ax - \alpha x\|_2 = \|x \cdot \alpha - Ax\|_2$$

we consider x to be a $n \times 1$ -matrix, $\alpha \in \mathbb{C}^1$ to be the unknown "vector" and $Ax \in \mathbb{C}^n$ to be the right hand side. Then according to corollary 6.7 the minimum is attained for

$$\alpha = (x^* x)^{-1} x^* (Ax) = \frac{\langle x, Ax \rangle}{\langle x, x \rangle}.$$

Definition 7.5. 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 $x \in \mathbb{C}^n$.

Theorem 7.6. Let $A \in \mathbb{R}^{n \times n}$ be symmetric and $x \in \mathbb{R}^n$ with $x \neq 0$. Then x is an eigenvector of A with eigenvalue λ if and only if $\nabla r_A(x) = 0$ and $r_A(x) = \lambda$.

Proof. The gradient of r_A can be calculated as

$$\begin{aligned} \nabla r_A(x) &= \left(\frac{\partial}{\partial x_i} \frac{\sum_{j,k=1}^n x_j a_{jk} x_k}{\sum_{j=1}^n x_j^2} \right)_{i=1, \dots, n} \\ &= \left(\frac{(\sum_{k \neq i} a_{ik} x_k + \sum_{j \neq i} x_j a_{ji} + 2a_{ii} x_i) \sum_{j=1}^n x_j^2 - \sum_{j,k} x_j a_{jk} x_k \cdot 2x_i}{(\sum_{j=1}^n x_j^2)^2} \right)_{i=1, \dots, n} \\ &= \frac{2Ax \cdot \langle x, x \rangle - 2\langle x, Ax \rangle \cdot x}{\langle x, x \rangle^2 x} \\ &= \frac{2}{\|x\|_2^2} (Ax - r_A(x) \cdot x). \end{aligned}$$

Assume $Ax = \lambda x$. Then $r_A(x) = \langle x, \lambda x \rangle / \langle x, x \rangle = \lambda$ and

$$\nabla r_A(x) = \frac{2}{\|x\|_2^2} (\lambda x - \lambda x) = 0.$$

If on the other hand $\nabla r_A(x) = 0$, then $Ax - r_A(x)x = 0$ and thus we get $Ax = r_A(x) \cdot x$. \square

For the remaining part of the chapter let x_1, \dots, x_n be a orthonormal system of eigenvectors and $\lambda_1, \dots, \lambda_n$ be the corresponding eigenvalues of a real, symmetric matrix A . We order the eigenvalues such that $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$.

Algorithm (power iteration).

input: $A \in \mathbb{R}^{n \times n}$ symmetric with $|\lambda_1| > |\lambda_2|$

output: $z^{(k)} \in \mathbb{R}^n$, $\lambda^{(k)} \in \mathbb{R}$ with $z^{(k)} \approx x_1$ and $\lambda^{(k)} \approx \lambda_1$

1: choose $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 = 1$

2: **for** $k=1,2,3,\dots$ **do**

3: $w^{(k)} = Az^{(k-1)}$

4: $z^{(k)} = \frac{w^{(k)}}{\|w^{(k)}\|_2}$

5: $\lambda^{(k)} = \langle z^{(k)}, Az^{(k)} \rangle$

6: **end for**

Remarks. 1) For practical usage the method, as is every iterative method, is stopped at some point where the result is “close enough” to the real one.

2) The algorithm calculates $z^{(k)} = A^k z^{(0)} / \|A^k z^{(0)}\|_2$ and $\lambda^{(k)} = r_A(z^{(k)})$. 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, \dots, 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.

Theorem 7.7. *Let $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$ and $\langle z^{(0)}, x_1 \rangle \neq 0$. Then there is a sequence $(\sigma^{(k)})_{k \in \mathbb{N}}$ with $\sigma^{(k)} \in \{-1, +1\}$ for all $k \in \mathbb{N}$ such that the sequences $(z^{(k)})$ and $(\lambda^{(k)})$ from the power iteration algorithm satisfy*

$$\|z^{(k)} - \sigma^{(k)} x_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) \quad (7.2)$$

and

$$|\lambda^{(k)} - \lambda_1| = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right). \quad (7.3)$$

Proof. a) Let x_1, \dots, x_n be an orthonormal system of eigenvectors with eigenvalues $\lambda_1, \dots, \lambda_n$ and

$$z^{(0)} = \sum_{i=1}^n \alpha_i x_i.$$

Since $\alpha_1 = \langle z^{(0)}, x_1 \rangle \neq 0$, we get

$$A^k z^{(0)} = \sum_{i=1}^n \alpha_i \lambda_i^k x_i = \alpha_1 \lambda_1^k \left(x_1 + \sum_{i=2}^n \frac{\alpha_i}{\alpha_1} \left|\frac{\lambda_i}{\lambda_1}\right|^k x_i \right)$$

and Pythagoras' theorem gives

$$\|A^k z^{(0)}\|_2^2 = |\alpha_1 \lambda_1^k|^2 \left(1 + \sum_{i=2}^n \left(\frac{\alpha_i}{\alpha_1}\right)^2 \left|\frac{\lambda_i}{\lambda_1}\right|^{2k} \right) \leq |\alpha_1 \lambda_1^k|^2 \left(1 + \left|\frac{\lambda_2}{\lambda_1}\right|^{2k} \sum_{i=2}^n \left(\frac{\alpha_i}{\alpha_1}\right)^2 \right).$$

Using the Taylor approximation $\sqrt{1+x^2} \approx 1 + x^2/2$ we can conclude

$$\|A^k z^{(0)}\|_2 = |\alpha_1 \lambda_1^k| \left(1 + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right) \right).$$

Now define $\sigma^{(k)} = \text{sgn}(\alpha_1 \lambda_1^k)$. Then

$$\left\| \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} - \sigma^{(k)} x_1 \right\|_2^2 = \left\| \frac{A^k z^{(0)}}{\alpha_1 \lambda_1^k} - x_1 \right\|_2^2 = \sum_{i=2}^n \left(\frac{\alpha_i}{\alpha_1}\right)^2 \left|\frac{\lambda_i}{\lambda_1}\right|^{2k} = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right)$$

and thus

$$\begin{aligned} \|z^{(k)} - \sigma^{(k)} x_1\|_2 &\leq \left\| \frac{A^k z^{(0)}}{\|A^k z^{(0)}\|_2} - \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} \right\|_2 + \left\| \frac{A^k z^{(0)}}{|\alpha_1 \lambda_1^k|} - \sigma^{(k)} x_1 \right\|_2 \\ &= \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right) + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) \end{aligned}$$

This finishes the proof of (7.2).

b) From theorem 7.6 we know $r_A(\sigma^{(k)}x_1) = \lambda_1$ and $\nabla r_A(\sigma^{(k)}x_1) = 0$. Taylor expansion of r_A around $\sigma^{(k)}x_1$ gives

$$\begin{aligned} r_A(z) &= r_A(x_1) + \langle \nabla r_A(\sigma^{(k)}x_1), z - \sigma^{(k)}x_1 \rangle + \mathcal{O}(\|z - \sigma^{(k)}x_1\|_2^2) \\ &= \lambda_1 + 0 + \mathcal{O}(\|z - \sigma^{(k)}x_1\|_2^2). \end{aligned}$$

Using (7.2) we get

$$|\lambda^{(k)} - \lambda_1| = |r_A(z^{(k)}) - \lambda_1| = \mathcal{O}(\|z^{(k)} - \sigma^{(k)}x_1\|_2^2) = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right).$$

This completes the proof of relation (7.3). \square

The power iteration algorithm helps to find the eigenvalue with the largest modulus and the corresponding eigenvector. The method can be modified to find different eigenvalues of A . This is done in the following algorithm.

Algorithm (inverse iteration).

input: $A \in \mathbb{R}^{n \times n}$ symmetric, $\lambda \in \mathbb{R}$

output: $z^{(k)} \in \mathbb{R}^n$, $\lambda^{(k)} \in \mathbb{R}$ with $z^{(k)} \approx x_j$ and $\lambda^{(k)} \approx \lambda_j$

where λ_j is the eigenvalue closest to λ

- 1: choose $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 = 1$
- 2: **for** $k=1,2,3,\dots$ **do**
- 3: solve $(A - \lambda I)w^{(k)} = z^{(k-1)}$
- 4: $z^{(k)} = \frac{w^{(k)}}{\|w^{(k)}\|_2}$
- 5: $\mu^{(k)} = \langle z^{(k)}, Az^{(k)} \rangle$
- 6: **end for**
- 7: return $\lambda^{(k)} = \lambda + \frac{1}{\mu^{(k)}}$.

Remarks. 1) For practical usage the method is stopped at some point where the result is “close enough” to the real one.

2) By comparison with the power iteration method we find that $\mu^{(k)}$ approximates the eigenvalue of $(A - \lambda I)^{-1}$ with the largest modulus. Since $(A - \lambda I)^{-1}$ has eigenvalues $\mu_i = 1/(\lambda_i - \lambda)$ this value is μ_j where λ_j is the eigenvalue closest to λ and we get

$$\lambda^{(k)} = \lambda + \frac{1}{\mu^{(k)}} \approx \lambda + \frac{1}{\mu_j} = \lambda + (\lambda_j - \lambda) = \lambda_j.$$

Using this argument it is easy to convert theorem 7.7 into a theorem about the speed of convergence for the inverse iteration algorithm.

The power iteration method only works if the initial vector $z^{(0)}$ satisfies the condition $\langle z^{(0)}, x_1 \rangle \neq 0$. This is no problem since $\dim\{z \in \mathbb{R}^n \mid \langle z, x \rangle = 0\} = n - 1 < n$. The probability of hitting this hyperplane with a randomly chosen initial vector $z^{(0)}$ is zero if the distribution has a density with respect to the Lebesgue measure on \mathbb{R}^n and also at least one of the basis vectors e_1, \dots, e_n satisfies the condition $\langle e_i, x_1 \rangle \neq 0$.

The following algorithm extends the idea of the power iteration algorithm: it runs the power iteration method for n orthonormal vectors simultaneously, re-orthonormalising them at every step. The result is an algorithm which approximates all eigenvectors and all eigenvalues at once.

Algorithm (simultaneous iteration).

input: $A \in \mathbb{R}^{n \times n}$ symmetric

output: $Q^{(k)}, \Lambda^{(k)} \in \mathbb{R}^{n \times n}$

with $Q^{(k)} \approx (x_1, x_2, \dots, x_n)$ and $\Lambda_{ii}^{(k)} \approx \lambda_i$ for $i = 1, \dots, n$

- 1: choose an orthogonal matrix $Q^{(0)} \in \mathbb{R}^{n \times n}$
- 2: **for** $k = 1, 2, 3, \dots$ **do**
- 3: $W^{(k)} = AQ^{(k-1)}$
- 4: calculate the QR-factorisation $W^{(k)} = Q^{(k)}R^{(k)}$
- 5: $\Lambda^{(k)} = (Q^{(k)})^T W^{(k)} Q^{(k)}$
- 6: **end for**

Theorem 7.8. Let $A \in \mathbb{R}^{n \times n}$ be symmetric with eigenvalues $\lambda_1, \dots, \lambda_n$ satisfying $\lambda_1 > \dots > \lambda_n$. Assume $\langle q_i^{(0)}, x_i \rangle \neq 0$ for $i = 1, \dots, n$. Then there are sequences $(\sigma_i^{(k)})_{k \in \mathbb{N}}$ for $i = 1, \dots, n$ with $\sigma_i^{(k)} \in \{+1, -1\}$ for all i, k with

$$\|q_i^{(k)} - \sigma_i^{(k)} x_i\|_2 = \mathcal{O}\left(|\xi|^k\right)$$

and

$$|\Lambda_{ii}^{(k)} - \lambda_i| = \mathcal{O}\left(|\xi|^{2k}\right)$$

for $i = 1, \dots, n$ where $q_1^{(k)}, \dots, q_n^{(k)}$ are the columns of $Q^{(k)}$ and $\xi = \max_{i=1, \dots, n-1} |\lambda_i| / |\lambda_{i+1}|$.

Remarks. Since the matrices $R^{(k)}$ are upper-triangular, the first column of $Q^{(k)}$ in step 4 of the algorithm is a multiple of the first column of the matrix $W^{(k)}$. Thus the first column $q_1^{(k)}$ of the matrix $Q^{(k)}$ performs the original power iteration algorithm with initial vector $q_1^{(0)}$.

Exercises

- 1) Give a proof by induction which shows that the matrix A from (7.1) really has determinant $(-1)^n p(z)$ where $p(z) = a_0 + a_1 z + \dots + a_{n-1} z^{n-1} + z^n$.
- 2) Give a proof of theorem 7.8.

Bibliography

- [Dem97] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
- [Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.
- [HJ85] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
- [SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, third edition, 2002.
- [TB97] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.