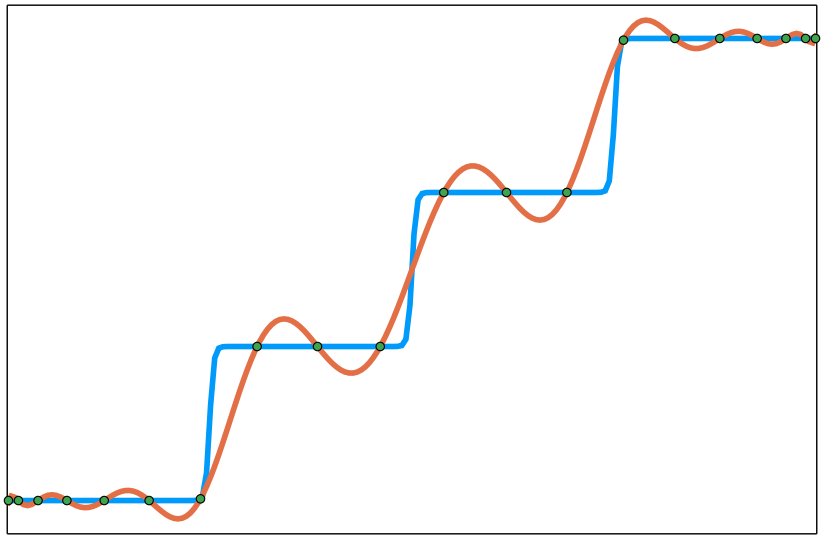
MATH-UA 9252: Numerical Analysis



Urbain Vaes
urbain.vaes@nyu.edu

NYU Paris, Fall term 2026

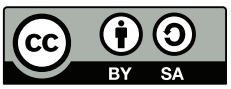
Weekly schedule:

- Lectures on Monday and Wednesday from 15:30 to 16:45 in room 406;
- Recitation on Monday and Wednesday from 16:50 to 17:35 in room 406;
- Office hour on Monday from 14:30 to 15:30 in room 606.

License

The copyright of these notes rests with the author and their contents are made available under a Creative Commons "Attribution-ShareAlike 4.0 Interational" license. You are free to copy, distribute, transform and build upon the course material under the following terms:

- **Attribution.** You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **ShareAlike.** If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.



Course syllabus

Course content. This course is aimed at giving a first introduction to classical topics in numerical analysis, including floating point arithmetics and round-off errors, the numerical solution of linear and nonlinear equations, iterative methods for eigenvalue problems, interpolation and approximation of functions, and numerical quadrature. If time permits, we will also cover numerical methods for solving ordinary differential equations.

Prerequisites. The course assumes a basic knowledge of linear algebra and calculus. Prior programming experience in Julia, Python or a similar language is desirable but not required.

Study goals. After the course, the students will be familiar with the key concepts of *stability*, *convergence* and *computational complexity* in the context of numerical algorithms. They will have gained a broad understanding of the classical numerical methods available for performing fundamental computational tasks, and be able to produce efficient computer implementations of these methods.

Education method. The weekly schedule comprises two lectures ($2 \times$ 1h15 per week) and an exercise session (1h30 per week). The course material includes rigorous proofs as well as illustrative numerical examples in the Julia programming language, and the weekly exercises blend theoretical questions and practical computer implementation tasks.

Assessment. Computational homework will be handed out on a weekly or biweekly basis, each of them focusing on one of the main topics covered in the course. The homework assignments will count towards 70% of the final grade, and the final exam will count towards 30%.

Literature and study material. A comprehensive reference for this course is the following textbook: A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume **37** of *Texts in Applied Mathematics*. Springer-Verlag, Berlin, 2007. Other pointers to the literature will be given within each chapter.

Acknowledgments

I am grateful to Vincent Legat, Tony Lelièvre, Gabriel Stoltz and Paul Van Dooren for allowing me to draw inspiration from their lectures notes in numerical analysis.

- Chapter 5 of these notes follows closely the structure of [15, Chapter 3].
- The presentation of the material in Chapter 2 is based on [7].
- Chapter 7 is based on [2, Chapter 2] and [15, Chapter 5].

I would also like to thank Jean-François Barthélémy and Khôi Nguyễn, who made contributions to several chapters, as well as the students who found several errors and typos in these notes.

Notations

Unless otherwise specified, we use the following notation throughout these notes.

- Lower case bold is used to denote vectors, e.g. $\boldsymbol{x} \in \mathbf{C}^n$, and upper case sans serif is used to denote matrices, e.g. $\mathsf{A} \in \mathbf{C}^{m \times n}$. The entries of a vector $\boldsymbol{x} \in \mathbf{C}^n$ are denoted by (x_i), and those of a matrix $\mathsf{A} \in \mathbf{C}^{m \times n}$ are denoted by (a_{ij}) or $(a_{i,j})$.
- The notations $\langle \bullet, \bullet \rangle$ and $\|\bullet\|$ without a subscript always refer to the Euclidean inner product (A.1) and induced norm.
- The sequence $x_1, x_2, \ldots$ is denoted by $(x_n)_{n \in \mathbf{N}}$, or sometimes just (x_n).
- The notation $B_R(\boldsymbol{y})$ refers to the open ball of radius R centered at $\boldsymbol{y}$.

Contents

Notations **iv**

1 Floating point arithmetic **4**
1.1 Binary representation of real numbers . . . 5
1.2 Set of values representable in floating point formats . . . 7
1.3 Arithmetic operations between floating point formats . . . 10
1.4 Encoding of floating point numbers . . . 13
1.5 Integer formats . . . 15
1.6 Exercises . . . 16
1.7 Discussion and bibliography . . . 24

2 Interpolation and approximation **26**
2.1 Interpolation . . . 27
2.2 Approximation . . . 41
2.3 Exercises . . . 49
2.4 Discussion and bibliography . . . 55

3 Numerical integration **56**
3.1 The closed Newton–Cotes method . . . 57
3.2 Composite methods with equidistant nodes . . . 58
3.3 Richardson extrapolation and Romberg's method . . . 65
3.4 Methods with non-equidistant nodes . . . 69
3.5 Introduction to probabilistic integration methods . . . 73
3.6 Exercises . . . 75
3.7 Discussion and bibliography . . . 81

4 Solution of linear systems of equation **82**
4.1 Conditioning . . . 83
4.2 Direct solution method . . . 87
4.3 Iterative methods for linear systems . . . 100
4.4 Exercises . . . 120
4.5 Discussion and bibliography . . . 129

5 Solution of nonlinear systems **130**

5.1 The bisection method . . . 131
5.2 Fixed point methods . . . 132
5.3 Convergence of fixed point methods . . . 133
5.4 Examples of fixed point methods . . . 137
5.5 A numerical experiment . . . 145
5.6 Exercises . . . 147
5.7 Discussion and bibliography . . . 150

6 Numerical computation of eigenvalues 151
6.1 Numerical methods for eigenvalue problems: general remarks . . . 152
6.2 Simple vector iterations . . . 152
6.3 Methods based on a subspace iteration . . . 157
6.4 Projection methods . . . 162
6.5 Exercises . . . 167
6.6 Discussion and bibliography . . . 172

7 Numerical ordinary differential equations 173
7.1 Analysis of the continuous problem . . . 174
7.2 One-step methods . . . 179
7.3 Multistep methods . . . 188
7.4 Absolute stability . . . 192
7.5 Exercises . . . 197

8 Optimization 199
8.1 Definition and characterization of convexity . . . 200
8.2 Unconstrained optimization . . . 202
8.3 Constrained optimization . . . 204

A Background material 207
A.1 Inner products and norms . . . 207
A.2 Completeness . . . 210
A.3 Contraction mappings and the Banach fixed point theorem . . . 211
A.4 Vector norms . . . 212
A.5 Matrix norms . . . 212
A.6 Diagonalization and spectral theorem . . . 214
A.7 Similarity transformation and Jordan normal form . . . 217
A.8 Oldenburger's theorem and Gelfand's formula . . . 218

B Brief introduction to Julia 220

C Chebyshev polynomials 229

Introduction

Goals of computer simulation

In a wide variety of scientific disciplines, ranging from physics to biology and economics, the phenomena under consideration are well-described by mathematical equations. More often than not, it is too difficult to solve these equations analytically, and so one has to recur to *computer simulation* in order to obtain approximate solutions. Computer simulation enables to gain understanding of the phenomena examined, to explain observations and to make predictions. It plays a crucial role in a number of practical applications including weather forecasting, drug discovery through molecular modeling, flight simulation, and structural engineering, to mention just a few.

Numerical simulation may also be employed in order to calibrate mathematical models of physical phenomena, particularly when observation through experiment is impractical or too costly. For example, it is frequently the case that the parameters in mathematical models for turbulence are estimated not from real data, but from synthetic data generated by computer simulation of the fundamental equations of fluid mechanics. Relying on "computer experiments" is attractive in this context because these enable to perform accurate measurements without disturbing the system being observed. Numerical simulation is also very useful to understand and build simplified models for physical phenomena at very small scales, if direct observation is beyond the capabilities of experimental physics.

The definition of numerical analysis

Numerical analysis sits at the interface between mathematics and computer science. Nick Trefethen, author of several influential works in mathematics, defines numerical analysis as *the study of algorithms for the problems of continuous mathematics* [14]. Devising and studying algorithms to solve mathematical problems is the central concern of numerical analysis and our main focus in this course. The word *continuous* in this definition is used to indicate that the problems in the realm of numerical analysis involve real or complex variables. Discrete problems, which involve variables that take finitely or countably many values, are usually studied in other fields of mathematics and computer science.

Sources of error in computational science

It is important for practitioners of computer simulation to be aware of the different sources of error likely to affect numerical results obtained in applications, which may be classified as follows:

- **Modeling error.** There may be a discrepancy between the mathematical model and the underlying physical phenomenon.
- **Data error.** The data of the problem, such as the initial conditions or the parameters entering the equations, are usually known only approximately.
- **Discretization error.** The *discretization* of mathematical equations, i.e. turning them into finite-dimensional problems amenable to computer simulation, adds another source of error.
- **Discrete solver error.** The method employed to solve the discretized equations, especially if it is of iterative nature, may also introduce an error.
- **Round-off errors.** Finally, the limited accuracy of computer arithmetics causes additional errors.

Of these, only the last three are in the domain of numerical analysis, and in this course we focus mainly on the *solver* and *round-off* errors. The order of magnitude of the overall error is dictated by the largest among the above sources of error.

Aims of this course

The aim of this course is to present the standard numerical methods for performing the tasks most commonly encountered in applications: the solution of linear and nonlinear systems of equations, the solution of eigenvalue problems, interpolation and approximation of functions, and numerical integration. For a given task, there are usually several numerical methods to choose from, and these often include parameters which must be fixed appropriately in order to guarantee a good efficiency. In order to guide these choices, we study carefully the *convergence* and *stability* of the various methods we present. Six topics will be covered in these lecture notes.

- **Floating point arithmetic.** In Chapter 1, we discuss how real numbers are represented, manipulated and stored on a computer. There is an uncountable infinity of real numbers, but only a finite subset of these can be represented exactly on a machine. This subset is specified in the *IEEE 754* standard, which is widely accepted today and employed in most programming languages, including `Julia`.
- **Interpolation and extrapolation of functions.** In Chapter 2, we focus on the topics of interpolation and approximation. *Interpolation* is concerned with the construction of a function within a given set, for example that of polynomials, that takes given values when evaluated at a discrete set of points. The aim of *approximation*, on the other hand,

is usually to determine, within a class of simple functions, which one is closest to a given function. Depending on the metric employed to measure closeness, this may or may not be a well-defined problem.

- **Numerical integration.** In Chapter 3, we study numerical methods for computing definite integrals. This chapter is strongly related to the previous one, as numerical approximations of the integral of a function are often obtained by first approximating the function, say by a polynomial, and then integrating this approximation exactly.

- **Solution of linear systems.** In Chapter 4, we study the standard numerical methods for solving linear systems. Linear systems are ubiquitous in science, often arising from the discretization of linear elliptic partial differential equations, which themselves govern a large number of physical phenomena including heat propagation, electromagnetism, gravitation and the deformation of solids.

- **Solution of nonlinear equations.** In Chapter 5, we present widely used methods for solving nonlinear equations. Like linear equations, nonlinear equations are omnipresent in science, a prime example being the Navier–Stokes equation describing the motion of fluid flows. Nonlinear equations are usually much more difficult to solve and require dedicated techniques.

- **Solution of eigenvalue problems.** In Chapter 6, we present and study the standard iterative methods for calculating the eigenfunctions and eigenvalues of a matrix. Eigenvalue problems have a large number of applications, for instance in quantum physics and vibration analysis. They are also at the root of the PageRank algorithm for ranking web pages, which played a key role in the early success of Google search.

Why Julia?

Throughout the course, the `Julia` programming language is employed to exemplify some of the methods and key concepts. In the author's opinion, the `Julia` language has several advantages compared to other popular languages in the context of scientific computing, such as `Matlab` or `Python`.

- Its main advantage over `Matlab` is that it is free and open source, with the byproduct that it benefits from contributions from a large number of contributors around the world. Additionally, `Julia` is a fully-fledged programming language that can be used for applications unrelated to mathematics.

- Its main advantages over `Python` are significantly better performance and a more concise syntax for mathematical operations, especially those involving vectors and matrices. It should be recognized, however, that although adoption of `Julia` is rapidly increasing, `Python` still enjoys a more mature ecosystem and is much more widely used.

Chapter 1

Floating point arithmetic

1.1 Binary representation of real numbers 5
1.1.1 Conversion between binary and decimal formats 6
1.2 Set of values representable in floating point formats 7
1.2.1 Denormalized floating point numbers 8
1.2.2 Relative error and machine epsilon 8
1.3 Arithmetic operations between floating point formats 10
1.4 Encoding of floating point numbers 13
1.5 Integer formats 15
1.6 Exercises 16
1.7 Discussion and bibliography 24

Introduction

When we study numerical algorithms in the next chapters, we assume implicitly that the operations involved are performed exactly. On a computer, however, only a subset of the real numbers can be stored and, consequently, many arithmetic operations are performed only approximately. This is the source of the so-called *round-off errors*. The rest of this chapter is organized as follows.

- In Section 1.1, we discuss the binary representation of real numbers.
- In Section 1.2, we describe the set of floating point numbers that can be represented in the usual floating point formats;
- In Section 1.3 we explain how arithmetic operations between floating point numbers behave. We insist in particular on the fact that, in a calculation involving several successive arithmetic operations, the result of each intermediate operation is stored as a floating point number, with a possible error.

- In Section 1.4, we briefly present how floating point numbers are encoded according to the IEEE 754 standard, which is widely adopted today. We discuss also the encoding of special values such as `Inf`, `-Inf` and `NaN`.
- Finally, in Section 1.5, we present the standard integer formats and their encoding.

In order to completely describe computer arithmetic, one would in principle need to also discuss the conversion mechanisms between different number formats, as well as a number of edge cases. A comprehensive discussion of the subject is beyond the scope of this course; our aim in this chapter is only to introduce the key concepts.

1.1 Binary representation of real numbers

Given any integer number $\beta \geqslant 2$, called the *base*, a real number x can always be expressed as a finite or infinite series of the form

$$x = \pm \sum_{k=-n}^{\infty} a_k \beta^{-k}, \qquad a_k \in \{0, \dots, \beta - 1\}. \tag{1.1}$$

The number x may then be denoted as $\pm(a_{-n}a_{-n+1}\dots a_{-1}a_0.a_1a_2\dots)_\beta$, where the subscript β indicates the base. This numeral system is called the *positional notation* and is universally used today, both by humans (usually with $\beta = 10$) and machines (usually with $\beta = 2$). If the base β is omitted, it is always assumed in this course that $\beta = 10$ unless otherwise specified – this is the *decimal* representation. The *digits* $a_{-n}, a_{-n+1}, \dots$ are also called *bits* if $\beta = 2$. In computer science, several bases other than 10 are regularly employed, for example the following:

- Base 2 (binary) is the usual choice for storing numbers on a machine. The binary format is convenient because the digits have only two possible values, 0 or 1, and so they can be stored using simple electrical circuits with two states. We employ the binary notation extensively in the rest of this chapter. Notice that, just like multiplying and dividing by 10 is easy in base 10, multiplying and dividing by 2 is very simple in base 2: these operations amount to shifting all the bits by one position to the left or right, respectively.
- Base 16 (hexadecimal) is sometimes convenient to represent numbers in a compact manner. In order to represent the values 0-15 with a single digit, 16 different symbols are required, which are conventionally denoted by $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. With this notation, we have $(FF)_{16} = (255)_{10}$, for example.

  The hexadecimal notation is often used in programming languages for describing colors specified by a triplet (r, g, b) of values between 0 and 255, corresponding to the primary colors *red*, *green* and *blue*. The number of possible values for each component is $256 = 16^2$, and so only 2 digits are required to represent these in the hexadecimal notation. Hexadecimal numbers are also employed in IPv6 addresses, which are used to identify computers connected to a network.

1.1.1 Conversion between binary and decimal formats

Obtaining the decimal representation of a binary number can be achieved from (1.1), using the decimal representations of the powers of 2. Since all the positive and negative powers of 2 have a finite decimal representation, any real number with a finite representation in base 2 has a finite representation also in base 10. For example, $(0.01)_2 = (0.25)_{10}$ and $(0.111)_2 = (0.875)_{10}$.

Example 1.1 (Converting a binary number to decimal notation). Let us calculate the decimal representation of $x = (0.\overline{10})_2$, where the horizontal line indicates repetition: $x = (0.101010\ldots)_2$. By definition, it holds that

$$x = \sum_{k=1}^{\infty} a_k 2^{-k},$$

where $a_k = 1$ if k is odd and 0 otherwise. Thus, the series may be rewritten as

$$x = \sum_{k=0}^{\infty} 2^{-(2k+1)} = \frac{1}{2}\sum_{k=0}^{\infty}(2^{-2})^k.$$

We recognize on the right-hand side a geometric series with common ratio $r = 2^{-2} = \frac{1}{4}$, and so we obtain

$$x = \frac{1}{2}\left(\frac{1}{1-r}\right) = \frac{2}{3} = (0.\overline{6})_{10}.$$

Obtaining the binary representation of a decimal number is more difficult, because negative powers of 10 have *infinite* binary representations, as Exercise 1.4 demonstrates. There is, however, a simple procedure to perform the conversion, which we present for the specific case of a real number x with decimal representation of the form $x = (0.a_1 \ldots a_n)_{10}$. In this setting, the bits $(b_1, b_2, \ldots)$ in the binary representation of $x = (0.b_1 b_2 b_3 \ldots)_2$ may be obtained as follows:

Algorithm 1 Conversion of a number to binary format

```
1:  i ← 1
2:  while x ≠ 0 do
3:      x ← 2x
4:      if x ⩾ 1 then
5:          b_i ← 1
6:      else
7:          b_i ← 0
8:      end if
9:      x ← x − b_i
10:     i ← i + 1
11: end while
```

Example 1.2 (Converting a decimal number to binary notation). Let us calculate the binary representation of $x = \frac{1}{3} = (0.\overline{3})_{10}$. We apply Algorithm 1 and collate the values of i and x obtained at the beginning of each iteration, i.e. just before line 3, in the table below.

i	x	Result
1	$\frac{1}{3}$	0.**0**000...
2	$\frac{2}{3}$	0.0**1**00...
3	$\frac{1}{3}$	0.00**0**0...

Since x in the last row is again $\frac{1}{3}$, successive bits alternate between 0 and 1, and the binary representation of x is given by $(0.\overline{01})_2$. This is not surprising since $2x = (0.\overline{6})_{10} = (0.\overline{10})_2$, as we saw in Example 1.1.

1.2 Set of values representable in floating point formats

We mentioned in the introduction that, because of memory limitations, only a subset of the real numbers can be stored exactly in a computer. Nowadays, the vast majority of programming languages comply with the IEEE 754 standard, which requires that the set of representable numbers be of the form

$$\mathbf{F}(p, E_{\min}, E_{\max}) = \Big\{(-1)^s 2^E (b_0.b_1 b_2 \dots b_{p-1})_2 : \\ s \in \{0,1\}, b_i \in \{0,1\} \text{ and } E_{\min} \leqslant E \leqslant E_{\max}\Big\}. \tag{1.2}$$

In addition to these, floating number formats provide the special entities `Inf`, `-Inf` and `NaN`, the latter being an abbreviation for *Not a Number*. Three parameters appear in the set definition (1.2). The parameter $p \in \mathbf{N}_{>0}$ is the number of significant bits (also called the precision), and $(E_{\min}, E_{\max}) \in \mathbf{Z}^2$ are respectively the minimum and maximum exponents. From the precision, the *machine epsilon* is defined as $\varepsilon_M = 2^{-(p-1)}$; its significance is discussed in Section 1.2.2.

For a number $x \in \mathbf{F}(p, E_{\min}, E_{\max})$, s is called the *sign*, E is the *exponent* and $b_0.b_1 b_2 \dots b_{p-1}$ is the *significand*. The latter can be divided into a *leading bit* b_0 and the *fraction* $b_1 b_2 \dots b_{p-1}$, to the right of the binary point. The most widely used floating point formats are the *single* and *double precision* formats, which are called respectively `Float32` and `Float64` in Julia. Their parameters, together with those of the lesser-known half-precision format, are summarized in Table 1.1. In the rest of this section we use the shorthand notation $\mathbf{F}_{16}$, $\mathbf{F}_{32}$ and $\mathbf{F}_{64}$. Note that $\mathbf{F}_{16} \subset \mathbf{F}_{32} \subset \mathbf{F}_{64}$.

	Half precision	Single precision	Double precision
p	11	24	53
$E_{\min}$	-14	-126	-1022
$E_{\max}$	15	127	1023

Table 1.1: Floating point formats. The first column corresponds to the *half-precision* format. This format, which is available through the `Float16` type in Julia, is more recent than the single and double precision formats. It was introduced in the 2008 revision to the IEEE 754 standard of 1985, a revision known as IEEE 754-2008.

Remark 1.1. Some definitions, notably that in [10, Section 2.5.2], include a general base β instead of the base 2 as an additional parameter in the definition of the number format (1.2). Since the binary format ($\beta = 2$) is always employed in practice, we focus on this case for simplicity in most of this chapter.

Remark 1.2. Given a real number $x \in \mathbf{F}(p, E_{\min}, E_{\max})$, the exponent E and significand are generally not uniquely defined. For example, the number $2.0 \in \mathbf{F}_{64}$ may be expressed as $(-1)^0 2^1 (1.00\ldots00)_2$ or, equivalently, as $(-1)^0 2^2 (0.100\ldots00)_2$.

In `Julia`, non-integer number literals are interpreted as `Float64` by default, which can be verified by using the `typeof` function. For example, the instruction "`a = 0.1`" is equivalent to "`a = Float64(0.1)`". In order to define a number of type `Float32`, the suffix `f0` must be appended to the decimal expansion. For instance, the instruction "`a = 4.0f0`" defines a floating point number `a` of type `Float32`; it is equivalent to writing "`a = Float32(4.0)`".

1.2.1 Denormalized floating point numbers

We can decompose the set $\mathbf{F}(p, E_{\min}, E_{\max})$ in two disjoint parts:

$$\begin{aligned}\mathbf{F}(p, E_{\min}, E_{\max}) = \Big\{ & (-1)^s 2^E (1.b_1 b_2 \ldots b_{p-1})_2 : \\ & s \in \{0,1\}, b_i \in \{0,1\} \text{ and } E_{\min} \leqslant E \leqslant E_{\max} \Big\} \\ \cup \Big\{ & (-1)^s 2^{E_{\min}} (0.b_1 b_2 \ldots b_{p-1})_2 : \ s \in \{0,1\}, b_i \in \{0,1\} \Big\}.\end{aligned}$$

The numbers in the second set are called *subnormal* or *denormalized*.

1.2.2 Relative error and machine epsilon

Let x be a nonzero real number and $\hat{x}$ be an approximation. We define the absolute and relative errors of the approximation as follows.

Definition 1.1 (Absolute and relative error)**.** The absolute error is given by $|x - \hat{x}|$, whereas the relative error is

$$\frac{|x - \hat{x}|}{|x|}$$

The following result establishes a link between the machine ε_M and the relative error between a real number and the closest member of a floating point format.

Proposition 1.1. *Let $x_{\min}$ and $x_{\max}$ denote the smallest and largest* non-denormalized *positive numbers in a format $F = \mathbf{F}(p, E_{\min}, E_{\max})$. If $x \in [-x_{\max}, -x_{\min}] \cup [x_{\min}, x_{\max}]$, then*

$$\min_{\hat{x} \in F} \frac{|x - \hat{x}|}{|x|} \leqslant \frac{1}{2} 2^{-(p-1)} = \frac{1}{2} \varepsilon_M. \tag{1.3}$$

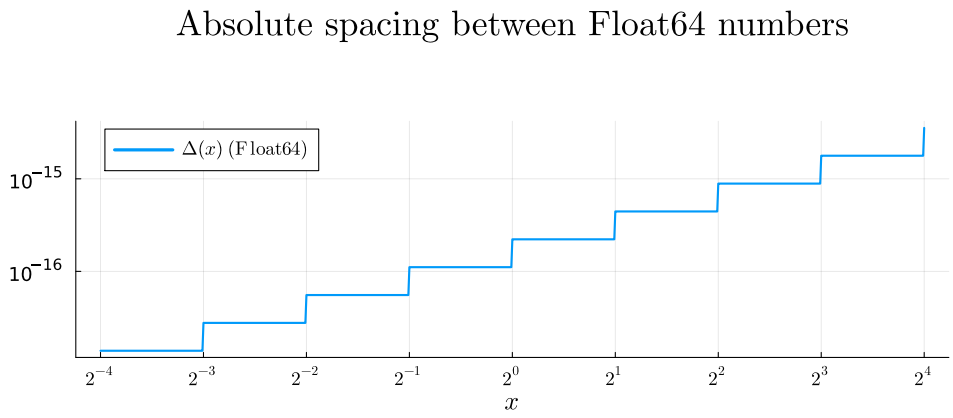


Figure 1.1: Absolute spacing between double-precision floating point numbers, for $x \in \mathbf{F}_{64}$. In this figure, $\Delta(x)$ denotes the distance between x and its successor in $\mathbf{F}_{64}$.

Proof. For simplicity, we assume that $x > 0$. Let $n = \lfloor \log_2(x) \rfloor$ and $y := 2^{-n}x$. Since $y \in [1, 2)$, it admits a binary representation of the form $(1.b_1 b_2 \dots)_2 \neq (1.\overline{1})_2$. Thus $x = 2^n(1.b_1 b_2 \dots)_2$, and from the assumption that $x_{\min} \leqslant x \leqslant x_{\max}$ we deduce that $E_{\min} \leqslant n \leqslant E_{\max}$. We now define the number $x_- \in F$ by truncating the binary expansion of x as follows:

$$x_- = 2^n(1.b_1 \dots b_{p-1})_2.$$

The distance between x_- and its successor in F, which we denote by x_+, is given by 2^{n-p+1}. Consequently, it holds that

$$(x_+ - x) + (x - x_-) = x_+ - x_- = 2^{n-p+1}.$$

Since both summands on the left-hand side are positive, this implies that either $x_+ - x$ or $x - x_-$ is bounded from above by $\frac{1}{2}2^{n-p+1} \leqslant \frac{1}{2}2^{-p+1}x$, which concludes the proof. □

The machine epsilon, which was defined as $\varepsilon_M = 2^{-(p-1)}$, coincides with the maximum relative spacing between a non-denormalized floating point number x and its successor in the floating point format, defined as the smallest number in the format that is strictly larger than x.

Figure 1.1 depicts the density of double-precision floating point numbers, i.e. the number of $\mathbf{F}_{64}$ members per unit on the real line. The figure shows that the density decreases as the absolute value of x increases. We also notice that the density is piecewise constant with discontinuities at powers of 2. Figure 1.2 illustrates the relative spacing between successive floating point numbers. Although the absolute spacing increases with the absolute value of x, the relative spacing oscillates between $\frac{1}{2}\varepsilon_M$ and ε_M.

The picture of the relative spacing between successive floating point numbers looks quite different for denormalized numbers. This is illustrated in Figure 1.3, which shows that the relative spacing increases beyond the machine epsilon in the denormalized range. Fortunately, in the usual $\mathbf{F}_{32}$ and $\mathbf{F}_{64}$ formats, the transition between denormalized and non-denormalized numbers occurs at such a small value that it rarely needs worrying about.

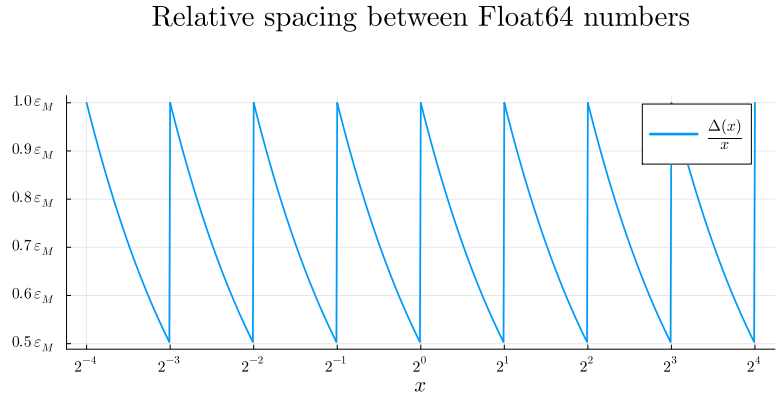


Figure 1.2: Relative spacing between successive double-precision floating point numbers in the "normal range". The relative spacing oscillates between $\frac{1}{2}\varepsilon_M$ and ε_M.

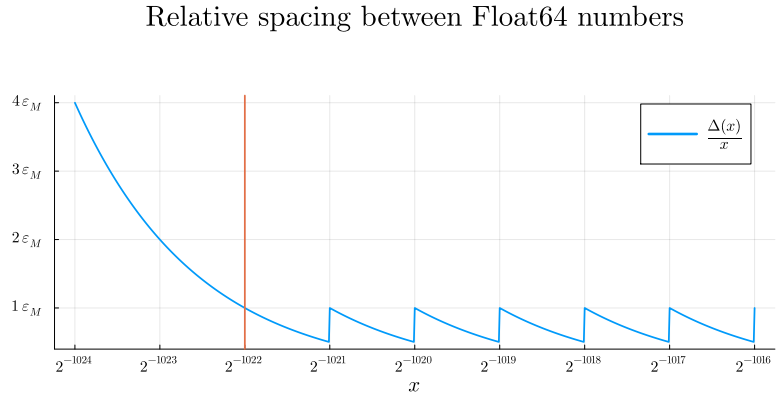


Figure 1.3: Relative spacing between successive double-precision floating point numbers, over a range which includes denormalized numbers. The vertical red line indicates the transition from denormalized to non-denormalized numbers.

Example 1.3. In Julia, the machine epsilon can be obtained using the `eps` function. For example, the instruction `eps(Float16)` returns ε_M for the half-precision format.

1.3 Arithmetic operations between floating point formats

Now that we have presented the set of values representable on a computer, we attempt in this section to understand precisely how arithmetic operations between floating point formats are performed. The key mechanism governing arithmetic operations on a computer is that of *rounding*, the action of approximating a real number regarded as infinitely precise by a number in a floating point format $\mathbf{F}(p, E_{\min}, E_{\max})$. The IEEE 754 standard stipulates that the default mechanism for rounding a real number x, called *round to nearest*, should behave as follows:

- **Standard case**: The number x is rounded to the *nearest representable number*, if this number is unique.
- **Edge case**: When there are two equally near representable numbers in the floating point format, the one with the least significant bit equal to zero is delivered.

- **Infinities**: If the real number x is larger than the largest representable number in the format, that is larger than $x_{\max} = 2^{E_{\max}}(2 - \varepsilon_M)$, then there are two cases,
  - If $x < 2^{E_{\max}}(2 - 2^{-p})$, then $x_{\max}$ is delivered;
  - Otherwise, the special value `Inf` is delivered.

  In other words, $x_{\max}$ is delivered if it would be delivered by following the rules of the first two bullet points in a different floating point format with the same precision but a larger exponent $E_{\max}$. A similar rule applies for large negative numbers.

When a binary arithmetic operation ($+$, $-$, $\times$, $/$) is performed on floating point numbers in format $\mathbf{F}$, the result delivered by the computer is obtained by rounding the exact result of the operation according to the rules given above. In other words, the arithmetic operation is performed as if the computer first calculated an intermediate exact result, and then rounded this intermediate result in order to provide a final result in $\mathbf{F}$.

Mathematically, arithmetic operations between floating point numbers in a given format $\mathbf{F}$ may be formalized by introducing the rounding operator $\mathrm{fl} : \mathbf{R} \to \mathbf{F}$ and by defining, for any binary operation $\circ \in \{+, -, \times, /\}$, the corresponding machine operation

$$\hat{\circ}\colon \mathbf{F} \times \mathbf{F} \to \mathbf{F}; (x, y) \mapsto \mathrm{fl}(x \circ y).$$

We defined this operator for arguments in the same floating point format $\mathbf{F}$. If the arguments of a binary arithmetic operation are of different types, the format of the end result, known as the *destination format*, depends on that of the arguments: as a rule of thumb, it is given by the most precise among the formats of the arguments. In addition, recall that a floating point literal whose format is not explicitly specified is rounded to double-precision format and so, for example, the addition `0.1 + 0.1` produces the result $\mathrm{fl}_{64}\big(\mathrm{fl}_{64}(0.1) + \mathrm{fl}_{64}(0.1)\big)$, where fl_{64} is the rounding operator to the double-precision format.

Example 1.4. Using the `typeof` function, we check that the floating point literal `1.0` is indeed interpreted as a double-precision number:

```
julia> a = 1.0; typeof(a)
Float64
```

When two numbers in different floating point formats are passed to a binary operation, the result is in the more precise format.

```
julia> typeof(Float16(1) + Float32(1))
Float32


julia> typeof(Float32(1) + Float64(1))
Float64
```

If a mathematical expression contains several binary arithmetic operations to be performed in succession, the result of each intermediate calculation is stored in a floating point format

dictated by the formats of its argument, and this floating point number is employed in the next binary operation. A consequence of this mechanism is that the machine operations $\hat{+}$ and $\hat{*}$ are generally *not associative*. For example, in general

$$(x \,\hat{+}\, y) \,\hat{+}\, z \neq x \,\hat{+}\, (y \,\hat{+}\, z)$$

Example 1.5. Let $x = 1$ and $y = 3 \times 2^{-13}$. Both of these numbers belong to $\mathbf{F}_{16}$ and, denoting by $\hat{+}$ machine addition in $\mathbf{F}_{16}$, we have

$$(x \,\hat{+}\, y) \,\hat{+}\, y = 1 \tag{1.4}$$

but

$$x \,\hat{+}\, (y \,\hat{+}\, y) = 1 + 2^{-10}. \tag{1.5}$$

To explain this somewhat surprising result, we begin by writing the normalized representations of x and y in the $\mathbf{F}_{16}$ format:

$$\begin{aligned} x &= (-1)^0 \times 2^0 \times (\texttt{1.0000000000})_2 \\ y &= (-1)^0 \times 2^{-12} \times (\texttt{1.1000000000})_2. \end{aligned}$$

The exact result of the addition $x + y$ is given by $r = 1 + 3 \times 2^{-13}$, which in binary notation is

$$r = (1.\underbrace{0000000000011}_{\text{11 zeros}})_2.$$

Since the length of the significand in the half-precision ($\mathbf{F}_{16}$) format is only $p = 11$, this number is not part of $\mathbf{F}_{16}$. The result of the machine addition $\hat{+}$ is therefore obtained by rounding r to the nearest member of $\mathbf{F}_{16}$, which is 1. This reasoning can then be repeated in order to conclude that, indeed,

$$(x \,\hat{+}\, y) \,\hat{+}\, y = x \,\hat{+}\, y = 1.$$

In order to explain the result of (1.5), note that the exact result of the addition $y + y$ is $r = 3 \times 2^{-12}$, which belongs to the floating point format, so it also holds that $y \,\hat{+}\, y = 3 \times 2^{-12}$. Therefore,

$$x \,\hat{+}\, (y \,\hat{+}\, y) = 1 \,\hat{+}\, 3 \times 2^{-12} = \mathrm{fl}_{16}(1 + 3 \times 2^{-12}).$$

The argument of the $\mathbf{F}_{16}$ rounding operator does not belong to $\mathbf{F}_{16}$, since its binary representation is given by

$$(1.\underbrace{0000000000011}_{\text{10 zeros}})_2.$$

This time the nearest member of $\mathbf{F}_{16}$ is given by $1 + 2^{-10}$.

When a numerical computation unexpectedly returns `Inf` or `-Inf`, we say that an *overflow error* occurred. Similarly, *underflow* is said to occur when a number is smaller in absolute value than the smallest positive representable number in a floating point format.

1.4 Encoding of floating point numbers

Once a number format is specified through parameters $(p, E_{\min}, E_{\max})$, the choice of encoding, i.e. the machine representation of numbers in this format, has no bearing on the magnitude and propagation of round-off errors. Studying encoding is, therefore, not essential for our purposes in this course, but we opted to cover the topic anyway in the hope that it will help the students build intuition on floating point numbers. We focus mainly on the single precision format, but the following discussion applies *mutatis mutandis* to the double and half-precision formats. The material in this section is for information purposes only.

We already mentioned in Remark 1.2 that a number in a floating point format may have several representations. On a computer, however, a floating point number is always stored in the same manner (except for the number 0, see Remark 1.4). The values of the exponent and significand which are selected by the computer, in the case where there are several possible choices, are determined from the following rules:

- Either $E > E_{\min}$ and $b_0 = 1$;
- Or $E = E_{\min}$, in which case the leading bit may be 0.

The following result proves that these rules enable to define the exponent and significand of a number in a set of floating point numbers uniquely.

Proposition 1.2. *Assume that*

$$(-1)^s 2^E (b_0.b_1 \dots b_{p-1})_2 = (-1)^{\tilde{s}} 2^{\tilde{E}} (\tilde{b}_0.\tilde{b}_1 \dots \tilde{b}_{p-1})_2, \tag{1.6}$$

where the parameter sets $(s, E, b_0, \dots, b_{p-1})$ and $(\tilde{s}, \tilde{E}, \tilde{b}_0, \dots, \tilde{b}_{p-1})$ both satisfy the above rule. Then $E = \tilde{E}$ and $b_i = \tilde{b}_i$ for $i \in \{0, \dots, p-1\}$.

Proof. We show that $E = \tilde{E}$, after which the equality of significands follows trivially. Let us assume for contradiction that $E > \tilde{E}$ and denote the left and right-hand sides of (1.6) by x and $\tilde{x}$, respectively. Then $E > E_{\min}$, implying that $b_0 = 1$ and so $2^E \leqslant |x| < 2^{E+1}$. On the other hand, it holds that $|\tilde{x}| < 2^{\tilde{E}+1}$ regardless of whether $\tilde{E} = E_{\min}$ or not. Since $E \geqslant \tilde{E} + 1$ by assumption, we deduce that $|\tilde{x}| < 2^E \leqslant |x|$, which contradicts the equality $x = \tilde{x}$. □

Now that we have explained how a unique set of parameters (sign, exponent, significand) can be assigned to any floating point number, we describe how these parameters are stored on the computer in practice. As their names suggest, the `Float16`, `Float32` and `Float64` formats use 16, 32 and 64 bits of memory, respectively. A naive approach for encoding these number formats would be to store the full binary representations of the sign, exponent and significand.

For the `Float32` format, this approach would require 1 bit for the sign, 8 bits to cover the 254 possible values of the exponent, and 24 bits for the significand, i.e. for storing $b_0, \ldots, b_{p-1}$. This leads to a total number of 33 bits, which is one more than is available, and this is without the special values `NaN`, `Inf` and `-Inf`. So how are numbers in the $\mathbf{F}_{32}$ format actually stored? To answer this question, we begin with two observations:

- If $E > E_{\min}$, then necessarily $b_0 = 1$ in the unique representation of the significand. Consequently, the leading bit need not be explicitly specified in this case; it is said to be *implicit*. We will see, as a consequence, that $p-1$ instead of p bits are in fact sufficient for the significand.

- In the $\mathbf{F}_{32}$ format, 8 bits at minimum need to be reserved for the exponent, which enables the representation of $2^8 = 256$ different values, but there are only 254 possible values for the exponent. This suggests that $256 - 254 = 2$ combinations of the 8 bits can be exploited in order to represent the special values `Inf`, `-Inf` and `NaN`.

Simplifying a little bit, we may view single precision floating point number as an array of 32 bits as illustrated below:



According to the IEEE 754 standard, the first bit is the sign s, the next 8 bits $e_0e_1 \ldots e_6e_7$ encode the exponent, and the last 23 bits $b_1b_2 \ldots b_{p-2}b_{p-1}$ encode the significand. Let us emphasize that when we say "encode the exponent" here, we just mean that the bits contain information from which the exponent can be uniquely determined, but we have not yet described how this is achieved. Let us introduce the integer number $e = (e_0e_1 \ldots e_6e_7)_2$; that is to say, $0 \leqslant e \leqslant 2^8 - 1$ is the integer number whose binary representation is given by $e_0e_1 \ldots e_6e_7$. One may determine the exponent and significand of a floating point number from the following rules.

- **Denormalized numbers**: If $e = 0$, then the implicit leading bit b_0 is zero, the fraction is $b_1b_2 \ldots b_{p-2}b_{p-1}$, and the exponent is $E = E_{\min}$. In other words, using the notation of Section 1.2, we have $x = (-1)^s 2^{E_{\min}} (0.b_1b_2 \ldots b_{p-2}b_{p-1})_2$. In particular, if $b_1b_2 \ldots b_{p-2}b_{p-1} = 00 \ldots 00$, then it holds that $x = 0$.

- **Non-denormalized numbers**: If $0 < e < 255$, then the implicit leading bit b_0 of the significand is 1 and the fraction is given by $b_1b_2 \ldots b_{p-2}b_{p-1}$. The exponent is given by

$$E = e - \text{bias} = E_{\min} + e - 1.$$

  where the exponent bias for the single and double precision formats are given in Table 1.2. In this case $x = (-1)^s 2^{e-\text{ bias}} (1.b_1b_2 \ldots b_{p-2}b_{p-1})_2$. Notice that $E = E_{\min}$ if $e = 1$, as in the case of denormalized numbers.

- **Infinities**: If $e = 255$ and $b_1b_2 \ldots b_{p-2}b_{p-1} = 00 \ldots 00$, then $x =$ `Inf` if $s = 0$ and `-Inf` otherwise.

- **Not a Number**: If $e = 255$ and $b_1 b_2 \dots b_{p-2} b_{p-1} \neq 00\dots00$, then $x =$ `NaN`. Notice that the special value `NaN` can be encoded in many different manners. These extra degrees of freedom were reserved for passing information on the reason for the occurrence of `NaN`, which is usually an indication that something has gone wrong in the calculation.

	Half precision	Single precision	Double precision
Exponent bias ($-E_{\min}+1$)	15	127	1023
Exponent encoding (bits)	5	8	11
Significand encoding (bits)	10	23	52

Table 1.2: Encoding parameters for floating point formats

Remark 1.3 (Encoding efficiency). With 32 bits, at most 2^{32} different numbers could in principle be represented. In practice, as we saw in Exercise 1.10, the `Float32` format enables to represent

$$(E_{\max} - E_{\min})2^p + 2^{p+1} - 1 = 253 \times 2^{24} + 2^{25} - 1 = 2^{32} - 2^{24} - 1 \approx 99.6\% \times 2^{32},$$

different real numbers, indicating a very good encoding efficiency.

Remark 1.4 (Nonuniqueness of the floating point representation of 0.0). The sign s is clearly unique for any number in a floating point format, except for 0.0, which could in principle be represented as

$$(-1)^0 2^{E_{\min}} (0.00\dots00)_2 \qquad \text{or} \qquad (-1)^1 2^{E_{\min}} (0.00\dots00)_2.$$

In practice, both representations of 0.0 are available on most machines, and these behave slightly differently. For example `1/(0.0) = Inf` but `1/(-0.0) = -Inf`.

1.5 Integer formats

The machine representation of integer formats is much simpler than that of floating point numbers. In this short section, we give a few orders of magnitude for common integer formats and briefly discuss overflow issues. Programming languages typically provide integer formats based on 16, 32 and 64 bits. In Julia, these correspond to the types `Int16`, `Int32` and `Int64`, the latter being the default for integer literals.

The most common encoding for integer numbers, which is used in Julia, is known as *two's complement*: a number encoded with p bits given by $b_{p-1} b_{p-2} \dots b_0$ corresponds to

$$x = -b_{p-1} 2^{p-1} + \sum_{i=0}^{p-2} b_i 2^i.$$

This encoding enables to represent uniquely all the integers from $N_{\min} = -2^{p-1}$ to $N_{\max} = 2^{p-1} - 1$. In contrast with floating point formats, integer formats do not provide special values like `Inf` and `NaN`. The number delivered by the machine when a calculation exceeds the

maximum representable value in the format, called the *overflow behavior*, generally depends on the programming language.

Since the overflow behavior of integer formats is not universal across programming languages, a detailed discussion is of little interest. We only mention that Julia uses a *wraparound* behavior, where $N_{\max} + 1$ silently returns $N_{\min}$ and, similarly, $N_{\min} - 1$ gives $N_{\max}$; the numbers loop back. This can lead to unexpected results, such as `2^64` evaluating to 0.

1.6 Exercises

Exercise 1.1. *Show that if a number $x \in \mathbf{R}$ admits a finite representation (1.1) in base β, then it also admits an infinite representation in the same base.* ***Hint:*** *You may have learned before that $(0.\overline{9})_{10} = 1$.*

Exercise 1.2. *How many digits does it take to represent all the integers from 0 to $10^{10} - 1$ in decimal and binary formats? What about the hexadecimal format?*

Exercise 1.3. *Find the decimal representation of $(0.000\overline{1100})_2$.*

Exercise 1.4. *Find the binary representation of $(0.1)_{10}$.*

Exercise 1.5. *Implement Algorithm 1 on a computer and verify that it works. Your function should take two arguments: an array of integers `[a_1, ..., a_n]` containing the digits after the decimal point and the number m of bits to return. The bits should be* returned *as an array `[b_1, ..., b_m]`.*

Exercise 1.6. *As mentioned above, Algorithm 1 works only for decimal numbers of the specific form $x = (0.a_1 \dots a_n)_{10}$. Find and implement a similar algorithm for integer numbers. More precisely, write a function that takes an integer n as argument and returns an array containing the bits of the binary expansion $(b_m \dots b_0)_2$ of n, from the least significant b_0 to the most significant b_m. That is to say, your code should return `[b_0, b_1, ...]`.*

```
function to_binary(n)
    # Your code comes here
end

# Check that it works
number = 123456789
bits = to_binary(number)
pows2 = 2 .^ range(0, length(bits) - 1)
@assert sum(bits'pows2) == number
```

Exercise 1.7. *Show that the successor of 1 in $\mathbf{F}_{64}$ is $1 + \varepsilon_{64}$, where ε_{64} is the machine epsilon for the double-precision format.*

Exercise 1.8. *Write down the values of the smallest and largest, in absolute value, positive real numbers representable in the $\mathbf{F}_{32}$ and $\mathbf{F}_{64}$ formats.*

Exercise 1.9 (Relative error and machine epsilon). *Prove that the inequality (1.3) is sharp. More precisely, find $x \in \mathbf{R}$ such that the inequality is an equality.*

Exercise 1.10 (Cardinality of the set of floating point numbers). *Show that, if $E_{\max} \geqslant E_{\min}$, then $\mathbf{F}(p, E_{\min}, E_{\max})$ contains exactly*

$$(E_{\max} - E_{\min})2^p + 2^{p+1} - 1$$

distinct real numbers. *(In particular, the special values `Inf`, `-Inf` and `NaN` are not counted.)* ***Hint:*** *Count first the numbers with $E > E_{\min}$ and then those with $E = E_{\min}$.*

Exercise 1.11. *Calculate the machine epsilon ε_{16} for the $\mathbf{F}_{16}$ format. Write the results of the arithmetic operations $1 \mathbin{\hat{+}} \varepsilon_{16}$ and $1 \mathbin{\hat{-}} \varepsilon_{16}$ in the form*

$$2^E(1.b_1 \dots b_{p-1})_2.$$

Exercise 1.12. *Let ε_{16} be the machine epsilon for the $\mathbf{F}_{16}$ format, and define $y = \frac{4}{3}\varepsilon_{16}$. What is the relative error between $\Delta = (1 + y) - 1$, and the machine approximation $\hat{\Delta} = (1 \mathbin{\hat{+}} y) \mathbin{\hat{-}} 1$?*

Exercise 1.13 (Numerical differentiation). *Let $f(x) = \exp(x)$. By definition, the derivative of f at 0 is given by*

$$f'(0) = \lim_{\delta \to 0} \left(\frac{f(\delta) - f(0)}{\delta} \right).$$

The expression within brackets on the right-hand side may be used with a small but nonzero δ as an approximation for $f'(0)$. Implement this approach using double-precision numbers and the same values for δ as in the table below. Explain the results you obtain.

δ	$\frac{\varepsilon_{64}}{4}$	$\frac{\varepsilon_{64}}{2}$	ε_{64}
Approximation of $f'(0)$	0	2	1

Exercise 1.14 (Avoiding overflow). *Write a code to calculate the weighted average*

$$S := \frac{\sum_{j=0}^{J} w_j j}{\sum_{j=0}^{J} w_j}, \qquad w_j = \exp(j), \qquad J = 1000.$$

You may need to first rewrite S differently.

Exercise 1.15. *Plot the function $x \mapsto \log(\mathrm{e}^{\mathrm{e}^x} - 1)$ over the interval $[0, 10]$.*

Exercise 1.16 (Calculating the sample variance). *Assume that $(x_n)_{1 \leqslant n \leqslant N}$, with $N = 10^6$, are independent random variables distributed according to the uniform distribution $\mathcal{U}(L, L + 1)$. That is, each x_n takes a random value uniformly distributed between L and $L + 1$ where $L = 10^9$. In Julia, these samples can be generated with the following lines of code:*

```
N, L = 10^6, 10^9
x = L .+ rand(N)
```

It is well known that the variance of $x_n \sim \mathcal{U}(L, L+1)$ *is given by* $\sigma^2 = \frac{1}{12}$. *Numerically, the variance can be estimated from the* sample variance*:*

$$s^2 = \frac{1}{N-1}\left(\left(\sum_{n=1}^{N} x_n^2\right) - N\bar{x}^2\right), \qquad \bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n. \tag{1.7}$$

Write a computer code to calculate s^2 *with the best possible accuracy. Can you find a formula that enables better accuracy than* (1.7)*?*

Remark 1.5. In order to estimate the true value of s^2 for your samples, you can use the `BigFloat` format, to which the array x can be converted by using the instruction

```
x = BigFloat.(x)
```

Exercise 1.17. *Euler proved that*

$$\frac{\pi^2}{6} = \lim_{N\to\infty} \sum_{n=1}^{N} \frac{1}{n^2}.$$

Using the default `Float64` *format, estimate the error obtained when the series on the right-hand side is truncated after* 10^{10} *terms. Can you rearrange the sum for best accuracy?*

Exercise 1.18. *Let* x *and* y *be positive real numbers in the interval* $[2^{-10}, 2^{10}]$ *(so that we do not need to worry about denormalized numbers, assuming we are working in single or double precision), and let us define the machine addition operator* $\hat{+}$ *for arguments in real numbers as*

$$\hat{+} : \mathbf{R} \times \mathbf{R} \to \mathbf{R}; (x, y) \mapsto \mathrm{fl}\Big(\mathrm{fl}(x) + \mathrm{fl}(y)\Big).$$

Prove the following bound on the relative error between the sum $x + y$ *and its machine approximation* $x \hat{+} y$*:*

$$\frac{\big|(x+y) - (x \hat{+} y)\big|}{|x+y|} \leqslant \frac{\varepsilon_M}{2}\Big(2 + \frac{\varepsilon_M}{2}\Big).$$

Hint: *decompose the numerator as*

$$(x+y) - (x \hat{+} y) = \Big(x - \mathrm{fl}(x)\Big) + \Big(y - \mathrm{fl}(y)\Big) + \Big(\mathrm{fl}(x) + \mathrm{fl}(y) - (x \hat{+} y)\Big),$$

and then use Proposition 1.1.

Exercise 1.19. *Is* `Float32(0.1) * Float32(10) == 1` *equal to* `true` *or* `false` *given the default rounding rule defined by the IEEE standard? Explain.*

Solution. By default, real numbers are rounded to the nearest floating point number. This can be checked in Julia with the command `rounding(Float32)`, which prints the default rounding mode. The exact binary representation of the real number $x = 0.1$ is

$$\begin{aligned} x &= (0.000\overline{1100})_2 \\ &= 2^{-4} \times (\underbrace{1.10011001100110011001100}_{\text{24 bits}}\overline{1100})_2 \end{aligned}$$

The first task is to determine the member of $\mathbf{F}_{32}$ that is nearest x. We have

$$\begin{aligned} x^- &:= \max\{y : y \in \mathbf{F}_{32} \text{ and } y \leqslant x\} = 2^{-4} \times (1.10011001100110011001100)_2 \\ x^+ &:= \min\{y : y \in \mathbf{F}_{32} \text{ and } y \geqslant x\} = 2^{-4} \times (1.10011001100110011001101)_2. \end{aligned}$$

Since the number $(0.\overline{1100})_2$ is closer to 1 than to 0, the number x is closer to x^+ than to x^-. Therefore, the number obtained when writing `Float32(0.1)` is x^+. To conclude the exercise, we need to calculate $\mathrm{fl}\big(10 \times x^+\big)$, and to this end we first write the exact binary representation of the real number $10 \times x^+ = (1010)_2 \times x^+$. We have

$$\begin{aligned} (1010)_2 \times x^+ = (1000)_2 \times x^+ + (10)_2 \times x^+ &= 2^{-4} \times \ (1100.11001100110011001101)_2 \\ &\quad +2^{-4} \times \quad (11.00110011001100110011001)_2 \\ &= 2^{-4} \times (\underbrace{10000.00000000000000000000}_{\text{24 bits}}01)_2. \end{aligned}$$

This can be checked in Julia by writing `bitstring(Float32(0.1) * Float64(10.0))`. Clearly, when rounding to the nearest $\mathbf{F}_{32}$ number, the number $2^{-4}(10000)_2 = 1$ is obtained; the boolean expression in the question is thus `true`. △

Remark 1.6. It should not be inferred from Exercise 1.19 that `Float32(1/i) * i` is always exact in floating point arithmetic. For example `Float32(1/41) * 41` does not evaluate to `1`, and neither do `Float16(1/11) * 11` and `Float64(1/49) * 49`.

Exercise 1.20. *Given that the default rounding rule specified by the IEEE 754 standard is "round to nearest, tie to even", does* `Float16(0.1) + Float16(0.2) == Float16(0.3)` *evaluate to* `true` *or* `false`*? Explain.*

Exercise 1.21. *Explain why* `Float32(sqrt(2))^2 - 2` *is not zero in Julia.*

Solution. The exact binary representation of $x := \sqrt{2}$, found using the `bitstring` in the `Float64` format, is given by

$$x = (\underbrace{1.01101010000010011110011}_{\text{24 bits}}001100\ldots)_2.$$

The first task is to determine the member of $\mathbf{F}_{32}$ that is nearest x. We have

$$\begin{aligned} x^- &:= \max\{x : x \in \mathbf{F}_{32} \text{ and } x \leqslant \sqrt{2}\} = (\underbrace{1.01101010000010011110011}_{\text{24 bits}})_2 \\ x^+ &:= \min\{x : x \in \mathbf{F}_{32} \text{ and } x \geqslant \sqrt{2}\} = (\underbrace{1.01101010000010011110100}_{\text{24 bits}})_2, \end{aligned}$$

and we calculate

$$x - x^- = 2^{-24}(0.01100\ldots)_2,$$
$$x^+ - x = 2^{-21}\Big(1 - (0.11001100\ldots)_2\Big) \geqslant 2^{-21}\Big(1 - (0.11001101)_2\Big) = 2^{-21}(0.00110011)_2.$$

We deduce that $x - x^- < x^+ - x$, and so $\mathrm{fl}(x) = x^-$. To conclude the exercise, we need to show that $\mathrm{fl}\Big((x^-)^2\Big)$ is not equal to 2. The exact binary expansion of $(x^-)^2$ is

$$(x^-)^2 = (\underbrace{1.11111111111111111111111}_{\text{24 bits}}01101100111111010101001)_2.$$

The member of $\mathbf{F}_{32}$ nearest this number is

$$(1.11111111111111111111111)_2 = 2 - 2^{-23},$$

which is precisely the result returned by Julia. △

Exercise 1.22 (Numerical differentiation)**.** *Let $f(x) = \exp(x)$ and let $d(\delta)$ be the approximation of $f'(x)$ obtained from the following piece of code:*

```
f, x = exp, 1
d(δ) = (f(x+δ) - f(x))/δ
```

Plot for fixed $x = 1$ the error `abs(d(δ) - exp(x))` *as a function of* `δ` *in logarithmic scale, and explain the result.*

Solution. We assume that $\delta \in \mathbf{F}_{64}$ for simplicity, and we let $\varepsilon = \varepsilon_M/2$. This is not a restrictive assumption as δ can only take floating point values in computer programs. The proof is rather technical, and so it is given for information purposes only. We rewrite $d(\delta)$ mathematically as

$$d(\delta) = \Big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\Big) \mathbin{\hat{/}} \delta,$$

where $\hat{f}(x) = \mathrm{fl}\Big(f(x)\Big)$. We wish to bound $|f'(x) - d(\delta)|$. By the triangle inequality,

$$|f'(x) - d(\delta)| \leqslant \left|f'(x) - \frac{f(x+\delta) - f(x)}{\delta}\right| + \left|\frac{f(x+\delta) - f(x)}{\delta} - \Big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\Big) \mathbin{\hat{/}} \delta\right|. \quad (1.8)$$

Bounding the first term in (1.8). By Taylor's theorem, there exists $\xi \in [x, x+\delta]$ such that

$$f(x+\delta) = f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(\xi).$$

Therefore, the first term in (1.8) satisfies

$$\left|f'(x) - \frac{f(x+\delta) - f(x)}{\delta}\right| = \frac{\delta}{2}\,|f''(\xi)| = \frac{\delta}{2}\,|f''(x)| + \mathcal{O}(\delta^2).$$

Bounding the second term in (1.8) – the roundoff error. This is more tedious but not difficult; the main ingredient is the triangle inequality. Specifically, we will use the bound

$$\begin{aligned}\left|\frac{f(x+\delta)-f(x)}{\delta} - \left(\hat{f}(x \hat{+} \delta) \hat{-} \hat{f}(x)\right) \hat{/} \delta\right| \\ \leqslant \delta^{-1}\left|f(x+\delta) - f(x \hat{+} \delta)\right| + \delta^{-1}\left|f(x \hat{+} \delta) - \hat{f}(x \hat{+} \delta)\right| + \delta^{-1}\left|\hat{f}(x) - f(x)\right| \\ +\delta^{-1}\left|\left(\hat{f}(x \hat{+} \delta) - \hat{f}(x)\right) - \left(\hat{f}(x \hat{+} \delta) \hat{-} \hat{f}(x)\right)\right| \\ +\left|\left(\hat{f}(x \hat{+} \delta) \hat{-} \hat{f}(x)\right)/\delta - \left(\hat{f}(x \hat{+} \delta) \hat{-} \hat{f}(x)\right) \hat{/} \delta\right|. \end{aligned} \tag{1.9}$$

Note that without absolute values, both sides are indeed equal. The first three terms on the right-hand side are together a bound from above for

$$\left|\frac{f(x+\delta)-f(x)}{\delta} - \frac{\hat{f}(x \hat{+} \delta) - \hat{f}(x)}{\delta}\right|, \tag{1.10}$$

while the two other terms account for the roundoff errors associated with the machine subtraction and division operators, respectively. We will show that the dominant terms in (1.9) are the first three; the latter two are negligible in comparison. Employing Proposition 1.1 with $x = f(a)$, with $x = a \pm b$ and with $x = a/b$, we deduce that the following inequalities are satisfied:

$$\left|\hat{f}(a) - f(a)\right| \leqslant \varepsilon\ |f(a)|, \qquad |(a \pm b) - (a \hat{\pm} b)| \leqslant \varepsilon |a \pm b|, \qquad |a/b - a \hat{/} b| \leqslant \varepsilon\ |a/b|. \tag{1.11}$$

The first two inequalities are valid for all $(a, b) \in \mathbf{F}_{64} \times \mathbf{F}_{64}$, while the third inequality is valid for all $(a, b) \in \mathbf{F}_{64} \times \mathbf{F}_{64} \backslash \{0\}$. The first inequality in (1.11) can be employed in order to bound the second and third term on the right-hand side of (1.9):

$$\begin{aligned}\delta^{-1}\left|f(x \hat{+} \delta) - \hat{f}(x \hat{+} \delta)\right| + \delta^{-1}\left|\hat{f}(x) - f(x)\right| &\leqslant \delta^{-1}\varepsilon\Big(\left|f(x+\delta)\right| + \left|f(x)\right|\Big) \\ &= 2\delta^{-1}\varepsilon|f(x)| + \mathcal{O}(\varepsilon). \end{aligned} \tag{1.12}$$

Using Taylor's theorem and then the second inequality in (1.11) (with $+$), we then bound the first term on right-hand side of (1.9) as

$$\begin{aligned}\delta^{-1}\left|f(x+\delta) - f(x \hat{+} \delta)\right| &= \delta^{-1}\left|f'(\xi)\Big((x+\delta) - (x \hat{+} \delta)\Big)\right| \\ &\leqslant \delta^{-1}\varepsilon\ |f'(\xi)|\ |x+\delta| = \delta^{-1}\varepsilon\ |f'(x)x| + \mathcal{O}(\varepsilon). \end{aligned} \tag{1.13}$$

Combining (1.12) and (1.13), we obtain that the expression in (1.10) scales as $\delta^{-1}\varepsilon$:

$$\left|\frac{f(x+\delta)-f(x)}{\delta} - \frac{\hat{f}(x \hat{+} \delta) - \hat{f}(x)}{\delta}\right| = \mathcal{O}(\delta^{-1}\varepsilon). \tag{1.14}$$

Next, using the second inequality in (1.11) (with $-$) and then (1.14), we bound the fourth term in (1.9) as follows:

$$\begin{aligned}\delta^{-1}\left|\left(\hat{f}(x \hat{+} \delta) - \hat{f}(x)\right) - \left(\hat{f}(x \hat{+} \delta) \hat{-} \hat{f}(x)\right)\right| \\ \leqslant \delta^{-1}\varepsilon\ |\hat{f}(x \hat{+} \delta) - \hat{f}(x)| = \delta^{-1}\varepsilon\underbrace{|f(x+\delta) - f(x)|}_{=\mathcal{O}(\delta)} + \mathcal{O}(\delta^{-1}\varepsilon^2) = \mathcal{O}\ (\varepsilon + \delta^{-1}\varepsilon^2). \end{aligned} \tag{1.15}$$

Notice that this term is smaller than the first three in (1.9) when $\delta \ll 1$. We deduce by a triangle inequality from (1.14) and (1.15) that

$$\delta^{-1}\Big|\big(f(x+\delta)-f(x)\big)-\big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\big)\Big| = \mathcal{O}(\delta^{-1}\varepsilon).$$

Finally, using the third inequality in (1.11) together with this equation, we bound the fifth term on the right-hand side of (1.9):

$$\begin{aligned}&\Big|\big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\big)/\delta - \big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\big) \mathbin{\hat{/}} \delta\Big| \\ &\quad \leqslant \varepsilon\Big|\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\Big|/\delta = \frac{\varepsilon}{\delta}\underbrace{\big|f(x+\delta)-f(x)\big|}_{\mathcal{O}(\delta)} + \mathcal{O}(\delta^{-1}\varepsilon^2) = \mathcal{O}(\varepsilon+\delta^{-1}\varepsilon^2).\end{aligned}$$

This term is also negligible in front of the other dominant contributions to the roundoff error given in (1.12) and (1.13). Going back to (1.9), we conclude that

$$\left|\frac{f(x+\delta)-f(x)}{\delta} - \big(\hat{f}(x \mathbin{\hat{+}} \delta) \mathbin{\hat{-}} \hat{f}(x)\big) \mathbin{\hat{/}} \delta\right| \leqslant \delta^{-1}\varepsilon\Big(2\,|f(x)| + |f'(x)x|\Big) + \mathcal{O}(\varepsilon+\delta^{-1}\varepsilon^2).$$

Concluding the proof. Going back to (1.8), we conclude that

$$|f'(x) - d(\delta)| \leqslant \frac{\delta}{2}\,|f''(x)| + \frac{\varepsilon}{\delta}\Big(2\,|f(x)| + |f'(x)x|\Big),$$

up to higher order terms. For fixed x, the right-hand side is minimized when

$$\delta = \sqrt{\varepsilon}\sqrt{\frac{2\,|f(x)| + |f'(x)x|}{|f''(x)|}},$$

which is a well-known formula for the optimal step size in numerical differentiation. The error as a function of δ for $x = 1$ is depicted in Figure 1.4.

△

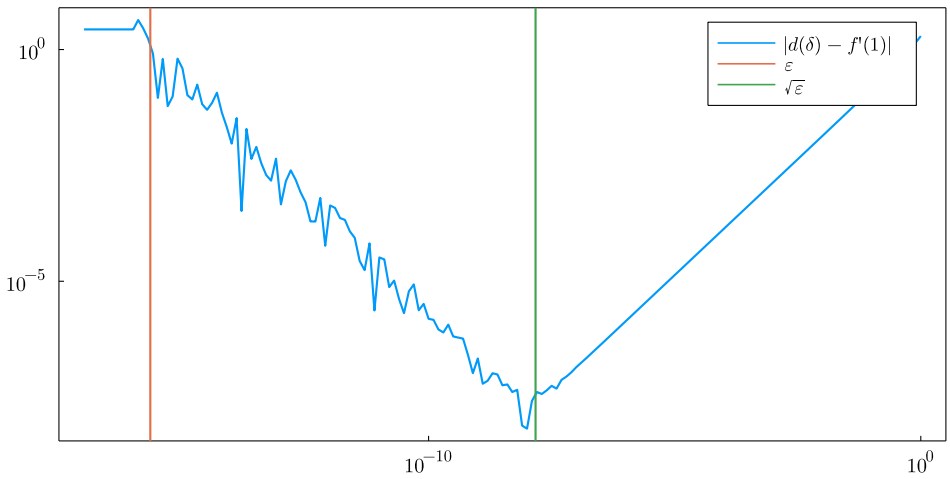


Figure 1.4: Solution to Exercise 1.22.

Exercise 1.23. *Explain why* `exp(log(Float16(7))) == 7` *is* `false`.

Solution. We begin by finding the binary representation of $\log(7)$. In the $\mathbf{F}_{64}$ format, the sign and exponent are encoded over 1 and 11 bits respectively, and so the fraction is given by the last 52 bits, which can be obtained from the command

```
julia> bitstring(log(7))[13:end]
"1111001000100111001010101110001100100101101001010111"
```

Therefore, since `exponent(log(7))` returns 0, we have

$$\log(7) = (\underbrace{1.11110010000}_{11 \text{ bits}}1001110010101011100011001001011010010101 1...)_2 \tag{1.16}$$

The number returned by the command `log(Float16(7))` is given by $\mathrm{fl}_{16}\big(\log(7)\big)$, where fl_{16} denotes the half-precision rounding operator. Rounding the right-hand side of (1.16) to 11 bits, we obtain

$$\mathrm{fl}_{16}\big(\log(7)\big) = (1.1111001001)_2 = 1.9462890625.$$

The number returned by the code `exp(log(Float16(7)))` is

$$\mathrm{fl}_{16}\Big(\exp\big(\mathrm{fl}_{16}\big(\log(7)\big)\big)\Big) = \mathrm{fl}_{16}\big(\exp(1.9462890625)\big).$$

The rounding operator appears twice on the left-hand side, because the computer rounds *after every operation*. To explain the result of the rounding operation, we begin by calculating the binary expansion of $\exp(1.9462890625)$.

```
julia> bitstring(exp(1.9462890625))[13:end]
"1100000000101011011101110000111000100001100010001110"

julia> exponent(exp(1.9462890625))
2
```

Therefore,

$$\exp(1.9462890625) = 2^2(\underbrace{1.11000000001}_{11 \text{ bits}}0101101110111000011100010000110001000 11...)_2,$$

and, rounding to 11 bits, we finally obtain

$$\begin{aligned}\mathrm{fl}_{16}\big(\exp(1.9462890625)\big) &= 2^2(1.11000000011)_2 = 4\Big(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{2^{10}}\Big)\\ &= 7+\frac{1}{2^8} = 7.00390625,\end{aligned}$$

which is different from 7. △

Exercise 1.24. *Determine the encoding of the following* `Float32` *numbers:*

- $x_1 = 2.0^{E_{\min}}$
- $x_2 = -2.0^{E_{\min}-p+1} = -2.0^{-149}$
- $x_3 = 2.0^{E_{\max}}(2 - 2^{-p+1})$

Check your results using the Julia function `bitstring`.

Exercise 1.25 (Summary). *True or false?*

1. *Let* $(\bullet)_2$ *denote binary representation. It holds that* $(0.1111)_2 + (0.0001)_2 = 1$.
2. *It holds that* $(1000)_2 \times (0.001)_2 = 1$.
3. *It holds that*
$$(0.\overline{1})_3 = \frac{1}{2}.$$
4. *In base 16, all the natural numbers from 1 to 200 can be represented using 2 digits.*
5. *In Julia,* `Float64(.1) == Float32(.1)` *evaluates to* `true`.
6. *The spacing (in absolute value) between successive double-precision (*`Float64`*) floating point numbers is constant.*
7. *It holds that* $(0.\overline{10101})_2 = (1.2345)_{10}$.
8. *Machine addition* $\hat{+}$ *is an associative operation. More precisely, given any three double-precision floating point numbers* x, y *and* z, *the following equality holds:*
$$(x \hat{+} y) \hat{+} z = x \hat{+} (y \hat{+} z).$$
9. *The machine epsilon is the smallest strictly positive number that can be represented in a floating point format.*
10. *Let* ε *denote the machine epsilon for the double-precision format. Let also* $\hat{+}$ *and* $\hat{/}$ *denote respectively the machine addition and the machine division operators for the double-precision format. It holds that* $1 \hat{+} (\varepsilon \hat{/} 64) = 1$ *and that* $\varepsilon \hat{/} 64 \neq 0$.
11. *Assume that* $x \in \mathbf{R}$ *belongs to the double-precision floating point format (that is, assume that* $x \in \mathbf{F}_{64}$*). Then* $-x \in \mathbf{F}_{64}$.

1.7 Discussion and bibliography

This chapter is mostly based on the original 1985 IEEE 754 standard [4] and the reference book [10]. A significant revision to the 1985 IEEE standard was published in 2008 [5], adding for example specifications for the half precision and quad precision formats, and a minor revision was published in 2019 [6]. The original IEEE standard and its revisions constitute the authoritative guide on floating point formats. It was intended to be widely disseminated and is

written very clearly and concisely, but is not available for free online. Another excellent source for learning about floating point numbers and round-off errors is D. Goldberg's paper "*What every computer scientist should know about floating-point arithmetic*" [3], freely available online.

Chapter 2

Interpolation and approximation

2.1 Interpolation 27

2.1.1 Vandermonde matrix 28

2.1.2 Lagrange interpolation formula 28

2.1.3 Gregory–Newton interpolation 29

2.1.4 Interpolation error 34

2.1.5 Interpolation at Chebyshev nodes 36

2.1.6 Hermite interpolation 39

2.1.7 Piecewise interpolation 40

2.2 Approximation 41

2.2.1 Least squares approximation of data points 42

2.2.2 Mean square approximation of functions 43

2.2.3 Orthogonal polynomials 45

2.2.4 Orthogonal polynomials and numerical integration: an introduction . . . 47

2.3 Exercises 49

2.4 Discussion and bibliography 55

Introduction

In this chapter, we study numerical methods for interpolating and approximating functions. The Cambridge dictionary defines interpolation as *the addition of something different in the middle of a text, piece of music, etc. or the thing that is added*. The concept of interpolation in mathematics is consistent with this definition; interpolation consists in finding, given a set of points (x_i, y_i), a function f in a finite-dimensional space that goes through these points. Throughout this course, you use the `plot` function in Julia, which performs piecewise linear interpolation for drawing functions, but there are a number of other standard interpolation methods. Our first goal in this chapter is to present an overview of these methods and the associated error estimates.

In the second part of this chapter, we focus on *function approximation*, which is closely related to the subject of mathematical interpolation. Indeed, a simple manner for approximating a general function by another one in a finite-dimensional space is to select a set of real numbers on the x axis, called *nodes*, and find the associated interpolant. As we shall demonstrate, not all sets of interpolation nodes are equal, and special care is required in order to avoid undesired oscillations. The field of function approximation is vast, so our aim in this chapter is to present only an introduction to the subject. In order to quantify the quality of an approximation, a metric on the space of functions, or a subset thereof, must be specified in order to measure errors. Without a metric, saying that two functions are close is almost meaningless!

2.1 Interpolation

Assume that we are given $n+1$ nodes $x_0, \ldots, x_n$ on the x axis, together with values $u_0, \ldots, u_n$, which may be the values taken by an unknown function $u(x)$ when evaluated at these points. Suppose that we are looking for an interpolation $\widehat{u}(x)$ in a subspace $\mathrm{Span}\{\varphi_0, \ldots, \varphi_n\}$ of the vector space of continuous functions, i.e. an interpolating function of the form

$$\widehat{u}(x) = \alpha_0 \varphi_0(x) + \cdots + \alpha_n \varphi_n(x),$$

where $\alpha_0, \ldots, \alpha_n$ are real coefficients. In order for $\widehat{u}(x)$ to be an interpolating function, we must require that

$$\forall i \in \{0, \ldots, n\}, \qquad \widehat{u}(x_i) = u_i.$$

This leads to a linear system of $n+1$ equations and $n+1$ unknowns, the latter being the coefficients $\alpha_0, \ldots, \alpha_n$. This system of equations in matrix form reads

$$\begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_n(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.1}$$

2.1.1 Vandermonde matrix

Since polynomials are very convenient for evaluation, integration, and differentiation, they are a natural choice for interpolation purposes. The simplest basis of the subspace of polynomials of degree less than or equal to n is given by the monomials:

$$\varphi_0(x) = 1, \qquad \varphi_1(x) = x, \qquad \ldots, \qquad \varphi_n(x) = x^n.$$

In this case, the linear system (2.1) for determining the coefficients of the interpolant reads

$$\begin{pmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.2}$$

The matrix on the left-hand side is called a *Vandermonde* matrix. If the abcissae $x_0, \ldots, x_n$ are distinct, then this is a full rank matrix, and so (2.2) admits a unique solution, implying as a corollary that the interpolating polynomial exists and is unique. It is possible to show that the condition number of the Vandermonde increases dramatically with n. Consequently, solving (2.2) is not a viable method in practice for calculating the interpolating polynomial.

2.1.2 Lagrange interpolation formula

One may wonder whether polynomial basis functions $\varphi_0, \ldots, \varphi_n$ can be defined in such a manner that the matrix in (2.1) is the identity matrix. The answer to this question is positive; it suffices to take as a basis the *Lagrange polynomials*, which are given by

$$\varphi_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}.$$

It is simple to check that

$$\varphi_i(x_j) = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Finding the interpolant in this basis is immediate:

$$\widehat{u}(x) = u_0\varphi_0(x) + \cdots + u_n\varphi_n(x).$$

While simple, this approach to polynomial interpolation has a few disadvantages:

- First, evaluating $\widehat{u}(x)$ is computationally costly when n is large.
- Second, all the basis functions change when adding new interpolation nodes.
- Finally, Lagrange interpolation is numerically unstable because of cancellations between large terms. Indeed, it is often the case that Lagrange polynomials take very large

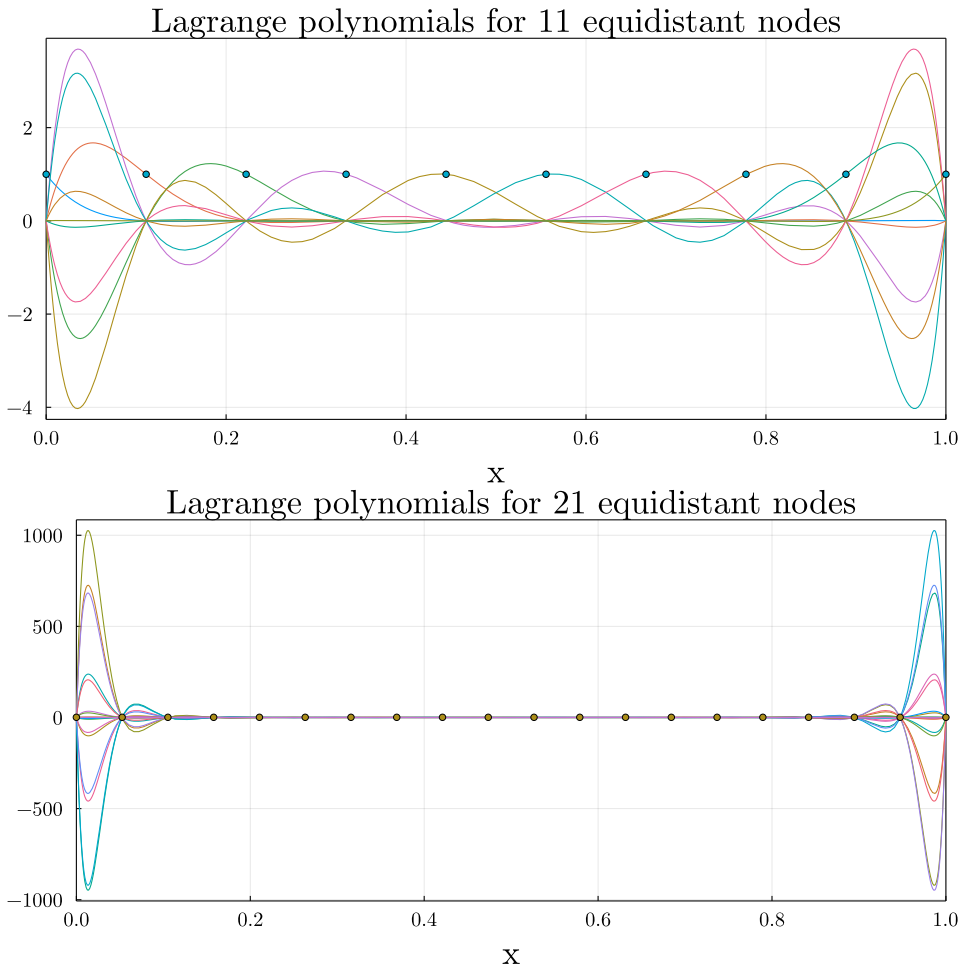


Figure 2.1: Lagrange polynomials associated with equidistant nodes over the $(0, 1)$ interval.

values over the interpolation intervals; this occurs, for example, when many equidistant interpolation nodes are employed, as illustrated in Figure 2.1.

2.1.3 Gregory–Newton interpolation

By Taylor's formula, any polynomial p of degree n may be expressed as

$$p(x) = p(0) + p'(0)x + \frac{p''(0)}{2}x^2 + \ldots + \frac{p^{(n)}(0)}{n!}x^n. \tag{2.3}$$

The constant coefficient can be obtained by evaluating the polynomial at 0, the linear coefficient can be identified by evaluating the first derivative at 0, and so on. Assume now that we are given the values taken by p when evaluated at the integer numbers $\{0, \ldots, n\}$. We ask the following question: can we find a formula similar in spirit to (2.3), but including only evaluations of p and not of its derivatives? To answer this question, we introduce the difference operator Δ which acts on functions as follows:

$$\Delta f(x) = f(x + 1) - f(x).$$

The operator Δ is a linear operator on the space of continuous functions. It maps constant functions to 0, and the linear function x to the constant function 1, suggesting a resemblance with the differentiation operator. In order to further understand this connection, let us define the *falling power* of a real number x as

$$x^{\underline{k}} = x(x-1)(x-2)\dots(x-k+1). \tag{2.4}$$

We then have that

$$\begin{aligned}\Delta x^{\underline{k}} &= (x+1)x(x-1)\dots(x-k+2) - x(x-1)(x-2)\dots(x-k+1) \\ &= \Big((x+1)-(x-k+1)\Big)\Big(x(x-1)\dots(x-k+2)\Big) = kx^{\underline{k-1}}\end{aligned} \tag{2.5}$$

In other words, the action of the difference operator on falling powers mirrors that of the differentiation operator on monomials. The falling powers form a basis of the space of polynomials, and so any polynomial in $\mathbf{P}(n)$, i.e. of degree less than or equal to n, can be expressed as

$$p(x) = \alpha_0 + \alpha_1 x^{\underline{1}} + \alpha_2 x^{\underline{2}} + \dots + \alpha_n x^{\underline{n}}. \tag{2.6}$$

It is immediate to show that $\alpha_i = \Delta^i p(0)/i!$, where $\Delta^i p$ denotes the function obtained after i applications of the operator Δ. Therefore, any polynomial of degree less than or equal to n may be expressed as

$$p(x) = p(0) + \Delta p(0) x^{\underline{1}} + \frac{\Delta^2 p(0)}{2} x^{\underline{2}} + \dots + \frac{\Delta^n p(0)}{n!} x^{\underline{n}}. \tag{2.7}$$

An expansion of the form (2.7) is called a *Newton series*, which is the discrete analog of the continuous Taylor series. From the definition of Δ, it is clear that the coefficients in (2.7) depend only on $p(0), \dots, p(n)$. We conclude that, given points $n+1$ points (i, u_i) for $i \in \{0, \dots, n\}$, the unique interpolating polynomial is given by (2.7), after replacing $p(i)$ by u_i.

Example 2.1. Let us use (2.6) in order to calculate the value of

$$S(n) := \sum_{i=0}^{n} i^2.$$

Since $\Delta S(n) = (n+1)^2$, which is a second degree polynomial in n, we deduce that $S(n)$ is a polynomial of degree 3. Let us now determine its coefficients.

n	0	1	2	3
$\Delta^0 S(n)$	**0**	1	5	14
$\Delta^1 S(n)$	**1**	4	9	
$\Delta^2 S(n)$	**3**	5		
$\Delta^3 S(n)$	**2**			

We conclude that

$$S(n) = \mathbf{1}n + \frac{\mathbf{3}}{2!}n(n-1) + \frac{\mathbf{2}}{3!}n(n-1)(n-2) = \frac{n(2n+1)(n+1)}{6}$$

Notice that when falling powers are employed as polynomial basis, the matrix in (2.1) is lower triangular, and so the algorithm described in Example 2.1 could be replaced by the forward substitution method. Whereas the coefficients of the Lagrange interpolant can be obtained immediately from the values of u at the nodes, calculating the coefficients of the expansion in (2.6) requires $\mathcal{O}(n^2)$ operations. However, Gregory–Newton interpolation has several advantages over Lagrange interpolation:

- If a point $(n+1, p_{n+1})$ is added to the set of interpolation points, only one additional term, corresponding to the falling power $x^{\underline{n+1}}$, needs to be calculated in (2.7). All the other coefficients are unchanged. Therefore, the Gregory–Newton approach is well-suited for incremental interpolation.

- The Gregory–Newton interpolation method is more numerically stable than Lagrange interpolation, because the basis functions do not take very large values.

- A polynomial in the form of a Newton series can be evaluated efficiently using Horner's method, which is based on rewriting the polynomial as

$$p(x) = \alpha_0 + x\Big(\alpha_1 + (x-1)\Big(\alpha_2 + (x-2)\Big(\alpha_3 + (x-3)\dots\Big)\Big)\Big).$$

  Evaluating this expression starting from the innermost bracket leads to an algorithm with a cost scaling linearly with the degree of the polynomial.

Non-equidistant nodes

So far, we have described the Gregory–Newton method in the simple setting where interpolation nodes are just a sequence of successive natural numbers. The method can be generalized to the setting of nodes $x_0 \neq \dots \neq x_n$ which are not necessarily equidistant. In this case, we take as basis the following functions instead of the falling powers:

$$\varphi_i(x) = (x - x_0)(x - x_1)\dots(x - x_{i-1}), \tag{2.8}$$

with the convention that the empty product is 1. By (2.1), the coefficients of the interpolating polynomial in this basis solve the following linear system:

$$\begin{pmatrix} 1 & & \dots & & 0 \\ 1 & x_1 - x_0 & & & \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & \vdots \\ \vdots & \vdots & & \ddots & \\ 1 & x_n - x_0 & \dots & \dots & \prod_{j=0}^{n-1}(x_n - x_j) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.9}$$

This system could be solved using, for example, forward substitution. Clearly $\alpha_0 = u_0$ from the first equation, and then from the second equation we obtain

$$\alpha_1 = \frac{u_1 - u_0}{x_1 - x_0} =: [u_0, u_1],$$

which may be viewed as an approximation of the slope of u at x_0. The right-hand side of this equation is an example of a *divided difference*. In general, divided differences are defined recursively as follows:

$$[u_0, u_1, \ldots, u_d] := \frac{[u_1, \ldots, u_d] - [u_0, \ldots, u_{d-1}]}{x_d - x_0}, \qquad [u_i] = u_i. \tag{2.10}$$

Let us start by observing that our divided differences coincide with the coefficients in (2.6).

Lemma 2.1. *Assume that* $(0, u_0), \ldots, (n, u_n) \in \mathbf{R}^2$. *Then, provided that* $(i, k) \in \mathbf{N}^2$ *satisfies* $i + k \leqslant n$, *we have*

$$\frac{1}{k!}\Delta^k u(i) = [u_i, \ldots, u_{i+k}].$$

Proof. The result is clearly true for $k = 0$, and we shall assume by induction that it holds for $k - 1$. Using our induction hypothesis, we get

$$\begin{aligned}\frac{1}{k!}\Delta^k u(i) &= \frac{1}{k}\left(\frac{\Delta^{k-1}u(i+1)}{(k-1)!} - \frac{\Delta^{k-1}u(i)}{(k-1)!}\right) \\ &= \frac{1}{k}\Big([u_{i+1}, \ldots, u_{i+k}] - [u_i, \ldots, u_{i+k-1}]\Big).\end{aligned}$$

By the definition of divided differences in (2.10), it follows that we have

$$\frac{1}{k!}\Delta^k u(i) = \frac{1}{k}(\underbrace{x_{i+k} - x_i}_{i+k-i})[u_i, \ldots, u_{i+k}] = [u_i, \ldots, u_{i+k}],$$

which allows us to conclude the proof. □

In light of Lemma 2.1, the reader will not be surprised that the coefficients in the $\varphi_0, \ldots, \varphi_n$ basis in (2.8) are given by the divided differences.

Proposition 2.2. *Assume that* $(x_0, u_0), \ldots, (x_n, u_n)$ *are* $n + 1$ *points in the plane with distinct abcissae. Then the interpolating polynomial of degree* n *may be expressed as*

$$p(x) = \sum_{i=0}^{n} [u_0, \ldots, u_i]\varphi_i(x),$$

where $\varphi_i(x)$, *for* $i = 0, \ldots, n$, *are the basis functions defined in* (2.8).

Proof. The statement is true for $n = 0$. Reasoning by induction, we assume that it holds true for polynomials of degree up to $n - 1$. Let $p_1(x)$ and $p_2(x)$ be the interpolating polynomials at the points $x_0, x_1, \ldots, x_{n-2}, x_{n-1}$ and $x_0, x_1, \ldots, x_{n-2}, x_n$, respectively. Then

$$p(x) = p_1(x) + \frac{x - x_{n-1}}{x_n - x_{n-1}}\big(p_2(x) - p_1(x)\big) \tag{2.11}$$

is a polynomial of degree n that interpolates all the data points. By the induction hypothesis, it holds that

$$p_1(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, u_1, \ldots, u_{n-2}, \mathbf{u_{n-1}}] \prod_{i=0}^{n-2}(x - x_i),$$
$$p_2(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, u_1, \ldots, u_{n-2}, \mathbf{u_n}] \prod_{i=0}^{n-2}(x - x_i).$$

Here we used a bold font in order to emphasize the difference between the two expressions. Substituting these expressions in (2.11), we obtain

$$p(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, \ldots, u_{n-2}] \prod_{i=0}^{n-2}(x - x_i)$$
$$+ \frac{[u_0, u_1, \ldots, u_{n-2}, u_n] - [u_0, u_1, \ldots, u_{n-2}, u_{n-1}]}{x_n - x_{n-1}} \prod_{i=0}^{n-1}(x - x_i).$$

In Exercise 2.4, we show that divided differences are invariant under permutations of the data points, and so we have that

$$\frac{[u_0, u_1, \ldots, u_{n-2}, u_n] - [u_0, u_1, \ldots, u_{n-2}, u_{n-1}]}{x_n - x_{n-1}} = [u_0, \ldots, u_n],$$

which enables to conclude. □

Example 2.2. Assume that we are looking for the third degree polynomial going through the following points:

$$(-1, 10), \qquad (0, 4), \qquad (2, -2), \qquad (4, -40).$$

We have to calculate the divided difference $\alpha_i = [u_0, \ldots, u_i]$ for $i \in \{0, 1, 2, 3\}$. To this end, it is convenient to use a table:

i	0	1	2	3
$[u_i]$	**10**	4	-2	-40
$x_{i+1} - x_i$	1	2	2	
$[u_i, u_{i+1}]$	**-6**	-3	-19	
$x_{i+2} - x_i$	3	4		
$[u_i, u_{i+1}, u_{i+2}]$	**1**	-4		
$x_{i+3} - x_i$	5			
$[u_i, u_{i+1}, u_{i+2}, u_{i+3}]$	**-1**			

We deduce that the expression of the interpolating polynomial is

$$p(x) = \mathbf{10} + (\mathbf{-6})(x + 1) + \mathbf{1}(x + 1)x + (\mathbf{-1})(x + 1)x(x - 2) = -x^3 + 2x^2 + -3x + 4.$$

2.1.4 Interpolation error

Assume that $u(x)$ is a continuous function and denote by $\hat{u}(x)$ its interpolation through the points (x_i, u_i), where $u_i = u(x_i)$ for $i = 0, \dots, n$. In this section, we study the behavior of the error in the limit as $n \to \infty$.

Theorem 2.3 (Interpolation error)**.** *Assume that u: $[a, b] \to \mathbf{R}$ is a function in $C^{n+1}([a, b])$ and let $x_0, \dots, x_n$ denote $n + 1$ distinct interpolation nodes. Then for all $x \in [a, b]$, there exists $\xi = \xi(x)$ in the interval $[a, b]$ such that*

$$e_n(x) := u(x) - \hat{u}(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0) \dots (x - x_n).$$

Proof. The statement is obvious if $x \in \{x_0, \dots, x_n\}$, so we assume from now on that x does not coincide with an interpolation node. Let us use the compact notation $\omega_n = \prod_{i=0}^{n}(x - x_i)$ and introduce the function

$$g(t) = e_n(t)\omega_n(x) - e_n(x)\omega_n(t). \tag{2.12}$$

The function g is smooth and takes the value 0 when evaluated at $x_0, \dots, x_n, x$. Since g has $n + 2$ roots in the interval $[a, b]$, Rolle's theorem implies that g' has at least $n + 1$ distinct roots in (a, b). Another application of Rolle's theorem then yields that g'' has at least n distinct roots in (a, b). Iterating this reasoning, we deduce that $g^{(n+1)}$ has one root t_* in (a, b). From (2.12), we calculate that

$$g^{(n+1)}(t) = e_n^{(n+1)}(t)\omega_n(x) - e_n(x)\omega_n^{(n+1)}(t) = u^{(n+1)}(t)\omega_n(x) - e_n(x)(n+1)!. \tag{2.13}$$

Here we employed the fact that $\hat{u}^{(n+1)}(t) = 0$, because $\hat{u}$ is a polynomial of degree at most n. Evaluating (2.13) at t_* and rearranging, we obtain that

$$e_n(x) = \frac{u^{(n+1)}(t_*)}{(n+1)!}\omega_n(x),$$

which completes the proof. □

As a corollary to Theorem 2.3, we deduce the following error bound.

Corollary 2.4 (Upper bound on the interpolation error)**.** *Assume that u is smooth in the interval $[a, b]$ and let*

$$C_{n+1} = \sup_{x \in [a,b]} \left|u^{(n+1)}(x)\right|.$$

Then

$$E_n := \sup_{x \in [a,b]} \left|e_n(x)\right| \leqslant \frac{C_{n+1}}{4(n+1)} h^{n+1} \tag{2.14}$$

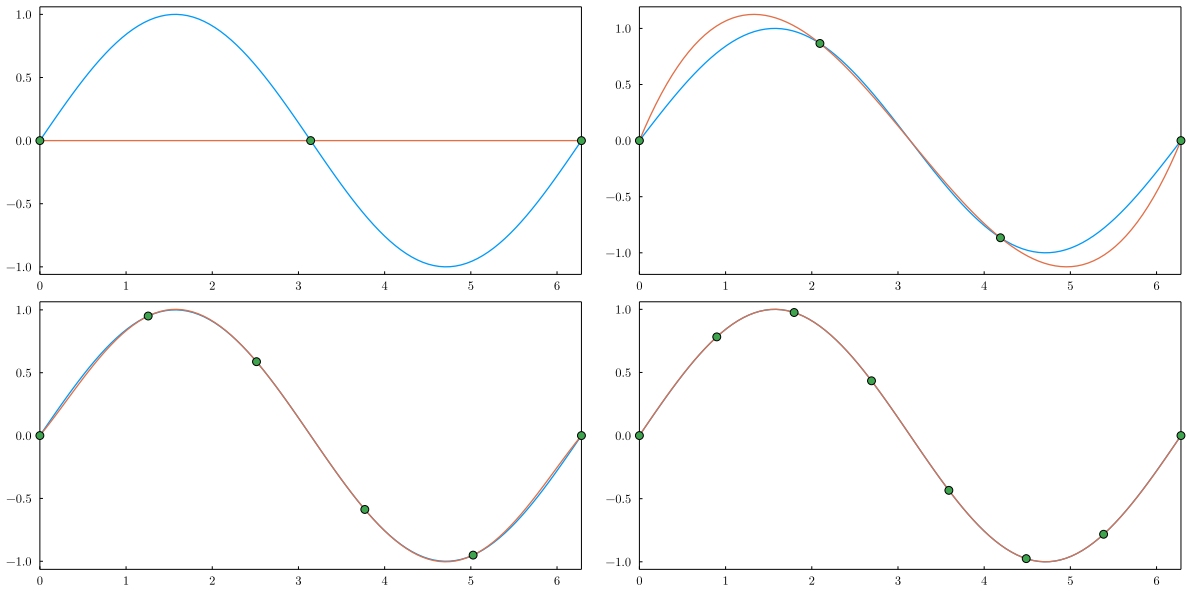


Figure 2.2: Interpolation (in orange) of the function $u(x) = \sin(x)$ (in blue) using 3, 4, 6, and 8 equidistant nodes.

where h is the maximum spacing between two successive interpolation nodes.

Proof. By Theorem 2.3, it holds that

$$\forall x \in [a, b], \qquad |e_n(x)| \leqslant \frac{C_{n+1}}{(n+1)!} \Big| (x - x_0) \dots (x - x_n) \Big|. \tag{2.15}$$

The product on the right-hand side is bounded from above by

$$\frac{h^2}{4} \times 2h \times 3h \times 4h \times \cdots \times nh = \frac{n! h^{n+1}}{4}. \tag{2.16}$$

The first factor comes from the fact that, if $x \in [x_i, x_{i+1}]$, then

$$\Big| (x - x_i)(x - x_{i+1}) \Big| \leqslant \frac{(x_{i+1} - x_i)^2}{4},$$

because the left-hand side is maximized when x is the midpoint of the interval $[x_i, x_{i+1}]$. Substituting (2.16) into (2.15), we deduce the statement. □

We now ask the following natural question: does E_n given in (2.14) tend to zero as the maximum spacing between successive nodes tends to 0? By Corollary 2.4, the answer to this question is positive when C_n does not grow too quickly as $n \to \infty$. For example the interpolation error for the function $u(x) = \sin(x)$ decreases very quickly as $n \to \infty$ when equidistant interpolation nodes are employed, as illustrated in Figure 2.2.

In some cases, however, the constant C_n grows quickly with n, to the extent that E_n may increase with n; in this case, the maximum interpolation error grows when nodes are added! The classic example illustrating this potential issue is that of the Runge function:

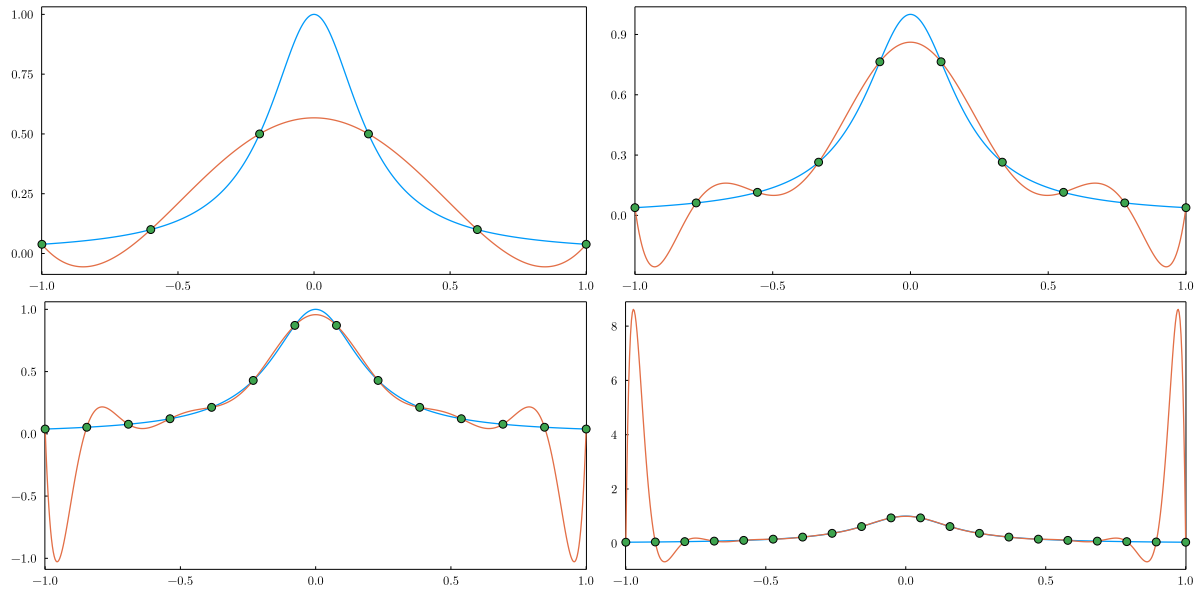


Figure 2.3: Interpolation (in orange) of the Runge function (2.17) (in blue) using 6, 10, 14, and 20 equidistant nodes.

$$u(x) = \frac{1}{1 + 25x^2}. \tag{2.17}$$

It is possible to show that, for this function, the upper bound in (2.14) tends to ∞ in the limit as the number n of interpolation nodes increases. We emphasize that this does not prove that $E_n \to \infty$ in the limit as $n \to \infty$, because (2.14) provides only an *upper bound* on the error. In fact, the interpolation error for the Runge function can either grow or decrease, depending on the choice of interpolation nodes. With equidistant nodes, it turns out that $E_n \to \infty$, as illustrated in Figure 2.3.

2.1.5 Interpolation at Chebyshev nodes

Sometimes, interpolation is employed as a technique for approximating functions. The spectral collocation method, for example, is a technique for solving partial differential equations where a discrete solution is first calculated, and then a continuous solution is constructed using polynomial or Fourier interpolation. When the interpolation nodes are not given a priori as data, it is natural to wonder whether these can be picked in such a manner that the interpolation error, measured in a function norm, is minimized. For example, given a continuous function $u(x)$ and a number of nodes $n + 1$, is it possible to choose nodes $x_0, \ldots, x_n$ such that

$$E := \sup_{x \in [a,b]} \left| u(x) - \hat{u}(x) \right|$$

is minimized? Here $\hat{u}$ is the polynomial interpolating u at the nodes. Achieving this goal in general is a difficult task, because $\xi = \xi(x)$ is unknown in the expression of the interpolation error from Theorem 2.3:

$$e_n(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0)\dots(x - x_n).$$

In view of this difficulty, we will focus on the simpler problem of finding interpolation nodes such that the product $(x - x_0)\dots(x - x_n)$ is minimized in the supremum norm. This problem amounts to finding the optimal interpolation nodes, in the sense that E is minimized, in the particular case where u is a polynomial of degree $n+1$, because in this case $u^{(n+1)}(\xi)$ is a constant factor. It turns out that this problem admits an explicit solution, which we will deduce from the following theorem.

Theorem 2.5 (Minimum ∞ norm)**.** *Assume that p is a monic polynomial of degree $n \geqslant 1$:*

$$p(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_{n-1}x^{n-1} + x^n.$$

Then it holds that

$$\sup_{x\in[-1,1]} \left|p(x)\right| \geqslant \frac{1}{2^{n-1}} =: E. \tag{2.18}$$

In addition, the lower bound is achieved for $p_*(x) = 2^{-(n-1)}T_n(x)$, where T_n is the Chebyshev polynomial of degree n:

$$T_n(x) = \cos(n \arccos x) \qquad (-1 \leqslant x \leqslant 1). \tag{2.19}$$

Proof. By Exercise C.5, the polynomial $x \mapsto 2^{-(n-1)}T_n(x)$ is indeed monic, and it is clear that it achieves the lower bound (2.18) since $|T_n(x)| \leqslant 1$ for all $x \in [-1, 1]$.

In order to prove (2.18), we assume by contradiction that there is a different monic polynomial q of degree n such that

$$\sup_{x\in[-1,1]} \left|q(x)\right| < E. \tag{2.20}$$

Let us introduce $x_i = \cos(i\pi/n)$, for $i = 0, \dots, n$, and observe that

$$p(x_i) = 2^{-(n-1)}T_n(x_i) = (-1)^i E.$$

The function $h(x) \coloneqq p(x) - q(x)$ is a polynomial of degree at most $n-1$ which, by the assumption (2.20), is strictly positive at $x_0, x_2, x_4, \dots$ and strictly negative at $x_1, x_3, x_5, \dots$. Therefore, the polynomial $h(x)$ changes sign at least n times and so, by the intermediate value theorem, it has at least n roots. But this is impossible, because $h(x) \neq 0$ and $h(x)$ is of degree at most $n-1$. □

Remark 2.1 (Derivation of Chebyshev polynomials). The polynomial p_* achieving the lower bound in (2.18) oscillates between the values $-E$ and E, which are respectively its minimum and maximum values over the interval $[-1, 1]$. It attains the values E or $-E$ at $n+1$ distinct points $x_0 < \dots < x_n$, with $x_0 = -1$ and $x_n = 1$. It turns out that these properties, which

can be shown to hold a priori using Chebyshev's *equioscillation theorem*, are sufficient to derive an explicit expression for the polynomial p_*, as we formally demonstrate hereafter.

The points $x_1, \dots, x_{n-1}$ are local extrema of p_*, and so $p_*'(x) = 0$ at these nodes. We therefore deduce that p_* satisfies the differential equation

$$n^2(E^2 - p_*(x)^2) = p_*'(x)^2(1 - x^2). \tag{2.21}$$

Indeed, both sides are polynomials of degree $2n$ with single roots at -1 and 1, with double roots at $x_1, \dots, x_{n-1}$, and with the same coefficient of the leading power. In order to solve (2.21), we rearrange the equation and take the square root:

$$\frac{p_*' \frac{x}{E}}{\sqrt{1 - \frac{p_*(x)^2}{E^2}}} = \pm \frac{n}{\sqrt{1 - x^2}} \qquad \iff \qquad \frac{\mathrm{d}}{\mathrm{d}\,x}\left(\arccos\left(\frac{p_*(x)}{E}\right)\right) = \pm n \frac{\mathrm{d}}{\mathrm{d}\,x} \arccos(x).$$

Integrating both sides and taking the cosine, we obtain

$$p_*(x) = E \cos\Big(C + n \arccos(x)\Big).$$

Requiring that $|p_*(-1)| = E$, we deduce $C = 0$.

From Theorem 2.5, we deduce the following corollary.

Corollary 2.6 (Chebyshev nodes)**.** *Assume that $x_0 < x_1 < \ldots < x_n$ are in the interval $[a, b]$. The supremum norm of the product $\omega(x) \coloneqq (x - x_0) \cdots (x - x_n)$ over $[a, b]$ is minimized when*

$$x_i = a + (b - a) \frac{1 + \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)}{2} \tag{2.22}$$

Proof. We consider the affine change of variable

$$\begin{aligned} \zeta \colon [-1, 1] &\to [a, b]; \\ y &\mapsto \frac{a + b + y(b - a)}{2}. \end{aligned}$$

The function

$$\begin{aligned} p(y) &\coloneqq \frac{2^{n+1}}{(b-a)^{n+1}} \omega\big(\zeta(y)\big) = \frac{2^{n+1}}{(b-a)^{n+1}} \big(\zeta(y) - x_0\big) \cdots \big(\zeta(y) - x_n\big) \\ &= \big(y - y_0\big) \cdots \big(y - y_n\big), \qquad y_i = \zeta^{-1}(x_i), \end{aligned}$$

is a monic polynomial of degree $n + 1$ such that

$$\sup_{y \in [-1,1]} |p(y)| = \frac{2^{n+1}}{(b-a)^{n+1}} \sup_{x \in [a,b]} |(x - x_0) \dots (x - x_n)|. \tag{2.23}$$

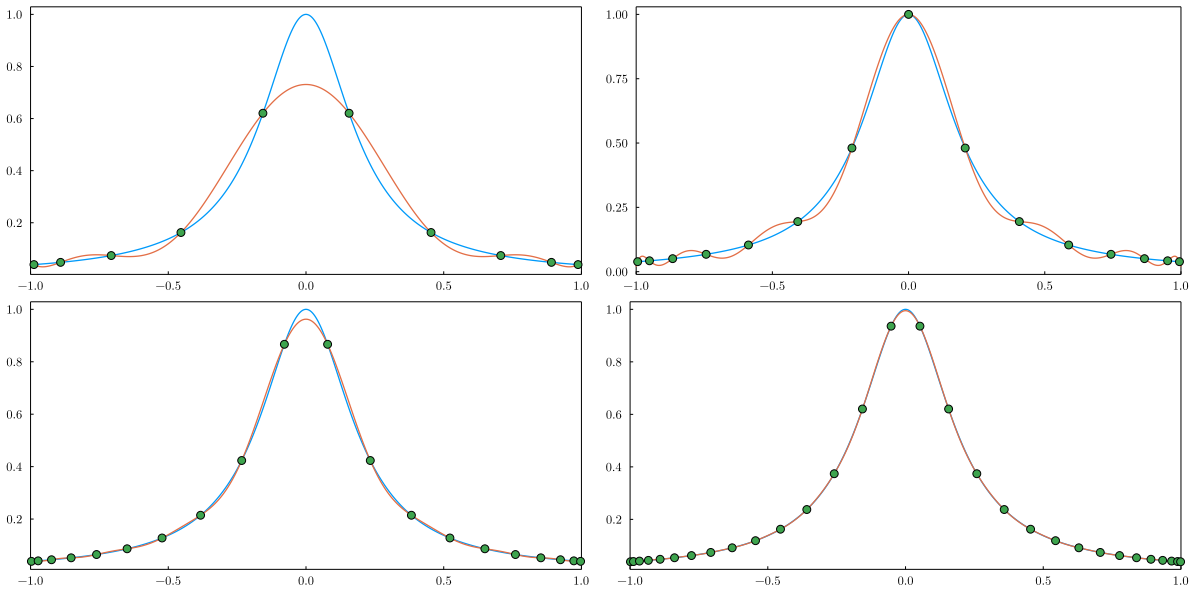


Figure 2.4: Interpolation (in orange) of the Runge function (2.17) (in blue) using 10, 15, 20, and 30 Chebyshev nodes.

By Theorem 2.5, the left-hand side is minimized when p is equal to $2^{-n}T_{n+1}$, i.e. when the roots of p coincide with the roots of T_{n+1}. This occurs when

$$y_i = \zeta^{-1}(x_i) = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right).$$

Applying the inverse change of variable $x_i = \zeta(y_i)$, we deduce the result. □

Corollary 2.6 is useful for interpolation. The nodes

$$x_i = a + (b-a)\frac{1+\cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)}{2}, \qquad i = 0, \ldots, n, \tag{2.24}$$

are known as Chebyshev nodes and, more often than not, employing these nodes for interpolation produces much better results than using equidistant nodes, both in the case where u is a polynomial of degree $n+1$, as we just proved, but also for general u. As an example we plot in Figure 2.4 the interpolation of the Runge function using Chebyshev nodes. In this case, the interpolating polynomial converges uniformly to the Runge function as we increase the number of interpolation nodes!

2.1.6 Hermite interpolation

Hermite interpolation, sometimes also called Hermite–Birkoff interpolation, generalizes Lagrange interpolation to the case where, in addition to the function values $u_0, \ldots, u_n$, the values of some of the derivatives are given at the interpolation nodes. For simplicity, we assume in this section that only the first derivative is specified. In this case, the aim of Hermite interpolation is to find, given data (x_i, u_i, u_i') for $i \in \{0, \ldots, n\}$, a polynomial $\hat{u}$ of degree at most $2n+1$ such that

$$\forall i \in \{0, \dots, n\}, \qquad \hat{u}(x_i) = u_i, \qquad \hat{u}'(x_i) = u_i'.$$

In order to construct the interpolating polynomial, it is useful to define the functions

$$\psi_i(x) = \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)^2, \qquad i = 0, \dots, n.$$

The function ψ_i is the square of the usual Lagrange polynomials associated with x_i, and it satisfies

$$\psi_i(x_i) = 1, \qquad \psi_i'(x_i) = \sum_{j=0, j\neq i}^{n} \frac{2}{x_i - x_j}, \qquad \forall j \neq i \qquad \psi_i(x_j) = \psi_i'(x_j) = 0.$$

We consider the following ansatz for $\hat{u}$:

$$\hat{u}(x) = \sum_{i=0}^{n} \psi_i(x) q_i(x),$$

where q_i are polynomials to be determined of degree at most one, so that $\hat{u}$ is of degree at most $2n + 1$. We then require

$$\hat{u}(x_i) = q_i(x_i), \qquad \hat{u}'(x_i) = \psi_i'(x_i) q(x_i) + q'(x_i).$$

From the first equation, we deduce that $q_i(x_i) = u_i$, and from the second equation we then have $q'(x_i) = \hat{u}'(x_i) - \psi_i'(x_i) u_i$. We conclude that the interpolating polynomial is given by

$$\hat{u}(x) = \sum_{i=0}^{n} \psi_i(x) \Big(u_i + \big(u_i' - \psi_i'(x_i) u_i \big)(x - x_i) \Big).$$

The following theorem gives an expression of the error.

Theorem 2.7 (Hermite interpolation error)**.** *Assume that $u\colon [a, b] \to \mathbf{R}$ is a function in $C^{2n+2}([a, b])$ and let $\hat{u}$ denote the Hermite interpolation of u at the nodes $x_0, \dots, x_n$. Then for all $x \in [a, b]$, there exists $\xi = \xi(x)$ in the interval $[a, b]$ such that*

$$u(x) - \hat{u}(x) = \frac{u^{(2n+2)}(\xi)}{(2n + 2)!} (x - x_0)^2 \dots (x - x_n)^2.$$

Proof. See Exercise 2.8. □

2.1.7 Piecewise interpolation

The interpolation methods we discussed so far are in some sense global; they aim to construct one polynomial that goes through all the data points. This approach is attractive because the interpolant is infinitely smooth but, as we observed, it is not always fruitful, in particular when equidistant interpolation nodes are employed. An alternative approach is to divide the domain in a number of small intervals and perform polynomial interpolation within each interval.

Although the resulting interpolating function is usually not smooth over the full domain, this "local" approach to interpolation is more robust.

Several methods belong in the category of piecewise interpolation. We mention, for instance, piecewise Lagrange interpolation and cubic splines interpolation. In this section, we briefly describe the former method, which is widely used in the context of the *finite element method*. Information on the latter method is available in [10, Section 8.7.1.].

For simplicity, we illustrate the method in dimension 1, but piecewise Lagrange interpolation can be extended to several dimensions. Assume that we wish to approximate a function $u\colon [a,b] \to \mathbf{R}$. We consider a subdivision $a = x_0 < x_1 < \ldots < x_n = b$ of the interval $[a,b]$ and let h denote the maximum spacing:

$$h = \max_{i \in \{0,\ldots,n-1\}} |x_{i+1} - x_i|.$$

Within each subinterval $I_i = [x_i, x_{i+1}]$, we consider a further subdivision

$$x_i = x_i^{(0)} < x_i^{(1)} < \ldots < x_i^{(m)} = x_{i+1},$$

where the nodes $x_i^{(0)}, \ldots, x_i^{(m)}$ are equally spaced with distance h/m. The idea of piecewise Lagrange interpolation is to calculate, for each interval I_i in the partition, the interpolating polynomial p_i at the nodes $x_i^{(0)}, \ldots, x_i^{(m)}$. The interpolant is then defined as

$$\hat{u}(x) = p_\iota(x), \tag{2.25}$$

where $\iota = \iota(x)$ is the index of the interval to which x belongs. Since $x_i^{(m)} = x_{i+1} = x_{i+1}^{(0)}$, the interpolant defined by (2.25) is continuous. If the function u belongs to $C^{m+1}([a,b])$, then by Corollary 2.4 the interpolation error within each subinterval may be bounded from above as follows:

$$\sup_{x \in I_i} |u(x) - \hat{u}(x)| \leqslant \frac{C_{m+1}(h/m)^{m+1}}{4(m+1)}, \qquad C_{m+1} := \sup_{x \in [a,b]} |u^{(m+1)}(x)|, \tag{2.26}$$

and so we deduce

$$\sup_{x \in [a,b]} |u(x) - \hat{u}(x)| \leqslant Ch^{m+1},$$

for an appropriate constant C independent of h. This equation shows that the error is guaranteed to decrease to 0 in the limit as $h \to 0$. In practice, the number m of interpolation nodes within each interval can be small.

2.2 Approximation

In this section, we focus on the subject of *approximation*, both of discrete data points and of continuous functions. We begin, in Section 2.2.1 with a discussion of least squares approximation

for data points, and in Section 2.2.2 we focus on function approximation in the mean square sense.

2.2.1 Least squares approximation of data points

Consider $n+1$ distinct x values $x_0 < \ldots < x_n$, and suppose that we know the values $u_0, \ldots, u_n$ taken by an unknown function u when evaluated at these points. We wish to approximate the function u by a function of the form

$$\widehat{u}(x) = \sum_{i=0}^{m} \alpha_i \varphi_i(x) \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}, \tag{2.27}$$

for some $m < n$. In many cases of practical interest, the basis functions $\varphi_0, \ldots, \varphi_m$ are polynomials. In contrast with interpolation, here we seek a function $\widehat{u}$ in a finite-dimensional function space of dimension m strictly lower than the number of data points. In order for $\widehat{u}$ to be a good approximation, we wish to find coefficients $\alpha_0, \ldots, \alpha_m$ such that the following linear system is approximately satisfied.

$$\mathsf{A}\boldsymbol{\alpha} := \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_m(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_m(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \ldots & \varphi_m(x_2) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_{n-2}) & \varphi_1(x_{n-2}) & \ldots & \varphi_m(x_{n-2}) \\ \varphi_0(x_{n-1}) & \varphi_1(x_{n-1}) & \ldots & \varphi_m(x_{n-1}) \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_m(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix} \approx \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{pmatrix} =: \boldsymbol{b}.$$

In general, since the matrix on the left-hand side has more lines than columns, there does not exist an exact solution to this equation. In order to find an approximate solution, a natural approach is to find coefficients $\alpha_0, \ldots, \alpha_m$ such that the residual vector $\boldsymbol{r} = \mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}$ is small in some vector norm. A particularly popular approach, known as least squares approximation, is to minimize the Euclidean norm of $\boldsymbol{r}$ or, equivalently, the square of the Euclidean norm:

$$\|\boldsymbol{r}\|^2 = \sum_{i=0}^{n} \left| u_i - \widehat{u}(x_i) \right|^2 = \sum_{i=0}^{n} \left(u_i - \sum_{j=0}^{m} \alpha_j \varphi_j(x_i) \right)^2.$$

Let us denote the right-hand side of this equation by $J(\boldsymbol{\alpha})$, which we view as a function of the vector of coefficients $\boldsymbol{\alpha}$. A necessary condition for $\boldsymbol{a}_*$ to be a minimizer is that $\nabla J(\boldsymbol{\alpha}_*) = 0$. The gradient of J, written as a column vector, is given by

$$\begin{aligned} \nabla J(\boldsymbol{\alpha}) &= \nabla \Big((\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b})^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) \Big) \\ &= \nabla \Big(\boldsymbol{\alpha}^T (\mathsf{A}^T \mathsf{A}) \boldsymbol{\alpha} - \boldsymbol{b} \mathsf{A} \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathsf{A}^T \boldsymbol{b} + \boldsymbol{b}^T \boldsymbol{b} \Big)) \\ &= 2(\mathsf{A}^T \mathsf{A}) \boldsymbol{\alpha} - 2\mathsf{A}^T \boldsymbol{b}. \end{aligned}$$

We deduce that $\boldsymbol{\alpha}_*$ solves the linear system

$$\mathsf{A}^T\mathsf{A}\boldsymbol{\alpha}_* = \mathsf{A}^T\boldsymbol{b}, \tag{2.28}$$

where the matrix on the left-hand side is given by:

$$\mathsf{A}^T\mathsf{A} := \begin{pmatrix} \sum_{i=0}^n \varphi_0(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_0(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_0(x_i)\varphi_m(x_i) \\ \sum_{i=0}^n \varphi_1(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_1(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_1(x_i)\varphi_m(x_i) \\ \vdots & \vdots & & \vdots \\ \sum_{i=0}^n \varphi_m(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_m(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_m(x_i)\varphi_m(x_i) \end{pmatrix}.$$

Equation (2.28) is a system of m equations with m unknowns, which admits a unique solution provided that $\mathsf{A}^T\mathsf{A}$ is full rank or, equivalently, the columns of A are linearly independent. The linear equations (2.28) are known as the *normal equations*. As a side note, we mention that the solution $\boldsymbol{\alpha}_* = (\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T\boldsymbol{b}$ coincides with the maximum likelihood estimator for α under the assumption that the data is generated according to $u_i = u(x_i) + \varepsilon_i$, for some function $u \in \operatorname{Span}\{\varphi_0, \dots, \varphi_m\}$ and random noise $\varepsilon_i \sim \mathcal{N}(0, 1)$.

Remark 2.2. From equation (2.28) we deduce that

$$\mathsf{A}\boldsymbol{\alpha}_* = \mathsf{A}(\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T\boldsymbol{b}.$$

The matrix $\Pi_\mathsf{A} := \mathsf{A}(\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T$ on the right-hand side is the orthogonal projection operator onto $\operatorname{col}(\mathsf{A}) \subset \mathbf{R}^n$, the subspace spanned by the columns of A. Indeed, it holds that $\Pi_\mathsf{A}^2 = \Pi_\mathsf{A}$, which is the defining property of a projection operator.

To conclude this section, we note that the matrix $\mathsf{A}^+ = (\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T$ is a left inverse of the matrix A, because $\mathsf{A}^+\mathsf{A} = \mathsf{I}$. It is also called the Moore–Penrose inverse or pseudoinverse of the matrix A, which generalizes the usual inverse matrix. In Julia, the backslash operator silently uses the Moore–Penrose inverse when employed with a rectangular matrix. Therefore, solving the normal equations (2.28) can be achieved by just writing `α = A\b`.

2.2.2 Mean square approximation of functions

The approach described in Section 2.2.1 can be generalized to the setting where the actual function u, rather than just discrete evaluations of it, is available. In this section, we seek an approximation of the form (2.27) such that the error $\hat{u}(x) - u(x)$, measured in some function norm, is minimized. Of course, the solution to this minimization problem depends in general on the norm employed, and may in some cases not even be unique. Instead of specifying a particular norm, as done in Section 2.2.1, in this section we retain some generality and assume only that the norm is induced by an inner product on the space of real-valued continuous functions:

$$\langle \bullet, \bullet \rangle \colon C([a,b]) \times C([a,b]) \to \mathbf{R}. \tag{2.29}$$

In other words, we seek to minimize

$$J(\boldsymbol{\alpha}) := \|\hat{u} - u\|^2 = \langle \hat{u} - u, \hat{u} - u \rangle.$$

This is again a function of the $m+1$ variables $\alpha_0, \ldots, \alpha_m$. Before calculating its gradient, we rewrite the function $J(\boldsymbol{\alpha})$ in a simpler form:

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \left\langle u - \sum_{j=0}^{m} \alpha_j \varphi_j, u - \sum_{k=0}^{m} \alpha_k \varphi_k \right\rangle \\ &= \sum_{j=0}^{m} \sum_{k=0}^{m} \alpha_j \alpha_k \langle \varphi_j, \varphi_k \rangle - 2 \sum_{j=0}^{m} \alpha_j \langle u, \varphi_j \rangle + \langle u, u \rangle = \boldsymbol{\alpha}^T \mathsf{G} \boldsymbol{\alpha} - 2 \boldsymbol{b}^T \boldsymbol{\alpha} + \langle u, u \rangle, \end{aligned}$$

where we introduced

$$\mathsf{G} := \begin{pmatrix} \langle \varphi_0, \varphi_0 \rangle & \langle \varphi_0, \varphi_1 \rangle & \cdots & \langle \varphi_0, \varphi_m \rangle \\ \langle \varphi_1, \varphi_0 \rangle & \langle \varphi_1, \varphi_1 \rangle & \cdots & \langle \varphi_1, \varphi_m \rangle \\ \vdots & \vdots & & \vdots \\ \langle \varphi_m, \varphi_0 \rangle & \langle \varphi_m, \varphi_1 \rangle & \cdots & \langle \varphi_m, \varphi_m \rangle \end{pmatrix}, \qquad \boldsymbol{b} := \begin{pmatrix} \langle u, \varphi_0 \rangle \\ \langle u, \varphi_1 \rangle \\ \vdots \\ \langle u, \varphi_m \rangle \end{pmatrix}. \tag{2.30}$$

Employing the same approach as in the previous section, we then obtain $\nabla J(\boldsymbol{\alpha}) = \mathsf{G}\boldsymbol{\alpha} - \boldsymbol{b}$, and so the minimizer of $J(\boldsymbol{\alpha})$ is the solution to the equation

$$\mathsf{G}\boldsymbol{\alpha} = \boldsymbol{b}. \tag{2.31}$$

The matrix G, known as the *Gram matrix*, is positive semi-definite and nonsingular provided that the basis functions are linearly independent, see Exercise 2.9. Therefore, the solution $\boldsymbol{\alpha}_*$ exists and is unique. In addition, since the Hessian of J is equal to G, the vector $\boldsymbol{\alpha}_*$ is indeed a minimizer. Note that if $\langle \bullet, \bullet \rangle$ is defined as a finite sum of the form

$$\langle f, g \rangle = \sum_{i=0}^{n} f(x_i) g(x_i), \tag{2.32}$$

then (2.31) coincides with the normal equations (2.28) from the previous section. We remark that (2.32) is in fact not an inner product on the space of continuous functions, but it is an inner product on the space of polynomials of degree less than or equal to n.

In practice, the matrix and right-hand side of the linear system (2.31) can usually not be calculated exactly, because the inner product $\langle \bullet, \bullet \rangle$ is defined through an integral; see (2.33) in the next section.

Remark 2.3. Rewriting the normal equations (2.31) in terms of $\hat{u}$ we obtain

$$\langle \hat{u} - u, \varphi_0 \rangle = 0, \qquad \ldots, \qquad \langle \hat{u} - u, \varphi_m \rangle = 0.$$

Therefore, the optimal approximation $\hat{u} \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}$ satisfies

$$\forall v \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}, \qquad \langle \hat{u} - u, v \rangle = 0.$$

This shows that the optimal approximation $\hat{u}$, in the sense of the norm $\|\bullet\|$, is the orthogonal projection of u onto $\mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}$.

2.2.3 Orthogonal polynomials

The Gram matrix G in (2.31) is equal to the identity matrix when the basis functions are orthonormal for the inner product considered. In this case, the solution to the linear system is

$$\alpha_i = \langle u, \varphi_i \rangle, \qquad i = 0, \ldots, m,$$

and so the best approximation $\hat{u}$ (for the norm induced by the inner product considered!) is simply given by

$$\hat{u} = \sum_{i=0}^{m} \langle u, \varphi_i \rangle \varphi_i.$$

The coefficients $\langle u, \varphi_i \rangle$ of the basis functions in this expansion are called *Fourier coefficients*. Given a finite dimensional subspace $\mathcal{S}$ of the space of continuous functions, an orthonormal basis can be constructed via the Gram–Schmidt process. In this section, we focus on the particular case where $\mathcal{S} = \mathbf{P}(n)$ – the subspace of polynomials of degree less than or equal to n. Another widely used approach, which we do not explore in this course, is to use trigonometric basis functions. We also assume that the inner product (2.29) is of the form

$$\langle f, g \rangle = \int_a^b f(x) g(x) w(x) \, \mathrm{d}\, x, \tag{2.33}$$

where $w(x)$ is a given nonnegative weight function such that

$$\int_a^b w(x) \, \mathrm{d}\, x > 0.$$

Let $\varphi_0(x), \varphi_1(x), \varphi_2(x) \ldots$ denote the orthonormal polynomials obtained by applying the Gram–Schmidt procedure to the monomials $1, x, x^2, \ldots$. These depend in general on the weight $w(x)$ and on the interval $[a, b]$. A few of the popular classes of orthogonal polynomials are presented in the table below:

Name	$w(x)$	$[a, b]$
Legendre	1	$[-1, 1]$
Chebyshev	$\frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$
Hermite	$\exp\left(-\frac{x^2}{2}\right)$	$[-\infty, \infty]$
Laguerre	e^{-x}	$[0, \infty]$

Orthogonal polynomials have a rich structure, and in the rest of this section we prove some of their key properties, one of which will be very useful in the context of numerical integration in Chapter 3. We begin by showing that orthogonal polynomials have distinct real roots.

Proposition 2.8. *Assume for simplicity that $w(x) > 0$ for all $x \in [a, b]$, and let $\varphi_0, \varphi_1, \ldots$ denote orthonormal polynomials of increasing degree for the inner product (2.33). Then for all $n \in \mathbf{N}$, the polynomial φ_n has n distinct roots in the open interval (a, b).*

Proof. Reasoning by contradiction, we assume that φ_n changes sign at only $k < n$ points of the open interval (a, b), which we denote by $x_1, \ldots, x_k$. Then

$$\varphi_n(x) \times (x - x_0)(x - x_1) \ldots (x - x_k)$$

is either everywhere nonnegative or everywhere nonpositive over $[a, b]$. But then

$$\int_a^b \varphi_n(x) \times (x - x_1) \ldots (x - x_k)\, w(x) \,\mathrm{d}\, x$$

is nonzero, which is a contradiction because the product $(x - x_1) \ldots (x - x_k)$ is a polynomial of degree k, which is orthogonal to φ_n by assumption. Indeed, being orthogonal to $\varphi_0, \ldots, \varphi_{n-1}$, the polynomial φ_n is also orthogonal to any linear combination of these polynomials. □

Next, we show that orthogonal polynomials satisfy a three-term recurrence relation.

Proposition 2.9. *Assume that $\varphi_0, \varphi_1, \ldots$ are orthonormal polynomials for some inner product of the form (2.33) such that φ_i is of degree i. Then*

$$\forall n \in \{1, 2, \ldots\}, \qquad \alpha_{n+1}\varphi_{n+1}(x) = (x - \beta_n)\varphi_n(x) - \alpha_n \varphi_{n-1}(x), \tag{2.34}$$

where

$$\alpha_n = \left\langle x\varphi_n, \varphi_{n-1} \right\rangle, \qquad \beta_n = \left\langle x\varphi_n, \varphi_n \right\rangle.$$

In addition, $\alpha_1\varphi_1(x) = (x - \beta_0)\varphi_0(x)$.

Proof. Since $x\varphi_n(x)$ is a polynomial of degree $n + 1$, it may be decomposed as

$$x\varphi_n(x) = \sum_{i=0}^{n+1} \gamma_{n,i}\varphi_i(x). \tag{2.35}$$

Taking the inner product of both sides of this equation with φ_i and employing the orthonormality assumption, we obtain an expression for the coefficients:

$$\gamma_{n,i} = \left\langle x\varphi_n, \varphi_i \right\rangle.$$

From the expression (2.33) of the inner product, it is clear that $\left\langle x\varphi_n, \varphi_i \right\rangle = \left\langle \varphi_n, x\varphi_i \right\rangle$. Since $x\varphi_i$ is a polynomial of degree $i + 1$ and φ_n is orthogonal to all polynomials of degree strictly less than n, we deduce that $\gamma_{n,i} = 0$ if $i < n - 1$. Consequently, we can rewrite the right-hand side of (2.35) as a sum involving only three terms

$$x\varphi_n(x) = \left\langle x\varphi_n, \varphi_{n-1}\right\rangle \varphi_{n-1}(x) + \left\langle x\varphi_n, \varphi_n\right\rangle \varphi_n(x) + \left\langle x\varphi_n, \varphi_{n+1}\right\rangle \varphi_{n+1}(x). \tag{2.36}$$

Since $\left\langle x\varphi_n, \varphi_{n+1}\right\rangle = \left\langle x\varphi_{n+1}, \varphi_n\right\rangle$, we obtain the statement after rearranging. □

Remark 2.4. Notice that the polynomials in Proposition 2.9 are orthonormal by assumption, and so the coefficient α_{n+1} is just a normalization constant. We deduce that

$$\varphi_{n+1}(x) = \frac{(x-\beta_n)\varphi_n(x) - \alpha_n\varphi_{n-1}(x)}{\|(x-\beta_n)\varphi_n(x) - \alpha_n\varphi_{n-1}(x)\|},$$

which enables to calculate the orthogonal polynomials recursively.

2.2.4 Orthogonal polynomials and numerical integration: an introduction

Equation (2.36) may be rewritten in matrix form as follows:

$$\begin{pmatrix} x\varphi_0(x) \\ x\varphi_1(x) \\ x\varphi_2(x) \\ \vdots \\ x\varphi_{m-1}(x) \\ x\varphi_m(x) \end{pmatrix} = \begin{pmatrix} \beta_0 & \alpha_1 & & & & \\ \alpha_1 & \beta_1 & \alpha_2 & & & \\ & \alpha_2 & \beta_2 & \alpha_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{m-1} & \beta_{m-1} & \alpha_m \\ & & & & \alpha_m & \beta_m \end{pmatrix} \begin{pmatrix} \varphi_0(x) \\ \varphi_1(x) \\ \varphi_2(x) \\ \vdots \\ \varphi_{m-1}(x) \\ \varphi_m(x) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_{m+1}\varphi_{m+1}(x) \end{pmatrix}.$$

Let T denote the matrix on the left-hand side of this equation, and let $r_0, \ldots, r_m$ denote the roots of φ_{m+1}. By Proposition 2.8, these are distinct and all belong to the interval (a, b). The second term on the right-hand side cancels out when x is a root of φ_{m+1}, and so

$$\forall r \in \left\{r_0, \ldots, r_m\right\}, \qquad \begin{pmatrix} r\varphi_0(r) \\ r\varphi_1(r) \\ \vdots \\ r\varphi_{m-1}(r) \\ r\varphi_m(r) \end{pmatrix} = \begin{pmatrix} \beta_0 & \alpha_1 & & & \\ \alpha_1 & \beta_1 & \alpha_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \alpha_{m-1} & \beta_{m-1} & \alpha_m \\ & & & \alpha_m & \beta_m \end{pmatrix} \begin{pmatrix} \varphi_0(r) \\ \varphi_1(r) \\ \vdots \\ \varphi_{m-1}(r) \\ \varphi_m(r) \end{pmatrix}.$$

In other words, for any root r of φ_{m+1}, the vector $\begin{pmatrix} \varphi_0(r) & \cdots & \varphi_m(r) \end{pmatrix}^T$ is an eigenvector of the matrix T, with associated eigenvalue equal to r. Since T is a symmetric matrix, the eigenvectors associated with distinct eigenvalues are orthogonal for the Euclidean inner product of $\mathbf{R}^{m+1}$, so given that the eigenvalues of T are distinct, we deduce that

$$\forall i \neq j, \qquad \sum_{i=0}^{m} \varphi_i(r_i)\varphi_i(r_j) = 0. \tag{2.37}$$

Let us construct the matrix

$$\mathsf{P} = \begin{pmatrix} \varphi_0(r_0) & \varphi_1(r_0) & \dots & \varphi_m(r_0) \\ \varphi_0(r_1) & \varphi_1(r_1) & \dots & \varphi_m(r_1) \\ \varphi_0(r_2) & \varphi_1(r_2) & \dots & \varphi_m(r_2) \\ \vdots & \vdots & \dots & \vdots \\ \varphi_0(r_m) & \varphi_1(r_m) & \dots & \varphi_m(r_m) \end{pmatrix}.$$

Equation (2.37) indicates that the rows of P are orthogonal, and so the matrix $\mathsf{D} = \mathsf{P}\mathsf{P}^T$ is diagonal with elements given by

$$d_{ii} = \sum_{j=0}^{m} |\varphi_j(r_i)|^2, \qquad i = 0, \dots, m.$$

(Here we start counting the rows from 0 for convenience.) Since $\mathsf{P}\mathsf{P}^T\mathsf{D}^{-1} = \mathsf{I}$, we deduce that the inverse of P is given by $\mathsf{P}^{-1} = \mathsf{P}^T\mathsf{D}^{-1}$. Consequently,

$$\mathsf{P}^T\mathsf{D}^{-1}\mathsf{P} = \mathsf{P}^{-1}\mathsf{P} = \mathsf{I},$$

which means that the *columns* of P are orthonormal for the inner product $(\boldsymbol{x}, \boldsymbol{y}) \mapsto \boldsymbol{x}^T\mathsf{D}^{-1}\boldsymbol{y}$. In other words, the polynomials $\varphi_1, \dots, \varphi_m$ are orthonormal for the inner product

$$\begin{aligned} \langle \bullet, \bullet \rangle_{m+1} \colon \mathbf{P}(m) \times \mathbf{P}(m) &\to \mathbf{R}; \\ (p, q) &\mapsto \sum_{i=0}^{m} \frac{p(r_i)q(r_i)}{d_{ii}}. \end{aligned}$$

We have thus shown that, if $\varphi_0, \varphi_1, \varphi_2, \dots$ is a family of orthonormal polynomials for an inner product $\langle \bullet, \bullet \rangle$, then these are also orthonormal for the inner product $\langle \bullet, \bullet \rangle_{m+1}$. We reformulate our findings in the following result where, since m was arbitrary in the previous reasoning, we add a superscript to indicate when the quantities involved depend on m.

Theorem 2.10. *Orthonormal polynomials $\varphi_0, \dots, \varphi_m$ for the inner product*

$$\langle f, g \rangle = \int_a^b f(x)g(x)w(x) \,\mathrm{d}\, x$$

are also orthonormal for the inner product

$$\langle f, g \rangle_{m+1} = \sum_{i=0}^{m} f\Big(r_i^{(m+1)}\Big) g\Big(r_i^{(m+1)}\Big) w_i^{(m+1)},$$

where $r_0^{(m+1)}, \dots, r_m^{(m+1)}$ are the roots of φ_{m+1} and the weights $w_i^{(m+1)}$ are given by

$$w_i^{(m+1)} = \frac{1}{\sum_{j=0}^{m} \left|\varphi_j\Big(r_i^{(m+1)}\Big)\right|^2}, \qquad i = 0, \dots, m.$$

As an immediate corollary, we deduce that

$$\forall (p, q) \in \mathbf{P}(m) \times \mathbf{P}(m), \qquad \langle p, q \rangle = \langle p, q \rangle_{m+1}, \tag{2.38}$$

Indeed, denoting by $p = \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m$ and $q = \beta_0\varphi_0 + \cdots + \beta_m\varphi_m$ the expansions of the polynomials p and q in the orthonormal basis, we have

$$\begin{aligned}\langle p, q\rangle &= \langle \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m, \beta_0\varphi_0 + \cdots + \beta_m\varphi_m\rangle \\ &= \sum_{i=0}^{m}\sum_{j=0}^{m} \alpha_i\beta_j\langle\varphi_i, \varphi_j\rangle = \alpha_0\beta_0 + \cdots + \alpha_m\beta_m = \sum_{i=0}^{m}\sum_{j=0}^{m} \alpha_i\beta_j\langle\varphi_i, \varphi_j\rangle_{m+1} \\ &= \langle \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m, \beta_0\varphi_0 + \cdots + \beta_m\varphi_m\rangle_{m+1} = \langle p, q\rangle_{m+1}.\end{aligned}$$

To conclude this section, we prove the following statement, which is another consequence of Theorem 2.10 and has applications to numerical integration.

Theorem 2.11. *It holds that*

$$\forall p \in \mathbf{P}(2m+1), \qquad \int_a^b p(x)\, w(x)\, \mathrm{d}\, x = \sum_{i=0}^{m} p\Big(r_i^{(m+1)}\Big) w_i^{(m+1)}. \tag{2.39}$$

Proof. Taking $q = 1$ in (2.38) and employing the definitions of $\langle \bullet, \bullet\rangle$ and $\langle \bullet, \bullet\rangle_{m+1}$, we have that (2.38) is satisfied for any $p \in \mathbf{P}(m)$. Next, any polynomial $p \in \mathbf{P}(2m+1)$ may be decomposed as $p(x) = \varphi_{m+1}(x)q(x) + \rho(x)$, for some polynomial q of degree m (the quotient of the polynomial division of p by φ_{m+1}) and some polynomial ρ of degree lower than or equal to m (the remainder of the polynomial division). Therefore, since (2.39) was already shown to hold for polynomials of degree up to m, we obtain

$$\begin{aligned}\int_a^b p(x)w(x)\, \mathrm{d}\, x &= \int_a^b \varphi_{m+1}(x)q(x)w(x)\, \mathrm{d}\, x + \int_a^b \rho(x)w(x)\, \mathrm{d}\, x \\ &= 0 + \int_a^b \rho(x)w(x)\, \mathrm{d}\, x = 0 + \sum_{i=0}^{m} \rho(r_i)w_i \\ &= \sum_{i=0}^{m} \varphi_{m+1}(r_i)q(r_i)w_i + \sum_{i=0}^{m} \rho(r_i)w_i = \sum_{i=0}^{m} p(r_i)w_i,\end{aligned}$$

where we dropped the $^{(m+1)}$ superscript for conciseness and we used, in the penultimate inequality, the fact that $r_0, \ldots, r_m$ are the roots of the polynomial φ_{m+1}. □

Since the left-hand side of (2.39) is an integral and the right-hand side is a sum, we have just constructed an integration formula, which enjoys a very nice property: it is exact for polynomials of degree up to $2m+1$! A formula of this type is called a *quadrature formula*, with $m+1$ *nodes* $r_0^{(m+1)}, \ldots, r_m^{(m+1)}$ and associated *weights* $w_0^{(m+1)}, \ldots, w_m^{(m+1)}$. Note that the nodes and weights of the quadrature depend on the weight $w(x)$ and on the degree m. We will revisit this subject in Chapter 3.

2.3 Exercises

Exercise 2.1. *Find the polynomial $p(x) = ax + b$ (a straight line) that goes through the points (x_0, u_0) and (x_1, u_1).*

Exercise 2.2. *Find the polynomial $p(x) = ax^2 + bx + c$ (a parabola) that goes through the points $(0, 1)$, $(1, 3)$ and $(2, 7)$.*

Exercise 2.3. *Prove the following recurrence relation for Chebyshev polynomials:*

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x), \qquad i = 1, 2, \ldots.$$

Exercise 2.4. *Show by recursion that*

$$[u_0, u_1, \ldots, u_n] = \sum_{j=0}^{n} \frac{u_j}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)}. \tag{2.40}$$

Deduce from this identity that

$$[u_0, u_1, \ldots, u_n] = [u_{\sigma_1}, u_{\sigma_2}, \ldots, u_{\sigma_n}],$$

for any permutation $\boldsymbol{\sigma}$ of $(0, 1, 2, \ldots, n)$.

Solution. The first statement (2.40) is clear when $n = 0$. Reasoning by induction, we assume that the statement is true up to $n-1$ and prove that it then also holds for n. Using the definition (2.10) and the induction hypothesis, we obtain that

$$\begin{aligned}[u_0, u_1, \ldots, u_n] &= \frac{[u_1, \ldots, u_n] - [u_0, \ldots, u_{n-1}]}{x_n - x_0} \\ &= \frac{1}{x_n - x_0}\left(\sum_{j=\mathbf{1}}^{n} \frac{u_j}{\prod_{k\in\{\mathbf{1},\ldots,n\}\setminus\{j\}}(x_j - x_k)} - \sum_{j=0}^{\mathbf{n-1}} \frac{u_j}{\prod_{k\in\{0,\ldots,\mathbf{n-1}\}\setminus\{j\}}(x_j - x_k)}\right)\end{aligned}$$

Rewriting the fractions with a common denominator leads to

$$[u_0, u_1, \ldots, u_n] = \frac{1}{x_n - x_0}\sum_{j=0}^{n} \frac{u_j\big((x_j - x_0) - (x_j - x_n)\big)}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)} = \sum_{j=0}^{n} \frac{u_j}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)},$$

which concludes the proof of the first statement. The second statement then follows immediately, because the right-hand side of (2.40) is invariant under permutations. △

Exercise 2.5. *Using the Gregory–Newton formula, find an expression for*

$$\sum_{i=1}^{n} i^4.$$

Exercise 2.6. *Let $(f_0, f_1, f_2, \ldots) = (1, 1, 2, \ldots)$ denote the Fibonacci sequence. Prove that there does not exist a polynomial p such that*

$$\forall n \in \mathbf{N}, \qquad f_n = p(n). \tag{2.41}$$

Solution. Assume by contradiction that $p\colon \mathbf{R} \to \mathbf{R}$ is a polynomial such that (2.41) is satisfied, and let n be the degree of this polynomial. Then it holds that $\Delta^{n+1}p = 0$ (2.5), where both sides are

viewed as functions from $\mathbf{R}$ to $\mathbf{R}$. On the other hand, since $p(n) = f_n$ for all $n \in \mathbf{N}$, we can calculate explicitly the values of taken by the function $\Delta^m p$ when evaluated at all the natural numbers, for all $m \in \mathbf{N}$. We collate a few values in the following table.

n	0	1	2	3	4	5	6
$\Delta^0 p(n)$	**1**	**1**	**2**	**3**	**5**	**8**	**13**
$\Delta^1 p(n)$	0	**1**	**1**	**2**	**3**	**5**	**8**
$\Delta^2 p(n)$	1	0	**1**	**1**	**2**	**3**	**5**
$\Delta^3 p(n)$	-1	1	0	**1**	**1**	**2**	**3**

It appears from these calculations that the Fibonacci sequence is shifted one position to the right with each additional application of Δ. In other words, our calculations suggest that

$$\forall (m, n) \in \mathbf{N} \times \mathbf{N}, \qquad \Delta^m p(m+n) = f_n, \tag{2.42}$$

which is a contradiction. To conclude, let us prove (2.42) rigorously. This equation is obvious for $m = 0$ by assumption. Now, reasoning by contradiction, we assume that (2.42) is true up to m. Then by definition of the difference operator Δ, we have

$$\begin{aligned} \Delta^{m+1} p(m+n+1) &= \Delta^m p(m+n+2) - \Delta^m p(m+n+1) \\ &= f_{n+2} - f_{n+1} = f_n. \end{aligned}$$

Here, we used the induction hypothesis (2.42) in the second equality, and the definition of the Fibonacci series in the third one. △

Exercise 2.7. *Using the Gregory–Newton formula, show that*

$$\forall n \in \mathbf{N}, \qquad 2^n = 1 + n + \frac{n^{\underline{2}}}{2!} + \frac{n^{\underline{3}}}{3!} + \frac{n^{\underline{4}}}{4!} + \cdots \tag{2.43}$$

Solution. Equation (2.43) is a particular case of the following more general statement: for any function $f \in \mathbf{R} \to \mathbf{R}$, it holds that

$$\forall n \in \mathbf{N}, \qquad f(n) = f(0) + \Delta f(0) n + \Delta^2 f(0) \frac{n^{\underline{2}}}{2!} + \Delta^3 f(0) \frac{n^{\underline{3}}}{3!} + \Delta^4 f(0) \frac{n^{\underline{4}}}{4!} + \cdots \tag{2.44}$$

In order to show this equation, it is sufficient to prove that for any $n_* \in \mathbf{N}$, the two sides of (2.44) coincide for every $n \in \{0, \ldots, n_*\}$. Since $n^{\underline{p}} = 0$ for all $n \in \{0, \ldots, p-1\}$ by definition (2.4) of the falling powers, the right-hand side of (2.44) coincides for all $n \in \{0, \ldots, n_*\}$ with

$$g(n) = f(0) + \Delta f(0) n + \Delta^2 f(0) \frac{n^{\underline{2}}}{2!} + \cdots + \Delta^{n_*} f(0) \frac{n^{\underline{n_*}}}{n_*!}.$$

We recognize on the right-hand side Newton's expression of the interpolating polynomial through the points $\big(0, f(0)\big), \ldots, \big(n_*, f(n_*)\big)$, and so $g(n) = f(n)$ for all $n \in \{0, \ldots, n_*\}$, which concludes the proof. △

Remark 2.5. Remarkably, equation (2.43) holds in fact for *any* $n \in \mathbf{R}_{>0}$. However, showing this more general statement is beyond the scope of this course.

Exercise 2.8. *Prove Theorem 2.7.*

Exercise 2.9. *Show that the matrix* G *in (2.30) is positive definite if the basis functions* $\varphi_0, \ldots, \varphi_m$ *are linearly independent.*

Exercise 2.10. *Write a Julia code for interpolating the following function using a polynomial of degree 20 over the interval* $[-1, 1]$.

$$f(x) = \tanh\left(\frac{x+1/2}{\varepsilon}\right) + \tanh\left(\frac{x}{\varepsilon}\right) + \tanh\left(\frac{x-1/2}{\varepsilon}\right), \qquad \varepsilon = .01.$$

Use equidistant and then Chebyshev nodes, and compare the two approaches in terms of accuracy. Plot the function f *together with the approximating polynomials.*

Exercise 2.11. *Write from scratch a function to obtain the polynomial interpolating the data points*

$$(x_0, u_0), \ldots, (x_n, u_n).$$

Your function should return the values taken by the interpolating polynomial when evaluated at the points $X_0, \ldots, X_m$. *You may use the following code to test your function*

```
import Plots
function interp(X, x, u)
    # Your code comes here
end
n, m = 10, 100
f(t) = cos(2π * t)
x = LinRange(0, 1, n)
X = LinRange(0, 1, m)
u = f.(x)
U = interp(X, x, u)
Plots.plot(X, f.(X), label="Original function")
Plots.plot!(X, U, label="Interpolation")
Plots.scatter!(x, u, label="Data")
```

Exercise 2.12. *We wish to use interpolation to approximate the following parametric function, called an* epitrochoid*:*

$$x(\theta) = (R+r)\cos\theta + d\cos\left(\frac{R+r}{r}\theta\right) \tag{2.45}$$

$$y(\theta) = (R+r)\sin\theta - d\sin\left(\frac{R+r}{r}\theta\right), \tag{2.46}$$

with $R = 5$, $r = 2$ *and* $d = 3$, *and for* $\theta \in [0, 4\pi]$. *Write a Julia program to interpolate* $x(\theta)$ *and* $y(\theta)$ *using 40 equidistant points. Use the* `BigFloat` *format in order to reduce the impact*

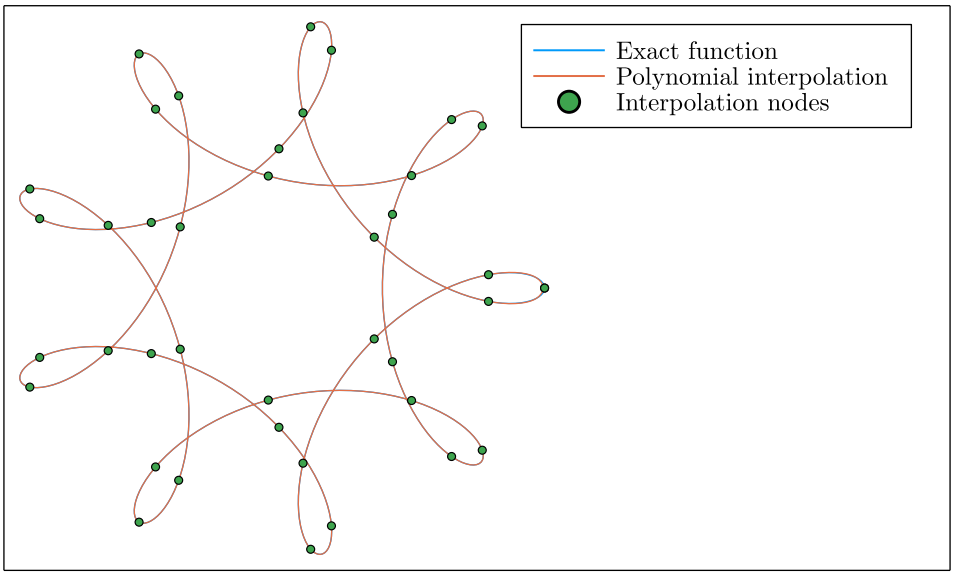


Figure 2.5: Solution for Exercise 2.12.

of round-off errors. After constructing the polynomial interpolations $\hat{x}(\theta)$ and $\hat{y}(\theta)$, plot the parametric curve $\theta \mapsto \big(\hat{x}(\theta), \hat{y}(\theta)\big)$. Your plot should look similar to Figure 2.5.

Exercise 2.13 (Solving the Laplace equation using a spectral method)**.** *The classical Laplace equation with homogeneous Dirichlet boundary conditions in dimension 1 reads*

$$\text{Find } u \in C^2\big([0,1]\big) \text{ such that } \quad \begin{cases} -u''(x) = f(x) & \forall x \in (0,1), \\ u(0) = u(1) = 0. \end{cases} \tag{2.47}$$

Our goal in this exercise is to approximate the exact solution $u(x)$ using interpolation. Specifically, we propose to proceed in two steps:

- *Interpolate the right-hand side using a polynomial with equidistant nodes. That is, find a polynomial $\hat{f} \in \mathbf{P}(n)$ such that*

$$\forall i \in \{0, \dots, n\}, \qquad \hat{f}(x_i) = f(x_i), \qquad x_i = \frac{i}{n}.$$

- *Solve (2.47) with $\hat{f}$ instead of f. Since $\hat{f}$ is a polynomial, this can be achieved analytically.*

Implement this program in the case where

$$f(x) = \exp\big(\sin(2\pi x)\big)\cos(2\pi x)^2 - \exp\big(\sin(2\pi x)\big)\sin(2\pi x),$$

and compare for various values of n the approximate solution you obtain with the exact solution to (2.47), which is given by $u(x) = (2\pi)^{-2}\exp\Big(\big(\sin(2\pi x)\big) - 1\Big)$ in this case.

Exercise 2.14 (Modeling the vapor pressure of mercury)**.** *The dataset loaded through the following Julia commands contains data on the vapor pressure of mercury as a function of the temperature.*

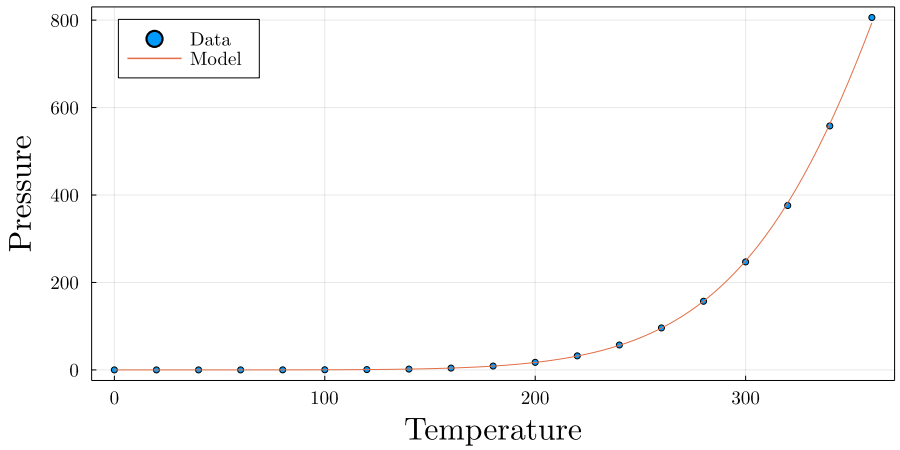


Figure 2.6: Solution for Exercise 2.14.

```
import RDatasets
data = RDatasets.dataset("datasets", "pressure")
```

Find a low-dimensional mathematical model of the form

$$p(T) = \exp\bigl(\alpha_0 + \alpha_1 T + \alpha_2 T^2 + \alpha_3 T^3\bigr) \tag{2.48}$$

for the pressure as a function of the temperature. Plot the approximation together with the data. An example solution is given in Figure 2.6.

Exercise 2.15. *Let* $u\colon [0, 2\pi] \to \mathbf{R}$ *and*

$$x_k = \left(\frac{2k\pi}{2n+1}\right), \qquad k = 0, \ldots, 2n. \tag{2.49}$$

We wish to interpolate u *at these nodes using complex exponentials:*

$$\hat{u} = \sum_{k=-n}^{n} a_k \mathrm{e}^{\mathrm{i}\,kx}.$$

Write a function

```
function fourier_interpolate(u, x, X)
    # Your code comes here ...
end
```

which takes three arguments:

- `u` *is the function to interpolate;*
- `x` *are the interpolation nodes, given by (2.49) in the test code below; you can assume that this array contains an odd number of elements.*
- `X` *is a one-dimensional array of values on the* x *axis.*

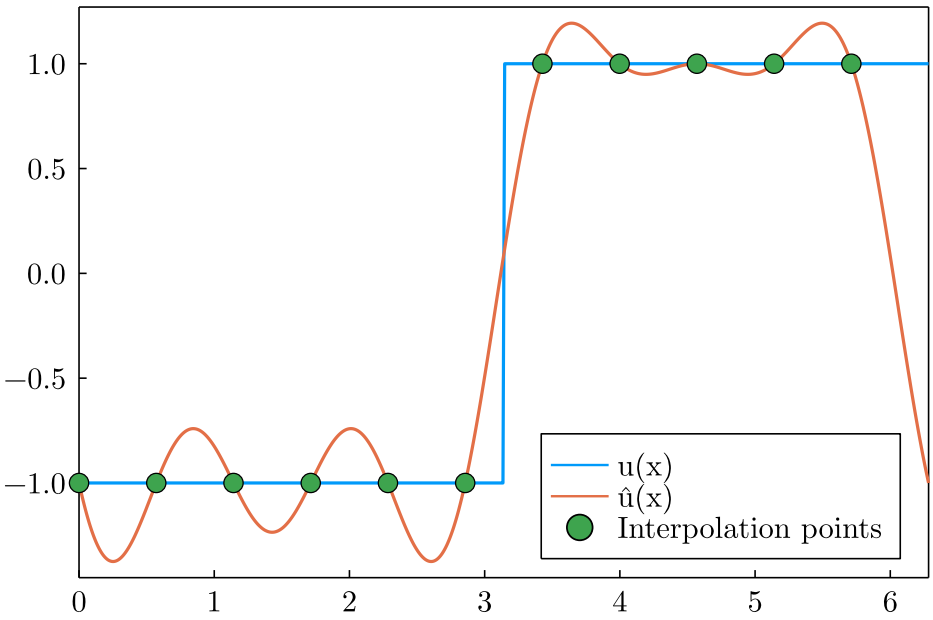


Figure 2.7: Example solution for Exercise 2.15.

The function should return a one-dimensional array containing the values that $\hat{u}$ takes when evaluated at the points contained in `X`*. You can use the following code to test your function:*

```
import Plots
n, m = 5, 1000
x = 2π/(2n+1) * (0:2n)
X = 2π/m * (0:m)

u(x) = sign(x - π)
# u(x) = exp(sin(x) + cos(5x))
# u(x) = x^2 * (x - 2π)^2 / π^4

@time U = fourier_interpolate(u, x, X)
Plots.plot(X, u.(X), label="u(x)", legend=:bottomright)
Plots.plot!(X, U, label="û(x)")
Plots.scatter!(x, u.(x), label="Interpolation points")
Plots.xlims!(0, 2π)
```

An example of the output plot is illustrated in Figure 2.7.

2.4 Discussion and bibliography

A comprehensive study of approximation theory would require to cover the L^∞ setting as well as other functional settings. A pillar of L^∞ approximation theorem is Chebyshev's equioscillation theorem, which we alluded to in Remark 2.1. An excellent introductory reference on approximation theory is [8] (in French). See also [10, Chapter 10] and the references therein.

Chapter 3

Numerical integration

3.1 The closed Newton–Cotes method 57
3.2 Composite methods with equidistant nodes 58
3.3 Richardson extrapolation and Romberg's method 65
3.4 Methods with non-equidistant nodes 69
3.5 Introduction to probabilistic integration methods 73
3.6 Exercises 75
3.7 Discussion and bibliography 81

Introduction

Integrals are ubiquitous in science and mathematics. In this chapter, we are concerned with the problem of calculating numerically integrals of the form

$$I = \int_\Omega u(\boldsymbol{x}) \, \mathrm{d} \, \boldsymbol{x}, \tag{3.1}$$

Perhaps somewhat surprisingly, the numerical calculation of such integrals when $n \gg 1$ is still a very active area of research today. In this chapter, however, we will focus for simplicity on the one-dimensional setting where $\Omega = [a, b] \subset \mathbf{R}$. We assume throughout this chapter that the function u is Riemann-integrable. Then, by definition,

$$I = \lim_{h \to 0} \sum_{i=0}^{n-1} u(t_i)(z_{i+1} - z_i),$$

where $a = z_0 < \cdots < z_n = b$ is a partition of the interval $[a, b]$ such that the maximum spacing between successive x values is equal to h, and with $t_i \in [x_i, x_{i+1}]$ for all $i \in \{0, \ldots, n-1\}$.

All the numerical integration formulas that we present in this chapter are based on a deterministic approximation of the form

$$\hat{I} = \sum_{i=0}^{n} w_i u(x_i), \tag{3.2}$$

where $x_0 < \ldots < x_n$ are the *integration points* and $w_0, \ldots, w_n$ are the *integration weights*. In many cases, integration formulas contain a small parameter that can be refined to improve the accuracy of the approximation. In methods based on equidistant interpolation nodes, for example, this parameter encodes the distance between nodes and is typically denoted by h. We shall often use the notation $\hat{I}_h$ to emphasize the dependence of the approximation on h. The difference $E_h = I - \hat{I}_h$ is called the *integration error* or *discretization error*. The *degree of precision*, defined hereafter, is an important measure of the quality of quadrature rule.

Definition 3.1. The *degree of precision* of an integration method is the smallest integer number d such that the integration error is zero for all $u \in \mathbf{P}(d)$, i.e. for all the polynomials of degree less than or equal to d.

We observe that, without loss of generality, we can assume that the integration interval is equal to $[-1, 1]$. Indeed, using the change of variable

$$\begin{aligned} \zeta\colon [-1,1] &\to [a,b]; \\ y &\mapsto \frac{b+a}{2} + \frac{(b-a)}{2} y, \end{aligned} \tag{3.3}$$

we have

$$\int_a^b u(x)\,\mathrm{d}\,x = \int_{-1}^{1} u\big(\zeta(y)\big)\,\zeta'(y)\,\mathrm{d}\,y = \frac{b-a}{2}\int_{-1}^{1} u\circ\zeta(y)\,\mathrm{d}\,y, \tag{3.4}$$

and the right-hand side is the integral of $u \circ \zeta$ over the interval $[-1, 1]$.

3.1 The closed Newton–Cotes method

Given a set of equidistant points $-1 = x_0 < \cdots < x_n = 1$, a natural method for approximating the integral (3.1) of a function $u\colon [-1, 1] \to \mathbf{R}$ is to first construct the interpolating polynomial $\hat{u}$ at the nodes, and then calculate the exact integral of this polynomial. By construction, this method is exact for polynomials of degree up to n, and so the degree of precision is equal to *at least* n. Let $\varphi_0, \ldots, \varphi_n$ denote the Lagrange polynomials associated with the integration nodes. Then we have

$$I \approx \int_{-1}^{1} \hat{u}(x)\,\mathrm{d}\,x = \int_{-1}^{1} \sum_{i=0}^{n} u(x_i)\varphi_i(x)\,\mathrm{d}\,x = \sum_{i=0}^{n} u(x_i) \underbrace{\int_{-1}^{1} \varphi_i(x)\,\mathrm{d}\,x}_{w_i}.$$

The weights are independent of the function u, and so they can be calculated a priori. The class of integration methods obtained using this approach are known as *Newton–Cotes methods*. We present a few particular cases:

- $n = 1$, $d = 1$ (trapezoidal rule):

$$\int_{-1}^{1} u(x)\,dx \approx u(-1) + u(1). \tag{3.5}$$

- $n = 2$, $d = 3$ (Simpson's rule):

$$\int_{-1}^{1} u(x)\,dx \approx \frac{1}{3}u(-1) + \frac{4}{3}u(0) + \frac{1}{3}u(1). \tag{3.6}$$

- $n = 3$, $d = 3$ (Simpson's $\frac{3}{8}$ rule):

$$\int_{-1}^{1} u(x)\,dx \approx \frac{1}{4}u(-1) + \frac{3}{4}u(-1/3) + \frac{3}{4}u(1/3) + \frac{1}{4}u(1).$$

- $n = 4$, $d = 5$ (Boole's rule):

$$\int_{-1}^{1} u(x)\,dx \approx \frac{7}{45}u(-1) + \frac{32}{45}u\left(-\frac{1}{2}\right) + \frac{12}{45}u(0) + \frac{32}{45}u\left(\frac{1}{2}\right) + \frac{7}{45}u(1).$$

Remark 3.1. Note that, although it is based on a quadratic polynomial interpolation, Simpson's rule (3.6) has a degree of precision equal to 3. This is because any integration rule with nodes and weights symmetric around $x = 0$ is exact for odd functions, in particular x^3. Likewise, the degree of precision of Boole's rule is equal to 5.

In principle, this approach could be employed in order to construct integration rules of arbitrary high degree of precision. In practice, however, the weights become more and more imbalanced as the number of interpolation points increases, with some of them becoming negative. As a result, roundoff errors become increasingly detrimental to accuracy. In addition, in cases where the interpolating polynomial does not converge to u, for example if u is Runge's function, the approximate integral may not converge to the correct value in the limit as $n \to \infty$, even in exact arithmetic!

The integration rules presented in this section, which are based on equidistant nodes that include the endpoints of the integration interval, are called *closed Newton–Cotes* methods. A similar approach can be employed in order to construct to integration rules based on equidistant nodes that do not include the endpoints; these are called *open Newton–Cotes* methods.

3.2 Composite methods with equidistant nodes

A natural alternative to the approach presented in Section 3.1 is to construct an integration rule using piecewise polynomial interpolation, which we studied in Section 2.1.7. After partitioning the integration interval in a number of subintervals, the integral can be approximated by using one of the rules presented in Section 3.1 within each subinterval.

Composite trapezoidal rule. Let us illustrate the composite approach with an example. To this end, we introduce a partition $a = x_0 < \cdots < x_n = b$ of the interval $[a, b]$ and assume that

the nodes are equidistant with $x_{i+1} - x_i = h$. Using (3.4) with $a = x_i$ and $b = x_{i+1}$, we first generalize (3.5) to an interval $[x_i, x_{i+1}]$ as follows:

$$\int_{x_i}^{x_{i+1}} u(x)\,\mathrm{d}\,x = \frac{h}{2}\int_{-1}^{1} u\circ\zeta(y)\,\mathrm{d}\,y \approx \frac{h}{2}\Big(u\circ\zeta(-1) + u\circ\zeta(1)\Big) = \frac{h}{2}\Big(u(x_i) + u(x_{i+1})\Big),$$

where $\approx$ in this equation indicates approximation using the trapezoidal rule. Applying this approximation to each subinterval of the partition, we obtain the composite trapezoidal rule:

$$\begin{aligned}\int_a^b u(x)\,\mathrm{d}\,x &= \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} u(x)\,\mathrm{d}\,x \approx \frac{h}{2}\sum_{i=0}^{n-1}\Big(u(x_i) + u(x_{i+1})\Big)\\ &= \frac{h}{2}\Big(u(x_0) + 2u(x_1) + 2u(x_2) + \cdots + 2u(x_{n-2}) + 2u(x_{n-1}) + u(x_n)\Big). \quad (3.7)\end{aligned}$$

Like the trapezoidal rule (3.5), the composite trapezoidal rule (3.7) has a degree of precision equal to 1. However, the integration error of the method depends on the parameter h, which represents the width of each subinterval: for very small h, equation (3.7) is expected to provide a good approximation of the integral. An error estimate can be obtained directly from the formula in Theorem 2.3 for the interpolation error, provided that we assume that $u \in C^2[a,b]$.

Theorem 3.1 (Integration error for the composite trapezoidal rule)**.** *Let $\hat{I}_h$ denote the approximate integral calculated using (3.7). Then*

$$\big|I - \hat{I}_h\big| \leqslant \frac{b-a}{12} C_2 h^2, \qquad C_2 \coloneqq \sup_{\xi\in[a,b]} |u''(\xi)|. \quad (3.8)$$

Proof. Denoting by $\hat{u}_h$ the piecewise linear interpolation of u, we have

$$\int_{x_i}^{x_{i+1}} u(x) - \hat{u}(x)\,\mathrm{d}\,x = \frac{1}{2}\int_{x_i}^{x_{i+1}} u''\big(\xi(x)\big)(x - x_i)(x - x_{i+1})\,\mathrm{d}\,x.$$

Since $(x - x_i)(x - x_{i+1})$ is nonpositive over the interval $[x_i, x_{i+1}]$, we deduce that

$$\left|\int_{x_i}^{x_{i+1}} u(x) - \hat{u}(x)\,\mathrm{d}\,x\right| \leqslant \frac{1}{2}\left(\sup_{\xi\in[a,b]} |u''(\xi)|\right)\int_{x_i}^{x_{i+1}} (x - x_i)(x_{i+1} - x)\,\mathrm{d}\,x = C_2\frac{h^3}{12}.$$

Summing the contributions of all the intervals, we obtain

$$\big|I - \hat{I}_h\big| \leqslant \sum_{i=0}^{n-1}\left|\int_{x_i}^{x_{i+1}} u(x) - \hat{u}(x)\,\mathrm{d}\,x\right| \leqslant n \times C_2\frac{h^3}{12} = \frac{b-a}{12}C_2 h^2,$$

which concludes the proof. □

The integration error therefore scales as $\mathcal{O}(h^2)$. (Strictly speaking, we have shown only that the integration error admits an upper bound that scales at $\mathcal{O}(h^2)$, but it turns out that the dependence on h of this bound is optimal).

Example 3.1 (Integration error for the composite trapezoidal rule). Let us approximate the integral of x^2 over $[0,1]$ via the trapezoidal rule. In other words, let us calculate the right-hand side of

$$\int_0^1 x^2 \,\mathrm{d}\, x \approx \frac{h}{2}(x_0^2 + 2x_1^2 + \ldots + 2x_{n-1}^2 + x_n^2),$$

which we shall denote $\hat{I}_h$ hereafter. Using $x_i = ih$, and $x_n = 1$, we obtain

$$\hat{I}_h = \frac{h}{2}\left(2h^2 \sum_{i=1}^{n-1} i^2 + 1\right) = \frac{h}{2}\left(\frac{h^2}{3}(n-1)n(2n-1) + 1\right) = \frac{h}{2}\left(\frac{h^2}{3}(2n^3 - 3n^2 + n) + 1\right).$$

Now, observing that $n = 1/h$ yields

$$\hat{I}_h = \frac{h}{2}\left(\frac{h^2}{3}(2h^{-3} - 3h^{-2} + h^{-1}) + 1\right) = \frac{h}{2}\left(\frac{2}{3}h^{-1} - 1 + \frac{h}{3} + 1\right) = \frac{1}{3} + \frac{h^2}{6},$$

which is exactly the error shown by Theorem 3.1.

Composite Simpson rule. The composite Simpson rule is derived in Exercise 3.2. Given an odd number $n+1$ of equidistant points $a = x_0 < x_1 < \cdots < x_n = b$, this rule is given by

$$\hat{I}_h = \frac{h}{3}\Big(u(x_0) + 4u(x_1) + 2u(x_2) + 4u(x_3) + 2u(x_4) + \cdots + 2u(x_{n-2}) + 4u(x_{n-1}) + u(x_n)\Big). \quad (3.9)$$

This approximation is obtained by integrating the piecewise quadratic interpolant over a partition of the integration interval into $n/2$ subintervals of equal width. Obtaining an optimal error estimate, in terms of the dependence on h, for this integration formula is slightly more involved.

Theorem 3.2 (Integration error for the composite Simpson rule). *Let $\hat{I}_h$ denote the approximate integral calculated using (3.9). Then*

$$\left|I - \hat{I}_h\right| \leqslant (b-a)\frac{C_4 h^4}{180}, \qquad C_4 \coloneqq \sup_{\xi \in [a,b]} \left|u^{(4)}(\xi)\right|. \quad (3.10)$$

Proof. For a given subinterval $[x_{2i}, x_{2i+2}]$, let us denote by $\hat{u}_2$ the quadratic interpolating polynomial at $x_{2i}, x_{2i+1}, x_{2i+2}$, and by $\hat{u}_3(\bullet;\alpha)$ the cubic interpolating polynomial relative to the nodes $x_{2i}, x_{2i+1}, x_{2i+2}, \alpha$, for some $\alpha \in [x_{2i}, x_{2i+1}]$ that does not coincide with the integration nodes. We have

$$\int_{x_{2i}}^{x_{2i+2}} u(x) - \hat{u}_2(x) \,\mathrm{d}\, x = \int_{x_{2i}}^{x_{2i+2}} u(x) - \hat{u}_3(x;\alpha) \,\mathrm{d}\, x + \int_{x_{2i}}^{x_{2i+2}} \hat{u}_3(x;\alpha) - \hat{u}_2(x) \,\mathrm{d}\, x. \quad (3.11)$$

The second term on the right-hand side is zero, because the integrand is a cubic polynomial with zeros at x_{2i}, x_{2i+1} and x_{2i+2}, and because

$$\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+1})(x - x_{2i+2}) = 0.$$

By Theorem 2.3, the first term in (3.11) is bounded from above as follows:

$$\begin{aligned}\left| \int_{x_{2i}}^{x_{2i+2}} u(x) - \hat{u}_3(x;\alpha) \,\mathrm{d}\, x \right| &\leqslant \int_{x_{2i}}^{x_{2i+2}} \left| \frac{u^{(4)}\big(\xi(x)\big)}{24} (x - x_{2i})(x - x_{2i+1})(x - x_{2i+2})(x - \alpha) \right| \mathrm{d}\, x \\ &\leqslant \frac{C_4}{24} \int_{x_{2i}}^{x_{2i+2}} \big|(x - x_{2i})(x - x_{2i+1})(x - x_{2i+2})(x - \alpha)\big| \,\mathrm{d}\, x.\end{aligned}$$

This inequality is valid for any $\alpha \in A := [x_{2i}, x_{2i+2}] \backslash \{x_{2i}, x_{2i+1}, x_{2i+2}\}$. Denoting by $h(\alpha)$ the integral on the right-hand side, we observe that

$$\lim_{\alpha \to x_{2i+1}} h(\alpha) = \int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+1})^2 (x_{2i+2} - x) \,\mathrm{d}\, x = \frac{4}{15} h^5.$$

Therefore, we conclude that

$$\left| \int_{x_{2i}}^{x_{2i+2}} u(x) - \hat{u}_2(x) \,\mathrm{d}\, x \right| \leqslant \inf_{\alpha \in A} \left| \int_{x_{2i}}^{x_{2i+2}} u(x) - \hat{u}_3(x;\alpha) \,\mathrm{d}\, x \right| \leqslant \frac{C_4}{90} h^5.$$

Summing the contributions of all the subintervals, we finally obtain

$$|I - \hat{I}_h| \leqslant \frac{n}{2} \times \frac{C_4 h^5}{90} = (b - a) \frac{C_4 h^4}{180}, \tag{3.12}$$

which concludes the proof. □

Remark 3.2. The cancellation of the second term in (3.11) also follows from the fact that the degree of precision of the Simpson rule (3.6) is equal to 3, and so

$$\int_{x_{2i}}^{x_{2i+2}} \hat{u}_3(x) - \hat{u}_2(x) \,\mathrm{d}\, x = \frac{1}{3}(\hat{u}_3 - \hat{u}_2)(x_{2i}) + \frac{4}{3}(\hat{u}_3 - \hat{u}_2)(x_{2i+1}) + \frac{1}{3}(\hat{u}_3 - \hat{u}_2)(x_{2i+2}) = 0,$$

where we used the short-hand notation $\hat{u}_3(x) = \hat{u}_3(x;\alpha)$.

General composite quadrature rules. In view of (3.4), any quadrature rule with $N + 1$ integration points of the form

$$\int_{-1}^{1} u(x) \,\mathrm{d}\, x \approx \sum_{j=0}^{N} w_j u(x_j) \tag{3.13}$$

admits a composite version for an arbitrary integration interval $[a, b]$ obtained by applying the rule locally in M equally-sized subintervals:

$$\int_a^b u(x)\,\mathrm{d}\,x \approx \frac{h}{2}\sum_{i=0}^{M-1}\sum_{j=0}^{N} w_j u\left(a + ih + \frac{h}{2} + \frac{x_j h}{2}\right), \qquad h := \frac{b-a}{M}. \tag{3.14}$$

Clearly, the single-interval rule (3.13) and the composite rule (3.14) have the same degree of precision. To conclude this section, we prove a relation between this degree of precision of a rule and the convergence rate with respect to h of the composite rule (3.14).

Theorem 3.3. *Suppose that the degree of precision of the single-interval rule (3.13) is equal to d. Then for any $u \in C^{(d+1)}[a, b]$, there is $c > 0$ such that*

$$\forall M \geqslant 1, \qquad \left|I[u] - \hat{I}_h[u]\right| \leqslant ch^{d+1}, \qquad c := \frac{C_{d+1}(b-a)}{(d+1)!}\left(1 + \frac{1}{2}\sum_{i=0}^{J}|w_j|\right). \tag{3.15}$$

Here $I[u]$ and $\hat{I}_h[u]$ denote respectively the left-hand side and the right-hand side of (3.14), and C_{d+1} is defined in (3.17).

Proof. Fix $i \in \{0, \dots M-1\}$, and let p_i denote the Taylor expansion of degree d of the function u around $z_i := a + ih$:

$$p_i(x) = u(z_i) + u'(z_i)(x - z_i) + \ldots + \frac{u^{(d)}(z_i)}{d!}(x - z_i)^d. \tag{3.16}$$

By the mean-value form of the remainder, it is simple to prove that

$$\forall x \in [z_i, z_{i+1}], \qquad \left|u(x) - p_i(x)\right| = \frac{C_{d+1}}{(d+1)!}h^{d+1}, \qquad C_{d+1} = \sup_{x\in[a,b]}\left|u^{(d+1)}(x)\right|. \tag{3.17}$$

Let us define the local contributions to the total integral as follows:

$$I^{(i)}[u] := \int_{z_i}^{z_{i+1}} u(x)\,\mathrm{d}\,x, \qquad \hat{I}_h^{(i)}[u] = \frac{h}{2}\sum_{i=0}^{N} w_i u\left(z_i + \frac{h}{2} + \frac{x_i h}{2}\right),$$

so that $I[u] = \sum_{i=1}^{M-1} I^{(i)}[u]$ and $\hat{I}_h[u] = \sum_{i=1}^{M-1} \hat{I}_h^{(i)}[u]$. Since p is a polynomial of degree at most d, it follows that

$$\begin{aligned}\left|I^{(i)}[u] - \hat{I}_h^{(i)}[u]\right| &= \left|I^{(i)}[u] - I^{(i)}[p] + \hat{I}_h^{(i)}[p] - \hat{I}_h^{(i)}[u]\right| \\ &\leqslant \left|I^{(i)}[u] - I^{(i)}[p]\right| + \left|\hat{I}_h^{(i)}[u] - \hat{I}_h^{(i)}[p]\right| = \left|I^{(i)}[u-p]\right| + \left|\hat{I}_h^{(i)}[u-p]\right|,\end{aligned}$$

where we used the triangle inequality and linearity in the second line. Thus, using (3.17) we deduce that

$$\left|I^{(i)}[u] - \hat{I}_h^{(i)}[u]\right| \leqslant \frac{C_{d+1}}{(d+1)!} h^{d+1} \int_{z_i}^{z_{i+1}} 1 \,\mathrm{d}\, x + \frac{h}{2} \frac{C_{d+1}}{(d+1)!} h^{d+1} \sum_{i=0}^{J} |w_j|$$
$$= \frac{C_{d+1}}{(d+1)!} h^{d+2} \left(1 + \frac{1}{2} \sum_{i=0}^{J} |w_j|\right).$$

Summing the contributions of the local errors, we obtain

$$\left|I[u] - \hat{I}_h[u]\right| \leqslant \sum_{i=0}^{M-1} \left|I^{(i)}[u] - \hat{I}_h^{(i)}[u]\right| \leqslant M \frac{C_{d+1}}{(d+1)!} h^{d+2} \left(1 + \frac{1}{2} \sum_{i=0}^{J} |w_j|\right),$$

which leads to the result since $Mh = b - a$. □

Remark 3.3. A few comments are in order.

- The constant c in (3.15) is not sharp in general, but the power of h is optimal.
- This result may be viewed as a generalization of Theorems 3.1 and 3.2, albeit with a worse constant prefactor.
- Instead of the polynomial p_i in (3.16), we could have used any polynomial approximation of u such that (3.17) is satisfied, with possibly a different prefactor but the same power of h on the right-hand side. A natural choice, for example, would have been to define p_i by interpolation through (possibly a subset or superset of) the local integration points.
- Theorem 3.3 further motivates why the degree of precision of an integration rule is an interesting metric.

Estimating the error a posteriori. In practice, it is useful to be able to estimate the integration error so that, if the error is deemed too large, a better approximation of the integral can be calculated by using a smaller value for the step size h. Calculating the exact error $I - \hat{I}_h$ is impossible in general, because this would require to know the exact value of the integral, but it is possible to calculate a rough approximation of the error based on two numerical approximations of the integral, as we illustrate formally hereafter for the composite Simpson rule.

Suppose that $\hat{I}_{2h}$ and $\hat{I}_h$ are two approximations of the integral, calculated using the composite Simpson rule with step size $2h$ and h, respectively. If we assume that the error proportionally to $\mathcal{O}(h^4)$ as (3.12) suggests, then it holds approximately that

$$I - \hat{I}_h \approx \frac{1}{2^4}(I - \hat{I}_{2h}). \tag{3.18}$$

This implies that

$$I - \hat{I}_{2h} = (I - \hat{I}_h) + (I_h - \hat{I}_{2h}) \approx \frac{1}{16}(I - \hat{I}_{2h}) + (\hat{I}_h - \hat{I}_{2h}).$$

Rearranging this equation gives an approximation of the error for $\hat{I}_{2h}$:

$$I - \hat{I}_{2h} \approx \frac{16}{15}(\hat{I}_h - \hat{I}_{2h}).$$

Using (3.18), we can then derive an error estimate for $\hat{I}_h$:

$$|I - \hat{I}_h| \approx \frac{1}{15}|\hat{I}_h - \hat{I}_{2h}|. \tag{3.19}$$

The right-hand side can be calculated numerically, because it does not depend on the exact value of the integral. In practice, the two sides of (3.19) are often very close for small h. In the code example below, we approximate the integral

$$I = \int_0^{\frac{\pi}{2}} \cos(x)\,\mathrm{d}\,x = 1 \tag{3.20}$$

for different step sizes and compare the exact error with the approximate error obtained using (3.19). The results obtained are summarized in Table 3.1, which shows a good match between the two quantities.

Table 3.1: Comparison between the exact integration error and the approximate integration error calculated using (3.19).

h	Exact error $\lvert I - \hat{I}_h\rvert$	Approximate error $\frac{1}{15}\lvert\hat{I}_h - \hat{I}_{2h}\rvert$
2^{-4}	$5.166847063531321 \times 10^{-7}$	$5.185892840930961 \times 10^{-7}$
2^{-5}	$3.226500089326123 \times 10^{-8}$	$3.229464703065806 \times 10^{-8}$
2^{-6}	$2.0161285974040766 \times 10^{-9}$	$2.016591486390477 \times 10^{-9}$
2^{-7}	$1.2600120946615334 \times 10^{-10}$	$1.260084925291949 \times 10^{-10}$

```
# Composite Simpson's rule
function composite_simpson(u, a, b, n)
    # Integration nodes
    x = LinRange(a, b, n + 1)
    # Evaluation of u at the nodes
    ux = u.(x)
    # Step size
    h = x[2] - x[1]
    # Approximation of the integral
    return (h/3) * sum([ux[1]; ux[end]; 4ux[2:2:end-1]; 2ux[3:2:end-2]])
end
# Function to integrate
u(x) = cos(x)
# Integration bounds
a, b = 0, π/2
# Exact integral
I = 1.0
```

```
# Number of subintervals
ns = [8; 16; 32; 64; 128]
# Approximate integrals
Ī = composite_simpson.(u, a, b, ns)
# Calculate exact and approximate errors
for i in 2:length(ns)
    println("Exact error: $(I - Ī[i]), ",
            "Approx error: $((Ī[i] - Ī[i-1])/15)")
end
```

3.3 Richardson extrapolation and Romberg's method

In the previous section, we showed how the integration error could be approximated based on two approximations of the integral with different step sizes. The aim of this section is to show that, by cleverly combining two approximations $\hat{I}_h$ and $\hat{I}_{2h}$ of an integral, an approximation even better than $\hat{I}_h$ can be constructed.

This approach is based on *Richardson's extrapolation*, which is a general method for accelerating the convergence of sequences, with applications beyond numerical integration. The idea is the following: assume that $J(h)$ is an approximation with step size h of some unknown quantity $J_* = \lim_{h\to 0} J(h)$, and that we have access to evaluations of J at $h, h/2, h/4, h/8 \dots$. If J extends to a smooth function over $[0, H]$, then by Taylor expansion it holds that

$$J(\eta) = J(0) + J'(0)\eta + J''(0)\frac{\eta^2}{2} + J^{(3)}(0)\frac{\eta^3}{3!} + \cdots + J^{(k)}(0)\frac{\eta^k}{k!} + \mathcal{O}(\eta^{k+1}).$$

Elimination of the linear error term. Let us assume that $J'(0) \neq 0$, so that the leading order term after the constant $J(0)$ scales as η. Then we have

$$\begin{aligned} J(h) &= J(0) + J'(0)h + \mathcal{O}(h^2) \\ J(h/2) &= J(0) + J'(0)\frac{h}{2} + \mathcal{O}(h^2). \end{aligned}$$

We now ask the following question: can we combine linearly $J(h)$ and $J(h/2)$ in order to construct an approximation $J_1(h/2)$ of $J(0)$ with an error scaling as $\mathcal{O}(h^2)$? Employing the ansatz $J_1(h/2) = \alpha J(h) + \beta J(h/2)$, we calculate

$$J_1(h/2) = (\alpha + \beta)J(0) + J'(0)h\left(\alpha + \frac{1-\alpha}{2}\right) + \mathcal{O}(h^2). \tag{3.21}$$

Since we want this expression to approximate $J(0)$ for small h, we need to impose that $\alpha + \beta = 1$. Then, in order for the term multiplying h to cancel out, we require that

$$\alpha + \frac{1-\alpha}{2} = 0 \qquad \Leftrightarrow \qquad \alpha = -1.$$

This yields the formula

$$J_1(h/2) = 2J(h/2) - J(h). \tag{3.22}$$

Notice that, in the case where J is a linear function, $J_1(h/2)$ is exactly equal to $J(0)$. This reveals a geometric interpretation of (3.22): the approximation $J_1(h/2)$ is simply the y intercept of the straight line passing through the points $\big(h/2, J(h/2)\big)$ and $\big(h, J(h)\big)$.

Elimination of the quadratic error term. If we had tracked the coefficient of h^2 in the previous paragraph, we would have obtained instead of (3.21) the following equation:

$$J_1(h/2) = J(0) - J^{(2)}(0)\frac{h^2}{4} + \mathcal{O}(h^3).$$

Provided that we have access also to $J(h/4)$, we can also calculate

$$J_1(h/4) = 2J(h/4) - J(h/2) = J(0) - J^{(3)}(0)\frac{h^2}{16} + \mathcal{O}(h^3).$$

At this point, it is natural to wonder whether we can combine $J_1(h/2)$ and $J_1(h/4)$ in order to produce an even better approximation of $J(0)$. Applying the same reasoning as in the previous section leads us to introduce

$$J_2(h/4) = \frac{4J_1(h/4) - J_1(h/2)}{4-1} = J(0) + \mathcal{O}(h^3).$$

This is an exact approximation of $J(0)$ if J is a quadratic polynomial, indicating that $J_2(h/4)$ is simply the y intercept of the quadratic polynomial interpolating the function J through the three points $\big(h/4, J(h/4)\big)$, $\big(h/2, J(h/2)\big)$ and $\big(h, J(h)\big)$.

Elimination of higher order terms. The procedure above can be repeated in order to eliminate terms of higher and higher orders. The following schematic illustrates, for example, the calculation of an approximation $J_3(h/8) = J(0) + \mathcal{O}(h^4)$.

$$\begin{array}{ccccccc}
J(h) & & & & & & \\
 & \searrow & & & & & \\
J(h/2) & \longrightarrow & J_1(h/2) & & & & \\
 & \searrow & & \searrow & & & \\
J(h/4) & \longrightarrow & J_1(h/4) & \longrightarrow & J_2(h/4) & & \\
 & \searrow & & \searrow & & \searrow & \\
J(h/8) & \longrightarrow & J_1(h/8) & \longrightarrow & J_2(h/8) & \longrightarrow & J_3(h/8) \\
\mathcal{O}(h) & & \mathcal{O}(h^2) & & \mathcal{O}(h^3) & & \mathcal{O}(h^4).
\end{array}$$

Here, the last row indicates the scaling of the error with respect to the parameter h in the limit as $h \to 0$. The linear combination in order to calculate $J_i(h/2^i)$ is always of the form

$$J_i(h/2^i) = \frac{2^i J_{i-1}(h/2^i) - J_{i-1}(h/2^{i-1})}{2^i - 1}, \qquad J_0 = J.$$

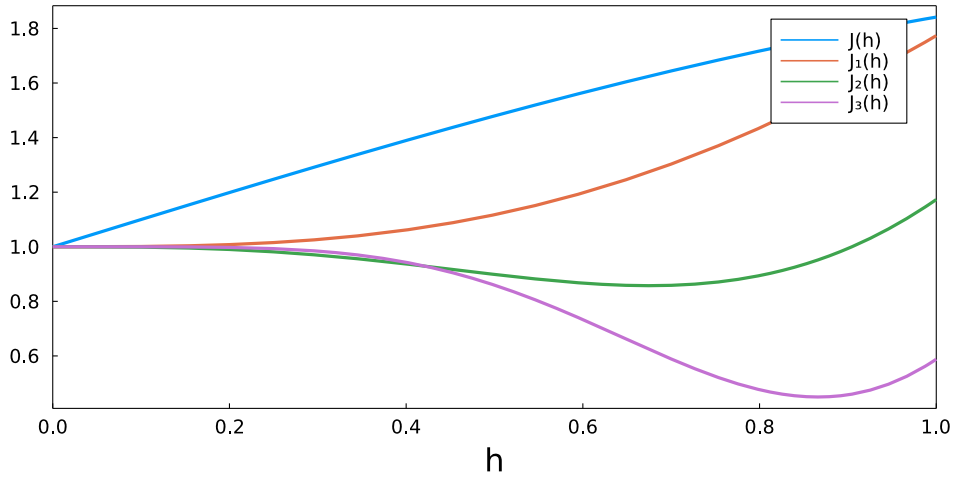


Figure 3.1: Illustration of the functions J_1, J_2 and J_3 constructed by Richardson extrapolation.

In practice we calculate the values taken by $J, J_1, J_2, \ldots$ at specific values of h, but these are in fact functions of h. In Figure 3.1, we plot these functions when $J(h) = 1 + \sin(h)$. It appears clearly from the figure that, for sufficiently small h, $J_3(h)$ provides the most precise approximation of $J(0) = 1$. Constructing the functions in Julia can be achieved in just a few lines of code.

```
J(h) = 1 + sin(h)
J_1(h) = 2J(h) - J(2h)
J_2(h) = (4J_1(h) - J_1(2h))/3
J_3(h) = (8J_2(h) - J_2(2h))/7
```

Generalization. Sometimes, it is known a priori that the Taylor development of the function J around zero contains only even powers of h. In this case, the Richardson extrapolation procedure can be slightly modified in order to produce approximations with errors scaling as $\mathcal{O}(h^4)$, then $\mathcal{O}(h^6)$, then $\mathcal{O}(h^8)$, etc. This procedure is illustrated below:

$$
\begin{array}{ccccccc}
J(h) & & & & & & \\
 & \searrow & & & & & \\
J(h/2) & \longrightarrow & J_1(h/2) & & & & \\
 & \searrow & & \searrow & & & \\
J(h/4) & \longrightarrow & J_1(h/4) & \longrightarrow & J_2(h/4) & & \\
 & \searrow & & \searrow & & \searrow & \\
J(h/8) & \longrightarrow & J_1(h/8) & \longrightarrow & J_2(h/8) & \longrightarrow & J_3(h/8) \\
\mathcal{O}(h^2) & & \mathcal{O}(h^4) & & \mathcal{O}(h^6) & & \mathcal{O}(h^8).
\end{array}
$$

This time, the linear combinations required for populating this table are given by

$$
J_i(h/2^i) = \frac{2^{2i} J_{i-1}(h/2^i) - J_{i-1}(h/2^{i-1})}{2^{2i} - 1}. \tag{3.23}
$$

Application to integration: Romberg's method Romberg's integration method consists of applying Richardson's extrapolation to the function

$$J(h) = \hat{I}_h = u(x_0) + 2u(x_1) + 2u(x_2) + \cdots + 2u(x_{n-1}) + 2u(x_n), \qquad h \in \left\{ \frac{b-a}{n} : n \in \mathbf{N} \right\}.$$

where $a = x_0 < x_1 < \cdots < x_n = b$ are equidistant nodes. The right-hand side of this equation is simply the composite trapezoidal rule with step size h. It is possible to show that $J(h)$ may be expanded as follows:

$$\forall k \in \mathbf{N}, \qquad J(h) = I + \alpha_1 h^2 + \alpha_2 h^4 + \cdots + \alpha_k h^{2k} + \mathcal{O}(h^{2k+2}). \tag{3.24}$$

This is the content of the following result.

Lemma 3.4. *Let $J(h)$ denote the approximation of the integral $I[u]$ by the composite trapezium rule, and assume that $u \in C^\infty[a,b]$. Then $J(h)$ may be expanded as in (3.24).*

Proof. Using the same notation as in Theorem 3.3, we introduce

$$I^{(i)}[u] = \int_{x_i}^{x_{i+1}} u(x)\,\mathrm{d}\,x, \qquad \hat{I}_h^{(i)}[u] = \frac{h}{2}\Big(u(x_i) + u(x_{i+1})\Big), \qquad i = 0, \ldots, n-1.$$

Fix $i \in \{0, \ldots, n-1\}$ and let p_i denote the Taylor expansion of u around $x_{i+\frac{1}{2}} := x_i + \frac{h}{2}$, of degree $2k+1$:

$$p_i(x) = u\Big(x_{i+\frac{1}{2}}\Big) + u'\Big(x_{i+\frac{1}{2}}\Big)\Big(x - x_{i+\frac{1}{2}}\Big) + \cdots + \frac{u^{(2k+1)}\Big(x_{i+\frac{1}{2}}\Big)}{(2k+1)!}\Big(x - x_{i+\frac{1}{2}}\Big)^{2k+1}.$$

From the mean-value form of the remainder, it is clear that

$$I^{(i)}[u] = I^{(i)}[p_i] + \mathcal{O}(h^{2k+3}), \qquad \hat{I}_h^{(i)}[u] = \hat{I}_h^{(i)}[p_i] + \mathcal{O}(h^{2k+3}).$$

Substituting p_i and noting that odd powers cancel when integrating, we have that

$$\begin{aligned} I^{(i)}[p_i] &= u\Big(x_{i+\frac{1}{2}}\Big)h + \omega_2 h^3 + \omega_4 h^5 + \cdots + \omega_{2k} h^{2k+1}, \\ \hat{I}_h^{(i)}[p_i] &= u\Big(x_{i+\frac{1}{2}}\Big)h + \eta_2 h^3 + \eta_4 h^5 + \cdots + \eta_{2k} h^{2k+1}, \end{aligned}$$

for appropriate coefficients. Thus we obtain that

$$\begin{aligned} I[u] - \hat{I}_h[u] &= \sum_{i=0}^{n} I^{(i)}[p_i] - I_h^{(i)}[p_i] \\ &= (\omega_2 - \eta_2)h^2 + (\omega_2 - \eta_2)h^4 + \ldots + (\omega_{2k} - \eta_{2k})h^{2k} + \mathcal{O}(h^{2k+2}), \end{aligned}$$

which concludes the proof. □

Richardson's extrapolation (3.23) can therefore be employed in order to compute approximations of the integral with increasing accuracy. The convergence of Romberg's method for calculating the integral (3.20) is illustrated in Figure 3.2.

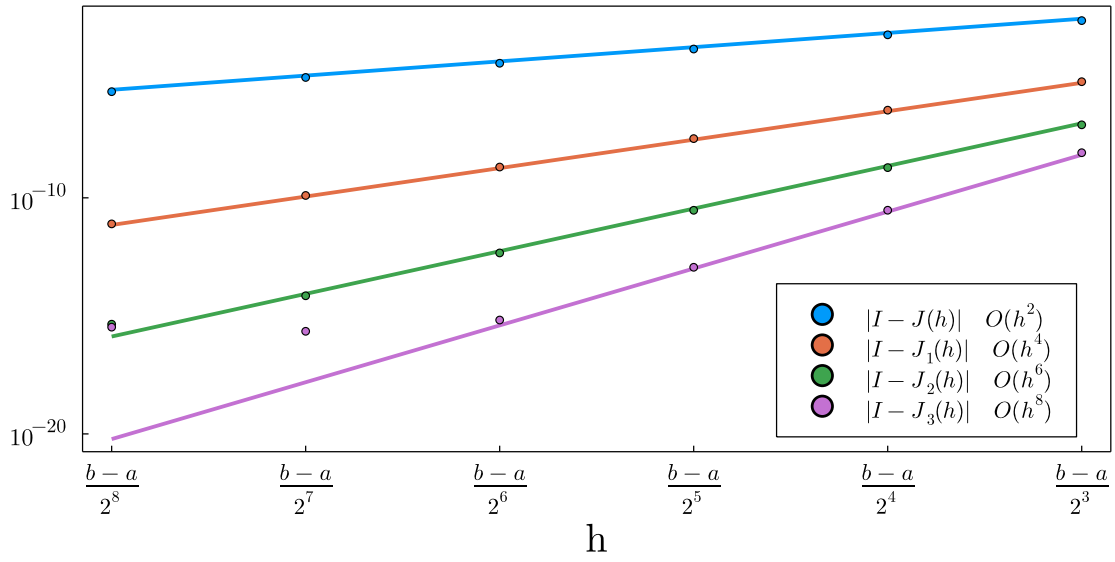


Figure 3.2: Convergence of Romberg's method. The straight lines correspond to the monomial functions $f(h) = C_i h^i$, with $i = 2, 4, 6, 8$ and for appropriate constants C_i. We observe a good agreement between the observed and theoretical convergence rates.

3.4 Methods with non-equidistant nodes

The Newton–Cotes method relies on equidistant integration nodes, and the only degrees of freedom are the integration weights. If the nodes are not fixed, then additional degrees of freedom are available, and these can be leveraged in order to construct a better integration formula. The total number of degrees of freedom for a general integration rule of the form (3.2) is $2n + 2$ which, in principle, should enable to construct an integration rule with a degree of precision equal to $2n + 1$.

A necessary condition for an integration rule of the form (3.2) to have a degree of precision equal to $2n + 1$ is that it integrates exactly all the monomials of degree 0 to $2n + 1$. This condition is also sufficient because, assuming that it is satisfied, we have by linearity of the functionals I and $\hat{I}$ that

$$\begin{aligned}\hat{I}(\alpha_0 + \alpha_1 x + \cdots + \alpha_{2n+1} x^{2n+1}) &= \alpha_0 \hat{I}(1) + \alpha_1 \hat{I}(x) + \cdots + \alpha_{2n+1} \hat{I}(x^{2n+1}) \\ &= \alpha_0 I(1) + \alpha_1 I(x) + \cdots + \alpha_{2n+1} I(x^{2n+1}) \\ &= I(\alpha_0 + \alpha_1 x + \cdots + \alpha_{2n+1} x^{2n+1}),\end{aligned}$$

Here $I(u)$ and $\hat{I}(u)$ denote respectively the exact integral of u and its approximate integral using (3.2). In order to find the nodes and weights of the integration rule, we can therefore solve the following nonlinear system of $2n + 2$ equations with $2n + 2$ unknowns:

$$\sum_{i=0}^{n} w_i x_i^d = \int_{-1}^{1} x^d \,\mathrm{d}\, x, \qquad d = 0, \ldots, 2n+1. \tag{3.25}$$

The quadrature rule thus obtained is called the *Gauss–Legendre quadrature*.

Example 3.2. Let us derive the Gauss–Legendre quadrature with $n+1 = 2$ nodes. The system of equations that we need to solve in this case is the following:

$$w_0 + w_1 = 2, \qquad w_0 x_0 + w_1 x_1 = 0, \qquad w_0 x_0^2 + w_1 x_1^2 = \frac{2}{3}, \qquad w_0 x_0^3 + w_1 x_1^3 = 0.$$

The solution to these equations is given by

$$-x_0 = x_1 = \frac{\sqrt{3}}{3}, \qquad w_0 = w_1 = 1.$$

Connection with orthogonal polynomials. Let $(L_n)_{n\in\mathbf{N}}$ denote the Legendre polynomials, i.e. the orthogonal polynomials with for the inner product

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x) \,\mathrm{d}\, x.$$

The nodes and weights of the Gauss–Legendre quadrature rules can be obtained constructively from Legendre polynomials, as shown in Section 2.2.4. We shall now demonstrate this connection in much more direct manner. Specifically, we prove that the integration nodes are given by the roots of a Legendre polynomial.

Theorem 3.5. *For every $n \in \mathbf{N}$, there exists a unique solution to the system of equations (3.25). The nodes $(x_i)_{i\in\{0,\ldots,n\}}$ are the roots of L_{n+1} and the weights are given by*

$$w_i = \int_{-1}^{1} \ell_i(x) \,\mathrm{d}\, x, \qquad i \in \{1, \ldots, n\}, \tag{3.26}$$

where ℓ_i is the Lagrange polynomial

$$\frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}.$$

In addition, the weights are all positive.

Proof. We show first that the solution to (3.25) exists, then that it is unique, and finally that the weights are positive.

Existence. We begin by showing that, if $(x_i)_{i\in\{0,\ldots n\}}$ are the roots of L_{n+1} and the weights are defined from (3.26), then the equations (3.25) are satisfied. To this end, it is sufficient to show that for all $p \in \mathbf{P}(2n+1)$,

$$\int_{-1}^{1} p(x)\,\mathrm{d}\,x = \sum_{i=0}^{n} w_i p(x_i). \tag{3.27}$$

Take $p \in \mathbf{P}(2n+1)$ and let $q \in \mathbf{P}(n)$ and $r \in \mathbf{P}(n)$ be the polynomials such that

$$p(x) = q(x)L_{n+1}(x) + r(x).$$

The quotient q and the remainder r can be obtained by Euclidean division of p by L_{n+1}. Since L_{n+1} is orthogonal to any polynomial in $\mathbf{P}(n)$, in particular q, and the nodes $(x_i)_{i\in\{0,\dots n\}}$ are the roots of L_{n+1}, it holds that

$$\int_{-1}^{1} p(x)\,\mathrm{d}\,x - \sum_{i=0}^{n} w_i p(x_i) = \int_{-1}^{1} r(x)\,\mathrm{d}\,x - \sum_{i=0}^{n} w_i r(x_i). \tag{3.28}$$

Given that $r \in \mathbf{P}(n)$, the remainder r must coincides with its polynomial interpolation at the points $x_0, \dots x_m$. Therefore,

$$r = r(x_0)\ell_0 + \cdots + r(x_n)\ell_n,$$

and so

$$\int_{-1}^{1} r(x)\,\mathrm{d}\,x = \sum_{i=0}^{n} r(x_i) \int_{-1}^{1} \ell_i(x)\,\mathrm{d}\,x = \sum_{i=0}^{n} r(x_i) w_i,$$

where we used (3.26) in the last equality. Consequently, the right-hand side of (3.28) is zero, and since p was arbitrary this implies that (3.27) is satisfied.

Uniqueness. Next, we show that the nodes are necessarily the roots of L_{n+1}. To this end, assume that $x_0, \dots, x_n$ and weights $w_0, \dots, w_n$ are such that the equations (3.25) are satisfied, and let

$$q(x) = (x - x_0)\dots(x - x_n).$$

Our goal is to show that $q(x)$ coincides with L_{n+1} up to a constant factor. In order to show this, it is sufficient to prove that $q(x)$ is orthogonal to x^d for all $d = 0, \dots, n$, because the only polynomial in $\mathbf{P}(n+1)$ that satisfies these orthogonality relations is the Legendre polynomial L_{n+1}, or a multiple thereof. For the values of d considered, the polynomial $q(x)x^d$ belongs to $\mathbf{P}(2n+1)$. Given that the integration rule with nodes $x_0, \dots, x_n$ and weights $w_0, \dots, w_n$ is exact for all the elements of $\mathbf{P}(2n+1)$ by assumption, we deduce that

$$\forall d \in \{0, \dots, n\}, \qquad \int_{-1}^{1} q(x)x^d\,\mathrm{d}\,x = \sum_{i=0}^{n} w_i q(x_i) x_i^d = 0.$$

Finally, we show that the weights are necessarily given by (3.26). Since the integration rule must be exact for any Lagrange polynomial ℓ_j, we have

$$\int_{-1}^{1} \ell_j(x) \,\mathrm{d}\, x = \sum_{i=0}^{n} w_i \ell_j(x_i) = w_j,$$

which concludes the proof of uniqueness.

Positivity of the weights. Since the integration rule is exact for all the polynomials in $\mathbf{P}(2n+1)$ and $\ell_j(x)^2 \in \mathbf{P}(2n+1)$, we deduce that

$$\int_{-1}^{-1} \left|\ell_j(x)\right|^2 \,\mathrm{d}\, x = \sum_{i=0}^{n} w_i \left|\ell_j(x_i)\right|^2 = w_j.$$

The left-hand side is positive, and so w_j must also be positive.

□

Since the integration weights are all positive, the Gauss–Legendre quadrature rules are less susceptible to roundoff errors than the Newton–Cotes methods. In addition, we have the following result.

Theorem 3.6. *Assume that $u \in C([-1,1])$, and let $\hat{I}_n(u)$ denote the approximation of $I(u)$ using the Gauss–Legendre quadrature. Then $\hat{I}_n(u) \to I(u)$ in the limit as $n \to \infty$.*

Proof. The positivity of the weights is crucial for the proof. By the Weierstrass approximation theorem, for all $\varepsilon > 0$ there exists polynomial p such that

$$E \coloneqq \max_{x \in [-1,1]} \left|p(x) - u(x)\right| \leqslant \varepsilon.$$

Since the degree of precision of the Gauss–Legendre quadrature with $n+1$ integration nodes is $2n+1$, there exists N sufficiently large such that $\hat{I}_n(p) = I(p)$ for all $n \geqslant N$. Thus

$$\begin{aligned} \forall n \geqslant N, \qquad \left|\hat{I}_n(u) - I(u)\right| &\leqslant \left|\hat{I}_n(u) - \hat{I}_n(p)\right| + \left|\hat{I}_n(p) - I(p)\right| + \left|I(p) - I(u)\right| \\ &= \left|\hat{I}_n(u) - \hat{I}_n(p)\right| + \left|I(p) - I(u)\right| \\ &\leqslant \sum_{i=0}^{n} |w_i| \; E + \int_{-1}^{1} E \,\mathrm{d}\, x = 2\varepsilon + 2\varepsilon = 4\varepsilon. \end{aligned}$$

Indeed, all the weights are positive and the Gauss–Legendre quadrature rule is exact for the constant function $u(x) = 1$, which implies that the weights add up to 2. Since $\varepsilon > 0$ was arbitrary, this concludes the proof. □

Computation of the integration nodes. We proved that the integration nodes are given by the roots of the Legendre polynomials. In practice, calculating these roots can be achieved by calculating the eigenvalues of a tridiagonal matrix, see Section 2.2.4. This is known as the Golub–Welsch algorithm.

Generalization to higher dimensions. Gauss–Legendre integration is ubiquitous in numerical methods for partial differential equations, in particular the *finite element method*. Its generalization to higher dimensions is immediate: for a function u: $[-1,1] \times [-1,1] \to \mathbf{R}$, we have

$$\int_0^1 \int_0^1 u(x,y) \,\mathrm{d}\, y \,\mathrm{d}\, x \approx \sum_{i=0}^{n} \sum_{j=0}^{n} w_i w_j u(x_i, y_i).$$

The degree of precision of this integration rule is the same as that of the corresponding one-dimensional rule, and this approach can be generalized to any dimension d. The associated computational cost, however, scales as n^d, and so it is not a good idea to use a deterministic method of this type in the high dimensional setting. The explosion of the computational cost as the dimension increases is known as *the curse of dimensionality*.

3.5 Introduction to probabilistic integration methods

So far in this chapter, we covered only *deterministic* integration formulas. Much of the research around the calculation of high-dimensional integrals today is concerned with *probabilistic* integration methods using probabilistic approaches. These methods are based on the connection between integrals and expectations. To illustrate the simplest probabilistic approach, called the *Monte Carlo method*, we consider the problem of approximating the integral

$$I = \int_0^1 u(x) \,\mathrm{d}\, x.$$

This integral may be expressed as the expectation $\mathbf{E}\left[u(X)\right]$, where $\mathbf{E}$ is the expectation operator and $X \sim \mathcal{U}(0,1)$ is a uniformly distributed random variable over the interval $[0,1]$. Therefore, in practice, the integral I may be approximated by generating a large number of *independent* samples $X_1, X_2, \ldots$ drawn from the distribution $\mathcal{U}(0,1)$ and averaging $f(X_i)$ over all of these samples:

$$\hat{I}_N = \frac{1}{N} \sum_{n=1}^{N} u(X_n).$$

The quantity $\hat{I}_N$, where the subscript N denotes the number of samples employed, is itself a random variable; it is called an *estimator* of the exact integral I. In Julia, the calculation of the integral using the Monte Carlo method can be achieved with the following code.

```
N = 1000
u(x) = x^2
X = rand(N)
Î = (1/N) * sum(u.(X))
```

Since the expectation operator is linear, we calculate that

$$\mathbf{E}\,[\hat{I}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbf{E}\,\Big[u(X_n)\Big] = \frac{1}{N}\sum_{n=1}^{N}\int_0^1 u(x)\,\mathrm{d}\,x = \frac{1}{N}\sum_{n=1}^{N} I = I.$$

The estimator $\hat{I}_N$ is therefore said to be *unbiased*, because the bias $\mathbf{E}\,[\hat{I}_N] - I$ is zero. Now assume that the function u is square integrable over the interval $[0,1]$ and let

$$\sigma^2 := \mathbf{V}\,\Big[u(X)\Big] := \mathbf{E}\,\left[\Big(u(X) - \mathbf{E}\,\Big[u(X)\Big]\Big)^2\right] = \int_0^1 |u(x) - I|^2\,\mathrm{d}\,x,$$

where $\mathbf{V}\,[X]$ denotes the variance of the random variable X. We can calculate explicitly the variance of the estimator $\hat{I}_N$:

$$\mathbf{V}\,[\hat{I}_N] = \mathbf{E}\,\left[\left(\frac{1}{N}\sum_{n=1}^{N} u(X_n) - I\right)^2\right] = \mathbf{E}\,\left[\frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\Big(u(X_n) - I\Big)\Big(u(X_m) - I\Big)\right].$$

Since the samples $X_1, X_2, \ldots$ are independent, it holds that

$$\mathbf{E}\,\Big[\Big(u(X_n) - I\Big)\Big(u(X_m) - I\Big)\Big] = \delta_{nm}\sigma^2,$$

where δ_{mn} is the Kronecker delta. Consequently, we deduce that

$$\mathbf{V}\,[\hat{I}_N] = \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\mathbf{E}\,\Big[\Big(u(X_n) - I\Big)\Big(u(X_m) - I\Big)\Big] = \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\delta_{mn}\sigma^2 = \frac{\sigma^2}{N}.$$

Therefore, the variance of $\hat{I}_N$ decreases as $1/N$ when the number of samples N increases, indicating that the estimator becomes increasingly accurate. To state this more precisely, we will use Chebyshev's inequality.

Theorem 3.7 (Chebyshev's inequality)**.** *Let Z be a random variable with mean m and variance s^2. Then for any real $k > 0$,*

$$\mathbf{P}\,\Big[\big|Z - m\big| \geqslant ks\Big] \leqslant \frac{1}{k^2}.$$

Let $\varepsilon > 0$. Employing Chebyshev's inequality with $k = \varepsilon/\sqrt{\mathbf{V}\,[\hat{I}_N]}$, we obtain

$$\mathbf{P}\,\Big[\big|\hat{I}_N - I\big| \geqslant \varepsilon\Big] \leqslant \frac{\mathbf{V}\,[\hat{I}_N]}{\varepsilon^2} = \frac{\sigma^2}{N\varepsilon^2}. \tag{3.29}$$

Consequently, for any $\varepsilon > 0$, the probability that the integration error $\big|\hat{I}_N - I\big|$ is greater than or equal to ε tends to zero in the limit as $N \to \infty$. In probabilistic jargon, we say that $\hat{I}_N$ converges *in probability* to I. Equation (3.29) can also be employed in order to construct a *confidence interval* for the exact integral, as we demonstrate in the following paragraph.

Construction of a $(1-\alpha)$ confidence interval. By definition, a $(1-\alpha)$ confidence interval for I is an interval $[z_1, z_2]$, the endpoints of which being random variables, such that the

probability that $I \in [z_1, z_2]$ is greater than or equal to $1 - \alpha$. To construct such an interval, we begin by finding ε such that the right-hand side of (3.29) is equal to α, which gives $\varepsilon = \sqrt{\sigma^2/N\alpha}$. For this value of ε, we have

$$\mathbf{P}\left[\left[\hat{I}_N - I\right| \geqslant \varepsilon\right] \leqslant \alpha.$$

Since $|\hat{I}_N - I| \geqslant \varepsilon$ if and only if $I \notin (\hat{I}_N - \varepsilon, \hat{I}_N + \varepsilon)$, we conclude that

$$\mathbf{P}\left[I \in \left(\hat{I}_N - \varepsilon, \hat{I}_N + \varepsilon\right)\right] \geqslant 1 - \alpha.$$

We have thus shown that

$$\left(\hat{I}_N - \sqrt{\frac{\sigma^2}{N\alpha}}, \hat{I}_N + \sqrt{\frac{\sigma^2}{N\alpha}}\right)$$

is a $(1 - \alpha)$ confidence interval for I.

3.6 Exercises

Exercise 3.1. *Derive the Simpson's integration rule (3.6).*

Exercise 3.2. *Derive the composite Simpson integration rule (3.9).*

Exercise 3.3. *Consider the integration rule*

$$\int_0^1 u(x)\,\mathrm{d}\,x \approx w_1 u(0) + w_2 u(1) + w_3 u'(0).$$

Find w_1, w_2 and w_3 so that this integration rule has the highest possible degree of precision.

Exercise 3.4. *Consider the integration rule*

$$\int_{-1}^1 u(x)\,\mathrm{d}\,x \approx w_1 u(x_1) + w_2 u'(x_1).$$

Find w_1, w_2 and x_1 so that this integration rule has the highest possible degree of precision.

Exercise 3.5. *What is the degree of precision of the following quadrature rule?*

$$\int_{-1}^1 u(x)\,\mathrm{d}\,x \approx \frac{2}{3}\left(2u\left(-\frac{1}{2}\right) - u(0) + 2u\left(\frac{1}{2}\right)\right).$$

Exercise 3.6. *The Gauss–Hermite quadrature rule with $n + 1$ nodes is an approximation of the form*

$$\int_{-\infty}^{\infty} u(x)\,\mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}\,x \approx \sum_{i=0}^{n} w_i u(x_i),$$

such that the rule is exact for all polynomials of degree less than or equal to $2n+1$*. Find the Gauss–Hermite rule with two nodes.*

Exercise 3.7. *Use Romberg's method to construct an integration rule with an error term scaling as* $\mathcal{O}(h^4)$*. Is there a link between the method you obtained and another integration rule seen in class?*

Exercise 3.8 (Improving the error bound for the composite trapezoidal rule)**.** *The notation used in this exercise is the same as in Section 3.2. In particular,* $\hat{I}_h$ *denotes the approximate integral obtained by using the composite trapezoidal rule (3.7), and* $\hat{u}_h$ *is the corresponding piecewise linear interpolant.*

A version of the mean value theorem states that, if $g\colon [a,b] \to \mathbf{R}$ *is a non-negative integrable function and* $f\colon [a,b] \to \mathbf{R}$ *is continuous, then there exists* $\xi \in (a,b)$ *such that*

$$\int_a^b f(x)g(x)\,\mathrm{d}\,x = f(\xi)\int_a^b g(x)\,\mathrm{d}\,x. \tag{3.30}$$

- *Using (3.30), show that, for all* $i \in \{0, \dots, n-1\}$*, there exists* $\xi_i \in (x_i, x_{i+1})$ *such that*

$$\int_{x_i}^{x_{i+1}} u(x) - \hat{u}_h(x)\,\mathrm{d}\,x = -u''(\xi_i)\frac{h^3}{12}.$$

- *Prove, by using the intermediate value theorem, that if* $f\colon [a,b] \to \mathbf{R}$ *is a continuous function, then for any set* $\xi_0, \dots, \xi_{n-1}$ *of points within the interval* (a,b)*, there exists* $c \in (a,b)$ *such that*

$$\frac{1}{n}\sum_{i=0}^{n-1} f(\xi_i) = f(c).$$

- *Combining the previous items, conclude that there exists* $\xi \in (a,b)$ *such that*

$$I - \hat{I}_h = -u''(\xi)(b-a)\frac{h^2}{12},$$

which is a more precise expression of the error than that obtained in (3.8).

Remark 3.4. One may convince oneself of (3.30) by rewriting this equation as

$$\frac{\int_a^b f(x)g(x)\,\mathrm{d}\,x}{\int_a^b g(x)\,\mathrm{d}\,x} = f(\xi).$$

The left-hand side is the average of $f(x)$ with respect to the probability measure with density given by

$$x \mapsto \frac{g(x)}{\int_a^b g(x)\,\mathrm{d}\,x}.$$

Exercise 3.9 (From the final exam of Spring 2022)**.** *Construct an integration rule of the form*

$$\int_{-1}^{1} u(x)\,\mathrm{d}\,x \approx w_1 u\Big(-\frac{1}{2}\Big) + w_2 u(0) + w_3 u\Big(\frac{1}{2}\Big)$$

with a degree of precision equal to at least 2. What is the degree of precision of the rule constructed?

Solution. The Lagrange polynomials associated with $-1/2$, 0 and $1/2$ are respectively

$$\begin{aligned} p_1(x) &= 2x\Big(x - \frac{1}{2}\Big), \\ p_2(x) &= -4\Big(x + \frac{1}{2}\Big)\Big(x - \frac{1}{2}\Big), \\ p_3(x) &= 2\Big(x + \frac{1}{2}\Big)x. \end{aligned}$$

We deduce that

$$w_1 = \int_{-1}^{1} p_1(x) = \frac{4}{3}, \qquad w_2 = \int_{-1}^{1} p_2(x) = -\frac{2}{3}, \qquad w_3 = \int_{-1}^{1} p_3(x) = \frac{4}{3}.$$

By construction, the degree of precision is at least 2. However, the integration rule is exact also when $u(x) = x^3$. Since the rule is not exact for $u(x) = x^4$, we conclude that the degree of precision is 3. △

Exercise 3.10 (From the final exam of Spring 2022)**.** *The Gauss–Laguerre quadrature rule with n nodes is an approximation of the form*

$$\int_0^{\infty} u(x)\,\mathrm{e}^{-x}\,\mathrm{d}\,x \approx \sum_{i=1}^{n} w_i u(x_i),$$

such that the rule is exact when u is a polynomial of degree less than or equal to $2n-1$.

- *Find the Gauss–Laguerre rule with one node ($n = 1$).*
- *Find the Gauss–Laguerre quadrature rule with two nodes ($n = 2$). You may find it useful to first calculate the Laguerre polynomial of degree 2.*

Solution. Below are the derivations of the Gauss–Laguerre rules with 1 and 2 nodes.

Gauss–Laguerre rule with 1 node. We are looking for w_1 and x_1 such that

$$\forall (a, b) \in \mathbf{R}^2, \qquad \int_0^{\infty} (a + bx)\,\mathrm{e}^{-x}\,\mathrm{d}\,x = w_1(a + bx_1).$$

The left-hand side is equal to

$$a\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}\,x + b\int_0^\infty x\mathrm{e}^{-x}\,\mathrm{d}\,x = a + b\int_0^\infty x\mathrm{e}^{-x}\,\mathrm{d}\,x.$$

Using integration by parts, we find the value of the remaining integral on the right-hand side:

$$\begin{aligned}\int_0^\infty x\mathrm{e}^{-x} &= \int_0^\infty -(x\mathrm{e}^{-x})' + \mathrm{e}^{-x}\,\mathrm{d}\,x \\ &= -(x\mathrm{e}^{-x})\Big|_{x=\infty} + (x\mathrm{e}^{-x})\Big|_{x=0} + \int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}\,x \\ &= 0 + 0 + 1.\end{aligned}$$

(To be fully rigorous, we would need to write the first term on the second line as a limit.) Therefore, we obtain

$$a + b = w_1(a + bx_1),$$

which implies that $w_1 = x_1 = 1$.

Gauss–Laguerre rule with 2 nodes. The integration nodes are given by the roots of the Laguerre polynomials, which are the orthogonal polynomials for the inner product

$$\langle f, g\rangle \coloneqq \int_0^\infty f(x)g(x)\,\mathrm{e}^{-x}\,\mathrm{d}\,x.$$

The first polynomial is $\ell_0(x) = 1$. It is simple to check that the only linear monic polynomial orthogonal to ℓ_0 is given by $\ell_1(x) = x - 1$. Next, by integration by parts we calculate that

$$\int_0^\infty x^2\,\mathrm{e}^{-x}\,\mathrm{d}\,x = \int_0^\infty -(x^2\mathrm{e}^{-x})' + 2x\mathrm{e}^{-x}\,\mathrm{d}\,x = 2.$$

and, similarly,

$$\int_0^\infty x^3\,\mathrm{e}^{-x}\,\mathrm{d}\,x = \int_0^\infty -(x^3\mathrm{e}^{-x})' + 3x^2\mathrm{e}^{-x}\,\mathrm{d}\,x = 6.$$

Consider the ansatz $\ell_2(x) = x^2 + a\ell_1(x) + b$. In order for ℓ_2 to be orthogonal to ℓ_0 and ℓ_1, it is necessary that

$$\begin{aligned}0 &= \int_0^\infty \ell_2(x)\,\ell_0(x)\,\mathrm{e}^{-x}\,\mathrm{d}\,x = 2 + b, \\ 0 &= \int_0^\infty \ell_2(x)\,\ell_1(x)\,\mathrm{e}^{-x}\,\mathrm{d}\,x = 4 + a\int_0^\infty \ell_1(x)\ell_1(x)\;\,dx = 4 + a.\end{aligned}$$

Therefore, we conclude that $a = -4$ and $b = -2$, which gives

$$\ell_2(x) = x^2 - 4x + 2.$$

The roots are given by $2 \pm \sqrt{2}$, so we have $x_1 = 2 - \sqrt{2}$ and $x_2 = 2 + \sqrt{2}$. It remains to find the weights. To this end, we need only two additional equations; it is sufficient to require that, for any $(a, b) \in \mathbf{R}^2$,

$$a+b=\int_0^\infty (a+bx)\,\mathrm{e}^{-x}\,\mathrm{d}\,x = w_1(a+bx_1)+w_2(a+bx_2)$$
$$= a(w_1+w_2)+2b(w_1+w_2)+\sqrt{2}b(w_2-w_1).$$

Letting $a=1$ and $b=0$, we obtain $w_1+w_2=1$. Then, letting $a=0$ and $b=1$, we deduce

$$1=2+\sqrt{2}(w_2-w_1) \qquad \Leftrightarrow w_2-w_1=-\frac{\sqrt{2}}{2}.$$

Therefore

$$w_1=\frac{2+\sqrt{2}}{4}, \qquad w_2=\frac{2-\sqrt{2}}{4},$$

which concludes the exercise.

△

Exercise 3.11 (Calculating the volume of hyperballs)**.** *Let B_d denote the d-dimensional unit ball for the Euclidean norm:*

$$B_d=\Big\{\boldsymbol{x}\in\mathbf{R}^d : \|\boldsymbol{x}\|\leqslant 1\Big\}.$$

The volume of B_d is defined as the integral of the characteristic function over B_d:

$$\mathrm{vol}(B_d)=\underbrace{\int_{\mathbf{R}}\dots\int_{\mathbf{R}}}_{d\text{ times}}\chi(\boldsymbol{x})\,\mathrm{d}\,x_1\dots\,\mathrm{d}\,x_d, \qquad \chi(\boldsymbol{x}):=\begin{cases}1 \text{ if } \boldsymbol{x}\in B_d\\ 0 \text{ otherwise.}\end{cases}$$

Complete the following tasks:

- *Write a function*

```
function hyperball_volume(dim, n)
    # Your code comes here
    return vol, σ
end
```

  that calculates the volume of the unit ball B_d with $d=$ `dim` *using a Monte Carlo approach with* `n` *samples drawn from an appropriate distribution. Your function should return an estimation of the volume together with the standard deviation of the estimator (which you should estimate from the samples).*

- *Using the function* `hyperball_volume`*, plot the volumes for d going from 1 to 15, together with a 99% confidence interval. See Figure 3.3 for an example solution with $n=10^7$.*

You are allowed to use your knowledge of the fact that $\mathrm{vol}(B_2)=\pi$ and $\mathrm{vol}(B_3)=4\pi/3$, but do not use the general formula for the volume of B_d.

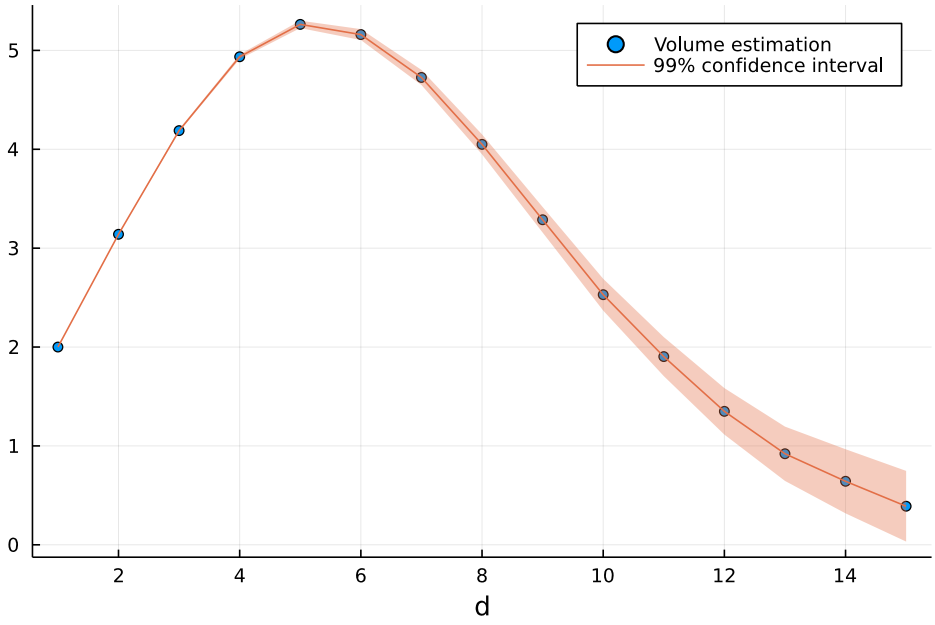


Figure 3.3: Example solution for Exercise 3.11.

Exercise 3.12. *Complete the following tasks:*

- *Write a function* `legendre(n)` *which returns the Legendre polynomial of degree* n*. To this end, you may use the* `Polynomials` *library and Rodrigues' formula:*

$$L_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}\, x^n} (x^2 - 1)^n.$$

- *Write a function* `get_nodes_and_weights(n)` *which returns the nodes and weights of the Gauss–Legendre quadrature with* n *nodes. In order to construct Lagrange polynomials, you may find it useful to use the* `fromroots` *functions.*

- *Write a function* `composite_gauss_legendre(u, a, b, n, N)`*, which returns an approximation of the integral*

$$\int_a^b u(x) \,\mathrm{d}\, x$$

obtained by partitioning the integration interval $[a, b]$ *into* N *cells, and applying the Gauss–Legendre quadrature within each cell.*

- *Take* $u(x) = \cos(x)$*,* $a = -1$ *and* $b = 1$*. Illustrate on the same plot the error for the values* $n \in \{1, 2, 3\}$ *and* N *varying from 1 to 40. For each value of* n*, estimate the order of convergence with respect to* N*, i.e. find* $\alpha(n)$ *such that*

$$|\hat{I}_{n,N} - I| \propto C N^{-\alpha},$$

where I *denotes the exact value of the Integral and* $\hat{I}_{n,N}$ *denotes its approximation.*

3.7 Discussion and bibliography

The presentation of part of the material follows that in [7], and some exercises come from [10, Chapter 9]. The main advantage of probabilistic integration approaches is that they generalize naturally to high-dimensional and infinite-dimensional settings.

Chapter 4

Solution of linear systems of equation

4.1 Conditioning 83
4.2 Direct solution method 87
4.2.1 LU decomposition 88
4.2.2 Backward and forward substitution 92
4.2.3 Gaussian elimination with pivoting 93
4.2.4 Direct method for Hermitian positive definite matrices 96
4.2.5 Direct methods for banded matrices 97
4.3 Iterative methods for linear systems 100
4.3.1 Basic iterative methods 100
4.3.2 The conjugate gradient method 109
4.4 Exercises 120
4.5 Discussion and bibliography 129

Introduction

This chapter is devoted to the numerical solution of linear problems of the following form:

$$\text{Find } \boldsymbol{x} \in \mathbf{C}^n \text{ such that} \qquad \mathsf{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \mathsf{A} \in \mathbf{C}^{n\times n}, \qquad \boldsymbol{b} \in \mathbf{C}^n. \tag{4.1}$$

Systems of this type appear in a variety of applications. They naturally arise in the context of linear partial differential equations, which we use as main motivating example. Partial differential equations govern a wide range of physical phenomena including heat propagation, gravity, and electromagnetism, to mention just a few. Linear systems in this context often have a particular structure: the matrix A is generally very sparse, which means that most of the entries are equal to 0, and it is often Hermitian and positive definite, provided that these properties are satisfied by the underlying operator.

There are two main approaches for solving linear systems:

- Direct methods enable to calculate the exact solution to systems of linear equations, up to round-off errors, in a finite number of steps. Although this is an attractive property, direct methods are usually too computationally costly for large systems: The cost of inverting a general $n \times n$ matrix, measured in number of floating operations, scales as n^3!
- Iterative methods, on the other hand, enable to progressively calculate increasingly accurate approximations of the solution. Iterations may be stopped once the *the residual* is sufficiently small. These methods are often preferable when the dimension n of the linear system is very large.

This chapter is organized as follows.

- In Section 4.1, we introduce the concept of *conditioning*. The condition number of a matrix provides information on the sensitivity of the solution to perturbations of the right-hand side $\boldsymbol{b}$ or matrix A. It is useful, for example, in order to determine the potential impact of round-off errors.
- In Section 4.2, we present the direct method for solving systems of linear equations. We study in particular the LU decomposition for an invertible matrix, as well as its variant for symmetric positive definite matrices, which is called the Cholesky decomposition.
- In Section 4.3, we present iterative methods for solving linear systems. We focus, in particular, on basic iterative methods based on a splitting of the matrix A and on the conjugate gradient method.

4.1 Conditioning

Before studying the properties of numerical methods for solving linear system, it is crucial to understand the impact of *round-off errors* on the solution, which impose a bound on the accuracy we can hope to achieve. We begin with a motivating example.

Example 4.1. Suppose that we wish to solve the following equation

$$\mathsf{A}\boldsymbol{x} := \begin{pmatrix} 1 & 1 \\ 1 & 1 - 10^{-12} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 10^{-12} \end{pmatrix} =: \boldsymbol{b}.$$

The exact solution is given by $(1 \ -1)^T$. In Julia, this equation can be solved as follows:

```
A = [1 1; 1 (1-1e-12)]
b = [0; 1e-12]
x = A\b
```

The solution returned by the program is the following:

```
  1.0000221222095027
 -1.0000221222095027
```

The relative error on the solution is of the order of 10^{-5}, which is about 12 orders of magnitude larger than the machine epsilon for the `Float64` type. In order to understand this, note that the final error on `x` arises from three sources:

- First, the machine representation `b` of the right-hand side is only *an approximation* of the true right-hand side $\boldsymbol{b}$.
- Similarly, the machine representation `A` of the matrix is only *an approximation* of the true matrix A.
- Finally, the operation `A\b` itself leads to additional round-off errors, as the computer implementation of the backslash operator is based on elementary arithmetic operations. Understanding the magnitude of the error introduced at this step is delicate, and we shall not address this question.

It follows from the first two items that, when writing `A\b`, we are in fact asking the computer for a solution $\boldsymbol{x} + \Delta\boldsymbol{x}$ to a perturbed equation

$$(\mathsf{A} + \Delta\mathsf{A})(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b},$$

where $\Delta\mathsf{A}$ and $\Delta\boldsymbol{b}$ represent the rounding errors on the matrix and right-hand side, respectively. Understanding the impact of the perturbations $\Delta\boldsymbol{b}$ and $\Delta\mathsf{A}$ is the goal of this section. We shall prove, in particular, that the magnitude of the error $\Delta\boldsymbol{x}$ is related to the perturbations $\Delta\boldsymbol{b}$ and $\Delta\mathsf{A}$ through a quantity known as the *condition number* of the matrix A.

In general, the condition number for a given problem measures the sensitivity of the solution to the input data. In order to define this concept precisely, we consider a general problem of the form $F(x, d) = 0$, with unknown x and data d. The linear system (4.1) can be recast in this form, with the input data equal to $\boldsymbol{b}$ or A or both. We denote the solution corresponding to perturbed input data $d + \Delta d$ by $x + \Delta x$. The absolute and relative condition numbers are defined as follows.

Definition 4.1 (Condition number for the problem $F(x, d) = 0$)**.** The absolute and relative condition numbers with respect to perturbations of d are defined as

$$K_{\text{abs}}(d) = \lim_{\varepsilon \to 0} \left(\sup_{\|\Delta d\| \leqslant \varepsilon} \frac{\|\Delta x\|}{\|\Delta d\|} \right), \qquad K(d) = \lim_{\varepsilon \to 0} \left(\sup_{\|\Delta d\| \leqslant \varepsilon} \frac{\|\Delta x\|/\|x\|}{\|\Delta d\|/\|d\|} \right).$$

The short notation K is reserved for the relative condition number, which is often more useful in applications.

In the rest of this section, we obtain an upper bound on the relative condition number for the linear system (4.1) with respect to perturbations first of $\boldsymbol{b}$, and then of A. We use the notation $\|\bullet\|$ to denote both a vector norm on $\mathbf{C}^n$ and the induced operator norm on matrices.

Proposition 4.1 (Perturbation of the right-hand side)**.** *Let $\boldsymbol{x} + \Delta\boldsymbol{x}$ denote the solution to the perturbed equation $\mathsf{A}(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b}$. Then it holds that*

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leqslant \|\mathsf{A}\|\|\mathsf{A}^{-1}\| \frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}, \tag{4.2}$$

Proof. It holds by definition of $\Delta\boldsymbol{x}$ that $\mathsf{A}\Delta\boldsymbol{x} = \Delta\boldsymbol{b}$. Therefore, we have

$$\|\Delta\boldsymbol{x}\| = \|\mathsf{A}^{-1}\Delta\boldsymbol{b}\| \leqslant \|\mathsf{A}^{-1}\|\|\Delta\boldsymbol{b}\| = \frac{\|\mathsf{A}\boldsymbol{x}\|}{\|\boldsymbol{b}\|}\|\mathsf{A}^{-1}\|\|\Delta\boldsymbol{b}\| \leqslant \frac{\|\mathsf{A}\|\|\boldsymbol{x}\|}{\|\boldsymbol{b}\|}\|\mathsf{A}^{-1}\|\|\Delta\boldsymbol{b}\|. \tag{4.3}$$

Here we employed (A.11), proved in Appendix A, in the first and last inequalities. Rearranging the inequality (4.3), we obtain (4.2). □

Proposition 4.1 implies that the relative condition number of (4.1) with respect to perturbations of the right-hand side is bounded from above by $\|\mathsf{A}\|\|\mathsf{A}^{-1}\|$. Exercise 4.1 shows that there are values of $\boldsymbol{x}$ and $\Delta\boldsymbol{b}$ for which the inequality (4.2) is an equality, indicating that the inequality is sharp.

Studying the impact of perturbations of the matrix A is slightly more difficult, because this time the variation $\Delta\boldsymbol{x}$ of the solution does not depend linearly on the perturbation of the data. Before stating and proving the main result, we show an ancillary lemma.

Lemma 4.2. *Let $\mathsf{B} \in \mathbf{C}^{n\times n}$ be such that $\|\mathsf{B}\| < 1$ in some submultiplicative matrix norm $\|\bullet\|$. Then $\mathsf{I} - \mathsf{B}$ is invertible and*

$$\|(\mathsf{I} - \mathsf{B})^{-1}\| \leqslant \frac{1}{1 - \|\mathsf{B}\|}, \tag{4.4}$$

where $\mathsf{I} \in \mathbf{C}^{n\times n}$ is the identity matrix.

Proof. It holds for any matrix $\mathsf{B} \in \mathbf{C}^{n\times n}$ that

$$\mathsf{I} - \mathsf{B}^{n+1} = (\mathsf{I} - \mathsf{B})(\mathsf{I} + \mathsf{B} + \cdots + \mathsf{B}^n). \tag{4.5}$$

Since $\|\mathsf{B}\| < 1$ in a submultiplicative matrix norm, both sides of the equation are convergent in the limit as $n \to \infty$. The left-hand side converges to the identity matrix I, and the right-hand side converges as $n \to \infty$ because $\{\mathsf{S}_0, \mathsf{S}_1, \dots\}$ with

$$\mathsf{S}_n := \mathsf{I} + \mathsf{B} + \cdots + \mathsf{B}^n$$

is a Cauchy sequence in the vector space of matrices endowed with the norm for which $\|\mathsf{B}\| < 1$. Indeed, by the triangle inequality and the submultiplicative property of the norm, it holds that

$$\begin{aligned}\|\mathsf{S}_{n+m} - \mathsf{S}_n\| &= \|\mathsf{B}^{n+1} + \cdots + \mathsf{B}^{n+m}\| \\ &\leqslant \|\mathsf{B}^{n+1}\| + \cdots + \|\mathsf{B}^{n+m}\| \leqslant \|\mathsf{B}\|^{n+1} + \cdots + \|\mathsf{B}\|^{n+m} \\ &\leqslant \frac{\|\mathsf{B}\|^{n+1}}{1 - \|\mathsf{B}\|} \xrightarrow[n\to\infty]{} 0,\end{aligned}$$

where we employed the formula for a geometric series in the last inequality. Equating the limits of both sides of (4.5), we obtain

$$\mathsf{I} = (\mathsf{I} - \mathsf{B}) \sum_{i=0}^{\infty} \mathsf{B}^i.$$

This implies that $(\mathsf{I} - \mathsf{B})$ is invertible with inverse given by a so-called *Neumann* series

$$(\mathsf{I} - \mathsf{B})^{-1} = \sum_{i=0}^{\infty} \mathsf{B}^i.$$

Applying the triangle inequality repeatedly, and then using the submultiplicative property of the norm, we obtain

$$\forall n \in \mathbf{N}, \qquad \left\| \sum_{i=0}^{n} \mathsf{B}^i \right\| \leqslant \sum_{i=0}^{n} \|\mathsf{B}^i\| \leqslant \sum_{i=0}^{n} \|\mathsf{B}\|^i = \frac{1}{1 - \|\mathsf{B}\|}.$$

where we used the summation formula for geometric series in the last equality. Letting $n \to \infty$ in this equation and using the continuity of the norm enables to conclude the proof. □

Proposition 4.3 (Perturbation of the matrix)**.** *Let $\boldsymbol{x} + \Delta\boldsymbol{x}$ denote the solution to the perturbed equation $(\mathsf{A} + \Delta\mathsf{A})(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b}$. If A is invertible and $\|\Delta\mathsf{A}\| < \|\mathsf{A}^{-1}\|^{-1}$, then*

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leqslant \|\mathsf{A}\|\|\mathsf{A}^{-1}\| \frac{\|\Delta\mathsf{A}\|}{\|\mathsf{A}\|} \left(\frac{1}{1 - \|\mathsf{A}^{-1}\Delta\mathsf{A}\|} \right). \tag{4.6}$$

Proof. Left-multiplying both sides of the perturbed equation with A^{-1}, we obtain

$$(\mathsf{I} + \mathsf{A}^{-1}\Delta\mathsf{A})(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{x} \quad \Leftrightarrow \quad (\mathsf{I} + \mathsf{A}^{-1}\Delta\mathsf{A})\Delta\boldsymbol{x} = -\mathsf{A}^{-1}\Delta\mathsf{A}\boldsymbol{x}. \tag{4.7}$$

Since $\|\mathsf{A}^{-1}\Delta\mathsf{A}\| \leqslant \|\mathsf{A}^{-1}\|\|\Delta\mathsf{A}\| < 1$ by assumption, we deduce from Lemma 4.2 that the matrix on the left-hand side is invertible with a norm bounded as in (4.4). Consequently, using in addition the assumed submultiplicative property of the norm, we obtain that

$$\|\Delta\boldsymbol{x}\| = \|(\mathsf{I} + \mathsf{A}^{-1}\Delta\mathsf{A})^{-1}\mathsf{A}^{-1}\Delta\mathsf{A}\boldsymbol{x}\| \leqslant \frac{\|\mathsf{A}^{-1}\Delta\mathsf{A}\|}{1 - \|\mathsf{A}^{-1}\Delta\mathsf{A}\|}\|\boldsymbol{x}\|.$$

which enables to conclude the proof. □

Using Proposition 4.3, we deduce that the relative condition number of (4.1) with respect to perturbations of the matrix A is also bounded from above by $\|\mathsf{A}\|\|\mathsf{A}^{-1}\|$, because the term between brackets on the right-hand side of (4.6) converges to 1 as $\|\Delta\mathsf{A}\| \to 0$.

Propositions 4.1 and 4.3 show that the condition number of the linear system (4.1), with respect to perturbations of either $\boldsymbol{b}$ or A, depends only on A. This motivates the following definition of the condition number *of a matrix*.

Definition 4.2 (Condition number of a matrix)**.** The condition number of a matrix A associated with a vector norm $\|\bullet\|$ is defined as

$$\kappa(\mathsf{A}) = \|\mathsf{A}\|\|\mathsf{A}^{-1}\|.$$

The condition number for the p-norm, defined in Definition A.4, is denoted by $\kappa_p(\mathsf{A})$.

Note that the condition number $\kappa(\mathsf{A})$ associated with an induced norm, i.e. a matrix norm induced by a vector norm, is at least one. Indeed, since the identity matrix has induced norm 1, it holds that

$$1 = \|\mathsf{I}\| = \|\mathsf{A}\mathsf{A}^{-1}\| \leqslant \|\mathsf{A}\|\|\mathsf{A}^{-1}\|.$$

Since the 2-norm of an invertible matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ coincides with the spectral radius $\rho(\mathsf{A}^T\mathsf{A})$, the condition number κ_2 corresponding to the 2-norm is equal to

$$\kappa_2(\mathsf{A}) = \sqrt{\frac{\lambda_{\max}(\mathsf{A}^T\mathsf{A})}{\lambda_{\min}(\mathsf{A}^T\mathsf{A})}},$$

where $\lambda_{\max}(\mathsf{A}^T\mathsf{A})$ and $\lambda_{\min}(\mathsf{A}^T\mathsf{A})$ are the maximal and minimal (both real and positive) eigenvalues of the matrix $\mathsf{A}^T\mathsf{A}$.

Example 4.2 (Perturbation of the matrix). Consider the following linear system with perturbed matrix

$$(\mathsf{A}+\Delta\mathsf{A})\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}0\\0.01\end{pmatrix}, \qquad \mathsf{A} = \begin{pmatrix}1 & 0\\0 & 0.01\end{pmatrix}, \qquad \Delta\mathsf{A} = \begin{pmatrix}0 & 0\\0 & \varepsilon\end{pmatrix},$$

where $0 < \varepsilon \ll 0.01$. Here the eigenvalues of A are given by $\lambda_1 = 1$ and $\lambda_2 = 0.01$. The solution when $\varepsilon = 0$ is given by $(0,1)^T$, and the solution to the perturbed equation is

$$\begin{pmatrix}x_1+\Delta x_1\\x_2+\Delta x_2\end{pmatrix} = \begin{pmatrix}0\\\frac{1}{1+100\varepsilon}\end{pmatrix}.$$

Consequently, we deduce that, in the 2-norm,

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \left|\frac{100\varepsilon}{1+100\varepsilon}\right| \approx 100\varepsilon = 100\frac{\|\Delta\mathsf{A}\|}{\|\mathsf{A}\|}.$$

In this case, the relative impact of perturbations of the matrix is close to $\kappa_2(\mathsf{A}) = 100$.

4.2 Direct solution method

In this section, we present the *direct method* for solving linear systems of the form (4.1) with a general invertible matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$. The direct method can be decomposed into three steps:

- First calculate the so-called LU decomposition of A, i.e. find an upper triangular matrix U and a *unit* lower triangular matrix L such that $\mathsf{A} = \mathsf{LU}$. A unit lower triangular matrix is a lower triangular matrix with only ones on the diagonal.
- Then solve $\mathsf{L}\boldsymbol{y} = \boldsymbol{b}$ using a method called *forward substitution*.
- Finally, solve $\mathsf{U}\boldsymbol{x} = \boldsymbol{y}$ using a method called *backward substitution*.

By construction, the solution $\boldsymbol{x}$ thus obtained is a solution to (4.1). Indeed, we have that

$$\mathsf{A}\boldsymbol{x} = \mathsf{LU}\boldsymbol{x} = \mathsf{L}\boldsymbol{y} = \boldsymbol{b}.$$

4.2.1 LU decomposition

In this section, we first discuss the existence and uniqueness of the LU factorization of a matrix. We then describe a numerical algorithm for calculating the factors L and U, based on *Gaussian elimination*.

Existence and uniqueness of the decomposition

We present a necessary and sufficient condition for the existence of a unique LU decomposition of a matrix. To this end, we define the principal submatrix of order i of a matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ as the matrix $\mathsf{A}_i = \mathsf{A}[1:i, 1:i]$, in Julia notation.

Proposition 4.4. *The LU factorization of a matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$, with L unit lower triangular and U upper triangular, exists and is unique if and only if the principal submatrices of A of all orders are nonsingular.*

Proof. We prove only the "if" direction; see [10, Theorem 3.4] for the "only if" implication.

The statement is clear if $n = 1$. Reasoning by induction, we assume that the result is proved up to $n-1$. Since the matrix A_{n-1} and all its principal submatrices are nonsingular by assumption, it holds that $\mathsf{A}_{n-1} = \mathsf{L}_{n-1}\mathsf{U}_{n-1}$ for a unit lower triangular matrix L_{n-1} and an upper triangular matrix U_{n-1}. These two matrices are nonsingular, for if either of them were singular then the product $\mathsf{A}_{n-1} = \mathsf{L}_{n-1}\mathsf{U}_{n-1}$ would be singular as well. Let us decompose A as follows:

$$\mathsf{A} = \begin{pmatrix} \mathsf{A}_{n-1} & \boldsymbol{c} \\ \boldsymbol{d}^T & a_{nn} \end{pmatrix}.$$

Let ℓ and $\boldsymbol{u}$ denote the solutions to $\mathsf{L}_{n-1}\boldsymbol{u} = \boldsymbol{c}$ and $\mathsf{U}_{n-1}^T\ell = \boldsymbol{d}$. These solutions exist and are unique, because the matrices L_{n-1} and U_{n-1} are nonsingular. Letting $u_{nn} = a_{nn} - (\ell^T\boldsymbol{u})^{-1}$, we check that A factorizes as

$$\begin{pmatrix} \mathsf{A}_{n-1} & \boldsymbol{c} \\ \boldsymbol{d}^T & a_{nn} \end{pmatrix} = \begin{pmatrix} \mathsf{L}_{n-1} & \mathbf{0}_{n-1} \\ \ell^T & 1 \end{pmatrix} \begin{pmatrix} \mathsf{U}_{n-1} & \boldsymbol{u} \\ \mathbf{0}_{n-1}^T & u_{nn} \end{pmatrix}.$$

This completes the proof of the existence of the decomposition. The uniqueness of the factors follows from the uniqueness of ℓ, $\boldsymbol{u}$ and u_{nn}. □

Proposition 4.4 raises the following question: are there classes of matrices whose principal matrices are all nonsingular? The answer is positive, and we mention, as an important example, the class of positive definite matrices. Proving this is the aim of Exercise 4.4.

Gaussian elimination algorithm for computing L and U

So far we have presented a condition under which the LU decomposition of a matrix exists and is unique, but not a practical method for calculating the matrices L and U. We describe in this section an algorithm, known as *Gaussian elimination*, for calculating the LU decomposition of a matrix. We begin by introducing the concept of *Gaussian transformation*.

Definition 4.3. A Gaussian transformation is a matrix of the form $\mathsf{M}_k = \mathsf{I} - \boldsymbol{c}^{(k)} \boldsymbol{e}_k^T$, where $\boldsymbol{e}_k$ is the column vector with entry at index k equal to 1 and all the other entries equal to zero, and $\boldsymbol{c}^{(k)}$ is a column vector of the following form:

$$\boldsymbol{c}^{(k)} = \begin{pmatrix} 0 & 0 & \dots & 0 & c_{k+1}^{(k)} & c_{k+2}^{(k)} & \dots & c_n^{(k)} \end{pmatrix}^T.$$

The action of a Gaussian transformation M_k left-multiplying a matrix $\mathsf{A} \in \mathbf{C}^{n \times n}$ is to replace the rows from index $k+1$ to index n by a linear combination involving themselves and the k-th row. To see this, let us denote by $(\boldsymbol{r}^{(i)})_{1 \leqslant i \leqslant n}$ the rows of a matrix $\mathsf{T} \in \mathbf{C}^{n \times n}$. Then, we have

$$\mathsf{M}_k \mathsf{T} = \left(\mathsf{I} - \boldsymbol{c}^{(k)} \boldsymbol{e}_k^T\right) \mathsf{T} = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & -c_{k+1}^{(k)} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -c_n^{(k)} & & & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{r}^{(1)} \\ \boldsymbol{r}^{(2)} \\ \vdots \\ \boldsymbol{r}^{(k)} \\ \boldsymbol{r}^{(k+1)} \\ \vdots \\ \boldsymbol{r}^{(n)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}^{(1)} \\ \boldsymbol{r}^{(2)} \\ \vdots \\ \boldsymbol{r}^{(k)} \\ \boldsymbol{r}^{(k+1)} - c_{k+1}^{(k)} \boldsymbol{r}^{(k)} \\ \vdots \\ \boldsymbol{r}^{(n)} - c_n^{(k)} \boldsymbol{r}^{(k)} \end{pmatrix}$$

We show in Exercise 4.2 that the inverse of a Gaussian transformation matrix is given by

$$(\mathsf{I} - \boldsymbol{c}^{(k)} \boldsymbol{e}_k^T)^{-1} = \mathsf{I} + \boldsymbol{c}^{(k)} \boldsymbol{e}_k^T. \tag{4.8}$$

The idea of the Gaussian elimination algorithm is to successively left-multiply A with Gaussian transformation matrices M_1, then M_2, etc. appropriately chosen in such a way that the matrix $\mathsf{A}^{(k)}$, obtained after k iterations, is upper triangular up to column k. That is to say, the Gaussian transformations are constructed so that all the entries in columns 1 to k under the diagonal of the matrix $\mathsf{A}^{(k)}$ are equal to zero. The resulting matrix $\mathsf{A}^{(n-1)}$ after $n-1$ iterations is then upper triangular and satisfies

$$\mathsf{A}^{(n-1)} = \mathsf{M}_{n-1} \dots \mathsf{M}_1 \mathsf{A}.$$

Rearranging this equation, we deduce that

$$\mathsf{A} = (\mathsf{M}_1^{-1} \dots \mathsf{M}_{n-1}^{-1})\mathsf{A}^{(n-1)}.$$

The first factor is lower triangular by (4.8) and Exercise 4.3. The product in the definition of the matrix L admits a simple explicit expression.

Lemma 4.5. *It holds that*

$$\mathsf{M}_1^{-1} \cdots \mathsf{M}_{n-1}^{-1} = (\mathsf{I} + \boldsymbol{c}^{(1)}\boldsymbol{e}_1^T) \cdots (\mathsf{I} + \boldsymbol{c}^{(n-1)}\boldsymbol{e}_{n-1}^T) = \mathsf{I} + \sum_{i=1}^{n-1} \boldsymbol{c}^{(i)}\boldsymbol{e}_i^T.$$

Proof. Notice that, for $i < j$,

$$\boldsymbol{c}^{(i)}\boldsymbol{e}_i^T\boldsymbol{c}^{(j)}\boldsymbol{e}_j^T = \boldsymbol{c}^{(i)}(\boldsymbol{e}_i^T\boldsymbol{c}^{(j)})\boldsymbol{e}_j^T = \boldsymbol{c}^{(i)}0\boldsymbol{e}_j^T = 0.$$

The statement then follows easily by expanding the product. □

A corollary of Lemma 4.5 is that all the diagonal entries of the lower triangular matrix L are equal to 1; the matrix L is *unit lower triangular*. The full expression of the matrix L given the Gaussian transformations is

$$\mathsf{L} = \mathsf{I} + \begin{pmatrix} \boldsymbol{c}^{(1)} & \dots & \boldsymbol{c}^{(n-1)} & \mathbf{0}_n \end{pmatrix} = \begin{pmatrix} 1 & & & & & \\ c_2^{(1)} & 1 & & & & \\ c_3^{(1)} & c_3^{(2)} & 1 & & & \\ c_4^{(1)} & c_4^{(2)} & c_4^{(3)} & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ c_n^{(1)} & c_n^{(2)} & c_n^{(3)} & \dots & c_n^{(n-1)} & 1 \end{pmatrix} \tag{4.9}$$

Therefore, the Gaussian elimination algorithms, if all the steps are well-defined, correctly gives the LU factorization of the matrix A. Of course, the success of the strategy outlined above for the calculation of the LU factorization hinges on the existence of an appropriate Gaussian transformation at each iteration. It is not difficult to show that, if the $(k+1)$-th diagonal entry of the matrix $\mathsf{A}^{(k)}$ is nonzero for all $k \in \{1, \dots, n-2\}$, then the Gaussian transformation matrices exist and are uniquely defined.

Lemma 4.6. *Assume that $\mathsf{A}^{(k)}$ is upper triangular up to column k included, with $k \leqslant n-2$. If $a_{k+1,k+1}^{(k)} > 0$, then there is a unique Gaussian transformation matrix M_{k+1} such that $\mathsf{M}_{k+1}\mathsf{A}^{(k)}$ is upper triangular up to column $k+1$. This transformation matrix is given by $\mathsf{I} - \boldsymbol{c}^{(k+1)}\boldsymbol{e}_{k+1}^T$, where*

$$\boldsymbol{c}^{(k+1)} = \begin{pmatrix} 0 & 0 & \dots & 0 & \frac{a_{k+2,k+1}^{(k)}}{a_{k+1,k+1}^{(k)}} & \frac{a_{k+3,k+1}^{(k)}}{a_{k+1,k+1}^{(k)}} & \dots & \frac{a_{n,k+1}^{(k)}}{a_{k+1,k+1}^{(k)}} \end{pmatrix}^T.$$

Proof. We perform the multiplication explicitly. Denoting denote by $(\boldsymbol{r}^{(i)})_{1 \leqslant i \leqslant n}$ the rows of $\mathsf{A}^{(k)}$, we have

$$\mathsf{M}_{k+1}\mathsf{A}^{(k)} = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & -c_{k+2}^{(k+1)} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -c_{n}^{(k+1)} & & & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{r}^{(1)} \\ \boldsymbol{r}^{(2)} \\ \vdots \\ \boldsymbol{r}^{(k+1)} \\ \boldsymbol{r}^{(k+2)} \\ \vdots \\ \boldsymbol{r}^{(n)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}^{(1)} \\ \boldsymbol{r}^{(2)} \\ \vdots \\ \boldsymbol{r}^{(k+1)} \\ \boldsymbol{r}^{(k+2)} - c_{k+2}^{(k+1)}\boldsymbol{r}^{(k+1)} \\ \vdots \\ \boldsymbol{r}^{(n)} - c_{n}^{(k+1)}\boldsymbol{r}^{(k+1)} \end{pmatrix}.$$

We need to show that the matrix on the right-hand side is upper triangular up to column $k+1$ included. This is clear by definition of $\boldsymbol{c}^{(k+1)}$ and from the fact that $\mathsf{A}^{(k)}$ is upper triangular up to column k by assumption. □

The diagonal elements $a_{k+1,k+1}^{(k)}$, where $k \in \{0, \dots, n-2\}$, are called the pivots. We now prove that, if an invertible matrix A admits an LU factorization, then the pivots are necessarily nonzero and the Gaussian elimination algorithm is successful.

Proposition 4.7 (Gaussian elimination works). *If A is invertible and admits an LU factorization, then the Gaussian elimination algorithm is well-defined and successfully terminates.*

Proof. We denote by $\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(n-1)}$, the columns of the matrix $\mathsf{L} - \mathsf{I}$. Then the matrices given by $\mathsf{M}_k = \mathsf{I} - \boldsymbol{c}^{(k)}\boldsymbol{e}_k^T$, for $k \in \{1, \dots, n-1\}$, are Gaussian transformations and it holds that

$$\mathsf{L} = \mathsf{M}_1^{-1} \cdots \mathsf{M}_{n-1}^{-1}$$

in view of Lemma 4.5. Since $\mathsf{A} = \mathsf{LU}$ by assumption, the result of the product

$$\mathsf{M}_{n-1} \cdots \mathsf{M}_1 \mathsf{A} = \mathsf{U}$$

is upper triangular. Let us use the notation $\mathsf{A}^{(k)} = \mathsf{M}_k \cdots \mathsf{M}_1 \mathsf{A}$. Our goal is to show that the matrices $\mathsf{M}_1, \dots, \mathsf{M}_{n-1}$ are the same as those obtained by the Gaussian elimination algorithm.

Of all the Gaussian transformations $\mathsf{M}_1, \dots, \mathsf{M}_{n-1}$, only M_1 acts on the second row of the matrix it multiplies. Therefore, the entry $(2,1)$ of U coincides with the entry $(2,1)$ of $\mathsf{A}^{(1)}$, which implies that $a_{2,1}^{(1)} = 0$. Then notice that $a_{3,1}^{(k)} = a_{3,1}^{(1)}$ for all $k \geqslant 1$, because the entry $(3,1)$ of the matrix $\mathsf{M}_2\mathsf{A}^{(1)}$ is given by $a_{3,1}^{(1)} - c_3^{(2)} a_{2,1}^{(1)} = a_{3,1}^{(1)}$, and all the other transformation matrices $\mathsf{M}_3, \dots, \mathsf{M}_{n-1}$ leave the third row invariant. Consequently, it holds that $a_{3,1}^{(1)} = u_{3,1} = 0$. Continuing in this manner, we deduce that $\mathsf{A}^{(1)}$ is upper triangular in the first column and that, since A is invertible by assumption, the first pivot a_{11} is nonzero. Since this pivot is nonzero, the matrix M_1 is uniquely defined by Lemma 4.6.

The reasoning can then be repeated with other columns, in order to deduce that $\mathsf{A}^{(k)}$ is upper triangular up to column k and that all the pivots $a_{kk}^{(k-1)}$ are nonzero. Therefore, all the Gaussian transformation matrices are uniquely defined given Lemma 4.6. □

Computer implementation

The Gaussian elimination procedure is summarized as follows.

$\mathsf{A}^{(0)} \leftarrow \mathsf{A}, \mathsf{L} \leftarrow \mathsf{I}$
for $i \in \{1, \ldots, n-1\}$ **do**
  Construct M_i as in Lemma 4.6.
  $\mathsf{A}^{(i)} \leftarrow \mathsf{M}_i \mathsf{A}^{(i-1)}, \mathsf{L} \leftarrow \mathsf{L}\mathsf{M}_i^{-1}$
end for
$\mathsf{U} \leftarrow \mathsf{A}^{(n-1)}$.

In practice, it is not necessary to explicitly create the Gaussian transformation matrices, or to perform full matrix multiplications. A more realistic version of the algorithm in Julia is given below. The code exploits the relation (4.9) between L and the parameters of the Gaussian transformations.

```
1  # A is an invertible matrix of size n x n
2  L = [i == j ? 1.0 : 0.0 for i in 1:n, j in 1:n]
3  U = copy(A)
4  for i in 1:n-1
5      for r in i+1:n
6          U[i, i] == 0 && error("Pivotal entry is zero!")
7          ratio = U[r, i] / U[i, i]
8          L[r, i] = ratio
9          U[r, i:end] -= U[i, i:end] * ratio
10     end
11 end
12 # L is unit lower triangular and U is upper triangular
```

Computational cost

The computational cost of the algorithm, measured as the number of floating point operations (flops) required, is dominated by the Gaussian transformations, in line 9 in the above code. All the other operations amount to a computational cost scaling as $\mathcal{O}(n^2)$, which is negligible compared to the cost of the LU factorization when n is large. This factorization requires

$$\overbrace{2\times}^{\text{- and *}} \sum_{\substack{i=1 \\ \texttt{for i in 1:n-1}}}^{n-1} \overbrace{(n-i)}^{\texttt{for r in i+1:n}} \underbrace{(n-i+1)}_{\texttt{indices [i:end]}} \quad \text{flops} = \frac{2}{3}n^3 + \mathcal{O}(n^2) \quad \text{flops}.$$

4.2.2 Backward and forward substitution

Once the LU factorization has been completed, the solution to the linear system can be obtained by first using forward, and then backward substitution, which are just bespoke methods for solving linear systems with lower and upper triangular matrices, respectively. Let us consider the case of a lower triangular system:

$$\mathsf{L}\boldsymbol{y} = \boldsymbol{b}$$

Notice that the unknown y_1 may be obtained from the first equation of the system. Then, since y_1 is known, the value of y_2 can be obtained from the second equation, etc. A simple implementation of this algorithm is as follows:

```
# L is unit lower triangular
y = copy(b)
for i in 2:n
    for j in 1:i-1
        y[i] -= L[i, j] * y[j]
    end
end
```

4.2.3 Gaussian elimination with pivoting

The Gaussian elimination algorithm that we presented in Section 4.2.1 relies on the existence of an LU factorization. In practice, this assumption may not be satisfied, and in this case a modified algorithm, called Gaussian elimination *with pivoting*, is required.

In fact, pivoting is useful even if the usual LU decomposition of A exists, as it enables to reduce the condition number of the matrices matrices L and U. There are two types of pivoting: partial pivoting, where only the rows are rearranged through a permutation at each iteration, and complete pivoting, where both the rows and the columns are rearranged at each iteration.

Showing rigorously why pivoting is useful is beyond the scope of this course. In this section, we only present the partial pivoting method. Its influence on the condition number of the factors L and U is studied empirically in Exercise 4.6. It is useful at this point to introduce the concept of a row permutation matrix.

Row permutation matrix

Definition 4.4. Let $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a permutation, i.e. a bijection on the set $\{1, \ldots, n\}$. The row permutation matrix associated with σ is the matrix with entries

$$p_{ij} = \begin{cases} 1 \text{ if } i = \sigma(j), \\ 0 \text{ otherwise.} \end{cases}$$

When a row permutation P left-multiplies a matrix $\mathsf{B} \in \mathbf{C}^{n \times n}$, row i of matrix B is moved to row index $\sigma(i)$ in the resulting matrix, for all $i \in \{1, \ldots, n\}$. A permutation matrix has a single entry equal to 1 per row and per column, and its inverse coincides with its transpose: $\mathsf{P}^{-1} = \mathsf{P}^T$.

Partial pivoting

Gaussian elimination with partial pivoting applies for any invertible matrix A, and it outputs 3 matrices: a row permutation P, a unit triangular matrix L, and an upper triangular matrix U. These are related by the relation

$$\mathsf{PA} = \mathsf{LU}.$$

This is sometimes called a PLU decomposition of the matrix A. It is not unique in general but, unlike the usual LU decomposition, it always exists provided that A is invertible. We take this for granted in this course.

The idea of partial pivoting is to rearrange the rows at each iteration of the Gaussian elimination procedure in such a manner that the pivotal entry is as large as possible in absolute value. One step of the procedure reads

$$\mathsf{A}^{(k+1)} = \mathsf{M}_{k+1}\mathsf{P}_{k+1}\mathsf{A}^{(k)}. \tag{4.10}$$

Here P_{k+1} is a simple row permutation matrix which, when acting on $\mathsf{A}^{(k)}$, interchanges row $k+1$ and row ℓ, for some index $\ell \geqslant k+1$. The row index ℓ is selected in such a way that the absolute value of the pivotal entry, in position $(k+1, k+1)$ of the product $\mathsf{P}_{k+1}\mathsf{A}^{(k)}$, is maximum. The matrix M_{k+1} is then the unique Gaussian transformation matrix ensuring that $\mathsf{A}^{(k+1)}$ is upper triangular up to column $k+1$, obtained as in Lemma 4.6. The resulting matrix $\mathsf{A}^{(n-1)}$ after $n-1$ steps of the form (4.10) is upper triangular and satisfies

$$\mathsf{A}^{(n-1)} = \mathsf{M}_{n-1}\mathsf{P}_{n-1}\cdots\mathsf{M}_1\mathsf{P}_1\mathsf{A} \quad\Leftrightarrow\quad \mathsf{A} = (\mathsf{M}_{n-1}\mathsf{P}_{n-1}\cdots\mathsf{M}_1\mathsf{P}_1)^{-1}\mathsf{A}^{(n-1)}.$$

The first factor in the decomposition of A is not necessarily lower triangular. However, using the notation $\mathsf{M} = \mathsf{M}_{n-1}\mathsf{P}_{n-1}\cdots\mathsf{M}_1\mathsf{P}_1$ and $\mathsf{P} = \mathsf{P}_{n-1}\cdots\mathsf{P}_1$, we have

$$\mathsf{PA} = \mathsf{PM}^{-1}\mathsf{U} = (\mathsf{PM}^{-1})\mathsf{U} =: \mathsf{LU}. \tag{4.11}$$

Lemma 4.8 below shows that, as the notation L suggests, the matrix $\mathsf{L} = (\mathsf{PM}^{-1})$ on the right-hand side is indeed lower triangular. Before stating and proving the lemma, we note that P is a row permutation matrix, and so the solution to the linear system $\mathsf{A}\boldsymbol{x} = \mathsf{b}$ can be obtained by solving $\mathsf{LU}\boldsymbol{x} = \mathsf{P}^T\boldsymbol{b}$ by forward and backward substitution. Since P is a very sparse matrix, the right-hand side $\mathsf{P}^T\boldsymbol{b}$ can be calculated very efficiently.

Lemma 4.8. *The matrix $\mathsf{L} = \mathsf{PM}^{-1}$ is unit lower triangular with all entries bounded in absolute value from above by 1. It admits the expression*

$$\mathsf{L} = \mathsf{I} + (\mathsf{P}_{n-1}\cdots\mathsf{P}_2\boldsymbol{c}^{(1)})\boldsymbol{e}_1^T + (\mathsf{P}_{n-1}\cdots\mathsf{P}_3\boldsymbol{c}^{(2)})\boldsymbol{e}_2^T + \cdots + (\mathsf{P}_{n-1}\boldsymbol{c}^{(n-2)})\boldsymbol{e}_{n-2}^T + \boldsymbol{c}^{(n-1)}\boldsymbol{e}_{n-1}^T.$$

Proof. Let $\mathsf{M}^{(k)} = \mathsf{M}_k\mathsf{P}_k\cdots\mathsf{M}_1\mathsf{P}_1$ and $\mathsf{P}^{(k)} = \mathsf{P}_k\cdots\mathsf{P}_1$. It is sufficient to show that

$$\mathsf{P}^{(k)}\big(\mathsf{M}^{(k)}\big)^{-1} = \mathsf{I} + (\mathsf{P}_k\cdots\mathsf{P}_2\boldsymbol{c}^{(1)})\boldsymbol{e}_1^T + (\mathsf{P}_k\cdots\mathsf{P}_3\boldsymbol{c}^{(2)})\boldsymbol{e}_2^T + \cdots + (\mathsf{P}_k\boldsymbol{c}^{(k-1)})\boldsymbol{e}_{k-1}^T + \boldsymbol{c}^{(k)}\boldsymbol{e}_k^T \tag{4.12}$$

for all $k \in \{1, \ldots, n-1\}$. The statement is clear for $k = 1$, and we assume by induction that it is true up to $k-1$. Then notice that

$$\begin{aligned}
\mathsf{P}^{(k)}\big(\mathsf{M}^{(k)}\big)^{-1} &= \mathsf{P}_k\Big(\mathsf{P}^{(k-1)}\big(\mathsf{M}^{(k-1)}\big)^{-1}\Big)\mathsf{P}_k^{-1}\mathsf{M}_k^{-1} \\
&= \mathsf{P}_k\Big(\mathsf{I} + (\mathsf{P}_{k-1}\cdots\mathsf{P}_2\boldsymbol{c}^{(1)})\boldsymbol{e}_1^T + \cdots + (\mathsf{P}_{k-1}\boldsymbol{c}^{(k-2)})\boldsymbol{e}_{k-2}^T + \boldsymbol{c}^{(k-1)}\boldsymbol{e}_{k-1}^T\Big)\mathsf{P}_k^{-1}\mathsf{M}_k^{-1} \\
&= \Big(\mathsf{I} + (\mathsf{P}_k\mathsf{P}_{k-1}\cdots\mathsf{P}_2\boldsymbol{c}^{(1)})\boldsymbol{e}_1^T + \cdots + (\mathsf{P}_k\mathsf{P}_{k-1}\boldsymbol{c}^{(k-2)})\boldsymbol{e}_{k-2}^T + (\mathsf{P}_k\boldsymbol{c}^{(k-1)})\boldsymbol{e}_{k-1}^T\Big)\mathsf{M}_k^{-1}.
\end{aligned}$$

In the last equality, we used that $\boldsymbol{e}_i^T \mathsf{P}_k^{-1} = (\mathsf{P}_k \boldsymbol{e}_i)^T = \boldsymbol{e}_i^T$ for all $i \in \{1, \ldots, k-1\}$, because the row permutation P_k does not affect rows 1 to $k-1$. Using the expression $\mathsf{M}_k^{-1} = \mathsf{I} + \boldsymbol{c}^{(k)} \boldsymbol{e}_k^T$, expanding the product and noting that $\boldsymbol{e}_j^T \boldsymbol{c}^{(k)} = 0$ if $j \leqslant k$, we obtain (4.12). The statement that the entries are bounded in absolute value from above by 1 follows from the choice of the pivot at each iteration. □

The expression of L in Lemma 4.8 suggests the iterative procedure given in Algorithm 2 for performing the LU factorization with partial pivoting. A Julia implementation of this algorithm is presented in Listing 1.

Algorithm 2 LU decomposition with partial pivoting

Assign $\mathsf{A}^{(0)} \leftarrow \mathsf{A}$ and $\mathsf{P} \leftarrow \mathsf{I}$
for $i \in \{1, \ldots, n-1\}$ **do**
  Find the row index $k \geqslant i$ such that $\mathsf{A}_{k,i}^{(i-1)}$ is maximum in absolute value.
  Interchange the rows i and k of matrices $\mathsf{A}^{(i-1)}$ and P, and of vectors $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(i-1)}$.
  Construct M_i with corresponding column vector $\boldsymbol{c}^{(i)}$ as in Lemma 4.6.
  Assign $\mathsf{A}^{(i)} \leftarrow \mathsf{M}_i \mathsf{A}^{(i-1)}$
end for
Assign $\mathsf{U} \leftarrow \mathsf{A}^{(n-1)}$.
Assign $\mathsf{L} \leftarrow \mathsf{I} + \begin{pmatrix} \boldsymbol{c}^{(1)} & \ldots & \boldsymbol{c}^{(n-1)} & \boldsymbol{0}_n \end{pmatrix}$.

```
# Auxiliary function
function swap_rows!(i, j, matrices...)
    for M in matrices
        M_row_i = M[i, :]
        M[i, :] = M[j, :]
        M[j, :] = M_row_i
    end
end

n = size(A)[1]
L, U = zeros(n, 0), copy(A)
P = [i == j ? 1.0 : 0.0 for i in 1:n, j in 1:n]
for i in 1:n-1
    # Pivoting
    index_row_pivot = i - 1 + argmax(abs.(U[i:end, i]))
    swap_rows!(i, index_row_pivot, U, L, P)

    # Usual Gaussian transformation
    c = [zeros(i-1); 1.0; zeros(n-i)]
    for r in i+1:n
        ratio = U[r, i] / U[i, i]
        c[r] = ratio
        U[r, i:end] -= U[i, i:end] * ratio
    end
    L = [L c]
end
L = [L [zeros(n-1); 1.0]]
# It holds that P*A = L*U
```

Listing 1: LU factorization with partial pivoting.

Remark 4.1. It is possible to show that, if the matrix A is column diagonally dominant in the sense that

$$\forall j \in \{1, \ldots, n\}, \qquad |a_{jj}| \geqslant \sum_{i=1, i\neq j}^{n} |a_{ij}|,$$

then pivoting does not have an effect: at each iteration, the best pivot is already on the diagonal.

4.2.4 Direct method for Hermitian positive definite matrices

The LU factorization with partial pivoting applies to any matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ that is invertible. If A is Hermitian positive definite, however, it is possible to compute a factorization into lower and upper triangular matrices at half the computational cost, using the so-called *Cholesky decomposition*.

Lemma 4.9 (Cholesky decomposition)**.** *If* A *is Hermitian positive definite, then there exists a lower-triangular matrix* $\mathsf{C} \in \mathbf{C}^{n\times n}$ *such that*

$$\mathsf{A} = \mathsf{C}\mathsf{C}^*. \tag{4.13}$$

Equation (4.13) is called the Cholesky factorization of A*. The matrix* C *is unique if we require that all its diagonal entries are positive.*

Proof. Since A is positive definite, its LU decomposition exists and is unique by Proposition 4.4 and Proposition 4.7. Let D denote the diagonal matrix with the same diagonal as that of U. Then

$$\mathsf{A} = \mathsf{L}\mathsf{D}(\mathsf{D}^{-1}\mathsf{U}).$$

Note that the matrix $\mathsf{D}^{-1}\mathsf{U}$ is unit upper triangular. Since A is Hermitian, we have

$$\mathsf{A} = \mathsf{A}^* = (\mathsf{D}^{-1}\mathsf{U})^*(\mathsf{L}\mathsf{D})^*.$$

The first and second factors on the right-hand side are respectively unit lower triangular and upper triangular, and so we deduce, by uniqueness of the LU decomposition, that $\mathsf{L} = (\mathsf{D}^{-1}\mathsf{U})^*$ and $\mathsf{U} = (\mathsf{L}\mathsf{D})^*$. But then

$$\mathsf{A} = \mathsf{L}\mathsf{U} = \mathsf{L}\mathsf{D}\mathsf{L}^* = (\mathsf{L}\sqrt{\mathsf{D}})(\sqrt{\mathsf{D}}\mathsf{L})^*.$$

Here $\sqrt{\mathsf{D}}$ denotes the diagonal matrix whose diagonal entries are obtained by taking the square root of those of D, which are necessarily real and positive because A is positive definite. This implies the existence of a Cholesky factorization with $\mathsf{C} = \mathsf{L}\sqrt{\mathsf{D}}$. □

Calculation of the Choleski factor

The matrix C can be calculated from (4.13). For example, developing the matrix product gives that $a_{1,1} = c_{1,1}^2$ and so $c_{1,1} = \sqrt{a_{1,1}}$. It is then possible to calculate $c_{2,1}$ from the equation $a_{2,1} = c_{2,1}c_{1,1}$, and so on. Implementing the Cholesky factorization is the goal of Exercise 4.7.

4.2.5 Direct methods for banded matrices

In applications related to partial differential equations, the matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ very often has a bandwidth which is small in comparison with n.

Definition 4.5. The bandwidth of a matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ is the smallest number $k \in \mathbf{N}$ such that $a_{ij} = 0$ for all $(i, j) \in \{1, \ldots, n\}^2$ with $|i - j| > k$.

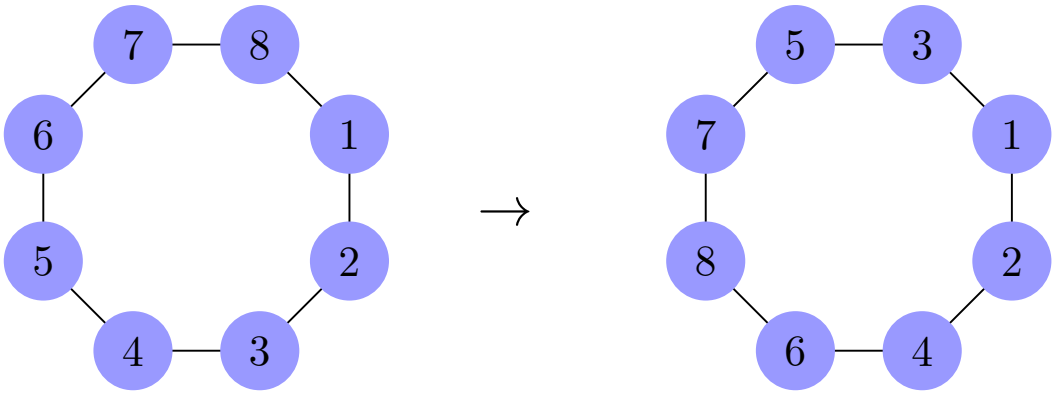
It is not difficult to show that, if A is a matrix with bandwidth k, then so are L and U in the absence of pivoting. This can be proved by equaling the entries of the product LU with those of the matrix A. We emphasize, however, that the sparsity structure *within* the band of A may be destroyed in L and U; this phenomenon is called *fill-in*.

Reducing the bandwidth: the Cuthill–McKee algorithm

The computational cost of calculating the LU or Cholesky decomposition of a matrix with bandwidth k scales as $\mathcal{O}(nk^2)$, which is much better than the general scaling $\mathcal{O}(n^3)$ if $k \ll n$. In applications, the bandwidth k is often related to the matrix size n. For example, if A arises from the discretization of the Laplacian operator, then $k = \mathcal{O}(\sqrt{n})$ provided that a good ordering of the vertices is employed. In this case, the computational cost scales as $\mathcal{O}(n^2)$.

Since a narrow band is associated with a lower computational cost of the LU decomposition, it is natural to wonder whether the bandwidth of a matrix A can be reduced. A possible strategy to this end is to use permutations. More precisely, is it possible to identify a row permutation matrix P such that PAP^T has minimal bandwidth? Given such a matrix, the solution to the linear system (4.1) can be obtained by first solving $(\mathsf{PAP}^T)\boldsymbol{y} = \mathsf{P}b$, and then letting $\boldsymbol{x} = \mathsf{P}^T\boldsymbol{y}$.

The Cuthill–McKee algorithm is a heuristic method for finding a good, but sometimes not optimal, permutation matrix P in the particular case where A is a *Hermitian* matrix. It is based on the fact that, to a Hermitian matrix A, we can associate a unique undirected graph whose adjacency matrix A_* has the same sparsity structure as that of A, i.e. zeros in the same places. For any row permutation matrix P_σ with corresponding permutation σ: $\{1, \ldots, n\} \to \{1, \ldots, n\}$ (see Definition 4.4), the matrices $\mathsf{P}_\sigma\mathsf{A}\mathsf{P}_\sigma^T$ and $\mathsf{P}_\sigma\mathsf{A}_*\mathsf{P}_\sigma^T$ also have the same sparsity structure. Therefore, minimizing the bandwidth of $\mathsf{P}_\sigma\mathsf{A}\mathsf{P}_\sigma^T$ is equivalent to minimizing the bandwidth of $\mathsf{P}_\sigma\mathsf{A}_*\mathsf{P}_\sigma^T$. The key insight for understanding the Cuthill–McKee method is that $\mathsf{P}_\sigma\mathsf{A}_*\mathsf{P}_\sigma^T$ is the adjacency matrix of the graph obtained by renumbering the nodes according to the permutation σ, i.e. by changing the number of the nodes from i to $\sigma(i)$. Consider, for example, the following graph and renumbering:



The associated adjacency matrices are given by:

$$\begin{pmatrix} 1 & 1 & & & & & & 1 \\ 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & & \\ & & & & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 \\ 1 & & & & & & 1 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & 1 & & & & & \\ 1 & 1 & & 1 & & & & \\ 1 & & 1 & & 1 & & & \\ & 1 & & 1 & & 1 & & \\ & & 1 & & 1 & & 1 & \\ & & & 1 & & 1 & & 1 \\ & & & & 1 & & 1 & 1 \\ & & & & & 1 & 1 & 1 \end{pmatrix}$$

We assume that the nodes are all self-connected, although this is not depicted, and so the diagonal entries of the adjacency matrices are equal to 1. This renumbering corresponds to the permutation

$$\begin{pmatrix} i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \sigma(i): & 1 & 2 & 4 & 6 & 8 & 7 & 5 & 3 \end{pmatrix},$$

and we may verify that the adjacency matrix of the renumbered graph can be obtained from the associated row permutation matrix:

$$\mathsf{PA}_*\mathsf{P}^T = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & & & & & & 1 \\ & & 1 & & & & & \\ & & & & & 1 & & \\ & & & 1 & & & & \\ & & & & & & 1 & \\ & & & & 1 & & & \end{pmatrix} \begin{pmatrix} 1 & 1 & & & & & & 1 \\ 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & & \\ & & & & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 \\ 1 & & & & & & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & & 1 & & & & \\ & & & & & 1 & & \\ & & & & & & & 1 \\ & & & & & & 1 & \\ & & & & 1 & & & \\ & & 1 & & & & & \end{pmatrix}$$

In this example, renumbering the nodes of the graph enables a significant reduction of the bandwidth, from 7 to 2. The Cuthill–McKee algorithm, which was employed to calculate the permutation, is an iterative method that produces an ordered n-tuple R containing the nodes in the new order; in other words, it returns $\left(\sigma^{-1}(1), \ldots, \sigma^{-1}(n)\right)$. The first step of the algorithm is to find the node i with the lowest *degree*, i.e. with the smallest number of connections to other nodes, and to initialize $R = (i)$. Then the following steps are repeated until R contains all the nodes of the graph:

- Define A_i as the set containing all the nodes which are adjacent to a node in R but not themselves in R;
- Sort the nodes in A_i according to the following rules: a node $i \in A_i$ comes before $j \in A_i$ if i is connected to a node in R that comes before all the nodes in R to which j is connected. As a tiebreak, precedence is given to the node with highest degree.
- Append the nodes in A_i to R, in the order determined in the previous item.

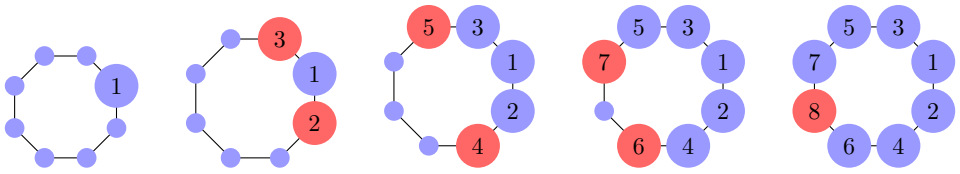


Figure 4.1: Illustration of the Cuthill–McKee algorithm. The new numbering of the nodes is illustrated. The first node was chosen randomly since all the nodes have the same degree. In this example, the ordered tuple R evolves as follows: $(1) \to (1, 2, 8) \to (1, 2, 8, 3, 7) \to (1, 2, 8, 3, 7, 4, 6) \to (1, 2, 8, 3, 7, 4, 6, 5)$.

The steps of the algorithm for the example above are depicted in Figure 4.1. Another example, taken from the original paper by Cuthill and McKeen [1], is presented in Figure 4.2.

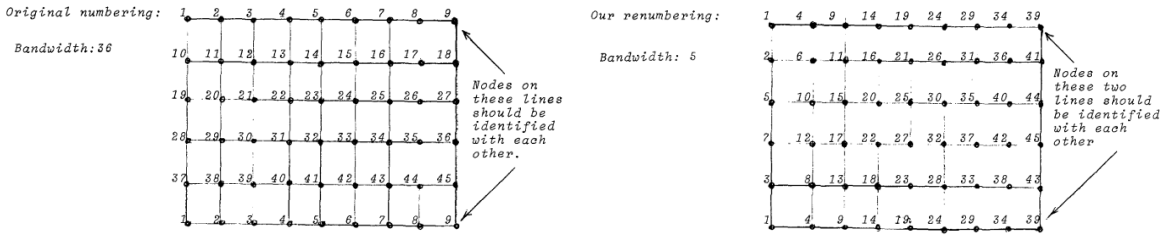


Figure 4.2: Example from the original Cuthill–McKee paper [1].

4.3 Iterative methods for linear systems

Iterative methods enjoy more flexibility than direct methods, because they can be stopped at any point if the residual is deemed sufficiently small. This generally enables to obtain a good solution at a computational cost that is significantly lower than that of direct methods. In this section, we present and study two classes of iterative methods: basic iterative methods based on a splitting of the matrix A, and the so-called Krylov subspace methods.

4.3.1 Basic iterative methods

The basic iterative methods are particular cases of a general *splitting method*. Given a splitting of the matrix of the linear system as $\mathsf{A} = \mathsf{M} - \mathsf{N}$, for a nonsingular matrix $\mathsf{M} \in \mathbf{C}^{n\times n}$ and a matrix $\mathsf{N} \in \mathbf{C}^{n\times n}$, together with an initial guess $\boldsymbol{x}^{(0)}$ of the solution, one step of this general method reads

$$\mathsf{M}\boldsymbol{x}^{(k+1)} = \mathsf{N}\boldsymbol{x}^{(k)} + \boldsymbol{b}. \tag{4.14}$$

For any choice of splitting, the exact solution $\boldsymbol{x}_*$ to the linear system is a fixed point of this iteration, in the sense that if $\boldsymbol{x}^{(0)} = \boldsymbol{x}_*$, then $\boldsymbol{x}^{(k)} = \boldsymbol{x}_*$ for all $k \geqslant 0$. Equation (4.14) is a linear system with matrix M, unknown $\boldsymbol{x}^{(k+1)}$, and right-hand side $\mathsf{N}\boldsymbol{x}^{(k)} + \boldsymbol{b}$. There is a compromise between the cost of a single step and the speed of convergence of the method. In the extreme case where $\mathsf{M} = \mathsf{A}$ and $\mathsf{N} = 0$, the method converges to the exact solution in one step, but performing this step amounts to solving the initial problem. In practice, in order for the method to be useful, the linear system (4.14) should be relatively simple to solve. Concretely, this means that the matrix M should be diagonal, triangular, block diagonal, or block triangular. The error $\boldsymbol{e}^{(k)}$ and residual $\boldsymbol{r}^{(k)}$ at iteration k are defined as follows:

$$\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}_*, \qquad \boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}.$$

Convergence of the splitting method

Before presenting concrete examples of splitting methods, we obtain a necessary and sufficient condition for the convergence of (4.14) for any initial guess $\boldsymbol{x}^{(0)}$.

Proposition 4.10 (Convergence). *The splitting method (4.14) converges for any initial $\boldsymbol{x}^{(0)}$ if and only if $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$. In addition, for any $\varepsilon > 0$ there exists $K > 0$ such that*

$$\forall k \geqslant K, \qquad \|\boldsymbol{e}^{(k)}\| \leqslant \left(\rho(\mathsf{M}^{-1}\mathsf{N}) + \varepsilon\right)^k \|\boldsymbol{e}^{(0)}\|. \tag{4.15}$$

Proof. Let $\boldsymbol{x}_*$ denote the solution to the linear system. Since $\mathsf{M}\boldsymbol{x}_* - \mathsf{N}\boldsymbol{x}_* = \boldsymbol{b}$, we have

$$\mathsf{M}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}_*) = \mathsf{N}(\boldsymbol{x}^{(k)} - \boldsymbol{x}_*).$$

Using the assumption that M is nonsingular, we obtain that the error satisfies the equation

$$\boldsymbol{e}^{(k+1)} = (\mathsf{M}^{-1}\mathsf{N})\boldsymbol{e}^{(k)}.$$

Applying this equality repeatedly, we deduce that

$$\boldsymbol{e}^{(k)} = (\mathsf{M}^{-1}\mathsf{N})\boldsymbol{e}^{(k-1)} = \ldots = (\mathsf{M}^{-1}\mathsf{N})^k \boldsymbol{e}^{(0)}. \tag{4.16}$$

Proof that $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$ is necessary for convergence. We prove the equivalent claim that if $\rho(\mathsf{M}^{-1}\mathsf{N}) \geqslant 1$, then there exists $\boldsymbol{x}^{(0)}$ such that the method is not convergent. Indeed, assume that $\boldsymbol{x}^{(0)} = \boldsymbol{x}_* + \boldsymbol{v}^{(0)}$, where $\boldsymbol{v}^{(0)}$ is the eigenvector of $\mathsf{M}^{-1}\mathsf{N}$ associated with the eigenvalue of largest modulus. Then $\boldsymbol{e}^{(0)} = \boldsymbol{v}^{(0)}$ and the right-hand side of (4.16) does not converge to 0 in the limit as $k \to 0$, because

$$\|(\mathsf{M}^{-1}\mathsf{N})^k \boldsymbol{e}^{(0)}\| = \rho(\mathsf{M}^{-1}\mathsf{N})^k \|\boldsymbol{v}^{(0)}\| \geqslant \|\boldsymbol{v}^{(0)}\|.$$

Thus, the condition $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$ is necessary to ensure convergence for all initial guess $\boldsymbol{x}^{(0)}$.

Proof that $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$ is sufficient for convergence. In order to show that the condition is also sufficient, note that by (4.16)

$$\forall k \geqslant 0, \qquad \|\boldsymbol{e}^{(k)}\| \leqslant \left\|(\mathsf{M}^{-1}\mathsf{N})^k\right\| \|\boldsymbol{e}^{(0)}\|.$$

By Gelfand's formula, proved in Proposition A.10 of Appendix A, it holds that

$$\lim_{k\to\infty} \left\|(\mathsf{M}^{-1}\mathsf{N})^k\right\|^{\frac{1}{k}} = \rho(\mathsf{M}^{-1}\mathsf{N}). \tag{4.17}$$

Therefore, we deduce that if $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$, then $\left\|(\mathsf{M}^{-1}\mathsf{N})^k\right\| \to 0$ and so $\boldsymbol{e}^{(k)} \to \boldsymbol{0}$. In addition, it follows from (4.17) that for all $\varepsilon > 0$ there is $K \in \mathbf{N}$ such that

$$\forall k \geqslant K, \qquad \left\|(\mathsf{M}^{-1}\mathsf{N})^k\right\|^{\frac{1}{k}} \leqslant \rho(\mathsf{M}^{-1}\mathsf{N}) + \varepsilon.$$

Rearranging this inequality gives (4.15).

□

At this point, it is natural to wonder whether there exist sufficient conditions on the matrix A such that the inequality $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$ is satisfied, which is best achieved on a case by case basis. In the next sections, we present four instances of splitting methods. For each of them, we obtain a sufficient condition for convergence. We are particularly interested in the case where

the matrix A is Hermitian and positive definite, which often arises in applications, and in the case where A is strictly row or column diagonally dominant. We recall that a matrix A is said to be row or column diagonally dominant if, respectively,

$$|a_{ii}| \geqslant \sum_{j \neq i} |a_{ij}| \quad \forall i \qquad \text{or} \qquad |a_{jj}| \geqslant \sum_{i \neq j} |a_{ij}| \quad \forall j.$$

Richardson's method

Arguably the simplest splitting of the matrix A is given by $\mathsf{A} = \frac{1}{\omega}\mathsf{I} - \left(\frac{1}{\omega}\mathsf{I} - \mathsf{A}\right)$, for some parameter $\omega \in \mathbf{R}$, which leads to *Richardson's method*:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \omega(\boldsymbol{b} - \mathsf{A}\boldsymbol{x}^{(k)}). \tag{4.18}$$

In this case the spectral radius which enters in the asymptotic rate of convergence is given by

$$\rho(\mathsf{M}^{-1}\mathsf{N}) = \rho\left(\omega\left(\frac{1}{\omega}\mathsf{I} - \mathsf{A}\right)\right) = \rho\left(\mathsf{I} - \omega\mathsf{A}\right)$$

The eigenvalues of the matrix $\mathsf{I} - \omega\mathsf{A}$ are given by $1 - \omega\lambda_i$, where $(\lambda_i)_{1 \leqslant i \leqslant L}$ are the eigenvalues of A. Therefore, the spectral radius is given by

$$\rho(\mathsf{M}^{-1}\mathsf{N}) = \max_{1 \leqslant i \leqslant L} |1 - \omega\lambda_i|.$$

If the eigenvalues of the matrix A do not either (i) all have a positive real part or (ii) all have a negative real part, then

$$\forall \omega \in \mathbf{R}, \qquad \max_{1 \leqslant i \leqslant L} |1 - \omega\lambda_i| \geqslant 1.$$

In other words, by Proposition 4.10, there is for any choice of $\omega \in \mathbf{R}$ some $\boldsymbol{x}^{(0)}$ such that Richardson's method is non-convergent. Therefore, in order for convergence to hold for all $\boldsymbol{x}^{(0)}$, it is necessary that the eigenvalues of A either all have a negative real part, or all have a positive real part. We focus in the next paragraph on the latter case and we also assume, for simplicity, that A is Hermitian.

Case of symmetric positive definite A. If the matrix A is Hermitian and positive definite, the eigenvalues of A are all real and positive, and it is possible to explicitly calculate the optimal value of ω for convergence. In order for convergence to be as fast as possible, the spectral radius of $\mathsf{M}^{-1}\mathsf{N}$ should be as small as possible, in view of Proposition 4.10. Denoting by $\lambda_{\min}$ and $\lambda_{\max}$ the minimum and maximum eigenvalues of A, it is not difficult to show that

$$\rho(\mathsf{M}^{-1}\mathsf{N}) = \max_{1 \leqslant i \leqslant L} |1 - \omega\lambda_i| = \max\Big\{|1 - \omega\lambda_{\min}|, |1 - \omega\lambda_{\max}|\Big\}. \tag{4.19}$$

The right-hand side is minimized $1 - \omega\lambda_{\min} = \omega\lambda_{\max} - 1$, in which case the two arguments of the maximum in (4.19) are equal. From this we deduce the optimal value of ω and the associated spectral radius:

$$\omega_{\text{opt}} = \frac{2}{\lambda_{\max} + \lambda_{\min}}, \qquad \rho_{\text{opt}} = 1 - \frac{2\lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{\kappa_2(\mathsf{A}) - 1}{\kappa_2(\mathsf{A}) + 1}.$$

We observe that the smaller the condition number of the matrix A, the better the asymptotic rate of convergence.

Remark 4.2 (Link to optimization). In the case where $\mathsf{A} \in \mathbf{R}^{n\times n}$ is symmetric and positive definite, the Richardson update (4.18) may be viewed as a step of the steepest descent algorithm, for the function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\mathsf{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega\nabla f(\boldsymbol{x}^{(k)}). \tag{4.20}$$

The gradient of this function is $\nabla f(\boldsymbol{x}) = \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$, and its Hessian matrix is A. Since the Hessian matrix is positive definite, the function is convex and attains its global minimum when ∇f is zero, i.e. when $\mathsf{A}x = \boldsymbol{b}$.

Jacobi's method

In Jacobi's method, the matrix M in the splitting is the diagonal matrix D with the same entries as those of A on the diagonal. We denote by L and U the lower and upper triangular parts of A, without the diagonal. One step of the method reads

$$\mathsf{D}\boldsymbol{x}^{(k+1)} = (\mathsf{D} - \mathsf{A})\boldsymbol{x}^{(k)} + \boldsymbol{b} = -(\mathsf{L} + \mathsf{U})\boldsymbol{x}^{(k)} + \boldsymbol{b} \tag{4.21}$$

Since the matrix D on the left-hand side is diagonal, this linear system with unknown $\boldsymbol{x}^{(k+1)}$ is very simple to solve. The equation (4.21) can be rewritten as

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} = b_1 \\ a_{21}x_1^{(k)} + a_{22}x_2^{(k+1)} + \cdots + a_{2n}x_n^{(k)} = b_2 \\ \vdots \\ a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \cdots + a_{nn}x_n^{(k+1)} = b_n. \end{cases}$$

The updates for each of the entries of $\boldsymbol{x}^{(k+1)}$ are independent, and so the Jacobi method lends itself well to parallel implementation. The computational cost of one iteration, measured in number of floating point operations required, scales as $\mathcal{O}(n^2)$ if A is a full matrix, or $\mathcal{O}(nk)$ if A is a sparse matrix with k nonzero elements per row on average. It is simple to prove the convergence of Jacobi's method is the case where A is diagonally dominant.

Proposition 4.11. *Assume that* A *is strictly (row or column) diagonally dominant. Then it holds that* $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$ *for the Jacobi splitting.*

Proof. Assume that λ is an eigenvalue of $\mathsf{M}^{-1}\mathsf{N}$ and $\boldsymbol{v}$ is the associated unit eigenvector. Then

$$\mathsf{M}^{-1}\mathsf{N}\boldsymbol{v} = \lambda\boldsymbol{v} \quad \Leftrightarrow \quad \mathsf{N}\boldsymbol{v} = \lambda\mathsf{M}\boldsymbol{v} \quad \Leftrightarrow \quad (\mathsf{N} - \lambda\mathsf{M})\boldsymbol{v} = 0.$$

In the case of Jacobi's splitting, this is equivalent to

$$-(\mathsf{L} + \lambda\mathsf{D} + \mathsf{U})\boldsymbol{v} = 0.$$

If $|\lambda| > 1$, then the matrix on the left-hand side of this equation is diagonally dominant and thus invertible (see Exercise 4.9). Therefore $\boldsymbol{v} = 0$, but this is a contradiction because $\boldsymbol{v}$ is vector of unit norm. Consequently, all the eigenvalues are bounded from above strictly by 1 in modulus. □

Gauss–Seidel's method

In Gauss Seidel's method, the matrix M in the splitting is the lower triangular part of A, including the diagonal. One step of the method then reads

$$(\mathsf{L} + \mathsf{D})\boldsymbol{x}^{(k+1)} = -\mathsf{U}\boldsymbol{x}^{(k)} + \boldsymbol{b} \tag{4.22}$$

The system is solved by forward substitution. The equation (4.22) can be rewritten equivalently as

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \cdots + a_{1n}x_n^{(k)} = b_1 \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} + a_{23}x_3^{(k)} + \cdots + a_{2n}x_n^{(k)} = b_2 \\ a_{32}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} + \cdots + a_{3n}x_n^{(k)} = b_3 \\ \vdots \\ a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + a_{n3}x_3^{(k+1)} + \cdots + a_{nn}x_n^{(k+1)} = b_n. \end{cases}$$

Given $\boldsymbol{x}^{(k)}$, the first entry of $\boldsymbol{x}^{(k+1)}$ is obtained from the first equation. Then the value of the second entry is obtained from the second equation, etc. Unlike Jacobi's method, the Gauss–Seidel method is sequential and the entries of $\boldsymbol{x}^{(k+1)}$ cannot be updated in parallel.

It is possible to prove the convergence of the Gauss–Seidel method in particular cases. For example, the method converges if A is strictly diagonally dominant. Proving this, using an approach similar to that in the proof of Proposition 4.11, is the goal of Exercise 4.18. It is also possible to prove convergence when A is Hermitian and positive definite. We show this in the next section for the relaxation method, which generalizes the Gauss–Seidel method.

Relaxation method

The relaxation method generalizes the Gauss–Seidel method. It corresponds to the splitting

$$\mathsf{A} = \left(\frac{\mathsf{D}}{\omega} + \mathsf{L}\right) - \left(\frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}\right). \tag{4.23}$$

When $\omega = 1$, this is simply the Gauss–Seidel splitting. The idea of the relaxation method is that, by letting ω be a parameter that can differ from 1, faster convergence can be achieved. This intuition will be verified later. The equation (4.14) for this splitting can be rewritten equivalently as

$$\begin{cases} a_{11}\big(x_1^{(k+1)} - x_1^{(k)}\big) = -\omega\Big(a_{11}x_1^{(k)} + a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} - b_1\Big) \\ a_{22}\big(x_2^{(k+1)} - x_2^{(k)}\big) = -\omega\Big(a_{21}x_1^{(k+1)} + a_{22}x_2^{(k)} + \cdots + a_{2n}x_n^{(k)} - b_2\Big) \\ \vdots \\ a_{nn}\big(x_n^{(k+1)} - x_n^{(k)}\big) = -\omega\Big(a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + \cdots + a_{nn}x_n^{(k)} - b_n\Big). \end{cases}$$

The coefficient on the right-hand side is larger than in the Gauss–Seidel method if $\omega > 1$, and smaller if $\omega < 1$. These regimes are called *over-relaxation* and *under-relaxation*, respectively.

To conclude this section, we establish a sufficient condition for the convergence of the relaxation method, and also of the Gauss–Seidel method as a particular case when $\omega = 1$, when the matrix A is Hermitian and positive definite. To this end, we begin by showing the following preparatory result, which concerns a general splitting $\mathsf{A} = \mathsf{M} - \mathsf{N}$.

Proposition 4.12. *Let A be Hermitian and positive definite. If the Hermitian matrix $\mathsf{M}^* + \mathsf{N}$ is positive definite, then $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$.*

Proof. First, notice that $\mathsf{M}^* + \mathsf{N}$ is indeed Hermitian because

$$(\mathsf{M}^* + \mathsf{N})^* = \mathsf{M} + \mathsf{N}^* = \mathsf{A} + \mathsf{N} + \mathsf{N}^*.$$

We will show that $\|\mathsf{M}^{-1}\mathsf{N}\|_{\mathsf{A}} < 1$, where $\|\bullet\|_{\mathsf{A}}$ is the matrix norm induced by the following norm on vectors:

$$\|\boldsymbol{x}\|_{\mathsf{A}} := \sqrt{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}.$$

Showing that this indeed defines a vector norm is the goal of Exercise 4.11. Since $\mathsf{N} = \mathsf{M} - \mathsf{A}$, it holds that

$$\|\mathsf{M}^{-1}\mathsf{N}\|_{\mathsf{A}} = \|\mathsf{I} - \mathsf{M}^{-1}\mathsf{A}\|_{\mathsf{A}},$$

and so

$$\|\mathsf{M}^{-1}\mathsf{N}\|_{\mathsf{A}} = \sup\Big\{\|\boldsymbol{x} - \mathsf{M}^{-1}\mathsf{A}\boldsymbol{x}\|_{\mathsf{A}} \colon \|\boldsymbol{x}\|_{\mathsf{A}} \leqslant 1\Big\}.$$

Take $\boldsymbol{x} \in \mathbf{C}^n$ with $\|\boldsymbol{x}\|_{\mathsf{A}} \leqslant 1$ and let $\boldsymbol{y} = \mathsf{M}^{-1}\mathsf{A}\boldsymbol{x}$. We calculate

$$\begin{aligned} \|\boldsymbol{x} - \mathsf{M}^{-1}\mathsf{A}\boldsymbol{x}\|_{\mathsf{A}}^2 &= \boldsymbol{x}^*\mathsf{A}\boldsymbol{x} - \boldsymbol{y}^*\mathsf{A}\boldsymbol{x} - \boldsymbol{x}^*\mathsf{A}\boldsymbol{y} + \boldsymbol{y}^*\mathsf{A}\boldsymbol{y} \\ &= \boldsymbol{x}^*\mathsf{A}\boldsymbol{x} - \boldsymbol{y}^*\mathsf{M}\mathsf{M}^{-1}\mathsf{A}\boldsymbol{x} - (\mathsf{M}^{-1}\mathsf{A}\boldsymbol{x})^*\mathsf{M}^*\boldsymbol{y} + \boldsymbol{y}^*\mathsf{A}\boldsymbol{y} \\ &= \boldsymbol{x}^*\mathsf{A}\boldsymbol{x} - \boldsymbol{y}^*\mathsf{M}\boldsymbol{y} - \boldsymbol{y}^*\mathsf{M}^*\boldsymbol{y} + \boldsymbol{y}^*(\mathsf{M} - \mathsf{N})\boldsymbol{y} \\ &= \boldsymbol{x}^*\mathsf{A}\boldsymbol{x} - \boldsymbol{y}^*(\mathsf{M}^* + \mathsf{N})\boldsymbol{y} \leqslant 1 - \boldsymbol{y}^*(\mathsf{M}^* + \mathsf{N})\boldsymbol{y} < 1, \end{aligned}$$

where we used in the last inequality the assumption that $\mathsf{M}^* + \mathsf{N}$ is positive definite. This inequality holds true for all $\boldsymbol{x} \in \mathbf{C}^n$ with $\|\boldsymbol{x}\|_{\mathsf{A}} = 1$, and so we conclude that $\|\mathsf{M}^{-1}\mathsf{N}\|_{\mathsf{A}} < 1$, which implies that $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$. □

As a corollary, we obtain a sufficient condition for the convergence of the relaxation method.

Corollary 4.13. *Assume that* A *is Hermitian and positive definite. Then the relaxation method converges if* $\omega \in (0, 2)$.

Proof. For the relaxation method, we have

$$\mathsf{M} + \mathsf{N}^* = \left(\frac{\mathsf{D}}{\omega} + \mathsf{L}\right) + \left(\frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}\right)^*.$$

Since A is Hermitian, it holds that $\mathsf{D}^* = \mathsf{D}$ and $\mathsf{U}^* = \mathsf{L}$. Therefore,

$$\mathsf{M} + \mathsf{N}^* = \frac{2-\omega}{\omega}\mathsf{D}.$$

The diagonal elements of D are all positive, because A is positive definite. (Indeed, if there was an index i such that $d_{ii} \leqslant 0$, then it would hold that $\boldsymbol{e}_i^T \mathsf{A} \boldsymbol{e}_i = d_{ii} \leqslant 0$, contradicting the assumption that A is positive definite.) We deduce that $\mathsf{M} + \mathsf{N}^*$ is positive definite if and only if $\omega \in (0, 2)$. We can then conclude the proof by using Proposition 4.12. □

Note that Corollary 4.13 implies as a particular case the convergence of the Gauss–Seidel method when A is Hermitian and positive definite. The condition $\omega \in (0, 2)$ is in fact necessary for the convergence of the relaxation method, not only in the case of a Hermitian positive definite matrix A but in general.

Proposition 4.14 (Necessary condition for the convergence of the relaxation method)**.** *Let* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *be an invertible matrix, and let* $\mathsf{A} = \mathsf{M}_\omega - \mathsf{N}_\omega$ *denote the splitting of the relaxation method with parameter* ω*. It holds that*

$$\forall \omega \neq 0, \qquad \rho(\mathsf{M}_\omega^{-1}\mathsf{N}_\omega) \geqslant |\omega - 1|.$$

Proof. We recall the following facts:

- the determinant of a product of matrices is equal to the product of the determinants.
- the determinant of a triangular matrix is equal to the product of its diagonal entries;
- the determinant of a matrix is equal to the product of its eigenvalues, to the power of their algebraic multiplicity. This can be shown from the previous two items, by passing to the Jordan normal form.

Therefore, we have that

$$\det(\mathsf{M}_\omega^{-1}\mathsf{N}_\omega) = \det(\mathsf{M}_\omega)^{-1}\det(\mathsf{N}_\omega) = \frac{\det\left(\frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}\right)}{\det\left(\frac{\mathsf{D}}{\omega} + \mathsf{L}\right)} = (1-\omega)^n.$$

Since the determinant on the left-hand side is the product of the eigenvalues of $\mathsf{M}_\omega^{-1}\mathsf{N}_\omega$, it is bounded from above in modulus by $\rho(\mathsf{M}_\omega^{-1}\mathsf{N}_\omega)^n$, and so we deduce $\rho(\mathsf{M}_\omega^{-1}\mathsf{N}_\omega)^n \geqslant |1-\omega|^n$. The statement then follows by taking the n-th root. □

Comparison between Jacobi and Gauss–Seidel for tridiagonal matrices

For tridiagonal matrices, the convergence rate of the Jacobi and Gauss–Seidel methods satisfy an explicit relation, which we prove in this section. We denote the Jacobi and Gauss–Seidel splittings by $\mathsf{M}_\mathcal{J} - \mathsf{N}_\mathcal{J}$ and $\mathsf{M}_\mathcal{G} - \mathsf{N}_\mathcal{G}$, respectively, and use the following notation for the entries of the matrix A:

$$\begin{pmatrix} a_1 & b_1 & & \\ c_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix}.$$

Before presenting and proving the result, notice that for any $\mu \neq 0$ it holds that

$$\begin{pmatrix} \mu & & & \\ & \mu^2 & & \\ & & \ddots & \\ & & & \mu^n \end{pmatrix} \mathsf{A} \begin{pmatrix} \mu^{-1} & & & \\ & \mu^{-2} & & \\ & & \ddots & \\ & & & \mu^{-n} \end{pmatrix} = \begin{pmatrix} a_1 & \mu^{-1}b_1 & & \\ \mu c_1 & \ddots & \ddots & \\ & \ddots & \ddots & \mu^{-1}b_{n-1} \\ & & \mu c_{n-1} & a_n \end{pmatrix}. \tag{4.24}$$

Proposition 4.15. *Assume that* A *is tridiagonal with nonzero diagonal elements, so that both* $\mathsf{M}_\mathcal{J} = \mathsf{D}$ *and* $\mathsf{M}_\mathcal{G} = \mathsf{L} + \mathsf{D}$ *are invertible. Then*

$$\rho(\mathsf{M}_\mathcal{G}^{-1}\mathsf{N}_\mathcal{G}) = \rho(\mathsf{M}_\mathcal{J}^{-1}\mathsf{N}_\mathcal{J})^2$$

Proof. If λ is an eigenvalue of $\mathsf{M}_\mathcal{G}^{-1}\mathsf{N}_\mathcal{G}$ with associated unit eigenvector $\boldsymbol{v}$, then

$$\mathsf{M}_\mathcal{G}^{-1}\mathsf{N}_\mathcal{G}\boldsymbol{v} = \lambda\boldsymbol{v} \quad\Leftrightarrow\quad \mathsf{N}_\mathcal{G}\boldsymbol{v} = \lambda\mathsf{M}_\mathcal{G}\boldsymbol{v} \quad\Leftrightarrow\quad (\mathsf{N}_\mathcal{G} - \lambda\mathsf{M}_\mathcal{G})\boldsymbol{v} = 0.$$

For fixed λ, there exists a nontrivial solution $\boldsymbol{v}$ to the last equation if and only if

$$p_\mathcal{G}(\lambda) := \det(\mathsf{N}_\mathcal{G} - \lambda\mathsf{M}_\mathcal{G}) = \det(-\lambda\mathsf{L} - \lambda\mathsf{D} - \mathsf{U}) = 0.$$

Likewise, λ is an eigenvalue of $\mathsf{M}_\mathcal{J}^{-1}\mathsf{N}_\mathcal{J}$ if and only if

$$p_\mathcal{J}(\lambda) := \det(\mathsf{N}_\mathcal{J} - \lambda\mathsf{M}_\mathcal{J}) = \det(-\mathsf{L} - \lambda\mathsf{D} - \mathsf{U}) = 0.$$

Now notice that, for $\lambda \neq 0$,

$$p_\mathcal{G}(\lambda^2) = \det\big(-\lambda^2\mathsf{L} - \lambda^2\mathsf{D} - \mathsf{U}\big) = \lambda^n \det\big(-\lambda\mathsf{L} - \lambda\mathsf{D} - \lambda^{-1}\mathsf{U}\big).$$

Applying (4.24) with $\mu = \lambda \neq 0$, we deduce

$$p_\mathcal{G}(\lambda^2) = \lambda^n \det\big(-\mathsf{L} - \lambda\mathsf{D} - \mathsf{U}\big) = \lambda^n p_\mathcal{J}(\lambda)$$

It is clear that this relation is true also if $\lambda = 0$. Consequently, it holds that if λ is an eigenvalue of the matrix $\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}}$ then λ^2 is an eigenvalue of $\mathsf{M}_{\mathcal{G}}^{-1}\mathsf{N}_{\mathcal{G}}$. Conversely, if λ is a nonzero eigenvalue of $\mathsf{M}_{\mathcal{G}}^{-1}\mathsf{N}_{\mathcal{G}}$, then the two square roots of λ are eigenvalues of $\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}}$. □

If a matrix A is tridiagonal and Toeplitz, i.e. if it is of the form

$$\begin{pmatrix} a & b & & \\ c & \ddots & \ddots & \\ & \ddots & \ddots & b \\ & & c & a \end{pmatrix},$$

then it is possible to prove that the eigenvalues of A are given by

$$\lambda_k = a + 2\sqrt{bc}\cos\left(\frac{k\pi}{n+1}\right), \qquad k = 1, \ldots, n. \tag{4.25}$$

In this case, the spectral radius of $\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}}$ can be determined explicitly.

Monitoring the convergence

In practice, we have access to the residual $\boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}$ at each iteration, but not to the error $\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}_*$, as calculating the latter would require to know the exact solution of the problem. Nevertheless, the two are related by the equation

$$\boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{e}^{(k)} \quad \Leftrightarrow \quad \boldsymbol{e}^{(k)} = \mathsf{A}^{-1}\boldsymbol{r}^{(k)}.$$

Therefore, it holds that $\|\boldsymbol{e}^{(k)}\| \leqslant \|\mathsf{A}^{-1}\|\|\boldsymbol{r}^{(k)}\|$. Likewise, the relative error satisfies

$$\frac{\|\boldsymbol{e}^{(k)}\|}{\|\boldsymbol{x}_*\|} = \frac{\|\mathsf{A}^{-1}\boldsymbol{r}^{(k)}\|}{\|\mathsf{A}^{-1}\boldsymbol{b}\|},$$

and since $\|\boldsymbol{b}\| = \|\mathsf{A}\mathsf{A}^{-1}\boldsymbol{b}\| \leqslant \|\mathsf{A}\|\|\mathsf{A}^{-1}\boldsymbol{b}\|$, we deduce

$$\frac{\|\boldsymbol{e}^{(k)}\|}{\|\boldsymbol{x}_*\|} \leqslant \kappa(\mathsf{A})\frac{\|\boldsymbol{r}^{(k)}\|}{\|\boldsymbol{b}\|}.$$

The fraction on the right-hand side is the *relative residual*. If the system is well conditioned, that is if $\kappa(A)$ is close to one, then controlling the relative residual enables a good control of the relative error.

Stopping criterion

In practice, several criteria can be employed in order to decide when to stop iterating. Given a small number ε (unrelated to the machine epsilon in Chapter 1), the following alternatives are available:

- Stop when $\|\boldsymbol{r}^{(k)}\| \leqslant \varepsilon$. The downside of this approach is that it is not *scaling invariant*: when used for solving the following rescaled system

$$k\mathsf{A}\boldsymbol{x} = k\boldsymbol{b}, \qquad k \neq 1,$$

a splitting method with rescaled initial guess $k\boldsymbol{x}^{(0)}$ will require a number of iterations that depends on k: fewer if $k \ll 1$ and more if $k \gg 1$. In practice, controlling the relative residual and the relative error is often preferable.

- Stop when $\|\boldsymbol{r}^{(k)}\|/\|\boldsymbol{r}^{(0)}\| \leqslant \varepsilon$. This criterion is scaling invariant, but the number of iterations is dependent on the quality of the initial guess $\boldsymbol{x}^{(0)}$.
- Stop when $\|\boldsymbol{r}^{(k)}\|/\|\boldsymbol{b}\|$. This criterion is generally the best, because it is both scaling invariant and the quality of the final iterate is independent of that of the initial guess.

4.3.2 The conjugate gradient method

As already mentioned in Remark 4.2, when the matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ in the linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ is symmetric and positive definite, the system can be interpreted as a minimization problem for the function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\mathsf{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}. \tag{4.26}$$

The fact that the exact solution $\boldsymbol{x}_*$ to the linear system is the unique minimizer of this function appears clearly when rewriting f as follows:

$$f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_*)^T\mathsf{A}(\boldsymbol{x} - \boldsymbol{x}_*) - \frac{1}{2}\boldsymbol{x}_*^T\mathsf{A}\boldsymbol{x}_*. \tag{4.27}$$

The second term is constant with $\boldsymbol{x}$, and the first term is strictly positive if $\boldsymbol{x} - \boldsymbol{x}_* \neq \boldsymbol{0}$, because A is positive definite. We saw that Richardson's method can be interpreted as a steepest descent with fixed step size,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega\nabla f(\boldsymbol{x}^{(k)}).$$

In this section, we will present and study other methods for solving the linear system (4.1) which can be viewed as optimization methods. Since A is symmetric, it is diagonalizable and the function f can be rewritten as

$$\begin{aligned} f(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_*)^T\mathsf{Q}\mathsf{D}\mathsf{Q}^T(\boldsymbol{x} - \boldsymbol{x}_*) - \frac{1}{2}\boldsymbol{x}_*^T\mathsf{A}\boldsymbol{x}_* \\ &= \frac{1}{2}(\mathsf{Q}^T\boldsymbol{e})^T\mathsf{D}(\mathsf{Q}^T\boldsymbol{e}) - \frac{1}{2}\boldsymbol{x}_*^T\mathsf{A}\boldsymbol{x}_*, \qquad \boldsymbol{e} = \boldsymbol{x} - \boldsymbol{x}_*. \end{aligned}$$

Therefore, we have that

$$f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n}\lambda_i\eta_i^2 - \frac{1}{2}\boldsymbol{x}_*^T\mathsf{A}\boldsymbol{x}_*, \qquad \boldsymbol{\eta} = \boldsymbol{Q}^T(\boldsymbol{x} - \boldsymbol{x}_*),$$

where $(\lambda_i)_{1\leqslant i\leqslant n}$ are the diagonal entries of D. This shows that f is a paraboloid after a change of coordinates.

Steepest descent method

The steepest descent method is more general than Richardson's method in the sense that the step size changes from iteration to iteration and the method is not restricted to quadratic functions of the form (4.26). Each iteration is of the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega_k \nabla f(\boldsymbol{x}^{(k)}).$$

It is natural to wonder whether the step size ω_k can be fixed in such a way that $f(\boldsymbol{x}^{(k+1)})$ is as small as possible. For the case of the quadratic function (4.26), this value of ω_k can be calculated explicitly for a general search direction $\boldsymbol{d}$, and in particular also when $\boldsymbol{d} = \nabla f(\boldsymbol{x}^{(k)})$. We calculate that

$$\begin{aligned} f(\boldsymbol{x}^{(k+1)}) &= f\Big(\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}\Big) = \frac{1}{2}\big(\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}\big)^T \mathsf{A}\big(\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}\big) - \big(\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}\big)^T \boldsymbol{b} \\ &= f\Big(\boldsymbol{x}^{(k)}\Big) + \frac{\omega_k^2}{2} \boldsymbol{d}^T \mathsf{A} \boldsymbol{d} - \omega_k \boldsymbol{d}^T \boldsymbol{r}^{(k)}. \end{aligned} \tag{4.28}$$

When viewed as a function of the real parameter ω_k, the right-hand side is a convex quadratic function. It is minimized when its derivative is equal to zero, i.e. when

$$\omega_k \boldsymbol{d}^T \mathsf{A} \boldsymbol{d} - \boldsymbol{d}^T (\mathsf{A}\boldsymbol{x}_k - \boldsymbol{b}) = 0 \qquad \Rightarrow \qquad \omega_k = \frac{\boldsymbol{d}^T \boldsymbol{r}^{(k)}}{\boldsymbol{d}^T \mathsf{A} \boldsymbol{d}}. \tag{4.29}$$

The steepest descent algorithm with step size obtained from this equation is summarized in Algorithm 3 below. By construction, the function value $f(\boldsymbol{x}^{(k)})$ is nonincreasing with k, which is equivalent to saying that the error $\boldsymbol{x} - \boldsymbol{x}_*$ is nonincreasing in the norm $\boldsymbol{x} \mapsto \sqrt{\boldsymbol{x}^T \mathsf{A} \boldsymbol{x}}$. In order to quantify more precisely the decrease of the error in this norm, we introduce the notation

$$E_k = \|\boldsymbol{x} - \boldsymbol{x}_*\|_{\mathsf{A}}^2 := (\boldsymbol{x}^{(k)} - \boldsymbol{x}_*)^T \mathsf{A} (\boldsymbol{x}^{(k)} - \boldsymbol{x}_*) = (\mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})^T \mathsf{A}^{-1} (\mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}).$$

We begin by showing the following auxiliary lemma.

Lemma 4.16 (Kantorovich inequality)**.** *Let $\mathsf{A} \in \mathbf{R}^{n \times n}$ be a symmetric and positive definite matrix, and let $\lambda_0 \leqslant \ldots \leqslant \lambda_n$ denote its eigenvalues. Then for all nonzero $\boldsymbol{z} \in \mathbf{R}^n$ it holds that*

$$\frac{(\boldsymbol{z}^T \boldsymbol{z})^2}{(\boldsymbol{z}^T \mathsf{A} \boldsymbol{z})(\boldsymbol{z}^T \mathsf{A}^{-1} \boldsymbol{z})} \geqslant \frac{4\lambda_1 \lambda_n}{(\lambda_1 + \lambda_n)^2}.$$

Proof. By the AM-GM (arithmetic mean-geometric mean) inequality, it holds for all $t > 0$ that

$$\begin{aligned} \sqrt{(\boldsymbol{z}^T \mathsf{A} \boldsymbol{z})(\boldsymbol{z}^T \mathsf{A}^{-1} \boldsymbol{z})} &= \sqrt{(t\boldsymbol{z}^T \mathsf{A} \boldsymbol{z})(t^{-1} \boldsymbol{z}^T \mathsf{A}^{-1} \boldsymbol{z})} \leqslant \frac{1}{2}\left(t\boldsymbol{z}^T \mathsf{A} \mathsf{z} + \frac{1}{t} \boldsymbol{z}^T \mathsf{A}^{-1} \mathsf{z}\right) \\ &= \frac{1}{2} \boldsymbol{z}^T \left(t\mathsf{A} + \frac{1}{t} \mathsf{A}^{-1}\right) \boldsymbol{z}. \end{aligned}$$

The matrix on the right-hand side is also symmetric and positive definite, with eigenvalues equal to $t\lambda_i + (t\lambda_i)^{-1}$. Therefore, we deduce

$$\forall t \geqslant 0, \qquad \sqrt{(\boldsymbol{z}^T \mathsf{A} \boldsymbol{z})(\boldsymbol{z}^T \mathsf{A}^{-1} \boldsymbol{z})} \leqslant \frac{1}{2} \left(\max_{i \in \{1, \dots, n\}} t\lambda_i + (t\lambda_i)^{-1} \right) \boldsymbol{z}^T \boldsymbol{z}. \tag{4.30}$$

The function $x \mapsto x + x^{-1}$ is convex, and so over any closed interval $[x_{\min}, x_{\max}]$ it attains its maximum either at $x_{\min}$ or at $x_{\max}$. Consequently, it holds that

$$\left(\max_{i \in \{1, \dots, n\}} t\lambda_i + (t\lambda_i)^{-1} \right) = \max \left\{ t\lambda_1 + \frac{1}{t\lambda_1}, t\lambda_n + \frac{1}{t\lambda_n} \right\}.$$

In order to obtain the best possible bound from (4.30), we should let t be such that the maximum is minimized, which occurs when the two arguments of the maximum are equal:

$$t\lambda_1 + \frac{1}{t\lambda_1} = t\lambda_n + \frac{1}{t\lambda_n} \qquad \Rightarrow \qquad t = \frac{1}{\sqrt{\lambda_1 \lambda_n}}.$$

For this value of t, the maximum in (4.30) is equal to

$$\sqrt{\frac{\lambda_1}{\lambda_n}} + \sqrt{\frac{\lambda_n}{\lambda_1}}.$$

By substituting this expression in (4.30) and rearranging, we obtain the statement. □

We are now able to prove the convergence of the steepest descent method.

Theorem 4.17 (Convergence of the steepest descent method)**.** *It holds that*

$$E_{k+1} \leqslant \left(\frac{\kappa_2(\mathsf{A}) - 1}{\kappa_2(\mathsf{A}) + 1} \right)^2 E_k.$$

Proof. Substituting $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}$ in the expression for E_{k+1}, we obtain

$$\begin{aligned} E_{k+1} &= (\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d} - \boldsymbol{x}_*)^T \mathsf{A} (\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d} - \boldsymbol{x}_*) \\ &= E_k - 2\omega_k \boldsymbol{d}^T \boldsymbol{r}^{(k)} + \omega_k^2 \boldsymbol{d}^T \mathsf{A} \boldsymbol{d} \\ &= E_k - \frac{(\boldsymbol{d}^T \boldsymbol{d})^2}{\boldsymbol{d}^T \mathsf{A} \boldsymbol{d}} = \left(1 - \frac{(\boldsymbol{d}^T \boldsymbol{d})^2}{(\boldsymbol{d}^T \mathsf{A} \boldsymbol{d})(\boldsymbol{d}^T \mathsf{A}^{-1} \boldsymbol{d})} \right) E_k, \end{aligned}$$

Using the Kantorovich inequality, we have

$$E_{k+1} \leqslant \left(1 - \frac{4\lambda_1 \lambda_n}{(\lambda_1 + \lambda_n)^2} \right) E_k \leqslant \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n} \right)^2 E_k = \left(\frac{\kappa_2(\mathsf{A}) - 1}{\kappa_2(\mathsf{A}) + 1} \right)^2 E_k.$$

We immediately deduce the statement from this inequality. □

Algorithm 3 Steepest descent method

Pick ε and initial $\boldsymbol{x}$
$\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
while $\|\boldsymbol{r}\| \geqslant \varepsilon\|\boldsymbol{b}\|$ **do**
    $\boldsymbol{d} \leftarrow \boldsymbol{r}$
    $\omega \leftarrow \boldsymbol{d}^T\boldsymbol{r}/\boldsymbol{d}^T\mathsf{A}\boldsymbol{d}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} - \omega\boldsymbol{d}$
    $\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
end while

Remark 4.3. The attentive reader will have noticed that the rate of convergence rate of the steepest descent method is not apparently better than that of Richardson's method with optimal ω; in both cases, some upper bound on the norm of the error is multiplied by

$$\frac{\kappa_2(\mathsf{A}) - 1}{\kappa_2(\mathsf{A}) + 1}$$

at each iteration. For the steepest descent method, this rate of convergence is always guaranteed, but for Richardson's method, this rate of convergence holds only for the optimal value of ω, which itself depends on the condition number $\kappa_2(\mathsf{A})$ and is computationally expensive to approximate. Indeed, as we shall see in Chapter 6, the simplest method for calculating the smallest eigenvalue of a matrix, which is necessary for estimating the condition number, requires to solve linear systems with matrix A.

Preconditioned steepest descent

We observe from Theorem 4.17 that the convergence of the steepest descent method is faster when the condition number of the matrix A is low. This naturally leads to the following question: can we reformulate the minimization of $f(\boldsymbol{x})$ in (4.26) as another optimization problem which is of the same form but involves a matrix with a lower condition number, thereby providing scope for faster convergence? In order to answer this question, we consider a linear change of coordinates $\boldsymbol{y} = \mathsf{T}^{-1}\boldsymbol{x}$, where T is an invertible matrix, and we define

$$\tilde{f}(\boldsymbol{y}) = f(\mathsf{T}\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^T(\mathsf{T}^T\mathsf{A}\mathsf{T})\boldsymbol{y} - (\mathsf{T}^T\boldsymbol{b})^T\boldsymbol{y}. \tag{4.31}$$

This function is of the same form as f in (4.26), with the matrix $\tilde{\mathsf{A}} := \mathsf{T}^T\mathsf{A}\mathsf{T}$ instead of A and the vector $\tilde{\boldsymbol{b}} := \mathsf{T}^T\boldsymbol{b}$ instead of $\boldsymbol{b}$. Its minimizer is $\boldsymbol{y}_* = \mathsf{T}^{-1}\boldsymbol{x}_*$. The steepest descent algorithm can be applied to (4.31) and, from an approximation $\boldsymbol{y}^{(k)}$ of the minimizer $\boldsymbol{y}_*$, an approximation $\boldsymbol{x}^{(k)}$ of $\boldsymbol{x}_*$ is obtained by the change of variable $\boldsymbol{x}^{(k)} = \mathsf{T}\boldsymbol{y}^{(k)}$. This approach is called *preconditioning*. By Theorem 4.17, the steepest descent method satisfies the following error estimate when applied to the function (4.31):

$$E_{k+1} \leqslant \left(\frac{\kappa_2(\mathsf{T}^T\mathsf{A}\mathsf{T})-1}{\kappa_2(\mathsf{T}^T\mathsf{A}\mathsf{T})+1}\right)^2 E_k, \qquad E_k = (\boldsymbol{y}^{(k)}-\boldsymbol{y}_*)^T\tilde{\mathsf{A}}(\boldsymbol{y}^{(k)}-\boldsymbol{y}_*),$$
$$= (\boldsymbol{x}^{(k)}-\boldsymbol{x}_*)^T\mathsf{A}(\boldsymbol{x}^{(k)}-\boldsymbol{x}_*).$$

Consequently, the convergence is faster than that of the usual steepest descent method if $\kappa_2(\mathsf{T}^T\mathsf{A}\mathsf{T}) < \kappa_2(\mathsf{A})$. The optimal change of coordinates is given by $\mathsf{T} = \mathsf{C}^{-T}$, where C is the factor of the Cholesky factorization of A as $\mathsf{C}\mathsf{C}^T$. Indeed, in this case

$$\mathsf{T}^T\mathsf{A}\mathsf{T} = \mathsf{C}^{-1}\mathsf{C}\mathsf{C}^T\mathsf{C}^{-T} = \mathsf{I} \qquad \Rightarrow \qquad \kappa_2(\mathsf{T}^T\mathsf{A}\mathsf{T}) = 1,$$

and the method converges in a single iteration! However, this iteration amounts to solving the linear system by direct Cholesky factorization of A. In practice, it is usual to define T from an approximation of the Cholesky factorization, such as the *incomplete Cholesky factorization*.

To conclude this section, we demonstrate that the change of variable from $\boldsymbol{x}$ to $\boldsymbol{y}$ need not be performed explicitly in practice. Indeed, one step of the steepest descent algorithm applied to function $\tilde{f}$ reads

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} - \tilde{\omega}_k(\tilde{\mathsf{A}}\boldsymbol{y}^{(k)}-\tilde{\boldsymbol{b}}), \qquad \tilde{\omega}_k = \frac{(\tilde{\mathsf{A}}\boldsymbol{y}^{(k)}-\tilde{\boldsymbol{b}})^T(\tilde{\mathsf{A}}\boldsymbol{y}^{(k)}-\tilde{\boldsymbol{b}})}{(\tilde{\mathsf{A}}\boldsymbol{y}^{(k)}-\tilde{\boldsymbol{b}})^T\tilde{\mathsf{A}}(\tilde{\mathsf{A}}\boldsymbol{y}^{(k)}-\tilde{\boldsymbol{b}})}.$$

Letting $\boldsymbol{x}^{(k)} = \mathsf{T}\boldsymbol{y}^{(k)}$, this equation can be rewritten as the following iteration:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \tilde{\omega}_k\boldsymbol{d}_k, \qquad \tilde{\omega}_k = \frac{\boldsymbol{d}_k^T\boldsymbol{r}^{(k)}}{\boldsymbol{d}_k^T\mathsf{A}\boldsymbol{d}_k}, \qquad \boldsymbol{d}_k = \mathsf{T}\mathsf{T}^T(\mathsf{A}\boldsymbol{x}^{(k)}-\boldsymbol{b}).$$

A comparison with (4.29) shows that the step size $\tilde{\omega}_k$ is such that $f(\boldsymbol{x}^{(k+1)})$ is minimized. This reasoning shows that the preconditioned conjugate gradient method amounts to choosing the direction $\boldsymbol{d}_k = \mathsf{T}\mathsf{T}^T\boldsymbol{r}^{(k)}$ at each iteration, instead of just $\boldsymbol{r}^{(k)}$, as is apparent in Algorithm 4. It is simple to check that $-\boldsymbol{d}_k$ is a descent direction for f:

$$-\nabla f(\boldsymbol{x})^T\Big(\mathsf{T}\mathsf{T}^T(\mathsf{A}\boldsymbol{x}-\boldsymbol{b})\Big) = -\Big(\mathsf{T}^T(\mathsf{A}\boldsymbol{x}-\boldsymbol{b})\Big)^T\Big(\mathsf{T}^T(\mathsf{A}\boldsymbol{x}-\boldsymbol{b})\Big) \leqslant 0.$$

Algorithm 4 Preconditioned steepest descent method

Pick ε, invertible T and initial $\boldsymbol{x}$
$\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x}-\boldsymbol{b}$
while $\|\boldsymbol{r}\| \geqslant \varepsilon\|\boldsymbol{b}\|$ **do**
    $\boldsymbol{d} \leftarrow \mathsf{T}\mathsf{T}^T\boldsymbol{r}$
    $\omega \leftarrow \boldsymbol{d}^T\boldsymbol{r}/\boldsymbol{d}^T\mathsf{A}\boldsymbol{d}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x}-\omega\boldsymbol{d}$
    $\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x}-\boldsymbol{b}$
end while

Conjugate directions method

Definition 4.6 (Conjugate directions)**.** Let A be a symmetric positive definite matrix. Two vectors $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are called A-orthogonal or conjugate with respect to A if $\boldsymbol{d}_1^T \mathsf{A} \boldsymbol{d}_2 = 0$, i.e. if they are orthogonal for the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathsf{A}} = \boldsymbol{x}^T \mathsf{A} \boldsymbol{y}$.

Assume that $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1}$ are n pairwise A-orthogonal nonzero directions. By Exercise 4.19, these vectors are linearly independent, and so they form a basis of $\mathbf{R}^n$. Consequently, for any initial guess $\boldsymbol{x}^{(0)}$, the vector $\boldsymbol{x}^{(0)} - \boldsymbol{x}_*$, where $\boldsymbol{x}_*$ is the solution to the linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$, can be decomposed as

$$\boldsymbol{x}^{(0)} - \boldsymbol{x}_* = \alpha_0 \boldsymbol{d}_0 + \cdots + \alpha_{n-1} \boldsymbol{d}_{n-1}.$$

Taking the $\langle \bullet, \bullet \rangle_{\mathsf{A}}$ inner product of both sides with $\boldsymbol{d}_k$, with $k \in \{0, \ldots, n-1\}$, we obtain an expression for the scalar coefficient α_k:

$$\alpha_k = \frac{\boldsymbol{d}_k^T \mathsf{A}(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k} = \frac{\boldsymbol{d}_k^T (\mathsf{A}\boldsymbol{x}^{(0)} - \boldsymbol{b})}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k}.$$

Therefore, calculating the expression of the coefficient does not require to know the exact solution $\boldsymbol{x}_*$, but only the residual $\boldsymbol{r}^{(0)}$! Given conjugate directions, the exact solution can be obtained as

$$\boldsymbol{x}_* = \boldsymbol{x}^{(0)} - \sum_{k=0}^{n-1} \alpha_k \boldsymbol{d}_k, \qquad \alpha_k = \frac{\boldsymbol{d}_k^T \boldsymbol{r}^{(0)}}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k}. \tag{4.32}$$

If $\boldsymbol{x}^{(0)} = \mathbf{0}$, then $\boldsymbol{r}^{(0)} = -\boldsymbol{b}$ and this equations gives that

$$\boldsymbol{x}_* = \sum_{k=0}^{n-1} \frac{\boldsymbol{d}_k^T \boldsymbol{b}}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k} \boldsymbol{d}_k = \left(\sum_{k=0}^{n-1} \frac{\boldsymbol{d}_k \boldsymbol{d}_k^T}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k} \right) \boldsymbol{b},$$

which implies that that the inverse of A is given by

$$\mathsf{A}^{-1} = \sum_{k=0}^{n-1} \frac{\boldsymbol{d}_k \boldsymbol{d}_k^T}{\nu_k}, \qquad \nu_k = \boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k.$$

The conjugate directions method is illustrated in Algorithm 5. Its implementation is very similar to that of the steepest descent method, the only difference being that the descent direction at iteration k is given by $\boldsymbol{d}_k$ instead of $\boldsymbol{r}^{(k)}$. In particular, the step size at each iteration is such that $f(\boldsymbol{x}^{(k+1)})$ is minimized.

Algorithm 5 Conjugate directions method

Assuming $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1}$ are given.
Pick initial $\boldsymbol{x}^{(0)}$
for k in $\{0, \ldots, n-1\}$ **do**
    $\boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}$
    $\omega_k = \boldsymbol{d}_k^T \boldsymbol{r}^{(k)} / \boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k$
    $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}_k$
end for

Let us now establish the connection between the Algorithm 5 and (4.32), which may not be immediately apparent because (4.32) involves only the initial residual $\mathsf{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$, while the residual at the current iteration $\boldsymbol{r}^{(k)}$ is used in the algorithm.

Proposition 4.18 (Convergence of the conjugate directions method)**.** *The vector $\boldsymbol{x}^{(k)}$ obtained after k iterations of the conjugate directions method is given by*

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(0)} - \sum_{i=0}^{k-1} \alpha_i \boldsymbol{d}_i, \qquad \alpha_i = \frac{\boldsymbol{d}_i^T \boldsymbol{r}^{(0)}}{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i}. \tag{4.33}$$

In particular, the method converges in at most n iterations.

Proof. Let us denote by $\boldsymbol{y}^{(k)}$ the solution obtained after k steps of Algorithm 5. Our goal is to show that $\boldsymbol{y}^{(k)}$ coincides with $\boldsymbol{x}^{(k)}$ defined in (4.33). The result is trivial for $k = 0$. Reasoning by induction, we assume that it is true up to k. Then performing step $k+1$ of the algorithm gives

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} - \omega_k \boldsymbol{d}_k, \qquad \omega_k = \frac{\boldsymbol{d}_k^T \boldsymbol{r}^{(k)}}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k}.$$

On the other hand, it holds from (4.33) that

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha_k \boldsymbol{d}_k, \qquad \alpha_k = \frac{\boldsymbol{d}_k^T \boldsymbol{r}^{(0)}}{\boldsymbol{d}_k^T \mathsf{A} \boldsymbol{d}_k}.$$

By the induction hypothesis, it holds that $\boldsymbol{y}^{(k)} = \boldsymbol{x}^{(k)}$, so in order to prove that $\boldsymbol{y}^{(k+1)} = \boldsymbol{x}^{(k+1)}$, it is sufficient to show that $\omega_k = \alpha_k$, i.e. that

$$\boldsymbol{d}_k^T \boldsymbol{r}^{(k)} = \boldsymbol{d}_k^T \boldsymbol{r}^{(0)} \quad \Leftrightarrow \quad \boldsymbol{d}_k^T (\boldsymbol{r}^{(k)} - \boldsymbol{r}^{(0)}) = 0 \quad \Leftrightarrow \quad \boldsymbol{d}_k^T \mathsf{A} (\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(0)}) = 0.$$

The latter equality is obvious from the A-orthonormality of the directions. □

Since ω_k in Algorithm 5 coincides with the expression in (4.29), the conjugate directions algorithm satisfies the following "local optimization" property: the iterate $\boldsymbol{x}^{(k+1)}$ minimizes f on the straight line $\omega \mapsto \boldsymbol{x}^{(k)} - \omega \boldsymbol{d}_k$. In contrast with the steepest descent method, however, the conjugate directions method also satisfies the following stronger property.

Proposition 4.19 (Optimality of the conjugate directions method)**.** *The iterate $\boldsymbol{x}^{(k)}$ is the minimizer of f over the set $\boldsymbol{x}^{(0)} + \mathcal{B}_k$, where $\mathcal{B}_k = \mathrm{Span}\{\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{k-1}\}$.*

Proof. By (4.32), it holds that

$$\boldsymbol{x}_* = \boldsymbol{x}^{(0)} - \sum_{i=0}^{n-1} \alpha_i \boldsymbol{d}_i, \qquad \alpha_i = \frac{\boldsymbol{d}_i^T \boldsymbol{r}^{(0)}}{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i}$$

On the other hand, any vector $\boldsymbol{y} \in \boldsymbol{x}^{(0)} + \mathcal{B}_k$ can be expanded as

$$\boldsymbol{y} = \boldsymbol{x}^{(0)} - \beta_0 \boldsymbol{d}_0 - \cdots - \beta_{k-1} \boldsymbol{d}_{k-1}.$$

Employing these two expressions, the formula for f in (4.27), and the A-orthogonality of the directions, we obtain

$$\begin{aligned} f(\boldsymbol{y}) &= \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x}_*)^T \mathsf{A}(\boldsymbol{y} - \boldsymbol{x}_*) - \frac{1}{2}\boldsymbol{x}_*^T \mathsf{A} \boldsymbol{x}_* \\ &= \frac{1}{2}\sum_{i=0}^{k-1} (\beta_i - \alpha_i)^2 \boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i + \frac{1}{2}\sum_{i=k}^{n-1} \alpha_i^2 \boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i - \frac{1}{2}\boldsymbol{x}_*^T \mathsf{A} \boldsymbol{x}_* \end{aligned}$$

This is minimized when $\beta_i = \alpha_i$ for all $i \in \{0, \dots, k-1\}$, in which case $\boldsymbol{y}$ coincides with the k -th iterate $\boldsymbol{x}^{(k)}$ of the conjugate directions method in view of Proposition 4.18. □

Remark 4.4. Let $\|\bullet\|_{\mathsf{A}}$ denote the norm induced by the inner product $\langle \bullet, \bullet \rangle_{\mathsf{A}}$. Since

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_*\|_{\mathsf{A}} = \sqrt{2f(\boldsymbol{x}^{(k)}) + \boldsymbol{x}_*^T \mathsf{A} \boldsymbol{x}_*},$$

Proposition 4.19 shows that $\boldsymbol{x}^{(k)}$ minimizes the norm $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_*\|_{\mathsf{A}}$ over $\boldsymbol{x}^{(0)} + \mathcal{B}_k$. This is not surprising since, by construction, the vector $\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(0)}$ is the orthogonal projection of $\boldsymbol{x}_* - \boldsymbol{x}^{(0)}$ onto $\mathcal{B}_k$, for the inner product $\langle \bullet, \bullet \rangle_{\mathsf{A}}$.

A corollary of (4.19) is that the gradient of f at $\boldsymbol{x}^{(k)}$, i.e. the residual $\boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}$, is orthogonal to any vector in $\{\boldsymbol{d}_0, \dots, \boldsymbol{d}_{k-1}\}$ for the usual Euclidean inner product. This can also be checked directly from the formula

$$\boldsymbol{x}^{(k)} - \boldsymbol{x}_* = \sum_{i=k}^{n-1} \alpha_i \boldsymbol{d}_i, \qquad \alpha_i = \frac{\boldsymbol{d}_i^T \boldsymbol{r}^{(0)}}{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i},$$

which follows directly from Proposition 4.18. Indeed, it holds that

$$\forall j \in \{0, \dots, k-1\}, \qquad \boldsymbol{d}_j^T \boldsymbol{r}^{(k)} = \boldsymbol{d}_j \mathsf{A}(\boldsymbol{x}^{(k)} - \boldsymbol{x}_*) = \sum_{i=k}^{n-1} \alpha_i \boldsymbol{d}_j^T \mathsf{A} \boldsymbol{d}_i = 0. \tag{4.34}$$

The conjugate gradient method

In the previous section, we showed that, given n conjugate directions, the solution to the linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ can be obtained in a finite number of iterations using Algorithm 5. The conjugate gradient method can be viewed as a particular case of the conjugate directions method. Instead of assuming that the conjugate directions are given, they are constructed iteratively as part of the algorithm. Given an initial guess $\boldsymbol{x}^{(0)}$, the first direction is the residual $\boldsymbol{r}^{(0)}$, which coincides

with the gradient of f at $\boldsymbol{x}^{(0)}$. The directions employed for the next iterations are obtained by applying the Gram-Schmidt process to the residuals. More precisely, given conjugate directions $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{k-1}$, and letting $\boldsymbol{x}^{(k)}$ denote the k-th iterate of the conjugate directions method, the direction $\boldsymbol{d}_k$ is obtained by

$$\boldsymbol{d}_k = \boldsymbol{r}^{(k)} - \sum_{i=0}^{k-1} \frac{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{r}^{(k)}}{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i} \boldsymbol{d}_i, \qquad \boldsymbol{r}^{(k)} = \mathsf{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}. \tag{4.35}$$

It is simple to check that $\boldsymbol{d}_k$ is indeed A-orthogonal to $\boldsymbol{d}_i$ for $i \in \{0, \ldots, k-1\}$, and that $\boldsymbol{d}_k$ is nonzero if $\boldsymbol{r}^{(k)}$ is nonzero. To prove the latter claim, we can take the Euclidean inner product of both sides with $\boldsymbol{r}^{(k)}$ and use Proposition 4.19 to deduce that

$$\boldsymbol{d}_k^T \boldsymbol{r}^{(k)} = (\boldsymbol{r}^{(k)})^T \boldsymbol{r}^{(k)} > 0. \tag{4.36}$$

Note also that since the directions are obtained by applying the Gram–Schmidt process to the residuals, it holds that

$$\forall k \in \{0, \ldots, n-1\}, \qquad \mathcal{B}_{k+1} := \operatorname{Span}\{\boldsymbol{d}_0, \ldots, \boldsymbol{d}_k\} = \operatorname{Span}\{\boldsymbol{r}^{(0)}, \ldots, \boldsymbol{r}^{(k)}\}. \tag{4.37}$$

The following result characterizes precisely the subspace $\mathcal{B}_{k+1}$.

Proposition 4.20. *Assume that $\|\boldsymbol{r}^{(k)}\| \neq 0$ for all $k < m \leqslant n$. Then it holds that*

$$\forall k \in \{0, \ldots m\}, \quad \operatorname{Span}\{\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(k)}\} = \operatorname{Span}\{\boldsymbol{r}^{(0)}, \mathsf{A}\boldsymbol{r}^{(0)}, \ldots, \mathsf{A}^k \boldsymbol{r}^{(0)}\} \tag{4.38}$$

The subspace on the right-hand side is called a Krylov subspace.

Proof. The result is clear for $k = 0$. Reasoning by induction, we prove that if the result is true up to $k < m$, then it is also true for $k + 1$. A simple calculation gives that

$$\begin{aligned} \boldsymbol{r}^{(k+1)} &= \mathsf{A}\big(\boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}_k\big) - \boldsymbol{b} \\ &= \boldsymbol{r}^{(k)} - \omega_k \mathsf{A}\boldsymbol{d}_k. \end{aligned} \tag{4.39}$$

From (4.35), we deduce that

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \omega_k \mathsf{A} \left(\boldsymbol{r}^{(k)} - \sum_{i=0}^{k-1} \frac{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{r}^{(k)}}{\boldsymbol{d}_i^T \mathsf{A} \boldsymbol{d}_i} \boldsymbol{d}_i \right).$$

By (4.37) and the induction hypothesis, the bracketed expression on the right-hand side belongs to $\mathcal{B}_{k+1}$, so the inclusion $\subset$ in (4.38) is clear. The inclusion $\supset$ then follows from the fact the dimension of the subspace

$$\operatorname{Span}\{\boldsymbol{d}_0, \ldots, \boldsymbol{d}_k\} = \operatorname{Span}\{\boldsymbol{r}^{(0)}, \ldots, \boldsymbol{r}^{(k)}\}$$

is equal to $k + 1$. □

It appears from (4.35) that the cost of calculating a new direction grows linearly with the iteration index. In fact, it turns out that only the last term in the sum is nonzero, and so the cost of calculating a new direction does not grow with the iteration index k. Indeed, notice that if $i \leqslant k-2$, then

$$\boldsymbol{d}_i^T \mathsf{A}\boldsymbol{r}^{(k)} = (\mathsf{A}\boldsymbol{d}_i)^T \mathsf{A}\boldsymbol{r}^{(k)} = 0,$$

because $\mathsf{A}\boldsymbol{d}_i \in \mathcal{B}_{i+2} \subset \mathcal{B}_k$ by Proposition 4.20, and $\boldsymbol{r}^{(k)}$ orthogonal to $\mathcal{B}_k$ for the Euclidean inner product by (4.34). This observation leads to Algorithm 6.

Algorithm 6 Conjugate gradient method

Pick initial $\boldsymbol{x}^{(0)}$
$\boldsymbol{d}_0 = \boldsymbol{r}^{(0)} = \mathsf{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$
for k in $\{0, \dots, n-1\}$ **do**
    if $\|\boldsymbol{r}^{(k)}\| = 0$ **then**
        Stop
    end if
    $\omega_k = \boldsymbol{d}_k^T \boldsymbol{r}^{(k)} / \boldsymbol{d}_k^T \mathsf{A}\boldsymbol{d}_k$
    $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \omega_k \boldsymbol{d}_k$
    $\boldsymbol{r}^{(k+1)} = \mathsf{A}\boldsymbol{x}^{(k+1)} - \boldsymbol{b}$
    $\beta_k = \boldsymbol{d}_k^T \mathsf{A}\boldsymbol{r}^{(k+1)} / \boldsymbol{d}_k^T \mathsf{A}\boldsymbol{d}_k$.
    $\boldsymbol{d}_{k+1} = \boldsymbol{r}^{(k+1)} - \beta_k \boldsymbol{d}_k$.
end for

Although the conjugate gradient method converges in a finite number of iterations, performing n iterations for very large systems would require an excessive computational cost, and so it is sometimes desirable to stop iterating when the residual is sufficiently small. To conclude this section, we study the convergence of the method.

Theorem 4.21 (Convergence of the conjugate gradient method)**.** *The error for the conjugate gradient method, measured as*

$$E_k := (\boldsymbol{x}^{(k)} - \boldsymbol{x}_*)^T \mathsf{A}(\boldsymbol{x}^{(k)} - \boldsymbol{x}_*),$$

satisfies the following inequality:

$$\forall q_k \in \mathbf{P}(k), \qquad E_{k+1} \leqslant \max_{1 \leqslant i \leqslant n} \big(1 + \lambda_i q_k(\lambda_i)\big)^2 E_0. \tag{4.40}$$

Here $\mathbf{P}(k)$ is the vector space of polynomials of degree less than or equal to k.

Proof. In view of Proposition 4.20, the iterate $\boldsymbol{x}^{(k+1)}$ can be written as

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(0)} + \sum_{i=0}^{k} \alpha_i \mathsf{A}^i \boldsymbol{r}^{(0)} = \boldsymbol{x}^{(0)} + p_k(\mathsf{A})\boldsymbol{r}^{(0)},$$

where p_k is a polynomial of degree k. By Proposition 4.19, p_k is in fact the polynomial of degree k such that $f(\boldsymbol{x}^{(k+1)})$ is minimized, and thus also E_{k+1} by (4.27). Noting that

$$\begin{aligned}\boldsymbol{x}^{(k+1)} - \boldsymbol{x}_* &= \boldsymbol{x}^{(0)} - \boldsymbol{x}_* + p_k(\mathsf{A})\boldsymbol{r}^{(0)} = \boldsymbol{x}^{(0)} - \boldsymbol{x}_* + p_k(\mathsf{A})\mathsf{A}(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*) \\ &= \Big(\mathsf{I} + \mathsf{A}p_k(\mathsf{A})\Big)(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*),\end{aligned}$$

we deduce that

$$\forall q_k \in \mathbf{P}(k), \qquad E_{k+1} \leqslant (\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)^T \mathsf{A}\Big(\mathsf{I} + \mathsf{A}q_k(\mathsf{A})\Big)^2 (\boldsymbol{x}^{(0)} - \boldsymbol{x}_*).$$

In order to exploit this inequality, it is useful to diagonalize A as $\mathsf{A} = \mathsf{Q}\mathsf{D}\mathsf{Q}^T$, for an orthogonal matrix Q and a diagonal matrix D. Since $q_k(\mathsf{A}) = \mathsf{Q}q_k(\mathsf{D})\mathsf{Q}^T$ for all $q_k \in \mathbf{P}(k)$, it holds that

$$\begin{aligned}\forall q_k \in \mathbf{P}(k), \qquad E_{k+1} &= \Big(\mathsf{Q}^T(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)\Big)^T \mathsf{D}\Big(\mathsf{I} + \mathsf{D}q_k(\mathsf{D})\Big)^2 \Big(\mathsf{Q}^T(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)\Big) \\ &\leqslant \max_{1 \leqslant i \leqslant n} \Big(1 + \lambda_i q_k(\lambda_i)\Big)^2 \underbrace{\Big(\mathsf{Q}^T(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)\Big)^T \mathsf{D}\Big(\mathsf{Q}^T(\boldsymbol{x}^{(0)} - \boldsymbol{x}_*)\Big)}_{E_0},\end{aligned}$$

which completes the proof. □

A corollary of Theorem 4.21 is that, if A has $m \leqslant n$ distinct eigenvalues, then the conjugate gradient method converges in at most m iterations. Indeed, in this case we can take

$$q_{m-1}(\lambda) = \frac{1}{\lambda}\left(\frac{(\lambda_1 - \lambda)\dots(\lambda_m - \lambda)}{\lambda_1 \dots \lambda_m} - 1\right).$$

It is simple to check that the right-hand side is indeed a polynomial, and that $1 + \lambda_i q_{m-1}(\lambda_i) = 0$ for all eigenvalues of A.

In general, finding the polynomial that minimizes the right-hand side of (4.40) is not possible, because the eigenvalues of A are unknown. However, it is possible to derive from this equation an error estimate with an explicit dependence on the condition number $\kappa = \kappa_2(\mathsf{A})$.

Theorem 4.22. *It holds that*

$$\forall k \geqslant 0, \qquad E_k \leqslant 4\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2k} E_0,$$

Proof. Theorem 4.21 implies that

$$\forall q_k \in \mathbf{P}(k), \qquad E_{k+1} \leqslant \max_{\lambda \in [\lambda_1, \lambda_n]} \Big(1 + \lambda q_k(\lambda)\Big)^2 E_0,$$

where λ_1 and λ_n are the minimum and maximum eigenvalues of A. Notice that

$$\Big\{1 + \lambda q_k \colon q_k \in \mathbf{P}(k)\Big\} = \Big\{p_k \colon p_k \in \mathbf{P}(k+1) \ \text{ and } \ p_k(0) = 1\Big\}$$

Therefore, it follows from Exercise C.7 that the right-hand side is minimized when

$$1 + \lambda q_k(\lambda) = \frac{T_{k+1}\left(\frac{\lambda_n+\lambda_1-2\lambda}{\lambda_n-\lambda_1}\right)}{T_{k+1}\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)}, \tag{4.41}$$

where T_{k+1} is the *Chebyshev* polynomial of degree $k+1$, see (C.1). We recall that $|T_{k+1}(x)| \leqslant 1$ for all $x \in [-1, 1]$. Consequently, by the expression of Chebyshev polynomials given in Exercise C.3, the following inequality holds true for all $\lambda \in [\lambda_1, \lambda_n]$:

$$\begin{aligned}\left|1 + \lambda q_k(\lambda)\right| \leqslant \frac{1}{T_{k+1}\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)} &= 2\left(\left(r + \sqrt{r^2-1}\right)^{k+1} + \left(r - \sqrt{r^2-1}\right)^{k+1}\right)^{-1}, \\ &= 2\left(\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^{k+1} + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k+1}\right)^{-1}.\end{aligned}$$

where $r = \frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}$. Since the first term in the bracket converges to zero as $k \to \infty$, it is natural to bound this expression by keeping only the second term, which after simple algebraic manipulations leads to

$$\forall \lambda \in [\lambda_1, \lambda_n], \qquad \left|1 + \lambda q_k(\lambda)\right| \leqslant 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k+1}.$$

From this inequality, the statement of the theorem follows immediately. □

4.4 Exercises

Exercise 4.1. *In the simple case where* A *is symmetric, find values of* $\boldsymbol{x}$, $\boldsymbol{b}$ *and* $\Delta\boldsymbol{b}$ *for which the inequality* (4.2) *is in fact an equality?*

Exercise 4.2 (Inverse of Gaussian transformation)**.** *Prove the formula* (4.8)*.*

Exercise 4.3. *Prove that the product of two lower triangular matrices is lower triangular.*

Exercise 4.4. *Assume that* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *is positive definite, i.e. that*

$$\forall \boldsymbol{x} \in \mathbf{R}^n \backslash \{\mathbf{0}_n\}, \qquad \boldsymbol{x}^T \mathsf{A} \boldsymbol{x} > 0.$$

Show that all the principal submatrices of A *are nonsingular.*

Exercise 4.5. *Implement the backward substitution algorithm for solving* $\mathsf{U}x = y$*. What is the computational cost of the algorithm?*

Exercise 4.6. *Compare the condition number of the matrices* L *and* U *with and without partial pivoting. For testing, use a matrix with pseudo-random entries generated as follows*

```
import Random
# Set the seed so that the code is deterministic
Random.seed!(0)
n = 1000 # You can change this parameter
A = randn(n, n)
```

Solution. See the Jupyter notebook for this chapter. △

Exercise 4.7. *Write a code for calculating the Cholesky factorization of a symmetric positive definite matrix* A *by comparing the entries of the product* CC^T *with those of the matrix* A*. What is the associated computational cost, and how does it compare with that of the* LU *factorization?*

Extra credit: *… if your code is able to exploit the potential banded structure of the matrix passed as argument for better efficiency. Specifically, your code will be tested with a matrix is of the type* `BandedMatrix` *defined in the* ***`BandedMatrices.jl`*** *package, which you will need to install. The following code can be useful for testing purposes.*

```
import BandedMatrices
import LinearAlgebra

function cholesky(A)
    m, n = size(A)
    m != n && error("Matrix must be square")
    # Convert to banded matrix
    B = BandedMatrices.BandedMatrix(A)
    B.u != B.l && error("Matrix must be symmetric")
    # --> Your code comes here <--
end
n, u, l = 20000, 2, 2
A = BandedMatrices.brand(n, u, l)
A = A*A'
# so that A is symmetric and positive definite (with probability 1).
C = @time cholesky(A)
LinearAlgebra.norm(C*C' - A, Inf)
```

For information, my code takes about 1 second to run with the parameters given here.

Exercise 4.8 (Matrix square root)**.** *Let* $A \in \mathbf{R}^{n\times n}$ *be a symmetric positive definite matrix. Show that* A *has a positive definite square root, i.e. that there exists a symmetric matrix* B *such that* $BB = A$*.*

Solution. Since A is symmetric, there exist a diagonal matrix D and an orthogonal matrix Q such that $A = QDQ^T$. Let $D^{1/2}$ denote the diagonal matrix obtained by applying the square root function to the entries of D, and notice that $D^{1/2}D^{1/2} = D$. Then it holds that

$$\mathsf{A} = \left(\mathsf{Q}\mathsf{D}^{1/2}\mathsf{Q}^T\right)\left(\mathsf{Q}\mathsf{D}^{1/2}\mathsf{Q}^T\right).$$

The matrix $\mathsf{A}^{1/2} := \mathsf{Q}\mathsf{D}^{1/2}\mathsf{Q}^T$ is a square root of the matrix A, in the sense that $\mathsf{A}^{1/2}\mathsf{A}^{1/2}$, and it is positive definite because the diagonal elements of $\mathsf{D}^{1/2}$ are strictly positive. △

Exercise 4.9. *Show that if* A *is row or column diagonally dominant, then* A *is invertible.*

Exercise 4.10. *Let* T *be a nonsingular matrix. Show that*

$$\|\mathsf{A}\|_{\mathsf{T}} := \|\mathsf{T}^{-1}\mathsf{A}\mathsf{T}\|_2$$

defines a matrix norm induced by a vector norm.

Exercise 4.11. *Let* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *be a symmetric positive definite matrix. Show that the functional*

$$\|\bullet\|_{\mathsf{A}} \colon \boldsymbol{x} \mapsto \sqrt{\boldsymbol{x}^T\mathsf{A}\boldsymbol{x}}$$

defines a norm on $\mathbf{R}^n$*.*

Solution. We need to prove that the three axioms of a norm are satisfied:

- (**Positivity**) Since A is positive definite, it holds that $\|\boldsymbol{x}\|_{\mathsf{A}} > 0$ for any $\boldsymbol{x} \in \mathbf{R}^n \setminus \{\mathbf{0}\}$.
- (**Homogeneity**) It is clear that $\|c\boldsymbol{x}\|_{\mathsf{A}} = |c|\ \|\boldsymbol{x}\|_{\mathsf{A}}$ for any $c \in \mathbf{R}$.
- (**Triangle inequality**) Let $\mathsf{A}^{1/2}$ denote the positive definite square root of A, which exists by Exercise 4.8. Then

  $$\|\boldsymbol{x}\|_{\mathsf{A}} = \|\mathsf{A}^{1/2}\boldsymbol{x}\|_2.$$

  The triangle inequality for $\|\bullet\|_{\mathsf{A}}$ then follows from that for $\|\bullet\|_2$:

  $$\|\boldsymbol{x}+\boldsymbol{y}\|_{\mathsf{A}} = \|\mathsf{A}^{1/2}\boldsymbol{x}+\mathsf{A}^{1/2}\boldsymbol{y}\|_2 \leqslant \|\mathsf{A}^{1/2}\boldsymbol{x}\| + \|\mathsf{A}^{1/2}\boldsymbol{y}\|_2 = \|\boldsymbol{x}\|_{\mathsf{A}} + \|\boldsymbol{y}\|_{\mathsf{A}}.$$

Another option for solving this exercise is to show that

$$\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\mathsf{A}} := \boldsymbol{x}^T\mathsf{A}\boldsymbol{y}$$

defines an inner product, with induced norm given by $\|\bullet\|_{\mathsf{A}}$. △

Exercise 4.12. *Show that the residual satisfies the equation*

$$\boldsymbol{r}^{(k+1)} = \mathsf{N}\mathsf{M}^{-1}\boldsymbol{r}^{(k)} = (\mathsf{I} - \mathsf{A}\mathsf{M}^{-1})\boldsymbol{r}^{(k)}.$$

Exercise 4.13. *Show that, if* A *and* B *are two square matrices, then* $\rho(\mathsf{A}\mathsf{B}) = \rho(\mathsf{B}\mathsf{A})$*.*

Exercise 4.14. *Is* $\rho(\bullet)$ *a norm? Prove or disprove.*

Exercise 4.15. *Prove that, if* A *is a diagonal matrix, then*

$$\|\mathsf{A}\|_1 = \|\mathsf{A}\|_2 = \|\mathsf{A}\|_\infty = \rho(\mathsf{A}).$$

Exercise 4.16. *Show that, for any matrix norm $\|\bullet\|$ induced by a vector norm,*

$$\rho(\mathsf{A}) \leqslant \|\mathsf{A}\|.$$

Exercise 4.17. *Let $\|\bullet\|$ denote the Euclidean vector norm on $\mathbf{R}^n$. We define in Appendix A the induced matrix norm as*

$$\|\mathsf{A}\| = \sup\Big\{\|\mathsf{A}\boldsymbol{x}\| \colon \|\boldsymbol{x}\| \leqslant 1\Big\}.$$

Show from this definition that, if A is symmetric and positive definite, then

$$\|\mathsf{A}\| = \|\mathsf{A}\|_* := \sup\Big\{|\boldsymbol{x}^T\mathsf{A}\boldsymbol{x}| \colon \|\boldsymbol{x}\| \leqslant 1\Big\}.$$

Solution. By the Cauchy–Schwarz inequality and the definition of $\|\mathsf{A}\|$, it holds that

$$\forall \boldsymbol{x} \in \mathbf{R}^n \text{ with } \|\boldsymbol{x}\| \leqslant 1, \qquad |\boldsymbol{x}^T\mathsf{A}\boldsymbol{x}| \leqslant \|\boldsymbol{x}\|\|\mathsf{A}\boldsymbol{x}\| \leqslant \|\boldsymbol{x}\|\|\mathsf{A}\|\|\boldsymbol{x}\| \leqslant \|\mathsf{A}\|.$$

This shows that $\|\mathsf{A}\|_* \leqslant \|\mathsf{A}\|$. Conversely, letting B denote a matrix square root of A (see Exercise 4.8), we have

$$\begin{aligned}\forall \boldsymbol{x} \in \mathbf{R}^n \text{ with } \|\boldsymbol{x}\| \leqslant 1, \qquad \|\mathsf{A}\boldsymbol{x}\| &= \sqrt{\boldsymbol{x}^T\mathsf{A}^T\mathsf{A}\boldsymbol{x}} = \sqrt{(\mathsf{B}\boldsymbol{x})^T\mathsf{B}\mathsf{B}(\mathsf{B}\boldsymbol{x})} = \sqrt{(\mathsf{B}\boldsymbol{x})^T\mathsf{A}(\mathsf{B}\boldsymbol{x})} \\ &= \|\mathsf{B}\boldsymbol{x}\|\sqrt{\boldsymbol{y}^T\mathsf{A}\boldsymbol{y}}, \qquad \boldsymbol{y} = \frac{\mathsf{B}\boldsymbol{x}}{\|\mathsf{B}\boldsymbol{x}\|}.\end{aligned}$$

It holds that $\|\mathsf{B}\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T\mathsf{A}\boldsymbol{x}} \leqslant \sqrt{\|\mathsf{A}\|_*}$. In addition $\|\boldsymbol{y}\| = 1$, so the expression inside the square root is bounded from above by $\|\mathsf{A}\|_*$, which enables to conclude the proof. △

Exercise 4.18. *Prove that, if the matrix A is strictly diagonally dominant (by rows or columns), then the Gauss–Seidel method converges, i.e. $\rho(\mathsf{M}^{-1}\mathsf{N}) < 1$. You can use the same approach as in the proof of Proposition 4.11.*

Exercise 4.19. *Let $\mathsf{A} \in \mathbf{R}^{n\times n}$ denote a symmetric positive definite matrix, and assume that the vectors $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n$ are pairwise A-orthogonal directions. Show that $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n$ are linearly independent.*

Exercise 4.20 (Steepest descent algorithm)**.** *Consider the linear system*

$$\mathsf{A}\boldsymbol{x} := \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} =: \boldsymbol{b}. \tag{4.42}$$

- *Show that* A *is positive definite.*
- *Draw the contour lines of the function*

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\mathsf{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}.$$

- *Plot the contour lines of* f *in Julia using the function* `contourf` *from the package* `Plots`.
- *Using Theorem 4.17, estimate the number* K *of iterations of the steepest descent algorithm required in order to guarantee that* $E_K \leqslant 10^{-8}$*, when starting from the vector* $\boldsymbol{x}^{(0)} = (2\ 3)^T$.
- *Implement the steepest descent method for finding the solution to (4.42), and plot the iterates as linked dots over the filled contour of* f.
- *Plot the error* E_k *as a function of the iteration index, using a linear scale for the* x *axis and a logarithmic scale for the* y *axis.*

Exercise 4.21. *Compute the number of floating point operations required for performing one iteration of the conjugate gradient method, assuming that the matrix* A *contains* $\alpha \ll n$ *nonzero elements per row.*

Exercise 4.22 (Solving the Poisson equation over a rectangle)**.** *We consider in this exercise Poisson's equation in the domain* $\Omega = (0,2) \times (0,1)$*, equipped with homogeneous Dirichlet boundary conditions:*

$$\begin{aligned} -\triangle f(x,y) &= b(x,y), \qquad x \in \Omega, \\ f(x) &= 0, \qquad\quad x \in \partial\Omega. \end{aligned}$$

The right-hand side is

$$b(x,y) = \sin(4\pi x) + \sin(2\pi y).$$

A number of methods can be employed in order to discretize this partial differential equation. After discretization, a finite-dimensional linear system of the form $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ *is obtained. A Julia function for calculating the matrix* A *and the vector* $\boldsymbol{b}$ *using the finite difference method is given to you on the course website, as well as a function to plot the solution. The goal of this exercise is to solve the linear system using the conjugate gradient method. Use the same stopping criterion as in Exercise 4.25.*

Exercise 4.23. *Show that if* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *is nonsingular, then the solution to the equation* $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ *belongs to the Krylov subspace*

$$\mathcal{K}_n(\mathsf{A}, \boldsymbol{b}) = \mathrm{Span}\Big\{\boldsymbol{b}, \mathsf{A}\boldsymbol{b}, \mathsf{A}^2\boldsymbol{b}, \ldots, \mathsf{A}^{n-1}\boldsymbol{b}\Big\}.$$

Exercise 4.24. *Write a function* `lu(A)` *for calculating the* LU *decomposition of a square matrix* $\mathsf{A} \in \mathbf{R}^{n\times n}$*, with* L *unit lower triangular and* U *upper triangular, not by Gaussian*

elimination but by comparing the entries of the product LU *with those of* A*. To this end, one option is to compare the entries one by one in the order* $(1,1)$*,* $(1,2)$*, …,* $(1,n)$*,* $(2,1)$*,* $(2,2)$*, …, i.e. row by row starting from the top. For example,*

- *Comparing the entry* $(1,k)$ *with* $k \in \{1,\ldots,n\}$ *gives*

$$\ell_{11}u_{1k} = a_{1k}.$$

  Since $\ell_{11} = 1$ *as* L *is unit lower triangular, this implies that* $u_{1k} = a_{1k}$*.*

- *Comparing the entry* $(2,1)$ *gives*

$$\ell_{21}u_{11} = a_{21}$$

  and so $\ell_{21} = a_{21}/u_{11}$*.*

- *Comparing the entry* $(2,k)$ *with* $k \in \{2,\ldots,n\}$ *gives*

$$\ell_{21}u_{1k} + \ell_{22}u_{2k} = a_{2k}.$$

  Given the previous items, the only unknown in this equation is u_{2k}*.*

- *Comparing the entry* $(3,1)$ *gives*

$$\ell_{31}u_{11} = a_{31},$$

  and so $\ell_{31} = a_{31}/u_{11}$*.*

- *Comparing the entry* $(3,2)$ *gives*

$$\ell_{31}u_{12} + \ell_{32}u_{22} = a_{32},$$

  Given the previous items, the only unknown in this equation is ℓ_{32}*.*

- *Comparing the entry* $(3,k)$ *with* $k \in \{3,\ldots,n\}$ *gives*

$$\ell_{31}u_{1k} + \ell_{32}u_{2k} + \ell_{33}u_{3k} = a_{3k},$$

  Given the previous items, the only unknown in this equation is u_{3k}*.*

Notice that a pattern seems to be emerging: when going through the entries row by row starting from the top left corner of the matrix, comparing the entry (i,j) *provides an equation for* ℓ_{ij} *if* $j < i$*, and an equation for* u_{ij} *if* $j \geqslant i$*. Do not use any external package for this exercise.*
Extra credit: *… if your code is able to exploit the potential banded structure of the matrix passed as argument for better efficiency. Specifically, your code will be tested with a matrix created as follows*

```
b, n = 5, 10000
A = [abs(i-j) <= b ? rand() : 0.0 for i in 1:n, j in 1:n]
```

Exercise 4.25. *Implement an iterative method based on a splitting for finding a solution to the following linear system on* $\mathbf{R}^n$.

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \qquad h = \frac{1}{n+1}.$$

Plot the norm of the residual as a function of the iteration index. Use as stopping criterion the condition

$$\|\boldsymbol{r}^{(k)}\| \leqslant \varepsilon \|\boldsymbol{b}\|, \qquad \varepsilon = 10^{-8}.$$

As initial guess, use a vector of zeros. The code will be tested with $n = 500$. *Do not use any library (except for plotting), and do not use the backslash operator.*

Exercise 4.26. *Find a formula for the optimal value of* ω *in the relaxation method given* n, *for the linear system in Exercise 4.25. The proof of Proposition 4.15, as well as the formula (4.25) for the eigenvalues of a tridiagonal matrix, are useful to this end.*

Solution. Corollary 4.13 and Proposition 4.14 imply that a sufficient and necessary condition for convergence, when A is Hermitian and positive definite, is that $\omega \in (0, 2)$. Let $M_\omega = \frac{1}{\omega}\mathsf{D} + \mathsf{L}$ and $N_\omega = \frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}$. A nonzero scalar $\lambda \in \mathbf{C}$ is an eigenvalue of $\mathsf{M}_\omega^{-1}\mathsf{N}_\omega$ if and only if

$$\det(\mathsf{M}_\omega^{-1}\mathsf{N}_\omega - \lambda \mathsf{I}) = 0 \quad \Leftrightarrow \quad \det(\mathsf{M}_\omega^{-1}) \det(\mathsf{N}_\omega - \lambda \mathsf{M}_\omega) = 0 \quad \Leftrightarrow \quad \det(\lambda \mathsf{M}_\omega - \mathsf{N}_\omega) = 0.$$

Substituting the expressions of M_ω and N_ω, we obtain that this condition can be equivalently rewritten as

$$\det\left(\lambda \mathsf{L} + \left(\frac{\lambda + \omega - 1}{\omega}\right)\mathsf{D} + \mathsf{U}\right) = 0 \quad \Leftrightarrow \quad \det\left(\sqrt{\lambda}\mathsf{L} + \left(\frac{\lambda + \omega - 1}{\omega}\right)\mathsf{D} + \sqrt{\lambda}\mathsf{U}\right) = 0$$

where we used (4.24) for the last equivalence. The equality of the determinants in these two equations is valid for $\sqrt{\lambda}$ denoting either of the two complex square roots of λ. This condition is equivalent to

$$\det\left(\mathsf{L} + \left(\frac{\lambda + \omega - 1}{\sqrt{\lambda}\omega}\right)\mathsf{D} + \mathsf{U}\right) = 0.$$

We recognize from the proof of Proposition 4.15 that this condition is equivalent to

$$\frac{\lambda + \omega - 1}{\sqrt{\lambda}\omega} \in \text{spectrum}(\mathsf{M}_\mathcal{J}^{-1}\mathsf{N}_\mathcal{J}).$$

In other words, for any $(\lambda, \mu) \in \mathbf{C}^2$ such that

$$\frac{(\lambda + \omega - 1)^2}{\lambda \omega^2} = \mu^2, \tag{4.43}$$

it holds that $\mu \in \text{spectrum}(\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}})$ if and only if $\lambda \in \text{spectrum}(\mathsf{M}_{\omega}^{-1}\mathsf{N}_{\omega})$. By (4.25), the eigenvalues of $\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}}$ are real and given by

$$\mu_j = \cos\Big(\frac{j\pi}{n+1}\Big), \qquad 1 \leqslant j \leqslant n. \tag{4.44}$$

Rearranging (4.43), we find

$$\lambda^2 + \lambda(2(\omega-1) - \omega^2\mu^2) + (\omega-1)^2 = 0.$$

For given $\omega \in (0,2)$ and $\mu \in \mathbf{R}$, this is a quadratic equation for λ with solutions

$$\lambda_\pm = \left(\frac{\omega^2\mu^2}{2} + 1 - \omega\right) \pm \omega\mu\sqrt{\frac{\omega^2\mu^2}{4} + 1 - \omega},$$

Since the first bracket is positive when the argument of the square root is positive, it is clear that

$$\max\big\{|\lambda_-|, |\lambda_+|\big\} = \left|\frac{\omega^2\mu^2}{2} + 1 - \omega + \omega|\mu|\sqrt{\frac{\omega^2\mu^2}{4} + 1 - \omega}\right|.$$

Combining this with (4.44), we deduce that the spectral radius of $\mathsf{M}_{\omega}^{-1}\mathsf{N}_{\omega}$ is given by

$$\rho(\mathsf{M}_{\omega}^{-1}\mathsf{N}_{\omega}) = \max_{j\in\{1,\dots,n\}}\left|\frac{\omega^2\mu_j^2}{2} + 1 - \omega + \omega|\mu_j|\sqrt{\frac{\omega^2\mu_j^2}{4} + 1 - \omega}\right|. \tag{4.45}$$

We wish to minimize this expression over the interval $\omega \in (0,2)$. While this can be achieved by algebraic manipulations, we content ourselves here with graphical exploration. Figure 4.3 depicts the amplitude of the modulus in (4.45) for different values of μ. It is apparent that, for given ω, the modulus increases as μ increases, which suggests that

$$\rho(\mathsf{M}_{\omega}^{-1}\mathsf{N}_{\omega}) = \left|\frac{\omega^2\mu_*^2}{2} + 1 - \omega + \omega|\mu_*|\sqrt{\frac{\omega^2\mu_*^2}{4} + 1 - \omega}\right|, \qquad \mu_* = \rho(\mathsf{M}_{\mathcal{J}}^{-1}\mathsf{N}_{\mathcal{J}}). \tag{4.46}$$

The figure also suggests that for a given value of μ, the modulus is minimized at the discontinuity of the first derivative, which occurs when the argument of the square root is zero. We conclude that the optimal ω satisfies

$$\frac{\omega_{\text{opt}}^2\mu_*^2}{4} + 1 - \omega_{\text{opt}} = 0 \quad \underset{\omega<2}{\Longrightarrow} \quad \omega_{\text{opt}} = 2\frac{1-\sqrt{1-\mu_*^2}}{\mu_*^2} = \frac{2}{1+\sqrt{1-\mu_*^2}} = \frac{2}{1+\sin\big(\frac{\pi}{n+1}\big)}.$$

△

Exercise 4.27 (Midterm 2022). *Let $\mathsf{A} \in \mathbf{R}^{n\times n}$ be a symmetric positive definite matrix and let $\boldsymbol{b} \in \mathbf{R}^n$. The steepest descent algorithm for solving $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ is given hereafter:*

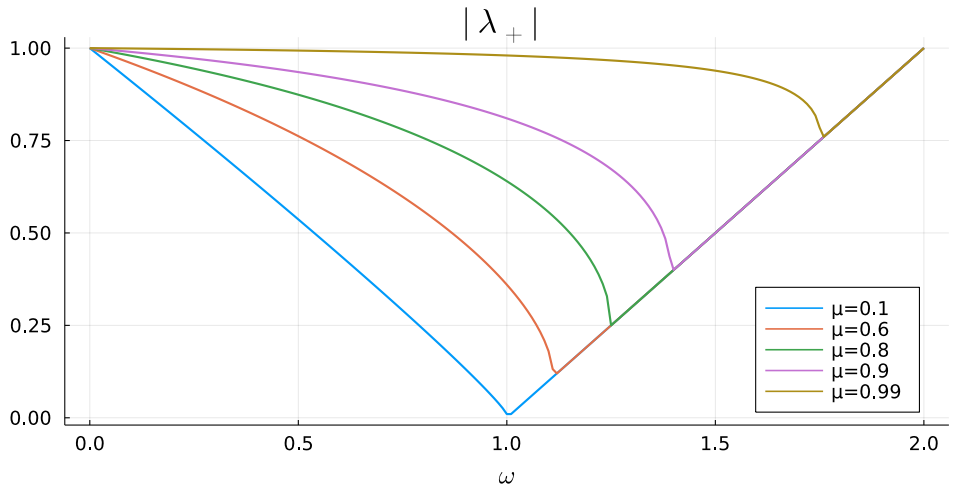


Figure 4.3: Modulus of $|\lambda_+|$ as a function of ω, for different eigenvalues of μ.

Pick $\varepsilon > 0$ *and initial* $\boldsymbol{x}$
$\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
while $\|\boldsymbol{r}\| \geqslant \varepsilon\|\boldsymbol{b}\|$ ***do***
    $\omega \leftarrow \boldsymbol{r}^T\boldsymbol{r}/\boldsymbol{r}^T\mathsf{A}\boldsymbol{r}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} - \omega\boldsymbol{r}$
    $\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
end while

- *Why is this method called the* steepest descent *algorithm?*
- *How many floating point operations does an iteration of this algorithm require?*
- *Are the following statements true of false? (2 marks)*

    1. *There exists a unique solution* $\boldsymbol{x}_*$ *to the linear system* $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$.
    2. *The iterates converge to* $\boldsymbol{x}_*$ *in at most* n *iterations.*
    3. *We consider the following modification of the algorithm:*

        Pick $\varepsilon > 0$, $\omega > 0$ *and initial* $\boldsymbol{x}$
        $\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
        while $\|\boldsymbol{r}\| \geqslant \varepsilon\|\boldsymbol{b}\|$ ***do***
            $\boldsymbol{x} \leftarrow \boldsymbol{x} - \omega\boldsymbol{r}$
            $\boldsymbol{r} \leftarrow \mathsf{A}\boldsymbol{x} - \boldsymbol{b}$
        end while

        If ω *is sufficiently small, then this algorithm converges.*

    4. *Here we no longer assume that* A *is positive definite. Instead, we consider that*

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

        In this case, the steepest descent algorithm is convergent for any initial $\boldsymbol{x}$.

Exercise 4.28 (Final exam Spring 2022). *Assume that* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *is a nonsingular matrix and that* $\boldsymbol{b} \in \mathbf{R}^n$. *We wish to solve the linear system* (4.1) *using an iterative method where each iteration is of the form*

$$\mathsf{M}\boldsymbol{x}_{k+1} = \mathsf{N}\boldsymbol{x}_k + \boldsymbol{b}. \tag{4.47}$$

Here $\mathsf{A} = \mathsf{M} - \mathsf{N}$ *is a splitting of* A *such that* M *is nonsingular, and* $\boldsymbol{x}_k \in \mathbf{R}^n$ *denotes the* k*-th iterate of the numerical scheme.*

1. *Let* $\boldsymbol{e}_k := \boldsymbol{x}_k - \boldsymbol{x}_*$, *where* $\boldsymbol{x}_*$ *is the exact solution to* (4.1). *Prove that*

$$\boldsymbol{e}_{k+1} = \mathsf{M}^{-1}\mathsf{N}\boldsymbol{e}_k.$$

2. *Let* $L = \|\mathsf{M}^{-1}\mathsf{N}\|_\infty$. *Prove that*

$$\forall k \in \mathbf{N}, \qquad \|\boldsymbol{e}_k\|_\infty \leqslant L^k \|\boldsymbol{e}_0\|_\infty. \tag{4.48}$$

3. *Is the condition* $\|\mathsf{M}^{-1}\mathsf{N}\|_\infty < 1$ *necessary for convergence when* $\boldsymbol{x}_0 \neq \boldsymbol{x}_*$?

4. *Assume that* A *is strictly row diagonally dominant, in the sense that*

$$\forall i \in \{1, \ldots, n\}, \qquad |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|.$$

   Show that, in this case, the inequality $\|\mathsf{M}^{-1}\mathsf{N}\|_\infty < 1$ *holds for the Jacobi method, i.e. when* M *contains just the diagonal of* A. *You may take for granted the following expression for the* ∞*-norm of a matrix* $\mathsf{X} \in \mathbf{R}^{n\times n}$*:*

$$\|\mathsf{X}\|_\infty = \max_{1 \leqslant i \leqslant n} \sum_{j=1}^{n} |x_{ij}|.$$

5. *Write down a few iterations of the Jacobi method when*

$$\mathsf{A} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \qquad \boldsymbol{b}\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \boldsymbol{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

   Is the method convergent?

4.5 Discussion and bibliography

In this chapter, we presented direct methods and some of the standard iterative methods for solving linear systems. We focused particularly on linear systems with a symmetric positive definite matrix. Section 4.2 is based on [10] [18] and Section 4.3 roughly follows [15, Chapter 2]. The book [11] is a very detailed reference on iterative methods for solving sparse linear systems. The reference [13] is an excellent introduction to the conjugate gradient method.

Chapter 5

Solution of nonlinear systems

5.1 The bisection method 131
5.2 Fixed point methods 132
5.3 Convergence of fixed point methods 133
5.4 Examples of fixed point methods 137
5.4.1 The chord method 137
5.4.2 The Newton–Raphson method 137
5.4.3 The secant method 143
5.5 A numerical experiment 145
5.6 Exercises 147
5.7 Discussion and bibliography 150

Introduction

This chapter concerns the numerical solution of nonlinear equations of the general form

$$\boldsymbol{f}(\boldsymbol{x}) = 0, \qquad \boldsymbol{f}\colon \mathbf{R}^n \to \mathbf{R}^n. \tag{5.1}$$

A solution to this equation is called a *zero* of the function f. Except in particular cases (for example linear systems), there does not exist a numerical method for solving (5.1) in a finite number of operations, so iterative methods are required.

In contrast with the previous chapter, it may not be the case that (5.1) admits one and only one solution. For example, the equation $1 + x^2 = 0$ does not have a (real) solution, and the equation $\cos(x) = 0$ has infinitely many. Therefore, convergence results usually contain assumptions on the function f that guarantee the existence and uniqueness of a solution in $\mathbf{R}^n$ or a subset of $\mathbf{R}^n$.

For an iterative method generating approximations $(\boldsymbol{x}_k)_{k \geqslant 0}$ of a root $\boldsymbol{x}_*$, we define the error as $\boldsymbol{e}_k = \boldsymbol{x}_k - \boldsymbol{x}_*$. If the sequence $(\boldsymbol{x}_k)_{k \geqslant 0}$ converges to $\boldsymbol{x}_*$ in the limit as $k \to \infty$ and if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{e}_{k+1}\|}{\|\boldsymbol{e}_k\|^q} = r, \tag{5.2}$$

then we say that $(\boldsymbol{x}_k)_{k\geqslant 0}$ converges with *order of convergence* q and *rate of convergence* r. In addition, we say that the convergence is linear $q = 1$, and quadratic if $q = 2$. The convergence is said to be superlinear if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{e}_{k+1}\|}{\|\boldsymbol{e}_k\|} = 0. \tag{5.3}$$

In particular, the convergence is superlinear if the order of convergence is $q > 1$.

Remark 5.1. The notion of order of convergence may be defined also when the limit in (5.2) does not exist. A more general definition for the order of convergence of a sequence $(\boldsymbol{x}_k)_{k\geqslant 0}$ converging to $\boldsymbol{x}_*$ is the following:

$$q(\boldsymbol{x}_0) = \inf\left\{p \in [1,\infty) : \limsup_{k\to\infty} \frac{\|\boldsymbol{e}_{k+1}\|}{\|\boldsymbol{e}_k\|^p} = \infty\right\},$$

or $q(\boldsymbol{x}_0) = \infty$ if the numerator and denominator of the fraction are zero for sufficiently large k. It is possible to define similarly the order of convergence of an iterative method for an initial guess in a neighborhood V of $\boldsymbol{x}_*$:

$$q = \inf\left\{p \in [1,\infty) : \sup_{\boldsymbol{x}_0\in V}\left(\limsup_{k\to\infty} \frac{\|\boldsymbol{e}_{k+1}\|}{\|\boldsymbol{e}_k\|^p}\right) = \infty\right\},$$

where the fraction should be interpreted as 0 if the numerator and denominator are zero. A more detailed discussion of this subject is beyond the scope of this course.

The rest of chapter is organized as follows:

- In Section 5.1, by way of introduction to the subject of numerical methods for nonlinear equations, we present and analyze the bisection method.
- In Section 5.2, we present a general method based on a fixed point iteration for solving (5.1). The convergence of this method is analyzed in Section 5.3.
- In Section 5.4, two concrete examples of fixed point methods are studied: the chord method and the Newton–Raphson method.

5.1 The bisection method

As an introduction to numerical methods for solving nonlinear equations, we present the bisection method. This method applies only in the case of a real-valued function $f\colon \mathbf{R} \to \mathbf{R}$, and relies on the knowledge of two points $a < b$ such that $f(a)$ and $f(b)$ have different signs. By the intermediate value theorem, there necessarily exists $x_* \in (a, b)$ such that $f(x_*) = 0$. The idea of the bisection method it to successively divide the interval in two equal parts, and

to retain, based on the sign of f at the midpoint $x_{1/2}$, the one that necessarily contains a root. If $f(x_{1/2})f(a) \geqslant 0$, then $f(x_{1/2})f(b) \leqslant 0$ and so there necessarily exists a root of f in the interval $[x_{1/2}, b)$ by the intermediate value theorem. In contrast, if $f(x_{1/2})f(a) < 0$, then there necessarily is a root in the interval $(a, x_{1/2})$. The algorithm is presented in Algorithm 7.

Algorithm 7 Bisection method

```
Assume that f(a)f(b) < 0 with a < b.
Pick ε > 0.
x ← a/2 + b/2
while |b − a| ⩾ ε do
    if f(x)f(a) ⩾ 0 then
        a ← x
    else
        b ← x
    end if
    x ← a/2 + b/2
end while
```

The following result establishes the convergence of the method.

Proposition 5.1. *Assume that $f\colon \mathbf{R} \to \mathbf{R}$ is a continuous function and $f(a)f(b) < 0$. Let $[a_j, b_j]$ denote the interval obtained after j iterations of the bisection method, and let $x_j = (a_j + b_j)/2$ denote the midpoint of the interval, Then there exists a root x_* of f such that*

$$|x_j - x_*| \leqslant (b_0 - a_0)2^{-(j+1)}. \tag{5.4}$$

Proof. By construction, $f(a_j)f(b_j) \leqslant 0$ and $f(b) \neq 0$. Therefore, by the intermediate value theorem, there exists a root of f in the interval $[a_j, b_j)$, implying that

$$|x_j - x_*| \leqslant \frac{b_j - a_j}{2}.$$

Since $b_j - a_j = 2^{-j}(b_0 - a_0)$, the statement follows. □

Although the limit in (5.2) may not be well-defined (for example, x_1 may be a root of f), the error $x_j - x_*$ is bounded in absolute value by the sequence $(\tilde{e}_j)_{j \geqslant 0}$, where $\tilde{e}_j = (b_0 - a_0)2^{-(j+1)}$ by Proposition 5.1. Since the latter sequence exhibits linear convergence to 0, the convergence of the bisection method is said to be linear, by a slight abuse of terminology.

5.2 Fixed point methods

Let $\boldsymbol{x}_*$ denote a zero of the function $\boldsymbol{f}$. The idea of iterative methods for (5.1) is to construct, starting from an initial guess $\boldsymbol{x}_0$, a sequence $(\boldsymbol{x}_k)_{k=0,1,\dots}$ approaching $\boldsymbol{x}_*$. A number of iterative methods for solving (5.1) are based on an iteration of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}(\boldsymbol{x}_k), \tag{5.5}$$

for an appropriate continuous function F. Assume that $\boldsymbol{x}_k$ converges to some point $\boldsymbol{x}_* \in \mathbf{R}^n$ in the limit as $k \to \infty$. Then, taking the limit $k \to \infty$ in (5.5), we find that $\boldsymbol{x}_*$ satisfies

$$\boldsymbol{F}(\boldsymbol{x}_*) = \boldsymbol{x}_*.$$

Such a point $\boldsymbol{x}_*$ is called a *fixed point* of the function $\boldsymbol{F}$. Several definitions of the function $\boldsymbol{F}$ can be employed in order to ensure that a fixed point of $\boldsymbol{F}$ coincides with a zero of $\boldsymbol{f}$. One may, for example, define $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} - \alpha^{-1}\boldsymbol{f}(\boldsymbol{x})$, for some nonzero scalar coefficient α. Then $\boldsymbol{F}(\boldsymbol{x}_*) = \boldsymbol{x}_*$ if and only if $\boldsymbol{f}(\boldsymbol{x}_*) = 0$. Later in this chapter, in Section 5.4, we study two instances of numerical methods which can be recast in the form (5.5). Before this, we study the convergence of the iteration (5.5) for a general function $\boldsymbol{F}$.

5.3 Convergence of fixed point methods

Equation (5.5) may be viewed as a *discrete-time* dynamical system. In order to study the behavior of the system as $k \to \infty$, it is important to understand the concept of stability of a fixed point. The concept of stability appears also in the field of ordinary differential equations, which are *continuous-time* dynamical systems. Before we define this concept, we introduce the following notation for the open ball of radius δ around $\boldsymbol{x} \in \mathbf{R}^n$:

$$B_\delta(\boldsymbol{x}) := \big\{\boldsymbol{y} \in \mathbf{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\| < \delta\big\}.$$

Definition 5.1 (Stability of fixed points)**.** Let $(\boldsymbol{x}_k)_{k\geqslant 0}$ denote iterates obtained from (5.5) when starting from an initial vector $\boldsymbol{x}_0$. Then we say that a fixed point $\boldsymbol{x}_*$ is

- an *attractor* if there exists a neighborhood $\mathcal{V}$ of $\boldsymbol{x}_*$ such that

$$\forall \boldsymbol{x}_0 \in \mathcal{V}, \qquad \boldsymbol{x}_k \xrightarrow[k\to\infty]{} \boldsymbol{x}_*. \tag{5.6}$$

  The largest neighborhood for which this is true, i.e. the set of values of $\boldsymbol{x}_0$ such that (5.6) holds true, is called the basin of attraction of $\boldsymbol{x}_*$.

- stable (in the sense of Lyapunov) if for all $\varepsilon > 0$, there exists $\delta > 0$ such that

$$\forall \boldsymbol{x}_0 \in B_\delta(\boldsymbol{x}_*), \qquad \forall k \in \mathbf{N}, \qquad \|\boldsymbol{x}_k - \boldsymbol{x}_*\| < \varepsilon.$$

- asymptotically stable if it is stable and an attractor.

- exponentially stable if there exists $C > 0$, $\alpha \in (0, 1)$, and $\delta > 0$ such that

$$\forall \boldsymbol{x}_0 \in B_\delta(\boldsymbol{x}_*), \quad \forall k \in \mathbf{N}, \qquad \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \leqslant C\alpha^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|.$$

- globally exponentially stable if there exists $C > 0$ and $\alpha \in (0, 1)$ such that

$$\forall \boldsymbol{x}_0 \in \mathbf{R}^n, \quad \forall k \in \mathbf{N}, \qquad \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \leqslant C\alpha^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|.$$

- unstable if it is not stable.

Clearly, global exponential stability implies exponential stability, which itself implies asymptotic stability and stability. If $\boldsymbol{x}_*$ is globally exponentially stable, then $\boldsymbol{x}_*$ is the unique fixed point of $\boldsymbol{F}$; showing this is the aim of Exercise 5.3. If $\boldsymbol{x}_*$ is an attractor, then the dynamical system (5.5) is said to be locally convergent to $\boldsymbol{x}_*$. The larger the basin of attraction of $\boldsymbol{x}_*$, the less careful we need to be when picking the initial guess $\boldsymbol{x}_0$. Global exponential stability of a fixed point can sometimes be shown provided that $\boldsymbol{F}$ satisfies a strong hypothesis.

Definition 5.2 (Lipschitz continuity)**.** A function $\boldsymbol{F}\colon \mathbf{R}^n \to \mathbf{R}^n$ is said to be *Lipschitz* continuous with constant L if

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n, \qquad \big\|\boldsymbol{F}(\boldsymbol{y}) - \boldsymbol{F}(\boldsymbol{x})\big\| \leqslant L\|\boldsymbol{y} - \boldsymbol{x}\|.$$

A function $\boldsymbol{F}\colon \mathbf{R}^n \to \mathbf{R}^n$ that is Lipschitz continuous with a constant $L < 1$ is called a *contraction*. For such a function, it is possible to prove that (5.5) has a unique globally exponentially stable fixed point.

Theorem 5.2. *Assume that $\boldsymbol{F}$ is a contraction. Then there exists a unique fixed point of (5.5), and it holds that*

$$\forall \boldsymbol{x}_0 \in \mathbf{R}^n, \quad \forall k \in \mathbf{N}, \qquad \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \leqslant L^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|. \tag{5.7}$$

Proof. Existence and uniqueness of the fixed point follows from the *Banach fixed point theorem*, see Theorem A.3, so here we show only global exponential convergence. Since $\boldsymbol{F}$ is a contraction, it holds that

$$\big\|\boldsymbol{x}_k - \boldsymbol{x}_*\big\| = \big\|\boldsymbol{F}(\boldsymbol{x}_{k-1}) - \boldsymbol{F}(\boldsymbol{x}_*)\big\| \leqslant L\|\boldsymbol{x}_{k-1} - \boldsymbol{x}_*\| \leqslant \ \ldots \ \leqslant L^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|, \tag{5.8}$$

which proves (5.7). □

It is possible to prove a weaker, local result under a less restrictive assumptions on the function $\boldsymbol{F}$.

Theorem 5.3. *Assume that $\boldsymbol{x}_*$ is a fixed point of (5.5) and that $\boldsymbol{F}\colon \mathbf{R}^n \to \mathbf{R}^n$ satisfies the local Lipschitz condition*

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \big\|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*)\big\| \leqslant L\|\boldsymbol{x} - \boldsymbol{x}_*\|, \tag{5.9}$$

with $0 \leqslant L < 1$ and $\delta > 0$. Then $\boldsymbol{x}_*$ is the unique fixed point of $\boldsymbol{F}$ in $B_\delta(\boldsymbol{x}_*)$ and, for all $\boldsymbol{x}_0 \in B_\delta(\boldsymbol{x}_*)$, it holds that

- *All the iterates $(\boldsymbol{x}_k)_{k\in\mathbf{N}}$ belong to $B_\delta(\boldsymbol{x}_*)$.*
- *The sequence $(\boldsymbol{x}_k)_{k\in\mathbf{N}}$ converges exponentially to $\boldsymbol{x}_*$.*

Proof. See Exercise 5.4. □

It is possible to guarantee that condition (5.9) holds provided that we have sufficiently good control of the derivatives of the function $\boldsymbol{F}$. The function $\boldsymbol{F}$ is said to be differentiable at $\boldsymbol{x}$ (in the sense of Fréchet) if there exists a linear operator $\mathrm{D}\boldsymbol{F}_{\boldsymbol{x}}\colon \mathbf{R}^n \to \mathbf{R}^n$ such that

$$\lim_{\boldsymbol{h}\to 0} \frac{\|\boldsymbol{F}(\boldsymbol{x}+\boldsymbol{h}) - \boldsymbol{F}(\boldsymbol{x}) - \mathrm{D}\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} = 0. \tag{5.10}$$

If $\boldsymbol{F}$ is differentiable, then all its first partial derivatives $\partial_j F_i$ exist and, in addition, it holds that $\mathrm{D}\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{h}) = \mathsf{J}_F(\boldsymbol{x})\boldsymbol{h}$ where $\mathsf{J}_F(\boldsymbol{x})$ is the Jacobian matrix of $\boldsymbol{F}$ at $\boldsymbol{x}$:

$$\mathsf{J}_F(\boldsymbol{x}) = \begin{pmatrix} \partial_1 F_1(\boldsymbol{x}) & \dots & \partial_n F_1(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \partial_1 F_n(\boldsymbol{x}) & \dots & \partial_n F_n(\boldsymbol{x}) \end{pmatrix}.$$

Proposition 5.4. *Let $\boldsymbol{x}_*$ be a fixed point of (5.5), and assume that there exists δ and a subordinate matrix norm such that $\boldsymbol{F}$ is differentiable everywhere in $B_\delta(\boldsymbol{x}_*)$ and*

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \|\mathsf{J}_F(\boldsymbol{x})\| \leqslant L < 1.$$

Then condition (5.9) is satisfied in the associated vector norm, and so the fixed point $\boldsymbol{x}_*$ is locally exponentially stable.

Proof. Let $\boldsymbol{x} \in B_\delta(\boldsymbol{x}_*)$. By the fundamental theorem of calculus and the chain rule, we have

$$\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*) = \int_0^1 \frac{\mathrm{d}}{\mathrm{d}t}\Big(\boldsymbol{F}\big(\boldsymbol{x}_* + t(\boldsymbol{x}-\boldsymbol{x}_*)\big)\Big)\mathrm{d}t = \int_0^1 \mathsf{J}_F\big(\boldsymbol{x}_* + t(\boldsymbol{x}-\boldsymbol{x}_*)\big)(\boldsymbol{x}-\boldsymbol{x}_*)\mathrm{d}t.$$

Therefore, it holds that

$$\|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*)\| \leqslant \int_0^1 \Big\|\mathsf{J}_F\big(\boldsymbol{x} + t(\boldsymbol{x}-\boldsymbol{x}_*)\big)\Big\|\,\mathrm{d}t\,\|\boldsymbol{x}-\boldsymbol{x}_*\| \leqslant \int_0^1 L\,\mathrm{d}t\,\|\boldsymbol{x}-\boldsymbol{x}_*\| = L\|\boldsymbol{x}-\boldsymbol{x}_*\|,$$

which is the statement. □

Remark 5.2. As a student observed during the lecture, in dimension $n = 1$, Proposition 5.4 can be proved by using the mean value theorem: since F is differentiable in $(x_* - \delta, x_* + \delta)$, there exists for all x in this interval a $\xi = \xi(x)$ also in this interval such that

$$F(x) - F(x_*) = F'(\xi)(x - x_*).$$

It then follows immediately that

$$\big|F(x) - F(x_*)\big| = |F'(\xi)(x-x_*)| \leqslant L\,|x - x_*|.$$

This proof does not carry over to the multi-dimensional setting, however.

In fact, it is possible to prove that a fixed point $\boldsymbol{x}_*$ is exponentially locally stable under an even weaker condition, involving only the derivative of $\boldsymbol{F}$ at $\boldsymbol{x}_*$.

Proposition 5.5. *Let $\boldsymbol{x}_*$ be a fixed point of (5.5) and that F is differentiable at $\boldsymbol{x}_*$ with*

$$\|\mathsf{J}_F(\boldsymbol{x}_*)\| = L < 1,$$

in a subordinate vector norm. Then the fixed point $\boldsymbol{x}_*$ is locally exponentially stable.

Proof. In this proof, the vector norm used is that associated with the matrix norm in the statement of the proposition. By the definition of differentiability (5.10), there exists for all $\varepsilon > 0$ a $\delta > 0$ such that

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*)\backslash\{\boldsymbol{x}_*\}, \qquad \frac{\|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*) - \mathsf{J}_F(\boldsymbol{x}_*)(\boldsymbol{x} - \boldsymbol{x}_*)\|}{\|\boldsymbol{x} - \boldsymbol{x}_*\|} \leqslant \varepsilon.$$

By the triangle inequality, this implies that for all $\boldsymbol{x} \in B_\delta(\boldsymbol{x}_*)$,

$$\begin{aligned}\|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*)\| &\leqslant \|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*) - \mathsf{J}_F(\boldsymbol{x}_*)(\boldsymbol{x} - \boldsymbol{x}_*)\| + \|\mathsf{J}_F(\boldsymbol{x}_*)(\boldsymbol{x} - \boldsymbol{x}_*)\| \\ &\leqslant \varepsilon\|\boldsymbol{x} - \boldsymbol{x}_*\| + \|\mathsf{J}_F(\boldsymbol{x}_*)\|\|(\boldsymbol{x} - \boldsymbol{x}_*)\| = (L + \varepsilon)\|\boldsymbol{x} - \boldsymbol{x}_*\|.\end{aligned}$$

We have thus shown that for all $\varepsilon > 0$, there exists $\delta > 0$ such that condition (5.9) is satisfied with constant $L + \varepsilon$. By taking ε sufficiently small, we can ensure that $L + \varepsilon < 1$, and so the fixed point $\boldsymbol{x}_*$ is locally exponentially stable by Theorem 5.3. □

The estimate in Theorem 5.2 suggests that when the fixed point iteration (5.5) converges, the convergence is linear. While this is usually the case, the convergence is superlinear if $\mathsf{J}_F(\boldsymbol{x}_*) = 0$.

Proposition 5.6. *Assume that $\boldsymbol{x}_*$ is a fixed point of (5.5) and that $\mathsf{J}_F(\boldsymbol{x}_*) = 0$. Then the convergence to $\boldsymbol{x}_*$ is superlinear, in the sense that if $\boldsymbol{x}_k \to \boldsymbol{x}_*$ as $k \to \infty$, then*

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|} = 0.$$

Proof. By Proposition 5.5, there exists $\delta > 0$ such that $(\boldsymbol{x}_k)_{k\geqslant 0}$ is a sequence converging to $\boldsymbol{x}_*$ for all $\boldsymbol{x}_0 \in B_\delta(\boldsymbol{x}_*)$. It holds that

$$\frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|} = \frac{\|\boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_*)\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|} = \frac{\|\boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_*) - \mathsf{J}_F(\boldsymbol{x}_*)(\boldsymbol{x}_k - \boldsymbol{x}_*)\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|}.$$

Since $\boldsymbol{x}_k - \boldsymbol{x}_* \to \boldsymbol{0}$ as $k \to \infty$, the right-hand side converges to 0 by (5.10). □

Similarly, if there exist $\delta > 0$, $C > 0$ and $q \in (1, \infty)$ such that

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \|\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_*)\| \leqslant C\|\boldsymbol{x} - \boldsymbol{x}_*\|^q, \tag{5.11}$$

then assuming that $(\boldsymbol{x}_k)_{k\geqslant 0}$ converges to $\boldsymbol{x}_*$, it holds for sufficiently large k that

$$\frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|^q} = \frac{\|\boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_*)\|}{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|^q} \leqslant C.$$

In this case, the order of convergence is at least q.

5.4 Examples of fixed point methods

As we mentioned in Section 5.2, there are several choices for the function $\boldsymbol{F}$ that guarantee the equivalence $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} \Leftrightarrow \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$.

5.4.1 The chord method

In the case where f is a function from $\mathbf{R}$ to $\mathbf{R}$, the simplest approach, sometimes called the *chord method*, is to define

$$F(x) = x - \alpha^{-1} f(x).$$

The fixed point iteration (5.4) in this case admits a simple geometric interpretation: at each step, the function f is approximated by the affine function $x \mapsto f(x_k) + \alpha(x - x_k)$, and the new iterate is defined as the zero of this affine function, i.e.

$$x_{k+1} = x_k - \alpha^{-1} f(x_k) = F(x_k). \tag{5.12}$$

This is illustrated in Figure 5.1. By Proposition 5.5, a sufficient condition to ensure local convergence is that

$$|F'(x_*)| = |1 - \alpha^{-1} f'(x_*)| < 1. \tag{5.13}$$

In order for this condition to hold true, the slope α must be of the same sign as $f'(x_*)$ and the inequality $|\alpha| \geqslant |f'(x_*)|/2$ must be satisfied. If $f'(x_*) = 0$, then the sufficient condition (5.13) is never satisfied; in this case, the convergence must be studied on a case-by-case basis. By Proposition 5.6, the convergence of the chord method is superlinear if $\alpha = f'(x_*)$. In practice, the solution x_* is unknown, and so this choice is not realistic. Nevertheless, the above reasoning suggests that, by letting the slope α vary from iteration to iteration in such a manner that α_k approaches $f'(x_*)$ as $k \to \infty$, fast convergence can be obtained. This is precisely what the Newton–Raphson method aims to achieve; see Section 5.4.2

When $\boldsymbol{f}$ is a function from $\mathbf{R}^n$ to $\mathbf{R}^n$, the above approach generalizes to

$$x_{k+1} = \boldsymbol{F}(\boldsymbol{x}_k), \qquad \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} - \mathsf{A}^{-1}\boldsymbol{f}(\boldsymbol{x}),$$

where A is an invertible matrix. The geometric interpretation of the method in this case is the following: at each step, the function $\boldsymbol{f}$ is approximated by the affine function $\boldsymbol{x} \mapsto \boldsymbol{x}_k + \mathsf{A}(\boldsymbol{x} - \boldsymbol{x}_k)$, and the next iterate is given by the unique zero of the latter function. Superlinear convergence is achieved when $\mathsf{A} = \mathsf{J}_f(\boldsymbol{x}_*)$. Notice that each iteration requires to calculate $\boldsymbol{y} := \mathsf{A}^{-1}\boldsymbol{f}(\boldsymbol{x}_k)$, which is generally achieved by solving the linear system $\mathsf{A}\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}_k)$.

5.4.2 The Newton–Raphson method

Let us first consider the case of a function from $\mathbf{R}$ to $\mathbf{R}$. A necessary condition for the Newton–Raphson method to apply is that f is differentiable. At each step, the function f is

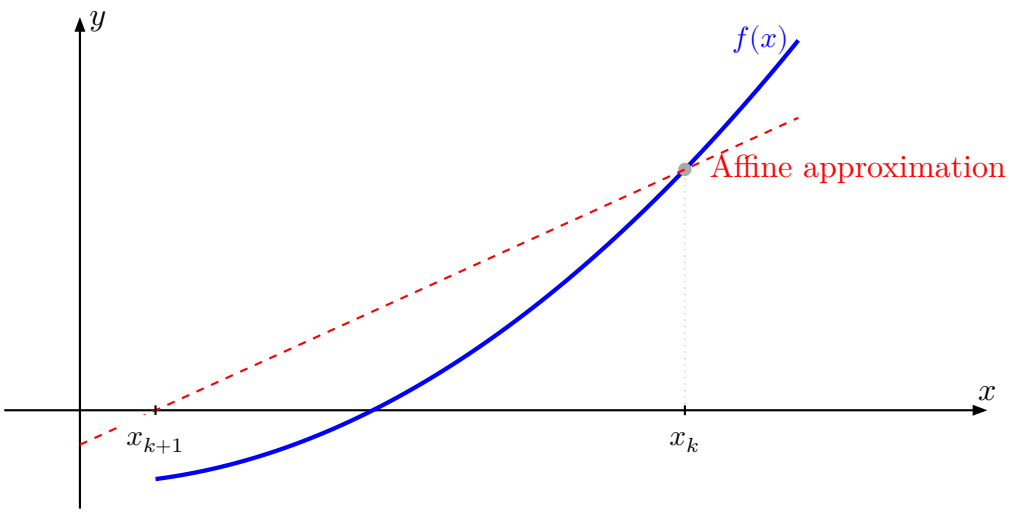


Figure 5.1: Graphical illustration of an iteration of the chord method.

approximated by the affine function $x \mapsto f(x_k) + f'(x_k)(x - x_k)$ and the unique zero of this function is returned. In other words, one iteration of the Newton–Raphson method reads

$$x_{k+1} = x_k - f'(x_k)^{-1} f(x_k). \tag{5.14}$$

For this iteration to be well-defined, it is necessary that $f'(x_k) \neq 0$. The Newton–Raphson method may be viewed as a variation on (5.12) where the slope α is adapted as the simulation progresses. If the method converges and f' is continuous, then $f'(x_k) \to f'(x_*)$ in the limit as $k \to \infty$, which is an indication that superlinear convergence could occur in view of our discussion in the previous section. Equation (5.14) may be recast as a fixed point iteration of the form (5.4) with

$$F(x) = x - \frac{f(x)}{f'(x)}.$$

If x_* is a simple root of f, that is if $f(x_*) = 0$ and $f'(x_*) \neq 0$, then x_* is a fixed point of the function F. If the function f is twice continuously differentiable, then the convergence of the Newton–Raphson method is superlinear by Proposition 5.6, because then

$$F'(x_*) = \frac{f(x_*) f''(x_*)}{f'(x_*)^2} = 0.$$

The geometric interpretation of the Newton–Raphson method in dimension 1 is the following: at each step, the function $\boldsymbol{f}$ is approximated by the affine function $x \mapsto x_k + f'(x_k)(x - x_k)$, which is *the tangent line to* f *at* x_k, and the next iterate is given by the unique zero of the latter function. This is illustrated in Figure 5.2.

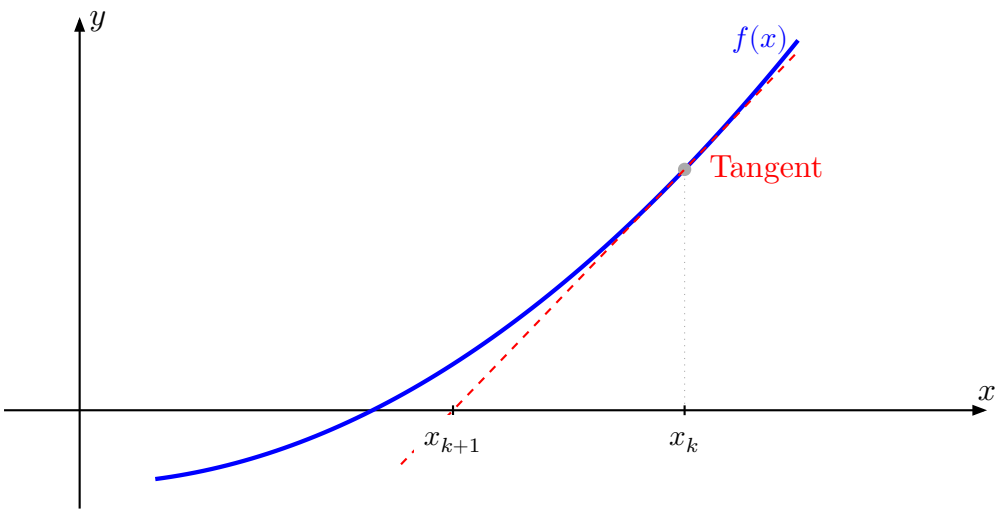


Figure 5.2: Graphical illustration of a Newton–Raphson iteration. The code used to create this figure is based on the answer https://tex.stackexchange.com/a/551205/125558 on LaTeX stack exchange.

The Newton–Raphson method may be generalized to nonlinear equations in $\mathbf{R}^n$ of the form (5.1). In this case $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} - \mathsf{J}_f(\boldsymbol{x})^{-1}\boldsymbol{f}(\boldsymbol{x})$, and so an iteration of the method reads

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathsf{J}_f(\boldsymbol{x}_k)^{-1}\boldsymbol{f}(\boldsymbol{x}_k). \tag{5.15}$$

In the rest of this section, we show that the iteration (5.15) is well-defined in a small neighborhood of a root of $\boldsymbol{f}$ under appropriate assumptions, and we demonstrate the *second order* convergence of the method, first in dimension 1 under simplifying assumption involving the second derivative of f, and then in the multi-dimensional setting under more general assumptions.

Convergence in the one-dimensional setting

We assume in this section that $(x_k)_{k\geqslant 0}$ is generated from the Newton–Raphson method (5.14) and prove the following result.

Theorem 5.7 (Quadratic convergence of Newton–Raphson)**.** *Assume that $f \in C^2(\mathbf{R})$ and that the following assumptions are satisfied:*

- *The first derivative of f is uniformly bounded away from zero:*

$$\inf_{x\in\,\mathbf{R}} |f'(x)| = m > 0.$$

- *The second derivative of f is uniformly bounded from above in absolute value:*

$$\sup_{x\in\,\mathbf{R}} |f''(x)| = M < \infty.$$

Then $f(x)$ has a unique root x_* and it holds for all initial $x_0 \in \mathbf{R}$ that

$$\forall k \in \mathbf{N}, \qquad |x_{k+1} - x_*| \leqslant \frac{M}{2m}|x_k - x_*|^2. \tag{5.16}$$

Proof. By assumption, the function f is continuous and either strictly increasing everywhere or strictly decreasing everywhere. Therefore there exists a unique root $x_* \in \mathbf{R}$ of f. In order to prove (5.16), we note that

$$x_{k+1} - x_* = x_k - \frac{f(x_k)}{f'(x_k)} - x_* = \frac{1}{f'(x_k)}\Big(f'(x_k)(x_k - x_*) - f(x_k)\Big). \tag{5.17}$$

By Taylor's theorem, there is $\xi \in \mathbf{R}$ such that

$$f(x_*) = f(x_k) + f'(x_k)(x_* - x_k) + \frac{1}{2}f''(\xi)(x_* - x_k)^2.$$

Since x_* is a root of f, the left-hand side of this equation is zero. Combining this equation with (5.17), we deduce that

$$x_{k+1} - x_* = \frac{f''(\xi)(x_k - x_*)^2}{2f'(x_k)}.$$

Taking absolute values and using the assumptions gives

$$|x_{k+1} - x_*| \leqslant \frac{M}{2m}(x_k - x_*)^2,$$

which concludes the proof. □

Remark 5.3. As a corollary of Theorem 5.7, we obtain that the Newton–Raphson method is convergent if

$$|x_k - x_*| \leqslant \frac{2m}{M}.$$

Convergence in the multi-dimensional setting

As a first step towards a proof of quadratic convergence for the Newton–Raphson method in the multi-dimensional setting, we begin by proving the following preparatory lemma, which we will then employ in the particular case where the matrix-valued function A is equal to J_f.

Lemma 5.8. *Let* $\mathsf{A}\colon \mathbf{R}^n \to \mathbf{R}^{n\times n}$ *denote a matrix-valued function on* $\mathbf{R}^n$ *that is both continuous and nonsingular at* $\boldsymbol{x}_*$*, and let* $\boldsymbol{f}$ *be a function that is differentiable at* $\boldsymbol{x}_*$ *where* $\boldsymbol{f}(\boldsymbol{x}_*) = 0$*. Then the function*

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{x} - \mathsf{A}(\boldsymbol{x})^{-1}\boldsymbol{f}(\boldsymbol{x})$$

is well-defined in a neighborhood $B_\delta(\boldsymbol{x}_*)$ *of* $\boldsymbol{x}_*$*. In addition,* $\boldsymbol{G}$ *is differentiable at* $\boldsymbol{x}_*$ *with*

$$\mathsf{J}_G(\boldsymbol{x}_*) = \mathsf{I} - \mathsf{A}(\boldsymbol{x}_*)^{-1}\mathsf{J}_f(\boldsymbol{x}_*). \tag{5.18}$$

Proof. It holds that

$$\mathsf{A}(\boldsymbol{x}) = \Big(\mathsf{A}(\boldsymbol{x}_*) - \big(\mathsf{A}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x})\big)\Big) = \mathsf{A}(\boldsymbol{x}_*)\Big(\mathsf{I} - \mathsf{A}(\boldsymbol{x}_*)^{-1}\big(\mathsf{A}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x})\big)\Big). \tag{5.19}$$

Let $\beta = \|\mathsf{A}(\boldsymbol{x}_*)^{-1}\|$ and $\varepsilon = (2\beta)^{-1}$. By continuity of the matrix-valued function A, there exists $\delta > 0$ such that

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \|\mathsf{A}(\boldsymbol{x}) - \mathsf{A}(\boldsymbol{x}_*)\| \leqslant \varepsilon.$$

For $\boldsymbol{x} \in B_\delta(\boldsymbol{x}_*)$ we have $\|\mathsf{A}(\boldsymbol{x}_*)^{-1}\big(\mathsf{A}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x})\big)\| \leqslant \|\mathsf{A}(\boldsymbol{x}_*)^{-1}\|\|\mathsf{A}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x})\| \leqslant \beta\varepsilon = \frac{1}{2}$, and so Lemma 4.2 implies that the second factor on the right-hand side of (5.19) is invertible with a norm bounded from above by 2. Therefore, we deduce that $\mathsf{A}(\boldsymbol{x})$ is invertible with

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \|\mathsf{A}(\boldsymbol{x})^{-1}\| \leqslant 2\|\mathsf{A}(\boldsymbol{x}_*)^{-1}\| = 2\beta, \tag{5.20}$$

which shows that $\boldsymbol{G}$ is well-defined in $B_\delta(\boldsymbol{x}_*)$. In order to prove (5.18), we need to show that

$$\lim_{\|\boldsymbol{h}\|\to 0} \frac{\Big\|\boldsymbol{G}(\boldsymbol{x}_* + \boldsymbol{h}) - \boldsymbol{G}(\boldsymbol{x}_*) - \big(\mathsf{I} - \mathsf{A}(\boldsymbol{x}_*)^{-1}\mathsf{J}_f(\boldsymbol{x}_*)\big)\boldsymbol{h}\Big\|}{\|\boldsymbol{h}\|} = 0$$

By definition of $\boldsymbol{G}$, and using the fact that $\boldsymbol{f}(\boldsymbol{x}_*) = \boldsymbol{0}$, we obtain that the argument of the norm in the numerator is equal to

$$\begin{aligned}
&\mathsf{A}(\boldsymbol{x}_*)^{-1}\boldsymbol{f}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h})^{-1}\boldsymbol{f}(\boldsymbol{x}_* + \boldsymbol{h}) + \mathsf{A}(\boldsymbol{x}_*)^{-1}\mathsf{J}_f(\boldsymbol{x}_*)\boldsymbol{h} \\
&\quad = \underbrace{\big(\mathsf{A}^{-1}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h})^{-1}\big)\mathsf{J}_f(\boldsymbol{x}_*)\boldsymbol{h}}_{=:\boldsymbol{v}_1} - \underbrace{\mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h})^{-1}\big(\boldsymbol{f}(\boldsymbol{x}_* + \boldsymbol{h}) - \boldsymbol{f}(\boldsymbol{x}_*) - \mathsf{J}_f(\boldsymbol{x}_*)\boldsymbol{h}\big)}_{=:\boldsymbol{v}_2}.
\end{aligned}$$

Noting that $\mathsf{A}^{-1}(\boldsymbol{x}_*) - \mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h})^{-1} = \mathsf{A}(\boldsymbol{x}_*)^{-1}\big(\mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h}) - \mathsf{A}(\boldsymbol{x}_*)\big)\mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h})^{-1}$, we bound the norm of the first term on the right-hand side as follows:

$$\forall \boldsymbol{h} \in B_\delta(\boldsymbol{0}), \qquad \|\boldsymbol{v}_1\| \leqslant 2\beta^2\|\mathsf{A}(\boldsymbol{x}_* + \boldsymbol{h}) - \mathsf{A}(\boldsymbol{x}_*)\|\|\mathsf{J}_f(\boldsymbol{x}_*)\|\|\boldsymbol{h}\|.$$

Clearly $\|\boldsymbol{v}_1\|/\|\boldsymbol{h}\| \to 0$ is the limit as $\boldsymbol{h} \to \boldsymbol{0}$ by continuity of the matrix function A. It also holds that $\|\boldsymbol{v}_2\|/\|\boldsymbol{h}\| \to 0$ by differentiability of $\boldsymbol{f}$ at $\boldsymbol{x}_*$, which concludes the proof. □

Using this lemma, we can show the following result on the convergence of the multi-dimensional Newton–Raphson method.

Theorem 5.9 (Convergence of Newton–Raphson)**.** *Let $\boldsymbol{f}\colon \mathbf{R}^n \to \mathbf{R}^n$ denote a function that is differentiable in a neighborhood $B_\delta(\boldsymbol{x}_*)$ of a point $\boldsymbol{x}_*$ where $\boldsymbol{f}(\boldsymbol{x}_*) = 0$. Assume that the Jacobian matrix $\mathsf{J}_f(\boldsymbol{x})$ is nonsingular and continuous at $\boldsymbol{x}_*$. Then $\boldsymbol{x}_*$ is an attractor of the Newton–Raphson iteration (5.15) and the convergence is superlinear.*

In addition, if there is $\alpha > 0$ such that the Lipschitz condition

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*), \qquad \|\mathsf{J}_f(\boldsymbol{x}) - \mathsf{J}_f(\boldsymbol{x}_*)\| \leqslant \alpha \|\boldsymbol{x} - \boldsymbol{x}_*\|$$

is satisfied, there exists $d \in (0, \delta)$ *and* $C > 0$ *such that*

$$\forall \boldsymbol{x}_k \in B_d(\boldsymbol{x}_*), \qquad \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| \leqslant C \|\boldsymbol{x}_k - \boldsymbol{x}_*\|^2.$$

In other words, the convergence is at least quadratic in $B_d(\boldsymbol{x}_*)$.

Proof. Using Lemma 5.8, we obtain that the Newton–Raphson update

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} - \mathsf{J}_f(\boldsymbol{x})^{-1} \boldsymbol{f}(\boldsymbol{x}),$$

is well-defined in a neighborhood $B_\delta(\boldsymbol{x}_*)$ of $\boldsymbol{x}_*$ for sufficiently small δ. In addition, the second statement in Lemma 5.8 gives that $\mathsf{J}_F(\boldsymbol{x}_*)^{-1} = \mathsf{I} - \mathsf{J}_F(\boldsymbol{x}_*)^{-1} \mathsf{J}_F(\boldsymbol{x}_*) = 0$, which establishes the superlinear convergence by Proposition 5.6.

In order to show that the convergence is quadratic, we begin by noticing that, since

$$\boldsymbol{f}(\boldsymbol{x}_k) = \int_0^t \frac{\mathrm{d}}{\mathrm{d}t} \boldsymbol{f}\big(\boldsymbol{x}_* + t(\boldsymbol{x}_k - \boldsymbol{x}_x)\big) \,\mathrm{d}t = \int_0^t \mathsf{J}_f\big(\boldsymbol{x}_* + t(\boldsymbol{x}_k - \boldsymbol{x}_x)\big)(\boldsymbol{x}_k - \boldsymbol{x}_*) \,\mathrm{d}t,$$

it holds for all $\boldsymbol{x}_k \in B_\delta(\boldsymbol{x}_*)$ that

$$\begin{aligned}
\|\boldsymbol{f}(\boldsymbol{x}_k) - \mathsf{J}_f(\boldsymbol{x}_*)(\boldsymbol{x}_k - \boldsymbol{x}_*)\| &= \left\| \int_0^1 \Big(\mathsf{J}_f\big(\boldsymbol{x}_* + t(\boldsymbol{x}_k - \boldsymbol{x}_*)\big) - \mathsf{J}_f(\boldsymbol{x}_*) \Big) (\boldsymbol{x}_k - \boldsymbol{x}_*) \,\mathrm{d}t \right\| \\
&\leqslant \int_0^1 \Big\| \mathsf{J}_f\big(\boldsymbol{x}_* + t(\boldsymbol{x}_k - \boldsymbol{x}_*)\big) - \mathsf{J}_f(\boldsymbol{x}_*) \Big\| \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \,\mathrm{d}t \\
&\leqslant \int_0^1 \alpha t \|\boldsymbol{x}_k - \boldsymbol{x}_*\|^2 \,\mathrm{d}t \leqslant \frac{\alpha}{2} \|\boldsymbol{x}_k - \boldsymbol{x}_*\|^2. \qquad (5.21)
\end{aligned}$$

Let $d \in (0, \delta)$ be sufficiently small to ensure that

$$\forall \boldsymbol{x} \in B_d(\boldsymbol{x}_*), \qquad \|\mathsf{J}_f(\boldsymbol{x})^{-1}\| \leqslant 2 \|\mathsf{J}_f(\boldsymbol{x}_*)^{-1}\|.$$

There exists such a d by (5.20). Using the inequality (5.21), we have that for all $\boldsymbol{x}_k \in B_d(\boldsymbol{x}_*)$,

$$\begin{aligned}
\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\| &= \|\boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{x}_*\| = \|\boldsymbol{x}_k - \boldsymbol{x}_* - \mathsf{J}_f(\boldsymbol{x}_k)^{-1} \boldsymbol{f}(\boldsymbol{x}_k)\| \\
&= \Big\| \mathsf{J}_f(\boldsymbol{x}_k)^{-1} \big(\boldsymbol{f}(\boldsymbol{x}_k) - \mathsf{J}_f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}_*) \big) \Big\| \leqslant \|\mathsf{J}_f(\boldsymbol{x}_k)^{-1}\| \|\boldsymbol{f}(\boldsymbol{x}_k) - \mathsf{J}_f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}_*)\| \\
&\leqslant \|\mathsf{J}_f(\boldsymbol{x}_k)^{-1}\| \Big(\|\boldsymbol{f}(\boldsymbol{x}_k) - \mathsf{J}_f(\boldsymbol{x}_*)(\boldsymbol{x}_k - \boldsymbol{x}_*)\| + \|\mathsf{J}_f(\boldsymbol{x}_*) - \mathsf{J}_f(\boldsymbol{x}_k)\| \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \Big) \\
&\leqslant \frac{3\alpha}{2} \|\mathsf{J}_f(\boldsymbol{x}_k)^{-1}\| \|\boldsymbol{x}_k - \boldsymbol{x}_*\|^2 \leqslant 3\alpha \|\mathsf{J}_f(\boldsymbol{x}_*)^{-1}\| \|\boldsymbol{x}_k - \boldsymbol{x}_*\|^2,
\end{aligned}$$

which concludes the proof. □

5.4.3 The secant method

The Newton–Raphson method exhibits very fast convergence, but it requires the knowledge of the derivatives of the function $\boldsymbol{f}$. To conclude this chapter, we describe a root-finding algorithm, known as the secant method, that enjoys superlinear convergence but does not require the derivatives of $\boldsymbol{f}$. This method applies only when $\boldsymbol{f}$ is a function from $\mathbf{R}$ to $\mathbf{R}$, and so we drop the vector notation in the rest of this section.

Unlike the other methods presented so far in Section 5.2, the secant method *can not* be recast as a fixed point iteration of the form $x_{k+1} = F(x_k)$. Instead, it is of the more general form $x_{k+2} = F(x_k, x_{k+1})$. The geometric intuition behind the method in the following: given x_k and x_{k+1}, the function f is approximated by the unique linear function that passes through $\big(x_k, f(x_k)\big)$ and $\big(x_{k+1}, f(x_{k+1})\big)$, and the iterate x_{k+2} is defined as the root of this linear function. In other words, f is approximated as follows:

$$\tilde{f}(x) = f(x_k) + \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}(x - x_k).$$

Solving $\tilde{f}(x) = 0$ gives the following expression for x_{k+2}:

$$x_{k+2} = \frac{f(x_{k+1})x_k - f(x_k)x_{k+1}}{f(x_{k+1}) - f(x_k)}, \tag{5.22}$$

Showing the convergence of the secant method rigorously under general assumptions is tedious, so in this course we restrict our attention to the case where f is a quadratic function. Extending the proof of convergence to a more general smooth function can be achieved by using a quadratic Taylor approximation of f around the root x_*, which is accurate in a close neighborhood of x_*.

Theorem 5.10 (Convergence of the secant method)**.** *Assume that f is a convex quadratic polynomial with a simple root at x_* and that the secant method converges: $\lim_{k\to\infty} x_k = x_*$. Then the order of convergence is given by the golden ratio*

$$\varphi = \frac{1 + \sqrt{5}}{2}.$$

More precisely, there exists a positive real number y_∞ such that

$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|^\varphi} = y_\infty. \tag{5.23}$$

Proof. Equation (5.22) implies that

$$x_{k+2} - x_* = \frac{f(x_{k+1})(x_k - x_*) - f(x_k)(x_{k+1} - x_*)}{f(x_{k+1}) - f(x_k)}.$$

By assumption, the function f may be expressed as

$$f(x) = \lambda(x - x_*) + \mu(x - x_*)^2, \qquad \lambda \neq 0.$$

Substituting this expression in (5.4.3) and letting $e_k = x_k - x_*$, we obtain

$$e_{k+2} = \frac{\mu e_k e_{k+1}(e_{k+1} - e_k)}{\lambda(e_{k+1} - e_k) + \mu(e_{k+1}^2 - e_k^2)} = \frac{\mu e_k e_{k+1}}{\lambda + \mu(e_{k+1} + e_k)}.$$

Rearranging this equation, we have

$$\frac{e_{k+2}}{e_{k+1}} = \frac{\mu e_k}{\lambda + \mu(e_{k+1} + e_k)}. \tag{5.24}$$

By assumption, the right-hand side converges to zero, and so the left-hand side must also converge to zero; the convergence is superlinear.

To conclude the proof, we first reason formally in order to guess the order convergence, and then give a rigorous proof that our guess is correct. If e_k is small, then it holds approximately by (5.24) that

$$\frac{e_{k+2}}{e_{k+1}} \approx \mu e_k. \tag{5.25}$$

Assume that there exists $q > 0$ such that the equation $e_{k+1} = Ce_k^q$ is valid for all k. Then it holds that $e_{k+2} = Ce_{k+1}^q = C(Ce_k^q)^q$ and (5.25) enables to determine q:

$$\frac{C(Ce_k^q)^q}{Ce_k^q} = \frac{\mu}{\lambda} e_k \qquad \Rightarrow \qquad C^q e_k^{q^2-q} = \frac{\mu}{\lambda} e_k \qquad \Rightarrow \qquad q^2 - q - 1 = 0. \qquad \Rightarrow \qquad q = \varphi.$$

Now comes the rigorous justification. Take absolute values in (5.24) to obtain, after rearranging,

$$\frac{|e_{k+2}|}{|e_{k+1}|^\varphi} = \left(\frac{|e_{k+1}|}{|e_k|^{\frac{1}{\varphi-1}}}\right)^{1-\varphi} \frac{\mu}{|\lambda + \mu(e_{k+1} + e_k)|} = \left(\frac{|e_{k+1}|}{|e_k|^\varphi}\right)^{1-\varphi} \frac{|\mu|}{|\lambda + \mu(e_{k+1} + e_k)|},$$

where we used that $\varphi = \frac{1}{\varphi-1}$, since φ is a root of the equation $\varphi^2 - \varphi - 1 = 0$. Thus, introducing the ratio $y_k = |e_{k+1}|/|e_k|^\varphi$, we have

$$y_{k+1} = y_k^{1-\varphi} \frac{|\mu|}{|\lambda + \mu(e_{k+1} + e_k)|}.$$

Taking logarithms in this equation, we deduce

$$\log(y_{k+1}) = (1-\varphi)\log(y_k) + c_k, \qquad c_k := \log\left(\frac{|\mu|}{|\lambda + \mu(e_{k+1} + e_k)|}\right).$$

This is a recurrence equation for $\log(y_k)$, whose explicit solution can be obtained from the variation-of-constants formula:

$$\log(y_k) = (1-\varphi)^{k-1}\log(y_1) + \sum_{i=1}^{k-1}(1-\varphi)^{k-1-i}c_i.$$

Since $(c_k)_{k \geqslant 0}$ converges to the constant $c_\infty = \log|\mu/\lambda|$ by the assumption that $e_k \to 0$, the sequence $\big(\log(y_k)\big)_{k \geqslant 0}$ converges to c_∞/φ (prove this!). Therefore, by continuity of the exponential function, it holds that

$$y_k = \exp\big(\log(y_k)\big) \xrightarrow[k\to\infty]{} \exp\left(\frac{c_\infty}{\varphi}\right) = \left|\frac{\mu}{\lambda}\right|^{\frac{1}{\varphi}}$$

and so we deduce (5.23). □

5.5 A numerical experiment

To conclude this chapter, we present the results of a numerical experiment. Specifically, we consider four different fixed point methods for calculating the square root of 2, i.e. for solving the nonlinear equation

$$f(x) := x^2 - 2 = 0. \tag{5.26}$$

The unique positive solution to this equation is $x_* = \sqrt{2}$. The methods we consider are the following:

- The chord method with large $\alpha = 10$.
- The chord method with the optimal parameter α, which is such that $F'(x_*) = 0$. The optimum value for α for solving (5.26) is given by $\alpha_* = 2\sqrt{2}$.
- The Newton–Raphson method, where each iteration is given by

  $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2 - 2}{2x_k} = \frac{1}{2}\left(x_k + \frac{2}{x_k}\right) =: F(x_k),$$

  with

  $$F(x) = \frac{1}{2}\left(x + \frac{2}{x}\right).$$

  Notice that $F'(x_*) = 0$ and that $F \in C^2\big((0,\infty)\big)$. Therefore, by Taylor's theorem it holds for all $x \in (x_* - 1, x_* + 1)$ that

  $$|F(x) - F(x_*)| = \left|F''\big(\xi(x)\big)\right| \leqslant L(x - x_*)^2, \qquad L := \sup_{|x - x_*| \leqslant 1} |F''(x)|.$$

  We deduce that the convergence is at least quadratic by (5.11).

Remark 5.4. Note that the ancient Babylonian method coincides with the Newton–Raphson method applied to (5.26).

The following code implements these methods. Note that we use the arbitrary precision `BigFloat` format with a precision we manually set to 2000 bits, which enables using a very small ε in the stopping criterion.

```
function count_digits(x, y)
    xdigits = split(string(x), "")
    ydigits = split(string(y), "")
    len = min(length(xdigits), length(ydigits))
    for i in 1:len
        xdigits[i] != ydigits[i] && return i-2
    end
end

function my_sqrt(a)
    exact = sqrt(a)
    f(x) = x*x - a
    fp(x) = 2x

    # Uncomment desired line
    F(x) = x - f(x)/10      # Chord method
    # F(x) = x - f(x)/(2√a)   # Chord method with optimal α
    # F(x) = 1/2 * (x + a/x)  # Babylonian / Newton Raphson

    r, ε = 1, 1e-200
    while abs(f(r)) > ε
        r = F(r)
        digits = ceil(Int, -log10(abs(r - exact)))
        println("Number of correct digits: $digits")
    end
end

# Sets the precision of BigFloats to 1000 bits
setprecision(2000)
my_sqrt(BigFloat(2))
```

For each of the methods, the number of correct digits of the approximation as the iterations progress is illustrated in Table 5.1. Observe that for all the methods except the first one, the number of correct digits is approximately doubled at each iteration, which is consistent with quadratic convergence.

Method	Chord $\alpha = 10$	Chord $\alpha = 2\sqrt{2}$	Newton–Raphson
# Iterations	**1357**	**8**	**9**
# Correct digits $i = 1$	1	1	1
# Correct digits $i = 2$	1	3	3
# Correct digits $i = 3$	1	6	6
# Correct digits $i = 4$	1	12	12
# Correct digits $i = 5$	1	26	24
# Correct digits $i = 6$	1	53	48
# Correct digits $i = 7$	1	107	97
# Correct digits $i = 8$	1	214	196
# Correct digits $i = 9$	1	n/a	392

Table 5.1: Comparison of different fixed point methods for calculating $\sqrt{2}$. Here i denotes the iteration index.

5.6 Exercises

Exercise 5.1. *Implement the bisection method for finding the solution(s) to the equation*

$$x = \cos(x).$$

Exercise 5.2. *Find a discrete-time dynamical system over* $\mathbf{R}$ *of the form*

$$x_{k+1} = F(x_k)$$

for which 0 *is an attractor but is not stable.*
Hint: *Use a function* F *that is discontinuous.*

Exercise 5.3. *Show that if* $\boldsymbol{x}_*$ *is a globally exponentially stable fixed point of* F*, then* F *does not have any other fixed point:* $\boldsymbol{x}_*$ *is the unique fixed point.*

Exercise 5.4. *Prove Theorem 5.3.*

Exercise 5.5. *Let* $\boldsymbol{x}_*$ *be a fixed point of (5.5). Show that if*

$$\rho\big(\mathsf{J}_F(\boldsymbol{x}_*)\big) < 1,$$

then $\boldsymbol{x}_*$ *is locally exponentially stable. It is sufficient by Proposition 5.5 to find a subordinate matrix norm such that* $\|\mathsf{J}_F(\boldsymbol{x}_*)\| < 1$*. In other words, this exercise amounts to showing that for any matrix* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *with* $\rho(\mathsf{A}) < 1$*, there exists a matrix norm such that* $\|\mathsf{A}\| < 1$*.*
Hint: *One may employ a matrix norm of the form* $\|\mathsf{A}\|_{\mathsf{T}} := \|\mathsf{T}^{-1}\mathsf{A}\mathsf{T}\|_2$*, which is a subordinate norm by Exercise 4.10. The Jordan normal form is useful for constructing the matrix* T*, and equation (4.24) is also useful.*

Solution. Let $\mathsf{J} = \mathsf{P}^{-1}\mathsf{A}\mathsf{P}$ denote the Jordan normal form of A, and let

$$\mathsf{E}_\varepsilon = \begin{pmatrix} \varepsilon & & & \\ & \varepsilon^2 & & \\ & & \ddots & \\ & & & \varepsilon^n \end{pmatrix}$$

By Eq. (4.24), the matrix $\mathsf{J}_\varepsilon := \mathsf{E}_\varepsilon^{-1}\mathsf{J}\mathsf{E}_\varepsilon$ coincides with J, except that the first superdiagonal is multiplied by ε. Let D denote the diagonal part of J_ε. We have that

$$\|\mathsf{J}_\varepsilon - \mathsf{D}\|_2 = \sqrt{\lambda_{\max}(\mathsf{E}_\varepsilon^T \mathsf{E}_\varepsilon)}.$$

The matrix $\mathsf{E}_\varepsilon^T\mathsf{E}_\varepsilon$ is diagonal with entries equal to either 0 or ε^2, and so $\|\mathsf{J}_\varepsilon - \mathsf{D}\|_2 < \varepsilon$. By the triangle inequality, we have

$$\|\mathsf{J}_\varepsilon\| \leqslant \|\mathsf{D}\| + \|\mathsf{J}_\varepsilon - \mathsf{D}\|_2 \leqslant \rho(\mathsf{A}) + \varepsilon. \tag{5.27}$$

Let $\|\mathsf{A}\|_\varepsilon := \|\mathsf{E}_\varepsilon^{-1}\mathsf{P}^{-1}\mathsf{A}\mathsf{P}\mathsf{E}_\varepsilon\|$. By Exercise 4.10 with $\mathsf{T} = \mathsf{P}\mathsf{E}_\varepsilon$, this is indeed a subordinate matrix norm. By (5.27) and the assumption that $\rho(\mathsf{A}) < 1$, it is clear that $\|\mathsf{A}\|_\varepsilon < 1$ provided that ε is sufficiently small. △

Remark 5.5. A corollary of Exercise 4.10 is that the spectral radius of a matrix A is *the infimum* of $\|\mathsf{A}\|$ over all subordinate matrix norms.

Exercise 5.6. *Calculate* $x = \sqrt[3]{3 + \sqrt[3]{3 + \sqrt[3]{3 + \sqrt{\dots}}}}$ *using the bisection method.*

Exercise 5.7. *Solve the equation* $f(x) = \mathrm{e}^x - 2 = 0$ *using a fixed point iteration of the form*

$$x_{k+1} = F(x_k), \qquad F(x) = x - \alpha^{-1} f(x).$$

Using your knowledge of the exact solution $x_* = \log 2$*, write a sufficient condition on* α *to guarantee that* x_* *is locally exponentially stable. Verify your findings numerically and plot, using a logarithmic scale for the* y *axis, the error in absolute value as a function of* k*.*

Exercise 5.8. *Implement the Newton–Raphson method for solving* $f(x) = \mathrm{e}^x - 2 = 0$*, and plot the error in absolute value as a function of the iteration index* k*.*

Exercise 5.9. *Find the point* (x, y) *on the parabola* $y = x^2$ *that is closest to the point* $(3, 1)$*.*

Exercise 5.10. *Consider the linear system*

$$\begin{cases} y = (x-1)^2 \\ x^2 + y^2 = 4 \end{cases}$$

By drawing these two constraints in the xy *plane, find an approximation of the solution(s). Then calculate the solution(s) using a fixed-point method.*

Exercise 5.11. *Find solutions* (ψ, λ)*, with* $\lambda > 0$*, to the following eigenvalue problem:*

$$\psi'' = -\lambda^2 \psi, \qquad \psi(0) = 0, \qquad \psi'(1) = \psi(1).$$

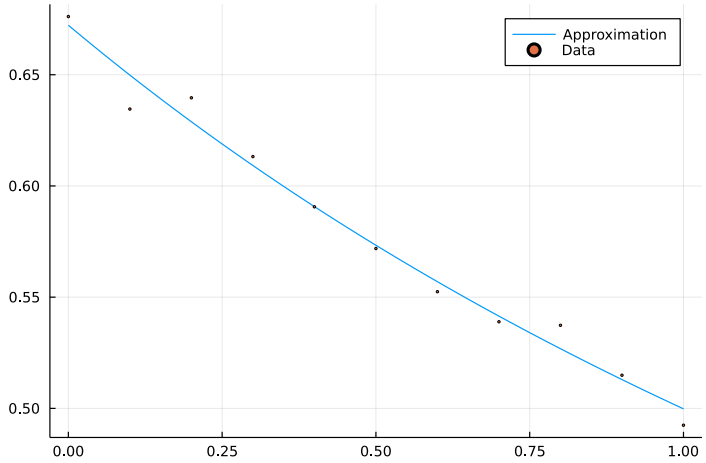


Figure 5.3: Solution to Exercise 5.12.

Exercise 5.12. *Suppose that we have n data points (x_i, y_i) of an unknown function $y = f(x)$. We wish to approximate f by a function of the form*

$$\tilde{f}(x) = \frac{a}{b + x}$$

by minimizing the sum of squares

$$\sum_{i=1}^{n} \left| \tilde{f}(x_i) - y_i \right|^2.$$

Write a system of nonlinear equations that the minimizer (a, b) must satisfy, and solve this system using the Newton–Raphson method starting from $(1, 1)$. The data is given below:

```
x = [0.0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1.0]
y = [0.6761488864859304; 0.6345697680852508; 0.6396283580587062;
0.6132010027973919;
     0.5906142598705267; 0.5718728461471725; 0.5524549902830562;
0.538938885654085;
     0.5373495476994958; 0.514904589752926; 0.49243437874655027]
```

Plot the data points together with the function $\tilde{f}$ over the interval $[0, 1]$. Your plot should look like Figure 5.3.

Exercise 5.13 (Nonlinear least-squares)**.** *Suppose that we are given n data points (x_i, y_i) of an unknown function $y = f(x)$. We wish to approximate f by a straight line*

$$\tilde{f}(x) = ax + b$$

by minimizing the sum of squared Euclidean distances *between the data points and the straight line $\tilde{f}$. Since the distance between a point (x_i, y_i) and the straight line is given by*

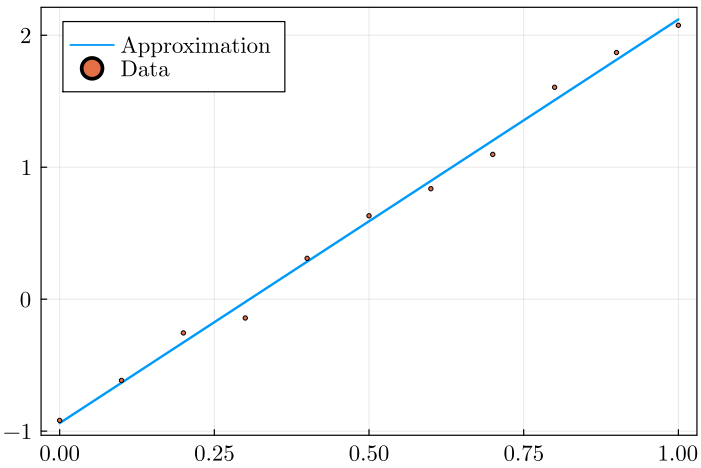


Figure 5.4: Solution to Exercise 5.13.

$$|y_i - ax_i - b\frac{|}{\sqrt{1+a^2}},$$

the objective function to minimize is given by

$$J(a,b) := \sum_{i=1}^{n} \frac{(y_i - ax_i - b)^2}{1+a^2}.$$

This is a smooth function of a and b, and so a necessary condition for a pair $(a_*, b_*) \in \mathbf{R}^2$ to be a minimizer is that

$$\nabla J(a_*, b_*) = \mathbf{0},$$

which is a nonlinear equation for the unknowns a_* and b_*. Solve this equation by using the Newton–Raphson method initialized at $(1,1)$, and then plot the data points together with the function $\tilde{f}$ over the interval $[0,1]$. Your plot should look like Figure 5.4. The data is given hereafter:

```
x = [0.0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1.0]
y = [-0.9187980789440975; -0.6159791344678258; -0.25568734869121856;
     -0.14269370171581808; 0.3094396057228459; 0.6318327173549161;
     0.8370437988106428; 1.0970402798788812; 1.6057799131867696;
     1.869090784869698; 2.075369730726694]
```

5.7 Discussion and bibliography

The content of this chapter is largely based on the lecture notes [15]. Several of the exercises are taken or inspired from [7]. The proof of convergence of the secant method is inspired from the general proof presented in the short paper [17]. For a detailed treatment of iterative methods for nonlinear equations, see the book [9].

Chapter 6

Numerical computation of eigenvalues

6.1 Numerical methods for eigenvalue problems: general remarks 152
6.2 Simple vector iterations 152
6.2.1 The power iteration 153
6.2.2 Inverse iteration 155
6.2.3 Rayleigh quotient iteration 156
6.3 Methods based on a subspace iteration 157
6.3.1 Simultaneous iteration 157
6.3.2 The QR algorithm 161
6.4 Projection methods 162
6.4.1 Projection method in a Krylov subspace 164
6.4.2 The Arnoldi iteration 164
6.4.3 The Lanczos iteration 166
6.5 Exercises 167
6.6 Discussion and bibliography 172

Introduction

Calculating the eigenvalues and eigenvectors of a matrix is a task often encountered in scientific and engineering applications. Eigenvalue problems naturally arise in quantum physics, solid mechanics, structural engineering and molecular dynamics, to name just a few applications. The aim of this chapter is to present an overview of the standard methods for calculating eigenvalues and eigenvectors numerically. We focus predominantly on the case of a Hermitian matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$, which is technically simpler and arises in many applications. The reader is invited to go through the background material in Section A.6 before reading this chapter. The rest of this chapter is organized as follows

- In Section 6.1, we make general remarks concerning the calculation of eigenvalues.

- In Section 6.2, we present standard methods based on a simple vector iteration.
- In Section 6.3, we present a method for calculating several eigenvectors simultaneously, based on iterating a subspace.
- In Section 6.4, we present method for constructing an approximation of the eigenvectors in a given subspace of $\mathbf{C}^n$.

6.1 Numerical methods for eigenvalue problems: general remarks

As mentioned in Section A.6, a complex number $\lambda \in \mathbf{C}$ is an eigenvalue of $\mathsf{A} \in \mathbf{C}^{n\times n}$ if and only if λ is a root of the characteristic polynomial $p_A\colon \mathbf{C} \to \mathbf{C}$ of A, which is given by

$$p_A(\lambda) = \det(\mathsf{A} - \lambda \mathsf{I}).$$

One may, therefore, calculate the eigenvalues of A by calculating the roots of the polynomial p_A using, for example, one of the methods presented in Chapter 5. While feasible for small matrices, this approach is not viable for large matrices, because the number of floating point operations required for calculating calculating the coefficients of the characteristic polynomial scales as the factorial of n.

In view of the prohibitive computational cost required for calculating the characteristic polynomial, other methods are required for solving large eigenvalue problems numerically. All the methods that we study in this chapter are of iterative nature. While some of them are aimed at calculating all the eigenpairs of the matrix A, other methods enable to calculate only a small number of eigenpairs at a lower computational cost, which is often desirable. Indeed, calculating all the eigenvalues of a large matrix is computationally expensive; on a personal computer, the following Julia code takes well over a second to terminate:

```
import LinearAlgebra
A = rand(2000, 2000)
LinearAlgebra.eigen(A)
```

In many applications, the matrix A is sparse, and in this case it is important to use algorithms for eigenvalue problems that do not destroy the sparsity structure. Note that the eigenvectors of a sparse matrix are generally not sparse.

To conclude this section, we introduce some notation used throughout this chapter. For a diagonalizable matrix A, we denote the eigenvalues by $\lambda_1, \ldots, \lambda_n$, with $|\lambda_1| \geqslant |\lambda_2| \geqslant \ldots \geqslant |\lambda_n|$. The associated normalized eigenvectors are denoted by $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. Therefore, it holds that

$$\mathsf{V}^{-1}\mathsf{A}\mathsf{V} = \mathsf{D} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \qquad \text{where } \mathsf{V} = (\boldsymbol{v}_1 \quad \ldots \quad \boldsymbol{v}_n).$$

6.2 Simple vector iterations

In this section, we present simple iterative methods aimed at calculating just one eigenvector of the matrix A, which we assume to be diagonalizable for simplicity.

6.2.1 The power iteration

The power iteration is the simplest method for calculating the eigenpair associated with the eigenvalue of A with largest modulus. Since the eigenvectors of A span $\mathbf{C}^n$, any vector $\boldsymbol{x}_0$ may be decomposed as

$$\boldsymbol{x}_0 = \alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_n \boldsymbol{v}_n. \tag{6.1}$$

The idea of the power iteration is to repeatedly left-multiply this vector by the matrix A, in order to amplify the coefficient of $\boldsymbol{v}_0$ relative to the other ones. Indeed, notice that

$$\mathsf{A}^k \boldsymbol{x}_0 = \lambda_1^k \alpha_1 \boldsymbol{v}_1 + \cdots + \lambda_n^k \alpha_n \boldsymbol{v}_n.$$

If λ_1 is strictly greater in modulus than the other eigenvalues, and if $\alpha_1 \neq 0$, then for large k the vector $\mathsf{A}^k \boldsymbol{x}_0$ is approximately aligned, in a sense made precise below, with the eigenvector $\boldsymbol{v}_1$. In order to avoid overflow errors at the numerical level, the iterates are normalized at each iteration. The power iteration is presented in Algorithm 8.

Algorithm 8 Power iteration

1: $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
2: **for** $i \in \{1, 2, \dots\}$ **do**
3: $\quad \boldsymbol{x} \leftarrow \mathsf{A}\boldsymbol{x}$
4: $\quad \boldsymbol{x} \leftarrow \boldsymbol{x}/\|\boldsymbol{x}\|$
5: **end for**

To precisely quantify the convergence of the power method, we introduce the notion of *acute angle* between vectors of $\mathbf{C}^n$.

$$\begin{aligned} \angle(\boldsymbol{x}, \boldsymbol{y}) &= \arccos\left(\frac{|\boldsymbol{x}^* \boldsymbol{y}|}{\sqrt{\boldsymbol{x}^* \boldsymbol{x}}\sqrt{\boldsymbol{y}^* \boldsymbol{y}}}\right) \\ &= \arcsin\left(\frac{\|(\mathsf{I} - \mathsf{P}_{\boldsymbol{y}})\boldsymbol{x}\|}{\|\boldsymbol{x}\|}\right), \qquad \mathsf{P}_{\boldsymbol{y}} := \frac{\boldsymbol{y}\boldsymbol{y}^*}{\boldsymbol{y}^* \boldsymbol{y}}. \end{aligned}$$

This definition generalizes the familiar notion of angle for vectors in $\mathbf{R}^2$ or $\mathbf{R}^3$, and we note that the angle function satisfies $\angle(\mathrm{e}^{\mathrm{i}\,\theta_1}\boldsymbol{x}, \mathrm{e}^{\mathrm{i}\,\theta_2}\boldsymbol{y}) = \angle(\boldsymbol{x}, \boldsymbol{y})$ as well as $\angle(\boldsymbol{x}, \boldsymbol{y}) \in [0, \pi/2]$. We can then prove the following convergence result.

Proposition 6.1 (Convergence of the power iteration)**.** *Suppose that* A *is diagonalizable and that* $|\lambda_1| > |\lambda_2|$*. Then, for every initial guess with* $\alpha_1 \neq 0$*, the sequence* $(\boldsymbol{x}_k)_{k \geqslant 0}$ *generated by the power iteration satisfies*

$$\lim_{k \to \infty} \angle(\boldsymbol{x}_k, \boldsymbol{v}_1) = 0.$$

Proof. By construction, it holds that

$$\boldsymbol{x}_k = \frac{\lambda_1^k \alpha_1 \boldsymbol{v}_1 + \cdots + \lambda_n^k \alpha_n \boldsymbol{v}_n}{\|\lambda_1^k \alpha_1 \boldsymbol{v}_1 + \cdots + \lambda_n^k \alpha_n \boldsymbol{v}_n\|} = \mathrm{e}^{\mathrm{i}\,\theta_k} \frac{\boldsymbol{v}_1 + \frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1} \boldsymbol{v}_2 + \cdots + \frac{\lambda_n^k \alpha_2}{\lambda_1^k \alpha_1} \boldsymbol{v}_n}{\left\| \boldsymbol{v}_1 + \frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1} \boldsymbol{v}_2 + \cdots + \frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1} \boldsymbol{v}_n \right\|}, \tag{6.2}$$

where

$$\mathrm{e}^{\mathrm{i}\,\theta_k} := \frac{\lambda_1^k \alpha_1}{|\lambda_1^k \alpha_1|}.$$

It follows from (6.2) that $\mathrm{e}^{-\mathrm{i}\,\theta_k} \boldsymbol{x}_k \to \boldsymbol{v}_1 / \|\boldsymbol{v}_1\| = \boldsymbol{v}_1$ in the limit as $k \to \infty$, where we employed the fact that $\|\boldsymbol{v}_1\| = 1$. Using the definition of the angle between two vectors in $\mathbf{C}^n$, and the continuity with respect to either argument of the $\mathbf{C}^n$ Euclidean inner product and of the arccos function, we obtain that

$$\begin{aligned} \angle(\boldsymbol{x}_k, \boldsymbol{v}_1) &= \arccos\left(\frac{|\boldsymbol{v}_1^* \boldsymbol{x}_k|}{\sqrt{\boldsymbol{v}_1^* \boldsymbol{v}_1} \sqrt{\boldsymbol{x}_k^* \boldsymbol{x}_k}} \right) = \arccos(|\boldsymbol{v}_1^* \boldsymbol{x}_k|) \\ &= \arccos\left(\left| \boldsymbol{v}_1^* \left(\mathrm{e}^{-\mathrm{i}\,\theta_k} \boldsymbol{x}_k \right) \right| \right) \xrightarrow[k \to \infty]{} \arccos(1) = 0, \end{aligned}$$

which concludes the proof. □

An inspection of the proof also reveals that the dominant term in the error, asymptotically in the limit as $k \to \infty$, is the one with coefficient $\frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1}$. Therefore, we deduce that

$$\angle(\boldsymbol{x}_k, \boldsymbol{v}_1) = \mathcal{O}\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right).$$

The convergence is slow if $|\lambda_2 / \lambda_1|$ is close to one, and fast if $|\lambda_2| \ll |\lambda_1|$. Once an approximation of the eigenvector $\boldsymbol{v}_1$ has been calculated, the corresponding eigenvalue λ_1 can be estimated from the *Rayleigh quotient:*

$$\rho_{\mathsf{A}} \colon \mathbf{C}_*^n \to \mathbf{C} \colon \boldsymbol{x} \mapsto \frac{\boldsymbol{x}^* \mathsf{A} \boldsymbol{x}}{\boldsymbol{x}^* \boldsymbol{x}}. \tag{6.3}$$

For any eigenvector $\boldsymbol{v}$ of A, the corresponding eigenvalue is equal to $\rho_{\mathsf{A}}(\boldsymbol{v})$. In order to study the error on the eigenvalue λ_1 for the power iteration, we assume for simplicity that A is Hermitian and that the eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are orthonormal. Substituting (6.2) in the Rayleigh quotient (6.3), we obtain

$$\rho_{\mathsf{A}}(\boldsymbol{x}_k) = \frac{\lambda_1 + \left| \frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1} \right|^2 \lambda_2 + \cdots + \left| \frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1} \right|^2 \lambda_n}{1 + \left| \frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1} \right|^2 + \cdots + \left| \frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1} \right|^2}.$$

Therefore, by reducing to a common denominator we deduce

$$|\rho_{\mathsf{A}}(\boldsymbol{x}_k) - \lambda_1| = \left| \frac{\lambda_1 + \left|\frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1}\right|^2 \lambda_2 + \cdots + \left|\frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1}\right|^2 \lambda_n}{1 + \left|\frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1}\right|^2 + \cdots + \left|\frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1}\right|^2} - \lambda_1 \right|$$
$$\leqslant \left|\frac{\lambda_2^k \alpha_2}{\lambda_1^k \alpha_1}\right|^2 |\lambda_2 - \lambda_1| + \cdots + \left|\frac{\lambda_n^k \alpha_n}{\lambda_1^k \alpha_1}\right|^2 |\lambda_n - \lambda_1| = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right).$$

The convergence of the eigenvalue in the particular case of a Hermitian matrix is faster than for a general matrix in $\mathbf{C}^{n\times n}$. For general matrices, it is possible to show using a similar argument that the error is of order $\mathcal{O}\left(|\lambda_2/\lambda_1|^k\right)$ in the limit as $k \to \infty$.

Essential convergence. It is useful at this point to introduce the concept of *essential convergence*. A sequence $(\boldsymbol{x}_k)$ in $\mathbf{C}^n$ is said to *converge essentially* to a vector $\boldsymbol{x}_\infty$ if there exists a sequence of complex numbers $(\mathrm{e}^{\mathrm{i}\,\varphi_k})$ of modulus 1 such that the sequence $(\mathrm{e}^{\mathrm{i}\,\varphi_k}\boldsymbol{x}_k)$ converges to $\boldsymbol{x}_\infty$. For a sequence $(\boldsymbol{x}_k)$ of normalized vectors, the essential convergence of $(\boldsymbol{x}_k)$ to $\boldsymbol{x}_\infty$ is equivalent to the convergence of $\angle(\boldsymbol{x}_k, \boldsymbol{x}_\infty)$ to 0. Proving this equivalence is the goal of Exercise 6.11. Reformulated in this new terminology, Proposition 6.1 states that the sequence $(\boldsymbol{x}_k)$ obtained from the power iteration converges essentially to $\boldsymbol{v}_1$.

6.2.2 Inverse iteration

The power iteration is simple but enables to calculate only the dominant eigenvalue of the matrix A, i.e. the eigenvalue of largest modulus. In addition, the convergence of the method is slow when $|\lambda_2| \approx |\lambda_1|$.

The inverse iteration enables a more efficient calculation of not only the dominant eigenvalue but also the other eigenvalues of A. It is based on applying the power iteration to $(\mathsf{A} - \mu\mathsf{I})^{-1}$, where $\mu \in \mathbf{C}$ is a shift. The eigenvalues of $(\mathsf{A} - \mu\mathsf{I})^{-1}$ are given by $(\lambda_1 - \mu)^{-1}, \ldots, (\lambda_n - \mu)^{-1}$, with associated eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. If $0 < |\lambda_J - \mu| < |\lambda_j - \mu|$ for all $j \neq J$, then the dominant eigenvalue of the matrix $(\mathsf{A} - \mu\mathsf{I})^{-1}$ is $(\lambda_J - \mu)^{-1}$, and so the power iteration applied to this matrix yields an approximation of the eigenvector $\boldsymbol{v}_J$. In other words, the inverse iteration with shift μ enables to calculate an approximation of the eigenvector of A corresponding to the eigenvalue nearest μ. The inverse iteration is presented in Algorithm 9. In practice, the inverse matrix $(\mathsf{A} - \mu\mathsf{I})^{-1}$ need not be calculated, and it is often preferable to solve a linear system at each iteration.

Algorithm 9 Inverse iteration

$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
for $i \in \{1, 2, \ldots\}$ **do**
    Solve $(\mathsf{A} - \mu\mathsf{I})\boldsymbol{y} = \boldsymbol{x}$
    $\boldsymbol{x} \leftarrow \boldsymbol{y}/\|\boldsymbol{y}\|$
end for
$\lambda \leftarrow \boldsymbol{x}^*\mathsf{A}\boldsymbol{x}/\boldsymbol{x}^*\boldsymbol{x}$
return $\boldsymbol{x}, \lambda$

An application of Proposition 6.1 immediately gives the following convergence result for the inverse iteration.

Proposition 6.2 (Convergence of the inverse iteration)**.** *Assume that* $\mathsf{A} \in \mathbf{C}^n$ *is diagonalizable and that there exist* J *and* K *such that*

$$0 < |\lambda_J - \mu| < |\lambda_K - \mu| \leqslant |\lambda_j - \mu| \qquad \forall j \neq J.$$

Assume also that $\alpha_J \neq 0$*, where* α_J *is the coefficient of* $\boldsymbol{v}_J$ *in the expansion of* $\boldsymbol{x}_0$ *given in (6.1). Then the iterates of the inverse iteration satisfy*

$$\lim_{k\to\infty} \angle(\boldsymbol{x}_k, \boldsymbol{v}_J) = 0.$$

More precisely,

$$\angle(\boldsymbol{x}_k, \boldsymbol{v}_J) = \mathcal{O}\left(\left|\frac{\lambda_J - \mu}{\lambda_K - \mu}\right|^k\right).$$

Proposition 6.2 states that $\boldsymbol{x}_k$ converges essentially to $\boldsymbol{v}_J$. Notice that the closer μ is to λ_J, the faster the inverse iteration converges. Note also that with $\mu = 0$, the inverse iteration enables to calculate the eigenvalue of A of smallest modulus.

6.2.3 Rayleigh quotient iteration

Since the inverse iteration is fast when μ is close to an eigenvalue λ_J, it is natural to wonder whether the method can be improved by progressively updating μ as the simulation progresses. Specifically, an approximation of the eigenvalue associated with the current vector may be employed in place of μ. This leads to the Rayleigh quotient iteration, presented in Algorithm 10.

Algorithm 10 Inverse iteration

$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
for $i \in \{1, 2, \dots\}$ **do**
    $\mu \leftarrow \boldsymbol{x}^* \mathsf{A} \boldsymbol{x} / \boldsymbol{x}^* \boldsymbol{x}$
    Solve $(\mathsf{A} - \mu \mathsf{I})\boldsymbol{y} = \boldsymbol{x}$
    $\boldsymbol{x} \leftarrow \boldsymbol{y} / \|\boldsymbol{y}\|$
end for
$\lambda \leftarrow \boldsymbol{x}^* \mathsf{A} \boldsymbol{x} / \boldsymbol{x}^* \boldsymbol{x}$
return $\boldsymbol{x}, \lambda$

It is possible to show that, when A is Hermitian, the Rayleigh quotient iteration converges to an eigenvector for almost every initial guess $\boldsymbol{x}_0$. Furthermore, if convergence to an eigenvector occurs, then μ converges cubically to the corresponding eigenvalue. See [12] and the references therein for more details.

6.3 Methods based on a subspace iteration

The subspace iteration resembles the power iteration but it is more general: not just one but several vectors are updated at each iteration.

6.3.1 Simultaneous iteration

Let $\mathsf{X}_0 = (\boldsymbol{x}_1 \quad \cdots \quad \boldsymbol{x}_p)$ denote an initial set of linearly independent vectors. Before we present the simultaneous iteration, we recall a statement concerning the QR decomposition of a matrix, which is related to the Gram–Schmidt orthonormalization process. We recall that the Gram–Schmidt method enables to construct, starting form an ordered set of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p\}$ in $\mathbf{C}^n$, a new set of vectors $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_p\}$ which are *orthonormal* and span the same subspace of $\mathbf{C}^n$ as the original vectors.

Proposition 6.3 (Reduced QR decomposition)**.** *Assume that $\mathsf{X} \in \mathbf{C}^{n\times p}$ has linearly independent columns. Then there exist a matrix $\mathsf{Q} \in \mathbf{C}^{n\times p}$ with orthonormal columns and an upper triangular matrix $\mathsf{R} \in \mathbf{C}^{p\times p}$ such that the following factorization holds:*

$$\mathsf{X} = \mathsf{QR}. \tag{6.4}$$

This decomposition is known as a reduced QR decomposition *if $p < n$, or simply* QR decomposition *if $p = n$, in which case X is a square matrix and Q is a unitary matrix. The decomposition is unique if we require that the diagonal elements of R are real and positive.*

Proof. The statement is clear when $p = 1$. Reasoning by induction, we assume that the result is true up to $p-1$, and prove that it then also holds true for p. We wish to show that there is a unique matrix $\mathsf{Q} \in \mathbf{C}^{n\times p}$ with orthonormal columns and a unique upper triangular matrix $\mathsf{R} \in \mathbf{C}^{p\times p}$ with real and positive diagonal elements such that (6.4) is satisfied. To this end, we decompose the matrices Q and R as follows:

$$\mathsf{Q} = \begin{pmatrix} \mathsf{Q}_{p-1} & \boldsymbol{q} \end{pmatrix}, \qquad \mathsf{R} = \begin{pmatrix} \mathsf{R}_{p-1} & \boldsymbol{r} \\ \mathbf{0}_{p-1}^T & r \end{pmatrix}. \tag{6.5}$$

Here $\mathsf{Q}_{p-1} \in \mathbf{C}^{n\times(p-1)}$ is a matrix with orthonormal columns, $\mathsf{R} \in \mathbf{C}^{(p-1)\times(p-1)}$ is an upper triangular matrix with positive real diagonal elements, $\boldsymbol{q} \in \mathbf{C}^n$ is a normalized vector orthogonal to all the columns of Q_{p-1}, $\boldsymbol{r} \in \mathbf{C}^{n-1}$ is a vector and $r \in \mathbf{R}_{>0}$ is a scalar. Let us also denote by $\mathsf{X}_{p-1} \in \mathbf{C}^{n\times(p-1)}$ the matrix containing the $p-1$ first columns of X, and by $\boldsymbol{x}_p \in \mathbf{C}^n$ the p-th column of X. Substituting (6.5) into (6.4), we then obtain

$$\begin{pmatrix} \mathsf{X}_{p-1} & \boldsymbol{x}_p \end{pmatrix} = \begin{pmatrix} \mathsf{Q}_{p-1}\mathsf{R}_{p-1} & \mathsf{Q}_{p-1}\boldsymbol{r} + \boldsymbol{q}r \end{pmatrix}, \tag{6.6}$$

By the induction hypothesis, there exist a unique choice of matrices Q_{p-1} and R_{p-1} with the required structure such that $\mathsf{X}_{p-1} = \mathsf{Q}_{p-1}\mathsf{R}_{p-1}$. Comparing the last column of both sides in (6.6), we obtain

Algorithm 11 Simultaneous iteration

1: $\mathsf{X} \leftarrow \mathsf{X}_0$
2: **for** $k \in \{1, 2, \dots\}$ **do**
3: $\quad \mathsf{Q}_k\mathsf{R}_k = \mathsf{A}\mathsf{X}_{k-1}$ (QR decomposition).
4: $\quad \mathsf{X}_k \leftarrow \mathsf{Q}_k$.
5: **end for**

$$\boldsymbol{x}_p = \mathsf{Q}_{p-1}\boldsymbol{r} + \boldsymbol{q}r. \tag{6.7}$$

Left-multiplying both sides by Q_{p-1}^* and employing the orthogonality between $\boldsymbol{q}$ and the columns of Q_{p-1}, we deduce that necessarily $\boldsymbol{r} = \mathsf{Q}_{p-1}^*\boldsymbol{x}_p$. It then follows from (6.7) that

$$\boldsymbol{q} = \frac{1}{r}(\boldsymbol{x}_p - \mathsf{Q}_{p-1}\mathsf{Q}_{p-1}^*\boldsymbol{x}_p), \qquad r = \|\boldsymbol{x}_p - \mathsf{Q}_{p-1}\mathsf{Q}_{p-1}^*\boldsymbol{x}_p\|.$$

It is simple to check that $\boldsymbol{q}$ is indeed orthogonal to the columns of Q, which concludes the proof. Note that $\mathsf{Q}_{p-1}\mathsf{Q}_{p-1}^*\boldsymbol{x}_p$ is the orthogonal projection of $\boldsymbol{x}_p$ onto the subspace spanned by the columns of Q_{p-1}. □

Note that the columns of the matrix Q of the decomposition coincide with the vectors that would be obtained by applying the Gram–Schmidt method to the columns of the matrix X. In fact, the Gram–Schmidt process is one of several methods by which the QR decomposition can be calculated in practice.

The simultaneous iteration method is presented in Algorithm 11. Like the normalization in the power iteration Algorithm 8, the QR decomposition performed at each step in Algorithm 11 enables to avoid overflow errors. Notice that when $p = 1$, the simultaneous iteration reduces to the power iteration. We emphasize that the factorization step at each iteration does not influence the subspace spanned by the columns of X. Therefore, this subspace after k iterations coincides with that spanned by the columns of the matrix $\mathsf{A}^k\mathsf{X}_0$. In fact, in exact arithmetic, it would be equivalent to perform the QR decomposition only once as a final step, after the `for` loop. Indeed, denoting by $\mathsf{Q}_k\mathsf{R}_k$ the QR decomposition of $\mathsf{A}\mathsf{X}_{k-1}$, we have

$$\begin{aligned} &\mathsf{X}_k = \mathsf{A}\mathsf{X}_{k-1}\mathsf{R}_k^{-1} = \mathsf{A}^2\mathsf{X}_{k-2}\mathsf{R}_{k-1}^{-1}\mathsf{R}_k^{-1} = \cdots = \mathsf{A}^k\mathsf{X}_0\mathsf{R}_1^{-1}\dots\mathsf{R}_k^{-1} \\ \Leftrightarrow\quad &\mathsf{X}_k(\mathsf{R}_k\dots\mathsf{R}_1) = \mathsf{A}^k\mathsf{X}_0. \end{aligned}$$

Since X_k has orthonormal columns and $\mathsf{R}_k\dots\mathsf{R}_1$ is an upper triangular matrix (see Exercise 4.3) with real positive elements on the diagonal (check this!), it follows that X_k can be obtained by QR factorization of $\mathsf{A}^k\mathsf{X}_0$. In order to show the convergence of the simultaneous iteration, we begin by proving the following preparatory lemma.

Lemma 6.4 (Continuity of the reduced QR decomposition)**.** *If $\mathsf{Q}_k\mathsf{R}_k \to \mathsf{QR}$, where $\mathsf{Q} \in \mathbf{C}^{n\times p}$ has orthonormal columns and $\mathsf{R} \in \mathbf{C}^{p\times p}$ is upper triangular with positive real entries on the diagonal, then $\mathsf{Q}_k \to \mathsf{Q}$.*

Proof. We reason by contradiction and assume there is $\varepsilon > 0$ and a subsequence $(\mathsf{Q}_{k_n})_{n \geqslant 0}$ such that $\|\mathsf{Q}_{k_n} - \mathsf{Q}\| \geqslant \varepsilon$ for all n. Since the set of matrices with normalized columns is a compact subset of $\mathsf{C}^{n\times p}$, there exists a further subsequence $(\mathsf{Q}_{k_{n_m}})_{m \geqslant 0}$ that converges to a limit Q_∞ which also has orthonormal columns and is at least ε away in norm from Q. But then

$$\mathsf{R}_{k_{n_m}} = \mathsf{Q}_{k_{n_m}}^* (\mathsf{Q}_{k_{n_m}} \mathsf{R}_{k_{n_m}}) \xrightarrow[m\to\infty]{} \mathsf{Q}_\infty^* (\mathsf{QR}) =: \mathsf{R}_\infty.$$

Since R_k is upper triangular with positive diagonal elements for all k, clearly R_∞ is also upper triangular with positive diagonal elements. But then $\mathsf{Q}_\infty \mathsf{R}_\infty = \mathsf{QR}$, and by uniqueness of the decomposition we deduce that $\mathsf{Q} = \mathsf{Q}_\infty$, which is a contradiction. □

Before presenting the convergence theorem, we introduce the following terminology: we say that $\mathsf{X}_k \in \mathbf{C}^{n\times p}$ converges essentially to a matrix X_∞ if each column of X_k converges essentially to the corresponding column of X_∞. We prove the convergence in the Hermitian case for simplicity. In the general case of $\mathsf{A} \in \mathbf{C}^{n\times n}$, it cannot be expected that X_k converges essentially to V, because the columns of X_k are orthogonal but eigenvectors may not be orthogonal. In this case, the columns of X_k converge not to the eigenvectors but to the so-called Schur vectors of A; see [12] for more information.

Theorem 6.5 (Convergence of the simultaneous iteration). *Assume that $\mathsf{A} \in \mathbf{C}^{n\times n}$ is Hermitian, that $\mathsf{X}_0 \in \mathbf{C}^{n\times p}$ has linearly independent columns, and finally that the subspace spanned by the column of X_0 satisfies*

$$\operatorname{col}(\mathsf{X}_0) \cap \operatorname{Span}\{\boldsymbol{v}_{p+1}, \ldots, \boldsymbol{v}_n\} = \emptyset. \tag{6.8}$$

If it holds that

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_p| > |\lambda_{p+1}| \geqslant |\lambda_{p+2}| \geqslant \ldots \geqslant |\lambda_n|, \tag{6.9}$$

then X_k converges essentially *to $\mathsf{V}_1 := (\boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{v}_p)$.*

Proof. Let $\mathsf{B} = \mathsf{V}^{-1}\mathsf{X}_0 \in \mathbf{C}^{n\times p}$, so that $\mathsf{X}_0 = \mathsf{VB}$, and note that $\mathsf{A}^k\mathsf{X}_0 = \mathsf{VD}^k\mathsf{B}$. We denote by $\mathsf{B}_1 \in \mathbf{C}^{p\times p}$ and $\mathsf{B}_2 \in \mathbf{C}^{(n-p)\times p}$ the upper $p \times p$ and lower $(n-p) \times p$ blocks of B, respectively. The matrix B_1 is nonsingular, otherwise the assumption (6.8) would not hold. Indeed, if there was a nonzero vector $\boldsymbol{z} \in \mathbf{C}^p$ such that $\mathsf{B}_1\boldsymbol{z} = 0$, then

$$\mathsf{X}_0\boldsymbol{z} = \mathsf{V}\begin{pmatrix}\mathsf{B}_1\\ \mathsf{B}_2\end{pmatrix}\boldsymbol{z} = (\mathsf{V}_1 \quad \mathsf{V}_2)\begin{pmatrix}\mathbf{0}\\ \mathsf{B}_2\boldsymbol{z}\end{pmatrix} = \mathsf{V}_2\mathsf{B}_2\boldsymbol{z}.$$

implying that $\mathsf{X}_0\boldsymbol{z} \in \operatorname{col}(\mathsf{X}_0)$ is a linear combination of the vectors in $\mathsf{V}_2 = (\boldsymbol{v}_{p+1} \quad \cdots \boldsymbol{v}_n)$, which contradicts the assumption. We also denote by D_1 and D_2 the $p \times p$ upper-left and the $(n-p) \times (n-p)$ lower-right blocks of D, respectively. From the expression of $\mathsf{A}^k\mathsf{X}_0$, we have

$$\begin{aligned}\mathsf{A}^k\mathsf{X}_0 &= \begin{pmatrix}\mathsf{V}_1 & \mathsf{V}_2\end{pmatrix}\begin{pmatrix}\mathsf{D}_1^k & \\ & \mathsf{D}_2^k\end{pmatrix}\begin{pmatrix}\mathsf{B}_1 \\ \mathsf{B}_2\end{pmatrix} = \mathsf{V}_1\mathsf{D}_1^k\mathsf{B}_1 + \mathsf{V}_2\mathsf{D}_2^k\mathsf{B}_2, \\ &= \big(\mathsf{V}_1 + \mathsf{V}_2\mathsf{D}_2^k\mathsf{B}_2\mathsf{B}_1^{-1}\mathsf{D}_1^{-k}\big)\mathsf{D}_1^k\mathsf{B}_1. \end{aligned} \tag{6.10}$$

The second term in the bracket on the right-hand side converges to zero in the limit as $k \to \infty$ by (6.9). Let $\widetilde{\mathsf{Q}}_k\widetilde{\mathsf{R}}_k$ denote the reduced QR decomposition of the bracketed term. By Lemma 6.4, we deduce from $\widetilde{\mathsf{Q}}_k\widetilde{\mathsf{R}}_k \to \mathsf{V}_1$ that $\widetilde{\mathsf{Q}}_k \to \mathsf{V}_1$. Rearranging (6.10), we have

$$\mathsf{A}^k\mathsf{X}_0 = \widetilde{\mathsf{Q}}_k(\widetilde{\mathsf{R}}_k\mathsf{D}_1^k\mathsf{B}_1).$$

Since the matrix between brackets is a $p \times p$ square invertible matrix, this equation implies that $\operatorname{col}(\mathsf{A}^k\mathsf{X}_0) = \operatorname{col}(\widetilde{\mathsf{Q}}_k)$. Denoting by $\mathsf{Q}_k\mathsf{R}_k$ the QR decomposition of $\mathsf{A}_k\mathsf{X}_0$, we therefore have $\operatorname{col}(\mathsf{Q}_k) = \operatorname{col}(\widetilde{\mathsf{Q}}_k)$, and so the projectors on these subspaces are equal. We recall that, for a set of orthonormal vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_p$ gathered in a matrix $\mathsf{Y} = (\boldsymbol{y}_1 \quad \cdots \quad \boldsymbol{y}_p)$, the projector on $\operatorname{col}(\mathsf{Y}) = \operatorname{Span}\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_p\} \subset \mathbf{C}^n$ is the square $n \times n$ matrix

$$\mathsf{Y}\mathsf{Y}^* = \boldsymbol{y}_1\boldsymbol{y}_1^* + \cdots + \boldsymbol{y}_p\boldsymbol{y}_p^*.$$

Consequently, the equality of the projectors implies $\mathsf{Q}_k\mathsf{Q}_k^* = \widetilde{\mathsf{Q}}_k\widetilde{\mathsf{Q}}_k{}^*$, and so $\mathsf{Q}_k\mathsf{Q}_k^* \to \mathsf{V}_1\mathsf{V}_1^*$. Similarly, noting that $\mathsf{Q}_k\texttt{[:,1:i]}\ \mathsf{R}\texttt{[1:i,1:i]}$ is the QR decompostion of the matrix $\mathsf{A}^k\mathsf{X}_0\texttt{[:,1:i]}$ for all $i \in \{1, \dots, p\}$, we obtain the convergence

$$\forall i \in \{1, \dots, p\}, \qquad \mathsf{Q}_k\texttt{[:,1:i]}\ \mathsf{Q}_k\texttt{[:,1:i]}^* \xrightarrow[k\to\infty]{} \boldsymbol{v}_1\boldsymbol{v}_1^* + \cdots + \boldsymbol{v}_i\boldsymbol{v}_i^*. \tag{6.11}$$

This is not surprising given that the first k columns of X_0 undergo a simultaneous iteration independent of the other columns.

Next, we establish the essential convergence of Q_k to V_1. To this end, we denote the columns of Q_k by $\boldsymbol{q}_1^{(k)}, \dots, \boldsymbol{q}_p^{(k)}$ and first show by induction that $\boldsymbol{q}_i^{(k)}\boldsymbol{q}_i^{(k)*} \to \boldsymbol{v}_i\boldsymbol{v}_i^*$ in the limit $k \to \infty$. For $i = 1$ this follows from (6.11). Assume now that $\boldsymbol{q}_\bullet^{(k)}\boldsymbol{q}_\bullet^{(k)*} \to \boldsymbol{v}_\bullet\boldsymbol{v}_\bullet^*$ for $\bullet$ up to $i-1$. Then

$$\begin{aligned}\boldsymbol{q}_i^{(k)}\boldsymbol{q}_i^{(k)*} &= \mathsf{Q}_k\texttt{[:,1:i]}\ \mathsf{Q}_k\texttt{[:,1:i]}^* - \boldsymbol{q}_1^{(k)}\boldsymbol{q}_1^{(k)*} - \cdots - \boldsymbol{q}_{i-1}^{(k)}\boldsymbol{q}_{i-1}^{(k)*} \\ &\xrightarrow[k\to\infty]{} \mathsf{V}_1\texttt{[:,1:i]}\ \mathsf{V}_1\texttt{[:,1:i]}^* - \boldsymbol{v}_1\boldsymbol{v}_1^* - \cdots - \boldsymbol{v}_{i-1}\boldsymbol{v}_{i-1}^* = \boldsymbol{v}_i\boldsymbol{v}_i^*.\end{aligned}$$

It remains to show that the convergence $\boldsymbol{q}_i^{(k)}\boldsymbol{q}_i^{(k)*} \to \boldsymbol{v}_i\boldsymbol{v}_i^*$ implies the desired essential convergence. Noting that $|a| = \sqrt{a\overline{a}}$ for every $a \in \mathbf{C}$, we have

$$|\boldsymbol{v}_i^*\boldsymbol{q}_i^{(k)}| = \sqrt{\boldsymbol{v}_i^*\boldsymbol{q}_i^{(k)}\boldsymbol{q}_i^{(k)*}\boldsymbol{v}_i} \xrightarrow[k\to\infty]{} \sqrt{\boldsymbol{v}_i^*\boldsymbol{v}_i\boldsymbol{v}_i^*\boldsymbol{v}_i} = 1,$$

Finally, observing that

$$\left\|\mathrm{e}^{-\mathrm{i}\,\theta_k}\boldsymbol{q}_i^{(k)} - \boldsymbol{v}_i\right\|^2 = 2 - 2|\boldsymbol{v}_i^*\boldsymbol{q}_i^{(k)}| \xrightarrow[k\to\infty]{} 0, \qquad \mathrm{e}^{\mathrm{i}\,\theta_k} = \frac{\boldsymbol{v}_i^*\boldsymbol{q}_i^{(k)}}{|\boldsymbol{v}_i^*\boldsymbol{q}_i^{(k)}|},$$

we conclude that $\boldsymbol{q}_i^{(k)}$ converges essentially to $\boldsymbol{v}_i$. □

In addition to this convergence result, it is possible to show that the error satisfies

$$\angle\Big(\mathrm{col}(\mathsf{X}_k), \mathrm{col}(\mathsf{V}_1)\Big) = \mathcal{O}\left(\left|\frac{\lambda_{p+1}}{\lambda_p}\right|^k\right).$$

Here, the angle between two subspaces $\mathcal{A}$ and $\mathcal{B}$ of $\mathbf{C}^n$ is defined as

$$\angle(\mathcal{A}, \mathcal{B}) = \max_{\boldsymbol{a}\in\mathcal{A}\setminus\{\mathbf{0}\}}\left(\min_{\boldsymbol{b}\in\mathcal{B}\setminus\{\mathbf{0}\}} \angle(\boldsymbol{a}, \boldsymbol{b})\right).$$

6.3.2 The QR algorithm

The QR algorithm, which is based on the QR decomposition, is one of the most famous algorithms for calculating *all* the eigenpairs of a matrix. We first present the algorithm and then relate it to the simultaneous iteration in Section 6.3.1. The method is presented in Algorithm 12.

Algorithm 12 QR algorithm

1: $\mathsf{X}_0 = \mathsf{A}$
2: **for** $i \in \{1, 2, \dots\}$ **do**
3: $\quad \mathsf{Q}_k\mathsf{R}_k = \mathsf{X}_{k-1}$ (QR decomposition)
4: $\quad \mathsf{X}_k = \mathsf{R}_k\mathsf{Q}_k$
5: **end for**

Successive iterates of the QR algorithm are related by the equation

$$\mathsf{X}_k = \mathsf{Q}_k^{-1}\mathsf{X}_{k-1}\mathsf{Q}_k = \mathsf{Q}_k^*\mathsf{X}_{k-1}\mathsf{Q}_k = \dots = (\mathsf{Q}_1 \dots \mathsf{Q}_k)^*\mathsf{X}_0(\mathsf{Q}_1 \dots \mathsf{Q}_k) \tag{6.12}$$

Therefore, all the iterates are related by a unitary similarity transformation, and so they all have the same eigenvalues as $\mathsf{X}_0 = \mathsf{A}$. Rearranging (6.12), we have

$$(\mathsf{Q}_1 \dots \mathsf{Q}_k)\mathsf{X}_k = \mathsf{A}(\mathsf{Q}_1 \dots \mathsf{Q}_k),$$

and so, introducing $\tilde{\mathsf{Q}}_k = \mathsf{Q}_1 \dots \mathsf{Q}_k$ and noting that $\mathsf{X}_k = \mathsf{Q}_{k+1}\mathsf{R}_{k+1}$ by the algorithm, we deduce

$$\tilde{\mathsf{Q}}_{k+1}\mathsf{R}_{k+1} = \mathsf{A}\tilde{\mathsf{Q}}_k.$$

This reveals that the matrix sequence $(\tilde{\mathsf{Q}}_k)_{k\geqslant 1}$ undergoes a simultaneous iteration and so, assuming that A is Hermitian with n distinct nonzero eigenvalues, we deduce that $\tilde{\mathsf{Q}}_k \to \mathsf{V}$ essentially in the limit as $k \to \infty$, by Theorem 6.5. As a consequence, by (6.12), it holds that $\mathsf{X}_k \to \mathsf{V}^*\mathsf{X}_0\mathsf{V} = \mathsf{D}$; in other words, the matrix X_k converges to a diagonal matrix with the eigenvalues of A on the diagonal.

6.4 Projection methods

In this section, we begin by presenting a general method for constructing an approximation of the eigenvectors of A in a given subspace $\mathcal{U}$ of $\mathbf{C}^n$. We then discuss a particular choice for the subspace $\mathcal{U}$ as a Krylov subspace, which is very useful in practice.

Assume that $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p\}$ is an orthonormal basis of $\mathcal{U}$. Then for any vector $\boldsymbol{v} \in \mathbf{C}^n$, the vector of $\mathcal{U}$ that is closest to $\boldsymbol{v}$ in the Euclidean distance is given by the orthogonal projection

$$\mathsf{P}_{\mathcal{U}}\boldsymbol{v} := \mathsf{U}\mathsf{U}^*\boldsymbol{v} = (\boldsymbol{u}_1\boldsymbol{u}_1^* + \cdots + \boldsymbol{u}_p\boldsymbol{u}_p^*)\boldsymbol{v}.$$

In practice, the eigenvectors of A are unknown, and so it is impossible to calculate approximations using this formula. The Rayleigh–Ritz method, which we present hereafter, is an alternative and practical method for constructing approximations of the eigenvectors and eigenvalues. In general, the subspace $\mathcal{U}$ does not contain any eigenvector of A, and so the problem

$$\mathsf{A}\boldsymbol{v} = \lambda\boldsymbol{v}, \qquad \boldsymbol{v} \in \mathcal{U} \tag{6.13}$$

does not admit a solution. Let us denote by U the matrix with columns $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p$. Since any vector $\boldsymbol{v} \in \mathcal{U}$ is equal to $\mathsf{U}\boldsymbol{z}$ for some vector $\boldsymbol{z} \in \mathbf{C}^p$, equation (6.13) is equivalent to the problem

$$\mathsf{A}\mathsf{U}\boldsymbol{z} = \lambda\mathsf{U}\boldsymbol{z},$$

which is a system of n equations with $p < n$ unknowns. The Rayleigh–Ritz method is based on the idea that, in order to obtain a problem with as many unknowns as there are equations, we can multiply this equation by U^*, which leads to the problem

$$\mathsf{B}\boldsymbol{z} := (\mathsf{U}^*\mathsf{A}\mathsf{U})\boldsymbol{z} = \lambda\boldsymbol{z}. \tag{6.14}$$

This is standard eigenvalue problem for the matrix $\mathsf{U}^*\mathsf{A}\mathsf{U} \in \mathbf{C}^{p\times p}$, which is much easier to solve than the original problem if $p \ll n$. Equivalently, equation (6.14) may be formulated as follows: find $\boldsymbol{v} \in \mathcal{U}$ such that

$$\boldsymbol{u}^*(\mathsf{A}\boldsymbol{v} - \lambda\boldsymbol{v}), \qquad \forall \boldsymbol{u} \in \mathcal{U}. \tag{6.15}$$

The solutions to (6.14) and (6.15) are related by the equation $\boldsymbol{v} = \mathsf{U}\boldsymbol{z}$. Of course, the eigenvalues of B in problem (6.14), which are called the Ritz values of A relative to $\mathcal{U}$, are in general different from those of A. Once an eigenvector $\boldsymbol{y}$ of B has been calculated, an approximate eigenvector of A, called a *Ritz vector* of A relative to $\mathcal{U}$, is obtained from the equation $\hat{\boldsymbol{v}} = \mathsf{U}\boldsymbol{y}$. The Rayleigh–Ritz algorithm is presented in full in Algorithm 13.

Algorithm 13 Rayleigh–Ritz

1: Choose $\mathcal{U} \subset \mathbf{C}^n$
2: Construct a matrix U whose columns are orthonormal and span $\mathcal{U}$
3: Find the eigenvalues $\hat{\lambda}_i$ and eigenvectors $\boldsymbol{y}_i \in \mathbf{C}^p$ of $\mathsf{B} \coloneqq \mathsf{U}^*\mathsf{A}\mathsf{U}$
4: Calculate the corresponding Ritz vectors $\hat{\boldsymbol{v}}_i = \mathsf{U}\boldsymbol{y}_i \in \mathbf{C}^n$.

It is clear that if $\boldsymbol{v}_i \in \mathcal{U}$, then λ_i is an eigenvalue of B in (6.14). In fact, we can show the following more general statement.

Proposition 6.6. *If $\mathcal{U}$ is an invariant subspace of A, meaning that $\mathsf{A}\mathcal{U} \subset \mathcal{U}$, then each Ritz vector of A relative to $\mathcal{U}$ is an eigenvector of A.*

Proof. Let $\mathsf{U} \in \mathbf{C}^{n\times p}$ and $\mathsf{W} \in \mathbf{C}^{n\times(n-p)}$ be matrices whose columns form orthonormal bases of $\mathcal{U}$ and $\mathcal{U}^\perp$, respectively. Here $\mathcal{U}^\perp$ denotes the orthogonal complement of $\mathcal{U}$ with respect to the Euclidean inner product. Then, since $\mathsf{W}^*\mathsf{A}\mathsf{U} = \mathsf{0}$ by assumption, it holds that

$$\mathsf{Q}^*\mathsf{A}\mathsf{Q} = \begin{pmatrix} \mathsf{U}^*\mathsf{A}\mathsf{U} & \mathsf{U}^*\mathsf{A}\mathsf{W} \\ \mathsf{W}^*\mathsf{A}\mathsf{U} & \mathsf{W}^*\mathsf{A}\mathsf{W} \end{pmatrix} = \begin{pmatrix} \mathsf{U}^*\mathsf{A}\mathsf{U} & \mathsf{U}^*\mathsf{A}\mathsf{W} \\ \mathsf{0} & \mathsf{W}^*\mathsf{A}\mathsf{W} \end{pmatrix}, \qquad \mathsf{Q} = \begin{pmatrix} \mathsf{U} & \mathsf{W} \end{pmatrix}.$$

If $(\boldsymbol{y}, \hat{\lambda})$ is an eigenvector of $\mathsf{U}^*\mathsf{A}\mathsf{U}$, then

$$\mathsf{Q}^*\mathsf{A}\mathsf{Q}\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} (\mathsf{U}^*\mathsf{A}\mathsf{U})\boldsymbol{y} \\ \boldsymbol{0} \end{pmatrix} = \hat{\lambda}\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{pmatrix} =: \hat{\lambda}\boldsymbol{x},$$

and so $(\boldsymbol{x}, \hat{\lambda})$ is an eigenpair of $\mathsf{Q}^*\mathsf{A}\mathsf{Q}$. But then $(\mathsf{Q}\boldsymbol{x}, \hat{\lambda}) = (\mathsf{U}\boldsymbol{y}, \hat{\lambda})$ is an eigenpair of A, which proves the statement. □

Remark 6.1. Proposition 6.6 can be proved more directly from (6.15) by noting that, if $\mathcal{U}$ is an invariant subspace of A, then this equation implies that $\mathsf{A}\boldsymbol{v} - \lambda\boldsymbol{v}$ belongs to both $\mathcal{U}$ and its orthogonal complement $\mathcal{U}^\perp$, and so this vector is $\mathbf{0}$.

If $\mathcal{U}$ is close to being an invariant subspace of A, then it is expected that the Ritz vectors and Ritz values of A relative to $\mathcal{U}$ will provide good approximations of some of the eigenpairs of A. Quantifying this approximation is difficult, so we only present without proof the following error bound. See [11] for more information.

Proposition 6.7. *Let A be a full rank Hermitian matrix and $\mathcal{U}$ a p-dimensional subspace of $\mathbf{C}^n$. Then there exists eigenvalues $\lambda_{i_1}, \ldots, \lambda_{i_p}$ of A which satisfy*

$$\forall j \in \{1, \ldots, p\}, \qquad |\lambda_{i_j} - \hat{\lambda}_j| \leqslant \|(\mathsf{I} - \mathsf{P}_{\mathcal{U}})\mathsf{A}\mathsf{P}_{\mathcal{U}}\|_2.$$

In the case where A is Hermitian, it is possible to show that the Ritz values are bounded from above by the eigenvalues of A. The proof of this result relies on the Courant–Fisher theorem for characterizing the eigenvalues of a Hermitian matrix, which is recalled in Theorem A.6 in the appendix.

Proposition 6.8. *If* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *is Hermitian, then*

$$\forall i \in \{1, \dots, p\}, \qquad \hat{\lambda}_i \leqslant \lambda_i$$

Proof. By the Courant–Fisher theorem, it holds that

$$\hat{\lambda}_i = \max_{\mathcal{S}\subset\mathbf{C}^p,\ \dim(\mathcal{S})=i} \left(\min_{\boldsymbol{x}\in\mathcal{S}\setminus\{0\}} \frac{\boldsymbol{x}^*\mathsf{B}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} \right)$$

Letting $\boldsymbol{y} = \mathsf{U}\boldsymbol{x}$ and then $\mathcal{R} = \mathsf{U}\mathcal{S}$, we deduce that

$$\begin{aligned}\hat{\lambda}_i &= \max_{\mathcal{S}\subset\mathbf{C}^p,\ \dim(\mathcal{S})=i} \left(\min_{\boldsymbol{y}\in\mathsf{U}\mathcal{S}\setminus\{0\}} \frac{\boldsymbol{y}^*\mathsf{A}\boldsymbol{y}}{\boldsymbol{y}^*\boldsymbol{y}} \right) \\ &= \max_{\mathcal{R}\subset\mathcal{U},\ \dim(\mathcal{R})=i} \left(\min_{\boldsymbol{y}\in\mathcal{R}\setminus\{0\}} \frac{\boldsymbol{y}^*\mathsf{A}\boldsymbol{y}}{\boldsymbol{y}^*\boldsymbol{y}} \right) \leqslant \max_{\mathcal{R}\subset\mathbf{C}^n,\ \dim(\mathcal{R})=i} \left(\min_{\boldsymbol{y}\in\mathcal{R}\setminus\{0\}} \frac{\boldsymbol{y}^*\mathsf{A}\boldsymbol{y}}{\boldsymbol{y}^*\boldsymbol{y}} \right) = \lambda_i,\end{aligned}$$

where we used the Courant–Fisher theorem for the matrix A in the last equality. □

This projection approach is sometimes combined with a simultaneous subspace iteration: an approximation X_k of the p first eigenvector is first calculated using Algorithm 11, and then the matrix X_k is used in place of U in Algorithm 13.

6.4.1 Projection method in a Krylov subspace

The power iteration constructs at iteration k an approximation of $\boldsymbol{v}_1$ in the one-dimensional subspace spanned by the vector $\mathsf{A}^k\boldsymbol{x}_0$, and only the previous iteration $\boldsymbol{x}_k$ is employed to construct $\boldsymbol{x}^{k+1}$. One may wonder whether, by employing all the previous iterates rather than only the previous one, a better approximation of $\boldsymbol{v}_1$ can be constructed. More precisely, instead of looking for an approximation in the subspace $\mathrm{Span}\{\mathsf{A}^k\boldsymbol{x}_0\}$, would it be useful to extend the search area to the Krylov subspace

$$\mathcal{K}_{k+1}(\mathsf{A}, \boldsymbol{x}_0) := \mathrm{Span}\Big\{\boldsymbol{x}_0, \mathsf{A}\boldsymbol{x}_0, \dots, \mathsf{A}^k\boldsymbol{x}_0\Big\}?$$

The answer to this question is positive, and the resulting method is often much faster than the power iteration. This is achieved by employing the Rayleigh–Ritz projection method Algorithm 13 with the choice $\mathcal{U} = \mathcal{K}_{k+1}(\mathsf{A}, \boldsymbol{x}_0)$. Applying this method requires to calculate an orthonormal basis of the Krylov subspace and to calculate the reduced matrix $\mathsf{U}^*\mathsf{A}\mathsf{U}$. The *Arnoldi method* enables to achieve these two goals simultaneously.

6.4.2 The Arnoldi iteration

This Arnoldi iteration is based on the Gram–Schmidt process and presented in Algorithm 14.

Algorithm 14 Arnoldi iteration for constructing an orthonormal basis of $\mathcal{K}_p(\mathsf{A}, \boldsymbol{u}_1)$

Choose $\boldsymbol{u}_1$ with unit norm.
for $j \in \{1, \dots p\}$ **do**
    $\boldsymbol{u}_{j+1} \leftarrow \mathsf{A}\boldsymbol{u}_j$
    for $i \in \{1, \dots, j\}$ **do**
        $h_{i,j} \leftarrow \boldsymbol{u}_i^* \boldsymbol{u}_{j+1}$
        $\boldsymbol{u}_{j+1} \leftarrow \boldsymbol{u}_{j+1} - h_{i,j}\boldsymbol{u}_i$
    end for
    $h_{j+1,j} \leftarrow \|\boldsymbol{u}_{j+1}\|$
    $\boldsymbol{u}_{j+1} \leftarrow \boldsymbol{u}_{j+1}/h_{j+1,j}$
end for

The iteration breaks down if $h_{j+1,j} = 0$, which indicates that $\mathcal{A}\boldsymbol{u}_j$ belongs to the Krylov subspace $\mathrm{Span}\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_j\} = \mathcal{K}_j(\mathsf{A}, \boldsymbol{u}_1)$, implying that $\mathcal{K}_{j+1}(\mathsf{A}, \boldsymbol{u}_1) = \mathcal{K}_j(\mathsf{A}, \boldsymbol{u}_1)$. In this case, the subspace $\mathcal{K}_j(\mathsf{A}, \boldsymbol{u}_1)$ is an invariant subspace of A because, by Exercise 6.2, we have

$$\mathsf{A}\mathcal{K}_j(\mathsf{A}, \boldsymbol{u}_1) \subset \mathcal{K}_{j+1}(\mathsf{A}, \boldsymbol{u}_1) = \mathcal{K}_j(\mathsf{A}, \boldsymbol{u}_1).$$

Therefore, applying the Rayleigh–Ritz with $\mathcal{U} = \mathrm{Span}\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_j\}$ yields exact eigenpairs in view of Proposition 6.6. If the iteration does not break down then, by construction, the vectors $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_p\}$ at the end of the algorithm are orthonormal. It is also simple to show by induction that they form a basis of $\mathcal{K}_p(\mathsf{A}, \boldsymbol{u}_1)$. The scalar coefficients $h_{i,j}$ can be collected in a matrix square $p \times p$ matrix

$$\mathsf{H} = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,p} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,p} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,p} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{p,p-1} & h_{p,p} \end{pmatrix}.$$

This matrix contains only zeros under the first subdiagonal; such a matrix is called a *Hessenberg* matrix. Inspecting the algorithm, we notice that the j-th column contains the coefficients of the projection of the vector $\mathsf{A}\boldsymbol{u}_j$ onto the basis $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_p\}$. In other words,

$$\mathsf{U}^*\mathsf{A}\mathsf{U} = \mathsf{H}, \tag{6.16}$$

We have thus shown that the Arnoldi algorithm enables to construct both an orthonormal basis of a Krylov subspace and the associated reduced matrix. In fact, we have the following equation

$$\mathsf{A}\mathsf{U} = \mathsf{U}\mathsf{H} + h_{p+1,p}(\boldsymbol{v}_{p+1}\boldsymbol{e}_p^*), \qquad \boldsymbol{e}_p = \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix} \in \mathbf{C}^p. \tag{6.17}$$

The Arnoldi algorithm, coupled with the Rayleigh–Ritz method, has very good convergence properties in the limit as $p \to \infty$, in particular for eigenvalues with a large modulus. The

following result shows that the residual $\boldsymbol{r} = \mathsf{A}\hat{\boldsymbol{v}} - \hat{\lambda}\boldsymbol{v}$ associated with a Ritz vector can be estimated inexpensively. Specifically, the norm of the residual is equal to the last component of the associated eigenvector of H multiplied by $h_{p+1,p}$.

Proposition 6.9 (Formula for the residual). *Let $\boldsymbol{y}_i$ be an eigenvector of H associated with the eigenvalues $\hat{\lambda}_i$, and let $\hat{\boldsymbol{v}}_i = \mathsf{U}\boldsymbol{y}_i$ denote the corresponding eigenvector. Then*

$$\mathsf{A}\hat{\boldsymbol{v}}_i - \hat{\lambda}\boldsymbol{v}_i = h_{p+1,p}(\boldsymbol{y}_i)_p \boldsymbol{v}_{p+1}.$$

Consequently, it holds that

$$\|\mathsf{A}\hat{\boldsymbol{v}}_i - \hat{\lambda}\boldsymbol{v}_i\| = |h_{p+1,p}(\boldsymbol{y}_i)_p|.$$

Proof. Multiplying both sides of (6.17) by $\boldsymbol{y}_i$, we obtain

$$\mathsf{AU}\boldsymbol{y}_i = \mathsf{UH}\boldsymbol{y}_i + h_{p+1,p}(\boldsymbol{v}_{p+1}\boldsymbol{e}_p^*)\boldsymbol{y}_i.$$

Using the definition of $\hat{\boldsymbol{v}}_i$ and rearranging the equation, we have

$$\mathsf{A}\widehat{\boldsymbol{v}_i} - \hat{\lambda}_i\boldsymbol{y}_i = h_{p+1,p}(\boldsymbol{v}_{p+1}\boldsymbol{e}_p^*)\boldsymbol{y}_i,$$

which immediately gives the result. □

In practice, the larger the dimension p of the subspace $\mathcal{U}$ employed in the Rayleigh–Ritz method, the more memory is required for storing an orthonormal basis of $\mathcal{U}$. In addition, for large values of p, computing the reduced matrix (6.16) and its eigenpairs becomes computationally expensive; the computational cost of computing the matrix H scales as $\mathcal{O}(p^2)$. To remedy these potential issues, the algorithm can be restarted periodically. For example, Algorithm 15 can be employed as an alternative to the power iteration in order to find the eigenvector associated with the eigenvalue with largest modulus.

Algorithm 15 Restarted Arnoldi iteration

1: Choose $\boldsymbol{u}_1 \in \mathbf{C}^n$ and $p \ll n$
2: **for** $i \in \{1, 2, \dots\}$ **do**
3: Perform p iterations of the Arnoldi iteration and construct $\mathcal{U}$;
4: Calculate the Ritz vector $\hat{\boldsymbol{v}}_1$ associated with the largest Ritz value relative to $\mathcal{U}$;
5: If this vector is sufficiently accurate, then stop. Otherwise, restart with $\boldsymbol{u}_1 = \hat{\boldsymbol{v}}_1$.
6: **end for**

6.4.3 The Lanczos iteration

The Lanczos iteration may be viewed as a simplified version of the Arnoldi iteration in the case where the matrix A is Hermitian. Let us denote by $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_p\}$ the orthonormal vectors generated by the Arnoldi iteration. When A is Hermitian, it holds that

$$h_{i,j} = \boldsymbol{u}_i^*(\mathsf{A}\boldsymbol{u}_j) = (\mathsf{A}\boldsymbol{u}_i)^*\boldsymbol{u}_j = \overline{h_{j,i}}.$$

Therefore, the matrix H is Hermitian. This is not surprising, since we showed that $\mathsf{H} = \mathsf{U}^*\mathsf{A}\mathsf{U}$ and the matrix A is Hermitian. Since H is also of Hessenberg form, we deduce that H is tridiagonal. An inspection of Algorithm 14 shows that the subdiagonal entries of H are real. Since A is Hermitian, the diagonal entries $h_{i,i} = \boldsymbol{u}_i^*(\mathsf{A}\boldsymbol{u}_j)$ are also real, and so we conclude that all the entries of the matrix H are in fact real. This matrix if of the form

$$\mathsf{H} = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_p \\ & & & \beta_p & \alpha_p \end{pmatrix}$$

Adapting the Arnoldi iteration to this setting leads to Algorithm 16.

Algorithm 16 Lanczos iteration for constructing an orthonormal basis of $\mathcal{K}_p(\mathsf{A}, \boldsymbol{u}_1)$

Choose $\boldsymbol{u}_1$ with unit norm.
$\beta_1 \leftarrow 0$, $\boldsymbol{u}_0 \leftarrow \mathbf{0} \in \mathbf{C}^n$
for $j \in \{1, \dots p\}$ **do**
    $\boldsymbol{u}_{j+1} \leftarrow \mathsf{A}\boldsymbol{u}_j - \beta_j \boldsymbol{u}_{j-1}$
    $\alpha_j \leftarrow \boldsymbol{u}_j^* \boldsymbol{u}_{j+1}$
    $\boldsymbol{u}_{j+1} \leftarrow \boldsymbol{u}_{j+1} - \alpha_j \boldsymbol{u}_j$
    $\beta_{j+1} \leftarrow \|\boldsymbol{u}_{j+1}\|$
    $\boldsymbol{u}_{j+1} \leftarrow \boldsymbol{u}_{j+1}/\beta_{j+1}$
end for

6.5 Exercises

Exercise 6.1. *PageRank is an algorithm for assigning a rank to the vertices of a directed graph. It is used by many search engines, notably Google, for sorting search results. In this context, the directed graph encodes the links between pages of the World Wide Web: the vertices of the directed graph are webpages, and there is an edge going from page i to page j if page i contains a hyperlink to page j.*

Let us consider a directed graph $G(V, E)$ with vertices $V = \{1, \dots, n\}$ and edges E. The graph can be represented by its adjacency matrix $\mathsf{A} \in \{0, 1\}^{n \times n}$, whose entries are given by

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$$

Let r_i denote the "value" assigned to vertex i. The idea of PageRank, in its simplest form, is to assign values to the vertices by solving the following system of equations;

$$\forall i \in V, \qquad r_i = \sum_{j \in \mathcal{N}(i)} \frac{r_j}{o_j}. \tag{6.18}$$

where o_j is the outdegree of vertex j, i.e. the number of edges leaving from j. Here the sum is over the set of nodes $\mathcal{N}(i)$, which denotes all the "incoming" neighbors of i, i.e. those that have an edge pointing towards node i.

- *Read the Wikipedia page on* PageRank *to familiarize yourself with the algorithm.*
- *Let $\boldsymbol{r} = (r_1 \quad \dots \quad r_n)^T$. Show using (6.18) that $\boldsymbol{r}$ satisfies*

$$\boldsymbol{r} = \mathsf{A}^T \begin{pmatrix} \frac{1}{o_1} & & \\ & \ddots & \\ & & \frac{1}{o_n} \end{pmatrix} \boldsymbol{r} =: \mathsf{A}^T \mathsf{O}^{-1} \boldsymbol{r}.$$

  In other words, $\boldsymbol{r}$ is an eigenvector with eigenvalue 1 of the matrix $\mathsf{M} = \mathsf{A}^T\mathsf{O}^{-1}$.

- *Show that M is a left-stochastic matrix, i.e. that each column sums to 1.*
- *Prove that the eigenvalues of any matrix $\mathsf{B} \in \mathbf{R}^{n\times n}$ coincide with those of B^T. You may use the fact that $\det(\mathsf{B}) = \det(\mathsf{B}^T)$.*
- *Using the previous items, show that 1 is an eigenvalue and that $\rho(\mathsf{M}) = 1$. For the second part, find a subordinate matrix norm such that $\|\mathsf{M}\| = 1$.*
- *Implement PageRank in order to rank pages from a 2013 snapshot of English Wikipedia. You can use either the simplified version of the algorithm given in (6.18) or the improved version with a damping factor described on Wikipedia. In the former case, the following are both sensible stopping criteria:*

$$\frac{\|\mathsf{M}\hat{\boldsymbol{r}} - \hat{\boldsymbol{r}}\|_1}{\|\hat{\boldsymbol{r}}\|_1} < 10^{-15} \qquad \text{or} \qquad \frac{\|\mathsf{M}\hat{\boldsymbol{r}} - \hat{\lambda}\hat{\boldsymbol{r}}\|_1}{\|\boldsymbol{r}\|_1} < 10^{-15}, \qquad \hat{\lambda} = \frac{\hat{\boldsymbol{r}}^T\mathsf{M}\hat{\boldsymbol{r}}}{\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}}},$$

  where $\hat{\boldsymbol{v}}$ is an approximation of the eigenvector corresponding to the dominant eigenvalue. A dataset is available on the course website to complete this part. This dataset contains a subset of the data publicly available here, and was generated from the full dataset by retaining only the 5% best rated articles. After decompressing the archive, you can load the dataset into Julia by using the following commands:

```
import CSV
import DataFrames

# Data (nodes and edges)
nodes = CSV.read("names.csv", DataFrames.DataFrame)
edges = CSV.read("edges.csv", DataFrames.DataFrame)

# Convert data to matrices
nodes = Matrix(nodes)
edges = Matrix(edges)
```

After you have assigned a rank to all the pages, print the 10 pages with the highest ranks. My code returns the following entries:

1. *United States*
2. *United Kingdom*
3. *World War II*
4. *Latin*
5. *France*
6. *Germany*
7. *English language*
8. *China*
9. *Canada*
10. *India*

- ***Extra credit:*** *Write a function* `search(keyword)` *that can be employed for searching the database. Here is an example of what it could return:*

```
julia> search("New York")
481-element Vector{String}:
 "New York City"
 "New York"
 "The New York Times"
 "New York Stock Exchange"
 "New York University"
 …
```

Exercise 6.2. *Show the following properties of the Krylov subspace* $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x})$.

- $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x}) \subset \mathcal{K}_{p+1}(\mathsf{A}, \boldsymbol{x})$.
- $\mathsf{A}\mathcal{K}_p(\mathsf{A}, \boldsymbol{x}) \subset \mathcal{K}_{p+1}(\mathsf{A}, \boldsymbol{x})$.
- *The Krylov subspace* $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x})$ *is invariant under rescaling: for all* $\alpha \in \mathbf{C}$,

$$\mathcal{K}_p(\mathsf{A}, \boldsymbol{x}) = \mathcal{K}_p(\alpha\mathsf{A}, \boldsymbol{x}) = \mathcal{K}_p(\mathsf{A}, \alpha\boldsymbol{x}).$$

- *The Krylov subspace* $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x})$ *is invariant under shift of the matrix* A*: for all* $\alpha \in \mathbf{C}$,

$$\mathcal{K}_p(\mathsf{A}, \boldsymbol{x}) = \mathcal{K}_p(\mathsf{A} - \alpha\mathsf{I}, \boldsymbol{x}).$$

- *Similarity transformation: If* $\mathsf{T} \in \mathbf{C}^{n\times n}$ *is nonsingular, then*

$$\mathcal{K}_p(\mathsf{T}^{-1}\mathsf{A}\mathsf{T}, \mathsf{T}^{-1}\boldsymbol{x}) = \mathsf{T}^{-1}\mathcal{K}_p(\mathsf{A}, \boldsymbol{x}).$$

Exercise 6.3. *The minimal polynomial of a matrix* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *is the monic polynomial* p *of lowest degree such that* $p(\mathsf{A}) = 0$. *Prove that, if* A *is Hermitian with* $m \leqslant n$ *distinct eigenvalues, then the minimal polynomial is given by*

$$p(t) = \prod_{i=1}^{m}(t - \lambda_i).$$

Exercise 6.4. *The minimal polynomial for a general matrix* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *is given by*

$$p(t) = \prod_{i=1}^{m} (t - \lambda_i)^{s_i}.$$

where s_i is the size of the largest Jordan block associated with the eigenvalue λ_i in the normal Jordan form of A. Verify that $p(\mathsf{A}) = 0$.

Exercise 6.5. *Let d denote the degree of the minimal polynomial of A. Show that*

$$\forall p \geqslant d, \qquad \mathcal{K}_{p+1}(\mathsf{A}, \boldsymbol{x}) = \mathcal{K}_p(\mathsf{A}, \boldsymbol{x}).$$

Deduce that, for $p \geqslant n$, the subspace $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x})$ is an invariant subspace of A.

Exercise 6.6. *Let $\mathsf{A} \in \mathbf{C}^{n\times n}$. Show that $\mathcal{K}_n(\mathsf{A}, \boldsymbol{x})$ is the smallest invariant subspace of A that contains $\boldsymbol{x}$.*

Exercise 6.7. *Consider the matrix*

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

- *Find the dominant eigenvalue of M by using the power iteration.*
- *Find the eigenvalue of M closest to 1 by using the inverse iteration.*
- *Find the other two eigenvalues of M by using a method of your choice.*

Exercise 6.8 (A posteriori error bound)**.** *Assume that $\mathsf{A} \in \mathbf{C}^{n\times n}$ is Hermitian, and that $\hat{\boldsymbol{v}}$ is a normalized approximation of an eigenvector which satisfies*

$$\|\hat{\boldsymbol{z}}\| := \|\mathsf{A}\hat{\boldsymbol{v}} - \hat{\lambda}\hat{\boldsymbol{v}}\| = \delta, \qquad \hat{\lambda} = \frac{\hat{\boldsymbol{v}}^*\mathsf{A}\hat{\boldsymbol{v}}}{\hat{\boldsymbol{v}}^*\hat{\boldsymbol{v}}}.$$

Prove that there is an eigenvalue λ of A such that

$$|\hat{\lambda} - \lambda| \leqslant \delta.$$

Hint: *Consider first the case where A is diagonal.*

Exercise 6.9 (Bauer–Fike theorem)**.** *Assume that $\mathsf{A} \in \mathbf{C}^{n\times n}$ is diagonalizable: $\mathsf{AV} = \mathsf{VD}$. Show that, if $\hat{\boldsymbol{v}}$ is a normalized approximation of an eigenvector which satisfies*

$$\|\boldsymbol{r}\| := \|\mathsf{A}\hat{\boldsymbol{v}} - \hat{\lambda}\hat{\boldsymbol{v}}\| = \delta$$

for some $\hat{\lambda} \in \mathbf{C}$, then there is an eigenvalue λ of A such that

$$|\hat{\lambda} - \lambda| \leqslant \kappa_2(\mathsf{V})\delta.$$

Hint*: Rewrite*

$$\|\hat{\boldsymbol{v}}\| = \|(\mathsf{A} - \hat{\lambda}\mathsf{I})^{-1}\boldsymbol{r}\| = \|\mathsf{V}(\mathsf{D} - \hat{\lambda}\mathsf{I})^{-1}\mathsf{V}^{-1}\boldsymbol{r}\|).$$

Exercise 6.10. *In Exercise 6.8 and Exercise 6.9, we saw examples* a posteriori *error estimates which guarantee the existence of an eigenvalue of* A *within a certain distance of the approximation* $\hat{\lambda}$*. In this exercise, our aim is to provide an answer to the following different question: given an approximate eigenpair* $(\hat{\boldsymbol{v}}, \hat{\lambda})$*, what is the smallest perturbation* E *that we need to apply to* A *in order to guarantee that* $(\hat{\boldsymbol{v}}, \hat{\lambda})$ *is an exact eigenpair, i.e. that*

$$(\mathsf{A} + \mathsf{E})\hat{\boldsymbol{v}} = \hat{\lambda}\hat{\boldsymbol{v}}?$$

Assume that $\hat{\boldsymbol{v}}$ *is normalized and let* $\mathcal{E} = \left\{\mathsf{E} \in \mathbf{C}^{n\times n} : (\mathsf{A} + \mathsf{E})\hat{\boldsymbol{v}} = \hat{\lambda}\hat{\boldsymbol{v}}\right\}$*. Prove that*

$$\min_{\mathsf{E}\in\mathcal{E}} \|\mathsf{E}\|_2 = \|\boldsymbol{r}\|_2 := \|\mathsf{A}\hat{\boldsymbol{v}} - \hat{\lambda}\hat{\boldsymbol{v}}\|. \tag{6.19}$$

To this end, you may proceed as follows:

- *Show first that any* $\mathsf{E} \in \mathcal{E}$ *satisfies* $\mathsf{E}\hat{\boldsymbol{v}} = -\boldsymbol{r}$*.*
- *Deduce from the first item that*

$$\inf_{\mathsf{E}\in\mathcal{E}} \|\mathsf{E}\|_2 \geqslant \|\boldsymbol{r}\|_2.$$

- *Find a rank one matrix* E_* *such that* $\|\mathsf{E}_*\|_2 = \|\boldsymbol{r}\|_2$*, and then conclude. Recall that any rank 1 matrix can be written in the form* $\mathsf{E}_* = \boldsymbol{u}\boldsymbol{w}^*$*, with norm* $\|\boldsymbol{u}\|_2\|\boldsymbol{w}\|_2$*.*

Equation (6.19) is a simplified version of the Kahan–Parlett–Jiang theorem and is an example of a backward error estimate*. Whereas forward error estimates quantify the distance between an approximation and the exact solution, backward error estimates give the size of the perturbation that must be applied to a problem so that an approximation is exact.*

Exercise 6.11. *Assume that* $(\boldsymbol{x}_k)_{k\geqslant 0}$ *is a sequence of normalized vectors in* $\mathbf{C}^n$*. Show that the following statements are equivalent:*

- $(\boldsymbol{x}_k)_{k\geqslant 0}$ *converges essentially to* $\boldsymbol{x}_\infty$ *in the limit as* $k \to \infty$*.*
- *The angle* $\angle(\boldsymbol{x}_k, \boldsymbol{x}_\infty)$ *converges to zero in the limit as* $k \to \infty$*.*
- *The projector* $\mathsf{P}_{\boldsymbol{x}_k}$ *converges to* $\mathsf{P}_{\boldsymbol{x}_\infty}$ *in the limit as* $k \to \infty$*.*

Exercise 6.12. *Assume that* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *is skew-Hermitian. Derive a Lanczos-like algorithm for constructing an orthonormal basis of* $\mathcal{K}_p(\mathsf{A}, \boldsymbol{x})$ *and calculating the reduced matrix*

$$\mathsf{U}^*\mathsf{A}\mathsf{U},$$

where $\mathsf{U} \in \mathbf{C}^{n\times p}$ *contains the vectors of the basis as columns. Implement your algorithm with* $p = 20$ *in order to approximate the dominant eigenvalue of the matrix* A *constructed by the following piece of code:*

```
n = 5000
A = rand(n, n) + im * randn(n, n)
A = A - A'  # A is now skew-Hermitian
```

Exercise 6.13. *Assume that* $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p\}$ *and* $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$ *are orthonormal bases of the same subspace* $\mathcal{U} \subset \mathbf{C}^n$*. Show that the projectors* $\mathsf{U}\mathsf{U}^*$ *and* $\mathsf{W}\mathsf{W}^*$ *are equal.*

Exercise 6.14. *Assume that* $\mathsf{A} \in \mathbf{C}^{n\times n}$ *is a Hermitian matrix with distinct eigenvalues, and suppose that we know the dominant eigenpair* $(\boldsymbol{v}_1, \lambda_1)$*, with* $\boldsymbol{v}_1$ *normalized. Let*

$$\mathsf{B} = \mathsf{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^*.$$

If we apply the power iteration to this matrix, what convergence can we expect?

Exercise 6.15. *Assume that* $\widehat{\boldsymbol{v}_1}$ *and* $\widehat{\boldsymbol{v}_2}$ *are two Ritz vectors of a Hermitian matrix* A *relative to a subspace* $\mathcal{U} \subset \mathbf{C}^n$*. Show that, if the associated Ritz values are distinct, then* $\widehat{\boldsymbol{v}_1}^* \widehat{\boldsymbol{v}_2} = 0$*.*

6.6 Discussion and bibliography

The content of this chapter is inspired mainly from [15] and also from [12]. The latter volume contains a detailed coverage of the standard methods for eigenvalue problems. Some of the exercises are taken from [18], and others are inspired from [12].

Chapter 7

Numerical ordinary differential equations

7.1 Analysis of the continuous problem 174
7.2 One-step methods 179
7.2.1 Forward Euler method 179
7.2.2 Backward Euler method 180
7.2.3 Analysis of general one-step methods 182
7.2.4 Widely used one-step methods 185
7.3 Multistep methods 188
7.3.1 Adams–Bashforth methods 190
7.3.2 Adams–Moulton methods 191
7.4 Absolute stability 192
7.5 Exercises 197

Introduction

This chapter concerns the numerical solution of ordinary differential equations (ODEs) of the following form:

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big), \\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0. \end{cases} \tag{7.1}$$

Here $\boldsymbol{f}\colon \mathbf{R} \times \mathbf{R}^n \to \mathbf{R}$ and $\boldsymbol{x}_0$ is the initial condition. Equations of this type are the building blocks of a plethora mathematical models in science and engineering. They have applications in celestial dynamics, molecular simulation and fluid mechanics, to mention just a few. Ordinary differential equations also arise after discretization of time-dependent partial differential equations, which are also ubiquitous in science. More often than not, it is not possible to find an

explicit solution of (7.1), and so one has to resort to numerical simulation. The rest of chapter is organized as follows:

- In Section 7.1, we define the concepts of local and global solutions for the continuous-time problem (7.1), and we recall fundamental results concerning the existence and uniqueness of a solution.
- In Section 7.2, we analyze the so-called *one-step* numerical methods to solve (7.1). We emphasize in particular the concepts of *consistency*, *stability* and *convergence*.
- In Section 7.3, we present *multistep* methods to solve (7.1), and discuss their drawbacks and advantages compared to one-step methods.
- Finally, in Section 7.4, we introduce the concept of *absolute stability* and discuss its relevance in the context of *stiff* differential equations.

7.1 Analysis of the continuous problem

A differentiable function $\boldsymbol{x}\colon I \to \mathbf{R}^n$, where I denotes an interval of $\mathbf{R}$ containing t_0, is a solution of (7.1) if $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ and the equation (7.1) is satisfied for all $t \in I$. The solution is called global if $I = \mathbf{R}$, and local otherwise.

Integral formulation. If $\boldsymbol{x}$ is a solution to (7.1), then it holds that

$$\forall t \in I, \qquad \boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}\big(s, \boldsymbol{x}(s)\big)\,\mathrm{d}s. \tag{7.2}$$

The converse statement is not true in general, because a solution to (7.2) need not necessarily be differentiable everywhere. However, if the integral formulation (7.2) holds, then necessarily $\boldsymbol{x}$ is absolutely continuous and (7.1) is satisfied for almost every t. Additionally, if (7.2) is satisfied and the function $\boldsymbol{f}$ is continuous, then the function $s \mapsto \boldsymbol{f}\big(s, \boldsymbol{x}(s)\big)$ is continuous, and so (7.1) is satisfied for all $t \in I$ by the fundamental theorem of analysis. We now focus on the integral formulation (7.2), and begin by establishing existence of a local solution.

Theorem 7.1 (Existence of a solution)**.** *Let $\boldsymbol{x}_0 \in \mathbf{R}^n$ and let $\Omega_{\mathcal{T},\mathcal{R}}$ denote the set*

$$\Big\{(t, \boldsymbol{x}) \in \mathbf{R} \times \mathbf{R}^n : |t - t_0| \leqslant \mathcal{T} \quad \text{and} \quad \|\boldsymbol{x} - \boldsymbol{x}_0\| \leqslant \mathcal{R}\Big\},$$

Assume that the following conditions are satisfied for some $\mathcal{T} > 0$ and $\mathcal{R} > 0$:

- *The function $\boldsymbol{f}$ is uniformly bounded on $\Omega_{\mathcal{T},\mathcal{R}}$:*

$$\forall (t, \boldsymbol{x}) \in \Omega_{\mathcal{T},\mathcal{R}}, \qquad \|\boldsymbol{f}(t, \boldsymbol{x})\| \leqslant M. \tag{7.3}$$

- *The function $\boldsymbol{f}$ satisfies the following Lipschitz condition: there is $L > 0$ such that*

$$\forall \big((t, \boldsymbol{x}_1), (t, \boldsymbol{x}_2)\big) \in \Omega_{\mathcal{T},\mathcal{R}} \times \Omega_{\mathcal{T},\mathcal{R}}, \qquad \|\boldsymbol{f}(t, \boldsymbol{x}_1) - \boldsymbol{f}(t, \boldsymbol{x}_2)\| \leqslant L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|. \tag{7.4}$$

Then there exists $T \in (0, \mathcal{T}]$ depending on $\mathcal{R}$, M and L such that the differential equation (7.2) has a local solution $\boldsymbol{x}$: $[t_0 - T, t_0 + T] \to \mathbf{R}^n$.

Proof. Fix $T \in (0, \mathcal{T}]$ and let $I = [t_0 - T, t_0 + T]$. Let also $\mathcal{X}$ denote the following subset of continuous functions defined from I to $\mathbf{R}^n$:

$$\mathcal{X} := \left\{ \boldsymbol{x} \in C(I, \mathbf{R}^n) : \sup_{t \in I} \big\|\boldsymbol{x}(t) - \boldsymbol{x}_0\big\| \leqslant \mathcal{R} \right\}$$

The set $\mathcal{X}$ endowed with supremum metric is a closed subset of $C(I, \mathbf{R}^n)$. Since $\mathcal{X}$ is a closed subset of a complete metric space, it is itself complete. Let Φ: $\mathcal{X} \to C(I, \mathbf{R}^n)$ denote the mapping

$$\Phi(\boldsymbol{x})\colon t \mapsto \boldsymbol{x}_0 + \int_0^t \boldsymbol{f}\big(s, \boldsymbol{x}(s)\big)\, \mathrm{d}s.$$

The right-hand side, being the integral of a bounded function, is indeed a continuous function. We will show that, for T sufficiently small,

- the mapping Φ maps $\mathcal{X}$ into $\mathcal{X}$;
- the mapping Φ is a contraction.

From (7.3), it follows that

$$\forall \boldsymbol{x} \in \mathcal{X}, \qquad \forall t \in I, \qquad \|\Phi(\boldsymbol{x})(t) - \boldsymbol{x}_0\| = \left\| \int_{t_0}^t \boldsymbol{f}\big(s, \boldsymbol{x}(s)\big)\, \mathrm{d}s \right\| \leqslant MT.$$

On the other hand, from the Lipschitz condition (7.4), it holds that

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{X}, \qquad \|\Phi(\boldsymbol{x}) - \Phi(\boldsymbol{y})\| \leqslant LT\|\Phi(\boldsymbol{x}) - \Phi(\boldsymbol{y})\|.$$

Therefore, it suffices to take $T < \min\big\{\mathcal{T}, \frac{\mathcal{R}}{M}, \frac{1}{L}\big\}$ to ensure that the above conditions are satisfied. For a value of T in this range, the Banach fixed point theorem, Theorem A.3, gives the existence of a unique fixed point $\boldsymbol{x}_* \in \mathcal{X}$ of Φ. Since a fixed point of Φ is a solution to (7.2) in view of the definition of Φ, the statement is proved. □

It may seem at first glance that uniqueness of the solution to (7.2) follows from the uniqueness of the fixed point guaranteed by Theorem A.3. However, this theorem implies uniqueness *only in the set $\mathcal{X}$*, a property known as *conditional uniqueness*. In order to prove that the solution is

unique over the full space $C([t_0 - T, t_0 + T], \mathbf{R}^n)$, additional assumptions and arguments are required. A simple approach is to rely on Grönwall's lemma.

Lemma 7.2 (Grönwall's lemma, simplified integral form)**.** *Suppose that* $u\colon [t_0 - T, t_0 + T] \to \mathbf{R}_{\geqslant 0}$ *is continuous, nonnegative, and satisfies*

$$\forall t \in [t_0, t_0 + T], \qquad u(t) \leqslant \alpha + \int_{t_0}^{t} \beta(s)\, u(s)\, \mathrm{d}s, \tag{7.5}$$

where $\alpha \geqslant 0$ *and* $\beta\colon [t_0, t_0 + T] \to \mathbf{R}_{\geqslant 0}$ *is continuous and nonnegative. Then*

$$\forall t \in [t_0, t_0 + T], \qquad u(t) \leqslant \alpha \exp\left(\int_{t_0}^{t} \beta(s)\, \mathrm{d}s \right). \tag{7.6}$$

Proof. Assume first that $\alpha > 0$, so that the logarithm in (7.7) is well-defined. By the fundamental theorem of calculus and (7.5), it holds that

$$\frac{\mathrm{d}}{\mathrm{d}t} \left(\alpha + \int_{t_0}^{t} \beta(s)\, u(s)\, \mathrm{d}s \right) \leqslant \beta(t) \left(\alpha + \int_{t_0}^{t} \beta(s)\, u(s)\, \mathrm{d}s \right)$$

Therefore we have

$$\frac{\mathrm{d}}{\mathrm{d}t} \log \left(\alpha + \int_{t_0}^{t} \beta(s)\, u(s)\, \mathrm{d}s \right) \leqslant \beta(t), \tag{7.7}$$

and after integrating and exponentiating, we obtain

$$\alpha + \int_{t_0}^{t} \beta(s)\, u(s)\, \mathrm{d}s \leqslant \alpha \exp\left(\int_{t_0}^{t} \beta(s)\, \mathrm{d}s \right)$$

The statement then follows by using (7.5) again. Assume next that $\alpha = 0$. If (7.5) is satisfied for $\alpha = 0$, then this condition is also satisfied for all $\alpha > 0$. Therefore the conclusion (7.6) holds for all $\alpha > 0$, and taking the limit $\alpha \to 0$ in this equation, we obtain the statement. □

Note that the estimate (7.6) is sharp, since the function $v\colon [t_0, t_0 + T]$ given by

$$v(t) = \alpha \exp\left(\int_{t_0}^{t} \beta(s)\, \mathrm{d}s \right)$$

satisfies (7.5) with equality. We are now ready to prove uniqueness under an appropriate condition.

Theorem 7.3 (Uniqueness of the solution)**.** *Let* $\boldsymbol{x}_0 \in \mathbf{R}^n$ *and let*

$$\Omega_{\mathcal{T},\mathcal{R}} \Big\{ (t, \boldsymbol{x}) \in \mathbf{R} \times \mathbf{R}^n : |t - t_0| \leqslant \mathcal{T} \ \text{ and } \ \|\boldsymbol{x} - \boldsymbol{x}_0\| \leqslant \mathcal{R} \Big\},$$

Assume that for all $\mathcal{T} \in \mathbf{R}_{>0}$ and $\mathcal{R} \in \mathbf{R}_{>0}$, there is $L_{\mathcal{T},\mathcal{R}}$ such that

$$\forall\Big((t,\boldsymbol{x}_1),(t,\boldsymbol{x}_2)\Big) \in \Omega_{\mathcal{T},\mathcal{R}} \times \Omega_{\mathcal{T},\mathcal{R}}, \qquad \|\boldsymbol{f}(t,\boldsymbol{x}_1) - \boldsymbol{f}(t,\boldsymbol{x}_2)\| \leqslant L_{\mathcal{T},\mathcal{R}}\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|. \tag{7.8}$$

Then if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in $C\Big([t_0 - T, t_0 + T], \mathbf{R}^n\Big)$ are local solutions to (7.2), it holds that $\boldsymbol{x}_1 = \boldsymbol{x}_2$.

Proof. Suppose that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are solutions to (7.2). Let $I = [t_0 - T, t_0 + T]$ and

$$R := \max\left\{\sup_{t\in I}\|\boldsymbol{x}_1(t) - \boldsymbol{x}_0\|, \sup_{t\in I}\|\boldsymbol{x}_2(t) - \boldsymbol{x}_0\|\right\} < \infty,$$

Since $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are solutions, it holds that

$$\forall t \in [t_0 - T, t_0 + T], \qquad \boldsymbol{x}_1(t) - \boldsymbol{x}_2(t) = \int_{t_0}^{t} \Big(f\big(s, \boldsymbol{x}_1(s)\big) - f\big(s, \boldsymbol{x}_2(s)\big)\Big)\,\mathrm{d}s.$$

Taking the norm and using (7.8), we obtain

$$\forall t \in [t_0, t_0 + T], \qquad \|\boldsymbol{x}_1(t) - \boldsymbol{x}_2(t)\| \leqslant L_{T,R}\int_{t_0}^{t} \|\boldsymbol{x}_1(s) - \boldsymbol{x}_2(s)\|\,\mathrm{d}s$$

Using Grönwall's lemma, we deduce that $\boldsymbol{x}_1(t) = \boldsymbol{x}_2(t)$ for all $t \in [t_0, t_0 + T]$. A similar argument can be employed to show that $\boldsymbol{x}_1 = \boldsymbol{x}_2$ on $[t_0 - T, t_0]$. □

Corollary 7.4 (Maximal solutions)**.** *Assume that $\boldsymbol{f}$ is continuous in t and satisfies the local Lipschitz condition (7.8). Then there exists $-\infty \leqslant T_- < T_+ \leqslant \infty$ such that $t_0 \in (T_-, T_+)$ and the following properties are satisfied.*

- *there exists a solution $\boldsymbol{x}_*\colon (T_-, T_+) \to \mathbf{R}^n$ to (7.2);*
- *if $\boldsymbol{x}\colon I \to \mathbf{R}^n$ is a local solution of (7.2), then $I \subset (T_-, T_+)$ and $\boldsymbol{x}(t) = \boldsymbol{x}_*(t)$ for all $t \in I$.*
- *If T_+ is finite, then $\lim_{t\to T_+}\big\|\boldsymbol{x}(t)\big\| = \infty$, and if T_- is finite, then $\lim_{t\to T_-}\big\|\boldsymbol{x}(t)\big\| = \infty$.*

The solution $\boldsymbol{x}_*$ is called the maximal solution of (7.2).

Proof. Let $\mathcal{J}$ denote the union of all the open intervals I such that there exists a solution in $C(I, \mathbf{R}^n)$ to (7.2). The open set $\mathcal{J}$ is connected and, by Theorem 7.1, it contains a neighborhood of t_0. Therefore $\mathcal{J}$ is of the form (T_-, T_+), where $-\infty \leqslant T_- < t_0 < T_+ \leqslant \infty$. In view of Theorem 7.3, all the local solutions coincide where they are defined, and so they can be patched together in order to construct a solution $\boldsymbol{x}_*\colon (T_-, T_+) \to \mathbf{R}$. It remains to prove the third item. To this end, suppose for contradiction that T_+ was finite and that there was $(t_n)_{n\in\mathbf{N}}$ such that $t_n \to T_+$ in the limit $n \to \infty$ and

$$K := \sup_{n\in\mathbf{N}}\|\boldsymbol{x}_*(t_n)\| < \infty.$$

Since $\boldsymbol{f}$ is continuous, there is M such that $|f(t,\boldsymbol{x})|$ is uniformly bounded from above by M for all $(t,\boldsymbol{x}) \in [T_- - 1, T_+ + 1] \times B_{K+1}(\mathbf{0})$. Furthermore, by the assumption (7.8), there is L such that for all $t \in [T_- - 1, T_+ + 1]$, the following Lipschitz condition holds:

$$\forall (\boldsymbol{x}_1, \boldsymbol{x}_2) \in B_{K+1}(\mathbf{0}) \times B_{K+1}(\mathbf{0}), \qquad \|\boldsymbol{f}(t,\boldsymbol{x}_1) - \boldsymbol{f}(t,\boldsymbol{x}_2)\| \leqslant L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|.$$

Consequently, Theorem 7.1 with $\mathcal{T} = \mathcal{R} = 1$ implies for all n the existence of a solution to

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big), \\ \boldsymbol{x}(t_n) = \boldsymbol{x}_*(t_n). \end{cases}$$

over the time interval $[t_n - T, t_n + T]$, where $T > 0$ depends only on M and L, and not on n. But then, for n sufficiently large, this solution extends beyond T_+, which contradicts the maximality of $\mathcal{J}$. An analogous reasoning can be employed for T_-. □

Example 7.1. Consider the ODE

$$\begin{cases} x'(t) = x(t)^2, \\ x(0) = 1. \end{cases}$$

The maximal solution is $x_*\colon (-\infty, 1) \to \mathbf{R}$ given by

$$\boldsymbol{x}_*(t) = \frac{1}{1-t}.$$

Existence of a unique global solution. In certain settings, it is possible to prove the maximal solution to (7.2) is globally defined for any initial condition. We discuss a few important examples.

- The first case is when $\boldsymbol{f}\colon \mathbf{R} \times \mathbf{R}^n$ is globally Lipschitz in its second argument, with a Lipschitz constant that depends continuously on the first argument.
- The second case, generalizing the first, is when the growth of $\boldsymbol{f}(t, \bullet)$ is at most affine:

  $$\forall (t,\boldsymbol{x}) \in \mathbf{R} \times \mathbf{R}^n, \qquad \|\boldsymbol{f}(t,\boldsymbol{x})\| \leqslant C(t) + L(t)\|\boldsymbol{x}\|,$$

  with continuous constants $C(t)$ and $L(t)$.
- The third case is when $\boldsymbol{f}$ is independent of t and there is a function $W \in C^1(\mathbf{R}^n)$ such that $W(\boldsymbol{x}) \to \infty$ in the limit as $\|\boldsymbol{x}\| \to \infty$ and

  $$\forall \boldsymbol{x} \in \mathbf{R}^n, \qquad \nabla W(\boldsymbol{x}) \cdot \boldsymbol{f}(\boldsymbol{x}) \leqslant c < \infty$$

  Such a function is called a *Lyapunov function*.

The strategy of proof for global existence usually relies on an argument by contradiction. Consider for example the third setting. Since the assumptions of Corollary 7.4 are satisfied, there exists a maximal solution $\boldsymbol{x}_*\colon (T_-, T_+) \to \infty$. Assume for contradiction that T_+ is finite.

Then the third item in Corollary 7.4 implies that $\lim_{t\to T_+}\|\boldsymbol{x}_*(t)\| \to \infty$, and so $W\big(\boldsymbol{x}_*(t)\big)$ blows up as t approaches T^+. On the other hand, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}W\big(\boldsymbol{x}_*(t)\big) = \nabla W\big(\boldsymbol{x}_*(t)\big)\cdot \boldsymbol{f}\big(\boldsymbol{x}_*(t)\big) \leqslant c.$$

Therefore $\lim_{t\to T_+} W\big(\boldsymbol{x}_*(t)\big) \leqslant W\big(\boldsymbol{x}_*(t_0)\big) + |c|\ (T_+ - t_0)$, which is a contradiction.

7.2 One-step methods

From now on, we assume for simplicity that $t_0 = 0$ and that the initial value problem (7.1) admits a unique solution over the interval $[0, T]$. Most numerical methods for ODEs construct an approximation of the solution at discrete points:

$$\boldsymbol{x}_n \approx \boldsymbol{x}(t_n), \qquad n = 0, 1, 2, \ldots.$$

The discretization points $(t_n)_{n\in\mathbf{N}}$ are commonly equidistant, i.e. $t_n = n\Delta$ where Δ is the *discretization step*. Sometimes, it is useful to employ a variable time step, but we assume throughout this section that the time step is fixed, for simplicity. We begin in Section 7.2.1 and Section 7.2.2 by studying the simplest one-step methods, namely the forward and backward Euler methods. Then, in Section 7.2.3, we present a general approach to the analysis of one-step methods. Finally, in Section 7.2.4, we present other widely used one-step methods in applications.

7.2.1 Forward Euler method

Assume that (7.1) has a unique solution $\boldsymbol{x}(t)$ over the interval $[0, T]$. If $\boldsymbol{x}(t)$ is twice continuously differentiable, then by Taylor's formula, we have

$$\boldsymbol{x}(t+\Delta) = \boldsymbol{x}(t) + \Delta \boldsymbol{f}(t, \boldsymbol{x}) + \frac{\Delta^2}{2}\boldsymbol{x}''(\tau), \qquad \tau \in (t, t+\Delta). \tag{7.9}$$

This motivates a method known as the *forward* or *explicit* Euler method:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \boldsymbol{f}(t_n, \boldsymbol{x}_n),$$

with the same initial condition as for the continuous equation (7.1). The convergence of this method can be proved under a global Lipschitz assumption on the function $\boldsymbol{f}$.

Theorem 7.5 (Convergence of the forward Euler method)**.** *Assume that there is $L \in \mathbf{R}_{>0}$ such that*

$$\forall (t, \boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}\times\mathbf{R}^n\times\mathbf{R}^n, \qquad \|\boldsymbol{f}(t,\boldsymbol{x}) - \boldsymbol{f}(t,\boldsymbol{y})\| \leqslant L\|\boldsymbol{x}-\boldsymbol{y}\|. \tag{7.10}$$

Suppose in addition that there exists a unique, twice continuously differentiable of (7.1) over the interval $[0, T]$, and let

$$M = \sup_{t\in[0,T]} \|\boldsymbol{x}''(t)\|$$

Then the following error estimate holds:

$$\forall n \in \left\{0, 1, \ldots, \left\lfloor \frac{T}{\Delta} \right\rfloor\right\}, \qquad \|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant \frac{\Delta M}{2}\left(\frac{\mathrm{e}^{Lt_n} - 1}{L}\right). \tag{7.11}$$

Proof. By Taylor's theorem, it holds that

$$\boldsymbol{x}(t_n) = \boldsymbol{x}(t_{n-1}) + \Delta \boldsymbol{f}\big(t_{n-1}, \boldsymbol{x}(t_{n-1})\big) + \frac{\Delta^2}{2}\boldsymbol{\alpha}_n, \qquad \boldsymbol{\alpha}_n := 2\int_0^1 (1-s)\boldsymbol{x}''(t_n + \Delta s)\,\mathrm{d}s.$$

Notice that that $\|\boldsymbol{\alpha}_n\| \leqslant M$. Therefore, it holds that

$$\begin{aligned}\boldsymbol{x}(t_n) - \boldsymbol{x}_n &= \left(\boldsymbol{x}(t_{n-1}) + \Delta\, \boldsymbol{f}\big(t_{n-1}, \boldsymbol{x}(t_{n-1})\big) + \frac{\Delta^2}{2}\boldsymbol{\alpha}_n\right) - \Big(\boldsymbol{x}_{n-1} + \Delta\, \boldsymbol{f}\big(t_{n-1}, \boldsymbol{x}_{n-1}\big)\Big) \\ &= \big(\boldsymbol{x}(t_{n-1}) - \boldsymbol{x}_{n-1}\big) + \Delta\Big(\boldsymbol{f}\big(t_{n-1}, \boldsymbol{x}(t_{n-1})\big) - \boldsymbol{f}\big(t_{n-1}, \boldsymbol{x}_{n-1}\big)\Big) + \frac{\Delta^2}{2}\boldsymbol{\alpha}_n,\end{aligned}$$

Let $\boldsymbol{e}_n = \boldsymbol{x}(t_n) - \boldsymbol{x}_n$ and $\boldsymbol{\varepsilon}_n = \frac{\Delta^2}{2}\boldsymbol{\alpha}_n$. The first term is the error at iteration $n-1$, and the second may be bounded from (7.10), which gives

$$\|\boldsymbol{e}_n\| \leqslant (1 + \Delta L)\|\boldsymbol{e}_{n-1}\| + \|\boldsymbol{\varepsilon}_n\|.$$

The structure of this equation is important, as it appears in the analysis of all one-step methods for ODEs. The first term is an amplification of the error at the previous iteration, and the second term is an upper bound on the additional error introduced at step n. Applying this inequality to the previous time steps, we obtain

$$\begin{aligned}\|\boldsymbol{e}_n\| &\leqslant (1 + \Delta L)\Big((1 + \Delta L)\|\boldsymbol{e}_{n-2}\| + \|\boldsymbol{\varepsilon}_{n-1}\|\Big) + \|\boldsymbol{\varepsilon}_n\| \\ &\leqslant \ \ldots \ \leqslant (1 + \Delta L)^n\|\boldsymbol{e}_0\| + \sum_{i=1}^{n}(1 + \Delta L)^{n-i}\|\boldsymbol{\varepsilon}_i\|.\end{aligned} \tag{7.12}$$

Since $\|\boldsymbol{\varepsilon}_i\| \leqslant \Delta^2 M/2$, we have by using the formula for geometric series that

$$\|\boldsymbol{e}_n\| \leqslant (1 + \Delta L)^n\|\boldsymbol{e}_0\| + \frac{(1 + \Delta L)^n - 1}{\Delta L}\left(\frac{\Delta^2 M}{2}\right).$$

The first term is zero because $\|\boldsymbol{e}_0\| = 0$. Using the bound $(1 + \Delta L)^n \leqslant \big(\exp(\Delta L)\big)^n = \mathrm{e}^{Lt_n}$ in the second term and rearranging, we finally obtain the statement (7.11). □

7.2.2 Backward Euler method

If we apply the Taylor expansion (7.9) backward around $t + \Delta$, instead of forward around t, then we obtain

$$\boldsymbol{x}(t) = \boldsymbol{x}(t+\Delta) - \Delta \boldsymbol{f}(t+\Delta, \boldsymbol{x}) + \mathcal{O}(\Delta^2).$$

This motivates the so-called *backward* or *implicit* Euler method:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \boldsymbol{f}(t_{n+1}, \boldsymbol{x}_{n+1}). \tag{7.13}$$

Observe that the right-hand side depends on $\boldsymbol{x}_{n+1}$. Therefore, given t_n and $\boldsymbol{x}_n$, this is a nonlinear equation for the unknown $\boldsymbol{x}_{n+1}$, which can be solved by using any of the methods studied in Chapter 5. Finding a solution to (7.13) amounts to finding a fixed point of the function

$$\boldsymbol{y} \mapsto \boldsymbol{F}(\boldsymbol{y}) := \boldsymbol{x}_n + \Delta \boldsymbol{f}(t_n, \boldsymbol{y}).$$

A priori, the existence and uniqueness of such a fixed point is not guaranteed. We proved in Theorem 5.2 that a sufficient condition for these two properties to hold is that $\boldsymbol{F}$ is globally Lipschitz with a constant strictly less than 1, which holds if and only if the function $\boldsymbol{y} \mapsto \boldsymbol{f}(t_n, \boldsymbol{y})$ is globally Lipschitz with a constant strictly less than $1/\Delta$. If the condition (7.10) holds, for example, then the backward Euler method (7.13) is guaranteed to be well defined for $\Delta < \frac{1}{L}$. Theorem 5.2 also ensures that, if $\boldsymbol{F}$ is globally Lipschitz with a constant less than 1, then the fixed point can be approximated by using the iteration

$$\boldsymbol{y}_{k+1} = \boldsymbol{F}(\boldsymbol{y}_k). \tag{7.14}$$

and there is exponential convergence $\boldsymbol{y}_k \to \boldsymbol{x}_{n+1}$ in the limit as $k \to \infty$. A natural starting point for (7.14) is $\boldsymbol{y}_0 = \boldsymbol{x}_n$. An alternative approach to the fixed point iteration (7.14) is to use the Newton–Raphson method for (7.13), which is faster in principle but must be initialized sufficiently close to the fixed point.

Using a reasoning similar to that employed for proving Theorem 7.5, we can prove the following result.

Theorem 7.6 (Convergence of the backward Euler method)**.** *If the assumptions of Theorem 7.5 hold and $\Delta < \frac{1}{L}$, then the following error estimate holds:*

$$\forall n \in \left\{0, 1, \dots, \left\lfloor \frac{T}{\Delta} \right\rfloor\right\}, \qquad \|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant \frac{\Delta M}{2} \left(\frac{\left(\frac{1}{1-\Delta L}\right)^n - 1}{L} \right). \tag{7.15}$$

Proof. The proof is left as an exercise. □

Remark 7.1. Note that, if $\Delta < \frac{1}{2L}$, then

$$\begin{aligned} \frac{1}{1-\Delta L} &= 1 + \Delta L + (\Delta L)^2 + (\Delta L)^3 + (\Delta L)^4 \dots \\ &\leqslant 1 + \Delta L + (\Delta L)^2 + \frac{1}{2}(\Delta L)^2 + \frac{1}{4}(\Delta L)^2 + \dots \\ &\leqslant 1 + \Delta L + 2(\Delta L)^2 \leqslant \exp\bigl(\Delta L + (\Delta L)^2\bigr), \end{aligned}$$

and so the error estimate (7.15) gives

$$\|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant \frac{\Delta M}{2}\left(\frac{\exp(Lt_n + \Delta L^2 t_n) - 1}{L}\right),$$

which makes it clear that the right-hand side of (7.15) is close, in absolute and relative terms, to that of (7.11) when $\Delta \ll 1$.

At this point, the reader may be wondering why one would use the backward Euler method instead of the forward Euler method, given that both methods have same order of convergence but iterations of the former are more computationally costly. The reason is that the backward Euler method, like many implicit methods, is more *stable* than its forward counterpart. Implicit methods are especially attractive in the context of *stiff* differential equations. We shall elaborate on this subject in Section 7.4.

7.2.3 Analysis of general one-step methods

In general, one-step methods to solve differential equations are of the abstract form

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \boldsymbol{\Phi}_\Delta(t_n, \boldsymbol{x}_n). \tag{7.16}$$

where $\boldsymbol{\Phi}_\Delta \colon \mathbf{R} \times \mathbf{R}^n \to \mathbf{R}^n$ is a function such that

$$\boldsymbol{\Phi}_\Delta(t, \boldsymbol{x}) \approx \frac{1}{\Delta}\int_t^{t+\Delta} \boldsymbol{f}\big(s, \boldsymbol{x}^{t,\boldsymbol{x}}(s)\big)\,\mathrm{d}s = \frac{\boldsymbol{x}^{t,\boldsymbol{x}}(t+\Delta) - \boldsymbol{x}}{\Delta}. \tag{7.17}$$

Here $\boldsymbol{x}^{t,\boldsymbol{x}}$ denotes the solution to the differential equation (7.1) with initial condition $\boldsymbol{x}(s) = \boldsymbol{x}$. The main goal of this section is to establish general conditions, known as *consistency* and *stability*, under which the numerical scheme (7.16) is convergent. As we observed in the proof of Theorem 7.5 – specifically in equation (7.12) – the error at the final iteration for the forward Euler method is a sum of local errors, each amplified by a factor depending to the number of iterations left to reach the final time. *Consistency* of a numerical method enables to control the size of local errors when they arise, while *stability* enables to control their growth.

We emphasize that both the forward and the backward Euler methods can be recast in the form (7.16). For the forward Euler method $\boldsymbol{\Phi}_\Delta(t, \boldsymbol{x}) = \boldsymbol{f}(t, \boldsymbol{x})$, while for the backward Euler method, the function $\boldsymbol{\Phi}_\Delta$ is defined implicitly as the function which to $(t, \boldsymbol{x})$ associates the solution $\boldsymbol{\phi} \in \mathbf{R}^n$ to the equation

$$\boldsymbol{\phi} = \boldsymbol{f}(t + \Delta, \boldsymbol{x} + \Delta\boldsymbol{\phi}).$$

Local truncation error and consistency

The local truncation error is the residual error obtained when substituting the exact solution of the differential equation in (7.16):

$$\boldsymbol{\eta}_{n+1} := \frac{\boldsymbol{x}(t_{n+1}) - \boldsymbol{x}(t_n)}{\Delta} - \boldsymbol{\Phi}_\Delta\big(t_n, \boldsymbol{x}(t_n)\big).$$

Since there is a division by Δ, the local truncation error has the same physical dimension as that of $\boldsymbol{x}'$, and so it should be viewed as an error *per time unit*.

Definition 7.1 (Consistency)**.** A numerical method is consistent if

$$\lim_{\Delta \to 0} \left(\max_{1 \leqslant n \leqslant N} \|\boldsymbol{\eta}_n\| \right) = 0, \qquad N = \left\lfloor \frac{T}{\Delta} \right\rfloor.$$

It is consistent with order p if there exists C such that

$$\forall \Delta > 0, \qquad \max_{1 \leqslant n \leqslant N} \|\boldsymbol{\eta}_n\| \leqslant C\Delta^p.$$

Proving the consistency of a numerical method is usually achieved on a case-by-case basis by application of Taylor's formula.

Stability

The stability of a numerical method qualifies its sensitivity to perturbations. Roughly speaking, it expresses that small perturbation of the right-hand side of (7.16) lead to small perturbations of the numerical solution.

Definition 7.2 (Stability)**.** A numerical method of the form (7.16) is stable if there exists a constant $S(T) > 0$ independent of Δ such that for all sequence $(\boldsymbol{y}_n)_{1 \leqslant n \leqslant N}$ satisfying

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \Delta \boldsymbol{\Phi}_\Delta(t_n, \boldsymbol{y}_n) + \Delta \boldsymbol{\delta}_{n+1}, \qquad \boldsymbol{y}_0 = \boldsymbol{x}_0, \tag{7.18}$$

it holds that

$$\max_{1 \leqslant n \leqslant N} \|\boldsymbol{x}_n - \boldsymbol{y}_n\| \leqslant S(T)\Delta \sum_{n=1}^{N} \|\boldsymbol{\delta}_n\|. \tag{7.19}$$

It is convenient to introduce the following norms for sequences of vectors $(\boldsymbol{u}_n)_{1 \leqslant n \leqslant N}$:

$$\|\boldsymbol{u}_\bullet\|_{\ell^1_T} = \sum_{n=1}^{N} \|\boldsymbol{u}_n\|, \qquad \|\boldsymbol{u}_\bullet\|_{\ell^\infty_T} = \max_{1 \leqslant n \leqslant N} \|\boldsymbol{u}_n\|,$$

With these notations, equation (7.19) may be rewritten compactly as follows:

$$\|\boldsymbol{x}_\bullet - \boldsymbol{y}_\bullet\|_{\ell^\infty_T} \leqslant S(T)\Delta \|\boldsymbol{\delta}_\bullet\|_{\ell^1_T}$$

One could argue that this equation is neater than (7.19); it bounds a norm of just one mathematical object, namely the sequence $(\boldsymbol{x}_n - \boldsymbol{y}_n)_{1 \leqslant n \leqslant N}$, by a norm of another object, namely the sequence $(\boldsymbol{\delta}_n)_{1 \leqslant n \leqslant N}$. Arguments for proving that a numerical scheme is stable often rely on some form of Lipschitz continuity. If the function $\boldsymbol{\Phi}_\Delta(t, \boldsymbol{y})$ is globally Lipschitz continuous with respect to $\boldsymbol{y}$, then stability is particularly simple to prove, as we now demonstrate.

Proposition 7.7. *Assume that there is $L_\Phi > 0$ such that for all $t \in [0,T]$ and $\Delta > 0$, the function $\boldsymbol{\Phi}_\Delta(t, \bullet)$ is globally Lipschitz continuous with constant L_Φ. Then the one-step method (7.16) is stable.*

Proof. By (7.16) and (7.18), it holds that

$$\boldsymbol{x}_n - \boldsymbol{y}_n = \boldsymbol{x}_{n-1} - \boldsymbol{y}_{n-1} + \Delta\Big(\boldsymbol{\Phi}_\Delta(t_{n-1}, \boldsymbol{x}_{n-1}) - \boldsymbol{\Phi}_\Delta(t_{n-1}, \boldsymbol{y}_{n-1})\Big) - \Delta\boldsymbol{\delta}_n.$$

Taking the Euclidean norm and employing the Lipschitz continuity assumption, we obtain

$$\|\boldsymbol{x}_n - \boldsymbol{y}_n\| \leqslant (1 + \Delta L_\Phi)\|\boldsymbol{x}_{n-1} - \boldsymbol{y}_{n-1}\| + \Delta\|\boldsymbol{\delta}_n\|.$$

By a reasoning similar to that in the proof of Theorem 7.5, we then obtain

$$\|\boldsymbol{x}_n - \boldsymbol{y}_n\| \leqslant (1 + \Delta L_\Phi)^n\|\boldsymbol{x}_0 - \boldsymbol{y}_0\| + \sum_{i=1}^{n}(1 + \Delta L_\Phi)^{n-i}\Delta\|\boldsymbol{\delta}_i\| \leqslant 0 + \mathrm{e}^{L_\Phi t_n}\Delta\sum_{i=1}^{n}\|\boldsymbol{\delta}_i\|.$$

We conclude that (7.19) is satisfied with $S(T) = \mathrm{e}^{L_\Phi T}$. □

Convergence

We are now ready to prove that consistency and stability of the numerical (7.16) together imply convergence, in the sense that

$$\lim_{\Delta\to 0}\Big(\max_{1\leqslant n\leqslant N}\|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\|\Big) = 0, \qquad N = \left\lfloor\frac{T}{\Delta}\right\rfloor.$$

This result is an instance of the *Lax equivalence theorem*, a pillar of numerical analysis with far-reaching applications.

Theorem 7.8 (Consistence and stability imply convergence)**.** *Assume that the one-step numerical method (7.16) is consistent and stable. Then the method is also convergent.*

Proof. By definition of the local truncation error, it holds that

$$\boldsymbol{x}(t_{n+1}) = \boldsymbol{x}(t_n) + \Delta\boldsymbol{\Phi}_\Delta\big(t_n, \boldsymbol{x}(t_n)\big) + \Delta\boldsymbol{\eta}_{n+1}.$$

Therefore, the sequence $\big(\boldsymbol{x}(t_n)\big)_{1\leqslant n\leqslant N}$ satisfies (7.18) with $\boldsymbol{\delta}_n = \boldsymbol{\eta}_n$, and so the stability estimate (7.19) implies that

$$\max_{1\leqslant n\leqslant N}\|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant S(T)\Delta\sum_{n=1}^{N}\|\boldsymbol{\eta}_n\|.$$

By consistency, the right-hand side converges to zero in the limit as $\Delta \to 0$, which concludes the proof. □

Remark 7.2. If we assume in Theorem 7.8 that the method is consistent with order p, then by adapting the proof, we find that the error satisfies

$$\max_{1 \leqslant n \leqslant N} \|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant CS(T)\Delta^p.$$

In this setting, the numerical scheme is said to be *convergent with order p*.

7.2.4 Widely used one-step methods

In this section, we motivate and describe some of the other widely-used one-step methods, namely methods of Taylor and Runge–Kutta type. We assume in this section that the equation (7.1) admits a unique smooth solution over the interval $[0, T]$.

Taylor methods

In order to construct a method with a smaller local truncation error than that of the forward Euler method, a Taylor expansion of higher order than (7.9) can be employed:

$$\boldsymbol{x}(t + \Delta) = \boldsymbol{x}(t) + \Delta \boldsymbol{x}'(t) + \cdots + \frac{\Delta^p}{p!}\boldsymbol{x}^{(p)}(t) + \mathcal{O}(\Delta^{p+1}). \tag{7.20}$$

Since $\boldsymbol{x}\colon [0, T] \to \mathbf{R}$ is a smooth solution to (7.1) by assumption, the time derivatives of $\boldsymbol{x}$ can be obtained by differentiation of (7.1):

$$\boldsymbol{x}'(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big), \qquad \boldsymbol{x}''(t) = \partial_t \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big) + \Big(\boldsymbol{f}(t, \boldsymbol{x}(t)) \cdot \nabla_{\boldsymbol{x}}\Big)\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big), \qquad \dots$$

In general, it is immediate to show inductively that $\boldsymbol{x}^{(p)}(t) = \boldsymbol{f}^{(p-1)}\big(t, \boldsymbol{x}(t)\big)$, where the functions $\boldsymbol{f}^{(p)}\colon \mathbf{R} \times \mathbf{R}^n \to \mathbf{R}$ are defined recursively from the following equation:

$$\boldsymbol{f}^{(p+1)} = \partial_t \boldsymbol{f}^{(p)}\big(t, \boldsymbol{x}(t)\big) + \Big(\boldsymbol{f}(t, \boldsymbol{x}(t)) \cdot \nabla_{\boldsymbol{x}}\Big)\boldsymbol{f}^{(p)}\big(t, \boldsymbol{x}(t)\big).$$

The Taylor expansion (7.20) motivates the so-called Taylor methods for integrating (7.1) numerically, which are based on the following iteration:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \boldsymbol{T}_\Delta^p\big(t_n, \boldsymbol{x}_n\big), \tag{7.21}$$

where

$$\boldsymbol{T}_\Delta^p(t, \boldsymbol{x}) \coloneqq \boldsymbol{f}(t, \boldsymbol{x}) + \frac{\Delta}{2!}\boldsymbol{f}^{(1)}(t, \boldsymbol{x}) + \cdots + \frac{\Delta^{p-1}}{p!}\boldsymbol{f}^{(p-1)}(t, \boldsymbol{x}).$$

Note that, for any p, the Taylor scheme (7.21) may be rewritten as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \frac{\mathrm{d}\boldsymbol{x}^{t_n, \boldsymbol{x}_n}}{\mathrm{d}t}(t_n) + \cdots + \frac{\Delta^p}{p!}\frac{\mathrm{d}^p \boldsymbol{x}^{t_n, \boldsymbol{x}_n}}{\mathrm{d}t^p}(t_n).$$

For $p = 1$, this scheme coincides with the forward Euler scheme.

Runge–Kutta methods

Runge–Kutta methods resemble Taylor methods, but they do not require to calculate the derivatives of the function $\boldsymbol{f}$. This is achieved by approximating the derivatives in Taylor methods by difference quotients. Consider for example the Taylor method of order $p = 2$:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \frac{\mathrm{d}\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t}(t_n) + \frac{\Delta^2}{2!}\frac{\mathrm{d}^2\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t^2}(t_n). \tag{7.22}$$

Substituting the approximation

$$\begin{aligned}\frac{\mathrm{d}^2\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t^2}(t_n) &\approx \frac{1}{\Delta}\left(\frac{\mathrm{d}\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t}(t_n+\Delta) - \frac{\mathrm{d}\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t}(t_n)\right)\\ &= \frac{1}{\Delta}\Big(\boldsymbol{f}\big(t_n+\Delta, \boldsymbol{x}^{t_n,\boldsymbol{x}_n}(t_n+\Delta)\big) - \boldsymbol{f}\big(t_n,\boldsymbol{x}_n\big)\Big)\\ &\approx \frac{1}{\Delta}\Big(\boldsymbol{f}\big(t_n+\Delta, \boldsymbol{x}_n+\Delta\boldsymbol{f}(t_n,\boldsymbol{x}_n)\big) - \boldsymbol{f}\big(t_n,\boldsymbol{x}_n\big)\Big)\end{aligned} \tag{7.23}$$

in (7.22), we obtain an explicit method known as *Heun's method*:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{\Delta}{2}\boldsymbol{f}(t_n,\boldsymbol{x}_n) + \frac{\Delta}{2}\boldsymbol{f}\big(t_n+\Delta, \boldsymbol{x}_n+\Delta\boldsymbol{f}(t_n,\boldsymbol{x}_n)\big).$$

It is possible to show that the local truncation error for this method also scales as Δ^2. Heun's method is a particular instance of a Runge–Kuta method. In general, an explicit Runge–Kutta method with s stages is of the form

$$\begin{aligned}\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \Delta\sum_{i=1}^{s} b_i\boldsymbol{k}_i\\ \boldsymbol{k}_1 &= \boldsymbol{f}(t_n,\boldsymbol{x}_n),\\ \boldsymbol{k}_2 &= \boldsymbol{f}\big(t_n + c_2\Delta, \boldsymbol{x}_n + \Delta(a_{21}\boldsymbol{k}_1)\big),\\ \boldsymbol{k}_3 &= \boldsymbol{f}\big(t_n + c_3\Delta, \boldsymbol{x}_n + \Delta(a_{31}\boldsymbol{k}_1 + a_{32}\boldsymbol{k}_2)\big),\\ &\vdots\\ \boldsymbol{k}_s &= \boldsymbol{f}\left(t_n + c_s\Delta, \boldsymbol{x}_n + \Delta\sum_{j=1}^{s-1} a_{sj}\boldsymbol{k}_j\right),\end{aligned}$$

with appropriate coefficients c_i and a_{ij}. Heun's iteration can be recast in this form as follows:

$$\begin{aligned}\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{\Delta}{2}(\boldsymbol{k}_1+\boldsymbol{k}_2)\\ \boldsymbol{k}_1 &= \boldsymbol{f}(t_n,\boldsymbol{x}_n)\\ \boldsymbol{k}_2 &= \boldsymbol{f}\big(t_n+\Delta, \boldsymbol{x}_n+\Delta\boldsymbol{k}_1\big).\end{aligned}$$

The approach we employed to construct Heun's method may be generalized to higher orders. For example, the most widely known Runge–Kutta method approximates the Taylor method of order $p = 4$ with the following iteration:

$$\begin{aligned}
\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{\Delta}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4), \\
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{x}_n), & \boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \frac{\Delta}{2}, \boldsymbol{x}_n + \Delta\frac{\boldsymbol{k}_1}{2}\right), \\
\boldsymbol{k}_3 &= \boldsymbol{f}\left(t_n + \frac{\Delta}{2}, \boldsymbol{x}_n + \Delta\frac{\boldsymbol{k}_2}{2}\right), & \boldsymbol{k}_4 &= \boldsymbol{f}(t_n + \Delta, \boldsymbol{x}_n + \Delta\boldsymbol{k}_3).
\end{aligned}$$

The local truncation error for this method scales as Δ^4 and, when $\boldsymbol{f}(t, \boldsymbol{x}) = \boldsymbol{f}(t)$, this method coincides with Simpson's formula (3.6) for the approximation of the integral in (7.17). The systematic derivation of Runge–Kutta methods is cumbersome, and so we do not address this issue in this course.

Remark 7.3. Explicit Runge–Kutta methods of a given order are not uniquely defined. For example, if we employ instead of (7.23) the approximation

$$\frac{\mathrm{d}^2\boldsymbol{x}^{t_n,\boldsymbol{x}_n}}{\mathrm{d}t^2}(t_n) \approx \frac{2}{\Delta}\left(\boldsymbol{f}\left(t_n + \frac{\Delta}{2}, \boldsymbol{x}_n + \frac{\Delta}{2}\boldsymbol{f}(t_n, \boldsymbol{x}_n)\right) - \boldsymbol{f}\big(t_n, \boldsymbol{x}_n\big)\right),$$

then we obtain by substitution in (7.22) the so-called *explicit midpoint method*, which is also a Runge–Kutta method of order 2:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{f}\left(t_n + \frac{1}{2}\Delta, \boldsymbol{x}_n + \frac{\Delta}{2}\boldsymbol{f}(t_n, \boldsymbol{x}_n)\right).$$

Implicit methods

To conclude this section, we mention two common implicit methods with a better order of convergence than that of the backward Euler method.

- The Crank–Nicolson method:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{\Delta}{2}\big(f(t_n, \boldsymbol{x}_n) + f(t_n + \Delta, \boldsymbol{x}_{n+1})\big). \tag{7.24}$$

  When $\boldsymbol{f}$ is independent of $\boldsymbol{x}$ and depends only on t, this method coincides with the trapezoidal rule for numerical integration.

- The implicit midpoint method:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta f\left(t_n + \frac{\Delta}{2}, \frac{\boldsymbol{x}_n + \boldsymbol{x}_{n+1}}{2}\right).$$

Similarly to the backward Euler method, each iteration of these methods requires the resolution of a nonlinear equation. Implicit methods often enjoy better stability than their explicit counterparts. This subject is further discussed in Section 7.4.

7.3 Multistep methods

The idea of multistep methods is to use, in the construction of a new iterate, information from not only the current but also previous iterations. This degree of freedom enables to construct more economical numerical methods than one-step methods for the same order of convergence, at the cost of a more difficult initialization. In this section we focus on *linear* multistep methods of the form

$$\begin{aligned}\boldsymbol{x}_{n+1} = a_0\boldsymbol{x}_n + a_1\boldsymbol{x}_{n-1} + \cdots + a_k\boldsymbol{x}_{n-k} \\ +\Delta\Big(b_{-1}\boldsymbol{f}(t_{n+1},\boldsymbol{x}_{n+1}) + b_0\boldsymbol{f}(t_n,\boldsymbol{x}_n) + \cdots + b_k\boldsymbol{f}(t_{n-k},\boldsymbol{x}_{n-k})\Big).\end{aligned} \tag{7.25}$$

This equation defines an explicit method if $b_{-1} = 0$, and an implicit method if $b_{-1} \neq 0$. Note that explicit methods of the form (7.25) require only one additional evaluation $f(t_n, \boldsymbol{x}_n)$ per iteration, in contrast with Runge–Kutta methods. When $b_{-1} \neq 0$, the iteration (7.25) is a nonlinear equation for the unknown $\boldsymbol{x}_{n+1}$, which must itself be solved by resorting to a numerical method.

Initialization. In order to initiate the numerical method (7.25), the values $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$ are required. These can be calculated by using a one-step method with an order of convergence matching that of the multistep method.

Local truncation error. Consistently with the setting of one-step methods, the local truncation error for (7.25) is defined as the residual error left when the exact solution is substituted in the numerical scheme:

$$\begin{aligned}\Delta\boldsymbol{\eta}_{n+1} \coloneqq \boldsymbol{x}(t_n+\Delta) - a_0\boldsymbol{x}(t_n) - a_1\boldsymbol{x}(t_n-\Delta) - \cdots - a_k\boldsymbol{x}(t_n-k\Delta) \\ -\Delta\Big(b_{-1}\boldsymbol{x}'(t_n+\Delta) + b_0\boldsymbol{x}'(t_n) + \cdots + b_k\boldsymbol{x}'(t_n-k\Delta)\Big).\end{aligned} \tag{7.26}$$

The multistep method (7.25) is said to be of order p if the maximum local truncation error over all the discretization points, in norm, scales as $\mathcal{O}(\Delta^p)$. The following result is useful for estimating the order of consistency of a linear multistep method.

Proposition 7.9. *The linear multistep method (7.25) is consistent with order p for any smooth $\boldsymbol{x}\colon [0,T] \to \mathbf{R}^n$ if and only if the local truncation error (7.26) is everywhere zero when $\boldsymbol{x}(t)$ is of the scalar form*

$$x(t) = t^q, \qquad q \in \{0, \ldots, p\}. \tag{7.27}$$

Proof. Assume that the method is consistent with order p, fix $q \in \{1, \ldots, p\}$, and let $x(t) = t^q$. Fix also $t \in [0,T]$ and consider the function $\xi\colon \{\Delta : t/\Delta \in \mathbf{N}_{>0}\} \to \mathbf{R}$ given by

$$\Delta\xi(\Delta) = \Delta\eta_{(t/\Delta)+1} = x(t+\Delta) - a_1 x(t) - a_2 x(t-\Delta) - \cdots - a_k x\Big(t-(k-1)\Delta\Big) \\ -\Delta\Big(b_0 x'(t+\Delta) + b_1 x'(t) + \cdots + b_k x'\Big(t-(k-1)\Delta\Big)\Big).$$

The quantity $\xi(\Delta)$ should be understood as the local truncation error at t for time step Δ. It is a polynomial in Δ of degree at most p and scaling as $\mathcal{O}(\Delta^{p+1})$. Therefore, it holds necessarily that $\xi(\Delta) = 0$.

Conversely, assume that the right-hand side of (7.26) is equal to zero for any function of the form (7.27). If $\boldsymbol{x}(t)$ denotes a smooth solution of (7.1), then by Taylor's theorem there is $C > 0$ independent of t_n such that

$$\forall t \in [0,T], \qquad \begin{cases} \|\boldsymbol{x}(t) - \boldsymbol{y}(t)\| \leqslant C|t-t_n|^{p+1} \\ \|\boldsymbol{x}'(t) - \boldsymbol{y}'(t)\| \leqslant C|t-t_n|^{p} \end{cases}, \qquad \boldsymbol{y}(t) := \boldsymbol{x}(t_n) + \sum_{i=1}^{p} \boldsymbol{e}_i (t-t_n)^i,$$

for appropriate vectors $\boldsymbol{e}_i \in \mathbf{R}^n$ depending on t_n. Substituting $\boldsymbol{x}(t) = \boldsymbol{y}(t) + \Big(\boldsymbol{x}(t) - \boldsymbol{y}(t)\Big)$ in the right-hand side of (7.26), we obtain

$$\Delta\|\boldsymbol{\eta}_{n+1}\| = \mathcal{O}(\Delta^{p+1}) + \Delta\ \mathcal{O}(\Delta^p) = \mathcal{O}(\Delta^{p+1}),$$

with the constant implicit in the big $\mathcal{O}$ notation independent of n. This concludes the proof. □

Example 7.2. In the one-dimensional setting, we wish to find parameters a_0, a_1 and b_1 such that the order of consistency of the following multistep scheme is as high as possible:

$$x_{n+1} = a_0 x_n + a_2 x_{n-1} + b_0 \Delta f(t_n, x_n).$$

Substituting $x(t) = 1$ in the formula (7.26) for the local truncation error, we obtain

$$\eta_{n+1} = x(t_n + \Delta) - a_0 x(t_n) - a_1 x(t_n - \Delta) - b_0 \Delta x'(t_n) = 1 - a_0 - a_1.$$

Therefore $a_1 = (1 - a_0)$. Next, substituting $x(t) = t - t_n$ in (7.26), we obtain

$$\eta_{n+1} = \Delta(2 - a_0 - b_0),$$

which gives $b_0 = 2 - a_0$. Finally, substituting $x(t) = (t-t_n)^2$, we obtain

$$\eta_{n+1} = \Delta^2 a_0,$$

and so $a_0 = 1$. We conclude that the best parameters, leading to a local truncation error scaling as $\mathcal{O}(\Delta^2)$, are given by $a_0 = 0$, $a_1 = 1$ and $b_0 = 2$. The resulting method reads

$$x_{n+1} = x_{n-1} + 2\Delta f(t_n, x_n),$$

and is known as the *multistep midpoint method*.

We now present two widely used systematic approaches for constructing multistep methods, known as the Adams–Bashforth and Adams–Moulton approaches.

i	0	1	2	3
$k=0$	1			
$k=1$	$\frac{3}{2}$	$-\frac{1}{2}$		
$k=2$	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	
$k=3$	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$

Table 7.1: Coefficients $(b_i)_{0 \leqslant i \leqslant k}$ of the Adams–Bashforth methods.

7.3.1 Adams–Bashforth methods

Let $\boldsymbol{x}\colon [0,T] \to \mathbf{R}^n$ denote the exact solution to the differential equation (7.1). Integrating this equation between t_n and t_{n+1}, we obtain

$$\boldsymbol{x}(t_{n+1}) = \boldsymbol{x}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)\,\mathrm{d}t. \tag{7.28}$$

The key idea of the Adams–Bashforth method is to approximate the function $t \mapsto \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$ by the interpolating polynomial $\hat{\boldsymbol{f}}$ of degree k at the nodes $t_{n-k}, \ldots, t_n$:

$$\hat{\boldsymbol{f}}(t) = \sum_{i=0}^{k} \boldsymbol{f}\big(t_{n-i}, \boldsymbol{x}(t_{n-i})\big) L_i(t), \qquad L_i(t) \coloneqq \prod_{\substack{j=0 \\ j \neq i}}^{k} \frac{t - t_{n-j}}{t_{n-i} - t_{n-j}}. \tag{7.29}$$

Substituting this approximation in (7.28), we obtain

$$\boldsymbol{x}(t_{n+1}) \approx \boldsymbol{x}(t_n) + \sum_{i=0}^{k} \boldsymbol{f}\big(t_{n-i}, \boldsymbol{x}(t_{n-i})\big) \int_{t_n}^{t_{n+1}} L_i(t)\,\mathrm{d}t.$$

This motivates the following *Adams–Bashforth* numerical scheme:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta \sum_{i=0}^{k} b_i \boldsymbol{f}\big(t_{n-i}, \boldsymbol{x}_{n-i}\big), \qquad b_i \coloneqq \int_0^1 \prod_{\substack{j=0 \\ j \neq i}}^{k} \frac{s+j}{-i+j}\,\mathrm{d}s. \tag{7.30}$$

Since the Lagrange polynomials $(L_i)_{0 \leqslant i \leqslant k}$ depend on k, so do the coefficients b_i. However, these are independent of Δ, and so they can be tabulated. The value of these coefficients for the first few Adams–Bashforth methods are collated in Table 7.1.

Local truncation error. Assuming $\boldsymbol{x} \in C^{k+2}\big([0,T], \mathbf{R}^n\big)$ and applying Theorem 2.3 for the interpolation error component-wise, we obtain

$$\forall t \in [0,T], \qquad \big\|\boldsymbol{x}'(t) - \hat{\boldsymbol{f}}(t)\big\|_\infty \leqslant \frac{|t - t_{n-k}| \cdots |t - t_n|}{(k+1)!} \sup_{t \in [0,T]} \big\|\boldsymbol{x}^{(k+2)}(t)\big\|_\infty,$$

where $\hat{\boldsymbol{f}}$ is the function defined in (7.29). Since

$$\Delta\boldsymbol{\eta}_{n+1} = \boldsymbol{x}(t_{n+1}) - \boldsymbol{x}(t_n) - \Delta\sum_{i=0}^{k} b_i \boldsymbol{f}\Big(t_{n-i}, \boldsymbol{x}(t_{n-i})\Big) = \int_{t_n}^{t_{n+1}} \Big(\boldsymbol{x}'(t) - \hat{\boldsymbol{f}}(t)\Big)\,\mathrm{d}t,$$

we deduce that

$$\|\boldsymbol{\eta}_{n+1}\| \leqslant C_k M_{k+2} \Delta^{k+1}, \qquad M_{k+2} \coloneqq \sup_{t\in[0,T]} \big\|\boldsymbol{x}^{(k+2)}(t)\big\|_\infty, \tag{7.31}$$

for an appropriate numerical constant C_k independent of n and of the problem data. Therefore the Adams–Bashforth method (7.30) is consistent with order $k+1$.

Convergence. By using a reasoning similar to that in the proof of Theorem 7.5, we can prove a convergence result of the Adams–Bashforth method.

Theorem 7.10. *Assume that the solution $\boldsymbol{x}\colon [0,T] \to \mathbf{R}^n$ to (7.1) is $k+2$ times continuously differentiable and that the global Lipschitz condition (7.10) is satisfied. Suppose also that*

$$\forall i \in \{0, \ldots, k\}, \qquad \|\boldsymbol{x}(t_i) - \boldsymbol{x}_i\| \leqslant \delta.$$

Then the following error estimate holds for the Adams–Bashforth method (7.30):

$$\forall n \in \left\{0, 1, \ldots, \left\lfloor \frac{T}{\Delta} \right\rfloor\right\}, \qquad \|\boldsymbol{x}(t_n) - \boldsymbol{x}_n\| \leqslant \delta \mathrm{e}^{LB} + C_k M_{k+2} \Delta^{k+1} \left(\frac{\mathrm{e}^{LBt_n} - 1}{LB}\right),$$

where C_k and M_{k+2} are the constants from (7.31), and with $B \coloneqq |b_0| + \cdots + |b_k|$.

Sketch of proof. Let $\boldsymbol{e}_n \coloneqq \boldsymbol{x}(t_{n+1}) - \boldsymbol{x}_{n+1}$. From the equation

$$\boldsymbol{x}(t_{n+1}) - \boldsymbol{x}_{n+1} = \boldsymbol{x}(t_n) - \boldsymbol{x}_n + \Delta\sum_{i=0}^{k} b_i \Big(\boldsymbol{f}\Big(t_{n-i}, \boldsymbol{x}(t_{n-i})\Big) - \boldsymbol{f}\Big(t_{n-i}, \boldsymbol{x}_{n-i}\Big)\Big) + \Delta\boldsymbol{\eta}_{n+1},$$

which is valid for $n \geqslant k$, we deduce that

$$\max\Big\{\|\boldsymbol{e}_0\|, \ldots, \|\boldsymbol{e}_{n+1}\|\Big\} \leqslant (1 + \Delta LB) \max\Big\{\|\boldsymbol{e}_0\|, \ldots, \|\boldsymbol{e}_n\|\Big\} + C_k M_{k+2} \Delta^{k+2}.$$

Since $\max\Big\{\|\boldsymbol{e}_0\|, \ldots, \|\boldsymbol{e}_k\|\Big\} \leqslant \delta$ by assumption, the statement easily follows. □

7.3.2 Adams–Moulton methods

The Adams–Moulton methods are very similar to their Adams–Bashforth cousins. The only difference is that the former are obtained by interpolating the function $t \mapsto \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$ in (7.28) at nodes shifted forward by Δ, i.e. at the nodes $t_{n-k+1}, \ldots, t_{n+1}$. This leads to the method

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\sum_{i=-1}^{k-1} b_i \boldsymbol{f}\Big(t_{n-i}, \boldsymbol{x}_{n-i}\Big), \qquad b_i \coloneqq \int_0^1 \prod_{\substack{j=-1 \\ j\neq i}}^{k-1} \frac{s+j}{-i+j}\,\mathrm{d}s. \tag{7.32}$$

i	-1	0	1	2
$k=0$	1			
$k=1$	$\frac{1}{2}$	$\frac{1}{2}$		
$k=2$	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	
$k=3$	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$

Table 7.2: Coefficients $(b_i)_{0 \leqslant i \leqslant k}$ of the Adams–Moulton methods.

Unlike the Adams–Bashforth methods, which are *explicit*, the Adams–Moulton methods are *implicit*. The value of the coefficients for the first few Adams–Moulton methods are collated in Table 7.2. Notice that, for $k = 0$, the Adams–Moulton method coincides with the backward Euler method, and for $k = 1$ it coincides with the Crank–Nicolson method.

7.4 Absolute stability

To conclude this chapter, we introduce the notion of *absolute stability* and explain its relevance. Absolute stability is a property of a numerical method in relation to the model equation

$$\begin{cases} x'(t) = \lambda x(t), \\ x(0) = 1. \end{cases} \tag{7.33}$$

A numerical scheme for approximating (7.33) is called *absolutely stable* if

$$|x_n| \to 0 \qquad \text{in the limit as } \; n \to \infty. \tag{7.34}$$

where $(x_n)_{n=0,1,\dots}$ denotes the numerical solution to (7.33). Whether a numerical method is absolutely stable or not depends on the parameters λ and Δ.

Example 7.3. The forward Euler method for (7.33) reads

$$x_{n+1} = x_n + \Delta\lambda x_n = (1 + \Delta\lambda)x_n.$$

Therefore $x_n \to 0$ if and only if $|1 + \Delta\lambda| \leqslant 1$.

As Example 7.3 illustrates, whether absolute stability holds for the forward Euler methods depends only the value of the product $\Delta\lambda \in \mathbf{C}$. This dependence on λ and Δ only through the product $\Delta\lambda$ holds in fact generally. Indeed, all the numerical schemes we considered in this chapter are invariant under linear time rescaling of the ordinary differential equation: the numerical solution of the rescaled equation, when the time step is rescaled by the same factor, coincides with the discrete function obtained by linear rescaling of the numerical solution to the original equation. This motivates the definition of *absolute stability region* as

$$\mathcal{A} := \{z \in \mathbf{C} : (7.34) \text{ holds when } \Delta\lambda = z\} \subset \mathbf{C}.$$

The exact solution to the model equation (7.33) diverges to ∞ as $t \to \infty$ if $\Re(\lambda) > 0$, and it converges to 0 if $\Re(\lambda) < 0$. Numerical schemes which exhibit a similar property at the discrete

level are called *A-stable*. More precisely, a numeric method is A-stable if the absolute stability region $\mathcal{A}$ contains the left half-plane, i.e. if

$$\{z \in \mathbf{C} : \Re(z) < 0\} \subset \mathcal{A}.$$

Before investigating whether the numerical schemes introduced previously in this chapter are absolutely stable, we address the following natural question: why focus on the simple model equation (7.33)? We provide a couple of motivations:

- First, note that equations of the form (7.33) are more relevant in science than might appear at first glance. Indeed, when discretizing in space a linear parabolic partial differential equation, one often obtains a linear differential equation of the form

  $$\boldsymbol{x}'(t) = \mathsf{A}\boldsymbol{x},$$

  where $\mathsf{A} \in \mathbf{C}^{n\times n}$. If the matrix $\mathsf{A} = \mathsf{Q}\mathsf{D}\mathsf{Q}^*$ is diagonalizable, then the vector $\boldsymbol{z}(t) := \mathsf{Q}^*\boldsymbol{x}(t)$ satisfies the differential equation

  $$\boldsymbol{z}'(t) = \mathsf{D}\boldsymbol{z}.$$

  In other words, each component of $\boldsymbol{z}$ satisfies an ordinary differential equation of the same form as the model equation (7.33). In applications, the components of $\boldsymbol{z}$ often encode the amplitudes of Fourier modes of the solution to the partial differential equation, and for dissipative equations all the eigenvalues of A have a negative real part. However, the spectral radius of A usually diverges as the number of discretization points increases. In this context, A-stability is particularly attractive, as it ensures that the numerical approximation remains well-behaved as the number of discretization points increases.

- Second, the model equation (7.33) may be viewed as a linearized approximation of a more interesting equation. Consider, for example, the following one-dimensional autonomous differential equation:

  $$\begin{cases} x'(t) = f\big(x(t)\big), \\ x(0) = x_0. \end{cases} \tag{7.35}$$

  Assume that $f(x_*) = 0$ for some $x_* \in \mathbf{R}$. Such a point is called a *critical point* of the differential equation. If $f'(x_*) < 0$, then x_* is an attractor of the equation, in the sense that $x(t) \to x_*$ provided that $x(0)$ is sufficiently close to x_*. This result, which is the counterpart of Proposition 5.5 for differential equations, is a particular case of a theorem due to Poincaré and Lyapunov; see [16, Theorem 7.1]. If $|x_0 - x_*|$ is sufficiently small, then the solution to (7.35) is expected to be close to that of the linearized equation

  $$\begin{cases} y'(t) = f'\big(x_*\big)\big(y(t) - x_*\big), \\ y(0) = x_0, \end{cases} \tag{7.36}$$

which is of the form (7.33). Often, studying the linearized equation (7.36) enables to gain insight into the behavior of the original equation (7.35), and analyzing the performance of a numerical method for the linearized equation (7.36) is useful to inform the choice of a numerical scheme for (7.35).

- More generally, if $x(t)$ and $x_\varepsilon(t)$ are respectively the solutions to

$$\begin{cases} x'(t) = f\big(t, x(t)\big), \\ x(0) = x_0 + \varepsilon, \end{cases} \qquad \text{and} \qquad \begin{cases} x_\varepsilon'(t) = f\big(t, x_\varepsilon(t)\big), \\ x_\varepsilon(0) = x_0 + \varepsilon, \end{cases} \tag{7.37}$$

then the difference $e(t) := x_\varepsilon(t) - x(t)$ satisfies the equation

$$\begin{aligned} e'(t) &= f\big(t, x_\varepsilon(t)\big) - f\big(t, x(t)\big) \approx \partial_x f\big(t, x(t)\big) e(t), \\ e(0) &= \varepsilon, \end{aligned} \tag{7.38}$$

which looks similar to (7.33) with $\partial_x f\big(t, x(t)\big)$ in place of λ. At a given time t, the solutions tend to converge to each other as time increases if $\partial_x f\big(t, x(t)\big) < 0$, and diverge from each other if $\partial_x f\big(t, x(t)\big) > 0$. Testing absolute stability with $\lambda = \partial_x f\big(t, x(t)\big)$ enables to determine whether this property holds true also at the discrete level. Although the latter statement is difficult to state precisely and prove generally, we illustrate its validity for the forward Euler method in Example 7.4.

Example 7.4. Let (x_n) and (x_n^ε) denote the numerical solutions obtained by applying the forward Euler method to the differential equations in (7.37). If $\varepsilon \ll 1$, then

$$\begin{aligned} x_{n+1}^\varepsilon - x_{n+1} &= x_n^\varepsilon - x_n + \Delta f(t_n, x_n^\varepsilon) - \Delta f(t_n, x_n) \\ &\approx x_n^\varepsilon - x_n + \Delta \partial_x f(t_n, x_n)(x_n^\varepsilon - x_n) = \big(1 + \Delta \partial_x f(t_n, x_n)\big)(x_n^\varepsilon - x_n). \end{aligned}$$

Therefore, the numerical solutions (x_n^ε) and (x_n) tend to become closer as n increases if

$$\Delta \partial_x f(t_n, x_n) \in \mathcal{A}. \tag{7.39}$$

The absolute stability regions of the forward and backward Euler methods are illustrated in green in Figure 7.1. For the forward Euler method, absolute stability holds if and only if $|1 + \Delta\lambda| < 1$, as we proved in Example 7.3. A similar reasoning gives that the absolute stability region for backward Euler method is given by $\{z \in \mathbf{C} : |1 - z|^{-1} < 1\}$. The backward Euler method is A-stable but the forward Euler method is not. Notice that, if the time step is sufficiently large, then the backward Euler method is absolutely stable even for values of λ with a positive real part, for which exact solutions to the model equation are divergent.

Example 7.5 (Absolute stability region of the Taylor methods). When applied to (7.33), the Taylor method of order p given in (7.21) reads

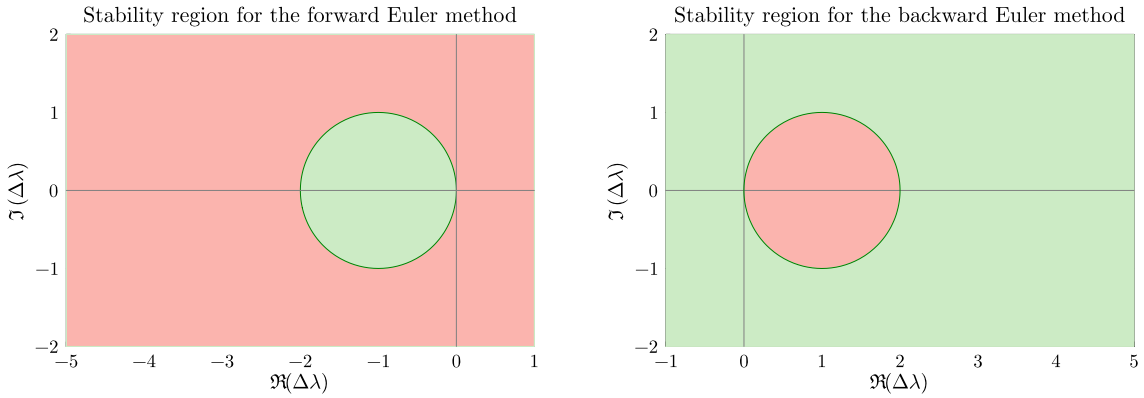


Figure 7.1: Absolute stability regions for the forward (left) and backward (right) Euler methods.

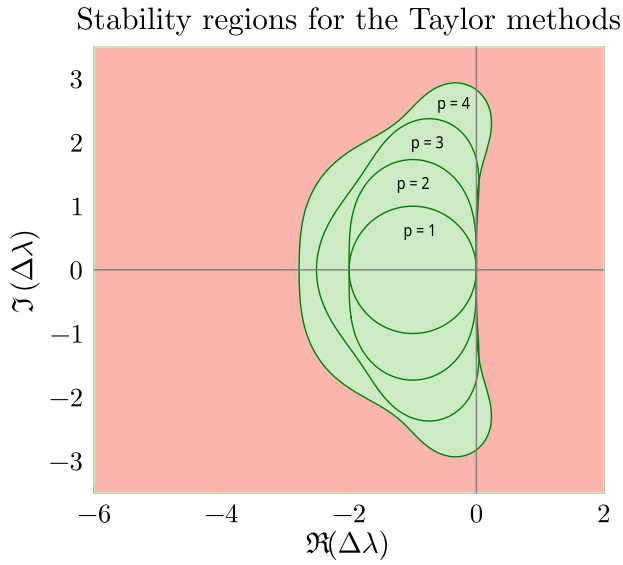


Figure 7.2: Stability regions for the first few Taylor methods.

$$x_{n+1} = \left(1 + \Delta\lambda + \frac{\Delta^2\lambda^2}{2} + \cdots + \frac{\Delta^p\lambda^p}{p!}\right) x_n.$$

Thus, the absolute stability region is given by

$$\left\{ z \in \mathbf{C} : \left| 1 + z + \frac{z^2}{2} + \cdots + \frac{z^p}{p!} \right| < 1 \right\}.$$

This region is illustrated for various values of p in Figure 7.2. We observe that the absolute stability region grows as p increases.

Stiff differential equations

In the context of ordinary differential equations, stiffness is not a precisely defined concept, but rather rather a generic term employed to describe equations with widely separated time scales. Roughly speaking, a differential equation of the form (7.1) is called stiff if the Jacobian matrix

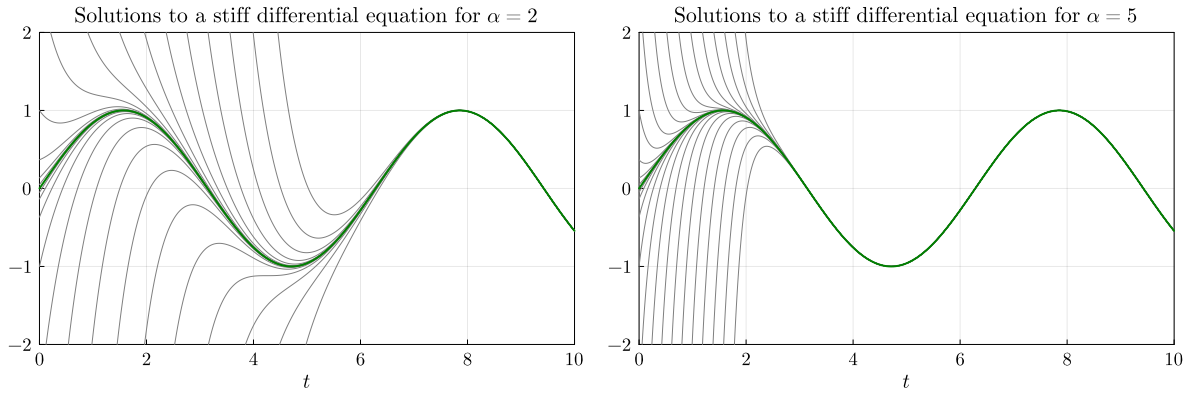


Figure 7.3: Solutions to (7.40) for various initial conditions when $\alpha = 2$ (left) and $\alpha = 5$ (right).

of f, with respect to the variable $\boldsymbol{x}$, has at least one eigenvalue with a large negative real part. In the one-dimensional setting, the solutions to stiff differential equations which are close at the initial time tend to converge quickly to each other, in view of (7.38). This is illustrated in Example 7.6.

Example 7.6 (Stiff differential equation). Consider the following equation [7, Chapter 4]:

$$\begin{cases} x'(t) = -\alpha\big(x(t) - \sin(t)\big) + \cos(t) \\ x(0) = x_0 \end{cases} \tag{7.40}$$

The exact solution to this equation is given by

$$x(t) = \sin(t) + x_0\, \mathrm{e}^{-\alpha t}.$$

When $\alpha \in \mathbf{R}$ is large, the distance between the solution and the function $t \mapsto \sin(t)$ converges to zero very quickly, regardless of the initial condition. This behavior is illustrated in Figure 7.3.

In the rest of this section, we use the differential equation (7.40) as a guiding example. For this problem, we have $\partial_x f(t, x) = \alpha$. Therefore, in view of (7.39), we expect that the forward Euler scheme is non-divergent if $|1 - \alpha\Delta| < 1$, i.e. if

$$\Delta < \Delta_* \sim= \frac{2}{\alpha}.$$

It turns out that this prediction is precise, as depicted in Figure 7.4. Note that if the equation is very stiff, that is to say if $\alpha \gg 1$, then a very small time step is required to ensure stability.

In contrast with the forward Euler scheme, the backward Euler scheme is stable regardless of the time step. Since the right-hand side of (7.40) is linear in x, the value of the iterate x_{n+1} can be calculated explicitly from x_n for the backward scheme:

$$x_{n+1} = \frac{x_n + \Delta\alpha \sin(t_{n+1}) + \Delta \cos(t_{n+1})}{1 + \Delta\alpha}.$$

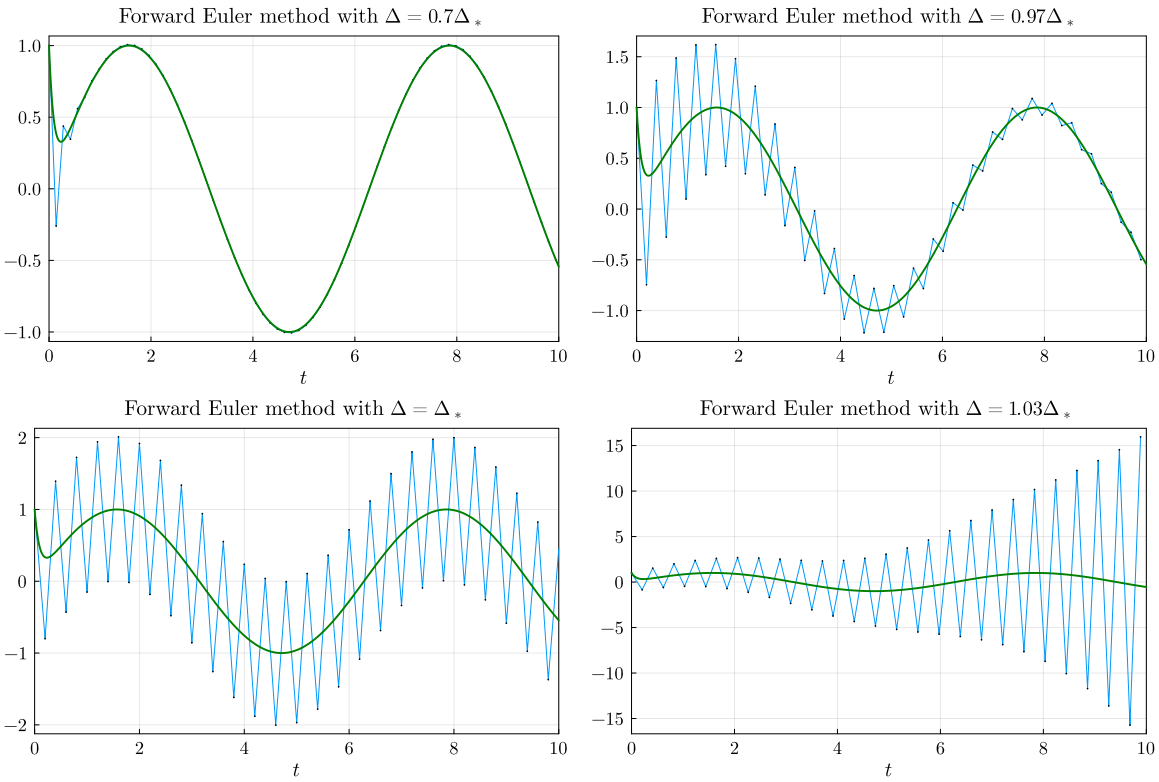


Figure 7.4: Numerical approximations of the solution to (7.40) with $\alpha = 10$ obtained with the forward Euler method, for four different values of Δ.

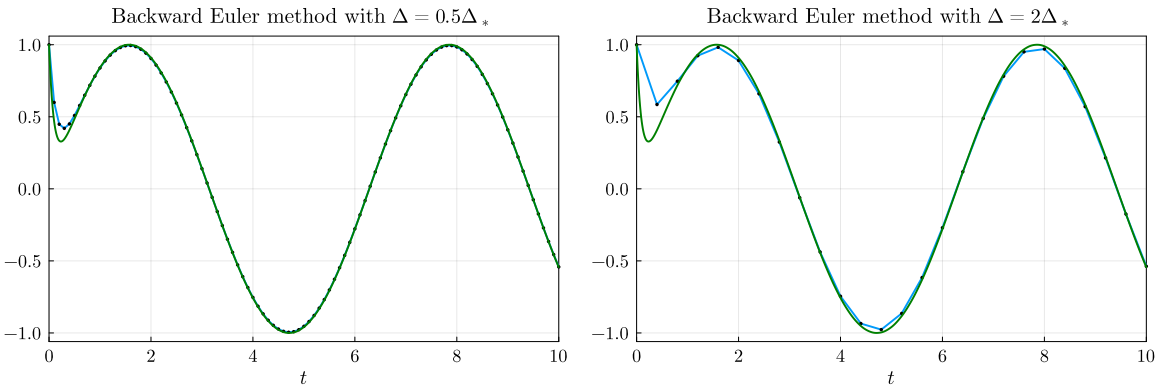


Figure 7.5: Numerical approximations of the solution to (7.40) with $\alpha = 10$ obtained with the backward Euler method, for two different values of Δ.

Numerical approximations obtained using this scheme are illustrated in Figure 7.5. We observe that the method is stable even for the large time step $\Delta = 2\Delta_*$.

7.5 Exercises

Exercise 7.1. *Show that the absolute stability region of the Crank–Nicolson method (7.24) is given by the left half-plane; see Figure 7.6.*

Exercise 7.2. *Calculate the absolute stability region for Gear's method.*

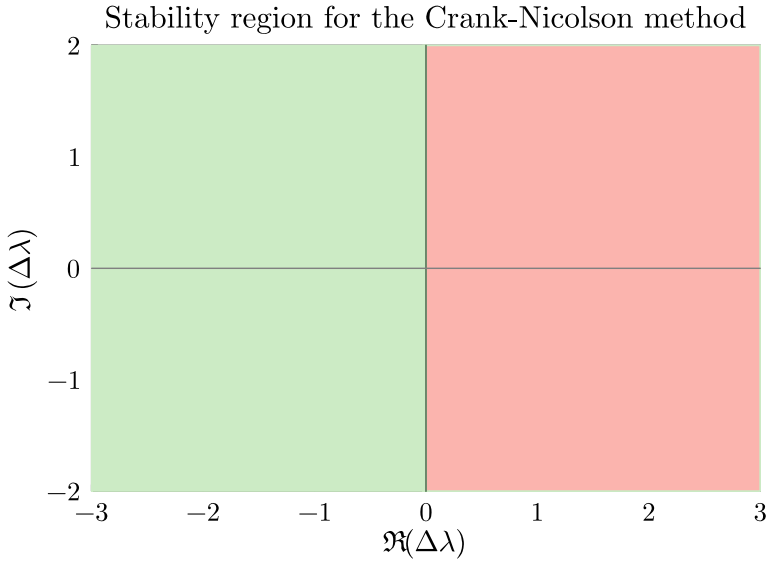


Figure 7.6: Absolute stability regions for the Crank Nicolson method.

Chapter 8

Optimization

8.1 Definition and characterization of convexity 200
8.2 Unconstrained optimization 202
8.3 Constrained optimization 204

In this chapter, we focus on optimization problems of the following form:

$$\text{Find} \quad \boldsymbol{x}_* \in \operatorname*{arg\,min}_{\boldsymbol{x} \in \mathcal{K}} J(\boldsymbol{x}), \tag{8.1}$$

where $\mathcal{K}$ is a given subset of $\mathbf{R}^n$ and $J\colon \mathcal{K} \to \mathbf{R}$ is a given *objective function*. We came across several examples of such problems earlier in these notes:

- In Chapter 2, in the context of least-squares approximation, we considered the problem of minimizing

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\|\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}\|^2.$$

- In Chapter 4, we observed that, if A is a symmetric and positive definite matrix, then solving the linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ amounts to finding the minimizer of the functional

$$J(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\mathsf{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}.$$

When $\mathcal{K} = \mathbf{R}^n$, equation (8.1) is an *unconstrained* optimization problem, and when $\mathcal{K} \subsetneq \mathbf{R}^n$, equation (8.1) is a *constrained* optimization problem. In practice, the set $\mathcal{K}$ is often an intersection of sets of the form

$$\Big\{\boldsymbol{x} \in \mathbf{R}^n : \phi(\boldsymbol{x}) \leqslant 0\Big\}, \qquad \text{or} \qquad \Big\{\boldsymbol{x} \in \mathbf{R}^n : \phi(\boldsymbol{x}) = 0\Big\},$$

for appropriate $\phi\colon \mathbf{R}^n \to \mathbf{R}$. Constraints of the former form are called *inequality constraints*, while constraints of the latter form are called *equality constraints*. Our aim in this chapter is to

give a brief introduction to numerical optimization. We focus on the simplest method, namely the *steepest descent method* with fixed step. The rest of this chapter is organized as follows:

- We begin in Section 8.1 by defining the notions of *convexity*, *strict convexity* and *strong convexity*, which play an important role in optimization.
- Then, in Section 8.2, we analyze the steepest descent method with fixed step in the setting of unconstrained optimization. To this end, we first establish conditions under which (8.1) is well posed.
- Finally, in Section 8.3, we extend the steepest descent method to the case of optimization with constraints.

Remark 8.1. For generality, we could consider the setting where the set $\mathcal{K}$ in (8.1) is a subset of some finite dimensional or infinite dimensional vector space V. An optimization problem over (a subset of) a finite dimensional vector space of dimension n can always be recast as an optimization problem over (a subset of) $\mathbf{R}^n$ – the type we study in this chapter – by fixing a basis. The case of an infinite dimensional vector space, however, is more delicate, and we do not address it here.

8.1 Definition and characterization of convexity

Definition 8.1 (Convexity)**.** Assume that $J\colon \mathcal{K} \to \mathbf{R}$.

- The function J is said to be *convex* if

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{K} \times \mathcal{K}, \quad \forall \theta \in [0,1], \quad J\Big(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}\Big) \leqslant \theta J(\boldsymbol{x}) + (1-\theta) J(\boldsymbol{y}). \tag{8.2}$$

- The function J is called *strictly convex* if (8.2) holds with strict inequality if $\boldsymbol{x} \neq \boldsymbol{y}$ and $\theta \in (0,1)$.
- The function J is called *strongly convex* with parameter $\alpha > 0$ if for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{K} \times \mathcal{K}$ and for all $\theta \in [0,1]$,

$$J\Big(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}\Big) \leqslant \theta J(\boldsymbol{x}) + (1-\theta) J(\boldsymbol{y}) - \frac{\alpha}{2}\theta(1-\theta)\|\boldsymbol{x} - \boldsymbol{y}\|^2. \tag{8.3}$$

If the function J is differentiable, then convexity, strict convexity and strong convexity can be characterized in terms of the gradient ∇J. We illustrate this for strong convexity, noting that a characterization of convexity is obtained by substituting $\alpha = 0$ in the following result.

Proposition 8.1. *A differentiable function $J\colon \mathbf{R}^n \to \mathbf{R}$ is strongly convex with parameter α if and only if*

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n, \qquad J(\boldsymbol{x}) \geqslant J(\boldsymbol{y}) + \Big\langle \nabla J(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2, \tag{8.4}$$

or, equivalently,

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n, \qquad \Big\langle \nabla J(\boldsymbol{x}) - \nabla J(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \Big\rangle \geqslant \alpha \|\boldsymbol{x} - \boldsymbol{y}\|^2. \tag{8.5}$$

Proof. For clarity, we divide the proof into items and prove one implication per item.

- (8.3) ⇒ (8.4). Rearranging (8.3), we have

$$\frac{J\Big(\boldsymbol{y} + \theta(\boldsymbol{x} - \boldsymbol{y})\Big) - J(\boldsymbol{y})}{\theta} \leqslant J(\boldsymbol{x}) - J(\boldsymbol{y}) - \frac{\alpha}{2}(1 - \theta)\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

  Taking the limit $\theta \to 0$, we deduce that

$$\Big\langle \nabla J(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \Big\rangle \leqslant J(\boldsymbol{x}) - J(\boldsymbol{y}) - \frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

  This gives (8.4) after rearranging.

- (8.4) ⇒ (8.3). To prove this implication, suppose that (8.4) holds, take $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n$ and let $\boldsymbol{z} = \theta\boldsymbol{x} + (1 - \theta)\boldsymbol{y}$. Using (8.4) successively with $(\boldsymbol{x}, \boldsymbol{z})$ and $(\boldsymbol{y}, \boldsymbol{z})$, we deduce

$$\begin{aligned} J(\boldsymbol{x}) &\geqslant J(\boldsymbol{z}) + \Big\langle \nabla J(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{z}\|^2, \\ J(\boldsymbol{y}) &\geqslant J(\boldsymbol{z}) + \Big\langle \nabla J(\boldsymbol{z}), \boldsymbol{y} - \boldsymbol{z} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{y} - \boldsymbol{z}\|^2. \end{aligned}$$

  Combining these inequalities, we deduce that

$$\begin{aligned} \theta J(\boldsymbol{x}) + (1 - \theta)J(\boldsymbol{y}) &\geqslant J(\boldsymbol{z}) + \Big\langle \nabla J(\boldsymbol{z}), \theta\boldsymbol{x} + (1 - \theta)\boldsymbol{y} - \boldsymbol{z} \Big\rangle \\ &\quad + \frac{\alpha\theta}{2}\|\boldsymbol{x} - \boldsymbol{z}\|^2 + \frac{\alpha(1 - \theta)}{2}\|\boldsymbol{y} - \boldsymbol{z}\|^2 \\ &= J(\boldsymbol{z}) + 0 + \frac{\alpha}{2}\theta(1 - \theta)\|\boldsymbol{x} - \boldsymbol{y}\|^2. \end{aligned}$$

  Rearranging gives (8.3).

- (8.4) ⇒ (8.5). Assuming that (8.4) holds and applying this inequality first to $(\boldsymbol{x}, \boldsymbol{y})$ and then to $(\boldsymbol{y}, \boldsymbol{x})$, we obtain

$$\begin{aligned} J(\boldsymbol{x}) &\geqslant J(\boldsymbol{y}) + \Big\langle \nabla J(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 \\ J(\boldsymbol{y}) &\geqslant J(\boldsymbol{x}) + \Big\langle \nabla J(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2. \end{aligned}$$

  Adding these equations and rearranging, we deduce (8.5).

- (8.5) ⇒ (8.4). Suppose that (8.5) holds and take $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n$. Using the fundamental theorem of analysis and (8.5), we have

$$
\begin{aligned}
J(\boldsymbol{x}) &= J(\boldsymbol{y}) + \int_0^1 \Big\langle \nabla J\big(\boldsymbol{y} + \theta(\boldsymbol{x}-\boldsymbol{y})\big), \boldsymbol{x}-\boldsymbol{y} \Big\rangle \,\mathrm{d}\,\theta \\
&\geqslant J(\boldsymbol{y}) + \int_0^1 \Big\langle \nabla J(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y} \Big\rangle + \alpha\theta\|\boldsymbol{x}-\boldsymbol{y}\|^2 \,\mathrm{d}\,\theta \\
&= J(\boldsymbol{y}) + \Big\langle \nabla J(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y} \Big\rangle + \frac{\alpha}{2}\|\boldsymbol{x}-\boldsymbol{y}\|^2,
\end{aligned}
$$

which gives (8.4).

We have proved all the implications required to conclude the proof. □

8.2 Unconstrained optimization

Throughout this section $\mathcal{K} = \mathbf{R}^n$. We begin by establishing conditions under which the optimization problem (8.1) admits a unique solution in this setting. We first prove existence of a global minimizer under appropriate conditions.

Proposition 8.2 (Existence of a global minimizer)**.** *Suppose that $J\colon \mathbf{R}^n \to \mathbf{R}$ is continuous and coercive, the latter meaning that $J(\boldsymbol{x}) \to \infty$ when $\|\boldsymbol{x}\| \to \infty$. Then there exists a global minimizer of J in $\mathbf{R}^n$.*

Proof. Let $(\boldsymbol{x}_n)_{n\in\mathbf{N}}$ be a minimizing sequence of J, i.e. a sequence in $\mathbf{R}^n$ such that

$$
J(\boldsymbol{x}_n) \to \inf_{\boldsymbol{x}\in\mathbf{R}^n} J(\boldsymbol{x}) \quad \text{as} \quad n \to \infty.
$$

The sequence $(\boldsymbol{x}_n)$ is bounded, because otherwise it would hold that $J(\boldsymbol{x}_n) \to \infty$ by coercivity. Therefore, since closed bounded sets in $\mathbf{R}^n$ are compact, there is a subsequence $(\boldsymbol{x}_{n_k})_{k\in\mathbf{N}}$ converging to some $\boldsymbol{x}_* \in \mathbf{R}^n$. Since J is continuous, we have that

$$
J(\boldsymbol{x}_*) = \lim_{k\to\infty} J(\boldsymbol{x}_{n_k}) = \inf_{\boldsymbol{x}\in\mathbf{R}^n} J(\boldsymbol{x}).
$$

We conclude that $\boldsymbol{x}_*$ is a minimizer of J. □

Remark 8.2. We relied crucially in the proof of Proposition 8.2 on the fact that closed bounded sets in $\mathbf{R}^n$ are compact. In the infinite-dimensional setting, coercivity and continuity alone are not sufficient to guarantee the existence of a minimizer.

Uniqueness of the minimizer can be established under a strict convexity assumption.

Proposition 8.3 (Uniqueness of the minimizer)**.** *If J is strictly convex, then there exists at most one global minimizer.*

Proof. Suppose for contradiction that there were two minimizers $\boldsymbol{x}_*$ and $\boldsymbol{y}_*$. Then by strict convexity we have

$$
J\left(\frac{\boldsymbol{x}_* + \boldsymbol{y}_*}{2}\right) < \frac{1}{2}\big(J(\boldsymbol{x}_*) + J(\boldsymbol{y}_*)\big) = J(\boldsymbol{x}_*),
$$

which contradicts the minimality of $J(\boldsymbol{x}_*)$. □

Finally, before introducing the steepest descent algorithm, we recall the following standard result from analysis, the proof of which is left as an exercise.

Theorem 8.4 (Euler condition)**.** *Suppose that $J\colon \mathbf{R}^n \to \mathbf{R}$ is differentiable.*

- *If $\boldsymbol{x}_*$ is a local minimizer of J, then $\nabla J(\boldsymbol{x}_*) = 0$.*
- *If J is convex, then $\nabla J(\boldsymbol{x}_*) = 0$ if and only if $\boldsymbol{x}_*$ is a global minimizer.*

Steepest descent method. In this section, we study the more general version of the steepest descent with *fixed step* given in Algorithm 17.

Algorithm 17 Steepest descent method

1: Pick λ, and initial $\boldsymbol{x}_0$.
2: **for** $k \in \{0, 1, \dots\}$ **do**
3: $\quad \boldsymbol{d}_k \leftarrow \nabla J(\boldsymbol{x}_k)$
4: $\quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \lambda \boldsymbol{d}_k$
5: **end for**

Remark 8.3. We encountered the steepest descent with fixed step for a quadratic objective function when we analyzed Richardson's method for solving linear equations in Chapter 4.

In practice, Algorithm 17 must be supplemented with an appropriate stopping criterion. This could be, for example, a criterion of the form $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leqslant \varepsilon$, or $\left|J(\boldsymbol{x}_{k+1}) - J(\boldsymbol{x}_k)\right| \leqslant \varepsilon$. It is sometimes also useful to use a normalized criterion of the form $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leqslant \varepsilon \|\boldsymbol{x}_0\|$. The steepest descent method may be viewed as a fixed point iteration for the function

$$\boldsymbol{F}_\lambda(\boldsymbol{x}) = \boldsymbol{x} - \lambda \nabla J(\boldsymbol{x}). \tag{8.6}$$

A point $\boldsymbol{x}_* \in \mathbf{R}^n$ is a fixed point of this function if and only if $\boldsymbol{x}_*$ is a solution to the nonlinear equation $\nabla J(\boldsymbol{x}_*) = 0$. We shall now prove the convergence of the steepest descent under appropriate assumptions on the function J.

Theorem 8.5 (Convergence of the steepest descent method)**.** *Suppose that J is differentiable, strongly convex with parameter α, and that its gradient $\nabla J\colon \mathbf{R}^n \to \mathbf{R}^n$ is Lipschitz continuous with parameter L:*

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n, \qquad \|\nabla J(\boldsymbol{x}) - \nabla J(\boldsymbol{y})\| \leqslant L \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{8.7}$$

Then provided that

$$0 < \lambda < \frac{2\alpha}{L}, \tag{8.8}$$

the steepest descent method with fixed step is convergent. More precisely, there exists $\rho \in (0,1)$ *such that for all* $k \geqslant 0$

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\| \leqslant \rho^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|. \tag{8.9}$$

Proof. Under the assumptions of the theorem, there exists a unique global minimizer of J, which is the unique fixed point of $\boldsymbol{F}_\lambda$. We begin by proving that $\boldsymbol{F}_\lambda$ defined in (8.6) is globally Lipschitz continuous. We have

$$\begin{aligned}
\|\boldsymbol{F}_\lambda(\boldsymbol{x}) - \boldsymbol{F}_\lambda(\boldsymbol{y})\|^2 &= \Big\|\boldsymbol{x} - \boldsymbol{y} - \lambda\big(\nabla J(\boldsymbol{x}) - \nabla J(\boldsymbol{y})\big)\Big\|^2 \\
&= \|\boldsymbol{x} - \boldsymbol{y}\|^2 - 2\lambda\Big\langle \boldsymbol{x} - \boldsymbol{y}, \nabla J(\boldsymbol{x}) - \nabla J(\boldsymbol{y})\Big\rangle + \lambda^2 \|\nabla J(\boldsymbol{x}) - \nabla J(\boldsymbol{y})\|^2 \\
&\leqslant (1 - 2\alpha\lambda + \lambda^2 L)\|\boldsymbol{x} - \boldsymbol{y}\|^2,
\end{aligned}$$

where we employed (8.5) for the second term and (8.7) for the third term. Thus, $\boldsymbol{F}_\lambda$ is globally Lipschitz continuous with constant $\rho = \sqrt{1 - 2\alpha\lambda + \lambda^2 L}$, which is less than 1 if and only (8.8) is satisfied. The bound (8.9) then follows by noting that

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\| = \|\boldsymbol{F}_\lambda(\boldsymbol{x}_{k-1}) - \boldsymbol{F}_\lambda(\boldsymbol{x}_*)\| \leqslant \rho \|\boldsymbol{x}_{k-1} - \boldsymbol{x}_*\| \leqslant \ \ldots \ \leqslant \rho^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|,$$

which concludes the proof. (Note that (8.9) also follows from Theorem 5.2.) □

Remark 8.4 (Convergence speed). The choice of λ minimizing the Lipschitz constant ρ is given by $\lambda_* = \frac{\alpha}{L^2}$, which corresponds to $\rho_* = 1 - \left(\frac{\alpha}{L}\right)^2$. Often, in practice, it holds that $\alpha \ll L$, in which case the convergence of the steepest descent with fixed step is slow.

8.3 Constrained optimization

In this section, we assume that $\mathcal{K} \subset \mathbf{R}^n$. We begin by establishing well-posedness of the optimization problem (8.1) in this setting.

Proposition 8.6 (Well posedness of (8.1) in the constrained setting)**.** *The two items below concern existence and uniqueness, respectively.*

- *Suppose that* $\mathcal{K} \subset \mathbf{R}^n$ *is* closed *and that* $J\colon \mathcal{K} \to \mathbf{R}$ *is continuous and coercive. Then there exists a global minimizer of* J *in* $\mathcal{K}$*.*
- *Suppose that* $\mathcal{K} \subset \mathbf{R}^n$ *is* convex *and that* $J\colon \mathcal{K} \to \mathbf{R}$ *is strictly convex. Then there exists at most one global minimizer.*

Proof. The proof is very similar to those of Proposition 8.2 and Proposition 8.3, and so we leave it to the reader. Note that the set $\mathcal{K}$ must be closed to ensure existence, and convex to guarantee uniqueness. These assumptions are clearly satisfied when $\mathcal{K} = \mathbf{R}^n$, so Proposition 8.6 indeed generalizes Proposition 8.2 and Proposition 8.3. □

The following theorem, which generalizes Theorem 8.4, establishes a characterization of the minimizer when J is differentiable.

Theorem 8.7 (Euler–Lagrange conditions)**.** *Suppose that $J\colon \mathcal{K} \to \mathbf{R}$ is differentiable and that $\mathcal{K} \subset \mathbf{R}^n$ is closed and convex. Then the following statements hold.*

- *If $\boldsymbol{x}_*$ is a local minimizer of J, then*

$$\forall \boldsymbol{x} \in \mathcal{K}, \qquad \langle \nabla J(\boldsymbol{x}_*), \boldsymbol{x} - \boldsymbol{x}_* \rangle \geqslant 0. \tag{8.10}$$

- *Conversely, if (8.10) is satisfied and J is convex, then $\boldsymbol{x}_*$ is a global minimizer of J.*

Proof. Suppose that $\boldsymbol{x}_*$ is a local minimizer of J. This means that there exists $\delta > 0$ such that

$$\forall \boldsymbol{x} \in B_\delta(\boldsymbol{x}_*) \cap \mathcal{K}, \qquad J(\boldsymbol{x}_*) \leqslant J(\boldsymbol{x}).$$

Therefore $J(\boldsymbol{x}_*) \leqslant J\Big((1-t)\boldsymbol{x}_* + t\boldsymbol{x}\Big)$ for all $t \in [0,1]$ sufficiently small. But then

$$\langle \nabla J(\boldsymbol{x}_*), \boldsymbol{x} - \boldsymbol{x}_* \rangle = \lim_{t \to 0} \frac{J\Big((1-t)\boldsymbol{x}_* + t\boldsymbol{x}\Big) - J(\boldsymbol{x}_*)}{t} \geqslant 0.$$

Conversely, suppose that (8.10) is satisfied and that J is convex. Since J is convex, equation (8.4) holds with $\alpha = 0$, and applying this equation with $\boldsymbol{y} = \boldsymbol{x}_*$, we deduce that $\boldsymbol{x}_*$ is a global minimizer. □

The steepest descent Algorithm 17 can be extended to optimization problems with constraints by introducing an additional projection step. In order to precisely formulate the algorithm, we begin by introducing the projection operator $\Pi_{\mathcal{K}}$.

Proposition 8.8 (Projection on a closed convex set)**.** *Suppose that $\mathcal{K}$ is a closed convex subset of $\mathbf{R}^n$. Then for all $\boldsymbol{x} \in \mathbf{R}^n$ there a unique $\Pi_{\mathcal{K}}\boldsymbol{x} \in \mathcal{K}$, called the orthogonal projection of $\boldsymbol{x}$ onto $\mathcal{K}$, such that*

$$\|\Pi_{\mathcal{K}}\boldsymbol{x} - \boldsymbol{x}\| = \inf_{\boldsymbol{y} \in \mathcal{K}} \|\boldsymbol{y} - \boldsymbol{x}\|.$$

Proof. The functional $J_{\boldsymbol{x}}(\boldsymbol{y}) = \|\boldsymbol{y} - \boldsymbol{x}\|^2$ is strongly convex, and so Proposition 8.6 immediately implies the existence and uniqueness of $\Pi_{\mathcal{K}}\boldsymbol{x}$. □

Remark 8.5. In view of Theorem 8.7, the projection $\Pi_{\mathcal{K}}\boldsymbol{x}$ is the unique element of $\mathcal{K}$ which satisfies

$$\forall \boldsymbol{y} \in \mathcal{K}, \qquad \langle \Pi_{\mathcal{K}}\boldsymbol{x} - \boldsymbol{x}, \boldsymbol{y} - \Pi_{\mathcal{K}}\boldsymbol{x} \rangle \geqslant 0. \tag{8.11}$$

We are now ready to present the steepest descent method with projection: see Algorithm 18. Like Algorithm 17, the steepest descent with projection may be viewed as a fixed point iteration, this time for the function

$$\boldsymbol{F}_\lambda(\boldsymbol{x}) \coloneqq \Pi_{\mathcal{K}}\Big(\boldsymbol{x} - \lambda \nabla J(\boldsymbol{x})\Big). \tag{8.12}$$

We now prove the convergence of the method.

Algorithm 18 Steepest descent with projection

1: Pick λ, and initial $\boldsymbol{x}_0$.
2: **for** $k \in \{0, 1, \dots\}$ **do**
3: $\quad \boldsymbol{d}_k \leftarrow \nabla J(\boldsymbol{x}_k)$
4: $\quad \boldsymbol{x}_{k+1} \leftarrow \Pi_{\mathcal{K}}(\boldsymbol{x}_k - \lambda \boldsymbol{d}_k)$
5: **end for**

Theorem 8.9 (Convergence of steepest descent with projection)**.** *Suppose that J is differentiable, strongly convex with parameter α, and that its gradient $\nabla J\colon \mathbf{R}^n \to \mathbf{R}^n$ is Lipschitz continuous with parameter L. Assume also that $\mathcal{K} \subset \mathbf{R}^n$ is closed and convex. Then provided that*

$$0 < \lambda < \frac{2\alpha}{L}, \tag{8.13}$$

the steepest descent method with fixed step is convergent. More precisely, there exists $\rho \in (0,1)$ such that for all $k \geqslant 0$

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\| \leqslant \rho^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|.$$

Proof. Under the assumptions, there exists a unique global minimizer $\boldsymbol{x}_* \in \mathcal{K}$. We already showed in the proof of Theorem 8.5 that the mapping $\boldsymbol{x} \mapsto \boldsymbol{x} - \lambda \nabla J(\boldsymbol{x})$ is a contraction if and only if λ satisfies (8.13). In order to prove that $\boldsymbol{F}_\lambda$ given in (8.12) is a contraction under the same condition, it is sufficient to prove that $\Pi_{\mathcal{K}}\colon \mathbf{R}^n \to \mathcal{K}$ satisfies the following estimate:

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n, \qquad \|\Pi_{\mathcal{K}}\boldsymbol{x} - \Pi_{\mathcal{K}}\boldsymbol{y}\| \leqslant \|\boldsymbol{x} - \boldsymbol{y}\|.$$

To this end, take $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{R}^n \times \mathbf{R}^n$ and let $\boldsymbol{\delta} = \Pi_{\mathcal{K}}\boldsymbol{x} - \Pi_{\mathcal{K}}\boldsymbol{y}$. By (8.11), it holds that

$$\begin{aligned} \|\boldsymbol{\delta}\|^2 &= \langle \boldsymbol{\delta}, \Pi_{\mathcal{K}}\boldsymbol{x} - \boldsymbol{x}\rangle + \langle \boldsymbol{\delta}, \boldsymbol{x} - \boldsymbol{y}\rangle + \langle \boldsymbol{\delta}, \boldsymbol{y} - \Pi_{\mathcal{K}}\boldsymbol{y}\rangle \\ &\leqslant 0 + \langle \boldsymbol{\delta}, \boldsymbol{x} - \boldsymbol{y}\rangle + 0 \leqslant \|\boldsymbol{\delta}\|\|\boldsymbol{x} - \boldsymbol{y}\|, \end{aligned}$$

which yields the required inequality. Therefore $\boldsymbol{F}_\lambda$ in (8.12) is a contraction and so, by the Banach fixed point theorem, it admits a unique fixed point $\boldsymbol{y}_* \in \mathcal{K}$. To show that $\boldsymbol{y}_* = \boldsymbol{x}_*$, note that if $\boldsymbol{F}_\lambda(\boldsymbol{y}_*) = \boldsymbol{y}_*$, then by (8.11) it holds that

$$\forall \boldsymbol{y} \in \mathcal{K}, \qquad \langle \lambda \nabla J(\boldsymbol{y}_*), \boldsymbol{y} - \boldsymbol{y}_*\rangle \geqslant 0.$$

Therefore, using Theorem 8.7, we obtain that $\boldsymbol{y}_*$ is a global minimizer of J, so $\boldsymbol{y}_* = \boldsymbol{x}_*$. □

Remark 8.6. The applicability of Algorithm 18 is limited in practice, as computing $\Pi_{\mathcal{K}}(\boldsymbol{x})$ analytically is possible only in simple settings.

Appendix A

Background material

A.1 Inner products and norms 207
A.2 Completeness 210
A.3 Contraction mappings and the Banach fixed point theorem 211
A.4 Vector norms 212
A.5 Matrix norms 212
A.6 Diagonalization and spectral theorem 214
A.7 Similarity transformation and Jordan normal form 217
A.8 Oldenburger's theorem and Gelfand's formula 218

In this chapter, we collect basic results that are useful for this course.

A.1 Inner products and norms

We begin by recalling the definitions of the fundamental concepts of *norm* and *inner product*. For generality, we consider the case of a *complex* vector space, i.e. a vector space for which the scalar field is $\mathbf{C}$.

Definition A.1. A norm on a complex vector space $\mathcal{X}$ is a function $\|\bullet\| : \mathcal{X} \to \mathbf{R}$ satisfying the following axioms:

- **Positivity**: $\forall \boldsymbol{x} \in \mathcal{X} \backslash \{\mathbf{0}\}, \quad \|\boldsymbol{x}\| > 0.$
- **Homogeneity**: $\forall (c, \boldsymbol{x}) \in \mathbf{C} \times \mathcal{X}, \quad \|c\boldsymbol{x}\| = |c| \, \|\boldsymbol{x}\|.$
- **Triangular inequality**: $\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{X}, \quad \|\boldsymbol{x} + \boldsymbol{y}\| \leqslant \|\boldsymbol{x}\| + \|\boldsymbol{y}\|.$

For example, the Euclidean norm on $\mathbf{C}^n$ is given by

$$\|\boldsymbol{x}\| = \sqrt{|x_1|^2 + \ldots + |x_n|^2}.$$

Definition A.2. An inner product on a *complex* vector space $\mathcal{X}$ is a function

$$\langle \bullet, \bullet \rangle : \mathcal{X} \times \mathcal{X} \to \mathbf{C}$$

satisfying the following axioms:

- **Conjugate symmetry**: Here $\overline{\bullet}$ denotes the complex conjugate.

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{X}, \qquad \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \overline{\langle \boldsymbol{y}, \boldsymbol{x} \rangle}.$$

- **Linearity**: For all $(\alpha, \beta) \in \mathbf{C}^2$ and all $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in \mathcal{X}^3$, it holds that

$$\langle \alpha \boldsymbol{x} + \beta \boldsymbol{y}, \boldsymbol{z} \rangle = \alpha \langle \boldsymbol{x}, \boldsymbol{z} \rangle + \beta \langle \boldsymbol{y}, \boldsymbol{z} \rangle.$$

- **Positive-definiteness**:

$$\forall \boldsymbol{x} \in \mathcal{X} \backslash \{0\}, \qquad \langle \boldsymbol{x}, \boldsymbol{x} \rangle > 0.$$

For example, the familiar Euclidean inner product on $\mathbf{C}^n$ is given by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^{n} x_i \overline{y}_i.$$

A vector space with an inner product is called an *inner product space*. Any inner product on $\mathcal{X}$ induces a norm via the formula

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}. \tag{A.1}$$

The Cauchy–Schwarz inequality enables to bound inner products using norms. It is also useful for showing that the functional defined in (A.1) satisfies the triangle inequality, which is the goal of Exercise A.2.

Proposition A.1 (Cauchy–Schwarz inequality)**.** *Let $\mathcal{X}$ be an inner product space. Then*

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{X}, \qquad \left| \langle \boldsymbol{x}, \boldsymbol{y} \rangle \right| \leqslant \|\boldsymbol{x}\| \|\boldsymbol{y}\|. \tag{A.2}$$

Proof. The statement is obvious if $\boldsymbol{y} = \mathbf{0}$, so we assume in the rest of the proof that $\boldsymbol{y} \neq \mathbf{0}$. Let us define p: $\mathbf{R} \ni t \mapsto \|\boldsymbol{x} + t\boldsymbol{y}\|^2$. Using the bilinearity of the inner product, we have

$$p(t) = \|\boldsymbol{x}\|^2 + 2t \langle \boldsymbol{x}, \boldsymbol{y} \rangle + t^2 \|\boldsymbol{y}\|^2.$$

This shows that p is a convex second-order polynomial with a minimum at $t_* = -\langle \boldsymbol{x}, \boldsymbol{y} \rangle / \|\boldsymbol{y}\|^2$. Substituting this value in the expression of p, we obtain

$$p(t_*) = \|\boldsymbol{x}\|^2 - 2\frac{\left| \langle \boldsymbol{x}, \boldsymbol{y} \rangle \right|^2}{\|\boldsymbol{y}\|^2} + \frac{\left| \langle \boldsymbol{x}, \boldsymbol{y} \rangle \right|^2}{\|\boldsymbol{y}\|^2} = \|\boldsymbol{x}\|^2 - \frac{\left| \langle \boldsymbol{x}, \boldsymbol{y} \rangle \right|^2}{\|\boldsymbol{y}\|^2}.$$

Since $p(t_*) \geqslant 0$ by definition of p, we obtain (A.2). □

Several norms can be defined on the same vector space $\mathcal{X}$. Two norms $\|\bullet\|_\alpha$ and $\|\bullet\|_\beta$ on $\mathcal{X}$ are said to be equivalent if there exist positive real numbers c_ℓ and c_u such that

$$\forall \boldsymbol{x} \in \mathcal{X}, \qquad c_\ell \|\boldsymbol{x}\|_\alpha \leqslant \|\boldsymbol{x}\|_\beta \leqslant c_u \|\boldsymbol{x}\|_\alpha. \tag{A.3}$$

As the terminology indicates, norm equivalence is an *equivalence relation*. When working with norms on finite-dimensional vector spaces, it is important to keep in mind the following result. The proof is provided for information purposes only.

Proposition A.2. *Assume that $\mathcal{X}$ is a finite-dimensional vector space. Then all the norms defined on $\mathcal{X}$ are pairwise equivalent.*

Proof. Let $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ be a basis of $\mathcal{X}$. Any $\boldsymbol{x} \in \mathcal{X}$ admits a unique representation in this basis as $\boldsymbol{x} = \lambda_1 \boldsymbol{e}_1 + \cdots + \lambda_n \boldsymbol{e}_n$. We will show that any norm $\|\bullet\|$ on $\mathcal{X}$ is equivalent to the norm $\|\bullet\|_*$ given by

$$\|\boldsymbol{x}\|_* = |\lambda_1| + \ldots + |\lambda_n|. \tag{A.4}$$

By the triangle inequality, it holds that

$$\begin{aligned} \|\boldsymbol{x}\| \leqslant |\lambda_1| \|\boldsymbol{e}_1\| + \cdots + |\lambda_n| \|\boldsymbol{e}_n\| &\leqslant \Big(|\lambda_1| + \cdots + |\lambda_n|\Big) \max\Big\{\|\boldsymbol{e}_1\|, \ldots, \|\boldsymbol{e}_n\|\Big\} \\ &= \|\boldsymbol{x}\|_* \max\Big\{\|\boldsymbol{e}_1\|, \ldots, \|\boldsymbol{e}_n\|\Big\}. \end{aligned} \tag{A.5}$$

It remains to show that there exists a positive constant ℓ such that

$$\forall \boldsymbol{x} \in \mathcal{X}, \qquad \|\boldsymbol{x}\| \geqslant \ell\Big(|\lambda_1| + \cdots + |\lambda_n|\Big). \tag{A.6}$$

To this end, we reason by contradiction. If this inequality were not true, then there would exist a sequence $(\boldsymbol{x}^{(i)})_{i \in \mathbf{N}}$ such that $\|\boldsymbol{x}^{(i)}\| \to 0$ as $i \to \infty$ and $\|\boldsymbol{x}^{(i)}\|_* = 1$ for all $i \in \mathbf{N}$. Since $\lambda_1^{(i)} \in [-1, 1]$ for all $i \in \mathbf{N}$, we can extract a subsequence, still denoted by $(\boldsymbol{x}^{(i)})_{i \in \mathbf{N}}$ for simplicity, such that the corresponding coefficient $\lambda_1^{(i)}$ satisfies $\lambda_1^{(i)} \to \lambda_1^* \in [-1, 1]$, by compactness of the interval $[-1, 1]$. Repeating this procedure for $\lambda_2, \lambda_3, \ldots$, taking a new subsequence every time, we obtain a subsequence $(\boldsymbol{x}^{(i)})_{i \in \mathbf{N}}$ such that $\lambda_j^{(i)} \to \lambda_j^*$ in the limit as $i \to \infty$, for all $j \in \{1, \ldots, n\}$. Therefore, it holds that $\boldsymbol{x}^{(i)} \to \boldsymbol{x}^* := \lambda_1^* \boldsymbol{e}_1 + \cdots \lambda_n^* \boldsymbol{e}_n$ in the $\|\bullet\|_*$ norm, and thus also in the $\|\bullet\|$ norm by (A.5). Since $\boldsymbol{x}^{(i)} \to \boldsymbol{0}$ in the latter norm by assumption, we deduce that $\boldsymbol{x}^* = \boldsymbol{0}$. But the vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ are linearly independent, and so this implies that $\lambda_1^* = \ldots = \lambda_n^* = 0$, which is a contradiction because we also have that

$$|\lambda_1^*| + \cdots + |\lambda_n^*| = \lim_{i \to \infty} \big|\lambda_1^{(i)}\big| + \cdots + \big|\lambda_n^{(i)}\big| = 1.$$

This concludes the proof of (A.6). □

Exercise A.1. *Show that $\|\bullet\|_*\colon \mathcal{X} \to \mathbf{R}$ defined in (A.4) is indeed a norm.*

Exercise A.2. *Using Proposition A.1, show that the function $\|\bullet\|$ defined by (A.1) satisfies the triangle inequality.*

A.2 Completeness

Assume that $\mathcal{X}$ is a vector space with a norm $\|\bullet\|$. Together, $(\mathcal{X}, \|\bullet\|)$ form a *normed vector space*. A sequence $(\boldsymbol{x}_n)_{n \geqslant 0}$ in $\mathcal{X}$ is convergent in this space if there exists $\boldsymbol{x}_* \in \mathcal{X}$ such that

$$\|\boldsymbol{x}_n - \boldsymbol{x}_*\| \to 0 \quad \text{in the limit} \quad n \to \infty.$$

In this case, we write $\lim_{n\to\infty} \boldsymbol{x}_n = \boldsymbol{x}_*$ or $\boldsymbol{x}_n \to \boldsymbol{x}_*$.

Definition A.3 (Cauchy sequence)**.** A sequence $(\boldsymbol{x}_n)_{n \geqslant 0}$ in $\mathcal{X}$ is said to be Cauchy if

$$\lim_{m,n\to+\infty} \|\boldsymbol{x}_n - \boldsymbol{x}_m\| = 0.$$

The normed vector space $(\mathcal{X}, \|\bullet\|)$ is called *complete* if every Cauchy sequence is convergent.

Every convergent sequence is Cauchy, but the converse is not always true.

Example A.1. Consider the case where $\mathcal{X} = C([-1, 1])$, the space of continuous functions from $[-1, 1]$ to $\mathbf{R}$, endowed with the norm

$$\|f\| = \int_{-1}^{1} |f(x)| \, \mathrm{d}\, x.$$

The sequence of functions $(f_n)_{n \geqslant 0}$ in $\mathcal{X}$ given by

$$f_n(x) = x^{\frac{1}{2n+1}} \tag{A.7}$$

is Cauchy but not convergent. Indeed, assume for contradiction that there existed $f_* \in \mathcal{X}$ such that

$$\|f_n - f_*\| \xrightarrow[n\to\infty]{} 0. \tag{A.8}$$

Then f_* necessarily coincides with sign function:

$$\operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

However, this function is discontinuous, which contradicts the statement that $f_* \in \mathcal{X}$.

In this course, all the vector spaces encountered are complete. For example,

- $\mathbf{R}^n$ with any vector norm is complete;
- $\mathbf{C}^{m\times n}$ with any matrix norm is complete.

In order to show that a sequence is convergent in a complete normed vector space, it is sufficient to show that the sequence is Cauchy. This approach is used in Lemma 4.2 and Theorem A.3.

Exercise A.3. *Prove that every convergent sequence is Cauchy.*

A.3 Contraction mappings and the Banach fixed point theorem

Let $(\mathcal{X}, \|\bullet\|)$ denote a normed vector space. A map $\phi\colon \mathcal{X} \to \mathcal{X}$ is called a contraction mapping if there is a constant $L \in (0,1)$ such that

$$\forall (x,y) \in \mathcal{X} \times \mathcal{X}, \qquad \|\phi(x) - \phi(y)\| \leqslant L\|x - y\|.$$

The importance of contraction mappings in this course stems from the following theorem.

Theorem A.3 (Banach fixed point theorem)**.** *Let $(\mathcal{X}, \|\bullet\|)$ be a complete normed space, and let $\phi\colon \mathcal{X} \to \mathcal{X}$ be a contraction mapping. Then ϕ has a unique fixed point in $\mathcal{X}$.*

Proof. We prove first existence and then uniqueness.
Existence. Take $x_0 \in \mathcal{X}$, and define the sequence $(x_k)_{k\in \mathbf{N}}$ inductively by

$$x_{k+1} = \phi(x_k). \tag{A.9}$$

It holds that

$$\big\|x_{k+1} - x_k\big\| = \big\|\phi(x_k) - \phi(x_{k-1})\big\| \leqslant L\|x_k - x_{k-1}\| \leqslant \ \dots \ \leqslant L^k\|x_1 - x_0\|.$$

Therefore, for any $n \geqslant m$, we have by the triangle inequality

$$\begin{aligned}\|x_n - x_m\| &\leqslant \|x_n - x_{n-1}\| + \cdots + \|x_{m+1} - x_m\| \\ &\leqslant (L^{n-1} + \cdots + L^m)\|x_1 - x_0\| \\ &\leqslant L^m(1 + L + L^2 + \cdots)\|x_1 - x_0\| = \frac{L^m}{1-L}\|x_1 - x_0\|.\end{aligned}$$

It follows that the sequence $(x_k)_{k\in \mathbf{N}}$ is Cauchy in $\mathcal{X}$, implying by completeness that $x_k \to x_*$ in the limit as $k \to \infty$, for some limit $x_* \in \mathcal{X}$. Being a contraction, the mapping ϕ is continuous, and so taking the limit $k \to \infty$ in (A.9), we obtain that

$$x_* = \lim_{k\to\infty} x_{k+1} = \lim_{k\to\infty} \phi(x_k) = \phi\Big(\lim_{k\to\infty} x_k\Big) = \phi(x_*).$$

In other words, x_* is a fixed point of ϕ.

Uniqueness. Assume that $y_* \in \mathcal{X}$ is a fixed point. Then,

$$\big\|y_* - x_*\big\| = \big\|\phi(y_*) - \phi(x_*)\big\| \leqslant L\|y_* - x_*\|,$$

which implies that $y_* = x_*$ since $L < 1$. □

Remark A.1. The Banach fixed point theorem holds also in complete metric spaces.

A.4 Vector norms

In the vector space $\mathbf{C}^n$, the most commonly used norms are particular cases of the p-norm, also called Hölder norm.

Definition A.4. Given $p \in [1, \infty]$, the p-norm of a vector $\boldsymbol{x} \in \mathbf{C}^n$ is defined as follows:

$$\|\boldsymbol{x}\|_p := \begin{cases} \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}} & \text{if } \ p < \infty, \\ \max\Big\{|x_1|, \ldots, |x_n|\Big\} & \text{if } \ p = \infty. \end{cases}$$

The values of p most commonly encountered in applications are 1, 2 and ∞. The 1-norm is sometimes called the *taxicab* or *Manhattan* norm, and the 2-norm is usually called the *Euclidean norm*. The explicit expressions of these norms are

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|, \qquad \|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}.$$

Notice that the infinity norm $\|\bullet\|_\infty$ may be defined as the limit of the p-norm as $p \to \infty$:

$$\|\boldsymbol{x}\|_\infty := \lim_{p \to \infty} \|\boldsymbol{x}\|_p.$$

In the rest of this chapter, the notations $\langle \bullet, \bullet \rangle$ and $\|\bullet\|$ without subscript always refer to the Euclidean inner product (A.1) and induced norm, unless specified otherwise.

A.5 Matrix norms

Given two norms $\|\bullet\|_\alpha$ and $\|\bullet\|_\beta$ on $\mathbf{C}^m$ and $\mathbf{C}^n$, respectively, we define the *operator norm* induced by $\|\bullet\|_\alpha$ and $\|\bullet\|_\beta$ of the matrix A as

$$\|\mathsf{A}\|_{\alpha,\beta} = \sup\Big\{\|\mathsf{A}\boldsymbol{x}\|_\alpha : \boldsymbol{x} \in \mathbf{C}^n, \|\boldsymbol{x}\|_\beta \leqslant 1\Big\}. \tag{A.10}$$

The term *operator norm* is motivated by the fact that, to any matrix $\mathsf{A} \in \mathbf{C}^{m \times n}$, there naturally corresponds the linear operator from $\mathbf{C}^n$ to $\mathbf{C}^m$ with action $\boldsymbol{x} \mapsto \mathsf{A}\boldsymbol{x}$. Matrix norms of the type (A.10) are also called *subordinate* matrix norms. An immediate corollary of the definition (A.10) is that, for all $\boldsymbol{x} \in \mathbf{C}^n$,

$$\|\mathsf{A}\boldsymbol{x}\|_\alpha = \|\mathsf{A}\hat{\boldsymbol{x}}\|_\alpha \|\boldsymbol{x}\|_\beta \leqslant \sup\Big\{\|\mathsf{A}\boldsymbol{y}\|_\alpha : \|\boldsymbol{y}\|_\beta \leqslant 1\Big\}\|\boldsymbol{x}\|_\beta = \|\mathsf{A}\|_{\alpha,\beta}\|\boldsymbol{x}\|_\beta, \qquad \hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_\beta}. \tag{A.11}$$

Exercise A.4. *Show that equation (A.10) defines a norm on* $\mathbf{C}^{m \times n}$.

The matrix p-norm is defined as the operator norm (A.10) in the particular case where $\|\bullet\|_\alpha$ and $\|\bullet\|_\beta$ are both Hölder norms with the same value of p.

Definition A.5. Given $p \in [1, \infty]$, the p-norm of a matrix $\mathsf{A} \in \mathbf{C}^{m\times n}$ is given by

$$\|\mathsf{A}\|_p := \sup\Big\{\|\mathsf{A}\boldsymbol{x}\|_p : \boldsymbol{x} \in \mathbf{C}^n, \|\boldsymbol{x}\|_p \leqslant 1\Big\}. \tag{A.12}$$

Not all matrix norms are induced by vector norms. For example, the Frobenius norm, which is widely used in applications, is not induced by a vector norm. It is, however, induced by an inner product on $\mathbf{C}^{m\times n}$.

Definition A.6. The Frobenius norm of $\mathsf{A} \in \mathbf{C}^{m\times n}$ is given by

$$\|\mathsf{A}\|_{\mathrm{F}} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{\frac{1}{2}}. \tag{A.13}$$

A matrix norm $\|\bullet\|$ is said to be submultiplicative if, for any two matrices $\mathsf{A} \in \mathbf{C}^{m\times n}$ and $\mathsf{B} \in \mathbf{C}^{n\times \ell}$, it holds that

$$\|\mathsf{AB}\| \leqslant \|\mathsf{A}\|\|\mathsf{B}\|.$$

All subordinate matrix norms, for example the p-norms, are submultiplicative, and so is the Frobenius norm.

Exercise A.5. *Write down the inner product on $\mathbf{C}^{m\times n}$ corresponding to (A.13).*

Exercise A.6. *Show that the matrix p-norm is submultiplicative.*

A.6 Diagonalization and spectral theorem

Definition A.7. A square matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ is said to be diagonalizable if there exists an invertible matrix $\mathsf{P} \in \mathbf{C}^{n\times n}$ and a diagonal matrix $\mathsf{D} \in \mathbf{C}^{n\times n}$ such that

$$\mathsf{AP} = \mathsf{PD}. \tag{A.14}$$

In this case, the diagonal elements of D are called the eigenvalues of A, and the columns of P are called the eigenvectors of A.

Denoting by $\boldsymbol{e}_i$ the i-th column of P and by λ_i the i-th diagonal element of D, we have by (A.14) that $\mathsf{A}\boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$ or, equivalently, $(\mathsf{A} - \lambda_i \mathsf{I}_n)\boldsymbol{e}_i = \mathbf{0}$. Here I_n is the $\mathbf{C}^{n\times n}$ identity matrix. Therefore, a complex number λ is an eigenvalue of A if and only if $\det(\mathsf{A} - \lambda \mathsf{I}_n) = 0$. In other words, the eigenvalues of A are the roots of $\det(\mathsf{A} - \lambda \mathsf{I}_n)$, which is called the *characteristic polynomial*.

Symmetric matrices and spectral theorem

The transpose of a matrix $\mathsf{A} \in \mathbf{C}^{m\times n}$ is denoted by $\mathsf{A}^T \in \mathbf{C}^{n\times m}$ and defined as the matrix with entries $a^T_{ij} = a_{ji}$. The conjugate transpose of A, denoted by A^*, is the matrix obtained by taking the transpose and taking the complex conjugate of all the entries. A real matrix that is equal to its transpose is necessarily square and called *symmetric*, and a complex matrix that is equal to its conjugate transpose is called *Hermitian*. Hermitian matrices, of which real symmetric matrices are a subset, enjoy many nice properties, the main one being that they are diagonalizable with a matrix Q that is unitary, i.e. such that $\mathsf{Q}^{-1} = \mathsf{Q}^*$. This is the content of the *spectral theorem*, a pillar of linear algebra with important generalizations to infinite-dimensional operators.

Theorem A.4 (Spectral theorem for Hermitian matrices)**.** *If $\mathsf{A} \in \mathbf{C}^{n\times n}$ is Hermitian, then there exists an unitary matrix $\mathsf{Q} \in \mathbf{C}^{n\times n}$ and a diagonal matrix $\mathsf{D} \in \mathbf{R}^{n\times n}$ such that*

$$\mathsf{AQ} = \mathsf{QD}.$$

Sketch of the proof. The result is trivial for $n = 1$. Reasoning by induction, we assume that the result is true for Hermitian matrices in $\mathbf{C}^{n-1\times n-1}$ and prove that it then also holds for $\mathsf{A} \in \mathbf{C}^{n\times n}$.

Step 1. Existence of a real eigenvalue. By the fundamental theorem of algebra, there exists at least one solution $\lambda_1 \in \mathbf{C}$ to the equation $\det(\mathsf{A} - \lambda \mathsf{I}_n) = 0$, to which there corresponds at least one solution $\boldsymbol{q}_1 \in \mathbf{C}^n$ of norm 1 to the equation $(\mathsf{A} - \lambda_1 \mathsf{I}_n)\boldsymbol{q}_1 = \mathbf{0}$. The eigenvalue λ_1 is necessarily real because

$$\lambda_1 \langle \boldsymbol{q}_1, \boldsymbol{q}_1 \rangle = \langle \lambda_1 \boldsymbol{q}_1, \boldsymbol{q}_1 \rangle = \langle \mathsf{A}\boldsymbol{q}_1, \boldsymbol{q}_1 \rangle = \langle \boldsymbol{q}_1, \mathsf{A}\boldsymbol{q}_1 \rangle = \langle \boldsymbol{q}_1, \lambda_1 \boldsymbol{q}_1 \rangle = \overline{\lambda}_1 \langle \boldsymbol{q}_1, \boldsymbol{q}_1 \rangle.$$

Step 2. Using the induction hypothesis. Next, take an orthonormal basis $(\boldsymbol{e}_2, \ldots, \boldsymbol{e}_n)$ of the orthogonal complement $\mathrm{Span}\{\boldsymbol{q}_1\}^\perp$ and construct the unitary matrix

$$V = (\boldsymbol{q}_1 \quad \boldsymbol{e}_2 \quad \dots \quad \boldsymbol{e}_n),$$

i.e. the matrix with columns $\boldsymbol{q}_1$, $\boldsymbol{e}_2$, etc. A calculation gives,

$$\mathsf{V}^*\mathsf{A}\mathsf{V} = \begin{pmatrix} \langle \boldsymbol{q}_1, \mathsf{A}\boldsymbol{q}_1\rangle & \langle \boldsymbol{q}_1, \mathsf{A}\boldsymbol{e}_2\rangle & \dots & \langle \boldsymbol{q}_1, \mathsf{A}\boldsymbol{e}_n\rangle \\ \langle \boldsymbol{e}_2, \mathsf{A}\boldsymbol{q}_1\rangle & \langle \boldsymbol{e}_2, \mathsf{A}\boldsymbol{e}_2\rangle & \dots & \langle \boldsymbol{e}_2, \mathsf{A}\boldsymbol{e}_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{e}_n, \mathsf{A}\boldsymbol{q}_1\rangle & \langle \boldsymbol{e}_n, \mathsf{A}\boldsymbol{e}_2\rangle & \dots & \langle \boldsymbol{e}_n, \mathsf{A}\boldsymbol{e}_n\rangle \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \langle \boldsymbol{e}_2, \mathsf{A}\boldsymbol{e}_2\rangle & \dots & \langle \boldsymbol{e}_2, \mathsf{A}\boldsymbol{e}_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \langle \boldsymbol{e}_n, \mathsf{A}\boldsymbol{e}_2\rangle & \dots & \langle \boldsymbol{e}_n, \mathsf{A}\boldsymbol{e}_n\rangle \end{pmatrix}.$$

Let us denote the $n-1 \times n-1$ lower right block of this matrix by V_{n-1}. This is a Hermitian matrix of size $n-1$ so, using the induction hypothesis, we deduce that $\mathsf{V}_{n-1} = \mathsf{Q}_{n-1}\mathsf{D}_{n-1}\mathsf{Q}_{n-1}^*$ for appropriate matrices $\mathsf{Q}_{n-1} \in \mathbf{C}^{n-1\times n-1}$ and $\mathsf{D}_{n-1} \in \mathbf{R}^{n-1\times n-1}$ which are unitary and diagonal, respectively.

Step 3. Constructing Q and D. Define now

$$\mathsf{Q} = \mathsf{V}\begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{Q}_{n-1} \end{pmatrix}.$$

It is not difficult to verify that Q is a unitary matrix, and we have

$$\mathsf{Q}^*\mathsf{A}\mathsf{Q} = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{Q}_{n-1}^* \end{pmatrix}\mathsf{V}^*\mathsf{A}\mathsf{V}\begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{Q}_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{Q}_{n-1}^* \end{pmatrix}\begin{pmatrix} \lambda_1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{V}_{n-1} \end{pmatrix}\begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{Q}_{n-1} \end{pmatrix}.$$

Developing the last expression, we obtain

$$\mathsf{Q}^*\mathsf{A}\mathsf{Q} = \begin{pmatrix} \lambda_1 & \mathbf{0}^T \\ \mathbf{0} & \mathsf{D}_{n-1} \end{pmatrix},$$

which concludes the proof. □

We deduce, as a corollary of the spectral theorem, that if $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are eigenvectors of a Hermitian matrix associated with different eigenvalues, then they are necessarily orthogonal for the Euclidean inner product. Indeed, since $\mathsf{A} = \mathsf{A}^*$ and the eigenvalues are real, it holds that

$$\begin{aligned}(\lambda_1 - \lambda_2)\langle \boldsymbol{e}_1, \boldsymbol{e}_2\rangle &= \langle \lambda_1\boldsymbol{e}_1, \boldsymbol{e}_2\rangle - \langle \boldsymbol{e}_1, \overline{\lambda}_2\boldsymbol{e}_2\rangle \\ &= \langle \mathsf{A}\boldsymbol{e}_1, \boldsymbol{e}_2\rangle - \langle \boldsymbol{e}_1, \mathsf{A}\boldsymbol{e}_2\rangle = \langle \mathsf{A}\boldsymbol{e}_1, \boldsymbol{e}_2\rangle - \langle \mathsf{A}^*\boldsymbol{e}_1, \boldsymbol{e}_2\rangle = 0.\end{aligned}$$

The largest eigenvalue of a matrix, in modulus, is called the *spectral radius* and denoted by ρ. The following result relates the 2-norm of a matrix to the spectral radius of $\mathsf{A}\mathsf{A}^*$.

Proposition A.5. *It holds that* $\|\mathsf{A}\|_2 = \sqrt{\rho(\mathsf{A}^*\mathsf{A})}$.

Proof. Since $\mathsf{A}^*\mathsf{A}$ is Hermitian, it holds by the spectral theorem that $\mathsf{A}^*\mathsf{A} = \mathsf{Q}\mathsf{D}\mathsf{Q}^*$ for some unitary matrix Q and real diagonal matrix D. Therefore, denoting by $(\mu_i)_{1 \leqslant i \leqslant n}$ the (positive) diagonal elements of D and introducing $\boldsymbol{y} \coloneqq \mathsf{Q}^*\boldsymbol{x}$, we have

$$\begin{aligned}\|\mathsf{A}\boldsymbol{x}\| &= \sqrt{\boldsymbol{x}^*\mathsf{A}^*\mathsf{A}\boldsymbol{x}} = \sqrt{\boldsymbol{x}^*\mathsf{Q}\mathsf{D}\mathsf{Q}^*\boldsymbol{x}} \\ &= \sqrt{\sum_{i=1}^{n}\mu_i y_i^2} \leqslant \sqrt{\rho(\mathsf{A}^*\mathsf{A})}\sqrt{\sum_{i=1}^{n} y_i^2} = \sqrt{\rho(\mathsf{A}^*\mathsf{A})}\|\boldsymbol{x}\|, \end{aligned} \tag{A.15}$$

where we used in the last equality the fact that $\boldsymbol{y}$ has the same norm as $\boldsymbol{x}$, because Q is unitary. It follows from (A.15) that $\|\mathsf{A}\| \leqslant \sqrt{\rho(\mathsf{A}^*\mathsf{A})}$, and the converse inequality also holds true since $\|\mathsf{A}\boldsymbol{x}\| = \sqrt{\rho(\mathsf{A}^*\mathsf{A})}\|\boldsymbol{x}\|$ if $\boldsymbol{x}$ is the eigenvector of $\mathsf{A}^*\mathsf{A}$ corresponding to an eigenvalue of modulus $\rho(\mathsf{A}^*\mathsf{A})$. □

To conclude this section, we recall and prove the Courant–Fisher theorem.

Theorem A.6 (Courant–Fisher Min-Max theorem)**.** *The eigenvalues $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$ of a Hermitian matrix are characterized by the relation*

$$\lambda_k = \max_{\mathcal{S},\, \dim(\mathcal{S})=k} \left(\min_{\boldsymbol{x}\in\mathcal{S}\setminus\{0\}} \frac{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} \right). \tag{A.16}$$

Proof. Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be normalized and pairwise orthogonal eigenvectors associated with the eigenvalues $\lambda_1, \ldots, \lambda_n$, and let $\mathcal{S}_k = \mathrm{Span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$. Any $\boldsymbol{x} \in \mathcal{S}_k$ may be expressed as a linear combination $\boldsymbol{x} = \alpha_1\boldsymbol{v}_1 + \cdots + \alpha_k\boldsymbol{v}_k$, and so

$$\forall \boldsymbol{x} \in \mathcal{S}_k, \qquad \frac{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} = \frac{\sum_{i=1}^{k}\lambda_i|\alpha_i|^2}{\sum_{i=1}^{k}|\alpha_i|^2} \geqslant \lambda_k.$$

Therefore, it holds that

$$\min_{\boldsymbol{x}\in\mathcal{S}_k\setminus\{0\}} \frac{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} \geqslant \lambda_k,$$

which proves the $\geqslant$ direction of (A.16). For the $\leqslant$ direction, let $\mathcal{U}_k = \mathrm{Span}\{\boldsymbol{v}_k, \ldots \boldsymbol{v}_n\}$. Using a well-known result from linear algebra, we calculate that, for any subspace $\mathcal{S} \subset \mathbf{C}^n$ of dimension k,

$$\begin{aligned}\dim(\mathcal{S} \cap \mathcal{U}_k) &= \dim(\mathcal{S}) + \dim(\mathcal{U}_k) - \dim(\mathcal{S} + \mathcal{U}_k) \\ &\geqslant k + (n - k + 1) - n = 1.\end{aligned}$$

Therefore, any $\mathcal{S} \subset \mathbf{C}^n$ of dimension k has a nonzero intersection with $\mathcal{U}_k$. But since any vector in $\mathcal{U}_k$ can be expanded as $\beta_1\boldsymbol{v}_k + \cdots + \beta_n\boldsymbol{v}_n$, we have

$$\forall \boldsymbol{x} \in \mathcal{U}_k, \qquad \frac{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} = \frac{\sum_{i=k}^{n}\lambda_i|\alpha_i|^2}{\sum_{i=k}^{n}|\alpha_i|^2} \leqslant \lambda_k.$$

This shows that

$$\forall \mathcal{S} \subset \mathbf{C}^n \;\text{ with }\; \dim(\mathcal{S}) = k, \qquad \min_{\boldsymbol{x}\in\mathcal{S}\setminus\{0\}} \frac{\boldsymbol{x}^*\mathsf{A}\boldsymbol{x}}{\boldsymbol{x}^*\boldsymbol{x}} \leqslant \lambda_k,$$

which enables to conclude the proof. □

Exercise A.7. *Prove that if* $\mathsf{A} \in \mathbf{R}^{n\times n}$ *is diagonalizable as in* (A.14)*, then* $\mathsf{A}^n = \mathsf{P}\mathsf{D}^n\mathsf{P}^{-1}$.

A.7 Similarity transformation and Jordan normal form

In this section, we work with matrices in $\mathbf{C}^{n\times n}$. A *similarity transformation* is a mapping of the type $\mathbf{C}^{n\times n} \ni \mathsf{A} \mapsto \mathsf{P}^{-1}\mathsf{A}\mathsf{P} \in \mathbf{C}^{n\times n}$, where $\mathsf{P} \in \mathbf{C}^{n\times n}$ is a nonsingular matrix. If two matrices are related by a similarity transformation, then they are called *similar*, because they may be viewed as two representations of the same linear mapping in different bases.

Definition A.8 (Jordan block)**.** A Jordan block with dimension n is a matrix of the form

$$\mathsf{J}_n(\lambda) = \begin{pmatrix} \lambda & 1 & & & \\ & \lambda & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda & 1 \\ & & & & \lambda \end{pmatrix}$$

The parameter $\lambda \in \mathbf{C}$ is called the eigenvalue of the Jordan block.

A Jordan block is diagonalizable if and only if it is of dimension 1. The only eigenvector of a Jordan block is $\begin{pmatrix}1 & 0 & \dots & 0\end{pmatrix}^T$. The power of a Jordan block admits an explicit expression.

Lemma A.7. *It holds that*

$$\mathsf{J}_n(\lambda)^k = \begin{pmatrix} \lambda^k & \binom{k}{1}\lambda^{k-1} & \binom{k}{2}\lambda^{k-2} & \dots & \dots & \binom{k}{n-1}\lambda^{k-n+1} \\ & \lambda^k & \binom{k}{1}\lambda^{k-1} & \dots & \dots & \binom{k}{n-2}\lambda^{k-n+2} \\ & & \ddots & \ddots & & \vdots \\ & & & \ddots & \ddots & \vdots \\ & & & & \lambda^k & \binom{k}{1}\lambda^{k-1} \\ & & & & & \lambda^k \end{pmatrix}. \tag{A.17}$$

Proof. The explicit expression of the Jordan block can be obtained by decomposing the block as $\mathsf{J}_n(\lambda) = \lambda\mathsf{I} + \mathsf{N}$ and using the binomial formula:

$$(\lambda\mathsf{I} + \mathsf{N})^k = \sum_{i=1}^{k} \binom{k}{i} (\lambda\mathsf{I})^{k-i}\mathsf{N}^i.$$

To conclude the proof, we use the fact that N^i is a matrix with zeros everywhere except for i-th super-diagonal, which contains only ones. Moreover $\mathsf{N}^i = \mathsf{0}_{n\times n}$ if $i \geqslant n$. □

A matrix is said to be of *Jordan normal form* if it is block-diagonal with Jordan blocks on the diagonal. In other words, a matrix $\mathsf{J} \in \mathbf{C}^{n\times n}$ is of Jordan normal form if

$$\mathsf{J} = \begin{pmatrix} J_{n_1}(\lambda_1) & & & & \\ & J_{n_2}(\lambda_2) & & & \\ & & \ddots & & \\ & & & J_{n_{k-1}}(\lambda_{k-1}) & \\ & & & & J_{n_k}(\lambda_k) \end{pmatrix}$$

with $n_1 + \cdots + n_k = n$. Note that $\lambda_1, \ldots, \lambda_k$ are the eigenvalues of A. We state without proof the following important result.

Proposition A.8 (Jordan normal form)**.** *Any matrix $\mathsf{A} \in \mathbf{C}^{n\times n}$ is* similar *to a matrix in Jordan normal form. In other words, there exists an invertible matrix $\mathsf{P} \in \mathbf{C}^{n\times n}$ and a matrix in normal Jordan form $\mathsf{J} \in \mathbf{C}^{n\times n}$ such that*

$$\mathsf{A} = \mathsf{P}\mathsf{J}\mathsf{P}^{-1}$$

A.8 Oldenburger's theorem and Gelfand's formula

The following result establishes a necessary and sufficient condition for the convergence of $\|\mathsf{A}^k\|$ to 0 in terms of the spectral radius of A, and for any matrix norm $\|\bullet\|$.

Proposition A.9 (Oldenburger)**.** *Let $\rho(\mathsf{A})$ denote the spectral radius of $\mathsf{A} \in \mathbf{C}^{n\times n}$ and $\|\bullet\|$ be a matrix norm. Then*

- *$\|\mathsf{A}^k\| \to 0$ in the limit as $k \to \infty$ if and only if $\rho(\mathsf{A}) < 1$.*
- *$\|\mathsf{A}^k\| \to \infty$ in the limit as $k \to \infty$ if and only if $\rho(\mathsf{A}) > 1$.*

Proof. Since all matrix norms are equivalent, we can assume without loss of generality that $\|\bullet\|$ is the 2-norm. We prove only the equivalence $\|\mathsf{A}^k\| \to 0 \Leftrightarrow \rho(\mathsf{A}) < 1$. The other statement can be proved similarly. By Proposition A.8, there exists a nonsingular matrix P such that $\mathsf{A} = \mathsf{P}\mathsf{J}\mathsf{P}^{-1}$, for a matrix $\mathsf{J} \in \mathbf{C}^{n\times n}$ which is in normal Jordan form. Since $\rho(\mathsf{A}) = \rho(\mathsf{J})$ and $\|\mathsf{A}^k\| \to 0$ if and only if $\|\mathsf{J}^k\| \to 0$, it is sufficient to show that $\|\mathsf{J}^k\| \to 0 \Leftrightarrow \rho(\mathsf{J}) < 1$. The latter statement follows from the expression of the power of a Jordan block given in Lemma A.7. □

With this result, we can prove Gelfand's formula, which relates the spectral radius to the asymptotic growth of $\|\mathsf{A}^k\|$, and is used in Chapter 4.

Proposition A.10 (Gelfand's formula)**.** *Let $\mathsf{A} \in \mathbf{C}^{n\times n}$. It holds for any norm that*

$$\lim_{k\to\infty} \|\mathsf{A}^k\|^{\frac{1}{k}} = \rho(\mathsf{A})$$

Proof. Let $0 < \varepsilon < \rho(\mathsf{A})$ and define $\mathsf{A}^+ = \frac{\mathsf{A}}{\rho(\mathsf{A})+\varepsilon}$ and $\mathsf{A}^- = \frac{\mathsf{A}}{\rho(\mathsf{A})-\varepsilon}$. It holds by construction that $\rho(\mathsf{A}^+) < 1$ and $\rho(\mathsf{A}^-) > 1$. Using Proposition A.9, we deduce that

$$\lim_{k\to\infty} \|(\mathsf{A}^+)^k\| = 0, \qquad \lim_{k\to\infty} \|(\mathsf{A}^-)^k\| = \infty.$$

Therefore, it holds that

$$\limsup_{k\to\infty} \left\|(\mathsf{A}^+)^k\right\|^{\frac{1}{k}} \leqslant 1, \qquad \liminf_{k\to\infty} \left\|(\mathsf{A}^-)^k\right\|^{\frac{1}{k}} \geqslant 1.$$

Substituting the expressions of A^+ and A^-, we deduce that

$$\limsup_{k\to\infty} \left\|\mathsf{A}^k\right\|^{\frac{1}{k}} \leqslant \rho(\mathsf{A}) + \varepsilon, \qquad \liminf_{k\to\infty} \left\|\mathsf{A}^k\right\|^{\frac{1}{k}} \geqslant \rho(\mathsf{A}) - \varepsilon.$$

Since ε was arbitrary, we obtain that

$$\rho(\mathsf{A}) \leqslant \liminf_{k\to\infty} \left\|\mathsf{A}^k\right\|^{\frac{1}{k}} \leqslant \limsup_{k\to\infty} \left\|\mathsf{A}^k\right\|^{\frac{1}{k}} \leqslant \rho(\mathsf{A}),$$

which implies the statement. □

Appendix B

Brief introduction to Julia

In this chapter, we very briefly present some of the basic features and functions of Julia. Most of the information contained in this chapter can be found in the online manual, to which we provide pointers in each section.

Installing Julia

The suggested programming environment for this course is the open-source text editor Visual Studio Code. You may also use *Vim* or *Emacs*, if you are familiar with any of these.

Task 1. *Install Visual Studio Code. Install also the Julia and Jupyter Notebook extensions.*

Obtaining documentation

To find documentation on a function from the Julia console, type "`?`" to access "help mode", and then the name of the function. Tab completion is helpful for listing available function names.

Task 2. *Read the help pages for* `if`, `while` *and* `for`. *More information on these keywords is available in the online documentation.*

Remark B.1 (Shorthand `if` notation). If there is no `elseif` clause, it is sometimes convenient to use the following shorthand notations instead of an `if` block.

```
condition = true

# Assign x = 0 if `condition` is true, else assign x = 2
x = condition ? 0 : 2

# Print "true" if `condition` is true
condition && println("true")

# Print "false" if `condition` is false
condition || println("false")
```

Installing and using a package [link to relevant manual section]

To install a package from the Julia REPL (Read Evaluate Print Loop, also more simply called the Julia console), first type "`]`" to enter the package REPL, and then type `add` followed by the name of the package to install. After it has been added, a package can be used with the `import` keyword. A function `fun` defined in a package `pack` can be accessed as `pack.fun`. For example, to plot the cosine function from the Julia console or in a script, write

```
import Plots
Plots.plot(cos)
```

Alternatively, a package may be imported with the `using` keyword, and then functions can be accessed without specifying the package name. While convenient, this approach is less descriptive; it does not explicitly show what package a function comes from. For this reason, it is often recommended to use `import`, especially in a large codebase.

Task 3. *Install the `Plots` package, read the documentation of the `Plots.plot` function, and plot the function $f(x) = \exp(x)$. The tutorial on plotting available at this link may be useful for this exercise.*

Remark B.2. We have seen that `?` and `]` enable to access "help mode" and "package mode", respectively. Another mode which is occasionally useful is "shell mode", which is accessed with the character `;` and allows to type `bash` commands, such as `cd` to change directory. See this part of the manual for additional documentation on Julia modes.

Printing output

The functions `println` and `print` enable to display output. The former adds a new line at the end and the latter does not. The symbol `$`, followed by a variable name or an expression within brackets, can be employed to perform *string interpolation*. For instance, the following code prints `a = 2, a^2 = 4`.

```
a = 2
println("a = $a, a^2 = $(a*a)")
```

To print a matrix in an easily readable format, the `display` function is very useful.

Defining functions [link to relevant manual section]

Functions can be defined using a `function` block. For example, the following code block defines a function that prints "Hello, NAME!", where NAME is the string passed as argument.

```
function hello(name)
    # Here * is the string concatenation operator
    println("Hello, " * name)
end
```

```
# Call the function
hello("Bob")
```

If the function definition is short, it is convenient to use the following more compact syntax:

```
hello(name) = println("Hello, " * name)
```

Sometimes, it is useful to define a function without giving it a name, called an *anonymous function*. This can be achieved in Julia using the arrow notation `->`. For example, the following expressions calculate the squares and cubes of the first 5 natural numbers. Here, the function `map` enables to transform the collection passed as second argument by applying the function passed as first argument to each element.

```
squares = map(x -> x^2, [1, 2, 3, 4, 5])
cubes = map(x -> x^3, [1, 2, 3, 4, 5])
```

The `return` keyword can be used for returning a value to the function caller. Several values, separated by commas, can be returned at once. For instance, the following function takes a number x and returns a tuple (x, x^2, x^3).

```
function powers(x)
    return x, x^2, x^3
end

# This is an equivalent definition in short notation
short_powers(x) = x, x^2, x^3

# This assigns a = 2, b = 4, c = 8
a, b, c = powers(2)
```

Like many other languages, including Python and Scheme, Julia follows a convention for argument-passing called "pass-by-sharing": values passed as arguments to a function are not copied, and the arguments act as new bindings within the function body. It is possible, therefore, to modify a value passed as argument, provided this value is of mutable type. Functions that modify some of their arguments usually end with an exclamation mark `!`. For example, the following code prints first `[4, 3, 2, 1]`, because the function `sort` does not modify its argument, and then it prints `[1, 2, 3, 4]`, because the function `sort!` does.

```
x = [4, 3, 2, 1]
y = sort(x)  # y is sorted
println(x); sort!(x); println(x)
```

Similarly, when displaying several curves in a figure, we first start with the function `plot`, and then we use `plot!` to modify the existing figure.

```
import Plots
Plots.plot(cos)
Plots.plot!(sin)
```

As a final example to illustrate argument-passing, consider the following code. Here two arguments are passed to the function `test`: an array, which is a mutable value, and an integer, which is immutable. The instruction `arg1[1] = 0` modifies the array to which both `a` and `arg1` are bindings. The instruction `arg2 = 2`, on the other hand, just causes the variable `arg2` to point to a new immutable value (`3`), but it does not change the destination of the binding `b`, which remains the immutable value `2`. Therefore, the code prints `[0, 2, 3]` and `3`.

```
function test(arg1, arg2)
    arg1[1] = 0
    arg2 = 2
end
a = [1, 2, 3]
b = 3
test(a, b)
println(a, b)
```

Task 4 (Euler–Mascheroni constant for the harmonic series)**.** *Euler showed that*

$$\lim_{N\to\infty}\left(-\ln(N)+\sum_{n=1}^{N}\frac{1}{n}\right)=\gamma:=0.577...$$

Write a function that returns an approximation of the Euler–Mascheroni constant γ by evaluating the expression between brackets at a finite value of N.

```
function euler_constant(N)
    # Your code comes here
end
```

Task 5 (Ancient algorithms)**.** *The goal of this exercise is to explore three of the oldest algorithms ever invented.*

- *Circa 1600 BC, the Babylonians invented an iterative method for calculating the square root of a number. Read the relevant information on the associated Wikipedia page and write a function that calculates the square root of the argument using this algorithm.*

```
function babylonian_square_root(n)
    # Your code comes here
end
# The function should return the square root of n
```

- *Circa 300 BC, the Greek mathematician Euclid of Alexandria published the* Elements*, his famous mathematical treatise. In one of the books, he proposes an algorithm for calculating the greatest common divisor of two numbers. This algorithm, which is still in common use today, is based on the observation that if $a > b \geqslant 0$ are natural numbers, then*

$$\gcd(a, b) = \gcd(b, r), \tag{B.1}$$

where r is the remainder of the division of a by b. Indeed, in view of the equation

$$a = qb + r,$$

the common divisors of $\{a, b\}$ coincide with those of $\{b, r\}$. Using (B.1), write a function to calculate the greatest common divisor of two numbers.

```
function euclid_gcd(a, b)
    # Your code comes here
end
```

- *Circa 200 BC, the Greek mathematician Eratosthenes of Cyrene invented a method for efficiently calculating the prime numbers, which is now known as the* sieve of Eratosthenes*. Read the associated Wikipedia page and write a function implementing this algorithm.*

```
function eratosthenes_sieve(n)
    # Your code comes here
end
# The function should return an array containing all the prime
# numbers less than or equal to n.
```

Task 6 (Tower of Hanoi)**.** *We consider a variation on the classic Tower of Hanoi problem, in which the number r of pegs is allowed to be larger than 3. We denote the pegs by $p_1, \ldots, p_r$, and assume that the problem includes n disks with radii 1 to n. The tower is initially constructed in p_1, with the disks arranged in order of decreasing radius, the largest at the bottom. The goal of the problem is to reconstruct the tower at p_r by moving the disks one at the time, with the constraint that a disk may be placed on top of another only if its radius is smaller.*

It has been conjectured that the optimal solution, which requires the minimum number of moves, can always be decomposed into the following three steps, for some $k \in \{1, n-1\}$:

- *First move the top k disks of the tower to peg p_2;*
- *Then move the bottom $n-k$ disks of the tower to p_r without using p_2;*
- *Finally, move the top of the tower from p_2 to p_r.*

This suggests a recursive procedure for solving the problem, known as the Frame-Stewart algorithm. Write a Julia function `T(n, r)` *returning the minimal number of moves necessary.*

Local and global scopes [link to relevant manual section]

Some constructs in Julia introduce scope blocks, notably `for` and `while` loops, as well as `function` blocks. The variables defined within these structures are not available outside them. For example

```
if true
    a = 1
end
println(a)
```

prints `1`, because `if` does not introduce a scope block, but

```
for i in [1, 2, 3]
    a = 1
end
println(a)
```

produces `ERROR: LoadError: UndefVarError: a not defined`. The variable `a` defined within the `for` loop is said to be in the *local scope* of the loop, whereas a variable defined outside of it is in the *global scope*. In order to modify a global variable from a local scope, the `global` keyword must be used. For instance, the following code

```
a = 1
for i in [1, 2, 3]
    global a += 1
end
println(a)
```

modifies the global variable `a` and prints 4.

Multi-dimensional arrays [link to relevant manual section]

A working knowledge of multi-dimensional arrays is important for this course, because vectors and matrices are ubiquitous in numerical algorithms. In Julia, a two-dimensional array can be created by writing its lines one by one, separating them with a semicolon `;`. Within a line, elements are separated by a space. For example, the instruction

```
M = [1 2 3; 4 5 6]
```

creates the matrix

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

More generally, the semicolon enables vertical concatenation while space concatenates horizontally. For example, `[M M]` defines the matrix

$$\begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 5 & 6 & 4 & 5 & 6 \end{pmatrix}$$

The expression `M[r, c]` gives the (r, c) matrix element of M, located at row r and column c. The special entry `end` can be used to access the last row or column. For instance, `M[end-1, end]` gives the matrix entry in the second to last row and the last column. From the matrix M above, the submatrix `[2 3; 5 6]` can be obtained with `M[:, 2:3]`. Here the row index `:` means "select all lines" and the column index `2:3` means "select columns 2 to 3". Likewise, the submatrix `[1 3; 4 6]` may be extracted with `M[:, [1; 3]]`.

Remark B.3 (One-dimensional arrays). The comma `,` can also be employed for creating one-dimensional arrays, but its behavior differs slightly from that of the vertical concatenation operator `;`. For example, `x = [1, [2; 3]]` creates a `Vector` object with two elements, the first one being `1` and the second one being `[1; 3]`, which is itself a `Vector`. In contrast, the instruction `x = [1; [1; 2]]` creates the same `Vector` as `[1; 2; 3]` would.

We also mention that the expression `x = [1 2 3]` produces not a one-dimensional `Vector` but a two-dimensional `Matrix`, with one row and three columns. This can be checked using the `size` function, which for `x = [1 2 3]` returns the tuple `(1, 3)`.

There are many built-in functions for quickly creating commonly used arrays. For example,

- `transpose(M)` gives the transpose of M, and `adjoint(M)` or `M'` gives the transpose conjugate. For a matrix with real-valued entries, both functions deliver the same result.
- `zeros(Int, 4, 5)` creates a 4×5 matrix of zeros of type `Int`;
- `ones(2, 2)` creates a 2×2 matrix of ones of type `Float64`;
- `range(0, 1, length=101)`, or `LinRange(0, 1, 101)`, creates an array of size 101 with elements evenly spaced between 0 and 1 included. More precisely, `range` returns an array-like object, which can be converted to a vector using the `collect` function.
- `collect(reshape(1:9, 3, 3))` creates a 3×3 matrix with elements

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Let us also mention the following shorthand notation, called *array comprehension*, for creating vectors and matrices:

- `[i^2 for i in 1:5]` creates the vector `[1, 4, 9, 16, 25]`.
- `[i + 10*j for i in 1:4, j in 1:4]` creates the matrix

$$\begin{pmatrix} 11 & 21 & 31 & 41 \\ 12 & 22 & 32 & 42 \\ 13 & 23 & 33 & 43 \\ 14 & 24 & 34 & 44 \end{pmatrix}.$$

- `[i for i in 1:10 if ispow2(i)]` creates the vector `[1, 2, 4, 8]`. The same result can be achieved with the `filter` function: `filter(ispow2, 1:10)`.

In contrast with Matlab, array assignment in Julia does not perform a copy. For example the following code prints `[1, 2, 3, 4]`, because the instruction `b = a` defines a new binding to the array `a`.

```
a = [2; 2; 3]
b = a
b[1] = 1
append!(b, 4)
println(a)
```

A similar behavior applies when passing an array as argument to a function, as we saw previously. The `copy` function can be used to perform a copy.

Task 7. *Create a 10 by 10 diagonal matrix with the i-th entry on the diagonal equal to i.*

Broadcasting

To conclude this chapter, we briefly discuss *broadcasting*, which enables to apply functions to array elements and to perform operations on arrays of different sizes. Julia really shines in this area, with syntax that is both explicit and concise. Rather than providing a detailed definition of broadcasting, which is available in this part of the official documentation, we illustrate the concept using examples. Consider first the following code block:

```
function welcome(name)
    return "Hello, " * name * "!"
end
result = broadcast(welcome, ["Alice", "Bob"])
```

Here `broadcast` returns an array with elements `"Hello, Alice!"` and `"Hello, Bob!"`, as would the `map` function. Broadcasting, however, is much more flexible because it can handle arrays with different sizes. For instance, `broadcast(gcd, 24, [10, 20, 30])` returns an array of size 3 containing the greatest common divisors of the pairs $(24, 10)$, $(24, 20)$ and $(24, 30)$. Similarly, the instruction `broadcast(+, 1, [1, 2, 3])` returns `[2, 3, 4]`. To understand the latter example, note that `+` (as well as `*`, `-` and `/`) can be called like any other Julia functions; the notation `a + b` is just syntactic sugar for `+(a, b)`.

Since broadcasting is so often useful in numerical mathematics, Julia provides a shorthand notation for it: the instruction `broadcast(welcome, ["Alice", "Bob"])` can be written compactly as `welcome.(["Alice", "Bob"])`. Likewise, the line `broadcast(+, 1, [1, 2, 3])` can be shortened to `(+).(1, [1, 2, 3])`, or to the more readable expression `1 .+ [1, 2, 3]`.

Task 8. *Explain in words what the following instructions do.*

```
reshape(1:9, 3, 3) .* [1 2 3]
reshape(1:9, 3, 3) .* [1; 2; 3]
reshape(1:9, 3, 3) * [1; 2; 3]
```

Appendix C

Chebyshev polynomials

The Chebyshev polynomials $(T_n)_{n\in\mathbf{N}}$ are given on $[-1,1]$ by the formula

$$\forall x \in [-1,1], \qquad T_n(x) = \cos\Big(n \arccos(x)\Big). \tag{C.1}$$

Although this formula makes sense only if $x \in [-1,1]$, the polynomials are defined for all $x \in \mathbf{R}$. Equivalently, the Chebyshev polynomials can be defined from the equation

$$\forall x \in [1,\infty), \qquad T_n(x) = \cosh\Big(n \operatorname{arccosh}(x)\Big), \tag{C.2}$$

where $\cosh(\theta) = \frac{1}{2}\big(\mathrm{e}^{\theta} + \mathrm{e}^{-\theta}\big)$ and arccosh: $[1,\infty) \to [0,\infty)$ is the inverse function of cosh. The first few Chebyshev polynomials are illustrated in Figure C.1. It is immediate to show the following properties from (C.1):

- The roots of T_n are given by

$$z_k = \cos\left(\frac{\pi}{2n} + \frac{k\pi}{n}\right), \qquad k = 0, \ldots, n-1.$$

  These are illustrated in Figure C.2.

- The polynomial T_n takes the value 1 or -1 when evaluated at

$$x_k = \cos\left(\frac{k\pi}{n}\right), \qquad k = 0, \ldots, n. \tag{C.3}$$

  More precisely, it holds that $T_n(x_k) = (-1)^k$.

Exercise C.1. *Show that (C.1) defines a polynomial of degree n, and find its expression in the usual polynomial notation.*

Solution. The key idea is to rewrite the cosine function in terms of the complex exponential:

$$\cos(n\theta) = \frac{1}{2}\big(\mathrm{e}^{\mathrm{i}\,n\theta} + \mathrm{e}^{-\mathrm{i}\,n\theta}\big) = \frac{1}{2}\Big(\big(\cos(\theta) + \mathrm{i}\sin(\theta)\big)^n + \big(\cos(\theta) - \mathrm{i}\sin(\theta)\big)^n\Big).$$

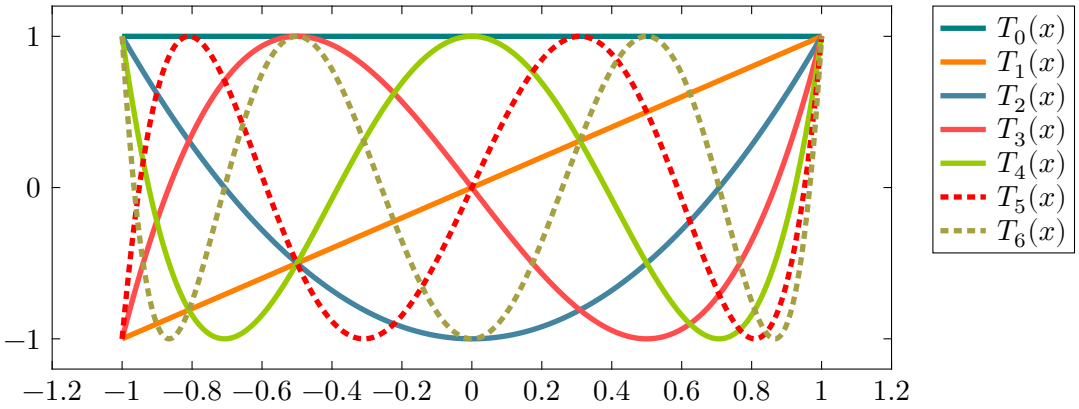


Figure C.1: Illustration of the first few Chebyshev polynomials over the interval $[-1, 1]$.

By expanding the powers on the right-hand side, we obtain

$$\big(\cos(\theta) + \mathrm{i}\sin(\theta)\big)^n = \sum_{j=0}^{n} \binom{n}{j} \cos(\theta)^{n-j}\, \mathrm{i}^j \sin(\theta)^j$$
$$\big(\cos(\theta) - \mathrm{i}\sin(\theta)\big)^n = \sum_{j=0}^{n} \binom{n}{j} \cos(\theta)^{n-j}\, (-\mathrm{i})^j \sin(\theta)^j.$$

The terms corresponding to odd values of j cancel out in the expression of $\cos(n\theta)$, and so we obtain the following expression for $\cos(n\theta)$ in terms of $\cos(\theta)$ and $\sin(\theta)$:

$$\begin{aligned}\cos(n\theta) &= \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{n}{2j} \cos(\theta)^{n-2j} \mathrm{i}^{2j} \sin(\theta)^{2j} \\ &= \sum_{j=0}^{\lfloor n/2 \rfloor} (-1)^j \binom{n}{2j} \cos(\theta)^{n-2j} \big(1 - \cos(\theta)^2\big)^j.\end{aligned}$$

Therefore, we conclude that

$$T_n(x) = \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{n}{2j} x^{n-2j} \big(x^2 - 1\big)^j. \tag{C.4}$$

△

Exercise C.2. *Show that the same polynomials are obtained from (C.2).*

Solution. Notice that

$$\begin{aligned}\cosh(n\xi) &= \frac{1}{2}\big(\mathrm{e}^{n\xi} + \mathrm{e}^{-n\xi}\big) \\ &= \frac{1}{2}\Big(\big(\cosh(\xi) + \sinh(\xi)\big)^n + \big(\cosh(\xi) - \sinh(\xi)\big)^n\Big).\end{aligned}$$

Using the binomial formula, we obtain

$$\cosh(n\xi) = \frac{1}{2}\sum_{j=0}^{n}\binom{n}{j}(\cosh(\xi)^{n-j}\sinh(\xi)^j + \cosh(\xi)^{n-j}(-1)^j\sinh(\xi)^j)$$
$$= \frac{1}{2}\sum_{j=0}^{n}\binom{n}{j}\cosh(\xi)^{n-j}(\sinh(\xi)^j + (-1)^j\sinh(\xi)^j).$$

The contributions of the odd values of j cancel out, and so we obtain

$$\cosh(n\xi) = \sum_{j=0}^{\lfloor n/2\rfloor}\binom{n}{2j}\cosh(\xi)^{n-2j}\sinh(\xi)^{2j}.$$

Since $\cosh(\xi)^2 - \sinh(\xi)^2 = 1$, we deduce that

$$\cosh(n\xi) = \sum_{j=0}^{\lfloor n/2\rfloor}\binom{n}{j}\cosh(\xi)^{n-2j}(\cosh(\xi)^2 - 1)^j,$$

which after the substitution of $\xi = \operatorname{arccosh}(x)$ leads to (C.4). △

Exercise C.3 (Yet another expression for the Chebyshev polynomials)**.** *Show that $T_n(x)$ may be defined from the formula*

$$T_n(x) = \frac{1}{2}\Big(x + \sqrt{x^2-1}\Big)^n + \frac{1}{2}\Big(x - \sqrt{x^2-1}\Big)^n \qquad \text{for} \quad |x| \geqslant 1. \tag{C.5}$$

Solution. We showed in the solution of Exercise C.2 that

$$\cosh(n\xi) = \frac{1}{2}\Big(\big(\cosh(\xi) + \sinh(\xi)\big)^n + \big(\cosh(\xi) - \sinh(\xi)\big)^n\Big).$$

Letting $\xi = \operatorname{arccosh}(x)$ in this equation and using that $\cosh(\xi)^2 - \sinh(\xi)^2 = 1$, we obtain

$$T_n(x) = \frac{1}{2}\Big(\big(x + \sqrt{x^2-1}\big)^n + \big(x - \sqrt{x^2-1}\big)^n\Big),$$

which is the required formula. △

Exercise C.4 (Recursion relation)**.** *Show that the Chebyshev polynomials satisfy the relation*

$$\forall n \in \{1, 2, \dots\}, \qquad T_{n+1} = 2xT_n - T_{n-1}. \tag{C.6}$$

Solution. It is sufficient to show the identity for $x \in [-1, 1]$, where the formula (C.1) applies. Using well-known trigonometric identities, we have

$$\cos\big((n+1)\theta\big) = \cos(n\theta)\cos(\theta) - \sin(n\theta)\sin(\theta)$$
$$\cos\big((n-1)\theta\big) = \cos(n\theta)\cos(\theta) + \sin(n\theta)\sin(\theta).$$

Adding both equations and rearranging, we obtain

$$\cos\Big((n+1)\theta\Big) = 2\cos(n\theta)\cos(\theta) - \cos\Big((n-1)\theta\Big).$$

Therefore, using this equation with $\theta = \arccos(x)$, we obtain the statement. △

Remark C.1. The recursion relation in Exercise C.4 can be employed to show by recursion that $T_n(x)$ is indeed a polynomial of degree n.

Exercise C.5. *Since $T_n \colon \mathbf{R} \to \mathbf{R}$ is a polynomial, it may be written in the standard form*

$$T_n(x) = \alpha_n^{(n)} x^n + \ldots + \alpha_1^{(n)} x + \alpha_0^{(n)}.$$

Prove that $\alpha_n^{(n)} = 2^{(n-1)}$ provided that $n \geqslant 1$.

Solution. From the definition (C.1), the Chebyshev polynomials of degrees 0 and 1 are given by $T_0(x) = 1$ and $T_1(x) = x$. The statement then follows by recursion, using Exercise C.4. △

Exercise C.6. *Let $\xi \in \mathbf{R} \setminus (-1, 1)$. Show that, among all the polynomials in $\mathbf{P}(n)$ that are bounded from above by 1 in absolute value uniformly over the interval $(-1, 1)$, the Chebyshev polynomial T_n achieves the largest absolute value when evaluated at ξ.*

Solution. Reasoning by contradiction, we assume that there exists $p \in \mathbf{P}(n)$ that satisfies

$$\sup_{x \in (-1,1)} |p(x)| \leqslant 1 \qquad \text{and} \qquad |p(\xi)| > |T_n(\xi)|.$$

Let $q(x) = p(x)T_n(\xi)/p(\xi)$. Then by construction $q(\xi) = T_n(\xi)$ and

$$\sup_{x \in (-1,1)} |q(x)| < 1.$$

Consequently, denoting by x_k the points defined in (C.3), we have that

$$\forall k \in \{0, \ldots, n\}, \qquad (-1)^k (T_n - q)(x_k) > 0.$$

In other words, the polynomial $T_n - q$ takes positive values at $\{x_0, x_2, x_4, \ldots\}$ and negative values at $\{x_1, x_3, x_5, \ldots\}$. Consequently, by the intermediate value theorem, $T_n - q$ possesses n distinct roots in the open interval $(-1, 1)$. Since, in addition, $(T_n - q)(\xi) = 0$, we deduce that $T_n - q$ has $n + 1$ distinct roots, which is a contradiction given that $T_n - q$ is a nonzero polynomial of degree at most n. △

Exercise C.7. *Assume that $0 < \lambda_1 < \lambda_2$. Prove that for any polynomial $p \in \mathbf{P}(n)$ that satisfies $p(0) = 1$, it holds that*

$$\sup_{\lambda \in (\lambda_1, \lambda_2)} |p(\lambda)| \geqslant \frac{1}{T_n(\xi)}, \qquad \xi := \frac{\lambda_2 + \lambda_1}{\lambda_2 - \lambda_1},$$

with equality for

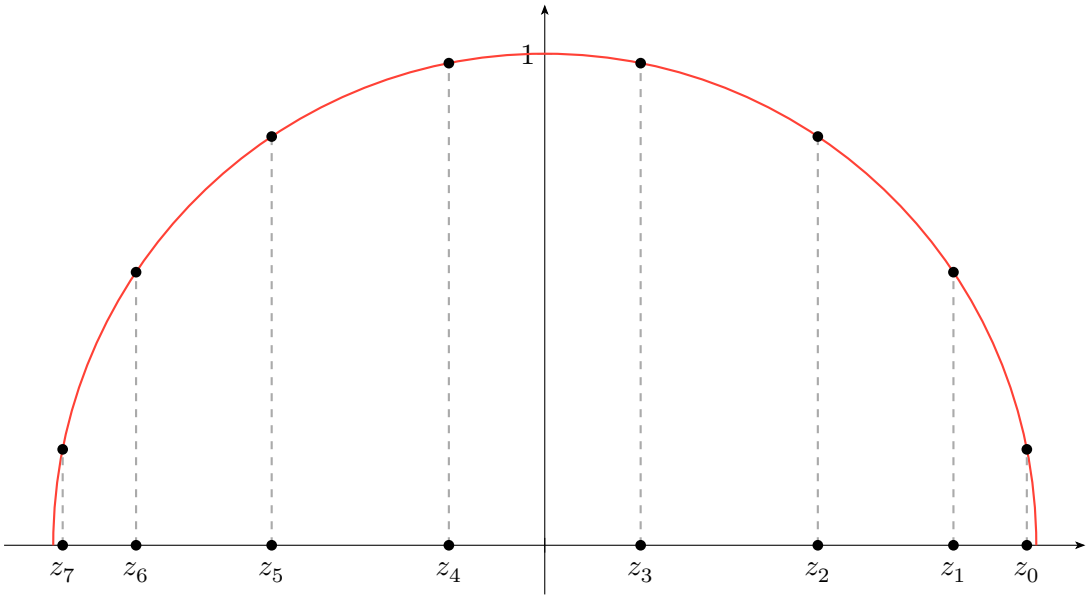


Figure C.2: Roots of the Chebyshev polynomial T_8.

$$p_*(\lambda) = \frac{T_n\left(\frac{\lambda_1+\lambda_2-2\lambda}{\lambda_2-\lambda_1}\right)}{T_n\left(\frac{\lambda_1+\lambda_2}{\lambda_2-\lambda_1}\right)}. \tag{C.7}$$

Solution. Assume that $p \in \mathbf{P}(n)$ is such that $p(0) = 1$, and let $q \in \mathbf{P}(n)$ be given by

$$q(\mu) = p\left(\frac{\lambda_1 + \lambda_2 - (\lambda_2 - \lambda_1)\mu}{2}\right) \quad \Leftrightarrow \quad p(\lambda) = q\left(\frac{\lambda_1 + \lambda_2 - 2\lambda}{\lambda_2 - \lambda_1}\right).$$

Since $\xi > 1$, it holds from (C.5) that $T_n(\xi) > 0$ and it follows from Exercise C.6 that

$$p(0) = q(\xi) \leqslant T_n(\xi) \sup_{\mu\in(-1,1)} |q(\mu)| = T_n(\xi) \sup_{\lambda\in(\lambda_1,\lambda_2)} |p(\lambda)|,$$

with equality when $q \propto T_n$, i.e. when

$$p(\lambda) \propto T_n\left(\frac{\lambda_1 + \lambda_2 - 2\lambda}{\lambda_2 - \lambda_1}\right).$$

The expression (C.7) then follows from the fact that $p_*(0) = 1$. △

Bibliography

[1] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

[2] A. Ernst and G. Stoltz. *Calcul Scientifique*. Lecture notes at École des Ponts, 2014.
URL: https://cermics.enpc.fr/cours/CS/poly.pdf.

[3] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, **23**(1):5–48, 1991.

[4] IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*:1–20, 1985.
DOI: 10.1109/IEEESTD.1985.82928.

[5] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*:1–70, 2008.
DOI: 10.1109/IEEESTD.2008.4610935.

[6] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019*:1–84, 2019.
DOI: 10.1109/IEEESTD.2019.8766229.

[7] V. Legat. *Mathématiques et méthodes numériques*. Lecture notes for the course EPL1104 at École polytechnique de Louvain, 2009.
URL: https://perso.uclouvain.be/vincent.legat/documents/epl1104/epl1104-notes-v8-2.pdf.

[8] A. Magnus. *Analyse numérique: approximation, interpolation, intégration*. Lecture notes for the course INMA2171 at École polytechnique de Louvain, 2010.
URL: https://perso.uclouvain.be/alphonse.magnus/num1a/m217111.pdf.

[9] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*, volume **30** of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.
DOI: 10.1137/1.9780898719468.
URL: https://doi-org.extranet.enpc.fr/10.1137/1.9780898719468.

[10] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume **37** of *Texts in Applied Mathematics*. Springer-Verlag, Berlin, second edition, 2007.
DOI: 10.1007/b98885.

[11] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.
DOI: 10.1137/1.9780898718003.
URL: https://doi-org.extranet.enpc.fr/10.1137/1.9780898718003.

[12] Y. Saad. *Numerical methods for large eigenvalue problems*, volume **66** of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2011.
DOI: 10.1137/1.9781611970739.ch1.
URL: https://doi-org.extranet.enpc.fr/10.1137/1.9781611970739.ch1.

[13] J. R. Shewchuk and others. An introduction to the conjugate gradient method without the agonizing pain, 1994.
URL: https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.

[14] L. N. Trefethen. The definition of numerical analysis. Technical report, 1992.
URL: https://cims.nyu.edu/~oneil/courses/sp18-math252/trefethen-def-na.pdf.

[15] P. Van Dooren. *Analyse numérique*. Lecture notes for the course INMA1170 at École polytechnique de Louvain, 2012.

[16] F. Verhulst. *Nonlinear differential equations and dynamical systems*. Universitext. Springer-Verlag, Berlin, second edition, 1996.
DOI: 10.1007/978-3-642-61453-8.
URL: https://doi-org.extranet.enpc.fr/10.1007/978-3-642-61453-8.

[17] M. Vianello and R. Zanovello. On the superlinear convergence of the secant method. *Amer. Math. Monthly*, **99**(8):758–761, 1992.
DOI: 10.2307/2324244.
URL: https://doi-org.extranet.enpc.fr/10.2307/2324244.

[18] C. Vuik and D. J. P. Lahaye. *Scientific Computing*. Lecture notes for the course wi4201 at Delft University of Technology, 2019.
URL: http://ta.twi.tudelft.nl/users/vuik/wi4201/wi4201_notes.pdf.