UNIVERSITY OF ARCHITECTURE,
CIVIL ENGINEERING AND GEODESY

Lecture Notes in Applied Mathematics

Mihail Konstantinov

FOUNDATIONS OF NUMERICAL ANALYSIS (with MATLAB examples)

Second Edition

VOLUME 1

Sofia
2007

Annotation. The first edition of these Lecture Notes appeared in 2005 and its paper version is already over. In the second edition some misprints in the first edition have been corrected and certain new material has been included.

In Volume 1 of the Lecture Notes we consider some aspects of the numerical analysis and its applications, which are connected with the effects of machine (finite) arithmetic on the result of numerical computations. We analyze the elements of the machine arithmetic and study selected issues of numerical linear algebra and some basic problems of mathematical analysis. Special attention is paid to the myths, or misconceptions, in the computational practice.

In Volume 2 we deal with the solution of differential equations (both ordinary and partial) as well as with the solution of finite equations and optimization problems.

The Lecture Notes cover parts of the discipline "Applied Mathematics" for the students in Hydro-Technical and Transport Engineering in both the University of Architecture, Civil Engineering and Geodesy - Sofia and the Technical University of Vienna. The lecture course is also used in the discipline "Numerical Matrix Analysis" for the master degree course in mathematics provided by the Technical University of Sofia.

These Lecture Notes are also intended for the students of technical and economical universities and are connected with the courses in Linear Algebra and Analytical Geometry (Mathematics I), Mathematical Analysis I (Mathematics II) and Applied Mathematics (Mathematics IV) for the engineering faculties.

The Lecture Notes may be used by students in other mathematical disciplines as well as by engineers and specialists in the practice.

An important feature of these Notes is the systematic use of the interactive computer system MATLAB (MATLAB is a trade mark of MathWork, Inc.) for the numerical solution of all mathematical problems considered. In certain cases similar results may be obtained using the freely distributed computer systems SYSLAB and Scilab.

Foundations of numerical analysis (with MATLAB examples). Volume 1

Contents

Chapter 1

Introduction

To Mihail, Konstantin, Simeon and Mihaela

1.1 Brief description of the book

In these lectures we consider the foundations of numerical analysis in the framework of the use of machine arithmetic.

The book is divided into two volumes, each of them consisting of three parts.

In Part I of Volume 1 we consider the foundations of floating point computations: Machine Arithmetic (Chapter 2) and Computational Problems and Algorithms (Chapter 3).

Part II is devoted to the numerical linear algebra. It consists of seven chapters. Basic facts about matrix computations can be found in the first three chapters: Operations with Matrices (Chapter 4), Orthogonal and Unitary matrices (Chapter 5), and Orthogonal/Unitary Matrix Decompositions (Chapter 6). The solution of linear algebraic equations and least squares problems is discusses in Chapters 7 and 8, respectively. Eigenstructure computations are described in Chapter 9, while the rank determination of a matrix is considered in Chapter 10.

Part III deals with the basic numerical mathematical analysis. The following problems are addressed: function evaluation (Chapter 11), general aspects of curve fitting (Chapter 13), interpolation and least squares approximation (Chapters 14 and 15). Numerical differentiation and integration are discussed in Chapters 16 and 17, respectively.

Finally, we consider some wide spread myths, or misconceptions, in numerical analysis.

In each chapter there is a section describing the solution of the corresponding problems via the program systems MATLAB and SYSLAB.

1.2 Main abbreviations

In these lectures we use the following abbreviations:

- $\mathbb{N}$ – the set of natural numbers $1, 2, \ldots$;
- $\overline{m, n}$ – the set of integers $m, m+1, \ldots, n$, where $m, n \in \mathbb{N}$ and $m \leq n$. When $m > n$ the set $\overline{m, n}$ is void;
- $\mathbb{Z}$ – the set of integers $0, \pm 1, \pm 2, \ldots$;
- $\mathbb{Q}$ – the set of rational numbers p/q, where $p \in \mathbb{Z}$, $q \in \mathbb{N}$ and p, q are coprime;
- $\mathbb{R}$ – the set of real numbers;
- $\mathbb{C}$ – the set of complex numbers. We have
$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C};$$
- $\imath = \sqrt{-1} \in \mathbb{C}$ – the imaginary unit;
- $\mathbb{M} \subset \mathbb{R}$ – the set of machine numbers;
- $\mathbb{R}^{m \times n}$ – the set of real $m \times n$ matrices $A = [a_{kl}]$. The element a_{kl} of A in position (k, l) (row k and column l) is denoted also as $A(k, l)$ in accordance with the MATLAB notation;
- $\mathbb{C}^{m \times n}$ – the set of complex $m \times n$ matrices;
- $\mathbb{R}^n = \mathbb{R}^{n \times 1}$ – the set of real column n–vectors $x = [x_k]$. The element x_k of x in position k is denoted also as $x(k)$ which is compatible with the MATLAB notation;
- $\mathbb{C}^n = \mathbb{C}^{n \times 1}$ – the set of complex column n–vectors;
- $A^\top \in \mathbb{R}^{n \times m}$ – the transpose of the matrix $A \in \mathbb{R}^{m \times n}$, defined by $A^\top(k, l) = A(l, k)$;

- $A^{\mathrm{H}} \in \mathbb{C}^{n\times m}$ – the complex conjugate transpose of the matrix $A \in \mathbb{C}^{m\times n}$, defined by $A^{\mathrm{H}} = \overline{A}^{\top}$, or $A^{\mathrm{H}}(k,l) = \overline{A(l,k)}$;
- $\|\cdot\|$ – a norm in $\mathbb{R}^n$ and $\mathbb{R}^{m\times n}$ or in $\mathbb{C}^n$ and $\mathbb{C}^{m\times n}$. We stress that whether a vector norm or a matrix norm is used, should become clear from the context.

We usually use the Euclidean norm

$$\|x\|_2 = \sqrt{x^{\top}x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

of the vector $x = [x_1, x_2, \ldots, x_n]^{\top} \in \mathbb{R}^n$. In the complex case

$$\|x\|_2 = \sqrt{x^{\mathrm{H}}x} = \sqrt{|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2}\,.$$

The norm of a vector or a matrix is a condensed measure of its size. The most used norms for matrices are the spectral norm, or 2–norm $\|A\|_2$, and the Frobenius norm $\|A\|_{\mathrm{F}}$.

The spectral norm $\|A\|_2$ is equal to the square root of the maximum eigenvalue of the matrix $A^{\mathrm{H}}A$ (or the maximum singular value $\sigma_{\max}(A)$ of A). According to another definition,

$$\|A\|_2 = \max\{\|Ax\|_2 : \|x\|_2 = 1\}$$

but this is not a practical way to compute the 2–norm of a matrix. We note that the computation of the spectral norm may be a difficult numerical task.

The Frobenius norm of the $m \times n$ matrix A is defined as

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{k=1}^{m}\sum_{l=1}^{n} |a_{kl}|^2}.$$

It is equal to the Euclidean norm of the mn–vector $\mathrm{vec}(A)$, obtained by stacking column-wise the elements of the matrix A.

If λ is a number, then we may define the *product* of the vector x with elements x_k and λ as the vector $\lambda x = x\lambda$ with elements λx_k.

Let x and y be n–vectors with elements x_k and y_k. Then the *sum* of the vectors x and y is the vector $z = x + y$ with elements $z_k = x_k + y_k$.

The *scalar product* (known also as *inner*, or *dot product*) of the vectors x and y is the number

$$(x,y) = y^{\top}x = x^{\top}y = x_1y_1 + x_2y_2 + \cdots + x_ny_n \in \mathbb{R}$$

in the real case, and

$$(x,y) = x^{\mathrm{H}}y = \overline{x_1}y_1 + \overline{x_2}y_2 + \cdots + \overline{x_n}y_n \in \mathbb{C}$$

in the complex case. Note that here the scalar product $(y,x) = y^{\mathrm{H}}x = \overline{(x,y)}$ is the complex conjugate of (x,y) (sometimes the scalar product of the complex vectors x and y is defined as $y^{\mathrm{H}}x$). With this notation we have

$$\|x\|_2 = \sqrt{(x,x)}.$$

In the computational practice we also use the norms

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

and

$$\|x\|_\infty = \max\{|x_k| : k = 1, 2, \ldots, n\}$$

which are easier for computation in comparison with the 2–norm $\|x\|_2$.

More generally, for each $p \geq 1$ (this p may not be an integer) we may define the so called "p–norm" as

$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{1/p}.$$

If the matrix $A \in \mathbb{C}^{n\times n}$ is nonsingular, then the quantity

$$\|x\|_{A,p} = \|Ax\|_p$$

is also a norm of the vector x.

With the help of the vector p–norm we may define a p–norm of the $m \times n$ matrix A by

$$\|A\|_p = \max\{\|Ax\|_p : x \in \mathbb{C}^n,\ \|x\|_p = 1\}.$$

When A is an $m \times n$ matrix with elements a_{kl} and λ is a number, then we may define the *product* of A and λ as the $m \times n$-matrix $\lambda A = A\lambda$ with elements λa_{kl}.

Let A and B be matrices of equal size and elements a_{kl} and b_{kl}, respectively. Then the *sum* of the matrices A and B is the matrix $C = A + B$ of the same size and with elements $c_{kl} = a_{kl} + b_{kl}$.

For $A \in \mathbb{C}^{m\times p}$ and $B \in \mathbb{C}^{p\times n}$ we may define the *standard product* of A and B as the matrix $C = AB \in \mathbb{C}^{m\times n}$ with elements

$$c_{kl} = a_{k1}b_{1l} + a_{k2}b_{2l} + \cdots + a_{kp}b_{pl}.$$

Thus the (k,l)–th element of AB is the product of the k–th row of A and the l–th column of B. That is why the standard matrix multiplication is known as multiplication "row by column". If the matrix A is partitioned column–wise as $[a_1, a_2, \ldots, a_p]$, $a_l \in \mathbb{C}^m$, and the matrix B is partitioned row–wise as $[b_1^\top, b_2^\top, \ldots, b_p^\top]^\top$, $b_l \in \mathbb{C}^{1\times n}$, then we have

$$AB = \sum_{s=1}^{p} a_s b_s.$$

For two arbitrary matrices $A = [a_{kl}] \in \mathbb{C}^{m\times n}$ and $B \in \mathbb{C}^{p\times q}$ we may define the *Kronecker product*

$$A \otimes B = [a_{kl}B] \in \mathbb{C}^{mp\times nq}.$$

This is an $m \times n$ block matrix with blocks of size $p \times q$.

The *H'Adamard product* of two matrices $A = [a_{kl}]$ and $B = [b_{kl}]$ of size $m\times n$ is the matrix $C = A \circ B$ of the same size with elements $c_{kl} = a_{kl}b_{kl}$.

For all matrix norms we have the inequalities

$$\begin{aligned}
\|\lambda A\| &\leq |\lambda|\,\|A\|, \\
\|A + B\| &\leq \|A\| + \|B\|, \\
\|AB\| &\leq \|A\|\,\|B\|,
\end{aligned}$$

which are often used in numerical analysis. We also have the estimate

$$\|AB\|_{\mathrm{F}} \leq \min\{\|A\|_{\mathrm{F}}\|B\|_2, \|A\|_2\|B\|_{\mathrm{F}}\}.$$

If A is an $n \times n$ non–singular matrix, we denote by A^{-1} its inverse ($AA^{-1} = A^{-1}A = I_n$) and by

$$k_A = \mathrm{cond}(A) = \|A\|\,\|A^{-1}\|$$

the condition number of A relative to inversion. In this definition usually the spectral matrix norm is presupposed.

For an arbitrary matrix $A \in \mathbb{C}^{m\times n}$ we denote by $A^\dagger \in \mathbb{C}^{n\times m}$ its Moore–Penrose pseudoinverse. If A is a full rank matrix then

$$k_A = \mathrm{cond}(A) = \|A\|\,\|A^\dagger\|.$$

The symbol $:=$ means "equal by definition". A final note is concerned with the widely used notations i, j for indexes of matrix and vector elements. In MATLAB the letters `i,j` are reserved for the imaginary unit $\imath = \sqrt{-1}$. That is why we prefer to use other letters for index notation, e.g. k, l, p, q, etc.

1.3 Software for scientific computations

In this section we give a brief description of the existing software (as of March, 2005) for numerical and symbolic mathematical computations.

In these Notes we often use the interactive system MATLAB[1] for the numerical solution of various mathematical problems, see [6, 4] and

`http://www.mathworks.com/products/matlab`

It must be pointed out that MATLAB [21, 22] has a simple syntax and a very friendly interface and may be used for the realization of user's own algorithms and programs. Other products can be called by MATLAB by the gateway functions.

In many cases similar results may be obtained by using the freely distributed computer systems SYSLAB (from SYStem LABoratory), see [24, 20], and Scilab[2] as well as many other commercial or free computer systems. More information about Scilab can be found at

`http://scilabsoft.inria.fr`

Two other powerful systems for both symbolic and numerical computations are MATHEMATICA[3], see

`http://www.wolfram.com/products/mathematica`
`http://documents.wolfram.com/mathematica`

and MAPLE, see

`http://www.maplesoft.com`

Information about a similar computer system MAXIMA is available at

`http://maxima.sorceforge.net`

The computer system Octave is described in [7].

A detailed list of freely available software for the solution of linear algebra problems is given by Jack Dongarra in his site

`http://www.netlib.org/utk/people/JackDongarra/la-sw.html`

[1]MATLAB is a trademark of MathWorks, Inc., USA.
[2]Scilab is a reserved mark of the National Laboratory INRIA, France.
[3]MATHEMATICA is a trade mark of Wolfram Research, Inc.

Here the interest is in software for high performance computers that is in open source on the web for solving numerical linear algebra problems. In particular dense, sparse direct and iterative systems and solvers are considered as well as dense and sparse eigenvalue problems. Further results can be found at

```
http://www.nhse.org/rib/repositories/nhse/catalog/
        #Numerical_Programs_and_Routines
```

and

```
http://www.netlib.org/utk/papers/iterative-survey
```

In particular the user can find here the routines and solvers from the packages BLAS, PSBLAS, LAPACK, ScaLAPACK and others.

A special software system called SLICOT is developed and maintained under the European Project NICONET, see

```
http://www.win.tue.nl/niconet
```

Some of the programs included in SLICOT have condition and accuracy estimators which makes them a reliable tool for scientific computing. Another feature of SLICOT is the high efficiency of most of its algorithms and their software implementation.

1.4 Concluding remarks

I am thankful to my colleagues from the University of Architecture, Civil Engineering and Geodesy, Sofia (see the web site `http://uacg.bg`) for their consideration and support.

I have typewritten these Lecture Notes personally by the word processing system LaTeX so all typos (if any) are mine. This *if any* is of course ambiguous since a good number of typos have been observed in the first edition.

Remarks and suggestions about these Notes will be accepted with thanks at my electronic addresses

```
mmk_fte@uacg.bg,  mikon@abv.bg,  misho.konstantinov@gmail.com,
```

see also my personal web page

```
http://uacg.bg/pw/mmk
```

PART I

FLOATING POINT COMPUTATIONS

Chapter 2

Machine arithmetic

2.1 Introductory remarks

It is well known that many factors contribute to the accurate and efficient numerical solution of mathematical problems. In simple terms these are the *arithmetic* of the machine on which the calculations are carried out, the *sensitivity* (or *conditioning* in particular) of the mathematical model to small changes of the data, and the *numerical stability* of the algorithms used. In this lecture we define these concepts and consider some related topics following the considerations from [12] and [27].

For many centuries numerical methods have been used to solve various problems in science and engineering, but the importance of numerical methods grew tremendously with the advent of digital computers after the Second World War. It became clear immediately that many of the classical analytic and numerical methods and algorithms could not be implemented directly as computer codes although they were perfectly suited for hand computations. What was the reason?

When doing computations "by hand" a person can choose the accuracy of each elementary calculation and estimate – based on intuition and experience – its influence on the final result. Using such man–controlled techniques the great masters of the past have done astonishing computations with an unbelievable accuracy. On the contrary, when computations are done automatically such an error control is usually not possible and the effect of errors in the intermediate calculations must be estimated in a more formal way.

Due to this observation, starting essentially with the works of J. Von Neumann and A. Turing, modern numerical analysis evolved as a fundamental basis of computer computations. One of the central themes of this analysis is to study the solution of computational problems in finite precision (or machine) arithmetic taking into account the properties of both the mathematical problem and the numerical algorithm for its solution. On the basis of such an analysis numerical methods may be evaluated and compared with respect to the accuracy that can be achieved.

When solving a computational problem on a digital computer, the accuracy of the computed solution generally depends on the three major factors, listed below.

1. The properties of the *machine arithmetic* – in particular, the rounding unit (or the relative machine precision) and the range of this arithmetic.

2. The properties of the computational problem – in particular, *the sensitivity* of its solution relative to changes in the data, often estimated by the *conditioning* of the problem.

3. The properties of the computational algorithm – in particular, the *numerical stability* of this algorithm.

It should be noted that only by taking into account all three factors we are able to estimate the accuracy of the computed solution.

The sensitivity of computational problems and its impact on the results of computations are discussed in several textbooks and monographs, see e.g. [11, 16, 23] and [17].

Of course in many applications some other factors may play an important role in the process of computation. For example in real–time applications such as the control of a moving vehicle or of a fast technological process the efficiency of the computational algorithm (measured by the number of necessary computational operations) may be decisive. In these Lecture Notes we shall not consider in detail the problems of efficiency although we shall give estimates for the number of computational operations for some algorithms.

2.2 Floating point arithmetic

In this section we consider the basic concepts of floating point arithmetic. A digital computer has only a finite number of internal states and hence it can operate with a finite, although possibly very large, set of numbers called *machine* *numbers.* As a result we have the so called *machine arithmetic*, which consists of the set of machine numbers together with the rules for performing algebraic operations on these numbers.

In machine arithmetics some very strange things may happen. Some of them are listed below.

Recall first the basic facts about representation of numbers in the so called *positional numeral systems.* A positive integer n can be represented in the standard 10–base system as follows. First we find the number $m \in \mathbb{N}$ such that $10^m \le n < 10^{m+1}$ (actually, m is the entire part of $\log_{10} n$). Then we have the unique representation

$$n = (n_m n_{m-1} \cdots n_1 n_0)_{10} := n_m 10^m + n_{m-1} 10^{m-1} + \cdots + n_1 10^1 + n_0 \times 10^0, \quad (2.1)$$

where the integers $n_0, n_1, \ldots, n_m$ satisfy $1 \le n_m \le 9$ and $0 \le n_k \le 9$, $k = 0, 1, \ldots, m-1$.

Instead of 10, we can use any integer $B > 1$ as a base[1]. When the base $B = 10$ is used, we usually omit the subindex 10 in (2.1).

For $B = 10$ we use the decimal digits $0, 1, \ldots, 9$ while for $B = 2$ the binary digits are $0, 1$. When $B > 10$ we have to introduce new B-base digits to denote the numbers $10, 11, \ldots, B - 1$.

Example 2.1 In the case $B = 16$ we use the hexadecimal digits $0, 1, \ldots, 9$ and $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$.

Example 2.2 The 10-base number 26 may be represented as

$$\begin{aligned} 26 &= (26)_{10} = 2 \times 10^1 + 6 \times 10^0 \\ &= (222)_3 = 2 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 \\ &= (11010)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0. \end{aligned}$$

Non-integral fractions have representations with entries to the right of the point, which is used to distinguish the entire part and the fractional part of the representation, e.g.

$$\frac{41}{4} = (41.25)_{10} = 4 \times 10^1 + 1 \times 10^0 + 2 \times \frac{2}{10} + 5 \times \frac{5}{10^2}.$$

More precisely, each number $x \in (0, 1)$ can be represented as

$$x = \sum_{k=1}^{\infty} b_k 10^{-k}. \quad (2.2)$$

[1]Non-integer and even negative bases can be used in principle.

The representation (2.2) is *finite* if $b_k = 0$ for $k > k_0$ and some $k_0 \in \mathbb{N}$, and *infinite* in the opposite case. An irrational number has always a unique infinite representation.

However, there are two important issues when representing integers in a positional number system.

First, a number can have two representations, e.g.

$$0.5 = 0.4999\ldots 999\ldots,$$

for $B = 10$, and

$$1 = (0.111\cdots 111\cdots) = \sum_{k=1}^{\infty} 2^{-k}$$

for $B = 2$. Moreover, in a B-base system all numbers (and only they) of the form pB^{-k}, where $p, k \in \mathbb{N}$, have exactly two representations. However, if a rational number x has an infinite B-base representation (unique or not), it is periodic, e.g.

$$x = (0.b_1 b_2 \ldots b_s \beta\beta\beta \ldots \beta\beta\beta \ldots),$$

where β is a B-digits string. For example,

$$\frac{1}{7} = (0.142857142857\ldots)_{10}$$

and the repeating string is 142857.

Second, when a number is rational and has a unique representation, this representation may be infinite (and hence periodic). Moreover, all rational numbers that cannot be represented in the form pB^{-k} with $p, k \in \mathbb{N}$, have unique periodic representations. For example, the number $1/3$ has representations

$$\frac{1}{3} = (0.333\ldots)_{10} = (0.010101\ldots)_2.$$

Models (but not equivalents!) of positional number systems are realized on digital computers. Such models are known under the general name of *machine arithmetics.*

There are different machine arithmetics obeying different standards, the most widely used being the ANSI/IEEE 754-1985 Standard for Binary Floating Point Arithmetic [13], [11, Chap. 2]. We shall refer to this standard briefly as IEEE standard, or IEEE floating point arithmetic.

There are different ways to represent real nonzero numbers x. For instance 3.1416, 299 000 000 and $-1/250$ may be represented in an uniform way as

$$3.14160 \times 10^0,$$
$$2.99000 \times 10^8,$$
$$-4.00000 \times 10^{-3}.$$

This representation, known as *normalized representation*, may be described as

$$x = \mathtt{s}\ \mathtt{B_0.B_1B_2}\cdots\mathtt{B_T} \times B^E, \tag{2.3}$$

where s is the sign of x, $B > 1$ is an integer (10 in the cases above) – the base of the floating point arithmetic, B_k are B-base digits such that $1 \le B_0 \le B - 1$ and $0 \le B_1, B_2, \ldots, B_T \le B - 1$ and E is an integer (positive, zero or negative). The integer $T + 1 \ge 1$ is the number of B-base digits in the floating point representation of real numbers. The number $x = 0$ has its own representation.

The integer $T + 1$ is called the *length of the machine word*, or the *B-base precision* of the floating point arithmetic. The number

$$\mathtt{B_0.B_1B_2}\cdots\mathtt{B_T} = B_0 + \frac{B_1}{B} + \frac{B_2}{B^2} + \cdots + \frac{B_T}{B^T}$$

is the *significand*, or the *mantissa* of the representation of the number x. The integer E, called the *exponent*, varies in certain interval, for example $-L + 2 \le E \le L$, where $L \in \mathbb{N}$.

In real life calculations we usually use the base $B = 10$. However, for technological reasons in most machine arithmetics the underlying base is $B = 2$, or B is a power of 2, e.g. $B = 2^4 = 16$. There are also machines with $B = 3$ or even with $B = 10$ but they are not wide spread.[2]

Of course, there are many other ways to represent numbers. In the *fixed point arithmetic* we have

$$x = \mathtt{A_N}\cdots\mathtt{A_1A_0.B_1B_2}\cdots\mathtt{B_T}$$

where A_k and B_l are B-base digits.

In the *rational arithmetic* the representation is of the form

$$x = \frac{\mathtt{A_N}\cdots\mathtt{A_1A_0}}{\mathtt{B_N}\cdots\mathtt{B_1B_0}},$$

[2]Some of the first computers in the 40-s of XX century were in fact 10-base machines. Also, in the 70-s of XX century there were Russian (then Soviet) experimental 3-base computers.

where A_k and B_l are B-base digits.

In the following we will not deal with a particular machine arithmetic but shall rather consider several issues which are essential in every computing environment. For a detailed treatment of this topic see [27] and [11]. For simplicity we consider a real arithmetic.

As mentioned above, there are real numbers that cannot be represented exactly in the form (2.3).

Example 2.3 The numbers $x \in \mathbb{R}\backslash\mathbb{Q}$ such as $\sqrt{2}$, e, π, etc., cannot be represented in the form (2.3) for any $1 < B \in \mathbb{N}$. The number $1/10$ cannot be represented exactly when $B = 2$. The number $1/9$ cannot be represented exactly when $B = 2$ or $B = 10$.

Let $\mathbb{M} \subset \mathbb{R}$ be the set of machine numbers. The set $\mathbb{M}$ is finite, contains the zero 0 and is symmetric with respect to 0, i.e., if $x \in \mathbb{M}$ then $-x \in \mathbb{M}$.

In order to map $x \in \mathbb{R}$ into $\mathbb{M}$, *rounding* is used to represent x by the number $\widehat{x} \in \mathbb{M}$ (denoted also as $\mathrm{round}(x)$ or $\mathrm{fl}(x)$) which is closest to x, with some rule to break ties when x is equidistant from two neighboring machine numbers. Of course, $\widehat{x} = x$ if and only if $x \in \mathbb{M}$. We shall use the hat notation to denote also other quantities computed in machine arithmetic.

Let $\bullet$ be one of the four arithmetic operations (summation $+$, subtraction $-$, multiplication $\times$ and division $/$). Thus for each two operands $x, y \in \mathbb{R}$ we have the result $x \bullet y \in \mathbb{R}$, where $y \neq 0$ if $\bullet$ is the division. For each operation

$$\bullet \in \{+, -, \times, /\}$$

there is a *machine analogue* $\odot$ of $\bullet$ which, given $x, y \in \mathbb{M}$, yields

$$x \odot y = \mathrm{fl}(x \bullet y) \in \mathbb{M}.$$

But it is possible that the operation $\odot$ cannot be performed in $\mathbb{M}$. We also note that even if $x, y \in \mathbb{M}$, then the number $x \bullet y$ may not be a machine number and hence $x \odot y \neq x \bullet y$.

As a result some strange things happen in $\mathbb{M}$:

– an arithmetic operation $\odot$ may not be performed even if the operands are from $\mathbb{M}$;

– the associative law is violated in the sense that

$$(x \odot y) \odot z \neq x \odot (y \odot z),$$

where $\bullet$ stands for $+$ or $\times$;

– the distributive law may be violated in the sense that

$$(x \oplus y) \otimes z \neq x \otimes z \oplus y \otimes z.$$

Since $\mathbb{M}$ is finite, there is a very large positive number $X_{\max} \in \mathbb{M}$ such that any $x \in \mathbb{R}$ can be represented in $\mathbb{M}$ when $|x| \leq X_{\max}$. Moreover, there is a very small positive number $X_{\min} \in \mathbb{M}$ such that if $|x| < X_{\min}$ then $\widehat{x} = 0$ even when $x \neq 0$. We say that a number $x \in \mathbb{R}$ is in the *standard range* of $\mathbb{M}$ if

$$X_{\min} \leq |x| \leq X_{\max}.$$

In the IEEE double precision arithmetic we have

$$X_{\max} \approx 1.8 \times 10^{308}, \quad X_{\min} \approx 4.9 \times 10^{-324},$$

where asymmetry is a consequence of the use of subnormal numbers [13], [11, Chap. 2] (for more details see Section 2.5).

In fact, according to (2.3) we have the *maximum normalized number*

$$X_{\max} = (B-1).(B-1)(B-1)\cdots(B-1) \times B^L = B^{L+1}(1 - B^{-T-1})$$

and the *minimum normalized number*

$$E_{\min} = 1.00\ldots0 \times B^{2-L} = B^{2-L}.$$

However, due to the existence of the so called subnormal, or denormal numbers, we have the much smaller *minimum (non-normalized) number*

$$X_{\min} = 0.00\cdots01 \times B^{-L+2} = B^{-L-T+2}.$$

In real floating point arithmetic the things are even more complex, see also Section 2.5.

If a number x with $|x| > X_{\max}$ appears as an initial data or as an intermediate result in a computational procedure, realized in $\mathbb{M}$, then there are two options.

First, the computations may be terminated and this is a very unpleasant situation. Second, the number x may be called something as "plus infinity" `Inf`, or "minus infinity" `-Inf`. Such a number may appear as a result of the operation 1/0. In this case a warning such as "divide by zero" should be issued. In further computations, if $x \in \mathbb{M}$ then $x/$`Inf` is considered as zero. Any of the above situations is called an *overflow* and must be avoided.

At the same time expressions such as 1/0 - 2/0 are not defined and they receive the name `NaN` from Not a Number.

If a number $x \neq 0$ with $|x| < X_{\min}$ appears during the computations, then it is rounded to $\widehat{x} = 0$ and this phenomenon is known as *underflow.* Although not so destructive as overflow, underflow should also be avoided. Over- and underflows may be avoided by appropriate *scaling* of the data as Example 2.4 below suggests. It should be noted that often numerical problems occur because the data are represented in units that are widely differing in magnitude.

Another close heuristic rule is that *if over- or underflow occur then the model is badly scaled and/or the method used to solve the corresponding problem is not appropriate for this purpose.*

Example 2.4 Consider the computation of the norm

$$y = \|x\| = \sqrt{x_1^2 + x_2^2}$$

of the real 2-vector $x = [x_1, x_2]^\top$, where the data x_1, x_2 and the result y are in the standard range of the machine arithmetic. In particular, this means that

$$X_{\min} \leq |x_1|,\ |x_2| \leq X_{\max}.$$

If, however, it is fulfilled that

$$x_1^2 > X_{\max}$$

then the direct calculation of y will give overflow. Another difficulty arises when

$$x_1^2,\ x_2^2 < X_{\min}.$$

Then we have the underflow

$$\mathrm{round}(x_1^2) = \mathrm{round}(x_2^2) = 0$$

resulting in the wrong answer $\widehat{y} = 0$ while the correct answer is $y \geq X_{\min}\sqrt{2}$.

Overflow may be avoided by using the scaling

$$\xi_k := \frac{x_k}{s},\ \ s := |x_1| + |x_2|$$

(provided $s \leq X_{\max}$) and computing the result from

$$y = s\sqrt{\xi_1^2 + \xi_2^2}\,.$$

Underflow can also be avoided to a certain extent by this scaling (we shall have at least $\widehat{y} \geq X_{\min}$ when $x_1^2, x_2^2 < X_{\min}$).

To avoid additional rounding errors in scaling procedures, the scaling factor is taken as a power of the base B of the machine arithmetic. In particular, in Example 2.4 the scaling factor may be taken as the closest to $|x_1| + |x_2|$ number of the form B^k, where $k \in \mathbb{Z}$.

Example 2.5 A more sophisticated example of scaling is the computation of the matrix exponential $\exp(A)$ of $A \in \mathbb{R}^{n\times n}$ by the truncated series

$$S_N(A) := \sum_{k=0}^{N} \frac{A^k}{k!}.$$

The motivation here is that $S_N(A)$ tends to $\exp(A)$ as $N \to \infty$. However, the direct use of this approximation for $\exp(A)$ may lead to large errors since the elements of the powers A^k of A may become very large before being suppressed by the denominator $k!$.

We may use the expression for $S_N(A)$ if e.g. $a := \|A\|_1 \leq 1$. For $a > 1$ we may proceed as follows. Setting $t := 1 + [\log_B a]$, where $[z]$ is the entire part of z, we may compute A^k from

$$A^k = s^k A_1^k, \quad A_1 := \frac{A}{s}, \quad s := B^t.$$

Since $\|A_1\|_1 \leq 1$ then the computation of the powers A_1^k of the matrix A_1 is relatively harmless. Note that here the scaling does not introduce additional rounding errors.

In practice, processing of quantities close to $X_{\min}$ or $X_{\max}$ may be more involved as described below.

First, there may be a constant $\mu_0 > 1$ such that all (small) numbers x, satisfying

$$|x| = \frac{X_{\min}}{\mu}, \ 1 \leq \mu \leq \mu_0,$$

are rounded to $X_{\min}$, while only numbers with

$$0 < |x| < \frac{X_{\min}}{\mu_0}$$

are rounded to zero (see Exercise 2.3).

Second, there may be a (large) constant X such that numbers x with

$$|x| \in [X_{\max}, X_{\max} + X]$$

are rounded to $X_{\max}$, while only numbers with $|x| > X_{\max} + X$ will overflow and will be addressed as `Inf` or `-Inf` (see Exercise 2.4).

A very important[3] characteristic of $\mathbb{M}$ is the *rounding unit* (or *relative machine precision*, or *machine epsilon*), denoted by $\mathbf{u}$, which is half the distance from 1 to the next larger machine number. Hence we should have

$$\mathbf{u} \approx \frac{B^{-T}}{2}.$$

When $B = 2$ it is fulfilled $\mathbf{u} \approx 2^{-T-1}$.

The quantity $\mathbf{u}$ determines the relative errors in rounding and performing arithmetic operations in floating point machine arithmetic. In particular, if

$$E_{\min} \le |x| \le X_{\max}$$

then the relative error in the approximation of x by its machine analogue $\widehat{x}$ satisfies the important bound

$$\frac{|x - \widehat{x}|}{|x|} \le \mathbf{u}. \tag{2.4}$$

Note, however, that for numbers x with

$$X_{\min} \le |x| < E_{\min}$$

the relative error in rounding them to $\widehat{x}$ may be considerably larger than (2.4) suggests. Finally, numbers x with

$$0 < |x| < X_{\min}$$

are rounded to zero with a relative error, equal to 1.

In IEEE double precision arithmetic we have

$$\mathbf{u} \approx 1.1 \times 10^{-16}.$$

This means that rounding is performed with a tiny relative error. For $\mathbf{u}$ thus defined we should have $1 + \mathbf{u} = 1$ in the machine arithmetic and $\mathbf{u}$ should be the largest positive number with this property. But, as we shall see later on, the machine relations $1 + 2.99999999\,\mathbf{u} = 1$ and $1 + 3\,\mathbf{u} > 1$ are possible.

The computation of $X_{\max}$, $X_{\min}$ and $\mathbf{u}$ in MATLAB is considered in detail in Section 2.5. We stress again that the properties of the floating point arithmetic may in fact have some singularities near the points $X_{\max}$ and $X_{\min}$.

[3]Eventually, the most important

Most machine arithmetics, including IEEE arithmetics, are built to satisfy the following property.

Main hypothesis of machine arithmetic *Let $x, y \in \mathbb{R}$ and $x \bullet y \in \mathbb{R}$ satisfy the inequalities*

$$E_{\min} \le |x|, |y|, |x \bullet y| \le X_{\max}.$$

Then the result of the machine analogue $\odot$ of $\bullet$ is

$$x \odot y = (x \bullet y)(1 + \varepsilon), \tag{2.5}$$

where $|\varepsilon| \le \mathbf{u}$.

Note that in some cases we have the slightly worse estimate $|\varepsilon| \le C\mathbf{u}$, where C is 2 or 3.

The property (2.5) has a special name among specialists, namely they call it the "$1 + \varepsilon$ property". It is amazing how many useful error estimates can be derived based upon this simple rule.

If this property holds, then arithmetic operations on two numbers are performed very accurately in $\mathbb{M}$, with a relative error of order of the rounding unit $\mathbf{u}$. Note, however, that in order the Main hypothesis of machine arithmetic to hold, it may be necessary to perform the computations with the so called "guard", or reserve digit. This is important in particular in subtraction. Modern machine arithmetics perform the subtraction $x \ominus y$, $x \ge y > 0$, exactly when $x \le 2y$.

Example 2.6 In IEEE double precision arithmetic the associative rule for summation and multiplication may be violated as follows. The computation of

$$1 + 10^{17} - 10^{17}$$

as

$$1 \oplus (10^{17} \ominus 10^{17})$$

gives the correct answer 1, while the result of the machine summation

$$(1 \oplus 10^{17}) \ominus 10^{17}$$

is 0.

In turn, the computation of

$$10^{155} 10^{155} 10^{-250}$$

as

$$10^{155} \otimes (10^{155} \otimes 10^{-250})$$

will produce an accurate approximation to the correct answer 10^{60}, while the attempt to compute

$$(10^{155} \otimes 10^{155}) \otimes 10^{-250}$$

will give an overflow.

Some computer platforms avoid such trivial attempts to “cheat” them; however even then similar phenomena may take place in a slightly more complicated way.

One of the most dangerous operations during the computations is the subtraction of numbers that are very close to each other (although in most cases this is done exactly in machine arithmetic). In this case a *subtractive cancellation* may occur.

Example 2.7 Consider the numbers

$$x = 0.123456789\xi, \ y = 0.123456789\eta,$$

which agree in their first 9 significant decimal digits. The result of the subtraction is

$$z = x - y = \pm 0.000000000|\xi - \eta|.$$

If the original x and y are subject to errors, likely to be at least of order 10^{-10}, then the subtraction has brought these errors into prominence and z may be dominated by error. If in particular the first 9 digits of x and y are true, and the digits ξ, η are uncertain, then there may be no true digit in the result z.

This phenomenon has nothing (or little) to do with the errors of the machine subtraction (if any), except for the fact that the inaccuracies in x and y may be due to rounding errors at previous steps.

Example 2.8 If we compute

$$y = \frac{(1+x)^2 - (2x+1)}{x^2}, \ x \neq 0,$$

in the ANSI/IEEE 754-1985 Standard for

$$x = 10^{-1}, 10^{-2}, \ldots, 10^{-n}, \ldots,$$

we shall see that for $n \geq 7$ the computed result $\widehat{y}$ is far from the correct answer $y = 1$; even negative values for $\widehat{y}$ will appear. The reason for this phenomenon is that we have cancellation in the numerator for small x. For $n > 8$ the result of the subtraction is of order $\varepsilon \approx 10^{-16}$ or less and there is no correct digit in the computed numerator. To worsen the situation, this wrong result is divided by a denominator that is less than 10^{-16}. This example is very instructive because the input of order 10^{-8} and an output 1 are by no means close to the boundaries of the standard range $[10^{-324}, 10^{308}]$ of $\mathbb{M}$.

Otherwise speaking, *cancellation is the loss of true significant digits when small numbers are computed by subtraction from large numbers.*

Example 2.9 A classical example of cancellation is the computation of $y = \exp(x)$ for $x < 0$ by truncating the series

$$\exp(x) := 1 + x + \frac{x^2}{2} + \cdots + \frac{x^n}{n!} + \cdots =: S_n(x) + \cdots ,$$

e.g. $y \simeq S_n(x)$. Theoretically, there is a very convenient way to decide where to cut this alternating series since the absolute truncation error

$$E_n(x) := |y - S_n(x)|$$

in this case does not exceed the absolute value $|x|^{n+1}/(n+1)!$ of the first truncated term.

Below we present the actual relative error

$$e(x) := E_{200} \exp(-x)$$

for some values of x and $n = 200$ (the computations are done by MATLAB 6.0 in machine arithmetic with rounding unit of order 10^{-16}). In all cases the truncation error is very small and may be neglected. Hence the numerical catastrophe is due only to cancellations.

x	$e(x)$
-15	1.04×10^{-5}
-16	2.85×10^{-4}
-17	1.02×10^{-3}
-18	4.95×10^{-2}
-19	5.44×10^{-1}
-20	1.02

Often subtractive cancellation can be avoided by a simple reformulation of the problem.

Example 2.10 The expression

$$y = \sqrt{1+x} - 1$$

may be computed as

$$y = \frac{x}{1+\sqrt{1+x}}$$

to avoid cancellation for small x.

2.3 Arithmetic operations with uncertain numbers

If we want to compute the quantity $z = x \bullet y$ from the data x, y, there may be two major sources of errors in the machine result $\widehat{z} = x \odot y$.

First, the operands x and y may be contaminated with previous errors, due e.g. to measurements or rounding. These errors will be manifested in the result, be it z or $\widehat{z}$. Errors of this type may be very dangerous in the subtraction of close numbers.

Second, according to the Main hypothesis of the machine arithmetic (or the $1+\varepsilon$ property), the computed result $\widehat{z}$ will differ from z due to rounding in performing the operation $\bullet$ (as $\odot$) in machine arithmetic. The relative error $|\widehat{z} - z|/|z|$ (if $z \neq 0$) is small of order $\mathbf{u}$, or 10^{-16} in the IEEE Standard.

In turn, performing arithmetic operations $z = x \bullet y$ in machine arithmetic as $\widehat{z} = x \odot y$, there are the following sources of errors: the rounding of the input data x and y to $\widehat{x}$ and $\widehat{y}$ and the rounding of the result $\widehat{x} \bullet \widehat{y}$ to $\widehat{z}$. Thus $\widehat{z}$ may sometimes not be equal to the rounded value $\mathrm{round}(z)$ of z. However, due to the Main hypothesis of machine arithmetic[4], we may assume that $\widehat{z}$ is equal to $\mathrm{round}(z)$. Thus we come to the interesting case when the uncertainty of the data is the only source of errors, i.e. when we perform *exact operations on inexact data.*

Suppose that we have two positive uncertain real numbers a and b. The uncertainty of a real number $a > 0$ may be expressed in different ways.

First, we may suppose that $a \in A := [a_1, a_2]$, where $0 < a_1 \leq a_2$. In particular, the number a will be considered as *exact* if $a_1 = a_2$ and in this case we assume $a = a_1$.

Second, we may represent a as $a = a_0(1 + \delta_a)$, where

$$a_0 := \frac{a_1 + a_2}{2}, \ |\delta_a| \leq \frac{a_2 - a_1}{a_1 + a_2}.$$

[4]Also due to the presence of guard digit and other hardware and software mechanisms

Here the number a will be exact when $\delta_a = 0$. We may also say that a is known within a *relative error* δ_a. Note that this error is relative to the "mean" value a_0 of a.

Let us now estimate the accuracy of the basic arithmetic operations performed on the uncertain operands a and b. For simplicity we will suppose that the relative errors in a and b have a common upper bound $\delta > 0$,

$$|\delta_a|, |\delta_b| \le \delta < 1.$$

Let us assume for definiteness that $a, b > 0$. It is easy to see that

$$a + b = a_0(1 + \delta_a) + b_0(1 + \delta_b) = a_0 + b_0 + a_0\delta_a + b_0\delta_b$$

and

$$\frac{|a + b - (a_0 + b_0)|}{a_0 + b_0} \le \frac{a_0|\delta_a| + b_0|\delta_b|}{a_0 + b_0} \le \delta\frac{a_0 + b_0}{a_0 + b_0} = \delta.$$

We see that the summation of two uncertain positive numbers is done with a relative error, bounded by the relative error in the data.

Furthermore, we have

$$ab = a_0(1 + \delta_a)b_0(1 + \delta_b) = a_0b_0(1 + \delta_a + \delta_b + \delta_a\delta_b)$$

and

$$\frac{|ab - a_0b_0|}{a_0b_0} \le |\delta_a| + |\delta_b| + |\delta_a\,|\delta_b| \le 2\delta + \delta^2.$$

Similarly, we have

$$\frac{|a/b - a_0/b_0|}{a_0/b_0} = \left|\frac{1 + \delta_a}{1 + \delta_b} - 1\right| = \frac{|\delta_a - \delta_b|}{1 + \delta_b} \le \frac{2\delta}{1 - \delta}.$$

In particular, if $\delta \le 1/2$ (which is the typical case) we have

$$\frac{|ab - a_0b_0|}{a_0b_0} \le 2\delta + \delta^2 \le 2\delta + \frac{\delta}{2} = \frac{5}{2}\,\delta$$

and

$$\frac{|a/b - a_0/b_0|}{a_0/b_0} \le \frac{2\delta}{1 - \delta} \le \frac{2\delta}{1 - 1/2} = 4\delta.$$

The most delicate arithmetic operation is the subtraction of equal sign numbers (positive numbers in our case). Suppose that $a_0 \ne b_0$. We have

$$a - b = a_0(1 + \delta_a) - b_0(1 + \delta_b) = a_0 - b_0 + a_0\delta_a - b_0\delta_b$$

and

$$\frac{|a - b - (a_0 - b_0)|}{|a_0 - b_0|} = \frac{|a_0\delta_a - b_0\delta_b|}{|a_0 - b_0|} \le \frac{a_0 + b_0}{|a_0 - b_0|}\,\delta.$$

Here the relative error in the result $a - b$ may be large even if the relative error δ in the data is small. Indeed, the quantity $(a_0 + b_0)/|a_0 - b_0|$ may be arbitrarily large.

Another way to address exact operations on uncertain data is via the so called *interval analysis.* If $a \in A$ and $b \in B$ is the uncertain data, where $A, B \subset \mathbb{R}$ are intervals, we can estimate the value of $a \bullet b$ by the corresponding interval

$$A \bullet B := \{a \bullet b : a \in A,\, b \in B\}.$$

For more details the reader may consult [14]. This approach may give pessimistic results in some cases of complex algorithms. However, recent interval techniques are very satisfactory in estimating the accuracy of computations. Moreover, interval methods are already implemented not only to estimate the accuracy of computed results but also to organize the computations in order to achieve high accuracy.

In many cases the errors in computational processes may be considered as random variables with a certain distribution law (the hypothesis of normal distribution is often used). Then stochastic methods may be applied in order to estimate the errors of arithmetic operations and even of complex computational algorithms.

2.4 Arithmetic operations by MATLAB and SYSLAB

MATLAB, SYSLAB and Scilab are powerful program systems for computation (numerical and symbolic) and for visualization. They also have many common features. These systems use a number of standard predefined commands, or functions - from simple arithmetic operations such as the summation `aa + b12` of two numbers, called `aa` and `b12`, to more involved operations such as

```
>> x = A\b
```

where `x` is the name of the solution x of the linear vector algebraic equation $Ax = b$.

Note that the symbol `>>` *appears automatically at the beginning of each command row.*

In what follows we shall use the typewriter font such as `typewriter font` for the variables and commands in the program systems MATLAB and SYSLAB.

The arithmetic operations summation, subtraction, multiplication and division in MATLAB, SYSLAB and Scilab are done by the commands

`+` for addition;

`-` for subtraction;

`*` for multiplication;

`/` for division.

For execution of the corresponding operation one must press the key `<Enter>` (it is also called the "carriage return" key).

Powers are computed by the command `^`. For example,

```
>> -x^(-1/3) <Enter>
```

will compute the quantity $-1/\sqrt[3]{x}$, where $x \neq 0$ is already defined.

A special command `sqrt` computes the non-negative square root of a non-negative number, e.g.

```
>> sqrt(2)
ans =
   1.4142
```

Of course, the same result will be found by the command

```
>> 2^0.5
```

Note that *the current value of the executed command is called* `ans` if some other name is not preassigned. Thus `ans` may be a scalar, a vector, a matrix or some other variable.

The percentage sign `%` is used as a symbol for comments. The whole line after this sign is not considered by MATLAB as something for execution.

Example 2.11 The expression

$$2 + 8\left(3 - \frac{6}{11}\right)^{5/7}$$

may be computed by the command

```
>> 2 + 8*(3 - 6/11)^(5/7)
%%% This is an expression %%%
```

which yields the result

```
ans =
   17.1929
```

In latest versions of MATLAB the computations are carried out in complex arithmetic. If $z = x + \imath y \in \mathbb{C}$, where $x, y \in \mathbb{R}$ and $\imath = \sqrt{-1}$, then the commands

```
>> real(z)
>> imag(z)
>> abs(z)
```

will produce the real numbers x, y and $|z| = \sqrt{x^2 + y^2}$, respectively. The command

```
>> conj(z)
```

will return the complex conjugate of z, that is $\overline{z} = x - \imath y$.

If we write z in exponential form,

$$z = r \exp(\imath\theta), \ r := |z|,$$

then the command

```
>> angle(z)
```

will give the phase angle (or argument) $\theta \in [0, 2\pi)$ of z.

The result in Example 2.11 is displayed in the short floating point format `format short`. In this format we have

```
>> 51/10000000000
ans =
  5.1000e-009
>> 299980000000
ans =
  2.9998e+011
>> 56743
ans =
       56743
>> sqrt(2)
ans =
   1.4142
```

Thus numbers in short format are displayed either in the form of integers (as 56743), or in the form

$$B_0.B_1B_2B_4B_4 \times 10^E$$

with a 5-digit significand and rounded to 4 decimal digits $B_1, \ldots, B_4$ after the decimal point. Note that *computations are still done with full 15– (or even 16–) decimal digits precision and only the result is displayed in a short form.*

The command `format long` allows to see numbers with 15 decimal digits, e.g.

```
>> format long
>> sqrt(2) =
          1.41421356237310
```

The command `format rat` gives an approximation of the answer by the ratio of some small integers, e.g.

```
>> format rat
>> sqrt(3)
ans =
  1352/780
```

Here the answer is produced by truncating continued fraction expansions. Note that *in this rational format some numbers are displayed by an integer fraction which differs from their exact value more than the rounding unit* $\mathbf{u} \approx 10^{-16}$ *suggests.*

Other predefined variables are `eps`, `realmax`, `realmin` (considered in the next section) as well as:

`pi` – an approximation of π, or 3.14159265358979;

`Inf` – the result of e.g. 1/0 (there is also `-Inf`);

`NaN` – a substitution for “Not a Number”, e.g. 0/0 or Inf/Inf;

`i,j` – notations for the imaginary unit $\imath = \sqrt{-1}$.

Some other useful variables are:

`who` – gives a list of the variables defined by the user in the current session of the program system;

`whos` – gives a list with more detailed information of the variables listed by `who`;

`clear` – deletes all variables defined by the user;

`clear name` – deletes the variable called `name`.

The reader is advised to use the commands described above in the command window of e.g. MATLAB.

General help is obtained by the command `help` and more detailed help – by the command `help function`, where `function` is the name of the function considered.

2.5 Identifying floating point arithmetic by MATLAB

The fundamental constants $X_{\max}$, $E_{\min}$ and $\mathbf{u}$ of the machine arithmetic can be found in MATLAB by the commands `realmax`, `realmin` and `eps`. On a Pentium platform, satisfying the IEEE Standards, this will give (after using the command `format long` for displaying numbers with 15 decimal digits)

```
>> Xmax = realmax
Xmax =
    1.797693134862316e+308
>> Emin = realmin
Emin =
    2.225073858507201e-308
>> eps
ans =
   2.220446049250313e-016
```

Here $X_{\max} \approx 1.7977 \times 10^{308}$ and $E_{\min} \approx 2.2251 \times 10^{-308}$ are the maximum and minimum positive normalized machine numbers, respectively. The constant `eps` is about twice the relative machine precision $\mathbf{u}$, i.e.

$$\mathbf{u} \approx \frac{\texttt{eps}}{2} = 1.110223024625157 \times 10^{-16}.$$

The minimum positive non-normalized number $X_{\min}$ is found as

```
>> Xmin = eps*realmin
Xmin =
    4.9406556458412465e-324
```

Further on we may compute the 2-base logarithms of the above constants as

```
>> log2(Xmax)
ans =
       1024
>> log2(Emin)
ans =
      -1022
>> log2(eps)
ans =
   -52
>> log2(Xmin)
ans =
      -1074
```

Hence, in bases 2, 16 and 10, the fundamental machine constants in the double precision of the IEEE standard, are

$$\begin{aligned}
X_{\max} &= 2^{1024} = 16^{256} \approx 1.8 \times 10^{308}, \\
E_{\min} &= 2^{-1022} = 16^{-255.5} \approx 2.2 \times 10^{-308}, \\
X_{\min} &= 2^{-1074} = 16^{-268.5} \approx 4.9 \times 10^{-324}, \\
\mathbf{u} &= 2^{-53} = 16^{-13.25} \approx 1.1 \times 10^{-16}.
\end{aligned}$$

As a summary, one has to remember the following practical rules about machine arithmetic.

- *The rounding unit* $\mathbf{u}$ *is about* 10^{-16} *which means that the rounding and the arithmetic operations are done with 15-16 true significant decimal digits.*
- *For safe and accurate arithmetic calculations the (non-zero) operands and the result must have absolute values in the normal range between* 10^{-308} *and* 10^{308}*. A tolerance of several orders of magnitude may increase the reliability of computations.*
- *Subtraction of close approximate numbers should be avoided whenever possible or at least be carefully watched.*

2.6 Symbolic and variable precision calculations in MATLAB

In addition to numeric calculations most modern computer systems may perform symbolic calculations. Some of the symbolic capabilities of the system Maple are incorporated in MATLAB. As a result one of the main characteristics of the machine arithmetic, the rounding unit $\mathbf{u}$, may be controlled in MATLAB using the *Variable Precision Arithmetic* option, or briefly VPA. In the standard machine arithmetic the number of digital digits in the representation of scalars, vectors and matrices is 15 or 16. In the default setting of VPA this number is already 32 but may as well be made larger or smaller. More information may be found in MATLAB environment using the command `help vpa`.

In VPA the variables are treated as symbols (strings of digits) in contrast to the standard machine arithmetic in which they are floating–point numbers from the set $\mathbb{M}$ of machine numbers. For this purpose MATLAB uses some commands from the computer system Maple.

To construct symbolic numbers, variables and objects the command `sym` is used. In particular the command `Y = sym(X)` constructs a symbolic object Y from X. If the input argument X is a string then the result Y is a symbolic variable. If the input is a numeric scalar, vector or matrix, the result is a symbolic representation of the given numeric data. To view the detailed description of this command use `help sym`.

The number of decimal digits in the symbolic representation of variables is set by the command `digits`. For example

```
>> digits(D)
```

sets to D the number of digits for subsequent calculations. In turn,

```
>> D = digits
```

returns the current setting of `digits`. The default value for the integer D is 32.

The command

```
>> R = vpa(S)
```

numerically evaluates each element of the array S using VPA with D decimal digit accuracy, where D is the current setting of `digits`.

Example 2.12 (i) The command

```
>> vpa(pi,780)
```

returns the decimal expansion of π with 780 digits. It is interesting that near digit 770 there are six consecutive 9's.

(ii) The command

```
>> vpa(hilb(3),7)
```

will return the 3×3 Hilbert matrix with 7–decimal digits accuracy, namely

```
>> ans =
   [       1., .5000000, .3333333]
   [ .5000000, .3333333, .2500000]
   [ .3333333, .2500000, .2000000]
```

There is also the command `double` which may be called for the expressions in `for`, `if` and while loops. In particular `double(x)` returns the double precision value for x. If x is already in double precision (which is the standard mode in MATLAB) this command will preserve the value for x.

2.7 Problems and exercises

Exercise 2.1 Find the binary expansions of 1/5, 1/7, 1/9 and 1/11.

Exercise 2.2 Write an algorithm and a program in MATLAB for conversion of 10-base to 2-base representations of integers and vice versa. Note that a convertor from B-base to β-base representations for $1 < B, \beta \leq 35$ can be found at

```
   www.cse.uiuc.edu/eot/modules/floating-point/base-conversion
```

Exercise 2.3 Find experimentally a number $\mu_0 > 1$ such that all numbers x, satisfying

$$|x| = \frac{X_{\min}}{\mu}, \ 1 \leq \mu \leq \mu_0,$$

are rounded to $X_{\min}$, while numbers with

$$0 < |x| < \frac{X_{\min}}{\mu_0}$$

are rounded to zero. Look, eventually, for $\mu_0 = 1.99\ldots9$

Exercise 2.4 Find a positive number X, as large as possible, such that the command

```
>> Xmax + X
```

returns the value of `Xmax`. Look for a number of the form $X = 2^{269.99\ldots9} \approx 10^{292}$.

Chapter 3

Computational problems and algorithms

3.1 Introductory remarks

In this chapter we consider the main properties of computational problems and computational algorithms. In studying computational problems we pay special attention to their sensitivity and their conditioning in particular. Concerning computational algorithms, we discuss various aspects of numerical stability.

We also derive an overall estimate for the accuracy of the computed solution taking into account the properties of the machine arithmetics, the conditioning of the problem and the stability characteristics of the computational algorithm. Effects of modularization of algorithms are also briefly discussed.

3.2 Computational problems

An important feature in assessing the results of computations in machine arithmetic is the formulation of the computational problem. Most computational problems can be written in explicit form as

$$y = f(x)$$

or, in implicit form, via the equation

$$F(x, y) = 0.$$

Here typically the *data* x and the *result* y are elements of vector spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, and $f : \mathcal{X} \to \mathcal{Y}$, $F : \mathcal{X} \times \mathcal{Y} \to \mathcal{Y}$ are given functions which are continuous or even differentiable.

Suppose that the data x is perturbed to $x + \delta x$, where the perturbation may result from measurement, modelling or rounding errors. Then the result y is changed to $y + \delta y$, where

$$\delta y = f(x + \delta x) - f(x).$$

Thus δy depends on both the data x and its perturbation δx.

The estimation of some quantitative measure $\mu(\delta y)$ of the size of δy as a function of the corresponding measure $\mu(\delta x)$ of δx is the aim of *perturbation analysis* of computational problems. If

$$x = [x_1, x_2, \ldots, x_n]^\top, \; y = [y_1, y_2, \ldots, y_m]^\top$$

are vectors, then we may use some vector norm, $\mu(x) = \|x\|$, or the vector absolute value

$$|x| := [|x_1|, |x_2|, \ldots, |x_n|]^\top$$

as such quantitative measures. In this Lectures we will use mainly norms, for simplicity.

To derive sensitivity estimates, we shall recall some basic mathematical concepts. The function $f : \mathcal{X} \to \mathcal{Y}$ is said to be *Lipschitz continuous* at a point $x \in \mathcal{X}$ if there is $r_0 > 0$ such that for every $r \in (0, r_0]$ there exists a quantity M (depending on x and r) such that

$$\|f(x + h) - f(x)\| \le M\|h\|$$

for all $h \in \mathcal{X}$ with $\|h\| \le r$. The smallest such quantity

$$M = M(x, r) := \inf \left\{ \frac{\|f(x + h) - f(x)\|}{\|h\|} : h \neq 0, \; \|h\| \le r \right\} \tag{3.1}$$

is called the *Lipschitz constant* of f in the r–neighborhood of x. Lipschitz continuous functions satisfy the perturbation bound

$$\|\delta y\| \le M(x, r)\|\delta x\| \quad \text{for} \quad \|\delta x\| \le r.$$

A computational problem $y = f(x)$, where f is Lipschitz continuous at x, is called *regular* at x. Otherwise the problem is called *singular*.

Example 3.1 Consider the polynomial equation

$$(y - 1)^n = y^n - ny^{n-1} + \cdots + (-1)^n = 0$$

which has an n–tuple solution $y = 1$. If the constant term $(-1)^n$ is changed to $(-1)^n - 10^{-n}$, then the perturbed equation will have n different roots

$$y_k = 1 + \frac{\varepsilon_k}{10}\,, \;\; k = 1, 2, \ldots, n,$$

where $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ are the primitive n–th roots of 1. Thus a small relative change of 10^{-n} in one of the coefficients leads to a large relative change of 0.1 in the solution.

In order to characterize cases when small changes in the data can lead to large changes in the result, we introduce the concept of condition number. For a regular problem $y = f(x)$, let $M(x, r)$ be as in (3.1). Then the number

$$K(x) := \lim_{r \to 0} M(x, r)$$

is called the *absolute condition number* of the computational problem $y = f(x)$. For singular problems we set $K(x) = \infty$.

We have

$$\|\delta y\| \leq K(x)\|\delta x\| + \Omega(\delta x), \tag{3.2}$$

where the scalar quantity $\Omega(h) \geq 0$ satisfies

$$\lim_{h \to 0} \frac{\Omega(h)}{\|h\|} = 0.$$

Let $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and suppose that the vector function

$$f = [f_1, f_2, \ldots, f_m]^\top : \mathbb{R}^n \to \mathbb{R}^m$$

is differentiable in the sense that all partial derivatives

$$\frac{\partial f_k(x)}{\partial x_l}$$

exist and are continuous. Then

$$K(x) = \left\| \frac{\partial f(x)}{\partial x} \right\|,$$

where

$$f_x(x) = \frac{\partial f(x)}{\partial x}$$

is the $m \times n$ Jacobi matrix of f at x with elements $\partial f_k / \partial x_l$, $k = 1, 2, \ldots, m$, $l = 1, 2, \ldots, n$, and $\| \cdot \|$ denotes both a vector norm and the corresponding subordinate matrix norm.

Suppose now that $x \neq 0$ and $y = f(x) \neq 0$. Then setting

$$\delta_x := \frac{\|\delta x\|}{\|x\|}, \ \delta_y := \frac{\|\delta y\|}{\|y\|}$$

we have the bound

$$\delta_y \leq k(x)\delta_x + \omega(\delta x), \quad \omega(h) := \frac{\Omega(h)}{\|y\|},$$

where

$$\lim_{h \to 0} \frac{\|\omega(h)\|}{\|h\|} = 0.$$

Here the quantity

$$k(x) := K(x)\frac{\|x\|}{\|y\|}$$

is the *relative condition number* of the problem $y = f(x)$. The relative condition number is more informative and is usually used in practice instead of the absolute condition number. Sometimes the relative condition number is denoted as $\operatorname{cond}(x)$.

We stress that *the relative condition number is one of the most important characteristics of a computational problem which is a measure of its sensitivity.* As we shall see later on, the size of $k(x)$ should be considered in the context of the particular machine arithmetic.

Condition numbers can be defined analogously for implicit problems, defined via the equation

$$F(x, y) = 0,$$

where x is the data, y is the solution and $F : \mathcal{X} \to \mathcal{Y}$ is a given function. If the function F is differentiable in both x and y at a point (x_0, y_0), where $F(x_0, y_0) = 0$, then we have

$$F(x_0 + \delta x, y_0 + \delta y) = F(x_0, y_0) + F_x(x_0, y_0)\delta x + F_y(x_0, y_0)\delta y + \Gamma(\delta x, \delta y).$$

Here

$$F_z(x_0, y_0) := \frac{\partial F}{\partial z}(x_0, y_0), \ z = x, y,$$

and

$$\frac{\|\Gamma(h, g)\|}{\|h\| + \|g\|} \to 0$$

for $\|h\| + \|g\| \to 0$.

If the matrix $F_y(x_0, y_0)$ is invertible then, since $F(x_0, y_0) = 0$, we have

$$\delta y = -(F_y(x_0, y_0))^{-1} F_x(x_0, y_0)\delta x - (F_y(x_0, y_0))^{-1}\Gamma(\delta x, \delta y).$$

Hence

$$\|\delta y\| \leq K(x_0, y_0)\|\delta x\| + \gamma(\delta x, \delta y),$$

where

$$K(x_0, y_0) := \left\|(F_y(x_0, y_0))^{-1} F_x(x_0, y_0)\right\|$$

is the *absolute condition number* of the problem, and the quantity

$$\gamma(h, g) := \left\|(F_y(x_0, y_0))^{-1} \Gamma(h, g)\right\|$$

is small compared to $\|h\| + \|g\|$. In a more general setting, i.e. in the solution of matrix equations, $F_y(x_0, y_0)$ is a linear operator $\mathcal{Y} \to \mathcal{Y}$.

A regular problem $y = f(x)$ is called *well conditioned* (respectively *ill conditioned*) if its relative condition number $k(x)$ is small (respectively large) in the context of the given machine arithmetic. In particular, the problem is very well conditioned if $k(x)$ is of order 1, and very ill conditioned if $\mathbf{u}\, k(x) \simeq 1$.

A more detailed classification of computational problems with regard to their conditioning may be done as follows. Denote first the reciprocal $1/k(x)$ of the relative condition number $k(x)$ as $\mathrm{rcond}(x)$. Thus small values of $\mathrm{rcond}(x)$ correspond to high sensitivity. More precisely, we have the following classification of computational problems $y = f(x)$ solved in machine arithmetic with rounding unit $\mathbf{u}$.

The problem is called:

- *well conditioned* when $\mathbf{u}^{1/3} < \mathrm{rcond}(x)$;
- *moderately conditioned* when $\mathbf{u}^{2/3} < \mathrm{rcond}(x) \leq \mathbf{u}^{1/3}$;
- *ill conditioned* when $\mathbf{u} < \mathrm{rcond}(x) \leq \mathbf{u}^{2/3}$;
- *very ill conditioned* when $\mathrm{rcond}(x) \leq \mathbf{u}$.

The computer solution of an ill conditioned problem may lead to large errors. In practice, the following heuristics may be used for the computational problem $y = f(x)$.

The rule of thumb. *Suppose that* $\mathbf{u}\, k(x) < 1$. *Then one can expect approximately*

$$-\log_{10}(\mathbf{u}\, k(x))$$

correct significant decimal digits in the largest components of the computed solution vector $\widehat{y}$.

Indeed, as a result of rounding the data x we work with $\widehat{x} = x + \delta x$, where $\|\delta x\| \leq \mathbf{u}\|x\|$. Even if no additional errors are made during the computation, then the computed result is

$$\widehat{y} = f(\widehat{x})$$

and we have

$$\|f(\widehat{x}) - f(x)\| \leq K(x)\|\delta x\| + \Omega(x) \leq \mathbf{u}\, K(x)\|x\| + \Omega(x).$$

Thus the relative error in the computed result satisfies the approximate inequality

$$\frac{\|\widehat{y} - y\|}{\|y\|} \leq \frac{\mathbf{u}\, K(x)\|x\|}{\|y\|} = \mathbf{u}\, k(x).$$

However, the rule of thumb may give pessimistic results since the probability of a problem to be ill conditioned is usually small [5].

Closely related to the sensitivity analysis is the task of estimating the *distance to the nearest singular problem.* Consider a computational problem $y = f(x)$. We call the quantity

$$\mathrm{Dist}(x) = \min\{\|h\| : \text{the problem } y = f(x+h) \text{ is singular}\},$$

the *absolute distance* to singularity of the problem $y = f(x)$. Similarly, for $x \neq 0$, the quantity

$$\mathrm{dist}(x) := \frac{\mathrm{Dist}(x)}{|x\|}$$

is the *relative distance* to singularity of the problem. Thus the distance to singularity is the distance from the point x to the set of singular problems.

For many computational problems the relative distance to singularity and the relative condition number are inversely proportional in the sense that $\mathrm{dist}(x)\,\mathrm{cond}(x) = 1$, see e.g. [5].

Example 3.2 The problem of solving the linear algebraic system

$$Ay = b$$

with a square matrix A and data $x = (A, b)$ is regular if and only if the matrix A is nonsingular. The relative distance to singularity for an invertible matrix A is $1/\mathrm{cond}(A)$, where

$$\mathrm{cond}(A) := \|A\|\,\|A^{-1}\|$$

is the relative condition number of A relative to inversion [11, Thm. 6.5].

Another difficulty that needs to be mentioned is the mathematical representation of the computational problem that one needs to solve. In particular in control theory, there are several different frameworks that are used. A classical example for such different frameworks is the representation of linear systems via matrices and vectors, as in the classical state space form, as rational matrix functions (via the Laplace transform), or even in a polynomial setting [9, 26]. These different approaches have different mathematical properties and often it is a matter of taste which framework is preferred.

From a numerical point of view, however, this is not a matter of taste, since the sensitivity may be drastically different. Numerical analysts usually prefer the matrix/vector setting over the representations via polynomial or rational functions, while for users of computer algebra systems the polynomial or rational approach is often more attractive. The reason for the preference for the matrix/vector approach in numerical methods is that the sensitivity of the polynomial or rational representation is usually much higher than that of a matrix/vector representation and this fact is often ignored in order to favor frameworks which are mathematically more elegant but numerically inadequate. The reason for the higher sensitivity is often an over condensation of the data, i.e., a representation of the problem with as few data as possible. Let us demonstrate this issue with a well known example.

Example 3.3 [28] Consider the computation of the eigenvalues of the matrix

$$A = Q^{\top}\mathrm{diag}(1, 2, \ldots, 20)Q,$$

where Q is a random orthogonal 20×20 matrix generated e.g. by the MATLAB command

```
>> [Q,R] = qr(rand(20))
```

Clearly the matrix A is symmetric and therefore diagonalizable with nicely separated eigenvalues $1, 2, \ldots, 20$. The problem of computing the eigenvalues of A is very well conditioned and numerical methods such as the symmetric QR algorithm lead to highly accurate results, see [10]. For example, the command

```
>> eig(A)
```

from MATLAB [21] yields all eigenvalues with about 15 correct decimal digits which is the highest possible accuracy of the standard machine computations.

The usual textbook approach to compute eigenvalues that is taught in first year linear algebra is that the eigenvalues of A are the roots of the characteristic polynomial

$$\det(\lambda I_{20} - A) = (\lambda - 1)(\lambda - 2)\cdots(\lambda - 20).$$

Using a numerical method such as

```
>> roots(poly(A))
```

from MATLAB to compute the roots of the characteristic polynomial `poly(A)` of A, however, yields a result with highly inaccurate large eigenvalues

```
ans =
  20.0003
  18.9970
  18.0117
  16.9695
  16.0508
  14.9319
  14.0683
  12.9471
  12.0345
  10.9836
  10.0062
   8.9983
   8.0003
   7.0000
   6.0000
   5.0000
   4.0000
   3.0000
   2.0000
   1.0000
```

(the accuracy for the small eigenvalues is slightly better). There are several reasons for this inaccuracy. First the coefficients of the polynomial range in the interval $[1, 20!]$, $20! \approx 2.43 \times 10^{18}$, and cannot all be represented accurately in the IEEE double precision arithmetic, while the elements of the matrix range in the ball of radius 20 around the origin. Second, the sensitivity of the larger roots with respect to perturbations in the coefficients is very large in this case.

In this section we have discussed the sensitivity (conditioning in particular) of a computational problem. This is a property of the problem and its mathematical representation in the context of the machine arithmetic used, and should not be confused with the properties of the computational method that is implemented to solve the problem.

In practice, linear sensitivity estimates of the type

$$\delta_y \leq k(x)\delta_x$$

are usually used neglecting second and higher terms in δ_x. This may sometimes lead to underestimation of the actual perturbation in the solution. Rigorous perturbation bounds can be derived by using the technique of non–linear perturbation analysis [18, 16].

3.3 Computational algorithms

In this section we discuss properties of computational algorithms and in particular their numerical stability. To define formally what an algorithm is may not be an easy task. For our purposes, however, we shall need the following simplified notion of an algorithm.

An algorithm to compute $y = f(x)$ is a decomposition

$$f = F_r \circ F_{r-1} \circ \cdots \circ F_1 \tag{3.3}$$

of the function f which gives a sequence of intermediate results

$$x_k = F_k(x_{k-1}),\ k = 1, 2, \ldots, r, \tag{3.4}$$

with $x_0 = x$ and $y = x_r$. Usually the computation of $F_k(\xi)$ requires simple algebraic operations on ξ such as arithmetic operations or taking roots, but it may also be a more complicated subproblem like solving a system of linear equations or computing the eigenvalues of a matrix.

The algorithm either gives the exact answer in exact arithmetic or for some problems, like eigenvalue problems or the solution of differential equations, the answer is an approximation to the exact answer in exact arithmetic. We will not analyze the latter case here but we will investigate only what happens with the computed value $\widehat{y}$ of x_r when the computations are done in machine arithmetic.

It is important to mention that two different algorithms, say (3.3), (3.4) and

$$\xi_i = \Phi_i(\xi_{i-1}),\ i = 1, 2, \ldots, s,\ \xi_0 = x,$$

where

$$f = \Phi_s \circ \Phi_{s-1} \circ \cdots \circ \Phi_1,$$

for computing $y = f(x) = \xi_s$, may give completely different results in machine arithmetic although in exact arithmetic they are equivalent.

Example 3.4 Consider the computational problem

$$y = f(x) = x_1^2 - x_2^2$$

with a vector input $x = (x_1, x_2)$ and a scalar output y. The problem may be solved by two algorithms $f = F_3 \circ F_2 \circ F_1$ and $f = \Phi_1 \circ \Phi_2 \circ \Phi_1$, where

$$F_1(x) = x_1^2,\ F_2(x) = x_2^2,\ F_3(x) = F_1(x) - F_2(x)$$

and

$$\Phi_1(x) = x_1 - x_2,\ \Phi_2(x) = x_1 + x_2,\ \Phi_3(x) = \Phi_1(c)\Phi_2(x).$$

Which algorithm produces a more accurate result depends on the data x. The detailed behavior of these algorithms is considered in [27].

In what follows we suppose that the data x is in the normal range of the machine arithmetic with characteristics $X_{\max}$, $X_{\min}$, $E_{\min}$ and $\mathbf{u}$, and that the computations do not lead to overflow or to a destructive underflow. As a result the answer computed by the algorithm (3.3) is $\widehat{y}$.

Our main goal will be to estimate the absolute error

$$E := \|\widehat{y} - y\|$$

and the relative error

$$e := \frac{\|\widehat{y} - y\|}{\|y\|},\ y \neq 0,$$

of the computed solution $\widehat{y}$ in case of a regular problem $y = f(x)$ when the data x belongs to a given set $X_0 \subset \mathcal{X}$.

Definition 3.1 *[23, 27, 11] The algorithm (3.3), (3.4) is said to be* numerically stable *on a given set X_0 of data if the computed quantity $\widehat{y}$ for $y = f(x)$, $x \in X_0$, is close to the solution $f(\widehat{x})$ of a problem with data $\widehat{x}$ near to x in the sense that*

$$\|\widehat{y} - f(\widehat{x})\| \leq \mathbf{u}\, a\|y\|, \quad \|\widehat{x} - x\| \leq \mathbf{u}\, b\|x\|, \tag{3.5}$$

where the constants $a, b > 0$ do not depend on $x \in X_0$.

Definition 3.2 *If the computed value $\widehat{y}$ is equal to the rounded value* round(y) *of the exact answer y, then the result is obtained with the* maximum achievable accuracy*.*

To obtain the result with such an accuracy is the maximum that can be required form a numerical algorithm implemented in machine arithmetic.

For a problem for which the data is inexact, perhaps itself being subject to rounding errors, numerical stability is in general the most we can ask of an algorithm. There are also other stability concepts for numerical algorithms.

Definition 3.3 *If in Definition 3.1 we take $a = 0$, then the algorithm is called* numerically backward stable*.*

Thus backward numerical stability means that the computed result $\widehat{y}$ is equal to the exact solution $f(\widehat{x})$ of a near problem (with data $\widehat{x}$ close to x).

Definition 3.4 *If in Definition 3.1 we take $b = 0$, then the algorithm is called* numerically forward stable*.*

Forward stability guarantees that the computed result $\widehat{y}$ satisfies the inequality

$$\|\widehat{y} - y\| \leq \mathbf{u}\, a\|y\|$$

with a common constant a that does not depend on $x \in X_0$. Thus the computed result and the exact solution are $\mathbf{u}$-close for all data $x \in X_0$. However, many classical algorithms even for important problems in numerical linear algebra are not forwardly stable as shown in the next example.

Example 3.5 For the solution of linear algebraic equations

$$Ay = c$$

with data $x = (A, c)$ we cannot expect forward stability for all cases with A invertible. Indeed, here the forward error $\|\widehat{y} - y\|$ may be of order $\mathbf{u}\, k(A)$, where

$$k(A) = \text{cond}(A) := \|A\|\, \|A^{-1}\|$$

is the condition number of A relative to inversion. But the number $k(A)$ is not bounded when A varies over the set of all invertible matrices. If, however, A is restricted to belong to the set of matrices with $k(A) \leq k_0$ for some moderate constant $k_0 > 1$, then the Gauss elimination algorithm and the algorithm based on QR decomposition (both algorithms implemented with pivoting) will be forwardly numerically stable. Moreover, as shown by Wilkinson, these algorithms are backwardly stable for all invertible matrices A.

The concept of backward stability, introduced by Wilkinson [28], tries to represent the error of a computational algorithm by showing that the computed solution is the exact solution of a problem with perturbed data, where the perturbation in the data is called the *equivalent data error* or the *equivalent perturbation.* For example, the elementary floating point operations are carried out in a backward stable way, because (2.5) shows that the computed answer is the correct one for slightly perturbed data $x(1+\delta_1)$ and $y(1+\delta_2)$, where δ_1, δ_2 are of order $\mathbf{u}$.

It is clear from the definitions that backward stability implies stability, but the converse is not true. In particular, some very simple algorithms are not backwardly stable, see the next example.

Example 3.6 The computational problem

$$y = f(x) := 1 + 1/x,$$

is solved directly and very accurately for $x \geq 1$. However, for $x > 1/\mathbf{u}$ we have $\widehat{y} = 1$ (a very accurate result!), but the method is not backwardly stable, since $f(x) > 1$ for all $x > 0$.

3.4 A general accuracy estimate

In this section we shall derive a very elegant accuracy estimate for the solution computed by a numerically stable algorithm. This estimate takes into account all three major factors that govern the accuracy of the computational process.

Consider the solution of the computational problem $y = f(x)$ by an algorithm that is numerically stable on a set of data X_0 (see Definition 3.1).

As in [23, 27], using the inequalities (3.5) and

$$\|f(x+h) - f(x)\| \leq K(x)\|h\| + \Omega(h)$$

(see (3.2)), we obtain the following estimate for the absolute error

$$\begin{aligned}
\|\widehat{y} - y\| &= \|\widehat{y} - f(\widehat{x}) + f(\widehat{x}) - f(x)\| \\
&\leq \|\widehat{y} - f(\widehat{x})\| + \|f(\widehat{x}) - f(x)\| \\
&\leq \mathbf{u}\, a\|y\| + K(x)\|\widehat{x} - x\| + \Omega(\widehat{x} - x) \\
&\leq \mathbf{u}\, a\|y\| + \mathbf{u}\, bK(x)\|x\| + \Omega(\widehat{x} - x).
\end{aligned}$$

Here $K(x)$ is the absolute condition number (the Lipschitz constant) of the problem $y = f(x)$, while the constants a, b depend on the algorithm used and are given in Definition 3.1.

Dividing by $\|y\|$ (in case $y \neq 0$) we find an estimate for the relative error

$$e := \frac{\|\widehat{y} - y\|}{\|y\|} \leq \mathbf{u} \left(a + bK(x) \frac{\|x\|}{\|y\|} + \frac{\omega(\widehat{x} - x)}{\mathbf{u}} \right). \tag{3.6}$$

Since $\omega(\widehat{x} - x)/\varepsilon \to 0$ for $\varepsilon \to 0$ we may ignore the third term in the brackets in the right–hand side of (3.6) resulting in the approximate estimate

$$e \leq \mathbf{u} \left(a + bK(x) \frac{\|x\|}{\|y\|} \right) = \mathbf{u}(a + bk(x)) \tag{3.7}$$

for the relative error in the computed solution.

Note that for a backwardly stable algorithm we have $a = 0$ and hence

$$e \leq \mathbf{u}\, bk(x).$$

Similarly, for a forwardly stable algorithm we have $b = 0$ and

$$e \leq \mathbf{u}\, a$$

for some constant a which may be chosen independently on $x \in X_0$ for a given set X_0 of data. In particular, we may assume that a is the supremum of $e/\mathbf{u}$ when x varies over X_0.

It is instructive to point out that the famous *rule of thumb* is a particular case of the estimate (3.7) when we presuppose a very stable algorithm with $a = 0$ and $b = 1$.

Having in mind the definition of the relative condition number $k(x)$ of the problem $y = f(x)$ which is an "achievable" quantity, and the fact that the data x is usually rounded to $\widehat{x}$ with $\|\widehat{x} - x\| \leq \mathbf{u}$, we may expect that for some problems we shall have $e \geq \mathbf{u}\, k(x)$. Hence for such problems we may assume that

$$\mathbf{u}\, k(x) \leq e \leq \mathbf{u}(a + bk(x)).$$

Inequality (3.7) shows very clearly the influence of all the three major factors that determine the accuracy of the computed solution:

– the machine arithmetic (the rounding unit $\mathbf{u}$ and implicitly the range of $\mathbb{M}$ through the requirement to avoid over– and underflows);

– the computational problem (the relative condition number $k(x)$);

– the computational algorithm (the constants a and b).

Recalling that $k(x) = K(x)\|x\|/\|y\|$ we see that large relative errors in the computed solution may be expected when $K(x)$ and/or $\|x\|$ are large as well as when $\|y\|$ is small. The latter is a well known rule in numerical analysis, namely that *large relative errors may occur when a small quantity is computed from large initial or intermediate data.* The catastrophic cancellation is a particular case of this rule.

Inequality (3.7) is an example of a *condition number based accuracy estimate* for the solution, computed in a machine arithmetic. In order to assess the accuracy of results and to be able to trust numerical results, such condition and accuracy estimates should accompany every computational procedure. Many modern software packages provide such estimates [1, 3], see also

```
http://www.win.tue.nl/niconet
```

It is, however, unfortunate common practice in engineering simulation to turn these facilities off, even though this service will warn the user of numerical methods about possible failures.

We note that the stability of an algorithm does not itself guarantee automatically a small error in the computed solution if the problem is ill conditioned, i.e. if the number $\mathbf{u}k(x)$ is about 1. The conclusion is that *good results can only be expected when a stable algorithm is used to solve a relatively well conditioned problem.*

3.5 Effects of modularization

In this section we illustrate some of the effects of modularization on the properties of computational processes.

As we have seen in (3.3), computational problems are typically modularized, which means that they are decomposed and solved by a sequence of subproblems. This facilitates the use of computational modules and is one of the reasons for success of numerical analysis. But one should be aware that this modularization may lead to substantial numerical difficulties. This happens if one or more of the created subproblems F_k that are combined to get the decomposition is ill conditioned. Consider for simplicity the decomposition

$$f = F_2 \circ F_1$$

of a regular problem $y = f(x)$ with absolute condition number $K = K(x)$ which is of order 1. Thus the original problem is very well conditioned.

First of all it may happen that one of the subproblems F_k is not regular.

Example 3.7 The scalar identity function $y = f(x) := x$ may be decomposed as

$$f = F_2 \circ F_1,$$

where $F_1(x) = x^3$ and $F_2(z) = z^{1/3}$. Here the function F_2 is not Lipschitz continuous at the point $z = 0$. Hence one of the subproblems is not even regular.

But even if the functions F_1, F_2 are Lipschitz continuous with constants K_1, K_2 respectively, then it may happen that one (or both) of these constants is large. We obtain the estimate

$$\begin{aligned} \|f(x+h) - f(x)\| &= \|F_2(F_1(x+h)) - F_2(F_1(x))\| \\ &\leq K_2\|F_1(x+h) - F_1(x)\| \leq K_2K_1\|h\|, \end{aligned}$$

where the quantity K_2K_1 may be much larger than the actual Lipschitz constant K of the function f at the point x.

Example 3.8 Consider the identity function $y = f(x) := x$ in $\mathbb{R}^2$. Define

$$F_1(x) = A^{-1}x$$

and

$$F_2(z) = Az,$$

where the matrix $A \in \mathbb{R}^{2\times 2}$ is non–singular. The original problem is perfectly conditioned with $K = 1$, while any of the constants $K_1 = \|A^{-1}\|$ and $K_2 = \|A\|$ may be arbitrarily large. Their product $K_1K_2 = \mathrm{cond}(A)$ may also be arbitrarily large.

If the computations are carried out with maximum achievable accuracy, then the computed value for $A^{-1}x$ is

$$\widehat{F}_1(x) = (I_2 + E_1)A^{-1}x,$$

where

$$E_1 := \mathrm{diag}(\varepsilon_1, \varepsilon_2)$$

and

$$|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}.$$

Similarly, the computed value for $A(A^{-1}x)$ becomes

$$(I_2 + E_2)A\widehat{F}_1(x) = (I_2 + E_2)A(I_2 + E_1)A^{-1}x,$$

where

$$E_2 := \operatorname{diag}(\varepsilon_3, \varepsilon_4)$$

and

$$|\varepsilon_3|, |\varepsilon_4| \leq \mathbf{u}.$$

Suppose that

$$\varepsilon_1 = -\varepsilon_2 = \mathbf{u} \simeq 10^{-16}, \quad \varepsilon_3 = \varepsilon_4 = 0$$

and

$$A = \begin{bmatrix} a & a+1 \\ a-1 & a \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

where $a = 10^8$. Then the computed result is

$$\widehat{x} = x + \mathbf{u}\xi,$$

where

$$\xi = [\xi_1, \xi_2]^\top$$

and

$$\begin{aligned} \xi_1 &= 4a^2 + 2a - 1, \\ \xi_2 &= 4a^2 - 2a - 1. \end{aligned}$$

Thus the actual relative error in the solution of the decomposed problem is

$$\mathbf{u}\frac{\|\xi\|}{\|x\|} \simeq 4a^2\,\mathbf{u} \simeq 4$$

and there are no correct digits in the computed result!

3.6 Solving some illustrative problems by MATLAB

The commands

```
>> x = [-1:-1:-18]'; x = 10.^x
```

will produce the 18–vector $x = [10^{-1}, 10^{-2}, \ldots, 10^{-18}]^\top$. With this vector we may compute the quantities

$$y(k) = \frac{(1 + x(k)) - 1}{x(k)}, \; k = 1, 2, \ldots, 18,$$

where $x(k)$ are the elements of x, from

```
>> y = ((1 + x) - 1)./x
```

The result is

```
y =
 1.00000000000000
 1.00000000000000
 0.99999999999989
 0.99999999999989
 1.00000000000655
 0.99999999991773
 1.00000000058387
 0.00000000392253
 1.00000008274037
 1.00000008274037
 1.00000008274037
 1.00008890058234
 0.99920072216264
 0.99920072216264
 1.11022302462516
                0
                0
                0
```

Since the exact result is 1, we see that for $k = 15$ there is a relative error of 11 percent while for $k \geq 16$ the result is totally wrong with relative error of 100 percent.

The computation of the vector of ones

$$y(k) = \frac{(1 + x(k))^2 - 1 - 2x(k)}{x^2(k)}$$

by the command

```
>> y = ((1 + x).^2 - 1 - 2*x)./x.^2
```

will give

```
y =
 1.00000000000002
 1.00000000000006
 0.99999999969742
 0.99999999171889
 1.00000139299870
 0.99992436739695
 1.01087806660425
-1.215...
 ...
```

We see that for $k = 7$ the result is computed by 11 percent relative error, while for $k \geq 8$ the result is completely useless (for $k = 8$ even the sign is wrong and the relative error is 221 percent).

Similarly, the computation of the quantities

$$y(k) = \frac{(1 + x(k))^3 - 1 - 3x(k) - 3x^2(k)}{x^3(k)},$$

all equal to 1, by the corresponding command gives

```
y =
   1.00000000000035
   0.99999999991242
   0.99999950547556
   0.99987350828302
   1.10953927987086
-202.14...
 ...
```

The results are self-evident.

The reader should be able to explain the above results and even give the approximate value for k, where the computational catastrophe occurs.

These results may be impressive for somebody who sees them for the first time. Especially instructive is the fact that completely wrong results are displayed with 15 (wrong) digits. Of course, neither MATLAB, nor the user's computer are defective or guilty in any way. Simply this is the way the floating point arithmetic works in certain cases.

Another easy computations contaminated with large errors are described in Exercises 3.2 and 3.3.

3.7 Conclusions

The next issues are extracted from the first version of the paper [12].

In order to assess the accuracy of calculations and to be able to trust numerical results, condition and accuracy estimates should accompany computational procedures and must be included in the corresponding computer codes. Users must be aware of possible difficulties accompanying the computational process and must know how to avoid them. These issues should also become an essential part of the curriculum to teach scientists and engineers how to use and develop modern computational software.

It is, however, unfortunately common practice in industrial use to turn these facilities off, even though this service will warn the user of numerical methods about possible failure. *What is the reason for such irresponsible action?* One reason is that the computation of the estimates somewhat slows down the method but, as we have learned in many discussions with engineers in industry, the main reason is that the user does not know how to interpret these warnings. But it should be clear that the developers of numerical software introduce these estimates not for the purpose of bothering the user with mathematical over cautiones. The idea is to make the user aware of difficulties with the computational problem or the numerical method and it should be an essential part of the curriculum to teach scientists and engineers how to interpret and use these warnings.

3.8 Problems and exercises

Exercise 3.1 The base $e \simeq 2.7183$ of the natural logarithms is equal to the limes of the sequence (e_n), where

$$e_n := \left(1 + \frac{1}{n}\right)^n .$$

However, to compute approximately e as e_n for some large n is not simply a bad idea but a catastrophe. For example taking $n = 10^7$ we compute e in MATLAB with a relative error of order 10^{-7} which is a good result. But taking $n = 10^{16}$ we get the completely wrong result $e_n = 1$ with a relative error 0.63.

Try to explain what happened (Hint: to compute e use the MATLAB command `exp(1)`).

Exercise 3.2 Compute the expression

$$y = \left(\left(\left(\frac{2}{x} + x\right) - x\right) - \frac{1}{x}\right)$$

for $x = 10^{-k}$, $k = 1, 2, \ldots, 10$. First compose the 10–vector X with components $X(k) = 10^{-k}$ and then use the command

```
>> Y = X.*(((2./X + X) - X) - 1./X)
```

to compute the 10-vector Y. Compare with the exact answer $Y = [1, 1, \ldots, 1]$ and comment the results.

Exercise 3.3 Form the 5×5 matrix

```
>> A = gallery(5)
```

in MATLAB, and compute the vector $v \in \mathbb{C}^5$ of its eigenvalues by the command

```
>> v = eig(A)
```

The determinant of A, computed by `det(A)`, is zero. Hence v should have a zero element. But the elements of v are of order 0.04 and are far from zero. Explain the result (more details about matrix computations by MATLAB are given in part II of the book).

PART II

NUMERICAL LINEAR ALGEBRA

Chapter 4

Operations with matrices

4.1 Introductory remarks

In Part II of the book we consider the solution of the main problems of numerical linear algebra in machine arithmetic with rounding unit $\mathbf{u}$.

In this chapter we shall consider the influence of rounding errors on the performance of matrix operations in machine arithmetic. In this framework we consider the data as exact, the only source of errors being the machine operations themselves.

We recall that matrices are denoted by upper case (usually Latin) letters. The element of the matrix A in position (p, q) is denoted as $a_{p,q}$ (or briefly a_{pq}), or $A(p, q)$.

The vectors are denoted by lower (sometimes upper) case letters. The p–th element of the vector x is denoted as x_p or $x(p)$.

We stress that the notations `A(p,q)` and `x(p)` are used in MATLAB, Scilab and SYSLAB to denote the corresponding elements of the matrix A and the vector x.

We note finally that the use of the indexes `i` or `j` in MATLAB may cause problems since these letters are reserved for denoting the imaginary unit $\imath = \sqrt{-1}$. So we advise the reader to use other letters for indexes.

4.2 Rounding of matrices

First we shall recall some special notations. If A is a matrix with elements $A(k, l)$ then $|A|$ denotes the *matrix absolute value* (or the *matrix module*) of A which is the matrix of the size of A and with elements $|A(k, l)|$.

If A and B are matrices of the same size then the *component–wise inequality*

$$A \preceq B$$

means that $A(k,l) \leq B(k,l)$ for all k,l. Similarly, $A \prec B$ means that $A \preceq B$ and $A \neq B$. We note that elsewhere $A \prec B$ is used do denote the relations $A(k,l) < B(k,l)$ for all k,l.

Definition 4.1 *The matrix norm* $\|\cdot\|$ *is said to be* monotone *if* $\|A\| \leq \|B\|$ *whenever* $|A| \preceq |B|$.

We stress that the 1–norm

$$\|A\|_1 = \max_l \sum_k |A(k,l)|,$$

the ∞–norm

$$\|A\|_i nfty = \|A^\top\|_1 = \max_k \sum_l |A(k,l)|$$

and the F–norm

$$\|A\|_{\mathrm{F}} = \left(\sum_{k,l} |A(k,l)|^2 \right)^{1/2}$$

are monotone. The 2–norm, however, is not monotone as the next example shows.

Example 4.1 Consider the matrices

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ -1.1 & 1 \end{bmatrix}.$$

We have

$$|A| = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \prec |B| = \begin{bmatrix} 1 & 1 \\ 1.1 & 1 \end{bmatrix}$$

but $\|A\|_2 = 2 > \|B\|_2 = 1.5$.

Let A be a real matrix with elements $A(k,l)$ in the normal range of the machine arithmetic, i.e. either $A(k,l) = 0$ or

$$E_{\min} \leq |A(k,l)| \leq X_{\max}.$$

Since in general the elements of A are not machine numbers then they cannot be stored exactly in the computer memory. Thus in machine arithmetic instead of A we shall have the rounded matrix $\widehat{A} :=$ round(A) with elements $\widehat{A}(k,l)$ satisfying

$$\widehat{A}(k,l) = A(k,l)(1+\alpha_{k,l}), \tag{4.1}$$

where the quantities $\alpha_{k,l}$ are relatively small, namely $|\alpha_{k,l}| \leq \mathbf{u}$. It follows from (4.1) that the component–wise inequality

$$|\widehat{A} - A| \preceq \mathbf{u}|A| \tag{4.2}$$

is fulfilled. Thus for all monotone norms we also have

$$\|\widehat{A} - A\| \leq \mathbf{u}\|A\|. \tag{4.3}$$

However, the estimate (4.2) is more informative than (refwhA1) because its provides a bound on the rounding error of each element of the matrix.

4.3 Summation of matrices

Definition 4.2 *Let A and B be two matrices of the same size. Then their* sum *is the matrix $C = A + B$ with elements $C(k,l) = A(k,l) + B(k,l)$.*

In the summation of these matrices in machine arithmetics instead of the exact sum C we obtain a matrix $\widehat{C} = \mathrm{fl}(A+B)$ which in general is different from C. For the elements of $\widehat{C}$ it is fulfilled that

$$\widehat{C}(k,l) = C(k,l)(1+\gamma_{kl}), \quad |\gamma_{kl}| \leq \mathbf{u}.$$

Hence

$$|\widehat{C}(k,l) - C(k,l)| = |C(k,l)\gamma_{kl}| \leq \mathbf{u}|C(k,l)|$$

and

$$|\widehat{C} - C| \preceq \mathbf{u}|C|.$$

Hence for any monotone norm, and in particular for the 1-norm and the F-norm, we have

$$\|\mathrm{fl}(A+B) - (A+B)\| \leq \mathbf{u}\|A+B\|.$$

Therefore, if $A + B \neq 0$, then

$$\frac{\|\mathrm{fl}(A+B) - (A+B)\|}{\|A+B\|} \leq \mathbf{u}.$$

This estimate shows that *the summation of two matrices in machine arithmetic is performed with small relative error of the order of the rounding unit* $\mathbf{u}$.

Of course, if for some k, l the elements $A(k,l)$ and $-B(k,l)$ are close approximate numbers, then the relative error in the element $C(k,l) = A(k,l) - (-B(k,l))$ may be large but this is not due to the execution of the machine operation $A+B$ itself.

4.4 Multiplication of matrices

In this section we shall analyze the computation of the product of two matrices in machine arithmetic. There are several concepts for the product of matrices. We shall consider the standard multiplication of matrices according to the following definition.

Definition 4.3 *Let A be a matrix of size $m \times n$ and B be a matrix of size $n \times p$. Then the* product *of A and B is the $m \times p$ matrix $C = AB$ with elements $C(k,l) = \sum_{s=1}^{n} A(k,s)B(s,l)$.*

The standard product is *associative* in the sense that $(AB)C = A(BC)$ and we write ABC for any of this quantities. This product also satisfies the *distributive rules* $(A_1 + A_2)B = A_1B + A_2B$ and $A(B_1 + B_2) = AB_1 + AB_2$. In the above relations it is supposed that the sizes of the involved matrices allow the correct performance of the corresponding matrix operations.

Consider first two non–orthogonal vectors $a, b \in \mathbb{R}^n$. It may be shown that the computation of the scalar product

$$(a,b) := a^\top b = \sum_{k=1}^{n} a_k b_k$$

is done with a relative error, for which we have the estimate

$$\frac{|\mathrm{fl}(a^\top b) - a^\top b|}{|a^\top b|} \le \mathbf{u} \frac{n\|a\|\,\|b\|}{|a^\top b|}.$$

On modern computer platforms, however, and for most modern systems for numerical computations, the multiplier n in the above estimate can be replaced with 1 or with a constant, close to 1. As Parlett says, this multiplier reflects our abilities to estimate the error rather than the behavior of the actual error.

But nevertheless the relative error in the computation of the scalar product may be very large if the scalar product $a^\top b$ is small compared to $\|a\|\,\|b\|$ (when the vectors a and b are almost orthogonal), i.e. if

$$\frac{\|a\|\,\|b\|}{|a^\top b|} \gg 1. \tag{4.4}$$

However, there are situations when the computation of the scalar product is done with the maximum possible accuracy in the sense that

$$\frac{|\mathrm{fl}(a^\top b) - a^\top b|}{|a^\top b|} \le \mathbf{u}, \tag{4.5}$$

see also Section 4.5.

Using the estimate (4.4) it can be shown that, given two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ with $AB \neq 0$, their product is computed in machine arithmetic with a relative error satisfying the estimate

$$\frac{\|\mathrm{fl}(AB) - AB\|_\mathrm{F}}{\|AB\|_\mathrm{F}} \le \mathbf{u}\, n\, \frac{\|A\|_\mathrm{F}\|B\|_\mathrm{F}}{\|AB\|_\mathrm{F}}.$$

This relative error may be very large since the quantity

$$\frac{\|A\|_\mathrm{F}\|B\|_\mathrm{F}}{\|AB\|_\mathrm{F}}$$

is not bounded (the denominator $\|AB\|_\mathrm{F}$ can be arbitrarily small compared to $\|A\|_\mathrm{F}\|B\|_\mathrm{F}$).

Of course, if the computations are so organized that the scalar product is computed with maximum accuracy then the product of two matrices is also computed with maximum accuracy, with a relative error of order $\mathbf{u}$.

There is an important particular case when the machine product of two matrices is always computed with small relative error. Let for example the matrix $A = [a_1, a_2, \ldots, a_n]$ has orthonormed columns $a_k \in \mathbb{R}^m$, i.e., $\|a_k\| = 1$ and $a_k^\top a_l = 0$ for $k \neq l$ (see also Chapter 5). Then we have

$$\|A\|_\mathrm{F} = \sqrt{n}, \quad \|AB\|_\mathrm{F} = \|B\|_\mathrm{F}$$

and hence

$$\frac{\|\mathrm{fl}(AB) - AB\|_\mathrm{F}}{\|AB\|_\mathrm{F}} \le \mathbf{u}\, n^{3/2}.$$

This is a very good result which shows that the multiplication with an orthogonal matrix is not dangerous in machine arithmetic. This is reflected in the rule that *the numerically reliable matrix algorithms involve only orthogonal (unitary in the complex case) transformations.* A certain exception here is the Gauss elimination algorithm with pivoting for the solution of linear algebraic equations.

4.5 Maximum accuracy computations

An important aspect in modern numerical computations, developed in the last 30 years, is related to the computational algorithms which guarantee *the maximum achievable accuracy* in the particular machine arithmetic. If $y = f(x) \neq 0$ is the computational problem and $\widehat{y}$ is the result, computed in machine arithmetic with rounding unit $\mathbf{u}$, the answer is *with maximum achievable accuracy* if the relative error $\|y - \widehat{y}\|/\|y\|$ is of order $\mathbf{u}$. For example, such an algorithm guarantees that the computed scalar product $\mathrm{fl}(a^\top b) \in \mathbb{M}$ of two vectors $a, b \in \mathbb{R}^n$ is the closest to $a^\top b \in \mathbb{R}$ machine number. This result may not be improved since it corresponds to the rounding error in writing the number $a^\top b$ in the computer memory.

It may be observed that, up to now, algorithms with maximum accuracy are able to solve a number of relatively simple problems (such as the scalar product of two vectors or the product of two matrices) with errors, comparable with the errors in rounding the scalar results to the nearest machine numbers. However, the development of such algorithms is a recent tendency in numerical analysis.

On the other hand, it is known that in modern personal computers the effect of maximum accuracy is achieved in an almost automatic manner since the arithmetic operations in the processor are done with much higher precision than the corresponding overall machine precision suggests. More details are given in [27].

4.6 Matrix operations by MATLAB

We recall that the symbols `i` and `j` are reserved in MATLAB for the imaginary unit $\imath = \sqrt{-1}$ and we shall avoid using them as indexes. Instead, we may use `k` and `l` or `p` and `q`. We also note that MATLAB is *case sensitive*, i.e. the system makes a difference between the quantities denoted as `A` and `a`. In contrast, SYSLAB is case insensitive.

If we have an $m \times n$ matrix A, the command

```
>> size(A)
```

will return the pair `m, n`. Hence, if x is a row n–vector, we shall have `size(x) = 1, n`. If x is a column n–vector, then the command `size(x)` will give the pair `n, 1`. For vectors we have also the command

```
>> length(x)
```

which gives `n` for both row and column n–vectors x.

Note that the symbol `>>` is generated automatically at the beginning of the command line in the command window of MATLAB[1], and that in various versions of MATLAB there may be different symbols for beginning of the command row.

Vectors and matrices may be defined in MATLAB in different ways.

Example 4.2 The commands

```
>> for p=1:5, x(p)=p^2; end
>> x
```

will produce and display the vector

```
x =
     1    4    9    16    25
```

with elements $x_p = x(p) = p^2$, $p = 1, 2, 3, 4, 5$.

Similarly, the commands

```
>> for k=1:10, for l=1:10, A(k,l)=1/(k+l-1);
     end
         end
```

will generate the 10×10 Hilbert matrix A with elements $A(k,l) = 1/(k+l-1)$, $k, l = 1, 2, \ldots, 10$.

The command

```
>> x = [a:h:b]
```

will produce a row vector x with first element $x(1) = a$, second element $x(2) = a+h$, etc. The last element of x, of the form $a+nh$, is the number with modulus not exceeding $|b|$ and closest to b. If $a = b$ the result shall be $x = a$, If $a < b$ and $h < 0$ the result will be an empty matrix (of size 1 by 0). The same result will be obtained for $h = 0$.

Example 4.3 The command

```
>> x = [1:0.3:2]
```

[1]Further on this symbol may be omitted.

will produce the vector

```
x =
    1.0000   1.3000   1.6000   1.9000
```

Also, we have

```
>> [2:-0.4:-1]
ans =
   2.0000  1.6000  1.2000  0.8000  0.4000  0.0000  -0.4000  -0.8000
```

The integer n–vector x with elements $1, 2, \ldots, n$ may also be produced by the simple command

```
>> x = 1:n
```

Another way to define vectors and matrices is to input the elements of the corresponding array one by one.

Example 4.4 The commands

```
>> A = [1 -3 0; pi 4 sqrt(2)]
```

will give the 2×3 matrix A,

```
A =
     1.0000   -3.0000         0
     3.1416    4.0000   1.4142
```

Note that we put a space between the elements of A, and semicolon ; for the beginning of the next row of A. The same effect is achieved by using commas for distinguishing between elements as in the command

```
>> A = [1,-3,0;pi,4,sqrt(2)]
```

After the commands

```
>> format long
>> A
```

the matrix A will be displayed within 15 decimal digits.

Arrays can be read also from outer files, and matrices may be constructed also by blocks.

Example 4.5 Suppose that we already have four matrices A_{11}, A_{12}, A_{21} and A_{22} of sizes $m\times n$, $m\times p$, $q\times n$ and $q\times p$, respectively . Then the $(m+q)\times(n+p)$–matrix

$$A := \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

may be constructed as

```
>> A = [A11, A12; A21, A22]
```

Once a matrix A or a vector x are defined, the commands

```
>> A(k,l)
>> x(p)
```

will produce the element of A in position (k,l) and the element of x in position p, respectively.

The commands

```
>> A(:,q)
>> A(p,:)
```

will give the q–th column and p–th row of A, respectively.

Similarly, the command

```
>> A(p:k,q:l)
```

returns the submatrix of A with elements at the crossing of rows $p, p+1, \ldots, k$ and columns $q, q+1, \ldots, l$, of A.

If the $m \times n$ matrix A is real, the command

```
>> A'
```

will produce the transposed $n \times m$ matrix $A^\top$ with elements $A^\top(k,l) = A(l,k)$. But if the matrix A is complex, the same command will give the complex conjugate (or Hermite conjugate) $n\times m$ matrix $A^{\mathrm{H}} = \overline{A}^\top$ with elements $A^{\mathrm{H}}(k,l) = \overline{A(l,k)}$.

Example 4.6 If x and y are column vectors, the command

```
>> z = [x',y']'
```

will return the vector $z = \begin{bmatrix} x \\ y \end{bmatrix}$ with `length(z) = length(x) + length(y)`.

In order to find the transpose $A^\top$ of a complex matrix A one may use the command

```
>> A.'
```

If A is a matrix with elements $A(k,l)$, the commands

```
>> triu(A)
>> tril(A)
```

will produce the upper triangular part and lower triangular parts of A, respectively. In particular, `triu(A)` is a matrix of the size of A and elements $A(k,l)$ if $k \le l$ and zero otherwise.

The command

```
>> diag(x)
```

where $x = [x_1, x_2, \ldots, x_n]$ is a vector, produces the diagonal $n \times n$ matrix $\mathrm{diag}(x_1, x_2, \ldots, x_n)$ with the elements of x on its main diagonal. If A is an $m \times n$ matrix, then

```
>> diag(A)
```

gives the k–vector of the diagonal elements of A, where $k := \min\{m,n\}$.

Example 4.7 For a square matrix A the command

```
>> triu(A) + tril(A) - diag(diag(A))
```

will return the matrix A (prove!).

The summation of two matrices, or arrays A and B of the same size is performed in MATLAB simply as

```
>> A + B
```

The standard multiplication of the matrices A and B is done by the command

```
>> A*B
```

Of course, here the number of columns of the matrix A must be equal to the number of rows of the matrix B.

If A and B are arrays of the same size, we may define their *H'Adamard product* $C = A \circ B$ from

```
>> C = A.*B
```

We recall that $C = A \circ B$ is a matrix of the size of A and B with elements $C(k,l) = A(k,l)B(k,l)$, i.e. the H'Adamard multiplication is done element–wise.

The *dot notation* above is typical for array operations. However, for summation and subtraction the dot notation is not necessary since the operation $\pm$ is done automatically in MATLAB. If e.g. B is of the size of A and has no zero elements, we may define the array

```
>> C = A./B
```

with elements $C(k,l) = A(k,l)/B(k,l)$.

In the above considerations the matrix sizes must agree in order to guarantee the correct performance of the corresponding matrix operations. If this is not the case MATLAB will return an error message of the type `Error using` (here follows the description of the operation such as $+$, $-$, $*$, etc.) `Matrix dimensions must agree`.

We stress that in MATLAB the term *dimension* is sometimes used instead of *size* for matrices and vectors. Mathematically this may be arguable since the dimension is an attribute of a subspace rather than of a vector or a matrix.

In MATLAB there are some "strange" matrix operations. If A is a matrix with elements $A(k,l)$ and a is a scalar, then the operation

```
>> B = A + a
```

produces a matrix B of the size of A and with elements $B(k,l) = A(k,l) + a$. Also, if a is a scalar, and A has only nonzero elements, then the command

```
>> B = a./A
```

gives a matrix B of the size of A with elements $B(k,l) = a/A(k,l)$.

If A is a square matrix and $m \in \mathbb{N}$, we may define the *matrix power* A^m inductively by $A^m := AA^{m-1}$, where A^0 is the identity matrix. In MATLAB A^m may be computed simply from

```
>> A^m
```

Note, however, that for an arbitrary matrix A (which may not even be square), the command

```
>> A.^m
```

will produce the matrix of the size of A and with elements $(A(k,l))^m$.

The *scalar product* (or *dot product*)

$$x^{\mathrm{H}}y = \sum_{k=1}^{n} \overline{x_k} y_k$$

of the vectors $x, y \in \mathbb{C}^n$ is computed by the command

```
>> dot(x,y)
```

Note that in the complex case the order of the vectors x, y in `dot(x,y)` is important. And some authors define the dot product of x and y as $y^{\mathrm{H}}x$, which is the complex conjugate to $x^{\mathrm{H}}y$.

There is a command in MATLAB for computation of the *Kronecker* (or *tensor*) *product* $C = A \otimes B$ of the matrices A and B of arbitrary sizes, namely

```
>> C = kron(A,B)
```

We recall that if A is $m \times n$ and B is $k \times l$ then

$$A \otimes B \in \mathbb{C}^{mk \times nl}$$

is an $m \times n$ block matrix with blocks of size $k \times l$, where the block in position (p, q) is $A(p, q)B$.

For 3–vectors x and y with elements x_k and y_k, we may find the so called *vector*, or *cross product* $x \times y$ by the command

```
>> z = cross(x,y)
```

We recall that

$$z = [x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1]^{\top}.$$

There are special commands for generating some special arrays. Let m and n be positive integers. The command

```
>> eye(n)
```

produces the identity $n \times n$ matrix $I_n = \mathrm{diag}(1, 1, \ldots, 1)$. The command

```
>> eye(m,n)
```

creates an $m \times n$ matrix with ones on the main diagonal and zeros otherwise, e.g.

```
>> eye(2,3)
ans =
      1     0     0
      0     1     0
```

Obviously, `eye(n,n)` is the same as `eye(n)`.

The commands

```
>> zeros(m,n)
>> ones(m,n)
```

form $m \times n$ matrices of zeros and ones, respectively. When $m = n$ one may use the short commands `zeros(n)` and `ones(n)`.

Example 4.8 If the matrix A is $m \times n$, then the commands

```
>> eye(size(A))
>> zeros(size(A))
>> ones(size(A))
```

will return a diagonal $m \times n$ matrix with ones on the diagonal, an $m \times n$ zero matrix and an $m \times n$ matrix of ones, respectively.

Some other specific matrices are generated as follows.

The command

```
>> compan(p)
```

where $p = [p_1, p_2, \ldots, p_{n+1}]$ is a real or complex $(n+1)$-vector, creates the companion matrix of the polynomial

$$P(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}.$$

We stress that in MATLAB an n–th degree polynomial P is identified with the $(n+1)$–vector p of its coefficients with $p(1) \neq 0$. A vector $[0, 0, \ldots, 0, a, b, \ldots, c]$ with several zero elements in first positions and $a \neq 0$, defines the same polynomial as the vector $[a, b, \ldots, c]$.

The command

```
>> hadamard(n)
```

generates a Hadamard $n \times n$ matrix H. The elements of H are -1 or 1 and H satisfies $H^\top H = nI_n$. It is supposed that n, $n/12$ or $n/20$ is a power of 2.

Hankel and Toeplitz matrices, having many applications, are defined and constructed in MATLAB as follows.

An $n \times n$ *Hankel matrix* H_n is symmetric and constant across its anti–diagonals. The Hankel matrix H_n is determined by its first column c and its last row r (with $c(n) = r(1)$). Hence there is a $(2n-1)$–vector h, such that the elements of H_n are determined by $H_n(k,l) = h(k+l-1)$ for $k,l = 1,2,\dots,n$.

Example 4.9 The Hankel 4×4 matrices have the form

$$H_4 = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \\ h_2 & h_3 & h_4 & h_5 \\ h_3 & h_4 & h_5 & h_6 \\ h_4 & h_5 & h_6 & h_7 \end{bmatrix}$$

The command

```
>> H = hankel(c)
```

creates a Hankel matrix whose first column is c and whose elements below its main anti–diagonal are zero. In Example 4.9 this will be the case with $h_5 = h_6 = h_7 = 0$.

The command

```
>> H = hankel(c,r)
```

creates a Hankel matrix with first column c and with last row r (note that the last element of c and the first element of r must be equal).

An $n \times n$ *Toeplitz matrix* T_n is, in general, non–symmetric and constant across its diagonals. The Toeplitz matrix T_n is determined by its first column c and its first row r (with $c(1) = r(1)$).

Hence there is a $(2n-1)$–vector t, such that the elements of T_n are determined by $T_n(k,l) = t(k-l+1)$ for $k \geq l$ and $T_n(k,l) = t(n+l-k)$ for $k < l$ and $k,l = 1,2,\dots,n$.

Example 4.10 The Toeplitz 4×4 matrices have the form

$$T_4 = \begin{bmatrix} t_1 & t_5 & t_6 & t_7 \\ t_2 & t_1 & t_5 & t_6 \\ t_3 & t_2 & t_1 & t_5 \\ t_4 & t_3 & t_2 & t_1 \end{bmatrix}$$

The command

```
>> T = toeplitz(c,r)
```

returns a Toeplitz matrix with first column c and with first row r (note that the first element of c and the first element of r must be equal).

The command

```
>> T = toeplitz(r)
```

returns a real symmetric Toeplitz matrix with first row r when r is real. When r is a complex vector, the matrix `toeplitz(r)` will also be complex and Hermitian.

The command

```
>> hilb(n)
```

produces the Hilbert $n \times n$ matrix with elements $1/(k+l-1)$ in position (k,l). The inverse Hilbert $n \times n$ matrix is created by the command

```
>> invhilb(n)
```

The reader is advised to create Hilbert and inverse Hilbert matrices in the rational format `format rat`.

The command

```
>> pascal(n)
```

produces the Pascal $n \times n$ integer matrix with elements from the Pascal triangle.

The command

```
>> V = vander(c)
```

where c is a column n–vector, creates the $n \times n$ Wandermonde matrix V_n whose $(n-1)$–th column is c (the elements of the n–th column of V_n are all equal to 1). Thus the elements of V are $V(k,l) = (c(l))^{n-k}$ for $k,l = 1,2,\ldots,n$.

The Wilkinson eigenvalue test matrix is produced by the command `wilkinson`.

The command

```
>> magic(n)
```

generates a magic $(n \times n)$–square. This is an integer matrix with pairwise disjoint elements from 1 to n^2 with row, column and diagonal sums all equal to $n(n^2+1)/2$. Note that n must not be 2.

Example 4.11 The command `magic(2)` will produce the matrix $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$ with column sums, equal to 5. But this is not a magic square since row sums are 4 and 6, while diagonal sums are 3 and 7 (as we mentioned above, the magic square cannot be 2×2).

For $n = 3$ and $n = 4$ we shall obtain the magic squares

```
>> magic(3)
ans =
   8    1    6
   3    5    7
   4    9    2
```

and

```
>> magic(4)
ans =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
```

For numerical experiments and other purposes it is useful to have dense matrices of arbitrary dimensions. The command

```
>> A = rand(m,n)
```

will produce an $m \times n$ matrix A with elements $A(k, l)$ randomly distributed between 0 and 1. The short command

```
>> A = rand(n)
```

will create a random $n \times n$ matrix A. Now the reader can easily construct matrices with random elements between any two numbers a and b.

Random matrices with normally distributed elements may be generated by the functions

```
>> A = randn(m,n)
>> A = randn(n)
```

which are similar to the functions `rand(m,n)` and `rand(n)`, respectively.

MATLAB contains a gallery of interesting matrix examples developed by N. Higham. The reader may find a description of the gallery by the help command `help gallery`.

The maximum and minimum elements of the real vector x may be found by the commands

```
>> max(x)
>> min(x)
```

respectively.

The command

```
>> y = abs(x)
```

returns the vector $y = |x|$ whose elements $y(p)$ are the absolute values $|x(p)|$ of the elements of x (here x may be a complex vector).

Let A be an $m \times n$ matrix, and let k and l be two positive integers with $kl = mn$. Then the command

```
>> reshape(A,k,l)
```

will return the $k \times l$ matrix whose elements are taken column–wise from matrix A. If $kl \neq mn$ the program will give an error message. Hence the column–wise vector representation $a := \mathrm{vec}(A)$ of A, which is an mn–vector, may be formed by the command

```
>> reshape(A,m*n,1)
```

or, automatically, by the somehow cumbersome function

```
>> reshape(A,(size(A))(1)*(size(A))(2),1)
```

The command

```
>> linspace(a,b)
```

produces a row 100–vector of linearly equally spaced points between the numbers a and b. The command

```
>> linspace(a,b,n)
```

creates a similar row n–vector.

The command

```
>> logspace(a,b)
```

produces a row 50–vector of logarithmically spaced points between 10^a and 10^b. The command

```
>> logspace(a,b,n)
```

returns a similar row n–vector.

4.7 Problems and exercises

Exercise 4.1 Create Hilbert and inverse Hilbert matrices H_n and H_n^{-1} for $n = 5, \ldots, 15$. Check whether the product of the computed quantities $\widehat{H}_n$ and $\widehat{H}_n^{-1}$ is the identity matrix I_n.

Exercise 4.2 Generate Pascal matrices P_n of orders $n = 3, 4, \ldots, 10$. Compute their determinants $\det(P_n)$ by the command `det` and their inverses P_n^{-1} by the command `inv`.

Exercise 4.3 Write a program which generates k random $m \times n$ matrices whose elements are: (i) uniformly, and (ii) normally distributed random variables in the interval $[a, b]$ (here the integers k, m, n and the reals a, b must be input data).

Exercise 4.4 Generate a sequence of n–vectors c for a fixed $n \geq 2$ with pair wise disjoint elements. Find the determinants `det(vander(n))` of the corresponding Vandermonde matrices V_n formed by `vander(c)` and whose $(n-1)$–th column is c. Formulate and prove a hypothesis about $\det(V_n)$ as a function of the elements of the vector c.

Chapter 5

Orthogonal and unitary matrices

5.1 Introductory remarks

In this chapter we recall the basic facts about real orthogonal and complex unitary matrices. These matrices play a fundamental role in the computational algorithms of modern numerical linear algebra.

5.2 Main definitions and results

In this section we give the definitions and main properties of orthogonal and unitary matrices.

Definition 5.1 *A real $n \times n$ matrix $U = [u_1, u_2, \dots, u_n] \in \mathbb{R}^{n\times n}$, $u_k \in \mathbb{R}^n$, is called* orthogonal *if*

$$U^\top U = I_n.$$

We note that this term is not very appropriate since orthogonality is a relation between two objects.

It follows from the above definition that $UU^\top = I_n$ (prove this!).

The columns u_p of an orthogonal matrix are orthonormed (orthogonal and normed) in the sense that $u_p^\top u_q = 0$ for $p \neq q$ and $u_p^\top u_p = \|u_p\|^2 = 1$.

According to Definition 5.1 an orthogonal matrix U is non–singular and its inverse is

$$U^{-1} = U^\top.$$

Thus the inversion of an orthogonal matrix requires no arithmetic operations and is done exactly in machine arithmetic. This is very important from numerical point of view.

The multiplication of a real vector x by an orthogonal matrix U rotates the vector without changing its 2–norm. Indeed, setting $y = Ux$ we see that $y^\top y = x^\top U^\top U x = x^\top x$ and hence $\|y\|_2 = \|x\|_2$.

Also, it follows from the definition of orthogonality that the elements $U(k,l)$ of U satisfy

$$-1 \le U(k,l) \le 1,$$

and that

$$\|U\|_2 = 1, \ \|U\|_{\mathrm{F}} = \sqrt{n}\,. \tag{5.1}$$

The matrix $A \in \mathbb{R}^{m\times n}$ has orthonormed columns if $A^\top A = I_n$. In this case by necessity $\operatorname{rank}(A) = n \le m$.

Similarly, the matrix $A \in \mathbb{R}^{m\times n}$ has orthonormed rows if the matrix $A^\top \in \mathbb{R}^{n\times m}$ has orthonormed columns, i.e. if $AA^\top = I_m$. Here $\operatorname{rank}(A) = m \le n$.

For real orthogonal matrices it follows from $I_n = U^\top U$ that $\det(I_n) = \det(U^\top U)$ and hence

$$1 = \det(U^\top)\det(U) = (\det(U))^2.$$

Therefore $\det(U) = 1$ or $\det(U) = -1$.

Definition 5.2 *Orthogonal matrices with determinant 1 are called* (pure) rotations*, while these with determinant -1 are called* reflections*.*

In the complex case we may define both orthogonal matrices and unitary matrices.

The $n \times n$ *complex orthogonal matrices* (again a name that is not very appropriate) are defined as in the real case by $U^\top U = I_n$. Note that for a complex nonzero vector $u \in \mathbb{C}^n$ we may have $u^\top u = 0$, e.g. $u = [1, \imath]^\top$, where $\imath = \sqrt{-1}$. However, in the complex case unitary matrices are usually more interesting from practical point of view than orthogonal matrices.

Definition 5.3 *The matrix $U \in \mathbb{C}^{n\times n}$ is called* unitary *if*

$$U^{\mathrm{H}} U = I_n.$$

We recall that $U^{\mathrm{H}} = \overline{U}^{\top}$, i.e. $U^{\mathrm{H}}(p,q) = \overline{U(q,p)}$.

The columns u_p of an unitary matrix U satisfy $u_p^{\mathrm{H}} u_q = 0$ for $p \neq q$ and $u_p^{\mathrm{H}} u_p = \|u_p\|^2 = 1$.

It follows from Definition 5.3 that an unitary matrix $U \in \mathbb{C}^{n\times n}$ is nonsingular with

$$U^{-1} = U^{\mathrm{H}}$$

and its spectral and Frobenius norms satisfy (5.1). Also we have $|U(k,l)| \leq 1$.

Similarly to the real case, the multiplication of a (complex) vector by an unitary matrix preserves the 2–norm of the vector. A generalization of this property is given below.

If the matrix U is unitary we have $|\det(U)|^2 = 1$ and hence the quantity $\det(U)$ lies on the unit circle in the complex plane $\mathbb{C}$.

Real orthogonal (or complex unitary) matrices can be used in order to transform real (or complex) vectors and matrices into simpler condensed form which contains zeros in prescribed positions.

These matrices are very favorable for numerical computations. Let U, V be unitary matrices and A be an arbitrary matrix of appropriate size. Then

$$\begin{aligned} \|UAV\|_2 &= \|A\|_2, \\ \|UAV\|_{\mathrm{F}} &= \|A\|_{\mathrm{F}}. \end{aligned}$$

Hence unitary transformations of the form $A \mapsto UAV$ preserve the 2–norm and the Frobenius norm of the matrix A.

Let the complex $n \times n$ matrix $U = U_0 + \imath U_1$ be unitary, where U_0, U_1 are real $n \times n$ matrices and $\imath = \sqrt{-1}$ is the imaginary unit. Since $U^{\mathrm{H}} = U_0^{\top} - \imath U_1^{\top}$, it follows from the unitarity of U that

$$\begin{aligned} & U_0^{\top} U_0 + U_1^{\top} U_1 = I_n, \\ & U_0^{\top} U_1 = U_1^{\top} U_0. \end{aligned}$$

Similarly, if the complex matrix U is orthogonal, we have

$$\begin{aligned} & U_0^{\top} U_0 - U_1^{\top} U_1 = I_n, \\ & U_0^{\top} U_1 + U_1^{\top} U_0 = 0. \end{aligned}$$

Real orthogonal and complex unitary matrices U are particular cases of the so called *normal* matrices A which are characterized by the property $A^{\top} A = AA^{\top}$ in the real case and $A^{\mathrm{H}} A = AA^{\mathrm{H}}$ in the complex case. Hence, given such a matrix U, there is a unitary matrix V such that the matrix $\Lambda := V^{\mathrm{H}}UV$ is diagonal with the eigenvalues λ_k of U on its diagonal. Since Λ is also orthogonal, we have $|\lambda_k| = 1$. Hence *the eigenvalues of U lie on the unit circle in the complex plane* $\mathbb{C}$. This is a more general property than the fact that $\det(U)$, being the product of the eigenvalues, lies on the unit circle.

Important classes of real orthogonal (or complex unitary) matrices are the elementary reflections and the elementary rotations. Elementary (or Householder) reflections are considered further on.

5.3 Orthogonal and unitary matrices in MATLAB

Let $A \in \mathbb{R}^{m\times n}$ be a matrix with $\mathrm{rank}(A) = r$. Then an important task is to construct an orthonormed basis $\{u_1, u_2, \ldots, u_r\} \subset \mathbb{R}^m$ for the *range*

$$\mathrm{Rg}(A) := \{Ax : x \in \mathbb{R}^n\} \subset \mathbb{R}^m$$

of this matrix.

The function

```
>> U = orth(A)
```

produces the matrix

$$U = [u_1, u_2, \ldots, u_r] \in \mathbb{R}^{m\times r}$$

with columns $u_p \in \mathbb{R}^m$ such that $u_p^\top u_q = \delta_{pq}$, where δ_{pq} is the Kronecker delta equal to 0 when $p \neq q$ and to 1 when $p = q$.

Example 5.1 The command

```
>> A = [1 1 2;2 2 4;3 3 -1]
A =
     1     1     2
     2     2     4
     3     3    -1
```

creates the matrix $A \in \mathbb{R}^{3\times 3}$ with $\mathrm{rank}(A) = 2$ since columns 1 and 2 of A coincide and columns 1 and 3 are linearly independent. This may be checked by the command

```
>> rank(A)
```

Now the command `orth(A)` will produce the orthonormed 3×2 matrix

```
ans =
   -0.3904   -0.2182
   -0.7807   -0.4364
   -0.4880    0.8729
```

with range, equal to $\mathrm{Rg}(A)$.

To produce (generically) an orthogonal $n \times n$ matrix U one may use the commands

```
>> A = rand(n); U = orth(A);
```

(the use of semicolons will prevent the display of the corresponding matrices). This approach works if the randomly generated matrix A is nonsingular which will almost surely be the case.

Another way to create a real orthogonal $n \times n$ matrix Q is to use the command sequence

```
>> A = rand(n); [Q,R] = qr(A)
```

Note that the second command above finds the QR decomposition $A = QR$ of the real randomly generated $n \times n$ matrix A.

A complex unitary matrix Q may be found from

```
>> A = rand(n) + i*rand(n); [Q,R] = qr(A)
```

Note that the last two commands `rand(n)` produce two different random $n \times n$ matrices.

Orthogonal and unitary matrices are widely used in MATLAB, Scilab and SYSLAB in order to ensure the reliability and accuracy of the corresponding computational algorithms.

5.4 Problems and exercises

Exercise 5.1 Show that any orthogonal matrix $U \in \mathbb{R}^{2\times 2}$ has one of the two forms

$$R := \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}, \; S := \begin{bmatrix} \cos\varphi & \sin\varphi \\ \sin\varphi & -\cos\varphi \end{bmatrix}$$

for some $\varphi \in [0, 2\pi)$. Find the determinants of R and S.

Exercise 5.2 Describe the set of all 2×2 complex unitary matrices.

Exercise 5.3 Show that the set of $n \times n$ complex orthogonal matrices is not bounded for $n > 1$.

Exercise 5.4 Let $U = U_0 + \imath U_1$ be a complex $n \times n$ matrix, where the matrices U_0, U_1 are real. Show that the real $(2n \times 2n)$–matrix $\begin{bmatrix} U_0 & -U_1 \\ U_1 & U_0 \end{bmatrix}$ is orthogonal if and only if the matrix U is unitary.

Chapter 6

Orthogonal and unitary matrix decompositions

6.1 Introductory remarks

In this chapter we consider two very useful orthogonal (unitary in the complex case) decompositions of a general matrix: the QR decomposition, or briefly QRD, and the singular value decomposition, or briefly SVD. Another important orthogonal decomposition of a square matrix – the Schur decomposition, is studied in Chapter 9.

6.2 Elementary unitary matrices

A general unitary matrix may be decomposed into a product of "elementary" unitary matrices. There are several types of matrices, considered as elementary. Among them are the plane (or Givens) rotations and the elementary (or Householder) reflections.

Definition 6.1 *An* elementary complex plane rotation *(or* Givens rotation*) is a unitary matrix, which differs from the identity matrix in at most four positions, occupied by the elements of a* 2×2 *unitary matrix and has determinant* 1.

More precisely, a *rotation* in the (p, q)-plane, $p < q$, is an $n \times n$ matrix R_{pq}, whose (k, l) elements r_{kl} are determined as follows. The 2×2 matrix

$$\begin{bmatrix} r_{pp} & r_{pq} \\ r_{qp} & r_{qq} \end{bmatrix} \tag{6.1}$$

is unitary and r_{kl} is the Kronecker delta if $\{k,l\} \cap \{p,q\} = \emptyset$.

An *elementary real plane rotation* is defined similarly. It is a real orthogonal matrix with the structure of R_{pq}, where the matrix (6.1) is orthogonal.

Another type of elementary unitary matrices are the elementary reflections. Let $u \in \mathbb{C}^n$ be a non-zero vector.

Definition 6.2 *The matrix*

$$H(u) := I_n - \frac{2uu^{\mathrm{H}}}{u^{\mathrm{H}}u} \in \mathbb{C}^{n\times n}$$

is said to be an elementary complex *(or* Householder*)* reflection*.*

It follows from this definition that

$$H(u) = H(\alpha u)$$

for each nonzero scalar α. The matrix $H = H(u)$ is both Hermitian and unitary,

$$H^{\mathrm{H}} = H,\ H^{\mathrm{H}}H = I_n.$$

Elementary real reflections are defined similarly as

$$H(v) := I_n - \frac{2vv^{\top}}{v^{\top}v} \in \mathbb{R}^{n\times n},\ 0 \neq v \in \mathbb{R}^n.$$

The matrix $H = H(v)$ is both symmetric and orthogonal,

$$H^{\top} = H,\ H^{\top}H = I_n.$$

The multiplication of a vector $x \in \mathbb{C}^n$ by a reflection $H(u)$ is reduced to the calculation of a single scalar product $u^{\mathrm{H}}x$, multiplication of a vector by a scalar and substraction of two vectors according to

$$H(u)x = x - \left(\frac{2u^{\mathrm{H}}x}{u^{\mathrm{H}}u}\right)u.$$

In particular we have

$$H(u)u = u - \left(\frac{2u^{\mathrm{H}}u}{u^{\mathrm{H}}u}\right)u = u - 2u = -u$$

and $\det(H(u)) = -1$.

Elementary reflections take their name from the fact that the mapping $x \mapsto H(u)x$ is a reflection relative to the one–dimensional subspace (hyperplane)

$$\mathrm{Ker}(u^{\mathrm{H}}) = \{x \in \mathbb{C}^n : u^{\mathrm{H}}x = 0\}.$$

Indeed, we have

$$u^{\mathrm{H}}(x + H(u)x) = 2u^{\mathrm{H}}\left(x - \frac{uu^{\mathrm{H}}x}{u^{\mathrm{H}}u}\right) = 2(u^{\mathrm{H}}x - u^{\mathrm{H}}x) = 0.$$

Thus the vector u is orthogonal to the sum of x and its image $H(u)x$. Equivalently, u is collinear to the difference $x - H(u)x$ since x and $H(u)x$ are of equal 2–norm. Thus, a multiplication with $H(u)$ reflects any vector relative to $\mathrm{Ker}(u^{\mathrm{H}})$.

Based on the above discussion we have an elegant solution to the following problem. Given two different vectors $x, y \in \mathbb{C}^n$ of equal 2–norm, find a unitary matrix U, which transforms x into y, i.e. $y = Ux$. It follows from the reflection property of $H(u)$ that a solution of this problem is $U = H(x - y)$ since

$$H(x - y)x = y, \ \|x\|_2 = \|y\|_2 > 0.$$

It is often necessary to transform a non–zero vector x into a form with only one nonzero element in the k–th position. Suppose that the vector $x \in \mathbb{C}^n$ is not proportional to the k–th column e_k of the identity matrix I_n (if it is, there is nothing to transform). Let $\alpha \in \mathbb{C}$ and $|\alpha| = 1$. Then the required transformation is

$$H(x - y)x = y, \ y := \alpha\|x\|_2 e_k \neq x. \tag{6.2}$$

In the real case we have $\alpha = \pm 1$ and

$$H(x - y)x = y, \ y := \pm\|x\|_2 e_k. \tag{6.3}$$

The choice of α in (6.2), respectively of the sign in (6.3), is done from numerical considerations in order to avoid possible cancellations in subtracting close quantities. If the argument of x_k is φ_k, i.e. $x_k = \rho_k \exp(\imath\varphi_k)$, then we choose the argument of α as $\varphi_k + \pi$ which gives $\alpha = -\exp(\imath\varphi_k)$. In this way the k–th element of the vector $x - y$ becomes $(\rho_k + \|x\|_2)\exp(\imath\varphi_k)$.

If in the real case if x_k is non–negative (respectively negative) we choose $y = \|x\|_2 e_k$ (respectively $y = -\|x\|_2 e_k$).

Since the matrix $H(x \mp \|x\|_2 e_k)$ is both Hermitian and unitary we have

$$x = H(x - y)y = \alpha\|x\|_2 H(x - y)e_k = \alpha\|x\|_2 h_k,$$

where h_k is the k–th column of $H(x - y)$. Hence, a matrix $H(u)$ transforms a vector $x \neq 0$ into $\alpha\|x\|_2 e_k$ if and only if its k–th column h_k is collinear to x.

Now we may solve the following problem. Given an unit vector $x \in \mathbb{C}^n$ find an $n \times (n - 1)$ matrix V such that the matrix $U := [x, V]$ is unitary. If x is collinear to some column e_k of I_n, then V contains the other columns of I_n. Suppose now that x is not collinear to a column of I_n. Let $h_1, h_2, \ldots, h_n$ be the columns of the reflection $H(x \mp e_1)$ which transforms x into e_1. Then a solution of our problem is

$$V = [h_2, h_3, \ldots, h_n].$$

Indeed, in this case $x = \pm h_1$.

Using a sequence of reflections an $m \times n$ matrix A of rank $r \geq 1$ may be reduced to a condensed upper triangular form $R = Q^{\mathrm{H}} A$, where Q is a product of reflections (see the next section). Let $m_1 < m_2 < \cdots < m_r$ be the numbers of the first linearly independent columns of A. Then the matrix R with elements r_{kl} may be characterized as follows.

– The last $m - r$ rows (if any) of the matrix R are zero.

– the first non–zero element in the k–th row ($k \leq r$) of R is $r_{k,m_k} > 0$.

6.3 QR decomposition

Let $A \in \mathbb{C}^{m \times n}$ be an arbitrary matrix of rank $r := \operatorname{rank}(A) \geq 1$ (the case $A = 0$ is not interesting). Then, as shown above, there exist an unitary matrix $U \in \mathbb{C}^{m \times m}$ and an upper triangular matrix $R \in \mathbb{C}^{m \times n}$ such that

$$A = QR \tag{6.4}$$

Definition 6.3 *The representation (6.4) is known as* QR decomposition[1]*, or* unitary-triangular decomposition *of the matrix A.*

Partitioning the matrices Q and R as

$$Q = [Q_1, Q_2],\ Q_1 \in \mathbb{C}^{m \times r},\ R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},\ R_1 \in \mathbb{C}^{r \times m}$$

we may write the decomposition in the form

$$A = Q_1 R_1. \tag{6.5}$$

The relation (6.5) is the so called *economic QR decomposition.*

The QR decomposition is a very useful tool in numerical linear algebra. In particular it is used for solution of linear algebraic equations and least squares problems as well as for rank determination.

[1]Sometimes the QR decomposition is called QR *factorization.*

From computational point of view it is convenient to rearrange the columns of A so as the diagonal elements $R_1(k,k)$ of the matrix R_1 to satisfy the inequalities

$$|R_1(1,1)| \geq |R_1(2,2)| \geq \cdots \geq |R_1(r,r)| > 0. \tag{6.6}$$

For this purpose a permutation matrix $E \in \mathbb{R}^{n\times n}$ is constructed so that the decomposition

$$AE = QR$$

to satisfy (6.6).

The QR decomposition of A is constructed in $r-1$ steps. Suppose for definiteness that $r = n \leq m$.

At the first step we construct a Householder reflection $Q_1 := H_1 \in \mathbb{C}^{m\times m}$ such that $H_1 a_1 = [\times_1, 0, \ldots, 0]^\top$, where $a_1 \in \mathbb{C}^m$ is the first column of A and $\times_1$ is the only nonzero element of the transformed vector $H_1 a_1$.

At the second step we set $Q_2 := \operatorname{diag}(1, H_2)$, where H_2 is an $(m-1)\times(m-1)$ reflection such that $H_2 a_2 = [\times_2, 0, \ldots, 0]^\top$. Here a_2 is the vector with elements $A_2(2,2), A_2(3,2), \ldots, A_2(m,2)$, where $A_2 := Q_1 A_1$.

We may continue this process until we obtain the matrix $A_{r-1} = R$.

6.4 Singular value decomposition

For any matrix $A \in \mathbb{C}^{m\times n}$ of rank $r := \operatorname{rank}(A) \geq 1$ there exist two unitary matrices $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$ and a diagonal matrix $S \in \mathbb{C}^{m\times n}$ such that

$$A = USV^{\mathrm{H}}. \tag{6.7}$$

Here

$$S := \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma := \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \in \mathbb{R}^{r\times r}.$$

Definition 6.4 *The numbers*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$$

are said to be the positive singular values *of the matrix A. The columns $v_1, v_2, \ldots, v_n \in \mathbb{C}^n$ of V are called the (right)* singular vectors *of A. The representation (6.7) is known as the* singular value decomposition*, or briefly* SVD*, of the matrix A.*

It may be shown that the positive singular values of A are the positive square roots of the positive eigenvalues of the Hermitian matrix $A^{\mathrm{H}}A$ (or AA^{H}).

The singular value decomposition may also be written as

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^{\mathrm{H}},$$

where $u_1, u_2, \ldots, u_m \in \mathbb{C}^m$ are the columns of the matrix U. We see that here the matrix A of rank r is represented as the sum of r rank–one matrices $\sigma_k u_k v_k^{\mathrm{H}}$.

When the matrix A is real, the matrices U and V may also be chosen real and in this case they are orthogonal matrices.

The SVD is one of the most useful tools in numerical linear algebra. In particular, the SVD is applicable to the rank determination of a matrix and to the solution of linear algebraic equations and least squares problems.

Another useful decomposition involves a pair of matrices with equal number of columns. Let $A \in \mathbb{C}^{m\times p}$ and $B \in \mathbb{C}^{n\times p}$. Then there exist unitary matrices $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$ ($U^{\mathrm{H}}U = I_m$, $V^{\mathrm{H}}V = I_n$), a matrix $W \in \mathbb{C}^{p\times q}$ with $q := \min\{m+n, p\}$ and non–negative diagonal matrices $C \in \mathbb{C}^{m\times q}$, $D \in \mathbb{C}^{n\times q}$, such that

$$A = UCW^{\mathrm{H}}, \quad B = VSW^{\mathrm{H}}, \tag{6.8}$$

where

$$C^{\top}C + S^{\top}S = I_q. \tag{6.9}$$

Definition 6.5 *The relations (6.8), (6.9) define the* generalized singular value decomposition*, or briefly* GSVD*, of the matrix pair* (A, B)*.*

6.5 Orthogonal and unitary decompositions by MATLAB

We recall that the orthogonal–triangular decomposition, known as QR decomposition, of the matrix $A \in \mathbb{C}^{m\times n}$, is a factorization of A of the form

$$A = QR,$$

where the matrix $Q \in \mathbb{C}^{m\times m}$ is unitary, i.e. $Q^{\mathrm{H}}Q = I_m$, and the matrix $R \in \mathbb{C}^{m\times n}$ is upper triangular. When the matrix A is real, the matrix Q is real orthogonal ($Q^{\top}Q = I_m$).

The command

```
>> qr(A)
```

returns only the matrix R. The command

```
>> [Q,R] = qr(A)
```

computes the matrices Q and R, while the command

```
>> [Q,R,E] = qr(A)
```

produces a permutation matrix $E \in \mathbb{R}^{n\times n}$ and the matrices Q and R such that $AE = QR$. In this case the absolute values $|R(k,k)|$ of the diagonal elements $R(k,k)$ of R are not increasing with k.

When $m > n$ the function

```
>> [Q1,R1] = qr(A,0)
```

computes the "economy" QR decomposition in which $Q_1 \in \mathbb{C}^{m\times n}$ is a matrix with orthonormed columns.

The singular value decomposition

$$A = USV^{\mathrm{H}}$$

of the real or complex (non–zero) $m \times n$ matrix A is found by the command

```
>> [U,S,V] = svd(A)
```

Here the matrices $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary in the complex case, and real orthogonal in the real case. The matrix S is diagonal of size $m \times n$ with the singular values $\sigma_1, \sigma_2, \ldots, \sigma_p$ of A on its main diagonal in descending order, where $p = \min\{m,n\}$. When $r := \operatorname{rank}(A) < p$ we shall have $\sigma_r > 0$ and $\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_p = 0$.

When A is the zero matrix the command `[U,S,V] = svd(A)` will return $U = I_m$, $S = 0$ and $V = I_n$.

The short command

```
>> s = svd(A)
```

returns the p–vector s, containing the singular values of A in descending order.

If $m > n$ we may compute the "economy size" singular value decomposition

```
>> [U1,S1,V] = svd(A,0)
```

where U_1 contains the first n columns of U, and S_1 contains the first n rows of S.

The generalized singular value decomposition of a pair of matrices $A \in \mathbb{C}^{m\times p}$, $B \in \mathbb{C}^{n\times p}$ (with the same number of columns) may be found by the command

```
>> [U,V,W,C,S] = gsvd(A,B)
```

Here the matrices $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary, W is a $p \times q$ matrix with $q = \min\{m+n, p\}$, and $C \in \mathbb{C}^{m\times q}$, $S \in \mathbb{C}^{n\times q}$ are nonnegative diagonal matrices, such that

$$A = UCW^{\mathrm{H}},\ B = VSW^{\mathrm{H}},\ C^{\mathrm{H}}C + S^{\mathrm{H}}S = I_q.$$

The short command

```
>> s = gsvd(A,B)
```

returns the q–vector s of generalized singular values. In fact, we have

```
s = sqrt(diag(C'*C./diag(S'*S))
```

More details about the MATLAB functions `svd` and `gsvd` may be found, as usual, by the commands `help gsvd` and `help svd`.

6.6 Problems and exercises

Exercise 6.1 Generate a sequence of random small size matrices A and B with the same number of columns and find:
– the QR decompositions of A and B;
– the singular value decompositions of A and B;
– the generalized singular value decompositions of the pairs (A, B).

Exercise 6.2 Prove that the positive singular values of the nonzero matrix A are the positive square roots of the positive eigenvalues of the matrix $A^{\mathrm{H}}A$ (or AA^{H}).

Exercise 6.3 Generate a sequence of random square matrices by the command `An = rand(n)` for $n = 10, 50, 100, 500$. Most probably these matrices will be non–singular. Find the SVD `[Un,Sn,Vn] = svd(An)` of each matrix A_n. When A_n is non–singular then we should have $A_n^{-1} = V_n S_n^{-1} U_n^{\top}$. Compute the quantities `dn = norm(Bn - Cn)`, where `Bn = inv(An)` and `Cn = Vn*inv(Sn)*Un'`. Theoretically d_n must be zero. Investigate the behavior of d_n with the increase of n.

Chapter 7

Linear algebraic equations

7.1 Statement of the problem

In this chapter we shall use the Euclidean norm for vectors and, according to this, the spectral norm of matrices, i.e. we assume $\|\cdot\| = \|\cdot\|_2$. We recall that

$$\|x\|_2 := \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}$$

for a real or complex vector $x = [x_1, x_2, \ldots, x_n]^\top$, and

$$\|A\|_2 := \sqrt{\lambda_{\max}(A^{\mathrm{H}}A)}$$

for a matrix A, where $A^{\mathrm{H}} := \overline{A}^\top$ and $\lambda_{\max}(M)$ is the maximum eigenvalue of the non–negative definite matrix M.

Consider the solution of the real linear vector equation

$$Ax = b \tag{7.1}$$

in machine arithmetic with rounding unit $\mathbf{u}$. We suppose that $A \in \mathbb{R}^{n\times n}$ is a non–singular (invertible) matrix, $b \in \mathbb{R}^n$ is a given non–zero vector (otherwise the solution is trivially zero) and $x \in \mathbb{R}^n$ is the solution. The case when A, b and x may be complex quantities is studied in the same way.

The more general matrix equation

$$AX = B, \tag{7.2}$$

where $A \in \mathbb{R}^{n\times n}$, $B = [B_1, B_2, \ldots, B_m] \in \mathbb{R}^{n\times m}$, $B_k \in \mathbb{R}^n$, and $X = [X_1, X_2, \ldots, X_m] \in \mathbb{R}^{n\times m}$, $X_k \in \mathbb{R}^n$, is reduced to m copies

$$AX_k = B_k,\ k = 1, 2, \ldots, m,$$

of equation (7.1).

Sometimes the row–wise versions of the equations (7.1) and (7.2) is used, namely

$$yA = c \tag{7.3}$$

and

$$YA = C, \tag{7.4}$$

where A is an $n \times n$ invertible matrix as above, c and y are row n–vectors and Y, C are $m \times n$ matrices. If Y_k and C_k are the rows of Y and C, respectively, then equation (7.4) is reduced to m equations $Y_k A = C_k$, $k = 1, 2, \ldots, m$, of type (7.3).

Obviously, equations (7.3) and (7.4) may be written in the form (7.1) and (7.2), respectively, e.g. $A^\top y^\top = c^\top$ and $A^\top Y^\top = C^\top$.

The more general matrix linear equation

$$\sum_{k=1}^{r} M_k X N_k = B, \tag{7.5}$$

where

$$M_k \in \mathbb{R}^{m\times m},\ N_k \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{m\times n}$$

are given matrices and $X \in \mathbb{R}^{m\times n}$ is the solution, can be reduced formally to equation (7.1). Indeed, setting

$$A := \sum_{k=1}^{r} N_k^\top \otimes M_k \in \mathbb{R}^{mn\times mn},\ b := \mathrm{vec}(B) \in \mathbb{R}^{mn}$$

and $x := \mathrm{vec}(X) \in \mathbb{R}^{mn}$, we obtain an equation of type (7.1). Here $M \otimes N$ is the Kronecker product of the matrices M and N, while $\mathrm{vec}(B)$ is the column–wise vector representation of the matrix B.

Of course, the possibility to reduce equation (7.5) into the form (7.1) does not mean that this is the numerically correct way to solve this equation. Usually matrix equations of type (7.5) are solved directly, by special methods, and without reduction into vector form.

7.2 Formal solutions

Formally, the solution of equation (7.1) may be written as

$$x = A^{-1}b,$$

where $A^{-1} \in \mathbb{R}^{n\times n}$ is the inverse of the matrix $A \in \mathbb{R}^{n\times n}$. Similarly, the solution of equation (7.3) is

$$y = cA^{-1}.$$

However, these representations of the solutions are rarely used in practice because the calculation of A^{-1} and the computation of the products $A^{-1}b$ and cA^{-1} may not be accurate and/or efficient.

Moreover, the computation of the matrix $F := A^{-1}$ (if necessary for some purpose) is done by solving n algebraic equations

$$Af_k = e_k,\ \ k = 1, 2, \ldots, n,$$

for the columns $f_k \in \mathbb{C}^n$ of the matrix $F = [f_1, f_2, \ldots, f_n]$, where e_k is the k–th column of the identity matrix I_n.

There is also another elegant formal (but practically useless) way to represent the elements $x_1, x_2, \ldots, x_n$ of the solution vector. Indeed, we have

$$b = Ax = \sum_{l=1}^{n} x_l a_l, \tag{7.6}$$

where a_l are the columns of the matrix $A = [a_1, a_2, \ldots, a_n]$. For a given integer $k \in \overline{1, n}$ consider the matrix

$$A_k = A_k(b) := [a_1, \ldots, a_{k-1}, b, a_{k+1}, \ldots, a_n],$$

obtained by replacing the k–th column a_k of A by the vector b. In view of (7.6) we have

$$\begin{aligned} \det(A_k) &= \det\left[a_1, \ldots, a_{k-1}, \sum_{l=1}^{n} x_l a_l, a_{k+1}, \ldots, a_n\right] \\ &= \sum_{l=1}^{n} x_l \det[a_1, \ldots, a_{k-1}, a_l, a_{k+1}, \ldots, a_n] \\ &= x_k \det(A). \end{aligned}$$

Since $\det(A) \neq 0$ we get

$$x_k = \frac{\det(A_k)}{\det(A)},\ \ k = 1, 2, \ldots, n. \tag{7.7}$$

These are the *Cramer formulae* – a beautiful and useless (for $n > 3$) representation of the elements of the solution vector $x = [x_1, x_2, \ldots, x_n]^\top$.

Unfortunately, some people still think that this is the proper way to solve linear equations.

7.3 Sensitivity of the solution

Consider the sensitivity of the problem of solving equation (7.1). Let δA, δb be perturbations in the data A, b, such that

$$\|\delta A\| < \frac{1}{\|A^{-1}\|}.$$

This inequality guarantees that the perturbed matrix $A + \delta A$ is also invertible. Let $x + \delta x$ be the (unique) solution of the perturbed equation

$$(A + \delta A)(x + \delta x) = b + \delta b. \tag{7.8}$$

It follows from (7.8) that

$$\delta x = A^{-1}(\delta b - \delta A x) - A^{-1}\delta A \delta x$$

and

$$\|\delta x\| \leq \|A^{-1}\|(\|\delta b\| + \|\delta A\|\,\|x\|) + \|A^{-1}\|\|\delta A\|\,\|\delta x\|.$$

Since $\|A^{-1}\|\,\|\delta A\| < 1$ we have

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

The sensitivity estimate becomes particularly elegant when relative norm–wise perturbations are used,

$$\delta_x := \frac{\|\delta x\|}{\|x\|}, \quad \delta_A := \frac{\|\delta A\|}{\|A\|}, \quad \delta_b := \frac{\|\delta b\|}{\|b\|}.$$

We get

$$\delta_x \leq \frac{k_A(\eta\delta_b + \delta_A)}{1 - k_A\delta_A}, \tag{7.9}$$

where

$$k_A = \operatorname{cond}(A) := \|A\|\,\|A^{-1}\| = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

is the condition number of A relative to inversion in the spectral norm, and

$$\eta = \eta(A, b) := \frac{\|b\|}{\|A\|\,\|x\|} = \frac{\|b\|}{\|A\|\,\|A^{-1}b\|}.$$

It follows from $Ax = b$ that

$$\|b\| \leq \|A\|\,\|x\|$$

and hence $\eta \leq 1$. Thus we come to the simpler widely used but less accurate estimate

$$\delta_x \leq \frac{k_A(\delta_b + \delta_A)}{1 - k_A \delta_A}. \tag{7.10}$$

On the other hand we have

$$\|A^{-1}b\| \leq \|A^{-1}\| \, \|b\|$$

which yields

$$\frac{1}{\|A^{-1}b\|} \geq \frac{1}{\|A^{-1}\| \, \|b\|}$$

and $\eta \geq 1/k_A$. Therefore the following estimates for η are valid

$$\frac{1}{k_A} \leq \eta \leq 1.$$

The lower bound $1/k_A$ for η is achieved when b is a multiple of the right singular vector of the matrix A^{-1}, corresponding to its maximum singular value equal to $\|A^{-1}\|$. In this case it may not be a good idea to replace η with 1. Moreover, the estimate (7.9) with $\eta < 1$ can be arbitrarily many times better than the estimate (7.10), in which we have replaced η with 1. Unfortunately the improved estimate (7.9) is not very popular among common users.

Consider now the case when the perturbations are due to rounding errors when writing the data in computer memory. Then we may assume that

$$\delta_A, \delta_b \leq \mathbf{u}$$

and hence

$$\delta_x \leq \frac{k_A(1+\eta)\mathbf{u}}{1 - k_A \mathbf{u}}.$$

When $k_A\mathbf{u} \ll 1$ we get the approximate inequality

$$\delta_x \leq k_A(1+\eta)\mathbf{u}.$$

Thus the quantity

$$c_A := k_A(1+\eta) \leq 2k_A$$

can be regarded as an estimate of the relative condition number of the solution to the algebraic linear vector equation (7.1).

7.4 Computational methods

The most used methods for the numerical solution of equation (7.1) (in the medium size case when n is of order 1000) are the Gauss elimination method[1] and the method of QR decomposition. The Gauss elimination method is based on the so called LU decomposition of the matrix A, considered below.

In the Gauss method the non–singular matrix A or, more generally, the matrix EA, where E is a permutation matrix, is decomposed as

$$A = LU,$$

or

$$EA = LU,$$

where L is a lower triangular matrix with elements, equal to 1, on its main diagonal and U is an upper triangular matrix with non–zero diagonal elements. The permutation matrix E is found from the condition to have a maximum absolute value pivot element at each step of the reduction of A into upper triangular form. The last two decompositions are known as *lower–upper decompositions*, or briefly, *LU decompositions* of A.

The decomposition $EA = LU$ corresponds to the so called *elimination with partial pivoting*. The pivoting is very important in order to improve the numerical behavior of the Gauss elimination method.

Setting $y = Ux$ we may write the equation $Ax = b$ as

$$Ly = c, \quad c := Eb.$$

Since the matrix $L = [l_{pq}]$ is lower triangular with $l_{pp} = 1$, we have

$$\begin{aligned} &y_1 = c_1, \\ &l_{p1}y_1 + l_{p2}y_2 + \cdots + l_{p,p-1}y_{p-1} + y_p = c_p, \ p = 2, 3, \ldots, n, \end{aligned} \tag{7.11}$$

where y_p and c_p are the elements of the vectors y and c, respectively. Now the elements of the vector y are determined recursively from (7.11) starting with y_1.

Having the vector y, the solution x is found from the linear triangular system

$$Ux = y$$

starting from the last equation (see the QR decomposition method described below).

[1]This method had been used in China some 3,000 years ago for $n = 3$.

In the QR decomposition method the matrix A (or EA) is decomposed as

$$A = QR,$$

where the matrix Q is orthogonal and the matrix $R = [r_{pq}]$ is upper triangular with nonzero diagonal elements. Thus x is obtained from the triangular system

$$Rx = c, \quad c := Q^\top b.$$

First we solve the last scalar equation $r_{nn}x_n = c_n$, obtaining $x_n = c_n/r_{nn}$. Then we solve the $(n-1)$–th equation

$$r_{n-1,n-1}x_{n-1} + r_{n-1,n}x_n = c_{n-1},$$

obtaining

$$x_{n-1} = \frac{c_{n-1} - r_{n-1,n}x_n}{r_{n-1,n-1}}.$$

At the $(n+1-k)$–th step we obtain

$$x_k = \frac{1}{r_{kk}}\left(c_k - \sum_{p=k+1}^{n} r_{kp}x_p\right),$$

where the quantities $x_n, x_{n-1}, \ldots, x_{k+1}$ are already computed. This process continues until we compute x_1 at the n–th step.

Another method to solve the linear equation

$$Ax = b,$$

applicable to singular and rectangular matrices A, is the reduction of A into the so called *row echelon form*.

Let A be an $m \times n$ (non–zero) matrix. Then there is a permutation matrix $E \in \mathbb{R}^{m\times m}$, a lower triangular matrix $L \in \mathbb{C}^{m\times m}$ with unit diagonal, and an $m \times n$ matrix R in row echelon form so that

$$EA = LR.$$

Here the matrix R of rank

$$r = \operatorname{rank}(R) = \operatorname{rank}(A) \le \min\{m, n\}$$

satisfies the following conditions.

1. The first r rows of R are linearly independent, while the rest $m - r$ rows (if any) are zero.

2. The first non–zero element of any non–zero row (called *pivot element*) is equal to 1. The rest of the elements of the column containing a pivot are all zeros.

3. The pivot of every row with number $p \in \overline{2, r}$ is placed to the right of the pivot of row $p-1$.

Example 7.1 The 4×5 matrix

$$R = \begin{bmatrix} 0 & 1 & 0 & \bullet & 0 \\ 0 & 0 & 1 & \bullet & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

of rank 3 is in row echelon form (we denote unspecified elements by $\bullet$).

Note that if the matrix $A \in \mathbb{C}^{n\times n}$ is non–singular, its row echelon form is simply the identity matrix I_n.

Some authors define the row echelon form $\widehat{R}$ in a less condensed manner: the pivot elements in $\widehat{R}$ are nonzero (positive in particular) and the elements in the column above the pilot may not be zero.

Example 7.2 The less condensed echelon form $\widehat{R}$ for the matrix of Example 7.1 is

$$\widehat{R} = \begin{bmatrix} 0 & + & \bullet & \bullet & \bullet \\ 0 & 0 & + & \bullet & \bullet \\ 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

where unspecified elements are denoted by $\bullet$ and positive elements are marked by $+$.

The low echelon form makes it possible to analyze the equation

$$Ax = b, \quad A \in \mathbb{C}^{m\times n},\ b \in \mathbb{C}^m, \tag{7.12}$$

where the nonzero matrix A may be singular (if $m = n$), or rectangular ($m \neq n$) of any rank $r = \operatorname{rank}(A)$ between 1 and $\min\{m, n\}$.

Theoretically, the system (7.12) has a solution if and only if the geometric condition

$$b \in \operatorname{Rg}(A) \tag{7.13}$$

is fulfilled, where

$$\mathrm{Rg}(A) := \{Ax : x \in \mathbb{C}^n\} \subset \mathbb{C}^m$$

is the *range* (or *image*) of the matrix A. The equivalent algebraic necessary and sufficient condition for existence of a solution is

$$\mathrm{rank}[A, b] = \mathrm{rank}(A). \tag{7.14}$$

Note that $\mathrm{rank}(A) \le \mathrm{rank}[A, b] \le 1 + \mathrm{rank}(A)$. Hence the equation (7.12) has no solution if and only if $\mathrm{rank}[A, b] = 1 + \mathrm{rank}(A)$.

Checking either the geometric condition (7.13) or the algebraic rank condition (7.14) for existence of a solution is not a good computational practice. The good procedure here is simply to transform the augmented matrix $[A, b]$ into some condensed form which clearly indicates whether or not the system is solvable.

Let $x^0 \in \mathbb{C}^n$ be a fixed (or particular) solution of (7.12).

Definition 7.1 *The* general solution *(or the set of all solutions) of equation (7.12)*

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

is the linear variety

$$X = x^0 + \mathrm{Ker}(A).$$

Here

$$\mathrm{Ker}(A) := \{x \in \mathbb{C}^n : Ax = 0\} \subset \mathbb{C}^n$$

is the *kernel* (or *null–space*) of the matrix A.

Note that if the equation has no solution then $X = \emptyset$ so the set X is always correctly defined.

It follows from Definition 7.1 that the solution depends on $n-r$ free parameters – this number is the dimension of the subspace $\mathrm{Ker}(A)$). In particular, when the solution exists it is unique if and only if when $r = n$, or equivalently when $\mathrm{Ker}(A)$ is the zero subspace.

To analyze equation (7.12) let us form the augmented matrix

$$C := [A, b] \in \mathbb{C}^{m\times(n+1)}$$

and compute its row echelon form D. We have the following possibilities.

– If D has zero rows (and hence $r < m$) then there are $m - r$ redundant equations which may be deleted obtaining an equivalent equation with r rows.

– If D has a row of the type $[0, 0, \ldots, 0, 1]$ (it will be the row $r+1$) then the equation has no solutions.

– If A is square and nonsingular, then (at least theoretically) the matrix D shall have the form $D = [I_n, x]$, where x is the solution. In this case the solution may formally be written as $x = A^{-1}b$.

Note, however, that the computation of the row echelon form D of the matrix C matrix may be a difficult numerical problem similar to the computation of the inverse of a matrix. The less condensed form $\widehat{D}$ of C may sometimes be computed more reliably.

Consider finally the case when the matrix A is real symmetric, or complex Hermitian, and positive definite (and hence nonsingular). Then it may be decomposed uniquely as

$$A = LL^{\mathrm{H}},$$

or as

$$A = U^{\mathrm{H}}U,$$

where the matrix L is lower triangular (or R is upper triangular) with positive diagonal elements. This special case of LU decomposition is called the *Cholesky decomposition.* It requires about half of the arithmetic operations necessary for the standard LU decomposition and is thus more efficient and (eventually) more accurate.

7.5 Errors in the computed solution

For both methods, described in the previous section, we have the following estimate for the relative error in the solution $\widehat{x} \in \mathbb{R}^n$, computed in machine arithmetic:

$$\delta = \frac{\|\widehat{x} - x\|}{\|x\|} \le \mathbf{u}\,\psi(n)\mathrm{cond}(A),\ \mathrm{cond}(A) := \|A\|\,\|A^{-1}\|,$$

where $\psi(n)$ is a polynomial in n of degree not greater than 3. This estimate is usually pessimistic which means that in practice it is much larger than the actual relative error δ in the computed solution $\widehat{x}$.

According to the famous heuristic "rule of thumb", if

$$\mathbf{u}\,\mathrm{cond}(A) < 1$$

then there are (probably)

$$[-\log_{10}(\mathbf{u}\,\mathrm{cond}(A))]$$

true decimal digits in the computed solution, where $[z]$ is the entire part of the number $z > 0$.

7.6 Solving linear algebraic equations by MATLAB

Linear algebraic equations

$$Ax = b,$$

where the matrix $A \in \mathbb{C}^{n\times n}$ is non–singular, $b \in \mathbb{C}^n$ is a given vector and $x \in \mathbb{C}^n$ is the solution vector, are solved by the command

```
>> x = A\b
```

The same command is used when b and x are matrices of the same size.

If the matrix A is singular or ill–conditioned, there will be a warning message, e.g. `Matrix is close to singular or badly scaled.  Results may be inaccurate.` In this case the condition number `cond(A)` of the matrix A will be estimated as 10^{16} or more. Accordingly, the reciprocal value `rcond(A)` of `cond(A)` will be estimated as 10^{-16} or less. We recall that 10^{-16} is written as `e-016` in MATLAB.

A warning message will also be issued if the sizes of A and b are incompatible.

Hence the computational algorithm which MATLAB uses for solving linear algebraic equations is *reliable.*

A reliable algorithm may not solve any problem. But it warns the user when the solution is eventually contaminated with large errors.

The LU decomposition of the matrix A is found by the command `lu`. In particular, the function

```
>> [L,U] = lu(A)
```

computes an upper triangular matrix U and a (not necessarily lower triangular) matrix L which is a product of lower triangular and permutation matrices such that $A = LU$.

The command

```
>> [L,U,E] = lu(A)
```

gives the lower triangular matrix L, the upper triangular matrix U and the permutation matrix E in the LU decomposition $EA = LU$ of the matrix A.

The row echelon form of a general matrix A is computed in MATLAB by the command

```
>> R = rref(A)
```

The same is done by the function

```
>> R = rref(A,tol)
```

using a tolerance `tol` to decide whether an element is considered as negligible.

The Cholesky decomposition of a positive definite matrix A is computed by the command

```
>> R = chol(A)
```

which gives the upper triangular matrix R such that $A = R^{\mathrm{H}}R$.

7.7 Norms and condition numbers in MATLAB

Vector norms are computed in MATLAB as follows. The Euclidean, or 2-norm

$$\|x\|_2 := \left(\sum_{k=1}^{n} |x_k|^2\right)^{1/2}$$

of the n–vector x with elements x_k is computed by the function

```
>> norm(x)
```

The command

```
>> norm(x,inf)
```

is the same as

```
>> max(abs(x))
```

and computes the norm

$$\|x\|_\infty := \max\{|x_k| : k = 1, 2, \ldots, n\}.$$

The 1–norm

$$\|x\|_1 := \sum_{k=1}^{n} |x_k|$$

is found by the command

```
>> norm(x,1)
```

More generally, for $p \geq 1$ the so called Hölder p–norm

$$\|x\|_p := \left(\sum_{k=1}^{n} |x_k|^p \right)^{1/p}$$

may found by the function

```
>> norm(x,p)
```

The 2–norm

$$\|A\|_2 := \max\{\|Ax\|_2 : x \in \mathbb{C}^n,\ \|x\|_2 = 1\}$$

of the $m \times n$ matrix A, equal to its maximum singular value σ_1, or to the square root of the maximum eigenvalue of the matrix $A^{\mathrm{H}}A$, is found by the command

```
>> norm(A)
```

The 1–norm

$$\|A\|_1 := \max\{\|Ax\|_1 : x \in \mathbb{C}^n,\ \|x\|_1 = 1\}$$

of the $m \times n$ matrix A, equal to the maximum of the 1–norms of its columns, is computed from

```
>> norm(A,1)
```

The function

```
>> norm(A,inf)
```

computes the norm

$$\|A\|_\infty := \max\{\|Ax\|_\infty : x \in \mathbb{C}^n,\ \|x\|_\infty = 1\},$$

of A, which is equal to the maximum of the 1–norms of its rows. Obviously,

$$\|A\|_\infty = \|A^\top\|_1.$$

Finally, the Frobenius norm

$$\|A\|_{\mathrm{F}} := \left(\sum_{k,l=1}^{m,n} |a_{kl}|^2 \right)^{1/2} = \sqrt{\mathrm{tr}(A^{\mathrm{H}}A)}$$

of the matrix A with elements a_{kl} may be found by the command

```
>> norm(A,'fro')
```

Note that

$$\|A\|_{\mathrm{F}} = \left(\sum_{k=1}^{r} \sigma_k^2 \right)^{1/2},$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ are the positive singular values, and r is the rank of A.

The condition number

$$\mathrm{cond}(A) = \mathrm{cond}_2(A) := \|A\|_2 \|A^{-1}\|_2$$

of the nonsingular matrix A in the 2–norm is computed by the command

```
>> cond(A)
```

The function

```
>> rcond(A)
```

gives an estimate for the reciprocal $1/\mathrm{cond}_1(A)$ of the condition number

$$\mathrm{cond}_1(A) := \|A\|_1 \|A^{-1}\|_1$$

of A in the 1–norm. If A is well conditioned then `rcond(A)` is close to 1. If A is very ill conditioned, `rcond(A)` is about the size of the rounding unit $\mathbf{u}$ or even less.

7.8 Problems and exercises

Exercise 7.1 Find in a closed form the condition numbers of the invertible matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{K}^{2\times 2}$$

in both the spectral and Frobenius norms, where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$.

Exercise 7.2 If A and B are invertible matrices of the same size, prove that

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

Exercise 7.3 Let A be an invertible matrix and δA be a perturbation in A such that

$$\|A\|\,\|\delta A\| < 1.$$

Show that the following estimates

$$\frac{\|(A+\delta A)^{-1} - A^{-1}\|}{\|A^{-1}\|} \le \frac{k_A \delta_A}{1 - k_A\delta_A} = \frac{\|A^{-1}\|\,\|\delta A\|}{1 - \|A^{-1}\|\,\|\delta A\|},$$
$$\frac{\|(A+\delta A)^{-1} - A^{-1}\|}{\|(A+\delta A)^{-1}\|} \le k_A\delta_A = \|A^{-1}\|\,\|\delta A\|, \quad k_A := \|A\|\,\|A^{-1}\|,$$

are valid. Hint: for the first estimate use the technique from Section 7.3 assuming that x and b are matrices, and $b = I_n$. Prove the second estimate using the identity from Exercise 7.2.

Exercise 7.4 Let $|A| = [|a_{kl}|]$ be the matrix module of the matrix A, i.e. the matrix of the modules of the elements of A. Consider the equation $Ax = b$ together with its perturbed version

$$(A+\delta A)(x+\delta x) = b + \delta b,$$

where A is an invertible $n\times n$ matrix. Prove that if the eigenvalues of the matrix

$$|A^{-1}|\,|\delta A|$$

have modules less than 1, it is fulfilled that

$$|\delta x| \preceq \left(I_n - |A^{-1}|\,|\delta A|\right) |A^{-1}|(|\delta b| + |\delta A|\,|x|).$$

This estimate is component–wise and may be much more informative than the norm–wise estimates considered above.

Exercise 7.5 Let $\widehat{x}$ be an approximate solution to the equation

$$Ax = b$$

with a square matrix A. Find the minimum norm perturbations δA, δb for which the vector $\widehat{x}$ is an exact solution to the perturbed equation with data $A+\delta A$ and $b + \delta b$, i.e

$$(A+\delta A)\widehat{x} = b + \delta b.$$

For this purpose find the so called *backward error*

$$\min\{\|[\delta A, \delta b]\|_{\mathrm{F}} : (A+\delta A)\widehat{x} = b + \delta b\}$$

solving the linear least squares problem (see Chapter 8)

$$\delta A\widehat{x} - \delta b = b - A\widehat{x}$$

with respect to the unknowns δA, δb.

Exercise 7.6 Show that for an invertible $n\times n$ matrix B and a nonzero n–vector y the relations

$$\sigma_{\min}(B)\|y\| \le \|By\| \le \sigma_{\max}(B)\|y\|$$

and

$$\frac{1}{\|B\|\,\|y\|} \le \frac{1}{\|By\|} \le \frac{\|B^{-1}\|}{\|y\|}$$

are valid. We recall that in the 2–norm it is fulfilled

$$\sigma_{\min}(B) = \frac{1}{\|B^{-1}\|},\ \sigma_{\max}(B) = \|B\|.$$

Exercise 7.7 Consider the linear algebraic equation $Ax = b$ with a square invertible matrix A under perturbations δA, δb, such that the matrix $A + \delta A$ is invertible and $\|\delta b\| < \|b\|$. Prove that the vector $b + \delta b$ and the solution

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

of the perturbed equation are nonzero. Then justify the estimate

$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \frac{k_A(\delta_A + \delta_b)}{1 - \delta_b}.$$

Can you obtain a better estimate?

Exercise 7.8 Generate random linear systems $Ax = b$ with a reference solution $x = [1, 1, \ldots, 1]^\top$ by the commands

```
>> A = rand(n); b = A*ones(n,1)
```

Compute the solution by any of the command lines

```
>> x1 = A\b
>> x2 = inv(A)*b
>> [Q,R] = qr(A); x3 = R\(Q'*b)
```

Analyze the relative errors

```
>> ek = norm(xk - x)/sqrt(n)
```

in the computed approximation to the exact solution x, and compare them with the heuristic bound `eps*cond(A)`.

Make the same experiment with the matrix `A = gallery(3)`, namely

$$A = \begin{bmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{bmatrix}.$$

Exercise 7.9 Determinants must be avoided in solving (even low size) linear algebraic equations since the computation of determinants may not be accurate and/or efficient.

For $n = 1, 2, \ldots, 10$ form the 3×3 matrices

```
>> an = [0,zeros(1,2);zeros(2,1),tn];
>> bn = [tn,zeros(2,1);zeros(1,2),0];
```

where

```
>> tn = [10^n,10^n+1;10^n-1,10^n].
```

Since $\det(tn) = 1$ we should have $\det(an) = \det(bn) = 1$, and $\det(cn) = 1$, where `cn=an*bn`. Compute now the determinants of the matrices an, bn and cn in MATLAB (or in SYSLAB, or in Scilab) by the function `det` and comment the numerical disaster.

Exercise 7.10 Use the Cramer formulae (7.7) to derive explicit expressions for the elements of the matrix A^{-1} (such expressions are of theoretical interest mainly).

Chapter 8

Least squares problems

8.1 Statement of the problem

Consider the *algebraic least squares problem*

$$Ax \simeq b, \tag{8.1}$$

where

$$A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m,\ x \in \mathbb{R}^n,\ m \gg n,$$

which is formulated as an optimization problem for finding x so that

$$\|Ax - b\| = \min! \tag{8.2}$$

Here usually $\|\cdot\|$ is the 2–norm although other norms are used as well.

The solution x of the problem (8.2) satisfies the so called *normal equation* $A^\top Ax = A^\top b$. When $r = \mathrm{rank}(A) = n$ the solution is unique and has the form

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

Note that this is a formal representation of the solution. In practice, the computation of the matrix $A^\top A$ in machine arithmetic may lead to loss of accuracy.

Example 8.1 Let

$$A = \begin{bmatrix} 1 & 0 \\ 0 & \delta \\ 0 & 0 \end{bmatrix},\ b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

where $\mathbf{u} < \delta < \sqrt{\mathbf{u}}$. Then the matrix A has condition number of order $1/\delta$ and the problem $Ax = b$ has an unique solution $x = [x_1, x_2]^\top = [1, 1/\delta]^\top$. In machine arithmetic with rounding unit $\mathbf{u}$ the rank–two matrix $A^\top A = \begin{bmatrix} 1 & 0 \\ 0 & \delta^2 \end{bmatrix}$ will be rounded to the rank–one machine matrix $\mathrm{fl}(A^\top A) = \mathrm{diag}(1, 0)$. As a result the normal equation $A^\top Ax = A^\top b$ becomes $x_1 = 1$, $0 = \delta$ and has no solution.

If $r < n$ then the solution of the problem is not unique (the solution set is an $n - r$ linear variety) and one may consider the additional condition

$$\|x\| = \min! \tag{8.3}$$

Now the least squares problem (8.2), (8.3) has a unique solution. We shall refer to (8.2), (8.3) as the *standard least squares problem.*

The least squares problem is formulated similarly in the complex case, as well as in the general case when the number of equations m may be less, equal to, or larger than the number of unknowns n.

The name of the problem comes from the fact that in (8.2) we minimize in fact the sum

$$\|r\|^2 = r_1^2 + r_2^2 + \cdots + r_m^2$$

of the squares of the components of the so called *residual vector*

$$r = r(x) := Ax - b.$$

There are various formulations of least squares problems. A problem arising in optimization may be formulated by (8.2) under the additional requirement that the elements x_k of x are non–negative,

$$x_k \geq 0, \ \ k = 1, 2, \ldots, n. \tag{8.4}$$

Another least squares problem coming from statistics is to find an n–vector x such that

$$Ax = b + \xi, \tag{8.5}$$

where the m–vector ξ is normally distributed ($m > n$) with zero mean and covariance matrix $V \in \mathbb{R}^{m \times m}$ (V is symmetric positive definite). Here the solution x minimizes the expression

$$(Ax - b)^\top V^{-1}(Ax - b)$$

and formally is defined from

$$x = (A^\top V^{-1} A)^{-1} A^\top V^{-1} b.$$

As in the standard least squares problem, this representation of the solution is usually *not used* for computations, see also the comments presented in the next section.

8.2 Computation of the solution

Theoretically the solution of (8.2), (8.3) may be computed from the condition that the vector r is orthogonal to the subspace $\mathrm{Rg}(A)$, spanned by the columns of A. This gives

$$A^\top r = A^\top (Ax - b) = 0 \tag{8.6}$$

or

$$A^\top A x = A^\top b. \tag{8.7}$$

Indeed, if r is orthogonal to any vector from $\mathrm{Rg}(A)$ it is also orthogonal to the vector $AA^\top r$, i.e.

$$r^\top A A^\top r = \|A^\top r\|^2 = 0,$$

which gives $A^\top r = 0$.

An elegant but formal way to prove (8.6) or (8.7) directly (without considerations about orthogonality) is as follows. Since we want to prove that $y := A^\top r = 0$, suppose the opposite, i.e. that $y \neq 0$. Then $z := Ay \neq 0$ since

$$r^\top z = r^\top A A^\top r = y^\top y = \|y\|^2 > 0.$$

Setting

$$\widetilde{x} := x - \frac{\|y\|^2}{\|z\|^2} y$$

we get

$$\begin{aligned} \|A\widetilde{x} - b\| &= \left\| r - \frac{\|y\|^2}{\|z\|^2} z \right\| = \|r\|^2 - 2r^\top z \frac{\|y\|^2}{\|z\|^2} + \frac{\|y\|^4}{\|z\|^2} \\ &= \|r\|^2 - \frac{\|y\|^4}{\|z\|^2} < \|r\|^2. \end{aligned}$$

But this contradicts to the fact that x minimizes $\|r\| = \|Ax - b\|$. Hence $A^\top r = 0$.

Equation (8.7) constitutes the so called *normal form* of the least squares problem. At first glance it seems preferable since it is (usually) an equation of much smaller size (m instead of n) with a symmetric non–negative definite matrix $A^\top A$. But in practice the formation of the matrix $A^\top A$ should be avoided since it may lead to the loss of rank when the computations are done in machine arithmetic.

Example 8.2 Suppose that we use a machine arithmetic with rounding unit $\mathbf{u}$ and consider the matrix

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

where the quantity ε satisfies

$$\mathbf{u} \le \varepsilon < \sqrt{\mathbf{u}}.$$

The matrix A is of full rank 2, which can well be recognized by the modern computer systems for matrix calculations.

If we form the matrix

$$A^\top A = \begin{bmatrix} 1 & 1 \\ 1 & 1+\varepsilon^2 \end{bmatrix}$$

we see that it is again of rank 2, but after rounding the element in position (2,2) is written in the computer memory as 1. Hence the machine matrix

$$\mathrm{fl}(A^\top A) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

is of rank 1 and the rounded normal system may even have no solution! This will be the case when e.g. $b = [1, 1, 0]^\top$.

Consider the solution of the problem with a matrix A of full column rank n using the method of QR decomposition.

Let

$$A = QR$$

be the QR decomposition of A, where the matrix $Q \in \mathbb{R}^{m\times m}$ is orthogonal ($Q^\top Q = I_m$) and the matrix R is of the form

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

Here $R_1 \in \mathbb{R}^{n\times n}$ is an upper triangular non–singular matrix. If we set

$$Q^\top b = \begin{bmatrix} c \\ d \end{bmatrix}, \quad c \in \mathbb{R}^n,$$

then

$$\|Ax - b\| = \|QRx - b\| = \|Q(Rx - Q^\top b)\| = \|Rx - Q^\top b\| = \min!$$

Since

$$\|Rx - Q^\top b\|^2 = \|R_1 x - c\|^2 + \|d\|^2$$

we must determine $x = [x_1, x_2, \ldots, x_n]^\top$ as the solution to the equation

$$R_1 x = c = [c_1, c_2, \ldots, c_n]^\top.$$

Since the matrix $R_1 = [r_{kl}]$ is upper triangular, this equation is solved directly by back substitution, namely

$$\begin{aligned} x_n &= \frac{c_n}{r_{nn}}, \\ x_{n-1} &= \frac{c_{n-1} - r_{n-1,n} x_n}{r_{n-1,n-1}}, \\ \cdots & \quad \cdots \\ x_1 &= \frac{c_1 - r_{12} x_2 - r_{13} x_3 - \cdots - r_{1n} x_n}{r_{11}}. \end{aligned}$$

In case when the matrix A is not of a full rank the least squares problem can be solved by using the singular value decomposition of the matrix A as follows.

Let the rank of the matrix $A \in \mathbb{R}^{m \times n}$ be $r < \min\{m, n\}$. Then the singular value decomposition of the matrix A is

$$A = USV^\top,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices ($U^\top U = I_m$, $V^\top V = I_n$) and

$$S := \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad \Sigma := \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \in \mathbb{R}^{r \times r}.$$

Here $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ are the singular values of A. We recall that they are the square roots of the positive eigenvalues of the matrix $A^\top A$.

Setting $y := V^\top x$ we obtain

$$r = Ax - b = USy - b = U(Sy - U^\top b).$$

Hence $\|r\| = \|Sy - U^\top b\|$ and since $\|y\| = \|x\|$ we have the new least squares problem

$$\begin{aligned} &\|Sy - U^\top b\| = \min! \\ &\|y\| = \min! \end{aligned}$$

which is easily solvable since the matrix S is diagonal. Indeed, set

$$y := \begin{bmatrix} u \\ v \end{bmatrix}, \quad U^\top b := \begin{bmatrix} c \\ d \end{bmatrix},$$

where $u, c \in \mathbb{R}^r$, $v \in \mathbb{R}^{n-r}$, $d \in \mathbb{R}^{m-r}$. Then

$$Sy - U^\top b = \begin{bmatrix} \Sigma u \\ 0 \end{bmatrix} - \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \Sigma u - c \\ -d \end{bmatrix}.$$

Hence

$$\|Sy - U^\top b\|^2 = \|\Sigma u - c\|^2 + \|d\|^2$$

and we see that the minimum of $\|Sy - U^\top b\|$ is achieved when $\Sigma u = c$, i.e.

$$u = \Sigma^{-1} c.$$

Here the determination of u does not require an actual inversion of a matrix since Σ is a diagonal matrix, i.e. the elements u_k of u are calculated very accurately from

$$u_k = \frac{c_k}{\sigma_k}, \; k = 1, 2, \ldots, r.$$

In this case the minimum value of $\|\Sigma y - U^\top b\|$, equal to the minimum of value of $\|Ax - b\|$, is $\|d\|$.

Up to now v is arbitrary. But

$$\|x\|^2 = \|y\|^2 = \|u\|^2 + \|v\|^2 = \|\Sigma^{-1} c\|^2 + \|v\|^2$$

and in order to minimize $\|x\| = \|y\|$ one has to choose $v = 0$.

Returning to the original unknown vector x we obtain the solution of the least squares problem as

$$x = V \begin{bmatrix} \Sigma^{-1} c \\ 0 \end{bmatrix} = A^\dagger b.$$

Here $A^\dagger \in \mathbb{R}^{n \times m}$ is the so called *Moore–Penrose pseudoinverse* of the matrix A, defined by the expression

$$A^\dagger := V S^\dagger U^\top,$$

where

$$S^\dagger := \begin{bmatrix} \Sigma^{-1} & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{n \times m}.$$

8.3 Error estimation

When using the computational scheme based on QR decomposition the error estimate for the computed solution $\widehat{x}$ is

$$\delta := \frac{\|\widehat{x} - x\|}{\|x\|} \leq 6mn\, k_A \gamma(A, b)\mathbf{u},$$

where

$$\gamma(A, b) := 1 + k_A \frac{k_A \|Ax - b\| + \|b\|}{\|A\|\, \|x\|}$$

and

$$k_A := \operatorname{cond}(A) = \|A\|\, \|A^\dagger\|$$

is the condition number of the full rank matrix A. Here $A^\dagger \in \mathbb{R}^{n\times m}$ is the pseudoinverse of the matrix A. We recall that if

$$A = USV^\top \in \mathbb{R}^{m\times n}$$

is the singular value decomposition of the matrix $A \in \mathbb{R}^{m\times n}$ of rank p, where the matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal ($U^\top U = I_m$, $V^\top V = I_n$) and

$$S = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p, 0) \in \mathbb{R}^{m\times n}, \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p > 0,$$

then the pseudoinverse of A is

$$A^\dagger = VS^\dagger U^\top \in \mathbb{R}^{n\times m}, \quad S^\dagger := \operatorname{diag}(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_p, 0) \in \mathbb{R}^{n\times m}.$$

8.4 Solving least squares problems by MATLAB

The algebraic linear least squares problem

$$Ax \simeq b,$$

where $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$ is the data and $x \in \mathbb{C}^n$ is the solution, is computed by the command

```
>> x = A\b
```

This MATLAB command realizes the so called “left division” of the vector b by the matrix A. We stress that the same command solves the linear algebraic equation $Ax = b$ when A is a non–singular matrix.

Consider now the matrix least squares problem

$$AX \simeq B,$$

where $B = [B_1, B_2, \dots, B_p]$ and $X = [X_1, X_2, \dots, X_p]$ are $m \times p$ matrices with columns B_k and X_k, respectively. In this case the solution X, computed by left division, will now minimize the norms $\|AX_k - B_k\|$ of the residual vectors $AX_k - B_k$.

The solution x of the least squares problem can also be expressed as

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

where $A^\dagger \in \mathbb{C}^{n\times m}$ is the pseudoinverse of A.

The pseudoinverse $Y := A^\dagger$ of A is produced by the command

```
>> Y = pinv(A)
```

The computation is based on the singular value decomposition of the matrix A. The computed pseudoinverse is in fact the pseudoinverse with a threshold

$$\tau = \max\{m, n\}\|A\|\mathbf{u},$$

where $\mathbf{u}$ is the rounding unit of the machine arithmetic.

The MATLAB command

```
>> Y = pinv(A,tol)
```

computes the pseudoinverse Y of A with a threshold `tol`, prescribed by the user.

The least squares problem (8.2), (8.4) is solved by the command

```
>> x = nnls(A,b)
```

In turn, the solution of the problem (8.5) may be computed from

```
>> x = lscov(A,b,V)
```

8.5 Problems and exercises

Exercise 8.1 Show that the pseudoinverse $A^\dagger \in \mathbb{R}^{n\times m}$ of $A \in \mathbb{R}^{m\times n}$ is the unique matrix such that the solution x of the least squares problem

$$\|Ax - b\| = \min!$$
$$\|x\| = \min!$$

can be represented as $x = A^\dagger b$ for all vectors $b \in \mathbb{R}^m$ (here $\|\cdot\|$ is the 2–norm). The same is valid when A, b and x are complex quantities.

Exercise 8.2 Form series of large size $2n \times n$ random matrices A and random $2n$–vectors b. Find the solution to the corresponding least squares problems $Ax \simeq b$ first by the command

```
>> x = A\b
```

and then by the function

```
>> xx = inv(A'*A)*(A'*b)
```

Compare the two solutions and the corresponding residual vectors (theoretically they are the same). Construct an example where the second approach leads to larger errors.

Exercise 8.3 Generate series of random $m \times n$ matrices A and m–vectors b. Solve the corresponding standard least squares problems by the command

```
>> x = A\b
```

and the least squares problems with the constraints (8.4) by the command

```
>> xx = nnls(A,b)
```

Compare the solutions `x` and `xx` and the norms of the corresponding residual vectors.

Chapter 9

Eigenvalue problems

9.1 Main definitions and properties

In this section we give a survey of the main definitions and properties concerning the so called eigenstructure (eigenvalues and eigenvectors) of real and complex square matrices.

Let A be an $n \times n$ matrix over $\mathbb{R}$ or $\mathbb{C}$.

Definition 9.1 *The number λ (real or complex) is said to be an* eigenvalue *of A if there exists a nonzero n–vector u such that*

$$Au = \lambda u, \tag{9.1}$$

or, equivalently,

$$(\lambda I_n - A)u = 0. \tag{9.2}$$

The vector u is called an eigenvector*, corresponding to the eigenvalue λ, and (λ, u) is called an* eigenpair *of the matrix A.*

We stress that even if the matrix A is real, some (or all) of its eigenpairs may be complex. Indeed, let $\lambda = \alpha + \imath\beta$ be a complex eigenvalue of A, where α, β are real with $\beta \neq 0$, and u be the corresponding eigenvector. Then

$$A\overline{u} = \overline{\lambda}\overline{u}$$

and hence $\overline{\lambda} = \alpha - \imath\beta$ is also an eigenvalue and $\overline{u}$ is an eigenvector, corresponding to $\overline{\lambda}$.

Further on, we may look for the eigenvector u in the form $u = r + \imath s$, where the vectors r, s are real. Substituting these expressions for λ and u in (9.1) we obtain

$$\begin{aligned} Ar &= \alpha r - \beta s, \\ As &= \beta r + \alpha s, \end{aligned}$$

or

$$AU = U\Lambda,$$

where

$$U := [r, s] \in \mathbb{R}^{n\times 2}, \ \Lambda := \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \in \mathbb{R}^{2\times 2}.$$

The column vector u above is also referred to as the *right eigenvector* of A corresponding to the eigenvalue λ of A. We may also define the *left eigenvector* w of A corresponding to the same eigenvalue λ. This is a row non–zero vector satisfying

$$wA = \lambda w.$$

Hence $A^{\mathrm{H}} w^{\mathrm{H}} = \overline{\lambda} w^{\mathrm{H}}$. The column vector $v := w^{\mathrm{H}}$ is also called a *left eigenvector* of A. It follows from

$$v^{\mathrm{H}} A = \lambda v^{\mathrm{H}}$$

that v is a (right) eigenvector of the matrix A^{H} corresponding to the eigenvalue $\overline{\lambda}$. When the matrix A is real, v is the (right) eigenvector of the matrix $A^{\top}$

Since equation (9.2) has a nonzero solution u, the matrix $\lambda I_n - A$ must be singular. Indeed, in the opposite case the zero solution of (9.2) should be unique and there should not be a non–zero solution. Hence

$$h_A(\lambda) := \det(\lambda I_n - A) = 0, \tag{9.3}$$

where I_n is the $n \times n$ identity matrix.

Definition 9.2 *The algebraic equation (9.3) of degree n is called the* characteristic equation *of the matrix A and $h_A(\lambda)$ is called the* characteristic polynomial *of A.*

Clearly, the eigenvalues of the matrix A are the roots of its characteristic equation and vice versa. Usually we are interested in the collection $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of all n eigenvalues of the $n \times n$ matrix A counted according to their algebraic multiplicity (as roots of the characteristic equation). The set of pair–wise different eigenvalues of A is sometimes called the *spectrum* of A while the collection of all eigenvalues is referred to as the *full spectrum* of A.

Thus the eigenvalues of a given matrix A are uniquely determined. In contrast, the eigenvector u, corresponding to the eigenvalue λ of A, is not unique. In particular, the vector ku, where k is a nonzero scalar, will also be an eigenvector of A, associated with the same eigenvalue λ. Further on we shall usually suppose that the right and left eigenvectors are normed (in the 2–norm) in the sense that

$$\|u\| = \|v\| = 1.$$

The characteristic equation of A has the form

$$\lambda^n - c_1\lambda^{n-1} + \cdots + (-1)^n c_n = 0.$$

Here the coefficient $c_k = c_k(A)$ of the characteristic polynomial $h_A(\lambda)$ of A is the sum of principal minors of A of order k. For example,

$$c_1 = \mathrm{tr}(A), \ \ c_n = \det(A).$$

In practice, the coefficients c_k are not *computed as sums of the principal minors of A. If, for some reasons, the coefficients c_k are to be computed, this is done by specialized algorithms.* As a rule, the computation of c_k from the elements of A should be avoided.

According to the famous *Caley–Hamilton theorem*, any matrix satisfies its characteristic equation in the sense that

$$h_A(A) = A^n - c_1A^{n-1} + \cdots + (-1)^n c_n I_n = 0.$$

However, the matrix A may also satisfy a monic polynomial of lower degree. The minimum degree polynomial with this property is called the *minimal polynomial* of A and is denoted by $\mu_A(\lambda)$. We have

$$\mu_A(A) = 0$$

by definition.

Example 9.1 The characteristic polynomial of the matrix aI_n, where a is a number and I_n is the identity $n \times n$ matrix, is

$$(\lambda - a)^n = \lambda^n - n\lambda^{n-1}a + \cdots + (-a)^n,$$

but its minimal polynomial is $\lambda - a$.

If the matrix A has pair–wise disjoint eigenvalues then $h_A(\lambda) = \mu_A(A)$ (prove!). If A has multiple eigenvalues, its characteristic and minimal polynomials may still be equal. More generally, these polynomials coincide if and only if each eigenvalue of A corresponds to exactly one Jordan block in the Jordan form of A (see the definitions given later on).

Some of the eigenvalues of A may be equal. Let $\lambda_1, \lambda_2, \ldots, \lambda_m$ $(m \leq n)$ be the pair–wise disjoint eigenvalues of A. Then the characteristic polynomial of A may be written as

$$h_A(\lambda) = (\lambda - \lambda_1)^{n_1}(\lambda - \lambda_2)^{n_2} \cdots (\lambda - \lambda_m)^{n_m},$$

where $n_k \geq 1$ is the *algebraic multiplicity* of the eigenvalue λ_k of A. We have

$$n_1 + n_2 + \cdots + n_m = n.$$

The set $\{\lambda_1, \lambda_2, \ldots, \lambda_m\} \subset \mathbb{C}$ of eigenvalues of A is the *spectrum* of A and is sometimes denoted by $\mathrm{spect}(A)$. We denote by $\mathrm{spect}(A)$ also the collection[1] $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of the eigenvalues of A counted according to their algebraic multiplicity.

We stress that the eigenvalues of A are usually *not* computed as the zeros of its characteristic polynomial, see Section 9.3.

Definition 9.3 *Two $n \times n$ matrices A and B are called* similar *if there is a non–singular $n \times n$ matrix X such that*

$$AX = XB,$$

or equivalently, if $B = X^{-1}AX$.

The characteristic polynomials h_A and h_B of two similar matrices A and B coincide. Indeed,

$$\begin{aligned} h_B(\lambda) &= \det(\lambda I - B) = \det(\lambda I - X^{-1}AX) = \det(X^{-1}(\lambda I - A)X) \\ &= \det(X^{-1})\det(\lambda I - A)\det(X) = h_A(\lambda). \end{aligned}$$

Thus similar matrices have equal spectra. However, two matrices with equal characteristic polynomials may not be similar.

[1]A collection is a set with possibly repeated elements.

Example 9.2 For the matrices $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = 0_{2\times 2}$ we have $h_A(\lambda) = h_B(\lambda) = \lambda^2$ but these matrices are not similar since $A \neq 0_{2\times 2}$ and $X^{-1}BX = 0_{2\times 2}$.

Consider the similar matrices A and $B = X^{-1}AX$. If (λ, u) is an eigenpair of A then (λ, x) with $x := X^{-1}u$ is an eigenpair of B. Indeed, we have

$$\lambda u = Au = XBX^{-1}u.$$

Multiplying these equalities on the left by X^{-1} we get $\lambda(X^{-1}u) = B(X^{-1}u)$ and hence $x = X^{-1}u$ is an eigenvector of B, corresponding to the eigenvalue λ. Similarly, if v is a left eigenvector of A, then the vector $y := X^{\mathrm{H}}v$ is a left eigenvector of B, corresponding to the same eigenvalue λ.

An important corollary from the above considerations is that the scalar product

$$v^{\mathrm{H}}u = (y^{\mathrm{H}}X^{-1})(Xx) = y^{\mathrm{H}}x \tag{9.4}$$

of the left and right eigenvectors, corresponding to a given eigenvalue of a matrix A, is invariant relative to similarity transformations of A.

Let A and B be arbitrary $n \times n$ matrices. Then

$$\mathrm{spect}(AB) = \mathrm{spect}(BA).$$

Indeed, suppose first that $\lambda = 0$ is an eigenvalue of AB. Then AB is singular and at least one of the factors A or B will also be singular. Hence BA is in turn singular and has a zero eigenvalue. Suppose now that $\lambda \in \mathrm{spect}(AB)$ is nonzero and let u be the corresponding eigenvector of AB, i.e. $ABu = \lambda u$. Multiplying this equality on the left by B we get $BA(Bu) = \lambda(Bu)$. Since $v := Bu$ is nonzero (otherwise ABu and hence λu and u should be zero) we see that v is an eigenvector of the matrix BA corresponding to the eigenvalues λ. Hence $\lambda \in \mathrm{spect}(BA)$.

Let now A be an $m \times n$ matrix and B be an $n \times m$ matrix. Then similar arguments as above show that *the nonzero eigenvalues of the matrices AB and BA coincide.*

Let the pair–wise disjoint eigenvalues of the $n \times n$ matrix A be $\lambda_1, \lambda_2, \ldots, \lambda_m$ with algebraic multiplicities $n_1, n_2, \ldots, n_m$, respectively, where $m \leq n$. It may be shown that there exists a non–singular matrix X such that the matrix

$$D := X^{-1}AX = \mathrm{diag}(D_1, D_2, \ldots, D_m) \tag{9.5}$$

is block–diagonal with blocks $D_k \in \mathbb{C}^{n_k \times n_k}$, $k = 1, 2, \ldots, m$, along its main diagonal. Moreover, the block D_k has the form

$$D_k = \lambda_k I_{n_k} + M_k, \tag{9.6}$$

where the matrix M_k is strictly upper triangular. Moreover, the matrix M_k may be reduced in a form with at most $n_k - 1$ nonzero elements, equal to 1, on its super–diagonal. Thus the matrix D_k has a single eigenvalue λ_k of algebraic multiplicity n_k.

The *complete eigenstructure* of a general real or complex $n \times n$ matrix A under similarity transformations is revealed by its *Jordan canonical form* as follows. There exists a non–singular matrix $X = X(A) \in \mathbb{C}^{n \times n}$ such that the matrix

$$J = J(A) := X^{-1} A X$$

is block–diagonal with 1×1 blocks

$$J_1(\lambda) := \lambda$$

and/or $p \times p$ ($p \geq 2$) blocks

$$J_p(\lambda) := \begin{bmatrix} \lambda & 1 & \ldots & 0 & 0 \\ 0 & \lambda & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & \lambda & 1 \\ 0 & 0 & \ldots & 0 & \lambda \end{bmatrix} = \lambda I_p + N_p \in \mathbb{C}^{p \times p}$$

on its main diagonal. Here $\lambda \in \operatorname{spect}(A)$, I_p is the $p \times p$ identity matrix, and N_p is a nilpotent $p \times p$ matrix ($N_p^p = 0$) of the form $\begin{bmatrix} 0 & I_{p-1} \\ 0 & 0 \end{bmatrix}$.

Definition 9.4 *The matrix J is called the* Jordan canonical form *of the matrix A.*

The Jordan form is unique within ordering of the blocks $J_q(\lambda)$ along its main diagonal. Thus the Jordan form J of A is a bidiagonal matrix, similar to A, with the eigenvalues of A on its main diagonal and with elements, equal to 1 or 0, at its super–diagonal.

Consider now the eigenstructure of the blocks $J_q(\lambda)$. The case $q = 1$ is trivial since here $J_1(\lambda) = \lambda$ is a scalar. Furthermore, each block $J_p(\lambda)$ with $p \geq 2$ has an eigenvector of the form $e_1 := [1, 0, \ldots, 0]^\top \in \mathbb{C}^p$,

$$J_p(\lambda) e_1 = \lambda e_1.$$

The block $J_p(\lambda)$ has also a chain of $p-1$ *principle vectors*

$$e_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, e_p := \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \in \mathbb{C}^p,$$

satisfying the equations

$$J_p(\lambda)e_k = e_{k-1} + \lambda e_k, \ k = 2, 3, \ldots, p.$$

To each eigenvector and principle vector of $J_p(\lambda)$ there is an eigenvector and a principle vector of size n of the matrix J and hence of the matrix A. The reader is advised to construct these vectors.

The set of all vectors and principle vectors of A are in fact the columns of the transformation matrix X (prove!).

Note that a given eigenvalue λ_k of A may correspond to several Jordan blocks $J_p(\lambda_k)$. The number of such blocks is the *geometric multiplicity* of λ_k and is denoted by ν_k. It may be shown that ν_k is the dimension of the subspace $\mathrm{Ker}(D_k)$, see the representation (9.5), (9.6). Note also that the matrix D_k has a Jordan form consisting of all Jordan blocks of A having λ_k as an eigenvalue.

Obviously, $\nu_k \le n_k$, where n_k is the algebraic multiplicity of A.

Definition 9.5 *The difference* $d_k := n_k - \nu_k$ *is called the* defect *of the eigenvalue* λ_k*. The sum* $d_1 + d_2 + \cdots + d_m$ *of the defects of the eigenvalues of* A *is the* defect *of* A*. The matrix is* A *is said to be* non–defective *when its defect is zero, and* defective *otherwise.*

Definition 9.6 *Matrices* A *with only* 1×1 *Jordan blocks in its Jordan form* J *are called* diagonalizable*.*

For a diagonalizable matrix A the Jordan form J is diagonal. It is easy to see that a matrix is diagonalizable if and only if it is non–defective.

Denote by $p_k \le n_k$ the size of the maximum Jordan block of A having λ_k as an eigenvalue. Then the minimal polynomial of A may be written as

$$\mu_A(\lambda) = (\lambda - \lambda_1)^{p_1}(\lambda - \lambda_2)^{p_2} \cdots (\lambda - \lambda_m)^{p_m}.$$

The Jordan form of a complex matrix A may be complex or real since its spectrum may be an arbitrary subset of $\mathbb{C}$. If the matrix A is real and has only real eigenvalues then its Jordan form J and the transformation matrix X may also be chosen real.

If, however, some (or all) of the eigenvalues of the real matrix A are complex, then its Jordan form J is complex by necessity. In this case it is possible to transform A into the so called *real Jordan form*

$$J^{\mathbb{R}} = J^{\mathbb{R}}(A) = X^{-1}AX$$

using a real transformation matrix X as follows.

First, to each real eigenvalue λ there correspond one or several Jordan blocks $J_q(\lambda)$.

Second, suppose that $\lambda = \alpha + \imath\beta$ ($\alpha, \beta \in \mathbb{R}$, $\beta \neq 0$) is a complex eigenvalue of the real matrix A. Then, as we already know, its complex conjugate $\overline{\lambda} = \alpha - \imath\beta$ is also an eigenvalue of A. In this case the real Jordan form may contain real Jordan blocks of the form

$$\Lambda := \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

or of the form

$$\begin{bmatrix} \Lambda & I_2 & \dots & 0 & 0 \\ 0 & \Lambda & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \Lambda & I_2 \\ 0 & 0 & \dots & 0 & \Lambda \end{bmatrix} = \lambda I_p \otimes \Lambda + N_p \otimes I_2 \in \mathbb{R}^{2p\times 2p}.$$

It is important to know that *the computation of the spectrum of a defective matrix may be a difficult numerical problem.* This is due to the high sensitivity of the eigenstructure of such matrices, see Section 9.2. In particular, the computation of the Jordan structure of such matrices in machine arithmetic may be contaminated with large errors.

We stress that the elements equal to 1 on the super–diagonal of J are introduced for theoretical reasons. Instead, we may use any nonzero elements. In this case we obtain the so called *quasi–Jordan form* with blocks

$$\begin{bmatrix} \lambda & d_1 & \dots & 0 & 0 \\ 0 & \lambda & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda & d_{p-1} \\ 0 & 0 & \dots & 0 & \lambda \end{bmatrix} \in \mathbb{C}^{p\times p}, \ d_1 d_2 \cdots d_{p-1} \neq 0,$$

instead of $J_p(\lambda)$. Briefly speaking, the quasi–Jordan form of A is a bidiagonal matrix, similar to A, and with no restrictions on the elements at its super–diagonal.

An important property of the quasi–Jordan form is that *it may be much less sensitive to perturbations* and hence *it may have better numerical properties than the Jordan form.*

9.2 Sensitivity of the eigenstructure

In this section we analyze the sensitivity of the eigenvalues and eigenvectors of the $n \times n$ matrix A when it is subject to perturbations $A \to A + \delta A$. Different eigenvalues may have different sensitivities. Also, eigenvectors may be more sensitive than eigenvalues. As a general rule, eigenvalues with only 1×1 Jordan blocks are less sensitive than eigenvalues with Jordan blocks of size $p > 1$.

Suppose first that λ is a simple eigenvalue of the matrix A, i.e. that it has algebraic multiplicity 1.

Let δA be a perturbation in A, which causes perturbations $\delta\lambda$ in the eigenvalue λ and δu, δv in the unit eigenvectors u, v, respectively. We have

$$(A + \delta A)(u + \delta u) = (\lambda + \delta\lambda)(u + \delta u).$$

Having in mind that $Au = \lambda u$ and within accuracy of first order (neglecting the second order small terms $\delta A\delta u$ and $\delta\lambda\delta u$) we obtain

$$(A - \lambda I_n)\delta u + \delta A u = \delta\lambda u.$$

Let us multiply this equality on the left by v^{H} having in mind that v is a left eigenvector,

$$v^{\mathrm{H}}(A - \lambda I_n) = 0.$$

As a result we get

$$v^{\mathrm{H}}\delta A u = (v^{\mathrm{H}}u)\delta\lambda.$$

Because λ is a simple eigenvalue, we have $v^{\mathrm{H}}u \neq 0$. To prove that let u be the first column of a non–singular matrix U. Then we have

$$B := U^{-1}AU = \begin{bmatrix} \lambda & b \\ 0 & C \end{bmatrix},$$

where b is a row $(n-1)$–vector and C is a matrix of size $(n-1) \times (n-1)$. Since λ is a simple eigenvalue of A and hence of B, then it is not an eigenvalue of C, i.e.

the matrix $\lambda I_{n-1} - C$ is invertible. An unit right eigenvector of B, associated to λ, is $x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Let $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ be the corresponding left eigenvector, where $y_1 \in \mathbb{C}$ and $y_2 \in \mathbb{C}^{n-1}$. Then it follows from $y^{\mathrm{H}}B = \lambda y^{\mathrm{H}}$ that

$$[\lambda \overline{y}_1, \lambda y_2^{\mathrm{H}}] = [\lambda \overline{y}_1, \overline{y}_1 b + y_2^{\mathrm{H}} C]$$

and

$$y_2^{\mathrm{H}}(\lambda I_{n-1} - C) = \overline{y}_1 b.$$

If $y_1 = 0$ then the invertibility of the matrix $\lambda I_{n-1} - C$ would give $y_2 = 0$ and hence the whole vector y would be zero which is impossible. Hence $y_1 \neq 0$. Having in mind (9.4) we obtain

$$v^{\mathrm{H}}u = y^{\mathrm{H}}x = \overline{y}_1 \neq 0.$$

Returning now to the equation for $\delta\lambda$ we find

$$\delta\lambda = \frac{v^{\mathrm{H}}\delta A u}{v^{\mathrm{H}}u}$$

and

$$|\delta\lambda| \leq \frac{\|v\|\,\|\delta A\|\,\|u\|}{|v^{\mathrm{H}}u|} = \frac{\|\delta A\|}{|v^{\mathrm{H}}u|}.$$

Moreover, if $\delta A = \rho v u^{\mathrm{H}}$ is a scalar multiple of the matrix vu^{H} with $\rho \neq 0$, then

$$\|\delta A\| = |\rho|\,\|vu^{\mathrm{H}}\| = |\rho|$$

and

$$\|v^{\mathrm{H}}\delta A u\| = \|\rho\, v^{\mathrm{H}}v\, u^{\mathrm{H}}u\| = |\rho|.$$

Thus the quantity

$$K_\lambda := \frac{1}{|v^{\mathrm{H}}u|}$$

is the *absolute condition number* for the eigenvalue λ.

When $\lambda \neq 0$ we have also an estimate in terms of relative perturbations as follows

$$\delta_\lambda \leq k_\lambda \delta_A,$$

where

$$\delta_\lambda := \frac{|\delta\lambda|}{|\lambda|},\ \delta_A := \frac{\|\delta A\|}{\|A\|}$$

and

$$k_\lambda := K_\lambda \frac{\|A\|}{|\lambda|}$$

is the *relative condition number* of the simple eigenvalue λ.

The condition numbers for the eigenvalues of A are called also *spectral condition numbers* for the matrix A.

In a similar way we may find an estimate for the perturbations in the eigenvectors when all eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of the $n \times n$ matrix A are pairwise disjoint. Let u_k and v_k be the eigenvector and left eigenvector corresponding to the eigenvalue λ_k such that $\|u_k\| = \|v_k\| = 1$.

Denote by $\lambda_k + \delta\lambda_k$, $u_k + \delta u_k$ and $v + \delta v_k$ the corresponding perturbed quantities associated with the perturbed matrix $A + \delta A$. Then we have (within first order terms)

$$\|\delta u_k\| \leq C_k \|\delta A\|,$$

where

$$C_k := \sum_{i \neq k} \frac{1}{|(\lambda_k - \lambda_i) v_i^{\mathrm{H}} u_i|} = \sum_{i \neq k} \frac{K_{\lambda_i}}{|\lambda_k - \lambda_i|}$$

is the *absolute condition number* of the eigenvector u_k. We see that the eigenvector condition numbers may be large if the matrix A has close eigenvalues even when the eigenvalues themselves are well conditioned.

9.3 Computation of the eigenstructure

In this section we consider the problem of computing the eigenstructure (i.e. eigenvalues and eigenvectors or principal vectors) of the matrix $A \in \mathbb{C}^{n \times n}$. Let λ be an eigenvalue of A and u and v be the corresponding normed right and left eigenvectors, i.e.

$$\begin{aligned} Au &= \lambda u, \ \|u\| = 1, \\ v^{\mathrm{H}} A &= \lambda v^{\mathrm{H}}, \ \|v\| = 1. \end{aligned}$$

We stress that the eigenvalues of A are usually not *computed as the roots of its characteristic equation.* There are two main reasons for that.

First, the coefficients c_k of the characteristic polynomial $h_A(\lambda)$ may be computed from the elements of A with large errors even if sophisticated algorithms for this purpose are used. And second, the roots of the characteristic equation $h_A(\lambda) = 0$ may be very sensitive to changes in the coefficients of $h_A(\lambda)$ even when the eigenvalues of the matrix A are not sensitive to perturbations in its elements.

In summary, the computation of the eigenvalues of A as roots of the characteristic equation of A is an example of a "bad" decomposition of a computational problem into subproblems which may be very sensitive.

The computational aspects of eigenvalue finding has an interesting history. In early times (second half of XIX century) the eigenvalues of A had been calculated as the roots of the characteristic polynomial $h_A(\lambda)$. In the second half of XX century things have changed. Now the eigenvalues are calculated directly, e.g. by QR type methods of Francis and Kublanovskaya. Moreover, if a polynomial $p(\lambda)$ is given, then its roots are computed as the eigenvalues of its companion matrix! Thus the computational paradigm in this case has been completely reversed.

The computation of the Jordan form J of A is usually a delicate problem. Indeed, an arbitrarily small perturbation δA in A may completely destroy and change the Jordan structure of A when A has multiple eigenvalues.

Example 9.3 The matrix $A = N_n = \begin{bmatrix} 0 & I_{n-1} \\ 0 & 0 \end{bmatrix}$, $n > 1$, is in Jordan form. Its only eigenvalue $\lambda = 0$ has algebraic multiplicity n and geometric multiplicity $\nu = 1$. Let us perturb A to $\widehat{A} := A + \delta A$, where all elements of δA are zero except the element in position $(n, 1)$, which is equal to $\alpha > 0$. Then the perturbed matrix $\widehat{A}$ has n distinct eigenvalues $\lambda_k = \varepsilon_k \alpha^{1/n}$, where $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ are the n roots of $(-1)^n$ and $\alpha^{1/n} > 0$. Hence the Jordan form of $\widehat{A}$ is $\widehat{J} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. We see that the 1–block Jordan form $J = A$ of A has been completely destroyed and changed into the n–block form $\widehat{J}$ of $\widehat{A}$.

However, the Jordan form may also be computed reliably, see the function `jord` from SYSLAB based on the algorithm of Kagström and Ruhe from [15]. For this purpose the quasi–Jordan form is used.

The system MATLAB has symbolic options (the command `jordan`) for finding the Jordan form of matrices with rational elements.

The most popular method for finding the eigenstructure of a medium sized general matrix is the QR method. This method computes the Schur form of the matrix by a sequence of QR decompositions.

According to a famous result of I. Schur from the beginning of XX century, for each $n \times n$ matrix A (real or complex) there is an unitary matrix $U \in \mathbb{C}^{n \times n}$ ($U^{\mathrm{H}} U = I_n$) such that the matrix

$$T = [t_{kl}] := U^{\mathrm{H}} A U \in \mathbb{C}^{n \times n} \qquad (9.7)$$

is upper triangular.

The matrices A and T are similar and their eigenvalues coincide. In turn, the eigenvalues of the triangular matrix T are its diagonal elements. Hence the desired eigenvalues $\lambda_k = \lambda_k(A)$ of A, counted according to their algebraic multiplicities, are equal to the diagonal elements t_{kk} $(k = 1, 2, \ldots, n)$ of the matrix T.

Definition 9.7 *The matrix T from (9.7) is called the* Schur form *of A. The columns of the unitary matrix U form the* Schur basis *for $\mathbb{C}^n$ relative to the matrix A.*

For real matrices A the transformation matrix U may be chosen real and orthogonal ($U^\top U = I_n$). Note that in this case the Schur form T will also be real but may be only quasi–triangular instead of triangular. The details of this transformation are given below.

i) If the matrix A has only real eigenvalues, then the matrix T will be upper triangular with $t_{kk} = \lambda_k(A)$, $k = 1, 2, \ldots, n$.

ii) If the matrix A has at least one pair of complex conjugate eigenvalues then the Schur form T will be upper quasitriangular with 1×1 blocks (corresponding to the real eigenvalues of A, if any) and 2×2 blocks (corresponding to pairs of complex eigenvalues of A) on its main diagonal.

Definition 9.8 *The matrix T described in i) and ii) is called the* real Schur form *of the real matrix A.*

For a general matrix A of order 5 or more, the Schur form cannot be constructed by a finite number of algebraic operations. This is a consequence of two facts. The first one is a theorem of Abel–Galois–Ruffini from the beginning of XIX century which states that the roots of a general polynomial of degree 5 or more may not be expressed as algebraic functions of its coefficients. The second fact is that the roots of a polynomial are the eigenvalues of its companion matrix.

Hence the eigenvalues of the companion $n \times n$ matrix of a general polynomial of degree $n \geq 5$ may not be expressed as algebraic functions of the matrix entries since otherwise the theorem of Abel–Galois–Ruffini will be violated.

The QR method to find the Schur form T and the matrix U of the Schur vectors of A, in its most simple (and hardly effective) form is as follows.

Denote $A_1 := A$ and let

$$A_1 =: Q_1 R_1$$

be the QR decomposition of A_1. Set

$$A_2 := R_1 Q_1$$

and let

$$A_2 =: Q_2 R_2$$

be the QR decomposition of A_2. Continuing this process, let

$$A_{k-1} =: Q_{k-1} R_{k-1}$$

be the QR decomposition of the matrix A_{k-1}, obtained at the $(k-1)$–th step. Then at the k–th step we form the matrix

$$A_k := R_{k-1} Q_{k-1}$$

and compute its QR decomposition

$$A_k =: Q_k R_k.$$

It may be shown that for a large class of matrices this process is convergent in the sense that A_k tends to an upper triangular matrix which, by necessity, is the Schur form of A. Indeed, we have

$$R_1 = Q_1^{\mathrm{H}} A_1 = Q_1^{\mathrm{H}} A$$

and

$$A_2 = R_1 Q_1 = Q_1^{\mathrm{H}} A Q_1.$$

Furthermore,

$$R_2 = Q_2^{\mathrm{H}} A_2$$

and

$$A_3 = R_2 Q_2 = Q_2^{\mathrm{H}} A_2 Q_2 = Q_2^{\mathrm{H}} Q_1^{\mathrm{H}} A Q_1 Q_2 = (Q_1 Q_2)^{\mathrm{H}} Q_1 Q_2.$$

In a similar way we obtain

$$A_k = (Q_1 Q_2 \cdots Q_{k-1})^{\mathrm{H}} A Q_1 Q_2 \cdots Q_{k-1}$$

for all $k \geq 2$. Since the matrix $Q_1 Q_2 \cdots Q_{k-1}$ is unitary (as a product of unitary matrices), we see that A_k is unitary similar to A.

In practice the QR algorithm is implemented in a more sophisticated manner in order to ensure the efficiency and accuracy of the computational procedure. In particular, the matrix A is preliminary reduced into the so called *Hessenberg form* $H = V^{\mathrm{H}}AV$ by a finitely constructed unitary transformation V. We recall that the Hessenberg form $H = [h_{kl}]$ is a matrix with $h_{kl} = 0$ for $k > l+1$. Thus an $n \times n$ Hessenberg matrix (having zero elements below its main subdiagonal) has at most $(n^2 + 3n - 2)/2$ nonzero elements.

9.4 Errors in the computed eigenvalues

When $\widehat{\lambda}$ is the computed approximation to the simple eigenvalue λ of the matrix A using the QR method in machine arithmetic with rounding unit $\mathbf{u}$, we have the estimate

$$\frac{|\widehat{\lambda} - \lambda|}{\|A\|} \leq \mathbf{u}\,\psi(n)K_\lambda,$$

where $\psi(n)$ is a quadratic or linear polynomial in n.

When the eigenvalue under consideration has a multiplicity $k > 1$, then the problem of its computation is singular, possibly very sensitive and one may expect that the computed value $\widehat{\lambda}$ is contaminated with absolute error of order

$$C_\lambda \mathbf{u}^{1/k},$$

where $C_\lambda > 0$ is a constant.

Note that in all cases the relative errors of non–zero eigenvalues may be very large.

Example 9.4 Consider the $n \times n$ Jordan block

$$A = \lambda_1 I_n + N_n = \begin{bmatrix} \lambda_1 & 1 & \dots & 0 & 0 \\ 0 & \lambda_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda_1 & 1 \\ 0 & 0 & \dots & 0 & \lambda_1 \end{bmatrix}$$

which has an eigenvalue λ_1 of multiplicity $n_1 = n$.

Let us perturb the matrix A to $A+E$, where E has a single nonzero element $(-1)^n\varepsilon \neq 0$ in position $(n, 1)$. Then the characteristic equation of the matrix $A + E$ is

$$(\lambda - \lambda_1)^n = \varepsilon$$

and the eigenvalues of $A + E$ are of the form

$$\lambda = \lambda_1 + \varepsilon^{1/n}.$$

Thus a perturbation of norm $\|E\| = |\varepsilon|$ causes perturbations in the eigenvalues of size $|\varepsilon|^{1/n}$. Let e.g. $n = 16$ (a very moderate size of the matrix!), $\lambda_1 = -0.01$ and $\varepsilon = 10^{-16}$, which is of the order of the rounding errors in this case. Then the change in the eigenvalues is of size 0.1. Thus even for a relatively small matrix the rounding errors can cause a relative perturbation of 1000 percent in the computed eigenvalues. In particular the matrix A is stable in the sense that λ_1 is in the open left–half of $\mathbb{C}$ but the perturbed matrix $A + E$ is not stable.

9.5 Generalized eigenstructure

In this section we consider some generalizations of the standard eigenvalue problem $Au = \lambda u$ studied above. For this purpose we shall recall the notion of a degree of a polynomial. The polynomial

$$p(x) = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0$$

is identified with the $(n+1)$–vector $p = [p_n, p_{n-1}, \ldots, p_0]$, where the coefficients p_k and the variable x are real or complex quantities. Since some of the coefficients of p may be zero (including p_n) we shall need the following definition.

Definition 9.9 *The* degree $\deg(p)$ *of the non–zero polynomial* p *is defined as*

$$\deg(p) := \max\{k : p_k \neq 0\}.$$

In the special case $p = 0$ *we set* $\deg(p) = -\infty$.

In particular, if $p_n = p_{n-1} = \cdots = p_1 = 0$ and $p_0 \neq 0$ then $\deg(p) = 0$.

Consider now the *generalized eigenproblem*

$$Au = \lambda Bu, \tag{9.8}$$

where $A, B \in \mathbb{C}^{n\times n}$ and $0 \neq u \in \mathbb{C}^n$. The problem is to find the *generalized eigenvalues* $\lambda \in \mathbb{C}$ and the corresponding *generalized eigenvectors* u of the pair (A, B). Note that when $B = I_n$ we have the standard eigenvalue problem $Au = \lambda u$ for the matrix A, considered above.

Denote by $\mathrm{spect}(A,B)$ the set of eigenvalues of the problem (9.8). With this notation we see that $0 \neq \lambda \in \mathrm{spect}(A,B)$ implies $Bu = \lambda^{-1}Au$ and hence $\lambda^{-1} \in \mathrm{spect}(B,A)$.

Since $(\lambda B - A)u = 0$ and $u \neq 0$ then theoretically the generalized eigenvalues may be found as the roots of the algebraic *characteristic equation*

$$\chi_{A,B}(\lambda) := \det(\lambda B - A) = 0. \tag{9.9}$$

Here $\chi_{A,B}(\lambda)$ is the *characteristic polynomial* of the pair (A,B). The degree $d = d(A,B) := \deg(\chi_{A,B})$ of this polynomial in λ is not larger than the rank of the matrix B but may as well be less. In particular the polynomial $\chi_{A,B}(\lambda)$ may even reduce to a constant. However, if $\mathrm{rank}(B) = n$ then $d = n$.

Example 9.5 Let A and B be 2×2 upper triangular matrices with diagonal elements a_1, a_2 and b_1, b_2, respectively. Then

$$\chi_{A,B}(\lambda) = (b_1\lambda - a_1)(b_2\lambda - a_2).$$

The degree d of this polynomial and the set $\mathrm{spect}(A,B)$ are determined as follows.

(i) If $b_1b_2 \neq 0$ then $d = 2$ and $\mathrm{spect}(A,B)$ has two elements a_1/b_1, a_2/b_2 which may coincide.

(ii) If $b_1a_2 \neq 0$ and $b_2 = 0$ (or if $b_2a_1 \neq 0$ and $b_1 = 0$) then $d = 1$ and $\mathrm{spect}(A,B)$ contains one element a_1/b_1 (or a_2/b_2).

(iii) if $b_1 = b_2 = 0$ and $a_1a_2 \neq 0$ then the polynomial is reduced to the nonzero constant a_1a_2 and hence $d = 0$. The set $\mathrm{spect}(A,B)$ is empty.

(iv) if $b_1 = a_1 = 0$ (or if $b_2 = a_2 = 0$) then the polynomial is reduced to 0 and hence $d = -\infty$. The set $\mathrm{spect}(A,B)$ is the whole complex plane $\mathbb{C}$.

Note that in case (i) the matrix B is of full rank 2 and in case (ii) it is of rank 1. In both cases $d = \mathrm{rank}(B)$. At the same time in cases (iii) and (iv) the matrix B may be of rank 1 and hence $d < \mathrm{rank}(B)$.

Having in mind the above example, we come to the following classification of generalized eigenproblems.

1. Let $\mathrm{rank}(B) = n$. Then $d = n$ and we have n generalized eigenvalues counted according to their algebraic multiplicities.

2. Let $1 \leq \mathrm{rank}(B) < n$ and let there exist two numbers $\lambda, \mu \in \mathbb{C}$ such that $\det(\lambda B - A) \neq 0$ and $\det(\mu B - A) = 0$. Here the polynomial $\chi_{A,B}(\lambda)$

is of degree d, $1 \le d \le \operatorname{rank}(B)$, and has d roots counted according to multiplicity.

3. Let $\chi_{A,B}(\lambda)$ be identically equal to $\det(A) \neq 0$. Then $d = 0$ and there are no generalized eigenvalues, i.e. $\operatorname{spect}(A, B) = \emptyset$. In this case one may consider the generalized eigenproblem $\mu Au = Bu$ which has n eigenvalues $\mu_1, \mu_2, \ldots, \mu_n$.

4. Let $\chi_{A,B}(\lambda)$ be identically equal to $\det(A) = 0$, i.e. $\det(\lambda B - A) = 0$ for all $\lambda \in \mathbb{C}$. Here $d = -\infty$, any $\lambda \in \mathbb{C}$ is a generalized eigenvalue and $\operatorname{spect}(A, B) = \mathbb{C}$.

The computational problem in Case 1 is *well posed* in the sense that for any perturbation δA in A and for a sufficiently small perturbation δB in B the generalized eigenproblem

$$(A + \delta A)(u + \delta u) = (\lambda + \delta\lambda)(B + \delta B)(u + \delta u) \tag{9.10}$$

will still have n eigenvalues $\lambda_k + \delta\lambda_k$ $(k = 1, 2, \ldots, n)$. Moreover, here the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ are continuous functions of A and B. This in particular will be the case when $\|\delta B\|_2 < 1/\|B^{-1}\|_2$.

In turn, Case 1 may be *regular* when the generalized eigenvalues are Lipschitz continuous functions of A and B, and *singular* in the opposite case. In the latter case the eigenvalues are Hölder continuous in A and B.

The computational problems in Cases 2, 3 and 4 are *ill posed* in the sense that for any $\varepsilon > 0$ there is $\delta B \neq 0$ with $\|\delta B\| < \varepsilon$ such that for all δA the perturbed eigenproblem (9.10) will have a full set of n eigenvalues. This is due to the fact that under an arbitrarily small perturbation δB the matrix B may always be changed into a full rank matrix $B + \delta B$.

We stress that the numerical solution of well posed singular generalized eigenproblems as well as of ill posed problems, may be contaminated with large errors. But even if the problem is well posed and regular, it may still pose a difficult numerical task if it is ill conditioned.

If the matrix B is non–singular (Case 1) then we have

$$B^{-1}Ax = \lambda x$$

and the generalized eigenvalues and eigenvectors of the pair (A, B) are simply the eigenvalues and eigenvectors of the matrix $B^{-1}A$. In particular, $\operatorname{spect}(A, B) = \operatorname{spect}(B^{-1}A, I_n)$.

If the matrix B is singular but A is non–singular (Case 3 and eventually Case 2) then we may consider the eigenproblem

$$A^{-1}Bx = \mu x,\ x \neq 0, \tag{9.11}$$

where $\mu = 1/\lambda$. Since here the matrix $A^{-1}B$ is singular, then $\mu = 0$ is an eigenvalue of $A^{-1}B$ of certain algebraic multiplicity. The corresponding generalized eigenvalue λ of the same multiplicity of (A, B) is called *infinite generalized eigenvalue.*

Definition 9.10 *The matrix pairs (A, B) and (F, G) from $\mathbb{C}^{n\times n} \times \mathbb{C}^{n\times n}$ are called* similar *if there are two non–singular matrices $Q, Z \in \mathbb{C}^{n\times n}$ such that*

$$F = Q^{-1}AZ, \quad G = Q^{-1}BZ.$$

If the matrices Q and Z are unitary, then the pairs (A, B) and (F, G) are said to be unitary similar.

Setting $v = Z^{-1}u$ we may rewrite the generalized eigenproblem (9.8) as

$$Fv = \lambda Gv.$$

Hence $\mathrm{spect}(A, B) = \mathrm{spect}(F, G)$ and the generalized eigenvalues are preserved under similarity transformations.

The invariant similarity structure of the matrix pair (A, B) is revealed by the so called *Kronecker form* of a matrix pair which is a generalization of the Jordan form of a single matrix. As may be expected, the numerical computation of the Kronecker form may be a difficult problem similarly to the computation of the Jordan form.

The unitary invariant similarity structure of the pair (A, B) is revealed by the *generalized Schur form* as follows. It may be shown that there exist unitary matrices $Q, Z \in \mathbb{C}^{n\times n}$ such that both matrices $T := Q^{\mathrm{H}}AZ = [t_{kl}]$ and $S := Q^{\mathrm{H}}BZ = [s_{kl}]$ are upper triangular. The reduction of (A, B) into the form (T, S) is done by the so called *QZ algorithm*, see [10].

Note that the numbers of nonzero diagonal elements of T and S are the ranks of A and B, respectively.

The characteristic equation of the unitary equivalent form (T, S) of the pair (A, B) has the simple form

$$\chi_{T,S}(\lambda) = \det(\lambda S - T) = \prod_{k=1}^{n} (s_{kk}\lambda - t_{kk}).$$

In the well posed case $s_{kk} \neq 0$ $(k = 1, 2, \ldots, n)$ the eigenvalues of the eigenproblem $Au = \lambda Bu$ are given by

$$\lambda_k = \frac{t_{kk}}{s_{kk}}, \ k = 1, 2, \ldots, n.$$

If for some $k \in \overline{1, n}$ it is fulfilled that $s_{kk} = t_{kk} = 0$ then the characteristic polynomial of (T, S) and hence of (A, B) is identically zero. Here $\mathrm{spect}(A, B) = \mathbb{C}$.

If finally for some $k \in \overline{1, n}$ the conditions $s_{kk} = 0$ and $t_{kk} \neq 0$ are valid then the eigenvalue λ_k is infinite.

There is also a symmetric formulation of the generalized eigenproblem, namely

$$\mu Au = \lambda Bu,$$

where $u \neq 0$ and $\lambda, \mu \in \mathbb{C}$. Here the generalized eigenvalue is the nonzero complex pair (λ, μ). Two pairs (λ_1, μ_1) and (λ_2, μ_2) are considered as *equivalent* if there is a nonzero constant ν such that $\lambda_2 = \nu\lambda_1$ and $\mu_2 = \nu\mu_1$.

9.6 Solving eigenproblems by MATLAB and SYSLAB

In MATLAB and SYSLAB the eigenstructure is computed by the function `eig`. In particular,

```
>> [U,D] = eig(A)
```

gives the matrix U of eigenvectors u and the diagonal matrix D of corresponding eigenvalues λ of the matrix A. The simpler command

```
>> E = eig(A)
```

returns the vector E of eigenvalues of A.

The generalized eigenstructure is found as follows. The function

```
>> [U,D] = eig(A,B)
```

gives the matrix U of eigenvectors u and the diagonal matrix D of corresponding eigenvalues λ of the eigenproblem $Au = \lambda Bu$. The simpler command

```
>> E = eig(A,B)
```

computes the vector E of generalized eigenvalues λ of the pair (A, B).

The function `eigs` is similar to `eig` and is used to compute a few eigenvalues of a square matrix. It has several forms and the reader is advised to use `help eigs` for more details.

The characteristic polynomial of the $n \times n$ matrix A may be computed by the command

```
>> p = poly(A)
```

Here p is a row $(n+1)$–vector containing the coefficients of the characteristic polynomial h_A of A. In particular, $p(1) = 1$, $p(2) = -\mathrm{tr}(A)$ and $p(n+1)$ is (or should be) equal to $(-1)^n \det(A)$.

Spectral condition numbers (condition numbers with respect to eigenvalues) may be found by the function

```
>> K =  condeig(A)
```

Here K is a vector with elements $|v_k^{\mathrm{H}} u_k|^{-1}$, where u_k and v_k are the right and left eigenvectors of A, corresponding to its eigenvalue λ_k, respectively.

The Hessenberg form of a matrix may be found by the function

```
>> hess(A)
```

The QZ factorization for generalized eigenproblems is computed by the function `qz`. In particular,

```
>> [T,S,Q,Z,U] = qz(A,B)
```

gives the generalized Schur form $(T, S) = (QAZ, QBZ)$ of (A, B), the transformation unitary matrices Q, Z and the eigenvector matrix U. Note that here the transformation is written as $A \to QAZ$, etc. instead of $A \to Q^{\mathrm{H}}AZ$ as in the previous section.

The function

```
>> [U,T] = schur(A)
```

computes the Schur form T and the transformation unitary matrix U for the matrix A, such that

$$A = UTU^{\mathrm{H}}.$$

The command

```
>> T = schur(A)
```

returns only the Schur form T of A.

If A is real, then the real quasi–triangular Schur form of A is computed by the function

```
>> [U,T] = schur(A,'real')
```

The functions `schur` and respectively the functions of the type `eig` give accurate results when the eigenvalues of A are not very ill conditioned. If A has very ill conditioned eigenvalues as well as when A has multiple eigenvalues with nonlinear elementary divisors (i.e. when the Jordan canonical form J_A of A is not diagonal but block diagonal) then the computations may be contaminated with large errors.

A reliable program `jord` to compute the Jordan form of a matrix is available in the program system SYSLAB. It is based on an algorithm proposed by Kagström and Ruhe[15]. The command

```
>> [X,J] = jord(a)
```

returns the transformation matrix X and the Jordan form J of A such that $AX = XJ$.

Example 9.6 The matrix

$$A = \begin{bmatrix} -90001 & 769000 & -690000 \\ -810000 & 6909999 & -6200000 \\ -891000 & 7601000 & -6820001 \end{bmatrix} \tag{9.12}$$

has a Jordan form

$$J_3(-1) = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \tag{9.13}$$

with a triple eigenvalue $\lambda_1 = -1$. The computation of the eigenvalues of A by the command `eig` in MATLAB gives

```
>> eig(A)
ans =
   -1.5994 + 1.0441i
   -1.5994 - 1.0441i
    0.1989
```

The result is a computational catastrophe. The system usually works with 15 true digital digits. Here we have none which is understandable since the relative error in the computation of spect(A) is about 233 percent. Even the conclusion about the stability of A is wrong since a positive eigenvalue 0.1989 has been computed. At the same time the sum -3 of computed eigenvalues is correct. Also, the determinant of A is computed exactly by the command `det(A)` as -1. The characteristic polynomial of A, however, is wrongly computed by `poly(A)` as

$$\lambda^3 + 3.0000\lambda^2 + 3.0120\lambda - 0.7255.$$

The fatal error is done in computing the free term as -0.7255 instead of 1.

At the same time the program

```
>> E = jord(A)
```

from SYSLAB computes the eigenvalues of A with 9 true digits!

The MATLAB command

```
>> J = jordan(A)
```

computes the Jordan form J of the matrix A in case when A is known exactly so that its elements are integers or ratios of relatively small integers. In turn, the command

```
>> [X, J] = jordan(A)
```

finds the Jordan form J of A and the non–singular matrix X such that $J = X^{-1}AX$. Hence the columns of X are the eigenvectors and/or the principal vectors of A. The command `J = jordan(A)` now computes correctly the Jordan form (9.13) of the matrix (9.12).

9.7 Problems and exercises

Exercise 9.1 Generate the matrix `A = gallery(5)` in MATLAB and compute its eigenvalues by the command `eig(A)`. Then compute the determinant `det(A)` of A. Explain the result having in mind that the determinant of a matrix is the product of its eigenvalues taken according to their algebraic multiplicity. Can you find the exact eigenvalues of A?

Exercise 9.2 Generate a sequence of random square small size matrices A. Find the eigenvalues and eigenvectors of A.

Exercise 9.3 Generate a sequence of random square matrices A and B of equal small size. Find the generalized eigenvalues and eigenvectors of the problem $Ax = \lambda Bx$. Compare the computed generalized eigenvalues with the eigenvalues of the matrix $A^{-1}B$ (the randomly generated matrix A will, most probably, be non–singular).

Exercise 9.4 The commands `eig(A)` and `roots(poly(A))` should produce identical results, namely the vector of eigenvalues of the square matrix A. Make experiments with various matrices A and compare the results (be careful with the ordering of the elements of the vectors produced by both commands). Explain the results obtained.

Exercise 9.5 Write a program in MATLAB for realization of a simplified version of the QR algorithm for reduction of a square matrix into Schur form. Use a preliminary transformation of the matrix into Hessenberg form to improve the efficiency and accuracy of the computations. Compare the execution of your program with the results obtained by the corresponding MATLAB commands.

Chapter 10

Rank determination and pseudoinversion

10.1 Statement of the problem

There are several definitions of the rank of a matrix A of size $m \times n$. We shall define the rank of A as the number of its linearly independent columns (this is also the number of its linearly independent rows).

The determination of the rank of a given matrix $A \in \mathbb{R}^{m \times n}$ in machine arithmetic with rounding unit $\mathbf{u}$ is a very delicate problem. The reason is that the rank is not a stable characteristic of the matrix in the following sense.

Suppose that the matrix A is not of a full rank, i.e. that

$$r := \text{rank}(A) < p := \min\{m, n\}.$$

Then an arbitrary small perturbation δA in A may increase the rank to its maximum possible value p, namely

$$\text{rank}(A + \delta A) = p.$$

In this chapter we consider the numerical computation of the rank of a given matrix. Another problem that is addressed here is the computation of the Moore–Penrose pseudoinverse of a general (rectangular) matrix A. For definiteness we consider real matrices. The complex case is treated similarly.

10.2 Difficulties in rank determination

The rank determination is connected to specific difficulties. Some of them are due to the fact that the rank may be very sensitive as a function of the matrix elements.

Example 10.1 Let the singular value decomposition of the matrix $A \in \mathbb{R}^{m\times n}$ of rank $r < p = \min\{m, n\}$ be

$$\begin{aligned} &A = USV^\top, \\ &S = S(A) = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, 0, \ldots, 0), \\ &\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0, \end{aligned}$$

where U and V are orthogonal matrices ($U^\top U = I_m$, $V^\top V = I_n$) and $\sigma_k = \sigma_k(A)$ are the singular values of A. We recall that the singular values are the nonnegative square roots of the eigenvalues of the matrix $A^\top A$.

The perturbation

$$\delta A = U\mathrm{diag}(0, 0, \ldots, 0, \eta_{r+1}, \eta_{r+2}, \ldots, \eta_p)V^\top$$

with

$$\eta_{r+1} = \eta_{r+2} = \cdots = \eta_p = \eta := \frac{\varepsilon}{\sqrt{p-r}}$$

has a spectral norm η and a Frobenius norm ε, where $\varepsilon > 0$ may be arbitrary small. At the same time it is easy to see that the perturbed matrix $A + \delta A$ is of full rank p.

Thus we have shown that the rank of a matrix, which is not of a full rank, is not stable. On the other hand when writing the data in the computer memory we obtain the rounded matrix $\widehat{A} = \mathrm{round}(A)$, instead of the exact matrix A, such that

$$\|\widehat{A} - A\| \leq \mathbf{u}\|A\|.$$

Therefore the rounded matrix may be of full rank even if the original one is not.

Thus we have the paradoxical situation when the (theoretical) rank of a dense matrix of even a moderate size is a non–computable quantity. Indeed, for such a matrix the rank can be computed (in a reasonable time) only by using a computer. But then the rank of the matrix is destroyed at the moment when we store the matrix in the computer memory.

The exact rank of medium size matrices with rational elements may also be computed by extended machine precision using the MATLAB command `vpa` (for Variable Precision Arithmetic). Here we do not consider such cases.

There is a radical way to deal with this situation and it is *to recognize that the "theoretical rank" of a matrix (defined e.g. as the number of its linearly independent columns) makes no sense for large general dense matrices!*

However, the reader must be aware that there are many other quantities that are not exactly computable (whatsoever this may mean). Such are e.g. the solutions of most equations. The shock with the rank is that it is an integer (and very important) characteristic of a matrix.

Of course, something can be saved from the concept of a rank. And it is described below as the new concept of a numerical rank of a matrix.

There are several ways to deal with the rank of a matrix in machine arithmetic, e.g. via QR decomposition or singular value decomposition. The most reliable among them is to determine the rank of a matrix as the number of its positive singular values. One of the reasons for this choice is that the singular values (in contrast to the eigenvalues) are always not sensitive to perturbations in the original matrix. And we know that these perturbations are inevitable when rounding the data and intermediate results occur.

10.3 Sensitivity of singular values

Let $A = USV^\top$ be the singular value decomposition of the matrix $A \in \mathbb{R}^{m\times n}$ of rank r, where the matrices U and V are orthogonal and $S = S(A) = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0)$. Here $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r > 0$ are the positive singular values of A. Using this decomposition and the unitary invariance of the matrix norms $\|\cdot\|_2$ and $\|\cdot\|_{\mathrm{F}}$, one can prove the following inequality known as the *Theorem of Mirsky*

$$\|S(A+\delta A) - S(A)\|_2 \leq \|\delta A\|_2.$$

We also have

$$\sqrt{\sum_{k=1}^{r}(\sigma_k(A+\delta A) - \sigma_k(A))^2} \leq \|\delta A\|_{\mathrm{F}}$$

and

$$\max\{|\sigma_k(A+\delta A) - \sigma_k(A)| : k = 1, 2, \ldots, r\} \leq \|\delta A\|_2.$$

Hence the singular values of any matrix are very well conditioned with an absolute condition number, equal to 1.

10.4 Numerical rank and numerical pseudoinverse

Suppose that

$$\widehat{\sigma}_1 \geq \widehat{\sigma}_2 \geq \cdots \geq \widehat{\sigma}_r$$

are the singular values of the matrix A, computed in machine arithmetic. If among them there are small ones (compared with $\|A\|$) then it may be expected that they are due to rounding errors and not to the rank of the initial matrix A.

When the singular values are grouped in two well defined clusters: "large" (of the order of $\|A\|$) and "small" (of the order of $\mathbf{u}\,\|A\|$) then the problem is easy – we may assume that the rank is the number of "large" singular values.

The problem of rank determination is much more subtle when the singular values form a sequence with uniformly decreasing size, e.g. $\widehat{\sigma}_k \approx 10^{-k}$. Here arises the task of deleting the singular values which are not inherent for the problem but are rather due to rounding or other type of errors. For this purpose one may introduce the concept of the numerical rank of a matrix.

Let $\tau > 0$ be a given (small) numerical threshold.

Definition 10.1 *The matrix A is of* numerical rank $s = s(A, \tau)$ *within the threshold τ, when it has exactly s singular values, larger than τ, i.e.*

$$\sigma_s > \tau, \quad \sigma_{s+1} \leq \tau.$$

If $\sigma_1 \leq \tau$ the numerical rank of A is assumed to be zero.

Usually the threshold is chosen as $\tau = \mathbf{u}C\|A\|$, where the constant C depends on the size of the matrix A.

In practice we do not know the exact singular values σ_k of A, but rather have their computed approximations $\widehat{\sigma}_k$. That is why the numerical rank of the matrix A in machine arithmetics with rounding unit $\mathbf{u}$ is determined as the number of the computed quantities $\widehat{\sigma}_k$, which are larger than $\tau = \mathbf{u}\,\sigma_1^*$ (we recall that $\|A\|_2 = \sigma_1$). There are also more conservative rank estimators in which the lower bound for the singular values, contributing to the rank, is accepted as $\mathbf{u}\,q\,\widehat{\sigma}_1$ or even $\mathbf{u}\,q^2\,\widehat{\sigma}_1$, where q is the order of the matrix in the sense $q = \max\{m, n\}$.

In accordance with the concept of numerical rank we may introduce the concept of numerical pseudoinverse of a matrix.

Let s be the numerical rank of A with a threshold $\tau > 0$.

Definition 10.2 *The numerical pseudoinverse* $A_\tau^\dagger \in \mathbb{R}^{m\times n}$ *of* A *is defined as* $A_\tau^\dagger = 0$ *if* $\sigma_1 \le \tau\|A\|$ *and*

$$A_\tau^\dagger = VS_\tau^\dagger U^\top,$$

where

$$S_\tau\dagger := \begin{bmatrix} \Sigma_\tau^{-1} & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times n},\ \Sigma_\tau := \mathrm{diag}(\sigma_1, \sigma_2, \ldots \sigma_s) \in \mathbb{R}^{s\times s}.$$

10.5 Rank determination and pseudoinversion by MATLAB

There is a special command

```
>> r = rank(A)
```

for the determination of the rank of the matrix $A \in \mathbb{C}^{m\times n}$ in the program systems MATLAB, SYSLAB and Scilab. This function estimates the number of linearly independent columns of A by the number of positive singular values of A. In fact, only singular values that are larger than `max(size(A))*norm(A)*eps`, or

$$\max\{m, n\}\|A\|_2\mathrm{eps},$$

are taken into account. Here eps $\approx 2\mathbf{u}$ is the machine epsilon. In the IEEE double precision standard we have eps $\approx 2.2 \times 10^{-16}$.

The command

```
>> rt = rank(A,tol)
```

computes the numerical rank of A with a user prescribed tolerance `tol`. It is equal to the number of singular values of A that are larger than the positive quantity `tol`. This function is also based on the singular value decomposition of the matrix A.

In turn, the singular value decomposition is computed by the commands `svd`. The command

```
>> [U,S,V]  = svd(A)
```

computes the corresponding matrices in the singular value decomposition $A = USV^{\mathrm{H}}$ of A. The command

```
>> sv = svd(A)
```

computes only the vector sv of positive singular values of A in descending order. The command

```
>> [U1,S1,V] = svd(A,0)
```

returns the "economy" size decomposition. If A is $m \times n$ with $m > n$ then the first n columns U_1 of U are computed and the matrix S_1 is of size $n \times n$.

The function `svds` finds a few singular values and vectors. There are many variants of this function and the reader is advised to see `help svds` for help.

The generalized singular value decomposition of the matrix pair (A, B) is computed by the function `gsvd`, consult `help gsvd` for more information.

Let $A = USV^{\mathrm{H}}$ be the SVD of the matrix $A \in \mathbb{C}^{m\times n}$ of rank r, where the matrices U and V are unitary ($U^{\mathrm{H}}U = I_m$, $V^{\mathrm{H}}V = I_n$) and

$$S := \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad \Sigma := \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \in \mathbb{R}^{r\times r}.$$

Then the Moore–Penrose pseudoinverse of A is defined as

$$A^{\dagger} := VS^{\dagger}U^{\mathrm{H}} \in \mathbb{C}^{n\times m},$$

where

$$S^{\dagger} := \begin{bmatrix} \Sigma^{-1} & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{n\times m}, \quad \Sigma^{-1} = \mathrm{diag}(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_r).$$

It may be shown that the Moore–Penrose pseudoinverse $X = A^{\dagger} \in \mathbb{C}^{n\times m}$ of the matrix $A \in \mathbb{C}^{m\times n}$ is also the only matrix satisfying the relations

$$AXA = A,\ XAX = X,\ (AX)^{\mathrm{H}} = AX,\ (XA)^{\mathrm{H}} = XA$$

(prove!).

The pseudoinverse of A is computed by the command

```
>> X = pinv(A)
```

This command in fact computes the numerical pseudoinverse of A with tolerance $\max\{m, n\}\|A\|_2\mathrm{eps}$, where the machine epsilon eps is about twice the rounding unit $\mathbf{u}$ the floating point machine arithmetic.

The numerical pseudoinverse of A with user defined tolerance `tol` is found by the command

```
>> Xt =  pinv(A,tol)
```

A relatively conservative tolerance may be defined in MATLAB as

```
>> tol = max(size(A))^2*norm(A)*eps
```

10.6 Problems and exercises

Exercise 10.1 Consider the least squares problem

$$Ax \simeq b.$$

Show that the Moore–Penrose pseudoinverse $A^\dagger$ of A is uniquely determined by the requirement that $x = A^\dagger b$, where x is the minimum 2–norm solution of the problem

$$\|Ax - b\|_2 = \min!$$

Exercise 10.2 Explain why $A^\dagger = VS^\dagger U^{\mathrm{H}}$ is not in general the singular value decomposition of the matrix $A^\dagger$.

Exercise 10.3 Generate a sequence of random $m \times n$ matrices A and compute their pseudoinverses with tolerances

```
tolk = max(size(A))^k*norm(A)*eps
```

for $k = 1, 2, \ldots$. Compare the computed numerical ranks, corresponding to different values of k.

PART III

NUMERICAL MATHEMATICAL ANALYSIS

Chapter 11

Evaluation of functions

11.1 Introductory remarks

The computation of the values of a given scalar, vector or a matrix function

$$x \mapsto y = f(x)$$

in machine arithmetic may be connected with significant difficulties and large errors especially if done by numerically unreliable algorithms. In this chapter we consider the source of errors and give estimates for the accuracy of the computed results. Then we discuss the fundamentals of calculations with complex numbers. We also briefly discuss the main built–in scalar functions in MATLAB.

11.2 Complex calculations

An important feature of function evaluation in MATLAB is that all computations are done in complex arithmetic and all functions are defined both for real complex values of their arguments. So a recollection of basic notations for complex numbers may be useful.

Complex numbers $z \in \mathbb{C}$ are written in algebraic or exponential form. The algebraic notation is $z = x + y\mathrm{i}$, where $x \in \mathbb{R}$ is the *real* and $y \in \mathbb{R}$ is the *imaginary* part of z, and $\mathrm{i} = \sqrt{-1}$ is the *imaginary unit* in $\mathbb{C}$. The number $\overline{z} = x - y\mathrm{i}$ is *conjugate* to z.

The *module*, or *absolute value* of z is the real quantity

$$|z| = \operatorname{abs} z = \sqrt{z\overline{z}} = \sqrt{x^2 + y^2} \geq 0.$$

For $z \neq 0$ the *principal argument*, or *angle* $\varphi = \arg z$ is uniquely defined from the relations

$$\cos\varphi = \frac{x}{|z|}, \ \sin\varphi = \frac{y}{|z|}, \ z \in (-\pi, \pi]$$

(the number $z = 0$ has no argument). In particular $\arg \overline{z} = -\arg z$ for $z \neq 0$.

We stress that any complex number $z \neq 0$ has a countable set of arguments $\varphi_k = \arg z \pm 2k\pi$, $k \in \mathbb{N}$.

In MATLAB the quantities $|z|$, $\arg z$ and $\overline{z}$ are computed by the commands `abs(z)`, `angle(z)` and `conj(z)`, respectively. In turn, the real and imaginary parts of z are computed from `real(z)` and `imag(z)`. The command `complex(x,y)` will produce $z = x + \mathrm{i}y$.

In exponential form the number $z \neq 0$ is written as

$$z = |z|(\cos\varphi + \mathrm{i}\sin\varphi) = |z|\mathrm{e}^{\mathrm{i}\,\varphi} = \mathrm{e}^{\log|z| + \mathrm{i}\,\varphi}. \tag{11.1}$$

Here log denotes the (real) natural logarithm. In Russian and Bulgarian literature the natural logarithm is often denoted as ln. The decimal logarithm is written as $\log_{10}$ while in Russia and Bulgaria the notation lg is used for the decimal logarithm. Finally, the logarithm with base $b > 0$, $b \neq 1$, is written as $\log_{\mathrm{b}} b$.

The real logarithm $u = \log x \in \mathbb{R}$ is uniquely defined for $x > 0$ as the solution of the equation $\mathrm{e}^u = x$. In the complex case we shall also define the logarithm w of $z \neq 0$ as a solution of the equation $\mathrm{e}^w = z$. The solution here is not unique. In particular if w_0 is any logarithm of z then $w = w_0 + 2k\pi\mathrm{i}$ shall also be a logarithm of z.

Definition 11.1 The quantity

$$\log z = \log|z| + \mathrm{i}\arg z$$

is the *principal logarithm* of $z \neq 0$.

Consider now complex powers of the form z^c, where both the base z and the exponent c are complex (real in particular) quantities. Usually it is assumed that $z \neq 0$ but the case $z = 0$ may also be of interest.

The next definition of the powers 0^c is in accordance with the MATLAB notation, where `NaN` is Not a Number, `Inf` is ∞ and `i` stands for the imaginary unit i.

Definition 11.2 *The power 0^c is defined as follows: (i) if $c \in \mathbb{C}\backslash\mathbb{R}$ (here by necessity $c \neq 0$) then 0^c is Not a Number of the form* `NaN + Nani` *, (ii) if $c < 0$ then 0^c is* `Inf`*, (iii) if $c = 0$ then $0^0 = 1$, and (iv) if $c > 0$ then $0^c = 0$.*

The case $z \neq 0$ follows from (11.1).

Definition 11.3 *Let* $z \neq 0$. *Then the* principal power z^c *for any* $c \in \mathbb{C}$ *is defined from*

$$z^c = \mathrm{e}^{c(\log|z| + \mathrm{i}\arg z)}. \tag{11.2}$$

In particular the *principal* m*–th root* ($m \in \mathbb{N}$) of z is 0 when $z = 0$, and

$$z^{1/m} = |z|^{1/m}\mathrm{e}^{(\mathrm{i}\arg z)/m}, \ z \neq 0.$$

For $c = a + \mathrm{i}b$ the power (11.2) may be written as

$$z^c = \rho \mathrm{e}^{\mathrm{i}\psi},$$

where

$$\begin{aligned} \rho &= \rho(z, c) := \mathrm{e}^{a\log|z| - b\arg z}, \\ \psi &= \psi(z, c) := a\,\mathrm{arg}z + b\log|z|. \end{aligned}$$

Here ψ may or may not be the principal argument of z^c.

Consider now complex functions of complex argument. Let $D \subset \mathbb{C}$ be a domain in the complex plane $\mathbb{C}$, i.e. an open and connected set. Certain domains are described in the next example, where $z_0 \in \mathbb{C}$ and $0 \le s < r$.

Example 11.1 The sets listed below are domains: (i) the whole plane $\mathbb{C}$, (ii) the disc $B_r(z_0) = \{z : |z - z_0| < r\}$ with center z_0 and radius r, (iii) the ring $R_{s,r}(z_0) = \{z : s < |z - z_0| < r\}$.

Suppose that the complex power series

$$\sum_{k=0}^{\infty} a_k (z - z_0)^k \tag{11.3}$$

is convergent in the disc $B_r(z_0)$, $r > 0$, and denote its sum by $f(z)$. Then $f : B_r(z_0) \to \mathbb{C}$ has derivatives $f^{(k)}(z_0) = k!a_k$ of any order $k \in \mathbb{N}$.

Definition 11.4 Functions which may be expanded in power series of type (11.3) are called *holomorphic*, or *analytical* in the corresponding domain. A holomorphic function defined by given power series in the whole plane $\mathbb{C}$ is called an *entire* function.

Real functions that may be represented as power series are the exponential functions, the trigonometric functions, the hyperbolic functions, the logarithmic functions, etc. The corresponding complex power series define complex functions which are analytical continuations of the corresponding real functions. Examples of such functions are

$$\begin{aligned}
\mathrm{e}^x &= \sum_{k=0}^{\infty} \frac{x^k}{k!}, \\
\sin x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}, \\
\cos x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}, \\
\sinh x &= \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}, \\
\cosh x &= \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}, \\
\log(1+x) &= \sum_{k=1}^{\infty} (-1)^{k-1} \frac{x^k}{k}, \ -1 < x < 1, \\
logx &= \log(1+x-1) = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{(x-1)^k}{k}, 0 < x < 2.
\end{aligned}$$

11.3 General considerations

Denote by $\widehat{y}$ the value of y, computed in machine arithmetic of type $(\mathbf{u}, X_{\max})$. Let us assume for simplicity that x and y are scalars. The case of vector (matrix) functions of vector (matrix) arguments is treated in a similar way. We shall also assume that the function f is differentiable and has a bounded derivative in a certain neighborhood of the argument x.

There are two main reasons which determine the absolute

$$E := |\widehat{y} - y|$$

and the relative

$$e := \frac{|\widehat{y} - y|}{|y|}, \ y \neq 0,$$

errors in the computed solution $\widehat{y}$. We shall consider these factors under the assumption that

$$E_{\min} < |x|, \ |y| < X_{\max}$$

and that during the computations all intermediate quantities are also in the normal range of the machine arithmetic. We also suppose that the computations are organized so that destructive cancellations may not occur.

The first source of errors is related to the (eventual) rounding of the argument x to the nearest machine number

$$\widehat{x} = x(1+\varepsilon),\ \ |\varepsilon| \le \mathbf{u}.$$

Note that when an arithmetic with base $\beta = 2$ or $\beta = 2^4 = 16$ is used, numbers such as 1/3, 1/5 or 1/10 are rounded.

Thus when storing the data x in the computer memory we make an absolute error

$$|\widehat{x} - x| \le \mathbf{u}\,|x|$$

and a relative error

$$\frac{|\widehat{x} - x|}{|x|} \le \mathbf{u},\ \ x \ne 0,$$

in the argument x. Therefore, in the most favorable case we may expect that the relative error in the result will not exceed considerably the rounding unit $\mathbf{u}$.

The second source of errors is the rounding in the performance of arithmetic operations in the computation of $f(\widehat{x})$ using the initial data $\widehat{x}$. Denote by $\widehat{f}(\widehat{x})$ the result of the implementation of the algorithm $\widehat{x} \mapsto \widehat{f}(\widehat{x})$ in machine arithmetic. Then we have

$$\widehat{y} = \widehat{f}(\widehat{x}).$$

Furthermore, for the absolute error we obtain

$$\begin{aligned} E &= |\widehat{f}(\widehat{x}) - f(x)| = |f(\widehat{x}) - f(x) + \widehat{f}(\widehat{x}) - f(\widehat{x})| \\ &\le |f(\widehat{x}) - f(x)| + |\widehat{f}(\widehat{x}) - f(\widehat{x})| =: E_1 + E_2, \end{aligned}$$

where $E_1 := |f(\widehat{x}) - f(x)|$ and $E_2 := |\widehat{f}(\widehat{x}) - f(\widehat{x})|$ are estimates for the contribution of the above two sources of errors in forming the absolute error E.

Within small terms of first order relative to $\mathbf{u}$ we get

$$E_1 \le |f'(x)|\,|\widehat{x} - x| \le \mathbf{u}|f'(x)|\,|x|$$

and

$$E_2 \le \mathbf{u}\,\Gamma,$$

where the constant $\Gamma = \Gamma(f, x) > 0$ depends on the function f and the argument x as well as on the algorithm used to compute $\widehat{f}(\widehat{x})$. Hence

$$E \leq \mathbf{u}(|f'(x)|\,|x| + \Gamma)$$

and

$$e \leq \mathbf{u}(k_f + \gamma), \ y = f(x) \neq 0. \tag{11.4}$$

Here

$$k_f = k_f(x) := \frac{|f'(x)|\,|x|}{|f(x)|}$$

is the relative condition number in the computation of the value of f at the point x, and

$$\gamma = \gamma(f, x) := \frac{\Gamma(f, x)}{|f(x)|}.$$

The reader is advised to compare the estimate (11.4) with the estimate (3.7) from Section 3.4.

We shall stress that while $k_f(x)$ depends only on the problem, i.e. on the function f and the argument x, the quantity $\gamma(f, x)$ depends also on the algorithm used.

Based on the above considerations we come to the following conclusion.

Proposition 11.1 *The relative error e in computing the value $f(x)$ of the function f at the point x may be large when the absolute values of the argument x and/or the derivative $f'(x)$ are large, or when the absolute value of $f(x)$ is small.*

To decrease the size $|x|$ of the argument x one can use various techniques as shown below (see also [2]).

Example 11.2 Consider the computation of $y = \log x$ in machine arithmetic with base $\beta = 2$. The argument $x \in [E_{\min}, X_{\max}]$ can always be represented as $x = m\,2^p$, where the integer p is the exponent of x and $m \in [0.5, 1]$ is the mantissa of x. Then one may use the next formula, based on rapidly convergent series

$$\begin{aligned} \ln x &= (p - 0.5)\ln x + 2u \sum_{i=0} \frac{u^{2i}}{2i+1}, \\ u &:= \frac{m\sqrt{2} - 1}{m\sqrt{2} + 1}, \ |u| \leq 3 - 2\sqrt{2} < 0.1716. \end{aligned}$$

Example 11.3 For reducing twice the size of the argument in some elementary functions one may use the following formulae:

$$\begin{aligned}
\exp x &= (\exp u)^2, \\
\cos x &= 2(\cos u)^2 - 1, \\
\sin x &= 2 \sin u \sqrt{1 - \sin^2 u}, \\
\tan x &= \frac{2 \tan u}{1 - \tan^2 u}, \\
\cot x &= \frac{1}{2}\left(\cot u - \frac{1}{\cot u}\right),
\end{aligned}$$

where $u := x/2$.

An important particular case is the evaluation of a polynomial function, i.e. the computation of

$$f(x) = \sum_{k=0}^{n} a_k x^k,$$

where the coefficients a_k and the argument x are real or complex numbers. The "naive" approach here is to use directly the formula for $f(x)$, computing first $y_{k,k} := a_k x^k$ for $k = 1, 2, \ldots, n$ by k multiplications,

$$y_{k,l} = x y_{k,l-1}(i), \; l = 1, 2, \ldots, k, \; y_{k,0} := a_k,$$

and then summing the quantities $a_0, y_{1,1} = a_1 x, \ldots, y_{n,n} = a_n x^n$ by n additions. This gives $1 + 2 + \cdots + n = n(n+1)/2$ multiplications plus n additions.

A more effective and more accurate procedure is to compute $f(x)$ as

$$f(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \cdots),$$

or, equivalently, by the algorithm $f(x) = z_n$, where

$$\begin{aligned}
z_0 &:= a_n, \\
z_k &:= a_{n-k} + x z_{k-1}, \; k = 1, 2, \ldots, n.
\end{aligned}$$

This approach known as the Horner rule, requires only n multiplications and n additions.

Example 11.4 For $n = 4$ the quantity

$$f(x) = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

may be computed by 4 multiplications and 4 additions as

$$f(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + x a_4))).$$

11.4 Computations with maximum accuracy

When the computations are done with the maximum achievable accuracy in the particular machine arithmetic, then the computed value $\widehat{y}$ of y will be the closest to y machine number round(y), which means that

$$\widehat{y} = y(1+\delta), \ |\delta| \leq \mathbf{u}.$$

Hence in this case it is fulfilled

$$E \leq \mathbf{u}\,|y|$$

and

$$e \leq \mathbf{u}, \ y \neq 0.$$

Note that elementary functions such as exp, sin, cos and others, are usually computed with maximum achievable accuracy in most modern computer systems for mathematical calculations such as MATLAB, MATHEMATICA, MAPLE, etc.

11.5 Function evaluation by MATLAB

The computer system MATLAB (from MATrix LABoratory), developed by MathWorks, Inc., has many in–built elementary and specialized functions.

The HELP topics are called by the command `help`. The computation of elementary functions is done by the commands from the topic `elfun`. The help for this topic is called by `help elfun`. We recall that the argument of any function may be a real or complex quantity. For many functions it may as well be a vector or a matrix.

We recall that In MATLAB the constants `i` and `j` are used for the imaginary unit $\mathrm{i} = \sqrt{-1}$ if they are not preassigned.

Next we describe the main functions that are included in MATLAB.

The following set of commands is used to perform simple complex operations.

Complex operations

`abs`	- Absolute value (module)
`angle`	- Phase angle, or argument
`complex`	- Construct complex data from real and imaginary parts
`conj`	- Complex conjugate
`imag`	- Complex imaginary part
`real`	- Complex real part
`unwrap`	- Unwrap phase angle (see `help unwrap`)
`isreal`	- True for real array
`cplxpair`	- Sort numbers into complex conjugate pairs

Example 11.5 The command

```
>> complex(2,-3)
```

will create the complex number $z = 2 - 3\mathrm{i}$ as

```
>> z =
      2.0000 - 3.0000i
```

(the same result is obtained from `z = 2 - 3*i`).

Trigonometric and hyperbolic functions

`sin`	- Sine
`sinh`	- Hyperbolic sine
`asin`	- Inverse sine (Arcus sine)
`asinh`	- Inverse hyperbolic sine
`cos`	- Cosine
`cosh`	- Hyperbolic cosine
`acos`	- Inverse cosine (Arcus cosine)
`acosh`	- Inverse hyperbolic cosine
`tan`	- Tangent
`tanh`	- Hyperbolic tangent
`atan`	- Inverse tangent (Arcus tangent)
`atanh`	- Inverse hyperbolic tangent
`sec`	- Secant
`sech`	- Hyperbolic secant
`asec`	- Inverse secant (Arcus secant)
`asech`	- Inverse hyperbolic secant
`csc`	- Cosecant
`csch`	- Hyperbolic cosecant
`acsc`	- Inverse cosecant (Arcus cosecant)
`acsch`	- Inverse hyperbolic cosecant
`cot`	- Cotangent
`coth`	- Hyperbolic cotangent
`acot`	- Inverse cotangent (Arcus cotangent)
`acoth`	- Inverse hyperbolic cotangent

Rounding and remainder

`fix`	- Round towards zero
`floor`	- Round towards minus infinity
`ceil`	- Round towards plus infinity
`round`	- Round towards nearest integer
`mod`	- Modulus (signed remainder after division)
`rem`	- Remainder after division
`sign`	- Signum

Exponential, logarithmic and power functions

`exp`	- Base e exponential
`log`	- Natural (base e) logarithm
`log10`	- Common (base 10) logarithm
`log2`	- Base 2 logarithm and dissect floating point number
`pow2`	- Base 2 power and scale floating point number
`sqrt`	- Square root
`nextpow2`	- Next higher power of 2

If z and c are given (complex) quantities then the MATLAB command

```
>> y = z^c
```

will produce the (complex) power z^c. Since the function $z \mapsto z^c$ may be multi–valued (or even infinitely–valued), a special convention is necessary to determine the corresponding branch of this function and thus to correctly interpret the result y. In MATLAB the powers z^c are computed according to Definitions 11.2 and 11.3.

Example 11.6 The MATLAB command

```
>> A = [1 2; 3 4]
```

(note that there must be a blank between 1 and 2 and between 3 and 4) will produce the matrix

```
A =
    1  2
    3  4
```

Next the command

```
>> X = sin(A)
```

will give

```
X =
    0.8415    0.9093
    0.1411   -0.7568
```

which is an approximation (with 4 significant digits) to the matrix $\begin{bmatrix} \sin(1) & \sin(2) \\ \sin(3) & \sin(4) \end{bmatrix}$.

Typing `format long` and then `X` we shall see the matrix X with 15 significant decimal digits.

The computation of specialized functions (including two functions of vectors), some number theoretic functions as well as coordinate transforms can be performed by the commands from the topic `specfun`. The help for this topic is called by `help specfun`. The following functions are included (for some of the functions the argument may be real or complex, as well as scalar, vector or matrix).

Specialized functions

`airy`	- Airy functions
`besselj`	- Bessel function of the first kind
`bessely`	- Bessel function of the second kind
`besselh`	- Bessel function of the third kind (Hankel function)
`besseli`	- Modified Bessel function of the first kind
`besselk`	- Modified Bessel function of the second kind
`beta`	- Beta function
`betainc`	- Incomplete beta function
`betaln`	- Logarithm of beta function
`ellipj`	- Jacobi elliptic integral
`ellipke`	- Complete elliptic integral
`erf`	- Error function
`erfc`	- Complementary error function
`erfcx`	- Scaled complementary error function
`erfinv`	- Inverse error function
`expint`	- Exponential integral function
`gamma`	- Gamma function
`gammainc`	- Incomplete gamma function
`gammaln`	- Logarithm of gamma function
`legendre`	- Associated Legendre function
`cross`	- Vector cross (vector) product
`dot`	- Vector dot (scalar) product

Number theoretic functions

`factor`	- Prime factors
`isprime`	- True for prime factors
`primes`	- Generate list of prime numbers
`gcd`	- Greatest common divisor
`lcm`	- Least common multiple
`rat`	- Rational approximation
`rats`	- Rational output
`perms`	- All possible permutations
`nchoosek`	- All combinations of n elements taken k at a time
`factorial`	- Factorial function

Coordinate transforms

`cart2sph`	- Transform Cartesian to spherical coordinates
`cart2pol`	- Transform Cartesian to polar coordinates
`pol2cart`	- Transform polar to Cartesian coordinates
`sph2cart`	- Transform spherical to Cartesian coordinates
`hsv2rgb`	- Convert hue-saturation-value colors to red-green-blue
`rgb2hsv`	- Convert red-green-blue colors to hue-saturation-value

The description of each function can be found by the command `help`. For example,

```
>> help acsch
```

invokes the description of the function for computing the inverse hyperbolic cosecant.

Example 11.7 Suppose that we want to compute the 5–vector y with elements

$$y_k = \frac{10\cos k}{1 + k\exp k},\ k = 1, 2, \ldots, 5.$$

By the commands

```
>> for k=1:5,
       x(k) = k;
   end
```

we form the vector of arguments $x = [1, 2, 3, 4, 5]$. Then the command

```
>> y = 10*\cos(x)./(1. + x.*\exp(x))
```

(the dots are for the array operations) will produce the result

```
        y  =
                 1.4531  − 0.2637  − 01616  − 0.0298  − 0.0038
```

Using the command `format long` and typing `y` we shall obtain the result with full precision.

The evaluation of polynomials (real or complex) in MATLAB is done as follows. The polynomial

$$P(x) = p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1} = \sum_{k=0}^{n} p_{n+1-k}x^k \tag{11.5}$$

of degree $\deg(P) = n$ is uniquely determined by the row $(n+1)$–vector

$$p := [p_1, p_2, \ldots, p_{n+1}]$$

with $p_1 \neq 0$. Thus, the vectors $[1, 0, 0]$ and $[1, 0, 0, 0]$ define the polynomials x^2 and x^3, respectively.

Note that usually in algebra textbooks polynomials are written in a slightly different form as

$$p_0x^n + p_1x^{n-1} + \cdots + p_{n-1}x + p_n = \sum_{k=0}^{n} p_{n-k}x^k$$

or

$$p_nx^n + p_{n-1}x^{n-1} + \cdots + p_1x + p_0 = \sum_{k=0}^{n} p_kx^k.$$

In MATLAB, however, only the form (11.5) is used (remember that here the first element $p_1 = p(1)$ of the coefficient vector p multiplies the highest degree of the variable x).

In what follows we shall identify any polynomial P of degree n with the $(n+1)$–vector p of its coefficients.

Let now X be a vector with elements $x_1, x_2, \ldots, x_n$, at which we want to evaluate the polynomial P. Then the command

```
>> Y = polyval(p,X)
```

returns the n–vector Y with elements $y_k = P(x_k)$, $k = 1, 2, \dots, n$.

Let now $r = [r_1, r_2, \dots, r_n]$ be a given n–vector. The coefficients $s_1, s_2, \dots, s_{n+1}$ of the monic n–th degree polynomial

$$S(x) := (x - r_1)(x - r_2)\cdots(x - r_n)$$

with roots $r_1, r_2, \dots, r_n$ may be computed by the function

```
>> s = poly(r)
```

Note that here

$$\begin{aligned} s_1 &= 1, \\ s_2 &= -(r_1 + r_2 + \cdots + r_n), \\ &\vdots \\ s_{n+1} &= (-1)^n r_1 r_2 \cdots r_n. \end{aligned}$$

The product (or convolution) $T = PQ$ of degree $n + m$ of two polynomials P and Q with coefficient vectors $p = [p_1, p_2, \dots, p_{n+1}]$ and $q = [q_1, q_2, \dots, q_{m+1}]$, respectively, is found by the command

```
>> t = conv(p,q)
```

Here t is the coefficient row $(n + m + 1)$–vector of T with elements

$$\begin{aligned} t_1 &= p_1 q_1, \\ t_2 &= p_1 q_2 + p_2 q_1, \\ &\vdots \\ t_{n+m+1} &= p_{n+1} q_{m+1}. \end{aligned}$$

The command

```
>> [s,r] = deconv(p,q)
```

divides the polynomials P and Q. If

$$\frac{P}{Q} = S + \frac{R}{Q},$$

where S is the quotient polynomial and R is the remainder polynomial, then s and r are the coefficient vectors of S and R, respectively. In the default case $\deg(P) \geq \deg(Q)$ we have $\deg(R) < \deg(Q)$ and $\deg(P) = \deg(S) + \deg(Q)$. If $\deg(P) < \deg(Q)$ then S is the zero polynomial (void) and $R = P$.

The command

```
>> [r,u,s] = residue(p,q)
```

computes the partial–fraction expansion of the ratio P/Q of the polynomials P and Q of degrees n and m, respectively. If Q has no multiple zeros, then

$$\frac{P(x)}{Q(x)} = S(x) + \frac{r_1}{x-u_1} + \frac{r_2}{x-u_2} + \cdots + \frac{r_m}{x-u_m}.$$

If $u_k = u_{k+1} = \cdots = u_{k+l-1}$ is a root of Q of multiplicity l, it corresponds to the expression

$$\frac{r_k}{x-u_k} + \frac{r_{k+1}}{(x-u_k)^2} + \cdots + \frac{r_{k+l-1}}{(x-u_k)^l}.$$

11.6 Problems and exercises

According to Proposition 11.1 in function evaluation one should avoid the use of large arguments whenever possible. In certain cases, even if the initial problem contains large arguments, the computations may be done using arguments of moderate size. The next two problems illustrate this idea.

Exercise 11.1 Write an algorithm for computing the values of the trigonometric functions

$$\sin x,\ \cos x,\ \tan x,\ \cot x$$

for any value of the argument x (but be careful with tan and cot!) using the corresponding values

$$\sin z,\ \cos z,\ \tan z,\ \cot z$$

for

$$0 \le z \le \frac{\pi}{2}$$

(for tan and cot the values $\pi/2$ and 0, respectively, are excluded). Try the same for

$$0 \le z \le \frac{\pi}{4}.$$

Exercise 11.2 Write an algorithm which computes the values of the exponential function e^x for all values of the argument x using the values e^z for $0 \le z \le 1$. Make the same for $0 \le z \le 0.5$.

Exercise 11.3 Two polynomials P and Q may not be added and subtracted directly in MATLAB since in general their coefficient vectors p and q will have different lengths when $\deg(P) \neq \deg(Q)$. Write a program `polysum` in MATLAB which will compute the sum $P + Q$ of polynomials of any degree.

Chapter 12

Evaluation of matrix functions

12.1 General considerations

In this section we consider two types of matrix functions in MATLAB: element–wise functions and genuinely matrix functions. We do not consider vector functions separately since vectors are matrices with one column or one row. We use the MATLAB notation $X(k,l)$ for the element of the matrix X in position (k,l).

A number of applications of the matrix functions are also discussed.

Let $\mathcal{X} \subset \mathbb{K}^{m\times n}$ and $\mathcal{Y} \subset \mathbb{K}^{p\times q}$ be given non–empty sets, where $\mathbb{K}$ stands for $\mathbb{R}$ or $\mathbb{C}$.

Definition 12.1 *(i) Any map $f : \mathcal{X} \to \mathcal{Y}$ is said to be a* matrix function*. (ii) A matrix function f is said to be* element–wise *if each element $Y(k,l)$ of the resulting matrix $Y = f(X)$ depends only on one element $X(r,s)$ of the data matrix X. (iii) If the matrix function f is not element–wise it is called a* genuine *matrix function.*

Most element–wise functions map $\mathbb{C}^{m\times n}$ in $\mathbb{C}^{m\times n}$ and each element $Y(k,l)$ of Y depends only on the element $X(k,l)$ of X at the same position.

Example 12.1 (i) The shift function $X \mapsto A + X$, where A is a fixed matrix of the size of X, and the H'Adamard multiplication function $X \mapsto A \circ X = X \circ A$, are element–wise matrix functions (we recall that the elements of the product $Y = A \circ X$ are $Y(k,l) = A(k,l)X(k,l)$).

(ii) The transposition $Y = X^\top$ and the complex conjugate transposition $Y = X^{\mathrm{H}} = \overline{A}^\top$ are element–wise matrix functions with $Y(k,l)$ depending on $X(l,k)$.

(iii) The standard matrix multiplication $X \mapsto AX$ (or $X \mapsto XA$) is a genuine matrix function since the element $Y(k,l)$ of Y depends in general on all elements in the l–th column (or in the k–th row) of X

When $\mathcal{X} \subset \mathbb{K}$ we have a matrix valued (vector valued if $\mathcal{Y}$ is a set of vectors) function of a scalar argument, while for $\mathcal{Y} \subset \mathbb{K}$ we have a scalar valued function of a matrix argument.

Example 12.2 (i) Let $v(t) = \dot{x}(t) \in \mathbb{R}^n$ be the velocity vector of a moving particle in $\mathbb{R}^n$ with a vector of coordinates $x(t)$ at the moment $t \in [t_0, t_1]$. Then x and v are vector valued functions of a scalar argument.

(ii) The determinant $X \mapsto \det(X)$ ($X \in \mathbb{K}^{n\times n}$) and the trace $X \mapsto \mathrm{tr}(X)$ are scalar valued functions, while the rank $X \mapsto \mathrm{rank}(X)$ is an integer valued function of a matrix argument.

(iii) The function $X \mapsto y$, where $y(\lambda) = \det(\lambda I - X)$ is the characteristic polynomial of $X \in \mathbb{K}^{n\times n}$, is a vector valued function of a matrix argument since a polynomial may be identified with the vector of its coefficients.

The functions described in Example 12.2, part (ii), are realized in MATLAB by the commands `det(X)`, `trace(X)` and `rank(X)`, respectively (the function `det(X)` requires a square argument X).

The characteristic polynomial of a matrix $X \in \mathbb{K}^{n\times n}$ with $n \geq 2$ is computed in MATLAB by the command

```
>> y = poly(X)
```

Here the result $y = [y_1, y_2, \ldots, y_{n+1}]$ is an $(n+1)$–vector with elements equal to the coefficients of the characteristic polynomial of X. In particular $y_1 = 1$, $y_2 = -\mathrm{tr}(A)$ and $y_{n+1} = (-1)^{n+1}\det(X)$.

We stress that the command `poly` is also used for vectors. If $x = [x_1, x_2, \ldots, x_n]$ is a given n–the vector then the command

```
>> y = poly(x)
```

will produce an $(n+1)$–vector $y = [1, y_2, \ldots, y_{n+1}]$ corresponding to the polynomial $\lambda^n + y_2\lambda^{n-1} + \cdots + y_n\lambda + y_{n+1}$ with roots $x_1, x_2, \ldots, x_n$. In particular $y_2 = -(x_1 + x_2 + \cdots + x_n)$ and $y_{n+1} = (-1)^n x_1 x_2 \cdots x_n$. For example, when x is a scalar then $y = [1, -x]$.

12.2 Element–wise matrix functions

A simple element–wise matrix function is the multiplication of the argument by a scalar. When a is a scalar and X is a matrix the command

```
>> Y = a*X
```

will compute the matrix $Y = aX$ with elements $Y(k,l) = aX(k,l)$.

The summation and subtraction $Y = A \pm X$ with A fixed define element–wise functions which are performed by the commands `+` and `-`. Here usually A and X are matrices of the same size and then $Y(k,l) = A(k,l) \pm X(k,l)$.

When a is a scalar and X is a matrix the command

```
>> Y = a + X
```

will add the quantity a to each element of X, i.e. $Y(k,l) = a + X(k,l)$. The same result will be computed by the command

```
>> Y = X + a*ones(size(X))
```

since $Y = X + aE$, where E is a matrix of the size of X with all its elements equal to 1. In MATLAB form we have `E = ones(size(X))`.

There is a set of commands in MATLAB for computing element–wise functions which are known under the name *dot commands*. They have a dot in their syntax, see also Section 4.6.

The H'Adamard multiplication $Y = A \circ X$ is performed by the dot command `.*`, namely

```
>> Y = A.*X
```

This gives the matrix Y with elements $Y(k,l) = A(k,l)X(k,l)$.

Let the matrix X has no zero elements and A be of the size of X. Then we may define the element–wise division

```
>> Y = X.\A
```

Here the resulting matrix Y has elements $Y(k,l) = A(k,l)/X(k,l)$. The same result is obtained by the command `Y = A./X`.

In addition to this “matrix by matrix” division, MATLAB has an option for “scalar by matrix” division. Indeed, let a be a scalar and let the matrix X has no zero elements. Then the command

```
>> Y = X.\a
```

will produce the matrix Y with elements $Y(k,l) = a/X(k,l)$. The same result will be computed from `Y = a./X`.

The next four element–wise matrix functions are determined in view of Definitions 11.2 and 11.3.

Let as before X and A be matrices of same size. We also suppose that if $X(k,l) = 0$ then the element $A(k,l)$ is real and non–negative. Then the scalar power $X(k,l)^{A(k,l)}$ is determined according to Definitions 11.2 and 11.3. Now the MATLAB command

```
>> Y = X.^A
```

yields the matrix Y with elements $Y(k,l) = X(k,l)^{A(k,l)}$. Thus we have defined a matrix element–wise power function.

Suppose now that if $A(k,l) = 0$ then $X(k,l)$ is real and non–negative. Then the scalar power $A(k,l)^{X(k,l)}$ is again determined according to Definitions 11.2 and 11.3. Under this assumption the MATLAB command

```
>> Y = A.^X
```

produces the matrix Y with elements $Y(k,l) = A(k,l)^{X(k,l)}$. Thus we have defined a matrix element–wise exponential function.

Let $a \in \mathbb{C}$ with the convention that if X has a zero element then a is real and non–negative. Then the command

```
>> Y = X.^a
```

yields the matrix Y with elements $Y(k,l) = (X(k,l))^a$. Thus we have defined a matrix element–wise power function.

Let again $a \in \mathbb{C}$ with the convention that if $a = 0$ then all elements of X are real and non–negative, Then the command

```
>> Y = a.^X
```

gives the matrix Y with elements $Y(k,l) = a^{X(k,l)}$. This is yet another matrix element–wise exponential function.

Element–wise matrix functions of $X \in \mathbb{K}^{m\times n}$ may be computed by the command

```
>> Y = fun(X)
```

where `fun` stands e.g. for `sin`, `exp`, or some other elementary or specialized scalar function which is build–in MATLAB. Here Y is also an $m \times n$ matrix with elements

$$Y(k,l) = \mathrm{fun}(X(k,l)).$$

Thus for element–wise matrix functions the element $Y(k,l)$ of the result Y depends only on the element $X(k,l)$ of the data X at the same position (k,l).

Example 12.3 Let $X = \begin{bmatrix} 0 & \pi/2 & \imath \\ \pi/6 & -1 & 10^6 \end{bmatrix}$. Then the commands `S = sin(X)`, `C = cos(X)` and `E = exp(X)` will return (in `format short`) the matrices

```
S =
   0.0000        1.0000             0 + 1.1752i
   0.5000       -0.8415       -0.3500
C =
   1.0000        0.0000        1.5431
   0.8660        0.5403        0.9368
E =
   1.0000        4.8105        0.5404 + 0.8415i
   1.6881        0.3679           Inf + Nani
```

respectively. It may be observed that $\sin(\imath) = \imath\sinh(1)$, $\cos(\imath) = \cosh(1)$ and $e^{\imath} = \cos(1) + \imath\sin(1)$. If we compute the element $E(2,3)$ separately as

```
>> exp(10^6)}
```

the result is `Inf` as expected. The same quantity was somehow computed as `Inf + Nani` using the component–wise command `exp(E)`.

12.3 Genuine matrix functions

Examples of genuine matrix functions are given by $Y = AXB$ or $Y = AX^{-1}B$ if X is invertible. More generally, a polynomial matrix function may be defined from

$$Y = \sum_{k=0}^{r} L_k(X),$$

where $L_k(X)$ is a k–linear form in X. In particular $L_0(X)$ is a constant matrix. It may be shown that $L_1(X)$ is a sum of terms of the form A_1XA_2 and $L_k(X)$, $k > 1$, is a sum of terms of the form $A_1XA_2X \cdots XA_kXA_{k+1}$. Note that here the matrices X and Y may have any size and the matrix coefficients A_k are such that all standard matrix products are correctly defined.

Example 12.4 The coefficients c_k of the characteristic polynomial

$$\lambda^n - c_1\lambda^{n-1} + c_2\lambda^{n-2} - \cdots + (-1)^n c_n$$

of the matrix X are k–linear forms in X.

Let $P_1, P_2, \ldots, P_k$ and $Q_1, Q_2, \ldots, Q_k$ be polynomial functions such that $Q_i(X)$ are square invertible matrices for $X \in \mathcal{X}$. Then a general fractional–polynomial function may be defined as a sum of terms of the form

$$P_1(X)Q_1^{-1}(X)P_2(X)Q_2^{-1}(X)\cdots P_k(X)Q_k^{-1}(X),\ X \in \mathcal{X}.$$

The integer powers of $X \in \mathbb{C}^{n\times n}$ are defined recursively as follows. We set $X^0 = I_n$ and $X^m = X(X^{m-1})$ for $m \in \mathbb{N}$. If X is invertible then $X^{-m} = (X^{-1})^m = (X^m)^{-1}$ for $m \in \mathbb{N}$.

Integer matrix powers are computed in MATLAB by the command

```
>> Y = X^m
```

with the convention that X must be invertible if $m < 0$. The same command may be used for non–integer values of m but this must be done with some precautions as shown below.

A particular case of a polynomial matrix function is the function

$$Y = p(X) = \sum_{k=0}^{r} p_{r+1-k}X^k,\ X \in \mathbb{C}^{n\times n}, \tag{12.1}$$

where p_k are scalars. It is the matrix counterpart of the scalar r–th degree polynomial

$$p(x) = p_1x^r + p_2x^{r-1} + \cdots + p_rx + p_{r+1},\ x \in \mathbb{C},$$

which is characterized by the row $(r+1)$–vector $p := [p_1, p_2, \ldots, p_{r+1}]$. The matrix Y from (12.1) is computed in MATLAB by the command

```
>> Y = polyvalm(p,X)
```

More general genuine matrix functions $\mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ of matrix argument may be defined in three main ways: (i) by matrix power series, (ii) by the Jordan or Schur form of the matrix argument and (iii) by contour matrix integrals in the complex domain.

Suppose first that

$$f(x) := \sum_{k=0}^{\infty} a_k x^k,\ x \in \mathbb{C}, \tag{12.2}$$

is complex power series ($a_k \in \mathbb{C}$), convergent in the disc $|x| < R$. Then we may define the matrix power series

$$f(X) := \sum_{k=0}^{\infty} a_k X^k,\ X \in \mathbb{C}^{n\times n}, \tag{12.3}$$

which is convergent for $\|X\| < R$. Note that for simplicity we use the same letter f for both the scalar function in (12.2) and for its matrix counterpart in (12.3).

We stress that the matrix series (12.3) may be written in a form containing powers of X which are less than n. Indeed, according to the famous Caily-Hamilton theorem the matrix X satisfies its characteristic polynomial $p(x) = x^n - c_1 x^{n-1} + \cdots + (-1)^n c_n = 0$, where c_k is the sum of principal minors of X of k–th order. In particular $c_1 = \mathrm{tr}(X)$ and $c_n = \det(X)$. Hence we have

$$X^n = c_1 X^{n-1} - c_2 X^{n-2} + \cdots + (-1)^{n-1} c_n I_n.$$

Thus all powers X^m, $m \geq n$, may be expressed by the matrices $X^{n-1}, X^{n-2}, \ldots, X, I_n$ and

$$f(X) = \sum_{k=0}^{n-1} b_k X^k,$$

where the coefficients b_k depend on $c_1, c_2, \ldots, c_n$ and $a_0, a_1, a_2, \ldots$. However, this way to find $f(X)$ may be unreliable since the coefficients c_k of the characteristic polynomial of X may not be computed accurately.

If $X = VZV^{-1}$, where the matrix $V \in \mathbb{C}^{n\times n}$ is non–singular, then we have the important equality

$$f(X) = Vf(Z)V^{-1}.$$

This allows to compute $f(X)$ via the matrix $f(Z)$. When Z is a canonical form of X (e.g. the Jordan or Schur form) the computation of $f(Z)$ may be much more easier in comparison with this of $f(X)$.

Using the well known power series for the exponential, sine and cosine scalar functions, we may define their matrix counterparts as

$$\mathrm{e}^X = \sum_{k=0}^{\infty} \frac{X^k}{k!} = I_n + \frac{X}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \cdots,$$

$$\begin{aligned}\sin(X) &= \sum_{k=1}^{\infty}(-1)^{k+1}\frac{X^{2k-1}}{(2k-1)!} = \frac{X}{1!} - \frac{X^3}{3!} + \frac{X^5}{5!} + \cdots, \qquad (12.4)\\ \cos(X) &= \sum_{k=0}^{\infty}(-1)^{k}\frac{X^{2k}}{(2k)!} = I_n - \frac{X^2}{2!} + \frac{X^4}{4!} - \frac{X^6}{6!} + \cdots.\end{aligned}$$

We also have the hyperbolic matrix functions

$$\begin{aligned}\sinh(X) &= \sum_{k=1}^{\infty}\frac{X^{2k-1}}{(2k-1)!} = \frac{X}{1!} + \frac{X^3}{3!} + \frac{X^5}{5!} + \cdots, \qquad (12.5)\\ \cosh(X) &= \sum_{k=0}^{\infty}\frac{X^{2k}}{(2k)!} = I_n + \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!} + \cdots.\end{aligned}$$

These five matrix series are convergent for all matrix arguments X. In particular we have $\mathrm{e}^0 = \cos(0) = \cosh(0) = I_n$ and $\sin(0) = \sinh(0) = 0$.

The matrix exponential function $X \rightarrow \mathrm{e}^X$, defined by the first equality in 12.4, is widely used in many scientific and engineering applications. Some applications to the solution of differential equations are listed below.

The matrix exponential has the following properties (among others)

$$\begin{aligned}\mathrm{e}^X\mathrm{e}^Y &= \mathrm{e}^{X+Y},\ XY = YX, \qquad (12.6)\\ \det(\mathrm{e}^X) &= \mathrm{e}^{\mathrm{tr}(X)}.\end{aligned}$$

It follows from the first property in 12.6 that

$$\mathrm{e}^{tX}\mathrm{e}^{sX} = \mathrm{e}^{(t+s)X},\ t, s \in \mathbb{C}, \qquad (12.7)$$

$$\mathrm{e}^X\mathrm{e}^{-X} = I_n. \qquad (12.8)$$

If H is an (usually small) increment of X we have the following estimate

$$\|\mathrm{e}^{X+H} - \mathrm{e}^X\| \le \|H\|\mathrm{e}^{\|H\|}\mathrm{e}^{\|X\|}.$$

This estimate is reachable for some matrices X and H but it may be too pessimistic when the matrix X has negative eigenvalues.

The matrix trigonometric and hyperbolic functions also have many applications. When the matrices X and Y commute we have the following generalizations of the well known trigonometric identities

$$\begin{aligned}\sin(X+Y) &= \sin(X)\cos(Y) + \cos(X)\sin(Y),\\ \cos(X+Y) &= \cos(X)\cos(Y) - \sin(X)\sin(Y).\end{aligned}$$

Consider now a differentiable matrix function of a scalar argument $F : \mathbb{R} \to \mathbb{C}^{n\times n}$. When $F(t) = tX$ for some $X \in \mathbb{C}^{n\times n}$ we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{e}^{tX} = X\mathrm{e}^{tX}.$$

In the general case it is fulfilled

$$\frac{\mathrm{d}\mathrm{e}^{F(t)}}{\mathrm{d}t} = \int_0^1 \mathrm{e}^{(1-s)F(t)}\frac{\mathrm{d}F(t)}{\mathrm{d}t}\mathrm{e}^{sF(t)}\mathrm{d}s.$$

Based on these properties the matrix exponential function may be used to express the solution of the vector differential homogeneous equation

$$y'(t) = Ay(t),\ t \in \mathbb{R}, \tag{12.9}$$

with initial condition

$$y(0) = y_0 \in \mathbb{C}^n, \tag{12.10}$$

as

$$y(t) = \mathrm{e}^{At}y_0.$$

In SYSLAB there is a special command `vecc` to compute the solution of the initial value problem (12.9), (12.10) at the moments $h, 2h, \ldots, Nh$, where $N \in \mathbb{N}$ and $h > 0$ is a given increment in t. The command

```
>> C = vecc(A,y0,h,N)
```

will produce the $(N+1)\times(n+1)$ matrix

$$C := \begin{bmatrix} 0 & y_0^\top \\ h & y^\top(h) \\ 2h & y^\top(2h) \\ \vdots & \vdots \\ Nh & y^\top(Nh) \end{bmatrix}.$$

The similar SYSLAB command

```
>> D = vecd(A,y0,N)
```

computes the solution

$$y(k) = A^k y_0$$

of the discrete system

$$y(k+1) = Ay(k),\ k = 0, 1, 2, \ldots,$$

with initial condition $y(0) = y_0$ at the moments $0, 1, \ldots, N$.

Similarly, the solution of the vector differential equation

$$y''(t) + Ay(t) = 0,\ t \in \mathbb{R}, \tag{12.11}$$

with initial conditions

$$y(0) = y_0 \in \mathbb{C}^n,\ y'(0) = y_0' \in \mathbb{C}^n, \tag{12.12}$$

is given by

$$y(t) = \cos(t\sqrt{A})y_0 + \sin(t\sqrt{A})(\sqrt{A})^{-1}y_0'.$$

Here it is supposed that the matrix $A \in \mathbb{C}^{n\times n}$ is Hermitian ($A = A^{\mathrm{H}}$) and positive definite, and that $\sqrt{A}$ is the (unique) positive definite square root of A.

The solution of the initial value problem (12.11), (12.12) may also be represented using matrix exponentials setting

$$z(t) = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}.$$

Indeed, we have

$$z'(t) = Mz(t),\ M = \begin{bmatrix} 0 & I_n \\ -A & 0 \end{bmatrix} \in \mathbb{C}^{2n\times 2n}$$

and hence

$$\begin{bmatrix} y(t) \\ y'(t) \end{bmatrix} = \mathrm{e}^{tM} \begin{bmatrix} y_0 \\ y_0' \end{bmatrix}.$$

It may be shown that

$$\mathrm{e}^{tM} = \begin{bmatrix} C(t) & S(t) \\ -AS(t) & C(t) \end{bmatrix},$$

where

$$\begin{aligned} C(t) &= \sum_{k=0}^{\infty} \frac{(-A)^k}{(2k)!} t^{2k} = I_n - \frac{A}{2!}t^2 + \frac{A^2}{4!}t^4 - \frac{A^3}{6!}t^6 + \cdots, \qquad (12.13) \\ S(t) &= \sum_{k=0}^{\infty} \frac{(-A)^k}{(2k+1)!} t^{2k+1} = tI_n - \frac{A}{3!}t + \frac{A^2}{5!}t^5 - \frac{A^3}{7!}t^7 + \cdots. \end{aligned}$$

For this representation of the solution the matrix $A \in \mathbb{C}^{n\times n}$ may be arbitrary and in particular it may be singular (for more properties of the functions C and S see Exercise 12.6).

The linear inhomogeneous equation

$$y'(t) = Ay(t) + g(t),\ t \in \mathbb{R}, \tag{12.14}$$

where $g : \mathbb{R} \to \mathbb{C}^n$ is a given continuous function, has a solution

$$y(t) = \mathrm{e}^{tA}c + \int_0^t \mathrm{e}^{(t-s)A}g(s)\mathrm{d}s = \mathrm{e}^{tA}\left(c + \int_0^t \mathrm{e}^{-sA}g(s)\mathrm{d}s\right), \tag{12.15}$$

where $c \in \mathbb{C}^n$ is an arbitrary vector. To solve the initial value problem for (12.14) with initial condition $y(0) = y_0$ we must choose $c = y_0$.

A two–point boundary value problem for equation (12.14) may be formulated by the boundary condition

$$B_0y(0) + B_1y(1) = b, \tag{12.16}$$

where $B_0, B_1 \in \mathbb{C}^{n\times n}$ are given matrices and $b \in \mathbb{C}^n$ is a given vector. If the matrix

$$B = B_0 + B_1\mathrm{e}^A$$

is invertible then the initial value problem (12.14), (12.16) has an unique solution of the form (12.15) with

$$c = B^{-1}\left(b - B_1\int_0^1 \mathrm{e}^{(1-s)A}g(s)\mathrm{d}s\right).$$

In these differential equations the solution vector $y(t) \in \mathbb{C}^n$ at the moment t may be replaced by a matrix $Y(t) \in \mathbb{C}^{n\times m}$, $m > 1$. The reader should make easily the necessary modifications.

Consider now the linear matrix differential equation

$$Y'(t) = G(t) + \sum_{k=1}^{r} A_kY(t)B_k,\ Y(t) \in \mathbb{C}^{n\times m},$$

where $G : \mathbb{R} \to \mathbb{C}^{n\times m}$ is a given function and $A_k \in \mathbb{C}^{n\times n}$, $B_k \in \mathbb{C}^{m\times m}$ are given matrices. This equation may in general be reduced to the form (Xat-n) setting

$$y(t) = \mathrm{vec}(Y(t)) \in \mathbb{C}^{nm},\ g(t) = \mathrm{vec}(G(t)) \in \mathbb{C}^{nm} \tag{12.17}$$

and

$$A = \sum_{k=0}^{r} B_k^\top \otimes A_k \in \mathbb{C}^{nm\times nm}.$$

Hence we may apply the derived formulae for the solution of initial and boundary value problems.

For some simpler matrix differential equations, however, we may give closed form solutions directly in matrix form. Consider for example the equation

$$Y'(t) = AY(t) + Y(t)B + G(t),$$

which is a particular case of equation (12.17). Here the general solution has the form

$$\begin{aligned} Y(t) &= \mathrm{e}^{tA} C \mathrm{e}^{tB} + \int_0^t \mathrm{e}^{(t-s)A} g(s) \mathrm{e}^{(t-s)B} \mathrm{d}s \\ &= \mathrm{e}^{tA} \left(C + \int_0^t \mathrm{e}^{-sA} G(s) \mathrm{e}^{-sB} \mathrm{d}s \right) \mathrm{e}^{tB}. \end{aligned}$$

Of course, the solution of the above linear differential equations may be found approximately by the ODE solvers of MATLAB. But when the computation of the matrix exponential e^{tA} may be done reliably and effectively, the use of the described closed form solutions may give more accurate results.

The matrix exponential $\mathrm{e}^X = \exp(X)$ may be computed in MATLAB by several algorithms. Note that we prefer the notation e^X for the exponential mathematical formulae in order to distinguish it from the MATLAB commands starting with `exp`.

The command

```
>> Y1 = expm(X)
```

is the same as `expm1(X)`. It computes e^X using Padé approximations, namely

$$Y_1 = \left(\sum_{k=0}^{p} a_k X^k \right) \left(\sum_{k=0}^{q} b_k X^k \right)^{-1}.$$

For a low order approximation one may choose $p = q = 1$, $a_0 = b_0 = 2$ and $a_1 = -b_1 = 1$. Similarly, when $p = 2$ and $q = 1$ we have $a_0 = b_0 = 6$, $a_1 = 4$, $a_2 = 1$ and $b_1 = -2$.

For $p = q = 2$ we have $a_0 = b_0 = 12$, $a_1 = -b_1 = 6$ and $a_2 = b_2 = 1$. The last Padé approximation, namely

$$Y_1 = (12I_n + 6X + X^2)(12I_n - 6X + X^2)^{-1}, \tag{12.18}$$

gives a good approximation of the matrix exponential for small matrices X.

Note the difference between the commands `exp(X)` *and* `expm(X)`*!* The first one is element–wise and finds the matrix with elements $\mathrm{e}^{(X(k,l))}$, while the second one computes the matrix exponential e^X as defined in (12.4).

The command

```
>> Y2 = expm2(X)
```

computes e^X using truncated Taylor series

$$Y_2 = \sum_{k=0}^{N} \frac{X^k}{k!} \tag{12.19}$$

We stress that for small X the Padé approximation (12.18) gives better results than the Taylor approximation

$$Y_2 = I_n + X + \frac{X^2}{2} + \frac{X^3}{6} + \frac{X^4}{24},$$

obtained from (12.19) for $N = 4$.

The command

```
>> Y3 = expm3(X)
```

computes the matrix exponential using the eigenstructure of X. In particular, if X is diagonalizable, $X = V\Lambda V^{-1}$, where V is the eigenvector matrix of X, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and λ_k are the eigenvalues of A, then

$$Y_3 = V\mathrm{diag}\left(e^{\lambda_1}, e^{\lambda_2}, \ldots, e^{\lambda_n}\right) V^{-1}.$$

When the matrix X is normal, i.e. $X^{\mathrm{H}}X = XX^{\mathrm{H}}$, then its Schur form is diagonal and hence the eigenvector matrix V may be chosen as unitary. In this case the program `expm3` works very accurately.

In general, when X is not diagonalizable, let $X = VJV^{-1}$, where $J = \mathrm{diag}(J_1, J_2, \ldots, J_r)$ is the Jordan form of X and V is the matrix of eigenvectors and principal vectors of X. Then

$$e^{tX} = Ve^{tJ}V^{-1} = V\mathrm{diag}\left(e^{tJ_1}, e^{tJ_2}, \ldots, e^{tJ_p}\right) V^{-1}.$$

When J_k is the scalar λ we have $e^{tJ_k} = e^{\lambda t}$. Let $J_k = \lambda I_m + N_m \in \mathbb{C}^{m\times m}$, $m > 1$, where N_m is the nilpotent matrix with elements $N_m(k, k+1) = 1$ and $N_m(k, l) = 0$ for $l \neq k + 1$. Then the matrix exponential e^{tJ_k} is computed by a finite sum of m terms as follows

$$e^{tJ_k} = e^{\lambda t}\left(I_m + \frac{tN_m}{1!} + \frac{(tN_m)^2}{2!} + \cdots + \frac{(tN_m)^{m-1}}{(m-1)!}\right).$$

An important function with many applications is the (natural) logarithmic function $x \mapsto y = \log(x)$ such that $e^y = x$. We stress that in the Russian and Bulgarian literature the natural logarithm is usually denoted as $\ln x$.

In the real case the logarithm is defined for $x > 0$. In the complex case we may define an analytical continuation of the real logarithm as follows. For $x \in (0, 2)$ we have

$$\log(x) = \log(1 + (x-1)) = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{(x-1)^k}{k}. \tag{12.20}$$

The power series in the right–hand side of (12.20) is convergent also for complex x from the open disc $D = \{x \in \mathbb{C} : |x-1| < 1\}$. Hence the sum $g(x)$ of this power series is the (unique) analytical continuation of the function $x \mapsto \log(x)$ from the interval $(0, 2)$ onto the disc D.

Definition 12.2 *An analytical function with initial element g is called* logarithmic *and is denoted as* Log*.*

Note that $\mathrm{Log}(x)$ is not defined for $x = 0$.

If $\arg(x) \in (-\pi, \pi]$ is the principal argument of $x \neq 0$ and $|x|$ is the module of x then all values of an logarithmic function are given by

$$\mathrm{Log}(x) = \ln(|x|) + \imath(\arg(x) + 2k\pi), \ k \in \mathbb{Z}.$$

Choosing $k = 0$ we obtain the *principal* logarithm

$$\log(x) = \log(|x|) + \imath \arg(x)$$

of the complex number $x \neq 0$.

Consider now the matrix (natural) logarithm function $Y = \log(X)$, where the argument $X \in \mathbb{C}^{n \times n}$ is non–singular and the matrix $Y \in \mathbb{C}^{n \times n}$ is the solution to the exponential equation

$$\mathrm{e}^Y = X. \tag{12.21}$$

If the matrix X is real and has negative eigenvalues, then $\log(X)$ is a complex matrix. We stress that singular matrices do not have logarithms. Indeed, the matrix e^Y which should be equal to X is non–singular in view of the relation $\det(\mathrm{e}^X) = \mathrm{e}^{\mathrm{tr}(X)} \neq 0$, see (12.6).

The solution Y of equation (12.21) is not unique similarly to the non–uniqueness of logarithmic functions. To obtain a particular solution we shall require that the eigenvalues μ_k of Y are the principal logarithms of the eigenvalues λ_k of the non–singular matrix X, i.e.

$$\mu_k = \log(|\lambda_k|) + \imath \arg(\lambda_k), \ k = 1, 2, \ldots, n.$$

When $\|X - I_n\| < 1$ we have

$$\log(X) = \sum_{k=0}^{\infty}(-1)^{k+1}\frac{(X - I_n)^k}{k}.$$

The following properties of the matrix logarithm may be useful. When X is positive definite and is either real symmetric or complex Hermitian matrix then $\log(X)$ is real symmetric or complex Hermitian, respectively. If the matrix X is complex and/or has negative eigenvalues, then $\log(X)$ is a complex matrix.

Closed form expressions for $\log(X)$ may be found using the Jordan form

$$J = V^{-1}XV = \mathrm{diag}(J_1, J_2, \ldots, J_r)$$

of X. We have

$$\log(X) = V\log(J)V^{-1} = V\mathrm{diag}(\log(J_1), \log(J_2), \ldots, \log(J_r))V^{-1},$$

Next, if $J_k = \lambda$ is a scalar then $\log(J_k) = \log(\lambda)$ is the standard principal logarithm of $\lambda \neq 0$. If $J_k = \lambda I_m + N_m \in \mathbb{C}^{m\times m}$, $m > 1$, is a Jordan block with $\lambda \neq 0$ then we have

$$\begin{aligned}\log(J_k) &= \log(\lambda)I_m + \sum_{l=0}^{m-1}(-1)^{l-1}\frac{N_m^l}{l\lambda^l} \\ &= \log(\lambda)I_m + \frac{N_m}{\lambda} - \frac{N_m^2}{2\lambda^2} + \frac{N_m^3}{3\lambda^3} + \cdots + (-1)^{m-1}\frac{N_m^{m-1}}{(m-1)\lambda^{m-1}}.\end{aligned}$$

When the non–singular matrix X is normal ($X^{\mathrm{H}}X = XX^{\mathrm{H}}$) then its Schur form is diagonal (and coincides with the Jordan form) and hence the matrix V may be chosen as unitary. Here the computation of $\log(X)$ in machine arithmetic is very accurate.

The function

```
>> Y = logm(X)
```

computes the natural logarithm $Y = \log(X)$ of the invertible matrix $X \in \mathbb{C}^{n\times n}$ (or, equivalently, the inverse of the exponential), which satisfies the exponential equation $\mathrm{e}^Y = X$ in Y.

A warning message is issued when the computed `expm(Y)` is not close to X.

The command

```
>> [Y,est] = logm(X)
```

computes the matrix logarithm $Y = \log(X)$ together with an estimate of the relative residual

$$\mathrm{est} = \frac{\|\mathrm{e}^Y - X\|}{\|X\|}.$$

This command does not give a warning message.

The principal square root Y of the matrix X is computed by the command

```
>> Y = sqrtm(X)
```

The matrix $Y = \sqrt{X}$ is a solution to the matrix equation $Y^2 = X$ in Y. This equation either has no solution or has families of solutions.

When the solution Y exists its eigenvalues μ_k are the square roots of the eigenvalues λ_k of X. In particular if X is diagonalizable then

$$Y = V\mathrm{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_n})V^{-1}.$$

To choose a particular solution Y it is supposed that the eigenvalues of Y have non–negative real parts. This solution is uniquely determined when equation $Y^2 = X$ is solvable.

If X has negative real or complex eigenvalues, the matrix Y will be complex. If the matrix X is real symmetric (or complex Hermitian) and non–negative definite, i.e. $\lambda_k \geq 0$, then Y will also be a symmetric (or complex Hermitian) and non–negative definite matrix.

Some singular matrices may have no square (or any other) root, see the next example and Exercise 12.4.

Example 12.5 The matrix $X = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ has no square roots and the command `sqrtm(X)` in this case gives

```
>> ans =
         NaN          Inf
         NaN          NaN
```

There is also a warning that the matrix is singular and may not have a square root.

The command

```
>> [Y,est] = sqrtm(X)
```

computes $Y = \sqrt{X}$ together with the relative residual

$$\text{est} = \frac{\|Y^2 - X\|_{\text{F}}}{\|X\|_{\text{F}}}$$

in the Frobenius norm. Thus est is found by the command

```
>> est = norm(Y^2-X,'fro')/norm(X,'fro')}.
```

The more sophisticated command

```
>> [Y,a,est] = sqrtm(X)
```

returns the square root $Y = \sqrt{X} \in \mathbb{C}^{n\times n}$, the stability factor a and the estimate est of the matrix square root condition number of Y. The residual

$$\frac{\|Y^2 - X\|_{\text{F}}}{\|X\|_{\text{F}}}$$

is bounded approximately by aneps, while the relative error

$$\frac{\|\widehat{Y} - Y\|_{\text{F}}}{\|Y\|_{\text{F}}}$$

in the computed square root $\widehat{Y}$ is bounded approximately by anesteps. Here eps $= 2.22 \times 10^{-16}$ is about twice the rounding unit $\mathbf{u}$.

Consider now general real or complex powers of a complex matrix, namely

$$Y = X^a, \ X \in \mathbb{C}^{n\times n},$$

where $a \in \mathbb{C}$. We shall suppose that either X is non–singular, or that it has zero eigenvalues but then a is real and nonnegative. If in addition the matrix power X^a exists then the eigenvalues μ_k of Y must satisfy $\mu_k = \lambda_k^a$, where the scalar powers λ_k^a are defined according to Definitions 11.2 and 11.3.

Under the above restrictions on X and a the (unique) matrix power X^a is computed in MATLAB simply by the command

```
>> Y = X^a
```

We would like to point out the difference of this command with the dot command

```
>> X.^a
```

which is element–wise.

Matrix trigonometric and hyperbolic functions, as well as other elementary or special functions (including functions defined by the user) of the matrix X are computed by the command

```
>> Y = funm(X,'fun')
```

where `fun` is the name of the function (build–in or defined by the user). For example, the matrix sine $\sin(X)$ may be computed by the command

```
>> funm(X,'sin')
```

We stress that for matrix exponentials, logarithms and square roots MATLAB advocates the use the functions `expm`, `logm` and `sqrtm` instead of `funm(X,'exp')` , `funm(X,'log')` and $\mathrm{A}^{0.5}$.

The use of the function `funm` in MATLAB needs some additional comments. First, this function must be defined on the spectrum of X, i.e. `funm(lambda,'fun')` must be correctly defined for any element `lambda` from the vector `eig(X)`. Second, this function usually uses the eigenstructure of the matrix argument. Hence if the matrix X is not diagonalizable (i.e. its Jordan form is not diagonal) then the result of the computations may not be very accurate. Similar difficulties arise when X is diagonalizable but is close to a matrix with multiple eigenvalues and purely conditioned eigenvectors. In all these cases a warning message is printed. Sometimes such a message may be issued even if the computed result is relatively accurate.

The matrix `Y = funm(X,'fun')` is in fact computed from

$$Y = \mathrm{fun}(X) = V\mathrm{diag}(\mathrm{fun}(\lambda_1), \mathrm{fun}(\lambda_2), \ldots, \mathrm{fun}(\lambda_n))V^{-1},$$

where

$$X = V\mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)V^{-1}.$$

Otherwise speaking, for `[L,V] = eig(A)` we have `Y = V*fun(L)/V`.

The command `funm(X,'fun')` works especially well when the Schur form of X is diagonal, or equivalently, when the matrix X is normal in the sense that $X^{\mathrm{H}}X = XX^{\mathrm{H}}$. In this case the matrix V may be chosen unitary.

The command `funm(X,'fun')` may also work well when X is not normal but is diagonalizable and the eigenvector matrix V of X is well conditioned.

The results from the implementation of `funm(X,'fun')` may not be satisfactory when either X is diagonalizable with ill conditioned eigenvector matrix or when X is not diagonalizable.

We stress that in all algorithms for computing matrix functions usually the matrix argument X is scaled in order to reduce the rounding errors. The most simple way is to work with a matrix $Z = 2^{-p}X$, where $p \in \mathbb{N}$ is chosen so that

$\|Z\| < 1$, which gives $p > \log_2(\|X\|)$. In particular the matrix exponential may be computed as $e^X = (e^Z)^q$, where $q = 2^p$.

Consider now a second way to define a matrix function of a general matrix X by its Jordan form of X. Let

$$X = V\mathrm{diag}(J_1, J_2, \ldots, J_p)V^{-1}.$$

be the Jordan decomposition of X, where V is the matrix of eigenvectors and principal vectors of X. If f is analytic function defined on the spectrum of X we may define the genuine matrix function $X \mapsto f(X)$ by

$$f(X) = V\mathrm{diag}(f(J_1), f(J_2), \ldots, (J_p))V^{-1}.$$

Hence we must have an expression for the matrix $f(J)$, where J is a Jordan block. If J is a scalar then the computation of $f(J)$ is straightforward. When J is $m \times m$ with $M > 1$ we have the following result.

Theorem 12.1 *Let* $J = \lambda I_m + N_m$, $m > 1$, *be a genuine Jordan block, where* $N_m = \begin{bmatrix} 0 & I_{m-1} \\ 0 & 0 \end{bmatrix}$. *Then we have*

$$f(J) = \begin{bmatrix} f_0(\lambda) & f_1(\lambda) & \cdots & f_{m-1}(\lambda) \\ 0 & f_0(\lambda) & \cdots & f_{m-2}(\lambda) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f_0(\lambda) \end{bmatrix},$$

where

$$f_k(\lambda) := \frac{f^{(k)}(\lambda)}{k!}.$$

This way to compute the values $Y = f(X)$ of the matrix function f relies heavily on the possibility to obtain the Jordan form of a general matrix which may be a very difficult numerical problem.

In contrast to the Jordan form $J = X^{-1}AX$ of the $n \times n$ matrix A the Schur form $T = U^{\mathrm{H}}AU$ may be computed in a numerically reliable way using e.g. the QR algorithm. We recall that the columns of U form an unitary basis for $\mathbb{C}^n$, while $T = [t_{k,l}]$ is an upper triangular matrix with the eigenvalues λ_k of A on its main diagonal, i.e. $t_{k,k} = \lambda_k$. Hence we may define $f(A)$ from

$$f(A) = Uf(T)U^{\mathrm{H}}.$$

Using this representation, a third approach to compute $f(A)$ is based on the following theorem[10].

Theorem 12.2 *The elements $f_{k,l} := f(T)(k,l)$ of the matrix $F := f(T)$ are given by $f_{k,l} = 0$ if $k > l$, $f_{k,k} = f(\lambda_k)$ and*

$$f_{k,l} = \sum_{s \in S_{k,l}} T(s_0, s_1)T(s_1, s_2) \cdots T(s_{p-1}, s_p) f[\lambda_{s_0}, \lambda_{s_1}, \ldots, \lambda_{s_p}],$$

for $k < l$, where $S_{k,l}$ is the set of all strictly increasing sequences $s = (s_0, s_1, \ldots, s_p)$ of integers with $s_0 = k$ and $s_p = l$ and $f[\mu_0, \mu_1, \ldots, \mu_p]$ is the p–th order divided difference of f at the points $\mu_0, \mu_1, \ldots, \mu_p$.

The computation of $F = f(T)$ via Theorem 12.2 requires $O(2^m)$ flops. Fortunately, there is another way to compute the upper triangular part of F when the eigenvalues of A are pair–wise disjoint. It is based on the identity $FT - TF = 0$, or

$$\sum_{s=k}^{l-1} t_{k,s} f_{s,l} - t_{s,l} f_{k,s} = 0,$$

and requires about $2m^3/3$ flops.

Theorem 12.3 *Let $t_{k,k} \neq t_{l,l}$. Then*

$$f_{k,l} = \frac{t_{k,l}(f_{l,l} - f_{k,k}) + \sum_{s=k+1}^{l-1} t_{k,s} f_{s,l} - t_{s,l} f_{k,s}}{t_{l,l} - t_{k,k}}, \ k < l.$$

Using Theorem 12.3 the upper triangular matrix F may be computed one super–diagonal at a time starting with the diagonal $f(t_{1,1}), f(t_{2,2}), \ldots, f(t_{n,n})$. Then comes $f(t_{1,2}), f(t_{2,3}), \ldots, f(t_{n-1,n})$, etc.

Suppose now that the matrix A has only $m < n$ pair–wise disjoint eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ with algebraic multiplicities $n_1, n_2, \ldots, n_m$, respectively. Then Theorem 12.3 cannot be directly applied. In this case the Schur form T of A and the matrix $F = f(T)$ may be rearranged so as

$$T = \begin{bmatrix} T_{1,1} & T_{1,2} & \ldots & T_{1,m} \\ 0 & T_{2,2} & \ldots & T_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & T_{m,m} \end{bmatrix}, \ F = \begin{bmatrix} F_{1,1} & F_{1,2} & \ldots & F_{1,m} \\ 0 & F_{2,2} & \ldots & F_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & F_{m,m} \end{bmatrix},$$

where $\mathrm{spect}(T_{k,k}) = \{\lambda_k\}$ and $\mathrm{spect}(T_{k,k} \cap \mathrm{spect}(T_{l,l} = \emptyset$ for $k \neq l$. In this case we have $F_{k,k} f(T_{k,k})$ and

$$F_{k,l}T_{l,l} - T_{k,k}F_{k,l} = T_{k,l}F_{l,l} - F_{k,k}T_{k,l} + \sum_{s=k+1}^{l-1} (T_{k,s}F_{s,l} - F_{k,s}T_{s,l}) \qquad (12.22)$$

for $k = 1, 2, \ldots, m-1$ and $l = k+1, k+2, \ldots, m$.

Relations 12.22 constitute a system of $m(m-1)/2$ Sylvester matrix equation in the unknown $n_k \times n_l$ matrices $F_{k,l}$. Each Sylvester equation has an unique solution since the linear matrix operator $\mathcal{L}_{k,l}$ defined from $\mathcal{L}_{k,l}(\Phi) = \Phi T_{l,l} - T_{k,k}\Phi$ is invertible. In fact, this operator has a single eigenvalue $\lambda_l - \lambda_k \neq 0$ of multiplicity $n_k n_l$ (prove!). Thus the blocks $F_{k,l}$ may be computed one block super–diagonal at a time starting with $f(T_{1,1}), f(T_{2,2}), \ldots, f(T_{m,m})$. Then comes $f(T_{1,2}), f(T_{2,3}), \ldots, f(T_{m-1,m})$, etc.

Having an analytical scalar function f there is a fourth way to define its matrix counterpart $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ via the contour integral

$$f(X) = \int_C f(z)(zI_n - X)^{-1}\mathrm{d}z.$$

Here $C \subset \mathbb{C}$ is a smooth contour encircling the spectrum of X (note that f in the integrand is a scalar function). This definition of the matrix version of a scalar function may not be very suitable for computations but it is very useful in theoretical considerations.

12.4 Problems and exercises

Exercise 12.1 Consider a new generalized block–wise dot operation for matrices as follows. Let A be an $mp \times nq$ block matrix with mn blocks $A_{k,l}$, $k = 1, 2, \ldots, m$, $l = 1, 2, \ldots, n$, where $m, n, p, q \in \mathbb{N}$ and at least one of the numbers p or q is larger than 1. Let B be an $p \times q$ matrix and denote by $\bullet$ one of the operations

```
%
+, -, *, /, \, ^%
```

Then

```
%
C = A.\bullet B%
```

is an $mp \times nq$ block matrix with mn blocks $C_{k,l} = A_{k,l}. \bullet B$. Write a program in MATLAB environment for performing the operation $.\bullet$.

Exercise 12.2 (i) Generate a sequence of random square matrices X and compute the matrix exponential e^X by the three commands `expm(X)`, `expm2(X)` and `expm3(X)`. Compare the results.

Use the above commands to compute the exponential e^{tX} for $t = 1, 2, \ldots, 10$, where

$$X = \begin{bmatrix} -2 & 1 & 0 \\ 0 & -2 & 1 \\ 0 & 0 & -2 \end{bmatrix} = J_3(-2) = -2I_3 + N_3.$$

Comment the results having in mind the exponential

$$\begin{aligned} \mathrm{e}^{tJ_m(\lambda)} &= \mathrm{e}^{\lambda t} \begin{bmatrix} 1 & t & t^2/2! & \ldots & t^{m-1}/(m-1)! \\ 0 & 1 & t & \ldots & t^{m-2}/(m-2)! \\ \vdots & \vdots & \vdots & \ddots & t \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \mathrm{e}^{\lambda t} \left(I_m + tN_m + \frac{(tN_m)^2}{2!} + \cdots + \frac{(tN_m)^{m-1}}{(m-1)!} \right) \end{aligned}$$

for the Jordan block $J_m(\lambda) = \lambda I_m + N_m$.

(ii) Find the matrix logarithms of the computed exponentials. Comment the results.

Exercise 12.3 Explain why a singular matrix does not have a logarithm. Use the fact that

$$\det(\exp(X)) = \exp(\operatorname{tr}(X)),$$

where $\operatorname{tr}(X)$ is the trace of the matrix X (the sum of its diagonal elements).

Exercise 12.4 Show that the matrix $X = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ has no square root XY, i.e. that the matrix equation $Y^2 = X$ has no roots, real or complex.

Exercise 12.5 Find all square roots of the identity matrix I_2 solving the matrix polynomial equation $X^2 - I_2 = 0$.

Exercise 12.6 (i) Show that the matrix functions $C, S : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$, defined in (12.13), satisfy the matrix differential homogeneous equation

$$Z''(t) + AZ(t) = 0,\ Z(t) \in \mathbb{C}^{n\times n}, \qquad (12.23)$$

with initial conditions

$$C(0) = I_n,\ C'(0) = 0,\ S(0) = 0,\ S'(0) = I_n.$$

(ii) Prove that the solution of equation (12.23) with initial conditions $Z(0) = Z_0$, $Z'(0) = Z'_0$, where Z_0, Z'_0 are given matrices, may be represented as

$$Z(t) = C(t)Z_0 + S(t)Z'_0.$$

(iii) Prove the following assertions. If the matrix A has a square root $A^{1/2}$ then the function C may be represented as $C(t) = \cos(tA^{1/2})$. If in addition the matrix A is non–singular then the function S may be represented as $A^{-1/2}\sin(tA^{1/2})$. In both cases $A^{1/2}$ may be taken as *any* of the square roots of A.

Exercise 12.7 Let the scalar analytical function f be defined on the spectrum of the matrix A and set $F = f(A)$. Prove the following assertions. (i) $AF = FA$. (ii) If A is upper (lower) triangular then F is upper (lower) triangular but the inverse may not be true. (iii) If A is Hermitian ($A = A^{\mathrm{H}}$) then F is Hermitian but the inverse may not be true. (iv) If A is normal ($A^{\mathrm{H}}A = AA^{\mathrm{H}}$) then F is normal but the inverse may not be true. Hint: to show that the inverse statements in (ii), (iii) and (iv) may not be true consider a scalar function f which is identically equal to a real constant.

Chapter 13

General aspects of curve fitting

13.1 Statement of the problem

Suppose that we have a set of experimental data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m). \tag{13.1}$$

We shall assume first that $(x_k, y_k) \in \mathbb{R}^2$ and $x_1 < x_2 < \cdots < x_m$. If we denote by X and Y the m–vectors with elements x_k and y_k, respectively, the data will briefly be denoted by the vector pair (X, Y).

The *curve fitting problem* is to construct a function

$$x \mapsto F(p, x) \in \mathbb{R},$$

approximating the data (13.1) in a certain sense, where

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{n+1} \end{bmatrix} \in \mathbb{R}^{n+1}$$

is a vector parameter of length $n + 1$ (we denote the number of unknown parameters p_l by $n + 1$ instead of n for reasons of tradition). In particular, $F(p, x)$ may be a polynomial in x of degree n with $n + 1$ unknown coefficients $p_1, p_2, \ldots, p_{n+1}$,

$$F(p, x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1} = p^\top \varphi(x),$$

where

$$\varphi(x) := \begin{bmatrix} x^n \\ x^{n-1} \\ \vdots \\ x \\ 1 \end{bmatrix}.$$

The function F is defined and differentiable in all its arguments in the domain $P \times X \to \mathbb{R}$, where $P \subset \mathbb{R}^{n+1}$ and $X \subset \mathbb{R}$ is an interval, containing the interval $[x_1, x_m]$.

The function F and the vector–parameter p have to be chosen so as the curve

$$\Gamma(p) := \{(x, F(p,x)) : x \in [x_1, x_m]\}$$

to approximate the data (13.1). For this purpose we may define the *residual vector*

$$R(p) := \begin{bmatrix} F(p,x_1) - y_1 \\ F(p,x_2) - y_2 \\ \vdots \\ F(p,x_m) - y_m \end{bmatrix} \in \mathbb{R}^m$$

and then minimize some of its norms. The approximation problem will be denoted as

$$F(p, x_k) \simeq y_k, \ k = 1, 2, \ldots, m. \qquad (13.2)$$

Typically, $F(p,x)$ is linear in p, see for example relation (13.4) below. In this case we shall say that the minimization problem (13.2) is *linear* in p.

The approximation of the data by F may now be formulated as the problem of minimizing some norm of $R(p)$ over all admissible values of p, e.g.

$$r(p) := \|R(p)\|_2 = \left(\sum_{k=1}^{m} (F(p,x_k) - y_k)^2\right)^{1/2} = \min! \qquad (13.3)$$
$$p \in P.$$

We stress that usually $n \le m-1$. If $n = m-1$ we may, in principle, solve the *interpolation problem*

$$R(p) = 0.$$

Since $r(p) \ge 0$ we may define

$$r^0 := \inf\{r(p) : p \in P\}.$$

Of course, it is possible that there is no p^0 such that $r(p^0) = r^0$, i.e. the infimum of r may not be reachable. For most interesting problems, however, there exists $p^0 \in P$ such that

$$r(p^0) = r^0 := \min\{r(p) : p \in P\}.$$

Further on we shall consider this case.

Note that usually $n < m - 1$ and then we must find the value p^0 of p which minimizes $r(p) > 0$.

It is possible that in (13.1) some of the quantities x_k or y_k is a vector, e.g. $x_k \in \mathbb{R}^p$, $y_l \in \mathbb{R}^q$. In this case $F : \mathbb{R}^{n+1} \times \mathbb{R}^p \to \mathbb{R}^q$, the residual vector is

$$R(p) = \begin{bmatrix} F(p, x_1) - y_1 \\ F(p, x_2) - y_2 \\ \vdots \\ F(p, x_m) - y_m \end{bmatrix} \in \mathbb{R}^{mq}$$

and the function r may again be defined as $r(p) = \|R(p)\|_2$.

Of course, instead of the 2–norm of $R(p)$, one may try to minimize some other norm, e.g.

$$r_1(p) := \|R(p)\|_1 = \sum_{k=1}^{m} |F(p, x_k) - y_k| = \min!,$$

or

$$r_\infty(p) := \|R(p)\|_\infty = \max\{|F(p, x_k) - y_k| : k = 1, 2, \ldots, m\}.$$

In the last two cases we have the so called L_1 and L_∞ approximation problem, respectively.

The data (13.1) can be regarded as generated by an unknown function

$$f : [x_1, x_m] \to \mathbb{R}$$

in the sense that

$$y_k = f(x_k),\ \ k = 1, 2, \ldots, m.$$

This assumption is essential when trying to find estimates for the accuracy of the curve fitting problem. In this case the estimates will depend on the properties of f.

When the problem is to approximate a given function f as above, one may solve also another problem, namely to determine the m knots $x_1, x_2, \ldots, x_m$ so that to achieve a better approximation.

13.2 The Weierstrass approximation theorem

The theoretical base for the approximation of continuous functions by algebraic polynomials

$$P(x) = p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1}$$

is the following theorem, due to Weierstrass.

Theorem 13.1 *Let*

$$f : [a,b] \to \mathbb{R}$$

be a continuous function. Then for each $\varepsilon > 0$ *there exists a polynomial* P *of degree* $n = n(\varepsilon)$ *such that*

$$\|f - P\|_\infty := \max\{|f(x) - P(x)| : x \in [a,b]\} < \varepsilon.$$

A similar theorem is valid for the approximation of continuous functions

$$f : [-\pi, \pi] \to \mathbb{R}$$

by trigonometric polynomials

$$T(x) = \sum_{k=0}^{n} (a_k \cos kx + b_k \sin kx).$$

13.3 Analytical solution

In some cases, e.g. when the expression $F(p,x)$ is linear or affine in p, the solution may be determined analytically from the conditions

$$\frac{\partial r^2(p)}{\partial p_l} = 0, \ l = 1, 2, \ldots, n+1.$$

Having in mind the expression (13.3) for $r(p)$ we obtain

$$\sum_{k=1}^{m} (F(p,x_k) - y_k) \frac{\partial F(p,x_i)}{\partial p_l} = 0, \ l = 1, 2, \ldots, n+1.$$

An important particular case is when $F(p,x)$ is linear in p,

$$F(p,x) = p^\top \varphi(x) = \sum_{l=1}^{n+1} p_l \varphi_l(x), \ \varphi(x) := [\varphi_1(x), \varphi_2(c), \ldots, \varphi_{n+1}(x)]^\top \in \mathbb{R}^{n+1}. \tag{13.4}$$

Often the expression for $F(p,x)$ is chosen from physical considerations.

Example 13.1 Suppose that we have a process described by the differential equation

$$\frac{\mathrm{d}y(x)}{\mathrm{d}x} = \lambda y(x),$$

where λ is an unknown parameter. Then we may look for an expression for y in the form

$$y(x) = F(p, x) := p_1 \exp(p_2 x)$$

and then determine λ from $\lambda = p_2$. Suppose that $y_k > 0$, $k = 1, 2, \ldots, m$. Since the dependence of F on p is nonlinear, it is appropriate to take logarithms from both $F(p, x_k)$ and y_k. Then we get the approximation problem

$$\ln p_1 + p_2 x_k \simeq \ln y_k, \ \ k = 1, 2, \ldots, m,$$

which is already linear in $\ln p_1$ and p_2.

For the model (13.4) we have

$$\sum_{k=1}^{m} (F(p, x_k) - y_k)\varphi_l(x_k) = 0, \ \ l = 1, 2, \ldots, n+1$$

and hence

$$Bp = c. \tag{13.5}$$

Here the elements $b_{l,k}$ and c_l of the matrix

$$B := \sum_{s=1}^{m} \varphi(x_s)\varphi^{\top}(x_s) \in \mathbb{R}^{(n+1)\times(n+1)}$$

and the vector

$$c := \sum_{s=1}^{m} y_s \varphi(x_s) \in \mathbb{R}^{n+1},$$

respectively, are given by

$$\begin{aligned} b_{l,k} &:= \sum_{s=1}^{m} \varphi_l(x_s)\varphi_k(x_s), \\ c_l &:= \sum_{s=1}^{m} y_s \varphi_l(x_s). \end{aligned}$$

Note, however, that *the determination of the vector p by the linear algebraic equation (13.5) may not be a good idea in machine arithmetic even for moderate values of m.* This assertion may seem strange since the matrix B is symmetric and nonnegative definite (and, typically, even positive definite). Hence the solution of equation (13.5) itself should not be a problem.

The real problem is that the elements $b_{l,k}$ of the matrix B may be computed with large errors which may deteriorate the accuracy of the computed value for p.

A better way to compute p is via the least squares problem

$$Ap \simeq y, \tag{13.6}$$

where

$$A := \begin{bmatrix} \varphi^\top(x_1) \\ \varphi^\top(x_2) \\ \vdots \\ \varphi^\top(x_m) \end{bmatrix} = \begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_{n+1}(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \dots & \varphi_{n+1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_m) & \varphi_2(x_m) & \dots & \varphi_{n+1}(x_m) \end{bmatrix} \in \mathbb{R}^{m\times(n+1)} \tag{13.7}$$

and

$$y := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m.$$

Note that the connection of the two approaches, based on (13.5) and (13.6), respectively, is given by

$$B = A^\top A, \ \ c = A^\top y.$$

13.4 Problems and exercises

Exercise 13.1 Let the numbers $0 < \lambda_1 < \lambda_2 < \cdots < \lambda_n$ be given and suppose that the function F is of the form

$$F(p,x) := p_1 \exp(\lambda_1 x) + p_2 \exp(\lambda_2 x) + \cdots + p_n \exp(\lambda_n x) + p_{n+1}.$$

Write a program in MATLAB for approximation of the data (13.1) with $m > n+1$ by the function F.

Exercise 13.2 Approximate the data as in Exercise 13.1 using the functions

$$F_c(p,x) := p_1 \cos(\lambda_1 x) + p_2 \cos(\lambda_2 x) + \cdots + p_n \cos(\lambda_n x) + p_{n+1}$$

and

$$F_s(p,x) := p_1 \sin(\lambda_1 x) + p_2 \sin(\lambda_2 x) + \cdots + p_n \sin(\lambda_n x) + p_{n+1}.$$

Chapter 14

Interpolation

14.1 Statement of the problem

Suppose that we want to interpolate the data (13.1). In this case we should choose $n = m - 1$ and determine the parameter vector p from the system of equations

$$F(p, x_k) = y_k,\ k = 1, 2, \dots, m. \tag{14.1}$$

In general the system of equations (14.1) is nonlinear. If, however, the function $F(\cdot, x)$ is linear as in (13.4), we get the linear equation

$$Ap = y,$$

where the matrix $A \in \mathbb{R}^{m\times m}$ is determined by (13.7) for $n = m - 1$. In this case it is important to establish conditions which guarantee that the matrix A is nonsingular. For this purpose the notion of a Chebyshov system is useful.

First we shall recall the concept of linear independence of a system of functions.

Let $M \subset \mathbb{R}$ be a given nontrivial interval and $\{\varphi_1, \varphi_2, \dots, \varphi_m\}$ be a system of functions $\varphi_k : M \to \mathbb{R}$. For any $p = [p_1, p_2, \dots, p_m]^\top \in \mathbb{R}^m$ denote by $L(p, \cdot)$,

$$L(p, x) := p^\top \varphi(x) = \sum_{k=1}^{m} p_k \varphi_k(x), \tag{14.2}$$

the *linear combination* of the functions φ_k with coefficients p_k.

Definition 14.1 The system of functions $\{\varphi_1, \varphi_2, \dots, \varphi_m\}$, where $\varphi_k : M \to \mathbb{R}$, is called *linearly independent* on M if the condition

$$L(p, x) = 0,\ x \in M,$$

implies $p = 0$. Otherwise the system of functions is called *linearly dependent.*

Example 14.1 A system of one function φ_1 is linearly independent on M exactly when $\varphi_1(x) \neq 0$ for some $x \in M$.

A system of $m \geq 2$ functions is linearly dependent on M if and only if one of the functions, say φ_1, may be represented as a linear combination,

$$\varphi_1(x) = \sum_{k=2}^{m} c_k \varphi_k(x), \ \ x \in M,$$

of the other functions $\varphi_2, \varphi_3, \ldots, \varphi_m$.

Thus the concept of linear independence of a system of functions is similar to this of a system of vectors in $\mathbb{R}^m$.

Definition 14.2 The system of functions $\{\varphi_1, \varphi_2, \ldots, \varphi_m\}$, where $\varphi_k : M \to \mathbb{R}$, is said to be a *Chebyshov*[1] *system* on M if for every $p \in \mathbb{R}^m$ the function $L(p, \cdot)$, determined by (14.2), has at most $m - 1$ zeros in M.

Note that a Chebyshov system may be defined similarly on any subset M of $\mathbb{R}$.

Example 14.2 The system of m functions $x \mapsto x^k$, $k = 0, 1, \ldots, m - 1$, is a Chebyshov system on any interval $[a, b]$. Indeed, it is known that any polynomial $p_1 x^{m-1} + p_2 x^{m-2} + \cdots + p_{m-1} x + p_m$ of degree $m - 1$ has no more than $m - 1$ zeros in $\mathbb{R}$ and in particular in any subset $[a, b]$ of $\mathbb{R}$.

A system of linearly independent functions may not be a Chebyshov system on a given set M as shown in the next example.

Example 14.3 The system $\{1, \sin\}$ is linearly independent on the interval $[0, \pi]$ but it is not a Chebyshov system since the linear combination $x \mapsto 2\sin x - 1$ of 1 and sin has two zeros $x_1 = \pi/6$ and $x_2 = 5\pi/6$ in $[0, \pi]$.

An important result for Chebyshov systems $\Phi := \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ on $[a, b]$ is as follows. Let

$$a \leq x_1 < x_2 < \cdots < x_m \leq b$$

Consider the matrix $A = [a_{k,l}] \in \mathbb{R}^{m \times m}$ determined by (13.7) for $n = m - 1$, i.e. $a_{k,l} = \varphi_l(x_k)$.

[1]P. Chebyshov, Russian mathematician, 1821-1894.

Theorem 14.1 *The system Φ is a Chebyshov system on $[a,b]$ if and only if the matrix A is non–singular.*

Proof. Let Φ be a Chebyshov system on $[a,b]$ and suppose that the matrix A is singular. Then there is a nonzero solution $p \in \mathbb{R}^m$ of the equation $Ap = 0$, i.e.

$$Ap = \begin{bmatrix} \varphi^\top(x_1) \\ \varphi^\top(x_2) \\ \vdots \\ \varphi^\top(x_m) \end{bmatrix} p = \begin{bmatrix} \varphi^\top(x_1)p \\ \varphi^\top(x_2)p \\ \vdots \\ \varphi^\top(x_m)p \end{bmatrix} = 0$$

and

$$L(p,x_k) = \varphi^\top(x_k)p = p^\top\varphi(x_k) = \sum_{s=1}^m p_s\varphi_s(x_k) = 0,\ k = 1,2,\ldots,m.$$

Hence there is a linear combination $L(p,\cdot) = p^\top\varphi(\cdot)$ of the functions φ_s which has m zeros in $[a,b]$. But this is impossible since the Chebyshov property guarantees that the function $L(p,\cdot)$ may have at most $m-1$ zeros in $[a,b]$. This contradiction shows that the matrix A is non–singular.

Suppose now that the matrix A is non–singular for any choice of the points $x_1 < x_2 < \cdots < x_m$. If Φ is not a Chebyshov system then there is a linear combination $L(p,\cdot) = p^\top\varphi(\cdot)$, $p \neq 0$, such that $L(p,x_k) = 0$ for $k = 1,2,\ldots,m$. But $L(p,x_k)$ is the k–th element of the vector Ap. Hence $Ap = 0$, which, together with $p \neq 0$, is a contradiction to the assumption that A is non–singular. This contradiction shows that Φ must be a Chebyshov system and the proof is complete. □

14.2 Interpolation by algebraic polynomials

In this section we shall consider the interpolation of a set of data $(x_k,y_k) \in \mathbb{R}^2$ by algebraic polynomials. Following the tradition, we shall denote the data as

$$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n),$$

where $a = x_0 < x_1 < \cdots < x_n = b$ and $n \geq 1$.

The system of functions $x \mapsto x^k$, $k = 0,1,\ldots,n$, is a Chebyshov system on any interval $[a,b]$. Indeed, any algebraic polynomial of degree n has at most n zeros in $\mathbb{C}$ and hence at most n zeros on every subset of $\mathbb{R}$ and on every interval $[a,b] \in \mathbb{R}$ in particular.

It is possible to compute the vector parameter

$$p := [p_0, p_1, \ldots, p_n]^\top \in \mathbb{R}^{n+1},$$

whose elements are the coefficients of the interpolating polynomial

$$L(p, x) := p_0 + p_1 x + \cdots + p_n x^n, \tag{14.3}$$

by using directly the interpolating conditions

$$L(p, x_k) = y_k,\ k = 0, 1, \ldots, n$$

(note that in (14.3) we use the traditional notation for polynomials and not the notation from MATLAB). This gives the linear equation $Ap = y$, where the elements of the matrix $A \in \mathbb{R}^{(n+1)\times(n+1)}$ are given by

$$a_{k,l} = x_k^l, \quad l, k = 0, 1, \ldots, n,$$

and $y = [y_0, y_1, \ldots, y_n]^\top \in \mathbb{R}^{n+1}$. Indeed, since the system of functions $\{1, x, \ldots, x^n\}$ is a Chebyshov one, the matrix A is non–singular. Moreover, since A is a Van der Monde matrix, it can be shown that

$$\det(A) = \prod_{l=0}^{n} \prod_{k=l+1}^{n-1} (x_k - x_l).$$

However, this way to compute p is not recommended for relatively large values of n, say $n > 2$. Alternative ways to solve the interpolation problem are described below.

Denote by

$$\omega(x) := \prod_{k=0}^{n} (x - x_k)$$

the monic $(n+1)$–th degree polynomial with zeros $x_0, x_1, \ldots, x_n$, and set

$$\omega_l(x) := \prod_{k \neq l} (x - x_k) = \frac{\omega(x)}{(x - x_l)},\ l = 0, 1, \ldots, n.$$

Thus $\omega_l(x)$ is a monic polynomial of degree n with zeros x_k, $k \neq l$. Hence $\omega_l(x_k) = 0$ for $k \neq l$. Note that

$$\omega'(x) = \sum_{l=0}^{n} \omega_l(x)$$

and

$$\omega(x_l) = \omega_l(x_l),\ l = 0, 1, \ldots, n.$$

We also have

$$\frac{1}{\omega(x)} = \sum_{l=0}^{n} \frac{1}{\omega'(x_l)(x - x_l)}.$$

Now set

$$L_s(x) := \frac{\omega_s(x)}{\omega_s(x_s)} = \frac{\omega(x)}{\omega'(x_s)(x - x_s)},\ s = 0, 1, \ldots, n,$$

or, equivalently,

$$L_s(x) = \frac{(x - x_0)\cdots(x - x_{s-1})(x - x_{s+1})\cdots(x - x_n)}{(x_s - x_0)\cdots(x_s - x_{s-1})(x_s - x_{s+1})\cdots(x_s - x_n)}.$$

The polynomials L_s satisfy

$$L_s(x_k) = \delta_{s,k}, \quad s, k = 0, 1, \ldots, n,$$

where $\delta_{s,k}$ is the Kronecker delta function, defined by $\delta_{k,k} = 1$ and $\delta_{s,k} = 0$ for $s \neq k$.

Example 14.4 Let $n = 2$. Then $\omega(x) = (x - x_0)(x - x_1)(x - x_2)$ and

$$\begin{aligned} L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \\ L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}, \\ L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}. \end{aligned}$$

Now the interpolating polynomial may be written as

$$L(x) = \sum_{s=0}^{n} y_s L_s(x) := \lambda(x) y, \tag{14.4}$$

where

$$\lambda(x) := [L_0(x), L_1(x), \ldots, L_n(x)] \in \mathbb{R}^{1\times(n+1)},\ y := [y_0, y_1, \ldots, y_n]^\top \in \mathbb{R}^{n+1}.$$

This form of the interpolating polynomial is called the *Lagrange*[2] *interpolation polynomial.*

[2]J. Lagrange, French mathematician and astronomer, 1736-1813.

A disadvantage of the Lagrange approach is that each polynomial L_s depends on the whole set of data. Hence if we augment the data by an additional point (x_{n+1}, y_{n+1}), or remove a point from the data set, or replace a point by another point, we should recompute all polynomials L_s in (14.4).

Another way to construct the interpolation polynomial is to use the formula

$$\begin{aligned} L(x) &= c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots \\ &\quad + c_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}) =: \sum_{k=0}^{n} c_k \psi_k(x), \end{aligned} \tag{14.5}$$

where $\psi_0(x) := 1$ and

$$\psi_k(x) := (x-x_0)(x-x_1)\cdots(x-x_{k-1}) = \prod_{s=0}^{k-1}(x-x_s), \ k = 1, 2, \ldots, n-1. \tag{14.6}$$

Here the coefficients c_k are calculated in a recurrent way, setting $x = x_k$ and using the conditions $L(x_k) = y_k$, $k = 0, 1, \ldots, n-1$, namely

$$\begin{aligned} c_0 &= y_0, \\ c_1 &= \frac{y_1 - c_0}{x_1 - x_0}, \\ c_2 &= \frac{y_2 - c_0 - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}, \\ c_k &= \frac{y_k - \sum_{s=0}^{k-1} c_s \psi_s(x_k)}{\psi_k(x_k)}, \quad k = 0, 1, \ldots, n. \end{aligned} \tag{14.7}$$

The Newton[3] interpolation scheme (14.5), (14.6), (14.7) may be preferable than the use of the Lagrange polynomial when there is a necessity of updating the data or adding new data points.

14.3 Errors in interpolation by algebraic polynomials

In this section we consider the important problem of estimating the errors in interpolation by algebraic polynomials. We take into account both the error of the method and the error due to the use of machine arithmetic.

Suppose that we want to interpolate points of the graph of the function

$$f : [a, b] \to \mathbb{R}$$

[3]I. Newton, English mathematician and physicist, 1643–1727.

by algebraic polynomials $L(x)$ of degree n. Here we have

$$y_k = f(x_k),\ \ k = 0, 1, \ldots, n,$$

where $a = x_0$ and $x_n = b$.

Suppose that f has a bounded $(n+1)$–th derivative $f^{(n+1)}$ on (a, b),

$$M_{n+1}(f) := \max\{|f^{(n+1)}(x)| : x \in (a, b)\} < \infty.$$

Let the point $u \in [a, b]$ be different from $x_0, x_1, \ldots, x_n$ and set

$$C = C(u) := \frac{f(u) - L(u)}{\omega(u)}. \tag{14.8}$$

Consider the function $F : [a, b] \to \mathbb{R}$, defined by

$$F(x) := f(x) - L(x) - C(u)\omega(x).$$

It has $n + 2$ zeros $x_0, x_1, \ldots, x_n$ and u. Applying the theorem of Roll we see that F' has at least $n + 1$ zeros in (a, b). Indeed, F' has a zero in each of the $n+1$ open intervals with endpoints at x_k and u. Furthermore, F'' has at least n zeros in (a, b). Continuing these considerations by induction, we see that $F^{(n+1)}$ has at least one zero $\xi \in (a, b)$. Since $L^{(n+1)} = 0$ and $\omega^{(n+1)}(x) = (n + 1)!$, we obtain

$$F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - C(u)(n + 1)! = 0$$

and

$$C(u) = \frac{f^{(n+1)}(\xi)}{(n + 1)!}.$$

Now it follows from the definition (14.8) of $C(u)$ that

$$|f(u) - L(u)| = C(u)\omega(u) = \frac{f^{(n+1)}(\xi)}{(n + 1)!}\omega(u).$$

Hence

$$|f(u) - L(u)| \le \frac{M_{n+1}(f)}{(n + 1)!}|\omega(u)|.$$

Denoting by

$$\|f\| := \max\{|f(x)| : x \in [a, b]\}$$

the norm of the continuous function f, we obtain the following estimate for the absolute error from interpolation

$$\|f - L\| \le \frac{M_{n+1}(f)\|\omega\|}{(n + 1)!}. \tag{14.9}$$

It follows from (14.9) that the interpolation error may be large when the bound $M_{n+1}(f)$ for the $(n+1)$–th derivative of f and/or the $\|\omega\|$ norm of ω are large. Note also that $\|\omega\|$ does not depend on f but only on the points $x_0, x_1, \ldots, x_n$.

Consider finally the estimation of the global error in the interpolation. The global error is equal to the truncation error plus the error due to the rounding in machine arithmetic with rounding unit $\mathbf{u}$. Below we shall neglect small terms of order $\mathbf{u}^2$ and higher.

Denote by $\widehat{L}(x)$ the computed value of $L(x)$. The absolute error in the computation of $f(x)$ by $\widehat{L}(x)$ at the point x is

$$\begin{aligned} E(x) &:= |f(x) - \widehat{L}(x)| = |f(x) - L(x) + L(x) - \widehat{L}(x)| \\ &\leq |f(x) - L(x)| + |L(x) - \widehat{L}(x)|. \end{aligned} \tag{14.10}$$

Denoting by $\widehat{L}_k(x)$ and $\widehat{\lambda}(x)$ the computed values of $L_k(x)$ and $\lambda(x)$, respectively, we have

$$\widehat{L}(x) = \mathrm{fl}(\mathrm{fl}(\lambda(x))y).$$

Suppose (rather optimistically!) that the computation of $L_k(x)$ is done with maximum achievable accuracy, i.e.

$$\widehat{L}_k(x) = L_k(x)(1 + \varepsilon_k),\ k = 0, 1, \ldots, n,$$

and

$$\widehat{\lambda}(x) = \lambda(x)(I_{n+1} + D),\ D := \mathrm{diag}(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_n),$$

where $|\varepsilon_k| \leq \mathbf{u}$. Further on we get

$$\begin{aligned} \widehat{L}(x) &= \mathrm{fl}(\lambda(x)y + \lambda(x)Dy) = (\mathrm{fl}(\lambda(x)y) + \lambda(x)Dy)(1 + \varepsilon_{n+1}) \\ &= \mathrm{fl}(\lambda(x)y) + \lambda(x)Dy + \varepsilon_{n+1}\mathrm{fl}(\lambda(x)y), \end{aligned}$$

where $|\varepsilon_{n+1}| \leq \mathbf{u}$. We already know from the error analysis of the scalar product that

$$\mathrm{fl}(\lambda(x)y) = \lambda(x)y + \delta = L(x) + \delta,$$

where

$$|\delta| \leq \mathbf{u}n|\lambda(x)|\,|y|.$$

Hence

$$|\widehat{L}(x) - L(x)| \leq |\delta| + \mathbf{u}|\lambda(x)|\,|y| + \mathbf{u}|L(x)|.$$

Denoting

$$N(x) := |\lambda(x)|\,|y| = \sum_{k=0}^{n} |y_k|\,|L_k(x)|$$

we get

$$|\widehat{L}(x) - L(x)| \le \mathbf{u}(|L(x)| + (n+1)N(x)).$$

Since $|L(x)| \le N(x)$ we may further simplify this estimate as

$$|\widehat{L}(x) - L(x)| \le \mathbf{u}(n+2)N(x). \tag{14.11}$$

Having in mind (14.10) and (14.11) we obtain the overall absolute error estimate

$$\frac{M_{n+1}(f)\|\omega\|}{(n+1)!} + \mathbf{u}(n+2)N(x). \tag{14.12}$$

The two terms at the right–hand side of (14.12) clearly indicate the contribution of the main two factors to the error estimate: the first term reflects the properties of f and the choice of the knots x_k, while the second term estimates the effect of rounding errors.

14.4 Interpolation by trigonometric polynomials

In this section we shall consider interpolation of data by *trigonometric polynomials* of order n in the form

$$T(x,c) := \frac{a_0}{2} + \sum_{k=1}^{n}(a_k \cos kx + b_k \sin kx), \tag{14.13}$$

where $a := [a_0, a_1, \ldots, a_n]$ and $b := [b_1, b_2, \ldots, b_n]$ are unknown vectors, and $c := [a, b]$. Since the number of unknowns (the elements of a and b) is $2n+1$, it is reasonable to suppose that the data consists of the same number $2n+1$ of points

$$(x_0, y_0),\ (x_1, y_1), \ldots, (x_{2n}, y_{2n}),$$

where $0 \le x_0 < x_1 < \cdots < x_{2n} < 2\pi$.

It may be shown easily by induction in n that the function

$$x \mapsto \sin\frac{x - x_1}{2}\sin\frac{x - x_2}{2}\sin\frac{x - x_{2n}}{2}$$

is a trigonometric polynomial of n–th order for any choice of $x_1, x_2, \ldots, x_{2n}$.

The interpolation by trigonometric polynomials is based on the following result.

Theorem 14.2 *There exists a unique trigonometric n–th order polynomial T such that*

$$T(x_k, c) = y_k,\ k = 0, 1, \ldots, 2n. \tag{14.14}$$

Proof. First we shall show that the polynomial (14.13) has at most $2n$ zeros in $x \in [0, 2\pi)$. Setting $z := \exp(\imath x)$ and having in mind that

$$\cos kx = \frac{z^k + z^{-k}}{2}, \ \sin kx = \frac{z^k - z^{-k}}{2\imath},$$

we may rewrite $T(x, c)$ as

$$T(x, c) = \sum_{k=-n}^{n} d_k z^k = z^{-n} P(z),$$

where the coefficients $d_k = d_k(c)$ are defined from

$$d_0 := \frac{a_0}{2}, \ d_{-k} := \frac{a_k + \imath b_k}{2}, \ d_k := \overline{d_{-k}}, \ k = 1, 2, \ldots, n,$$

and P is an algebraic polynomial of degree $2n$. The polynomial P has $2n$ zeros (counted according to multiplicity) $z_1, z_2, \ldots, z_{2n}$. Now the equation $\exp(\imath x) = z_k$ has a unique root in the strip $0 \le \operatorname{Re} x < 2\pi$ for any $k = 1, 2, \ldots, 2n$. Hence the equation $T(x, c) = 0$ has no more than $2n$ zeros in the interval $[0, 2\pi)$.

It is now clear that the linear homogeneous system

$$T(x_k, c) = 0, \ k = 0, 1, \ldots, 2n,$$

in c has only the trivial solution $c = 0$. Hence the system (14.14) has an unique solution c. $\square$

There is a simple closed form solution for the practically important case when the points x_k are equidistant, i.e.

$$x_k = \frac{2k\pi}{2n+1}, \ k = 0, 1, \ldots, 2n.$$

Here the interpolating polynomial may be written as

$$\frac{1}{2n+1} \sum_{k=0}^{2n} y_k \frac{\sin \frac{2n+1}{2}(x - x_k)}{\sin \frac{x - x_k}{2}}$$

or in the form (14.13) with

$$\begin{aligned} a_k &= \frac{2}{2n+1} \sum_{s=0}^{2n} y_s \cos kx_s, \\ b_k &= \frac{2}{2n+1} \sum_{s=0}^{2n} y_s \sin kx_s, \ k = 0, 1, \ldots, n. \end{aligned}$$

14.5 Interpolation by fractional–polynomial functions

The interpolation by fractional–polynomial functions (FPF) is also known as *Padé interpolation.*

Consider the interpolating function Y,

$$Y(c,x) = \frac{P(c,x)}{Q(c,x)} := \frac{a_0 + a_1 x + \cdots + a_p x^p}{1 + b_1 x + \cdots + b_q x^q},$$

depending on the scalar argument x and on the vector–parameter $c = [a^\top, b^\top]^\top$, where $a := [a_0, a_1, \ldots, a_p]$ and $b := [b_1, b_2, \ldots, b_q]$. Since Y depends on $m := p + q + 1$ parameters, it is reasonable to interpolate the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m),\ x_1 < x_2 < \cdots < x_m.$$

The interpolation conditions yield the vector algebraic equation

$$Ac = y,\ y := [y_1, y_2, \ldots, y_m]^\top \in \mathbb{R}^m,$$

where the elements of the matrix $A \in \mathbb{R}^{m\times m}$ are determined by

$$A(k,l) := x_k^{l-1},\ 1 \le l \le p+1,$$

and

$$A(k,l) := -y_k x_k^{l-p-1},\ p+2 \le l \le m.$$

We stress that here the matrix A is not automatically non–singular as in the case of interpolation by algebraic polynomials.

The problem of function interpolation by FPF may also be difficult when the data is close to the data generated itself by some FPF with close zeros of the numerator and denominator. In this case the problem is *ill posed* in the sense of H'Adamard.

14.6 Hermite interpolation

Consider the problem of constructing a function F, satisfying the following set of conditions imposed on F and its derivatives

$$F^{(s)}(p, x_k) =: \frac{\partial^s F(p, x_k)}{\partial x^s} = y_k^{(s)},\ s = 0, 1, \ldots, r_k - 1,\ k = 1, 2, \ldots, m, \quad (14.15)$$

where $p \in \mathbb{R}^l$ is a vector–parameter and $r_k \ge 1$ are given integers. In this problem it is reasonable to assume

$$l = r := r_1 + r_2 + \cdots + r_m,$$

equalizing the number l of unknowns with the number r of imposed conditions. This is the so called *Hermite interpolation problem.* For $r_1 = r_2 = \cdots = r_m = 1$ we obtain the standard interpolation problem

$$F(p, x_k) = y_k := y_k^{(0)}, \ k = 1, 2, \ldots, m.$$

More generally, if $r_k = 1$ for some k there are no restrictions on the derivatives of the interpolation function at the point x_k and only the condition $F(p, x_k) = y_k$ has to be satisfied.

Example 14.5 Consider the problem of constructing a polynomial

$$H(p, x) = p_1 x^2 + p_2 x + p_3$$

of second degree satisfying the conditions

$$H(p, 0) = 0, \ H_x'(p, 0) = a, \ H(p, 1) = b. \tag{14.16}$$

Here $m = 2$, $r_1 = 2$, $r_2 = 1$, $x_1 = 0$, $x_2 = 1$, $y_1 = 0$, $y_1' = a$ and $y_2 = b$. Substituting the expression for $H(p, x)$ in (14.16) we get the system of equations for p_k

$$\begin{aligned} &p_3 = 0, \\ &p_2 = a, \\ &p_1 + p_2 + p_3 = b \end{aligned}$$

and hence

$$H(p, x) = (b - a)x^2 + ax.$$

As mentioned above, to solve the interpolation problem (14.15) we have to satisfy r conditions. Hence for this purpose we may try to use an algebraic polynomial of degree $r - 1$.

Note that the Hermite interpolation problem is of some interest even in the case $m = 1$. Here $r = r_1$ and one has to find a polynomial

$$H(p, x) = p_1 x^{r-1} + \cdots + p_{r-1} x + p_r,$$

of degree $r - 1$ satisfying the conditions

$$\frac{\partial^s H(p, x_1)}{\partial x^s} = y_1^{(s)}, \ s = 0, 1, \ldots, r - 1.$$

The solution of this problem is given by the Taylor's formula

$$\sum_{s=0}^{r-1} \frac{y_1^{(s)}}{s!}(x-x_1)^s.$$

In what follows we shall consider an important particular case of the Hermite interpolation problem.

Let $r_1 = r_2 = \cdots = r_m = 2$. This means that we want to interpolate the data,

$$F(p, x_k) = y_k, \ k = 1, 2, \ldots, m,$$

and to have given slopes

$$F'_x(p, x_k) = y'_k, \ k = 1, 2, \ldots, m,$$

at the knots x_k. This gives $2m$ conditions and we should have $l = 2m$. Hence we may use an algebraic polynomial of degree $2m-1$,

$$H(p, x) := p_1 x^{2m-1} + p_2 x^{2m-2} + \cdots + p_{2m-1}x + p_{2m}.$$

In principle, it is possible to determine the vector $p := [p_1, p_2, \ldots, p_{2m}]^\top$ by the system of equations

$$\begin{aligned} H(p, x_k) &= p_1 x_k^{2m-1} + p_2 x_k^{2m-2} + \cdots + p_{2m-1}x_k + p_{2m} = y_k, \qquad (14.17) \\ H'_x(p, x_k) &= p_1(2m-1)x_k^{2m-2} + p_2(2m-2)x_k^{2m-3} + \cdots + p_{2m-1} = y'_k, \\ & \quad k = 1, 2, \ldots, m. \end{aligned}$$

Indeed, it may be proved that for $m > 1$ the determinant of the system (14.17) is equal to

$$\prod_{k=1}^{m-1} \prod_{l=k+1}^{n} (x_k - x_l)^4 > 0.$$

In the case $m = 1$ this determinant is -1.

Example 14.6 For $m = 1$ we have $H(p, x) = p_1 x + p_2$ and the system (14.17) becomes

$$\begin{aligned} x_1 p_1 + p_2 &= y_1, \\ p_1 &= y'_1. \end{aligned}$$

Hence

$$H(p, x) = y'_1(x - x_1) + y_1.$$

Example 14.7 Let $m = 2$, $x_1 = 0$ and $x_2 = 1$. Then

$$H(p, x) = p_1x^3 + p_2x^2 + p_3x + p_4$$

and the system (14.17) becomes

$$\begin{aligned}
&p_4 = y_1,\\
&p_3 = y_1',\\
&p_1 + p_2 + p_3 + p_4 = y_2,\\
&3p_1 + 2p_2 + p_3 = y_2'.
\end{aligned}$$

The solution of this system is

$$\begin{aligned}
p_1 &= y_1' + y_2' + 2(y_1 - y_2),\\
p_2 &= -2y_1' - y_2' + 3(y_2 - y_1),\\
p_3 &= y_1',\\
p_4 &= y_1.
\end{aligned}$$

If in particular $y_1' = y_2' = 0$ then the polynomial is

$$H(p, x) = (y_1 - y_2)(2x^3 - 3x^2) + y_1.$$

However, the calculation of p from the linear system (14.17) of order $2m$ should be avoided. An easy way to do this is to look for $H(p, x)$ in the form

$$H(p, x) = \sum_{k=1}^{m} (y_k + b_k(x - x_k))L_k^2(x). \tag{14.18}$$

Here L_k are the Lagrange polynomials of degree $m - 1$, corresponding to the points $x_1, x_2, \ldots, x_m$,

$$\begin{aligned}
L_k(x) &:= \frac{\omega_k(x)}{\omega_k(x_k)},\\
\omega_k(x) &:= \prod_{1\le l\le m, l\ne k} (x - x_l),\ k = 1, 2, \ldots, m,
\end{aligned}$$

while the constants $b_1, b_2, \ldots, b_m$ are to be determined.

Having in mind that $L_k(x_l) = \delta_{k,l}$, where $\delta_{k,l}$ is the Kronecker delta, we see that $H(p, x_k) = y_k$, $k = 1, 2, \ldots, m$. Hence we have to satisfy only the conditions $H_x'(p, x_k) = y_k'$. Since

$$H_x'(p, x) = \sum_{k=1}^{m} (b_kL_k(x) + 2(y_k + b_k(x - x_k))L_k'(x))L_k(x),$$

we obtain

$$H'_x(p, x_s) = b_s + 2y_s L'_s(x_s) = y'_s,\ s = 1, 2, \ldots, m,$$

and

$$b_s = y'_s - 2y_s L'_s(x_s),\ s = 1, 2, \ldots, m. \tag{14.19}$$

Hence the relations (14.18) and (14.19) solve the Hermite interpolation problem with $r_1 = r_2 = \cdots = r_m = 2$.

14.7 Spline interpolation

Let $T := [a, b] \in \mathbb{R}$ be an interval and

$$a = x_1 < x_2 < \cdots < x_n = b$$

be a division of the interval T. Suppose that we want to interpolate the data $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, $m \geq 2$, from $\mathbb{R}^2$, or

x	x_1	x_2	$\ldots$	x_m
y	y_1	y_2	$\ldots$	y_m

by a function $S : T \to \mathbb{R}$ which is polynomial on each interval $T_k := [x_k, x_{k+1}]$ and has a second continuous derivative on T. Such a function is called a *spline.* When S is a cubic polynomial on each T_k we have a *cubic spline.*

We stress that for $m > 2$ the spline is *not* a polynomial, but a piece-polynomial function.

In what follows we shall discuss the determination of the cubic spline.

On each interval T_k the spline, being a cubic polynomial, is defined by 4 parameters. Since we have $m - 1$ intervals, the total number of parameters of the spline is $4m - 4$.

We have $m - 2$ internal points $x_2, \ldots, x_{m-1}$. The conditions for continuity of S, S' and S'' at these points are $3(m - 2) = 3m - 6$. In addition we have m interpolation conditions

$$s(x_k) = y_k,\ k = 1, 2, \ldots, m. \tag{14.20}$$

So up to now there are $3m - 6 + m = 4m - 6$ conditions and $4m - 4$ unknowns. Therefore we have two conditions less than unknowns. The two missing conditions may be chosen as $S''(x_1) = 0$, $S''(x_n) = 0$. The resulting spline is called *natural.*

Of course, instead of the latter natural two conditions we can choose any other two conditions on S' or S'' at the end points x_1 and x_m, e.g. $S'(x_1) = S'_1$, $S'(x_m) = S'_m$, where $S'_1, S'_m \in \mathbb{R}$ are given numbers. Yet another way to determine the missing conditions is presented below.

The cubic spline is easily constructed, see e.g. [8]. Consider the interval T_k and set

$$h_k := x_{k+1} - x_k,\ u = u(x) := \frac{x - x_k}{h_k},\ v = v(x) := 1 - u.$$

Note that when $x \in T_k$ we have $u, v \in [0, 1]$.

We shall find the spline in the form

$$S(x) = uy_{k+1} + vy_k + h_k^2((u^3 - u)z_{k+1} + (v^3 - v)z_k),\ x \in T_k,$$

where z_k, z_{k+1} are constants which shall be determined later on. We stress that for this choice of s it is fulfilled

$$S(x_k) = y_k,\ S(x_{k+1}) = y_{k+1}$$

which shows that the interpolation conditions (14.20) are always satisfied.

Further on we have

$$\begin{aligned} S'(x) &= d_k + h_k((3u^2 - 1)z_{k+1} - (3v^2 - 1)z_k), \\ S''(x) &= 6(uz_{k+1} + vz_k), \\ S'''(x) &= \frac{6}{h_k}(z_{k+1} - z_k), \end{aligned}$$

where

$$d_k := \frac{y_{k+1} - y_k}{h_k}.$$

Note that $u(x_k) = 0$, $v(x_k) = 1$ and $u(x_{k+1}) = 1$, $v(x_{k+1}) = 0$. Hence the right derivative of S at x_k and its left derivative at x_{k+1} are determined from

$$\begin{aligned} S'(x_k + 0) &= d_k - h_k(z_{k+1} + 2z_k), \\ S'(x_{k+1} - 0) &= d_k + h_k(2z_{k+1} + z_k). \end{aligned}$$

Now the conditions

$$S'(x_k - 0) = S'(x_k + 0),\ k = 2, \ldots, m - 1,$$

yield the equations

$$h_{k-1}z_{k-1} + 2(h_{k-1} + h_k)z_k + h_kz_{k+1} = d_k - d_{k-1},\ k = 2, \ldots, m - 1.$$

Thus we have $m-2$ linear equations for m unknowns $z_1, z_2, \ldots, z_m$. The missing two conditions may be chosen in different ways. If we want to construct the natural spline then $z_1 = 0$, $z_n = 0$.

There is also another approach. Suppose that

$$P_1 : T_1 \cup T_2 \cup T_3 \to \mathbb{R},\ P_m : T_{m-2} \cup T_{m-1} \cup T_m \to \mathbb{R}$$

are the cubic polynomials interpolating the first four points (x_k, y_k), $k = 1, 2, 3, 4$ and the last four points (x_l, y_l), $l = m-3, m-2, m-1, m$, respectively. Then we may require the fulfilment of the conditions

$$S'''(x_1) = P_1'''(x_1),\ S'''(x_m) = P_m'''(x_m).$$

This leads to the equations

$$\begin{aligned} -h_1 z_1 + h_1 z_2 &= h_1^2 d_1^{(3)}, \\ h_{n-1} z_{n-1} - h_{n-1} z_n &= -h_{n-1}^2 d_{n-3}^{(3)}, \end{aligned}$$

where

$$d_k^{(1)} := d_k,\ d_k^{(2)} := \frac{d_{k+1}^{(1)} - d_k^{(1)}}{x_{k+2} - x_k},\ \ d_k^{(3)} := \frac{d_{k+1}^{(2)} - d_k^{(2)}}{x_{k+3} - x_k}.$$

As a result we obtain the linear algebraic equation

$$Az = b, \tag{14.21}$$

where $z = [z_1, z_2, \ldots, z_m]^\top \in \mathbb{R}^m$. The elements $a_{k,l}$ of the matrix $A \in \mathbb{R}^{m\times m}$ are determined from

$$\begin{aligned} a_{1,1} &= -h_1,\ a_{1,2} = h_1,\ a_{1,l} = 0,\ l > 2, \\ a_{k,k-1} &= h_{k-1},\ a_{k,k} = 2(h_{k-1} + h_k),\ a_{k,k+1} = h_k, \\ a_{s,k} &= 0 \text{ for } s < k-1 \text{ and } s > k+1,\ k = 2, \ldots, n-2, \\ a_{m,m-1} &= h_{m-1},\ a_{m,m} = -h_{m-1},\ a_{s,m} = 0 \text{ for } s < m-1, \end{aligned}$$

and the elements b_k of the vector $b = [b_1, b_2, \ldots, b_m]^\top \in \mathbb{R}^m$ are given by

$$\begin{aligned} b_1 &= h_1^2 d_1^{(3)}, \\ b_k &= d_k - d_{k-1},\ k = 2, \ldots, m-1, \\ b_m &= -h_{m-1}^2 d_{m-3}^{(3)}. \end{aligned}$$

The matrix A is symmetric, three–diagonal and non–singular. Hence there is a unique solution to equation (14.21). Applying the Gauss elimination method we obtain the solution from

$$\begin{aligned} z_m &= \frac{t_m}{a_m}, \\ z_k &= \frac{t_k - h_k z_{k+1}}{a_k}, \ k = m-1, m-2, \ldots, 2, 1. \end{aligned}$$

Here the quantities t_k and a_k ($k = 1, 2, \ldots, m$) are determined by the recurrence relations

$$\begin{aligned} t_1 &= b_1, \\ t_k &= b_k + \frac{h_{k-1} t_{k-1}}{a_{k-1}}, \ k = 2, \ldots, m-1, \\ t_m &= b_m - \frac{h_{m-1} t_{m-1}}{a_{m-1}} \end{aligned}$$

and

$$\begin{aligned} a_1 &= -h_1, \\ a_k &= a_{k,k} - \frac{h_{k-1}^2}{a_{k-1}}, \ k = 2, \ldots, m-1, \\ a_m &= -h_{m-1} - \frac{h_{m-1}^2}{a_{m-1}}. \end{aligned}$$

Due to the special structure of the data, the determination of z is usually done very accurately in machine arithmetic.

Note finally that if we want to use the explicit expression for the spline then we may use the formulae

$$S(x) = y_k + \beta_k (x - x_k) + \gamma (x - x_k)^2 + \delta_k (x - x_k)^3, \ x \in T_k,$$

where

$$\begin{aligned} \beta_k &:= \frac{y_{k+1} - y_k}{h_k} - h_k (2z_k + z_{k+1}), \\ \gamma_k &:= 3z_k, \\ \delta_k &:= \frac{z_{k+1} - z_k}{h_k}. \end{aligned}$$

Example 14.8 Let $h_1 = h_2 = \cdots = h_{m-1} = h = (b-a)/(m-1)$. Then the matrix A from (14.21) has the form $A = hA_1$, where

$$A_1 := \begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 4 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 4 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -1 \end{bmatrix}.$$

14.8 Back interpolation

Suppose that we want to compute an approximation for the root $x_0 \in [a,b]$ of the continuous function $f : [a,b] \to \mathbb{R}$ (more details for root finding are given elsewhere).

Note that a convenient sufficient condition for existing of such a root is the inequality

$$f(a)f(b) \leq 0.$$

Indeed, the last inequality will be fulfilled if either (i) $f(a)f(b) = 0$ or (ii) $f(a)f(b) < 0$. In case (i) we have $f(a) = 0$ and/or $f(b) = 0$ and hence $x_0 = a$ and/or $x_0 = b$. This case, however, is not interesting. In the interesting case (ii) the function f has a root $x_0 \in (a,b)$.

Suppose that we have the data

$$(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m), \; y_k := f(x_k).$$

Then a possible approach to find an approximation ξ for x_0 is to determine an interpolation function $F(p, \cdot)$ with $F(p, x_k) = y_k$ and to compute ξ as the root (or one of the roots) of the equation $F(p, x) = 0$ in x which lies in (a,b). This approach, however, may not be recommended since it involves the solution of an additional nonlinear equation.

Another, more effective approach, is based on the so called *back interpolation* which consists in inverting the roles of x and y and approximating the data

$$(y_1, x_1), (y_2, x_2), \dots, (y_m, x_m). \tag{14.22}$$

This approach should be applied when the function f is monotone, i.e. either $y_1 < y_2 < \cdots < y_m$ or $y_1 > y_2 > \cdots > y_m$.

Let $G(q,\cdot)$, where $q \in \mathbb{R}^m$ is a vector–parameter, be the function interpolating the data (14.22),

$$G(q, y_k) = x_k,\ \ k = 1, 2, \ldots, m.$$

Then the approximation of x_0 is simply found as

$$x_0 \approx G(q, 0).$$

Usually the back interpolation is used for a small number of knots, e.g. $m = 3$, when an interpolating polynomial of second degree may be implemented, and in combination with other root finding techniques.

Example 14.9 Suppose that we have the data

$$(x_1, y_1) = (0, -1),\ (x_2, y_2) = (1, a),\ (x_3, y_3) = (2, 1),$$

where $0 < a < 1$. The function f with $f(0) = -1$ and $f(2) = 1$ has a root x_0 in the interval $(0, 2)$. Since $f(1) = a > 0$ there is a root even in the interval $(0, 1)$.

The standard interpolation by algebraic polynomials gives

$$y(x) := -1 + (2a + 1)x - ax^2.$$

The equation $y(x) = 0$ has two roots. Taking the root from the interval $(0, 1)$ we get

$$x_0 \approx \frac{2}{2a + 1 + \sqrt{4a^2 + 1}}.$$

At the same time the back interpolation gives

$$x(y) = \frac{1 - a - a^2}{1 - a^2} + y + \frac{a}{1 - a^2} y^2$$

and hence

$$x_0 \approx \frac{1 - a - a^2}{1 - a^2}.$$

Note that in this case the approximation to x_0 will be in the interval $(0, 1)$ if $a < (\sqrt{5} - 1)/2 \approx 0.618$.

14.9 Interpolation by MATLAB

There are several commands in MATLAB for interpolation of one– and two–dimensional data interpolation.

Consider first the one–dimensional interpolation. Let $X = [x_1, x_2, \ldots, x_m]$ and $Y = [y_1, y_2, \ldots, y_m]$ be two m–vectors, representing the data, and let the elements of X be in descending or ascending order, for example $x_1 < x_2 < \cdots < x_m$. We shall consider Y as a function F (usually unknown) of X so that $y_k = F(x_k)$, $k = 1, 2, \ldots, m$.

We shall be interested in computing the values Y^0 for Y at the elements of a given vector X^0. Here X^0 is an arbitrary n–vector whose elements $x_1^0, x_2^0, \ldots, x_n^0$ are in the interval $[x_1, x_m]$. The interesting case is when $x_1^0 < x_2^0 < \cdots < x_n^0$ and X and X^0 have no common elements although the program will work in this case as well.

One simple interpolation is performed by the command

```
>> Y0 = interp1(X,Y,X0,'nearest')
```

It returns an n–vector Y^0 with elements y_k^0, corresponding to the arguments x_k^0, and produced by the so called “nearest neighbor” interpolation of the data (X, Y). Let for example $x_k^0 \in [x_p, x_{p+1}]$. Then $y_k^0 = y_p$ when $x_p \leq x_k^0 < (x_p + x_{p+1})/2$ and $y_k^0 = y_{p+1}$ when $(x_p + x_{p+1})/2 \leq x_k^0 \leq x_{p+1}$. Hence here we have a discontinuous piece–wise constant interpolation of the data.

The command

```
>> Y0 = interp1(X,Y,X0)
```

returns an n–vector Y^0 with elements $y_k^0 = L(x_k^0)$, where L is a continuous piece–wise linear function such that $y_k = L(x_k)$, $k = 1, 2, \ldots, m$. The same result is obtained by the command `interp1(X,Y,XX,'linear')`.

The command

```
>> Y0 = interp1(Y,X0)
```

is similar to `interp1(X,Y,X0)` and assumes that `X = 1:m`, i.e. that $x_k = k$ for $k = 1, 2, \ldots, m$.

The command

```
>> Y0 = interp1(X,Y,X0,'spline')
```

produces an n–vector Y^0 with elements $y_k^0 = S(x_k^0)$, where S is the cubic spline, interpolating the data defined by the pair (X, Y).

The similar command

```
>> Y0 = interp1(X,Y,X0,'cubic')
```

returns an n–vector Y^0 with elements $y_k = H(u_k)$, where H is the cubic Hermite interpolant of the data (X, Y). The same result is obtained by replacing `'cubic'` by `'pchip'`.

As mentioned above, the functions described so far are intended for interpolation, i.e. for computing $Y0$ for points $X0$ in the interval $[x_1, x_m]$, spanned by X. If we want to extrapolate the behavior of Y outside this interval, we may use the command

```
>> YY = interp1(X,Y,X0,'method','extrap')
```

Here `'method'` may stand for `'linear'`, `'spline'` or `'cubic'`.

Consider now the two–dimensional interpolation. Suppose that the data is given by the vector triple (X, Y, Z) of the m–vectors X, Y and Z, where Z is considered as a function of X, Y. The problem is to find an interpolant $Z = G(X, Y)$ such that $z_k = G(x_k, y_k)$ for $k = 1, 2, \ldots, m$.

Let X^0, Y^0 be two n–vectors at which we want to interpolate the data, i.e. we want to determine an n–vector Z^0 with elements z_k^0 such that $z_k^0 = G(x_k^0, y_k^0)$ for $k = 1, 2, \ldots, n$.

The command

```
>> Z0 = interp2(X,Y,Z,X0,Y0,'method')
```

returns an n–vector Z^0 with elements z_k^0, corresponding to the points (x_k^0, y_k^0). The string `'method'` defines the method used. As in the one–dimensional case, this may be `'nearest'`, `'linear'`, or `'cubic'`. Note also that in versions of MATLAB 6.0 and higher, the last two strings are replaced by `bilinear` and `bicubic`.

The command

```
>> D = griddata(X,Y,Z,X0,Y0)
```

returns an n–vector D that represents a mesh, on which the interpolation of the surface $Z = G(X, Y)$ is done. There are also other options of this command which may be viewed by using `help griddata`.

The reader is advised to generate sets of data (X, Y) and (X, Y, Z) and to experiment with the commands described above.

Cubic spline one–dimensional data interpolation can also be done in MATLAB by variants of the command `spline`.

For example, the command

```
>> spline(X,Y,X0)
```

is the same as `interp1(X,Y,X0,'spline')`.

The command

```
>> S = spline(X,Y)
```

computes the cubic spline function S (we recall that S is a piece–wise cubic function) which interpolates the data (X, Y). To evaluate S at a given set of points, e.g. the elements x_k^0 of the vector X^0, one may use the command

```
>> P0 = ppval(P,X0)
```

The result will be the same using the command `spline(X,Y,XX)`.

14.10 Problems and exercises

Exercise 14.1 Construct various "experimental" data with $X = [1, 2, \ldots, m]$ and $Y = [y_1, y_2, \ldots, y_m]$ randomly chosen (e.g. `X = 1:m; Y = rand(1,m)` or `Y = randn(1,m)`). Choose an n–vector X^0 with elements between 1 and m. Interpolate the data by the methods `'nearest'`, `'linear'`, `spline'` and `'cubic'`.

Exercise 14.2 As in Exercise 14.1, construct various "experimental" data with $X = [x_1, x_2, \ldots, x_m]$, $Y = [y_1, y_2, \ldots, y_m]$ and $Z = [z_1, z_2, \ldots, z_m]$ randomly chosen, e.g. `X = rand(1,m); Y = rand(1,m); Z = randn(1,m)`. Choose n–vectors X^0 and Y^0 with elements between 0 and 1. Interpolate the corresponding 2D–data by the methods `'nearest'`, `'linear'` and `'cubic'` (or `'bilinear'` and `'bicubic'`).

Exercise 14.3 Construct a numerical experiment with reference data generated by a known function `fun`, choosing e.g. `X = 1:10; Y = sin(X)`. Choose a 9–vector `X0 = 1.5:1:9.5` with elements $1.5, 2, 5, \ldots, 9.5$. Interpolate the data (X, Y) at the elements of X^0 by using the methods `'nearest'`, `'linear'`, `spline'` and `'cubic'`. Compare the computed result `Y0` with the "exact" result `sin(X0)` by analyzing the absolute errors `error = norm(Y0 - sin(X0))` for the different interpolation methods.

Make similar experiments with other functions with "wilder" behavior (the sine function considered above is a very "pet" function).

Exercise 14.4 Similarly to Exercise 14.3, construct numerical 2D–interpolation tests with data (X, Y, Z), generated by a known (reference) function $Z = F(X, Y)$.

Exercise 14.5 Write an algorithm and a MATLAB program for interpolation by fractional-polynomial functions by the scheme described in Section 14.5. Make experiments by randomly chosen data. For each case compute the condition number of the matrix A.

Exercise 14.6 Make an analysis of the rounding errors in the computation of $H(p, x)$ for the case $r_1 = r_2 = \cdots = r_m = 2$ (see Section 14.6).

Chapter 15

Least squares approximation

15.1 Introductory remarks

In this chapter we consider the least squares approximation of data. This type of approximation is usually used when we have a large number of observations and an approximating function with much less number of free parameters.

The least squares approximation has two important features. First, it fits very well data that is contaminated by normally distributed uncertainties (this has been established by Gauss[1]). And second, in many cases it leads to a closed form solution.

15.2 Algebraic polynomials

Suppose that we want to approximate the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

by a low order polynomial

$$P(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1},$$

where $n \le m - 1$. Note that usually $n/m \ll 1$. Then we can solve the least squares problem

$$Ap \simeq y \tag{15.1}$$

[1]C. Gauss, German mathematician, 1777–1855

relative to the vector $p := [p_1, p_2, \dots, p_{n+1}]^\top \in \mathbb{R}^{n+1}$, where

$$A := \begin{bmatrix} x_1^n & x_1^{n-1} & \dots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \dots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_m^n & x_m^{n-1} & \dots & x_m & 1 \end{bmatrix}, \; y := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

The elements of the $m \times (n+1)$ matrix A are given by $A(k,l) = x_k^{n+1-l}$, $k = 1, 2, \dots, m$, $l = 1, 2, \dots, n+1$.

The solution of (15.1) may formally be written as

$$p = A^\dagger y,$$

where $A\dagger$ is the Moore–Penrose pseudoinverse of A. When A is of full rank, we have $A^\dagger = (A^\top A)^{-1} A^\top$. We stress, however, that this elegant expression for $A^\dagger$ is usually not used in machine computations.

Moreover, in practice the least squares problem is solved directly (by using QR decomposition or singular value decomposition of A) without computing the pseudoinverse $A^\dagger$.

15.3 Fractional–polynomial functions

Let again the data

$$(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$$

be given. Suppose that we want to use a least squares approximation by the fractional–polynomial function

$$Y(c, x) = \frac{P(a, x)}{Q(b, x)} := \frac{a_0 + a_1 x + \cdots + a_p x^p}{1 + b_1 x + \cdots + b_q x^q},$$

where

$$\begin{aligned} a &:= [a_0, a_1, \dots, a_p] \in \mathbb{R}^{1\times(p+1)}, \; b := [b_1, b_2, \dots, b_q] \in \mathbb{R}^{1\times q}, \\ c &:= [a, b] =: [c_0, c_1, \dots, c_{p+q+1}] = [a_0, a_1, \dots, a_n, b_1, b_2, \dots, b_p]^\top \in \mathbb{R}^{1\times(p+q+1)} \end{aligned}$$

and $p + q \le m - 1$ (usually $(p+q)/m \ll 1$). Then the least squares problem may be formulated as

$$\rho(c) := \sum_{k=1}^{m} (Y(c, x_k) - y_k)^2 = \min! \qquad (15.2)$$

Such problems arise in practice, for example in the modelling of chemical processes.

Unfortunately, the function ρ is not quadratic in c as in the case when Y is a polynomial in x. Hence the system of equations

$$\frac{\partial \rho(c)}{\partial c_k} = 0,\ \ k = 0, 1, \ldots, p + q + 1,$$

is not linear and cannot be solved directly in order to determine of c.

Below we describe a recurrence relation for computing the solution c of the optimization problem (15.2).

Suppose that we have determined the l–th approximation

$$c^{(l)} = [a^{(l)}, b^{(l)}]$$

of c, starting from some initial vector $c^{(0)} = [a^{(0)}, b^{(0)}]$, e.g. $c^{(0)} = [0, 0]$. Then the $(l + 1)$–th approximation $c^{(l+1)}$ is computed as the solution $\gamma = [\alpha, \beta]$ of the standard linear least squares problem

$$\rho_l(\gamma) := \sum_{k=1}^{m} \left(\frac{P(\alpha, x_k) - y_k Q(\beta, x_k)}{Q(b^{(l)}, x_k)} \right)^2 = \min!$$

This problem can now be solved by QR decomposition or singular value decomposition. We leave the details to the reader.

15.4 Trigonometric polynomials

Consider the least squares approximation of the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

with $x_1 < x_2 < \cdots < x_m < x_1 + 2\pi$ by trigonometric polynomials

$$T(c, x) := \frac{a_0}{2} + \sum_{l=1}^{n} (a_l \cos lx + b_l \sin lx).$$

Here we have denoted by $c = [a, b]^\top$, where $a := [a_0, a_1, \ldots, a_n]$ and $b := [b_1, b_2, \ldots, b_n]$, the $(2n+1)$–vector of unknown coefficients. Since we have $2n+1$ coefficients and m points (x_k, y_k), it is reasonable to suppose that $2n + 1 < m$ (otherwise it would be possible to solve the interpolation problem $T(c, x_k) = y_k$, $k = 1, 2, \ldots, m$).

The least squares approximation problem

$$T(c, x_k) \simeq y_k,\ k = 1, 2, \ldots, m,$$

may now be written in the standard form

$$Ac \simeq y,\ y := [y_1, y_2, \ldots, y_m]^\top,$$

where the elements of the matrix $A \in \mathbb{R}^{m\times(2n+1)}$ are given by $A(k, 1) = 1/2$, $A(k, l) = \cos((l-1)x_k)$ for $2 \le l \le n+1$ and $A(k, l) = \sin(l-n-1)x_k$ for $n+2 \le l \le 2n+1$.

15.5 Approximation of multi–dimensional surfaces

Consider the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N),$$

where $x_k = [\xi_{k,1}, \xi_{k,2}, \ldots, \xi_{k,m}]^\top \in \mathbb{R}^m$ and $y_k \in \mathbb{R}$. The case when y_k is also a vector is treated similarly. Thus we model a surface

$$y = f(x),\ x := [\xi_1, \xi_2, \ldots, \xi_m]^\top \in \mathbb{R}^m,$$

in the $(m+1)$–dimensional space of the variables $x \in \mathbb{R}^m$ and $y \in \mathbb{R}$.

Suppose first that we want to approximate the data by the affine function

$$F_1(c, x) := a + b^\top x = a + \sum_{k=1}^m b_k\xi_k =: c_{m+1} + c_1\xi_1 + c_2\xi_2 + \cdots + c_m\xi_m$$

in $x = [\xi_1, \xi_2, \ldots, \xi_m]^\top$. Let $n := m+1 < N$. Then we may formulate the least squares problem

$$Ac \simeq y,$$

where

$$A := [X, e] \in \mathbb{R}^{N\times n}.$$

Here $X \in \mathbb{R}^{N\times m}$ is a matrix with elements $X(k, l) = \xi_{k,l}$ ($k = 1, 2, \ldots, N$, $l = 1, 2, \ldots, m$) and $e = [1, 1, \ldots, 1]^\top \in \mathbb{R}^N$.

Suppose next that we want to approximate the data by the quadratic function

$$F_2(c, x) := a + b^\top x + x^\top Qx = a + \sum_{k=1}^m b_k\xi_k + \sum_{k\le s}^m \sum_{s=1}^m q_{k,s}\xi_k\xi_s,$$

where $a \in \mathbb{R}$, $b \in \mathbb{R}^m$ and $Q = Q^\top \in \mathbb{R}^{m \times m}$. Here the vector c contains the scalar a, the m elements b_k of b and the

$$\mu := \frac{m(m+1)}{2}$$

quantities $q_{k,s}$ for $k = 1, 2, \ldots, s$ and $s = 1, 2, \ldots, m$. Thus we have a total of

$$n := \mu + m + 1 = \frac{(m+1)(m+2)}{2}$$

free parameters, i.e. $c = [c_1, c_2, \ldots, c_n]^\top \in \mathbb{R}^n$.

Consider the vector

$$u := \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix} \in \mathbb{R}^\mu,$$

where

$$q_k := \begin{bmatrix} q_{k,k} \\ q_{k,k+1} \\ \vdots \\ q_{k,m} \end{bmatrix} \in \mathbb{R}^{m-k+1}, \ k = 1, 2, \ldots, m.$$

If we denote $u = [u_1, u_2, \ldots, u_\mu]^\top \in \mathbb{R}^\mu$ then for

$$s = s(k,l) := \frac{k(2m-k+1)}{2} + l, \ k = 0, 1, \ldots, m-1, \ l = 1, 2, \ldots, m-k, \quad (15.3)$$

we have

$$u_s = q_{k+1,k+l}, \ k = 0, 1, \ldots, m-1, \ l = 1, 2, \ldots, m-k.$$

The relation (15.3) determines uniquely the integers k and l,

$$k = k(s), \ l = l(s), \ s = 1, 2, \ldots, \mu.$$

Now setting

$$c = \begin{bmatrix} u \\ b \\ a \end{bmatrix} \in \mathbb{R}^n$$

we have

$$c_p = \begin{cases} u_p, & p = 1, 2, \ldots, \mu \\ b_{p-\mu}, & p = \mu+1, \mu+2, \ldots, \mu+m \\ a, & p = \mu+m+1 \end{cases}$$

Suppose that $N > n$ which is the standard assumption in least squares approximation. Then we may formulate the least squares problem

$$Ac = Uu + Bb + ea \simeq y,$$

where

$$A := [U, B, e] \in \mathbb{R}^{N\times n},\ y := [y_1, y_2, \ldots, y_N]^\top \in \mathbb{R}^N$$

and

$$U := [u_{\alpha,\beta}] \in \mathbb{R}^{N\times\mu},\ B := [b_{\alpha,\gamma}] \in \mathbb{R}^{N\times m},\ e := [1, 1, \ldots, 1]^\top \in \mathbb{R}^N.$$

We have

$$\begin{aligned} u_{\alpha,\beta} &= \xi_{\alpha,k(\beta)+1}\xi_{\alpha,k(\beta)+l(\beta)}, \quad \alpha = 1, 2, \ldots, N,\ \beta = 1, 2, \ldots, \mu, \\ b_{\alpha,\gamma} &= \xi_{\alpha,\gamma}, \quad \alpha = 1, 2, \ldots, N,\ \gamma = 1, 2, \ldots, m. \end{aligned}$$

The above (relatively complex) relations are illustrated by the next example.

Example 15.1 Let $m = 2$. Then $\mu = 3$, $n = 6$ and

$$\begin{aligned} F_2(c, x) &= a + b_1\xi_1 + b_2\xi_2 + q_{1,1}\xi_1^2 + q_{1,2}\xi_1\xi_2 + q_{2,2}\xi_2^2 \\ &=: c_6 + c_4\xi_1 + c_5\xi_2 + c_1\xi_1^2 + c_2\xi_1\xi_2 + c_3\xi_2^2. \end{aligned}$$

Thus the α–th row ($\alpha = 1, 2, \ldots, N$) of A is the row 7–vector

$$[\xi_{\alpha,1}^2, \xi_{\alpha,1}\xi_{\alpha,2}, \xi_{\alpha,2}^2, \xi_{\alpha,1}, \xi_{\alpha,2}, 1].$$

15.6 General case of linear dependence on parameters

One of the most important cases of least squares approximation is when the approximating function depends linearly on the unknown parameters. So far, such were the cases considered in Sections 15.2, 15.4 and 15.5 (the last one with vector argument of the approximating function). In this section we shall consider the general case when the approximating function is linear in the unknown parameters. We also suppose that the argument and the function itself are scalars, i.e.

$$Y = F(x, c) := c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x). \tag{15.4}$$

Here $c := [c_1, c_2, \ldots, c_n]^\top$ is the unknown vector–parameter and x is the argument, varying in certain interval. The system of functions $\{f_1, f_2, \ldots, f_n\}$ usually has some "nice" properties, e.g. it is linearly independent and/or it is a Chebyshov system.

Suppose that we want to approximate the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

by a function F, where usually $n < m$ and even $n/m \ll 1$. Note that the case when x_k and/or y_k are vectors, is treated similarly but is more involved.

As we already know, the approach based on the minimization of the function

$$c \mapsto \sum_{k=1}^{m} (F(x_k, c) - y_k)^2$$

leads to the so called *normal* system of equations

$$\Phi c = \varphi,$$

where the elements Φ_{rs} of the symmetric $n \times n$ matrix Φ and the elements φ_r of the n–vector φ are determined by

$$\Phi_{rs} := \sum_{k=1}^{n} f_r(x_k) f_s(x_k), \;\; \varphi_r := \sum_{k=1}^{n} y_r f_r(x_k).$$

However, the computation of the matrix Φ in machine arithmetic may lead to loss of true digits in the computed matrix $\widehat{\Phi}$ and hence to loss of accuracy in the computed solution $\widehat{c}$.

The best way to find c is via the least squares problem

$$Ac \simeq y, \tag{15.5}$$

where

$$A := \begin{bmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \ldots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \ldots & f_n(x_m) \end{bmatrix} \in \mathbb{R}^{m\times n}, \; y := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m. \tag{15.6}$$

Hence

$$c = A^\dagger y.$$

Of course, the solution of the least squares problem (15.5) is done directly by QR decomposition or singular value decomposition of the matrix A and the formation of the Moore–Penrose pseudoinverse $A^\dagger$ of A is avoided.

Consider finally the case when the data is generated by measurements, where each measurement (x_k, y_k) is characterized by a weighting coefficient $\omega_k > 0$. Consider for example the case when the argument x_k is exact while y_k is the mean of several measurements,

$$y_k = \frac{\eta_{k,1} + \eta_{k,2} + \cdots + \eta_{k,p_k}}{p_k},$$

where $p_k \geq 6$. Then, if D_k is the dispersion (or variance) of the measurements for y_k, one may assume $\omega_k = 1/D_k$. In this case it is reasonable to define c as the vector minimizing the function

$$c \mapsto \sum_{k=1}^{m} \omega_k (F(x_k, c) - y_k)^2$$

Setting $\Omega := \mathrm{diag}(\omega_1, \omega_2, \ldots, \omega_m) \in \mathbb{R}^{m\times m}$ we see that this weighted least squares problem leads to the normal equation

$$A^\top \Omega A c = A^\top \Omega y,\ c := [c_1, c_2, \ldots, c_n]^\top \in \mathbb{R}^n, \tag{15.7}$$

where the matrix $A \in \mathbb{R}^{m\times n}$ and the vector $y \in \mathbb{R}^m$ are defined in (15.6).

As we already know, the formation of the normal equation (15.7) should be avoided since, in machine arithmetic, the computed matrix $\mathrm{fl}(A^\top \Omega A)$, and, to a certain extent the computed vector $\mathrm{fl}(A^\top \Omega y)$, may be contaminated with large relative errors. So the best way to solve the latter least squares problems is via the relation

$$\Omega^{1/2} A c \simeq \Omega^{1/2} y.$$

This may be done by QR or singular value decomposition of the matrix $\Omega^{1/2} A$.

15.7 Least squares approximation by MATLAB

Let X, Y be the m–vectors, corresponding to the data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m),$$

which must be approximated in a least squares sense.

The least squares approximation by algebraic polynomials is done in MATLAB by the command

```
>> c = polyfit(X,Y,N)
```

The output $c = [c_1, c_2, \ldots, c_{N+1}]$ is the vector of the coefficients of the N–th degree polynomial

$$p(x) = c_1 x^N + c_2 x^{N-1} + \cdots + c_N x + c_{N+1}$$

which approximates the data, where $m \geq N + 1$.

Suppose that we want to approximate the data at a set of arguments, which are the elements $x_1^0, x_2^0, \ldots, x_n^0$ of the n–vector X^0. The approximated values are found by the command

```
>> Y0 = polyval(c,X0)
```

where X^0 is a vector with elements in the interval $[x_1, x_m]$. The answer Y^0 is a vector with elements $y_k^0 = p(x_k^0)$, $k = 1, 2, \ldots, n$.

The least squares approximation by general functions of type (15.4), described in Section 15.6, requires some additional efforts from the user. To solve the linear least squares problem $Ac \simeq y$ by the command

```
>> c = A\b
```

one has to compute first the matrix A and the vector y by the expressions (15.6).

15.8 Problems and exercises

Exercise 15.1 Using the recurrence scheme, proposed in Section 15.3, find an approximation

$$y = \frac{a_0 + a_1 x + a_2 x^2}{1 + b_1 x}$$

for the data

x	0	1	2	3	4
y	1	1	5/3	5/2	17/5

Exercise 15.2 Using MATLAB find the eigenvalues and eigenvectors of the matrix $A_1 \in \mathbb{R}^{n\times n}$ from Example 14.8 for $n = 4, 5, \ldots, 20$. Formulate a hypothesis about the behavior of the eigenvalues of A_1 as a function of n.

Exercise 15.3 Prove that the system of functions $x \mapsto \cos lx$, $l = 0, 1, \ldots, m$, is a Chebyshov system on the interval $[0, \pi]$, i.e. that for any collection of constants $p_0, p_1, \ldots, p_m$ the equation

$$\sum_{l=0}^{m} p_l \cos lx = 0$$

in x has at most m zeros in $[0, \pi]$.

Exercise 15.4 Let $\lambda_1 < \lambda_2 < \cdots < \lambda_m$ be given constants. Prove that:

– the system of functions $x \mapsto \exp(\lambda_k x)$, $k = 1, 2, \ldots, m$, is a Chebyshov system on any interval $[a, b]$;

– the system of functions $x \mapsto x^{\lambda_k}$, $k = 1, 2, \ldots, m$, is a Chebyshov system on any interval $[a, b]$ with $a > 0$.

Exercise 15.5 Write a MATLAB program for solving the least squares approximation problem by trigonometric polynomials as described in Section 15.4.

Exercise 15.6 Write a MATLAB program for solving the least squares approximation problem arising in the approximation of multi–dimensional surfaces as in Section 15.5.

Chapter 16

Numerical differentiation

16.1 Statement of the problem

The differentiation of a function

$$f : (a, b) \to \mathbb{R}$$

at a given point $x \in (a, b)$ may be a delicate problem, which often is connected with theoretical and numerical difficulties. One of the reasons is that the operator of differentiation is not bounded. In other words, the derivative $f' : (a, b) \to \mathbb{R}$ of a bounded function f may not be bounded.

Example 16.1 The function $x \mapsto \sin(x^{-1})$, $x \in (0, 1)$, is bounded and differentiable on $(0, 1)$, but its derivative $f'(x) = -x^{-2}\cos(x^{-1})$ is not bounded in a neighborhood of the point $x = 0$.

Another difficulty in differentiation is of numerical character and is due to the use of machine arithmetic. Instead of going into theoretical details we shall illustrated this issue by the following example in which $f : (a, b) \to \mathbb{R}$ is differentiable and $x \in (a, b)$. The computations are done in an arithmetic with rounding unit $\mathbf{u}$. We also denote $r = [-\log_{10}(\mathbf{u})]$, where $[z]$ is the entire part of the number $z \geq 0$. We recall that for the double precision in the IEEE Standard we have $\mathbf{u} \approx 10^{-16}$ and hence $r = 16$.

Example 16.2 Suppose that we want to compute the derivative $f'(x)$ approximating it by the expression

$$D(x, h) := \frac{f(x + h) - f(x)}{h}$$

for some small value of $h > 0$, e.g. $h \ll 1$ if $|x| \leq 1$ and $h/|x| \ll 1$ if $|x| > 1$. This is justified by the fact that $D(x, h)$ tends to $f'(x)$ as h tends to 0. But we cannot achieve any prescribed accuracy with this type of approximation in machine arithmetic. Indeed, if h is so small that $f(x+h)$ and $f(x)$ coincide within their first $p < r$ digital digits then there will be about $r - p = 16 - p$ true decimal digits in the computed difference $\mathrm{fl}(f(x+h) - f(x))$ and probably the same number of true digits in the computed value for $D(x, h)$. If h is of order 10^{-p} then we may expect that the exact difference between $f'(x)$ and $D(x, h)$ will also be of order 10^{-p} and the number of true digits cannot be larger than the minimum of the numbers $r - p$ and p. Choosing $p = r/2$ we may expect that $f'(x)$ and $D(x, h)$ have $r/2 = 8$ coinciding left–most digital digits.

16.2 Description of initial data

Consider a function $f : T \to \mathbb{R}$, where $T \subset \mathbb{R}$ is an interval. The problems of differentiation and integration of the data may be defined for various types of initial data. According to [8] there are four main types of data. We shall consider only three of them.

1. The values $f(x)$ of f are known for a prescribed finite set of values of the argument $\{x_0, x_1, \ldots, x_n\} \subset T$.
2. The values $f(x)$ of f may be computed numerically for each $x \in T$.
3. There is an explicit expression for $f(x)$ which is suitable for symbolic computations.

We stress that cases 2 and 3 are not disjoint, i.e. they may occur simultaneously.

16.3 Computation of derivatives

In what follows we shall consider the problem of numerical differentiation, which consists in finding an approximation for the derivative $f'(x)$ of the function f at the point x when the computations are done in machine arithmetic with rounding unit $\mathbf{u}$. The numerical approximation of $f'(x)$ is done in two stages.

At the first stage one finds an approximation formula for computing $f'(x)$ as the ratio of two finite increments, e.g.

$$f'(x) \approx \frac{\Delta f(x)}{\Delta x}.$$

Here the quantities $\Delta f(x)$ and Δx depend on the approximation formula. At this stage we introduce the so called *truncation*, or *approximation error* E_T which satisfies

$$E_T = O(\Delta x^m),\ \Delta x \to 0.$$

Here the integer $m \in \mathbb{N}$ determines the *order* of approximation.

At the second stage one computes the increments $\Delta f(x)$ and Δx so as to minimize the rounding errors and to avoid significant cancellation in the computation of $\Delta f(x)$ (some cancellation is unavoidable in the process of numerical differentiation). As a result we get another error E_R, in which the effects of rounding and cancellations are taken into account.

Thus the total error is

$$E = E_T + E_R$$

and it depends on f, x, Δx, of the computational algorithm implemented and on the machine arithmetic (mainly on the rounding unit $\mathbf{u}$). At this second stage one looks for an increment for x, which minimizes the total error.

The derivatives of higher order can be computed in a similar way.

It is important to stress again that the naive strategy simply to decrease Δx without taking into account the rounding errors and the mutual cancellation may soon lead to a numerical catastrophe, see Example 16.2 and Problems 16.1 and 16.2.

As an heuristic rule for determining of Δx one may use

$$\Delta x \approx \mathbf{u}^{1/(m+1)}$$

when an approximation formula with truncation error of order m is used and $x = \mathrm{O}(1)$.

In order to analyze truncation errors we shall assume that the function f has a sufficient number of derivatives in the interval (a, b).

There are different approaches to the approximation of the derivatives of f at a given point x, see e.g. [27]. One of the most transparent ones is based on the Taylor's formulae which we shall write in the form

$$f(x+h) = f(x) + f'(x)h + f''(x+\lambda h)\frac{h^2}{2},\ \lambda \in (0,1),$$
$$f(x-h) = f(x) - f'(x)h + f''(x-\mu h)\frac{h^2}{2},\ \mu \in (0,1).$$

Hence

$$f'(x) = \frac{f(x+h) - f(x)}{h} - f''(x+\lambda h)\frac{h}{2},$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} + f''(x - \mu h)\frac{h}{2}.$$

Thus for $h > 0$ we have the formulae for *forward*

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

and *backward*

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}.$$

differentiation. In both cases the truncation error is bounded from above by the quantity

$$\frac{c_2(f)h}{2},$$

where

$$c_n(f) := \sup\{|f^{(n)}(x)| : x \in (a,b)\}.$$

A better estimate can be obtained using the Taylor formulae in the form

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f'''(x+\lambda h)\frac{h^3}{6},\ \lambda \in (0,1), \\ f(x-h) &= f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f'''(x-\mu h)\frac{h^3}{6},\ \mu \in (0,1). \end{aligned}$$

Subtracting the second formula from the first one and dividing by $2h$ we get

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - (f'''(x+\lambda h) + f'''(x-\mu h))\frac{h^2}{12}.$$

We also have

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - f'''(\theta)\frac{h^2}{6},\ \theta \in (x-h, x+h).$$

Hence the approximation formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{16.1}$$

has a truncation error, which is bounded from above by

$$\frac{c_3(f)h^2}{6}.$$

In a similar way we may show that the following formulae are valid

$$\begin{aligned} f'(x-h) &= \frac{-3f(x-h) + 4f(x) - f(x+h)}{2h} + f'''(\theta_1)\frac{h^2}{3}, \\ f'(x+h) &= \frac{f(x-h) - 4f(x) + 3f(x+h)}{2h} + f'''(\theta_2)\frac{h^2}{3}, \end{aligned}$$

where $\theta_1, \theta_2 \in (x-h, x+h)$. These formulae can be used when $x - h = a$ is the left end of the interval $[a, b]$, or $x + h = b$ is the right end of this interval.

When the values of f are contaminated with errors one may use differentiation formulae derived after differentiation of polynomials approximating f in least squares sense. Without going into details we shall give a set of three such formulae:

$$\begin{aligned}
f'(x) &\approx \frac{3f(x+h) + 10f(x) - 18f(x-h) + 6f(x-2h) - f(x-3h)}{12h}, \\
f'(x) &\approx \frac{8(f(x+h) - f(x-h)) + f(x-2h) - f(x+2h)}{12h}, \\
f'(x) &\approx \frac{f(x+3h) - 6f(x+2h) + 18f(x+h) - 10f(x) - 3f(x-h)}{12h}.
\end{aligned}$$

As we briefly mentioned above, the numerical computation of derivatives meets two main difficulties.

First, the operation of differentiation is not bounded. In practice this means that the derivative of a function may be very large even if the function itself is of moderate size, see Example 16.1. In terms of the estimates presented above this behavior means that the constants $c_n(f)$ may be very large, which in turn requires (in order to decrease the truncation error) very small increments h.

This, however, leads to the second difficulty – the increment h cannot be too small because this will cause a catastrophic cancellation. Indeed, when decreasing h the quantities $f(x + h)$ and $f(x - h)$ become close, and, when they are of the same sign (which is the typical case) the number of lost true digits in the subtraction $f(x + h) - f(x - h)$ will increase. Since the quantities $f(x \pm h)$ are in turn computed with some rounding errors, and the result of the subtraction is divided to the small number $2h$ it becomes clear, that the danger of computational disaster is quite real. And this disaster inevitably occurs if no measures are taken to overcome it.

It is shown in [27] that the relative error in computing the derivative $f'(x) \neq 0$ by the formula (16.1) in machine arithmetic can be bounded from above by the quantity

$$\widehat{e}(h, \mathbf{u}) := \frac{b_1 h^2}{2} + b_2 \frac{\mathbf{u}}{h} + 2\mathbf{u}$$

where

$$b_1 := \frac{c_3(f)}{3|f'(x)|}, \; b_2 := |x| + \gamma \frac{|f(x)|}{|f'(x)|}.$$

The function $\widehat{e}(\cdot, \mathbf{u})$ has minimum in h for

$$h = \widehat{h} := \left(\frac{b_2}{b_1}\mathbf{u}\right)^{1/3} = O(\mathbf{u}^{1/3}).$$

For this value of h we have

$$\widehat{e}(\widehat{h}, \mathbf{u}) = \frac{3b_1^{1/3}b_2^{2/3}}{2}\mathbf{u}^{2/3} + 2\mathbf{u}.$$

This yields the following important conclusion.

When the computations are done in a standard machine arithmetic with $\mathbf{u} \approx 10^{-16}$ *then the optimal increment* h *for moderate values of* x *is of order from* 10^{-5} *to* 10^{-6}.

The use of smaller h may rapidly decrease the accuracy of the computed derivative. In turn, increasing h will lead to larger truncation error and hence to a larger total error. Thus, typically, the dependence of the total error on $h > 0$ is a function, tending to $+\infty$ for $h \to 0+$ and for $h \to +\infty$, and having an absolute minimum at some $h_0 > 0$.

An adaptive algorithm and a reliable computer program, called SDERF and written in Fortran 77, are considered in [27]. In this book approximation formulae are also presented for computing of derivatives of any order and with any prescribed truncation error.

16.4 Numerical differentiation by MATLAB

There are several commands in MATLAB for differentiation of polynomials.

As above, we shall represent the n–th degree polynomial

$$P(x) = p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1}, \; p_1 \neq 0,$$

by its coefficient row $(n+1)$–vector $p = [p_1, p_2, \ldots, p_{n+1}]$.

The command

```
>> d = polyder(p)
```

returns the n-vector $d = [np_1, (n-1)p_2, \ldots, p_n]$ corresponding to the coefficients of the derivative

$$P'(x) = np_1x^{n-1} + (n-1)p_2x^{n-2} + \cdots + 2p_{n-1}x + p_n$$

of P.

The coefficients of the derivative $D := (PQ)'$ of the product of two polynomials P and Q may be found by the command

```
>> d = polyder(p,q)
```

where q is the vector determining the polynomial Q.

Finally, the derivative $(P/Q)'$ of the ratio of the polynomials P and Q is computed in the form U/V by the command

```
>> [u,v] = polyder(p,q)
```

16.5 Problems and exercises

The use of small increments for the argument in the numerical calculation of derivatives is dangerous in machine arithmetic. The next three problems show what happens in practice. You can solve the problems in any computing environment and in particular using the interactive systems MATLAB, SYSLAB or Scilab.

Exercise 16.1 Write a program for computing the derivative of the sine function at the point $x = \pi/4$ using the formula

$$\frac{\sin(\pi/4+h)-\sin(\pi/4)}{h} = \frac{\sin(\pi/4+h)-\sqrt{2}/2}{h}$$

for a sequence of increments

$$h = 10^{-k},\ k = 1, 2, \ldots.$$

The exact answer is $\sqrt{2}/2$. For which k the computational catastrophe occurred? Compare the corresponding $h = 10^{-k}$ with $\mathbf{u}$.

Exercise 16.2 Write a program for computing the derivative of the exponential function $\exp x$ at the point $x = 1$ using the formula

$$\frac{\exp(1+h)-e}{h},\ e = \exp(1) = 2.72\ldots,$$

for

$$h = 10^{-k},\ k = 1, 2, \ldots.$$

The exact answer is e. For which k the computational disaster happened? Compare the corresponding h with $\mathbf{u}$.

Exercise 16.3 Make similar experiments with other simple functions f using the approximate formula

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

for a decreasing sequence of values for h.

Exercise 16.4 On the basis of Problems 16.1, 16.2 and 16.3 formulate a hypothesis for the observed phenomenon, which includes the rounding unit $\mathbf{u}$.

Exercise 16.5 When the derivative is computed using an approximation formula of order m in a machine arithmetic with rounding unit $\mathbf{u}$ the estimate of the total error may be written as

$$\widehat{e}(h, \mathbf{u}) = \frac{a\mathbf{u}}{h} + bh^m + c\,\mathbf{u}.$$

Find the minimum of the function $\widehat{e}(\cdot, \mathbf{u})$ in h. Comment the asymptotic behavior of this minimum for large values of m.

Chapter 17

Numerical integration

17.1 Statement of the problem

One of the fundamental problems in mathematical analysis and its applications is the evaluation of the *definite integral*

$$J(f,[a,b]) := \int_a^b f(x)\mathrm{d}x\,, \tag{17.1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a given function. Of course, in (17.1) it is enough to define the function f only on the interval $[a,b]$, $a < b$. However, we prefer this notation in order to consider the integral of the same function f on various intervals.

Definition 17.1 *A differentiable function* $F : [a,b] \to \mathbb{R}$*, such that*

$$F'(x) = f(x),\ \ x \in (a,b),$$

is called a primitive function *(or an* antiderivative*) for the function* f*.*

The primitive, when it exists, is not unique. If F_1 and F_2 are two primitives for f then the difference $F_1 - F_2$ is a constant function.

Definition 17.2 *The set* $\mathrm{Int}(f)$ *of all primitives is said to be the* indefinite integral *for* f*.*

If $F \in \mathrm{Int}(f)$ then any other member of $\mathrm{Int}(x)$ is of the form $F + C$, where C is a constant. The set $\mathrm{Int}(f)$ is always well defined since the case $\mathrm{Int}(f) = \emptyset$ is not excluded.

If $f, g : [a,b] \to \mathbb{R}$ are two functions such that $f(x) = g(x)$ for all $x \in (a,b)$ then $\mathrm{Int}(f) = \mathrm{Int}(g)$.

If F is a primitive for f then we may introduce the *definite integral in the sense of Newton* as

$$\int_a^b f(x)\mathrm{d}x := F(b) - F(a). \tag{17.2}$$

Note that here the case $a \geq b$ is not excluded when f is defined on an interval containing the points a and b.

There are functions which do not have a primitive. To give an example of such a function recall that the function $f : [a, b] \to \mathbb{R}$ has a discontinuity of first kind at the point $c \in (a, b)$ if either the limit l of f at c exists but $l \neq f(c)$, or the left l_- and right l_+ limits of f at c exist but $l_- \neq l_+$.

Example 17.1 If the function f has a discontinuity of first kind at some point $c \in (a, b)$ then f has no primitive. In particular the function $x \mapsto \mathrm{sign}(x - c)$, $c := (a + b)/2$, has no primitive. Here $\mathrm{sign}(x) = x/|x|$ if $x \neq 0$ and $\mathrm{sign}(0) = 0$.

The continuous function $F : [a, b] \to \mathbb{R}$ is called a *generalized primitive* for f if there is a finite set of points $X := \{x_1, x_2, \ldots, x_m\} \subset (a, b)$ such that

$$F'(x) = f(x),\ x \in [a, b]\backslash X.$$

Note that some authors define the generalized primitive also for the case when the set X is at most countable.

The Newton integral for functions f having a generalized primitive F may again be defined by the equality (17.2). In this case the function F is continuous and piece–wise differentiable. We stress that most (practically, all) functions used in mathematical modelling in engineering *do have* a generalized primitive. Hence the treatment of the definite integral in the sense of Newton is general enough.

Example 17.2 Consider the integral $\int_a^b \mathrm{sign}(x)\mathrm{d}x$. The function sign : $\mathbb{R} \to \mathbb{R}$ is not differentiable at the point $x = 0$ and has no primitive on intervals for which this point is internal. But the function sign has a generalized primitive $F : \mathbb{R} \to \mathbb{R}$, defined from $F(x) = |x| + C$ for any constant C. Hence $\int_a^b \mathrm{sign}(x)\mathrm{d}x = |b| - |a|$ for all $a, b \in \mathbb{R}$.

There are many other definitions of a definite integral and we shall not go into particular details here. We shall only note that when two such integrals exists (e.g. the integrals in the sense of Riemmann and in the sense of Newton) they are equal. All types of integrals satisfy the three fundamental conditions, listed below, where f, g are functions and λ, μ are constants.

- $J(\lambda f + \mu g, [a,b]) = \lambda J(f,[a,b]) + \mu J(g,[a,b])$ (linearity);
- $J(f,[a,c]) + J(f,[c,b]) = J(f,[a,b])$;
- $J(\mathbf{1},[a,b]) = b - a$, where $\mathbf{1}$ is the function $x \mapsto 1$;

Further on we shall consider a fixed interval $[a,b]$ with $a < b$ and shall write $J(f)$ (or even simply J) instead of $J(f,[a,b])$.

For simplicity we shall suppose that f is piece–wise continuous, i.e. that the interval $[a,b]$ can be represented as the disjoint union of a finite number of subintervals, such that on each of them the function f is bounded and continuous. Then the function f has a generalized primitive (the construction of such a generalized primitive is left as an exercise to the reader).

Often it is impossible, or ineffective, to perform the integration analytically (e.g. in closed form) using the formula (17.2).

Example 17.3 The integrals

$$J_n := \int_0^1 \frac{\mathrm{d}x}{1+x^n},\ n \in \mathbb{N},$$

may be computed as $J_n = F_n(1) - F_n(0)$, where the primitive F_n of the function $x \mapsto 1/(1+x^n)$ may be found in closed form using the MATHEMATICA integrator from e.g.

`http://integrals.wolfram.com/index.jsp`

The monstrous expressions for $F(x)$ when $n > 2$ will soon discourage the reader to compute J_n in this way.

In such cases the integral has to be solved numerically in machine arithmetic.

Fortunately, the operation of integration is bounded and well conditioned. Indeed, let $|f(x)| \leq M$ for $x \in (a,b)$. Then $|J(f)| \leq (b-a)M$. Moreover, if $\delta f : (a,b) \to \mathbb{R}$ is a bounded integrable perturbation to f, then

$$J(f+\delta f) - J(f) = \int_a^b \delta f(x)\mathrm{d}x$$

and

$$|J(f+\delta f) - J(f)| \leq (b-a)\Delta,$$

where $|\delta f(x)| \leq \Delta$ for $x \in (a,b)$. Thus *the length $b-a$ of the interval $[a,b]$ plays the role of absolute condition number for the operation of integration.*

Since the integral is not very sensitive to changes in the data, where the possible dangers in its numerical computation are hidden? The reader can get the idea on the basis of the next example.

Example 17.4 Suppose that f is continuous and that we approximate $J(f)$ by the expression

$$Q(f) = \sum_{k=0}^{n} a_k f(x_k),$$

where $a \leq x_0 < x_1 < \cdots < x_n \leq b$ and the constants a_k do not depend on f. Consider another continuous function $g : [a,b] \to \mathbb{R}$ such that $g(x_0) = f(x_0)$ and $g(x) = f(x)$ for $x \in [a, x_0] \cup [x_1, b]$. Then $Q(f) = Q(g)$. At the same time

$$J(f) - J(g) = \int_{x_0}^{x_1} (f(x) - g(x))\mathrm{d}x. \tag{17.3}$$

The right–hand side of (17.3) may be arbitrary large in absolute value. If e.g.

$$g(x) = f(x) - \frac{\pi A}{2(x_1 - x_0)} \sin \pi \frac{x - x_0}{x_1 - x_0}, \ x \in (x_0, x_1)$$

then $J(f) - J(g) = A$, where A is an arbitrary constant. Hence the expression $Q(f)$ computes $J(f)$ with an arbitrary large error.

To compute numerically the integral $J(f)$ one has to choose the so called *quadrature formula* $Q(f)$, which is the theoretical approximation of $J(f)$.

17.2 Quadrature formulae

Information about the basic quadrature schemes is given in the courses of Mathematical Analysis and Numerical Methods. In this section we consider only the three simplest quadratures: the formula of rectangulars, the formula of trapezoids and the Simpson formula. We also analyze the corresponding truncation errors as well as the total error which includes the contribution of rounding errors done in machine arithmetic.

Let the interval of integration $[a,b]$ be divided into n subintervals $[x_k, x_{k+1}]$ by $n+1$ points $a = x_1 < x_2 < \cdots < x_{n+1} = b$. Then we have

$$J = \sum_{k=1}^{n} J_k,$$

where

$$J := J(f, [a,b]) = \int_a^b f(x)\mathrm{d}x$$

and

$$J_k := \int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x, \ k = 1, 2, \ldots, n.$$

Denote

$$h_k := x_{k+1} - x_k, \quad \xi_k := \frac{x_k + x_{k+1}}{2}, \quad k = 1, 2, \ldots, n.$$

The *simple quadrature formula*

$$Q_k \approx J_k$$

is an approximation for J_k. In turn, the *composite quadrature formula*

$$Q := \sum_{k=1}^{n} Q_k \approx J$$

is an approximation for J.

The simple quadrature formula Q_k is of (local) *order* $m + 1$ for some $m \in \mathbb{N}$ if the *absolute local error* $|J_k - Q_k|$ satisfies the asymptotic estimate

$$|J_k - Q_k| = O(h_k^{m+1}), \ h_k \to 0.$$

In this case the following estimate for the *absolute global error* $|J - Q|$ is valid

$$|J - Q| = O(h^m), \ h \to 0,$$

where

$$h := \max\{h_k : k = 1, 2, \ldots, n\}.$$

We also say that the composite formula Q is of asymptotic order m.

We recall that the relation

$$y = O(z), \ z \to 0, \ z \geq 0,$$

means that there is a constant $K > 0$ such that $|y| \leq Kz$ for $z \to 0$ and $z \geq 0$.

It must be pointed out that there are no universal quadrature formula in the following sense. For each quadrature formula Q and each $A > 0$ there exists a continuous function f (and even a function f with infinitely many derivatives $f^{(k)}$, $k \in \mathbb{N}$) such that $|J(f) - Q(f)| > A$. That is why quadrature formulae are usually analyzed only on certain classes of functions, e.g. polynomials, functions with bounded N–th derivative for some $N \in \mathbb{N}$, etc.

The first simple integration tool is the *formula of rectangulars*

$$Q_k^R := f(\xi_k)h_k.$$

Here the integral J_k is approximated by the (signed) area Q_k^R of the rectangular with basis h_k and height $f(\xi_k)$ (the area is negative whenever $f(\xi_k) < 0$). The corresponding composite formula is

$$Q^R := \sum_{k=1}^{n} Q_k^R = \sum_{k=1}^{n} f(\xi_k)h_k. \tag{17.4}$$

Another integrator is based on the simple *formula of trapezoids*

$$Q_k^T := \frac{f(x_k) + f(x_{k+1})}{2}h_k,$$

in which the integral J_k is approximated by the (signed) area Q_k^T of the trapezoid with vertices at the points $(x_k, 0)$, $(x_{k+1}, 0)$, (x_{k+1}, f_{k+1}) and (x_k, f_k) (note that when $f_k f_{k+1} < 0$ the trapezoid is not convex). The composite quadrature formula of trapezoids is

$$Q^T := \sum_{k=1}^{n} Q_k^T = \sum_{k=1}^{n} \frac{f(x_k) + f(x_{k+1})}{2}h_k. \tag{17.5}$$

It is easy to see that if f is piece–wise affine in the sense that $f(x) = A_k x + B_k$ for $x \in (x_k, x_{k+1})$ then both formulae Q^R and Q^T produce the exact result

$$J = \sum_{k=1}^{n} h_k(A_k\xi_k + B_k)$$

(prove!).

In order to estimate the accuracy of the formulae Q^R and Q^T we shall use the Taylor formula for f around the point ξ_k. For this purpose we shall suppose that f has a bounded fifth derivative $f^{(5)}$ on $[a, b]$.

For $x \in [x_k, x_{k+1}]$ we have the Taylor expansion

$$f(x) = \sum_{s=0}^{4} \frac{f^{(s)}(\xi_k)}{s!}(x - \xi_k)^s + \frac{f^{(5)}(\theta_k)}{5!}(x - \xi_k)^5, \tag{17.6}$$

where the point $\theta_k = \theta_k(x)$ is between ξ_k and x.

A direct calculation shows that

$$\int_{x_k}^{x_{k+1}} (x - \xi_k)^s \mathrm{d}x = 0$$

when s is odd, and

$$\int_{x_k}^{x_{k+1}} (x - \xi_k)^s \mathrm{d}x = \frac{h_k^{s+1}}{2^s(s+1)}$$

when s is even. Now the integration of both sides of (17.6) gives

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x = f(\xi_k)h_k + \frac{f''(\xi_k)h_k^3}{24} + \frac{f^{(4)}(\xi_k)h_k^5}{1920} + O(h_k^6), \tag{17.7}$$

or

$$J_k = Q_k^R + \frac{f''(\xi_k)h_k^3}{24} + \frac{f^{(4)}(\xi_k)h_k^5}{1920} + O(h_k^6). \tag{17.8}$$

From (17.8) we may find an estimate for the absolute *local error* of the formula of rectangulars (neglecting terms of order $O(h_k^5)$) as

$$\left|J_k - Q_k^R\right| \le \frac{M_2 h_k^3}{24}, \tag{17.9}$$

where

$$M_k := \sup\left\{\left|f^{(k)}(x)\right| : x \in [a,b]\right\}, \quad k = 0,1,2,\ldots.$$

To find an estimate for the global error we sum both sides of (17.8) from $k = 1$ to $k = n$ to obtain

$$|J - Q^R| = \left|\sum_{k=1}^{n} \frac{f''(\xi_k)}{24} h_k^3\right| \le \frac{M_2}{24}\sum_{k=1}^{n} h_k^3 \le \frac{nM_2}{24} \max_{1\le k\le n} h_k^3.$$

Suppose that

$$h_1 = h_2 = \cdots = h_n = (b-a)/n.$$

Then we have

$$|J - Q^R| \le \frac{M_2(b-a)^3}{24n^2}. \tag{17.10}$$

To analyze the accuracy of the formula of trapezoids we use the Taylor formulae for f around the point ξ_k noting that $x_k = \xi_k - h_k/2$ and $x_{k+1} = \xi_k + h_k/2$, namely

$$\begin{aligned} f(x_k) &= f(\xi_k) - \frac{f'(\xi_k)h_k}{2} + \frac{f''(\xi_k)h_k^2}{8} - \frac{f'''(\xi_k)h_k^3}{48} \\ &\quad + \frac{f^{(4)}(\xi_k)h_k^4}{384} + O(h_k^5), \\ f(x_{k+1}) &= f(\xi_k) + \frac{f'(\xi_k)h_k}{2} + \frac{f''(\xi_k)h_k^2}{8} + \frac{f'''(\xi_k)h_k^3}{48} \\ &\quad + \frac{f^{(4)}(\xi_k)h_k^4}{384} + O(h_k^5). \end{aligned} \tag{17.11}$$

Adding both sides of these equalities and multiplying the result by $h_k/2$ we obtain

$$\frac{f(x_k) + f(x_{k+1})}{2}h_k = f(\xi_k)h_k + \frac{f''(\xi_k)h_k^3}{8} + \frac{f^{(4)}(\xi_k)h_k^5}{384} + O(h_k^6),$$

or

$$Q_k^T = Q_k^R + \frac{f''(\xi_k)}{8}h_k^3 + \frac{f^{(4)}(\xi_k)h_k^5}{384} + O(h_k^6).$$

Combining this equality with (17.8) we see that

$$J_k = Q_k^T - \frac{f''(\xi_k)}{12}h_k^3 - \frac{f^{(4)}(\xi_k)h_k^5}{480} + O(h_k^6). \tag{17.12}$$

Hence the absolute local error for the formula of trapezoids is (neglecting terms of order $O(h_k^5)$)

$$\left|J_k - Q_k^T\right| \leq \frac{M_2 h_k^3}{12},$$

while the global error is

$$|J - Q^T| \leq \frac{M_2(b-a)^3}{12n^2}. \tag{17.13}$$

The above relations show that the formulae of rectangulars and trapezoids are of local order 3 and of global error 2. At he same time the comparison of (17.10) and (17.13) shows that the error for the formula of trapezoids has an error which is about twice less than the error for the formula of rectangulars.

Another more accurate quadrature formula is derived as follows. The relations (17.8) and (17.12) yield

$$J_k = Q_k^S - \frac{f^{(4)}(\xi_k)h_k^5}{2880} + O(h_k^6), \tag{17.14}$$

where

$$Q_k^S := \frac{2Q_k^R + Q_k^T}{3} = (f(x_k) + 4f(\xi_k) + f(x_{k+1}))\frac{h_k}{6}$$

is the famous simple *Simpson formula.*

The global Simpson formula is

$$Q^S = \sum_{k=1}^{n}(f(x_k) + 4f(\xi_k) + f(x_{k+1}))\frac{h_k}{6}.$$

It follows from (17.14) that the absolute local error for the composite Simpson formula (neglecting terms of order $O(h_k^6)$) is

$$\left|J_k - Q_k^S\right| \leq \frac{M_4 h_k^5}{2880}.$$

Hence the Simpson formula is of global order 4. In particular the Simpson formula is exact for polynomials of order up to 3. Indeed, for such polynomials we have $M_4 = 0$.

In case of equidistant knots $x_k = x_1 + (k-1)h$ $(h := (b-a)/n)$ the global error for the Simpson formula is bounded as

$$|J - Q^S| \le \frac{M_4(b-a)^5}{2880n^4}. \qquad (17.15)$$

The global Simpson formula has a simple form for the case when $n = 2m$ and $x_k = x_1 + (k-1)h$, where $h = (b-a)/(2m)$. Applying the simple Simpson formula for the intervals $[x_k, x_{k+2}]$ we obtain

$$S = \frac{b-a}{2m}\left(f_1 + f_{2m+1} + 4\sum_{k=1}^{m} f_{2k} + 2\sum_{k=1}^{m-1} f_{2k+1}\right), \qquad (17.16)$$

where

$$f_k := f(x_k),\ k = 1, 2, \ldots, 2m+1.$$

We leave the derivation of an estimate for the global error of the quadrature formula (17.16) to the reader.

The low order formulae Q^R, Q^T and even the Simpson formula Q^S are rarely used in modern computational practice due to their relatively insufficient accuracy.

Among more exact formulae we shall mention the Gauss integration schemes. Here the integral J_k over the interval $[x_k, x_{k+1}]$ is first transformed into an integral over the interval $[-1, 1]$ by the substitution

$$x = x(t) := \frac{h_k}{2}t + \xi_k,$$

namely

$$J_k = \frac{h_k}{2}G_k,$$

where

$$G_k := \int_{-1}^{1} g(t)\mathrm{d}t,\ g(t) := f(x(t)).$$

The N–th order Gauss quadrature formula for computation of G_k is defined as

$$Q_k^G(N) := \sum_{k=1}^{N} \omega_{N,k}\, g(t_{N,k}),\ \omega_{N,k} := \frac{2}{(1-t_{N,k}^2)(P_N'(t_{N,k}))^2}.$$

Here $t_{N,k} \in (-1, 1)$ $(k = 1, 2, \ldots, N)$ are the roots of the *Legendre polynomial* P_N of N–th degree $(N = 2, 3, \ldots)$.

An important observation here is that the weights $\omega_{N,k}$ and the knots $t_{N,k}$ do not depend on the particular integrand g and hence on f.

Legendre polynomials may be defined in various ways, e.g. by the recurrent formula

$$P_{N+1}(t) = tP_N(t) + \frac{t^2-1}{N+1}\frac{\mathrm{d}P_N(t)}{\mathrm{d}t}, \quad P_0 := 1,$$

or by the *Rodriguez formula*

$$P_N(t) = \frac{1}{2^N N!}\frac{\mathrm{d}^N}{\mathrm{d}x^N}(x^2-1)^N, \ N = 0, 1, 2, \ldots, \ P_0 := 1.$$

For example, the first six Legendre polynomials are

$$\begin{aligned} P_0(t) &= 1, \\ P_1(t) &= t, \\ P_2(t) &= \frac{3t^2-1}{2}, \\ P_3(t) &= \frac{5t^3-3t}{2}, \\ P_4(t) &= \frac{35t^4-30t^2+3}{8}, \\ P_5(t) &= \frac{63t^5-70t^3+15t}{8}. \end{aligned}$$

Now the reader may easily calculate the knots $t_{N,k}$ and the coefficients g_k for $N = 2, 3, 4, 5, 6$. In particular, we have

$$\begin{aligned} t_{2,1} &= -\frac{1}{\sqrt{3}} \approx -0.577350, \ g_{2,1} = 1, \\ t_{2,2} &= \frac{1}{\sqrt{3}} \approx 0.577350, \ g_{2,2} = 1 \end{aligned}$$

for $N = 2$ and

$$\begin{aligned} t_{3,1} &= -\sqrt{\frac{3}{5}} \approx -0.774597, \ g_{3,1} = \frac{5}{9} \approx 0.555556, \\ t_{3,2} &= 0, \ g_{3,2} = \frac{8}{9} \approx 0.888889, \\ t_{3,3} &= \sqrt{\frac{3}{5}} \approx 0.774597, \ g_{3,3} = \frac{5}{9} \approx 0.555556 \end{aligned}$$

for $N = 3$.

The absolute local error for the Gauss N–th order formula is

$$\left| J_k - \frac{h_k}{2}Q_k^G(N) \right| \le C_N M_{2N} h_k^{2N+1},$$

where

$$C_N := \frac{(N!)^4}{(2N+1)((2N)!)^3}.$$

Hence the Gauss formula, being of local order $2N+1$, is very exact if M_{2N} is not growing too fast with N. Indeed, the error is bounded by a multiple of h_k^{2N+1} which is small when h_k is small. Moreover, the coefficients C_N decrease very fast, e.g.

$$\begin{aligned}
C_2 &= \frac{1}{4320} \approx 2.3 \times 10^{-4}, \\
C_3 &= \frac{1}{2016000} \approx 5.0 \times 10^{-7}, \\
C_4 &= \frac{1}{1778112000} \approx 5.6 \times 10^{-10}.
\end{aligned}$$

Due to the last reason, the Gauss formula is often implemented globally in the form

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2} \sum_{k=1}^{N} g_{N,k}\varphi(t_{N,k}).$$

17.3 Integration in machine arithmetic

When the computations are done in machine arithmetic instead of Q we obtain the quantity $\widehat{Q}$. Then the problem is to compute such an approximation $\widehat{Q}$, which satisfies the absolute

$$|\widehat{Q} - J| \le D$$

or relative

$$\frac{|\widehat{Q} - J|}{|J|} \le d, \ J \ne 0,$$

estimates. Here D and d are prescribed upper bounds for the absolute and relative errors, respectively, in the computation of the integral.

For the absolute error we have the estimates

$$\begin{aligned}
&|\widehat{Q} - J| = |\widehat{Q} - Q + Q - J| \\
&\le |\widehat{Q} - Q| + |Q - J| =: E_{\mathrm{d}} + E_{\mathrm{tr}}.
\end{aligned}$$

Here E_{d} is the total rounding error in the computation of Q in machine arithmetic and E_{tr} is the truncation error which is done in the approximation of J by Q.

As we already know, the quadrature formula is based on a division of the interval $[a,b]$ into subintervals $[x_k, x_{k+1}]$ and the use of local quadrature formulae in each subinterval. Set

$$h := \max\{h_k : k = 1, 2, \ldots, n\}.$$

The truncation error for the composite quadrature formula used satisfies the estimate

$$E_{\mathrm{tr}} \leq \widehat{E}_{\mathrm{tr}} = \widehat{E}_{\mathrm{tr}}(h) := c_1 h^m, \ h \to 0,$$

where the integer m depends on the quadrature formula.

An analysis of the total absolute error in the numerical integration is done in [27]. The estimate for this error is of the form

$$\begin{aligned} \widehat{E} &= \widehat{E}(h) = c_1(b-a)h^m + (b-a)\left(c_2 + \frac{c_3}{h} + c_4 h^{m-1}\right)\mathbf{u} \\ &=: \widehat{E}_{\mathrm{tr}}(h) + \widehat{E}_{\mathrm{d}}(h, \mathbf{u}), \end{aligned}$$

where c_1, c_2, c_3, c_4 are positive constants depending on the method of integration and on the particular function f.

In practice, the numerical integration is done by adaptive algorithms, which take into account the behavior of the function f in different parts of the interval $[a, b]$.

A locally adaptive scheme for numerical integration is proposed in [25] together with a computer program QUANC8 written in Fortran 77. A globally adaptive integration algorithm is presented in [27] together with the computer program SQGGK in Fortran 77. Reliable algorithms for numerical integration are incorporated in the interactive computer systems MATLAB, MATHEMATICA, MAPLE, GAUSS, REDUCE and others, which are intended for numerical and symbolic computations, especially suitable in engineering applications.

17.4 Multiple integrals

In this section we briefly consider the numerical calculation of multiple integrals

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_m}^{b_m} f(x_1, x_2, \ldots, x_m)\mathrm{d}x_1\mathrm{d}x_2\cdots\mathrm{d}x_m,$$

or briefly

$$\int_R f(x)\mathrm{d}x,$$

where $x = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$ and

$$R := \{x \in \mathbb{R}^m : a_k \leq x_k \leq b_k, \ k = 1, 2, \ldots, m\} \subset \mathbb{R}^m.$$

The general case of an integral

$$\int_\Omega f(x)\mathrm{d}x,$$

where $\Omega \subset \mathbb{R}^m$ is a certain bounded domain, may be reduced (at least formally) to an integral over a rectangular.

Indeed, suppose that $R \subset \mathbb{R}^m$ is the smallest rectangular containing Ω, i.e.

$$a_k := \inf\{x_k : x \in \Omega\},\ b_k := \sup\{(x_k : x \in \Omega\}, \quad k = 1, 2, \ldots, m.$$

If we denote by χ_Ω the characteristic function of Ω,

$$\chi_\Omega(x) := \begin{cases} 1, & x \in \Omega, \\ 0, & x \notin \Omega \end{cases}$$

then we have

$$\int_\Omega f(x)\mathrm{d}x = \int_R \chi_\Omega(x) f(x)\mathrm{d}x.$$

Consider first the case $m = 2$. We shall denote the integral as

$$J := \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\mathrm{d}x\mathrm{d}y = \int_{a_1}^{b_1} \mathrm{d}x \int_{a_2}^{b_2} f(x,y)\mathrm{d}y =: \int_R f(\mathbf{x})\mathrm{d}\mathbf{x}, \qquad (17.17)$$

where $\mathbf{x} = (x, y) \in \mathbb{R}^2$ and $f : R \to \mathbb{R}$ is a piece–wise continuous function. This means that the rectangular R may be represented as the disjoint union of subsets $R_1, R_2, \ldots$, with nonempty interior such that f is continuous on each set R_k.

Let $F : R \to \mathbb{R}$ be a continuous function which is piece–wise differentiable in its second argument and

$$\frac{\partial F(x,y)}{\partial y} = f(x,y),\ (x,y) \in R_k.$$

In particular, we may choose

$$F(x,y) = \int_{c_k}^{y} f(x,s)\mathrm{d}s,\ (x,y) \in R_k.$$

Setting

$$G(x) := F(x, b_2) - F(x, b_1),\ x \in [a_1, b_1],$$

we have

$$J = G(b_1) - G(a_1).$$

To compute numerically the integral (17.17) we divide the rectangular R into nm $(n, m \in \mathbb{N})$ “small” rectangulars

$$R_{k,l} := [x_k, x_{k+1}] \times [y_l \times y_{l+1}], \quad k = 1, 2, \ldots, n,\ l = 1, 2, \ldots, m,$$

where

$$a_1 = x_1 < x_2 < \cdots < x_{n+1} = b_1,\ a_2 = y_1 < y_2 < \cdots < y_{m+1} = b_2.$$

A *simple quadrature* $Q_{k,l}$ is an approximation to the integral

$$J_{k,l} := \int_{x_k}^{x_{k+1}} \int_{y_l}^{y_{l+1}} f(x,y)\mathrm{d}x\mathrm{d}y.$$

The corresponding *composite quadrature formula* is

$$Q := \sum_{k=1}^{n} \sum_{l=1}^{m} Q_{k,l} \approx J.$$

Set

$$h_k := x_{k+1} - x_k,\ \xi_k := \frac{x_k + x_{k+1}}{2},$$
$$g_l := y_{l+1} - y_l,\ \eta_l := \frac{y_l + y_{l+1}}{2}$$

and

$$f_{\alpha,\beta} = f_{\alpha,\beta}^{k,l} := f(\xi_k + \alpha h_k/2, \eta_l + \beta g_l/2), \quad \alpha, \beta = 0, \pm 1.$$

Then a simple quadrature is given by

$$Q_{k,l}^{(1)} := (2f_{0,0} + f_{1,0} + f_{0,1} + f_{-1,0} + f_{0,-1})\,\frac{h_k g_l}{6}.$$

A more accurate (Simpson type) simple quadrature is

$$Q_{k,l}^{(2)} := (16 f_{0,0} + 4(f_{1,0} + f_{0,1} + f_{-1,0} + f_{0,-1}) + f_{1,1} + f_{1,-1} + f_{-1,1} + f_{-1,-1})\,\frac{h_k g_l}{36}.$$

Consider finally the triple integral

$$J = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \int_{a_3}^{b_3} f(x,y,z)\mathrm{d}x\mathrm{d}y\mathrm{d}z.$$

Here we divide the parallelepiped

$$R = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \subset \mathbb{R}^3$$

into nmp $(n, m, p \in \mathbb{N})$ smaller parallelepipedae

$$R_{k,l,s} := [x_k, x_{k+1}] \times [y_l, y_{l+1}] \times [z_s, z_{s+1}]$$

using the knots $a_1 = x_1 < x_2 < \cdots < x_{n+1} = b_1$, $a_2 = y_1 < y_2 < \cdots < y_{m+1} = b_2$, $a_3 = z_1 < z_2 < \cdots < z_{p+1} = b_3$.

Here a simple quadrature $Q_{k,l,s}$ is an approximation to the integral of f over $R_{k,l,s}$. Set

$$\begin{aligned} h_k &:= x_{k+1} - x_k,\ \xi_k := \frac{x_k + x_{k+1}}{2}, \\ g_l &:= y_{l+1} - y_l,\ \eta_l := \frac{y_l + y_{l+1}}{2}, \\ d_s &:= z_{s+1} - z_s,\ \zeta_s := \frac{z_s + z_{s+1}}{2} \end{aligned}$$

and

$$f_{\alpha,\beta,\gamma} = f^{k,l,s}_{\alpha,\beta,\gamma} := f(\xi_k + \alpha h_k/2, \eta_l + \beta g_l/2, \zeta_s + \gamma d_s/2), \quad \alpha, \beta, \gamma = 0, \pm 1.$$

Then a very simple quadrature is given by

$$Q_{k,l,s} := (f_{1,0,0} + f_{0,1,0} + f_{0,0,1} + f_{-1,0,0} + f_{0,-1,0} + f_{0,0,-1}) \frac{h_k g_l d_s}{6}.$$

The reader is advised to generalize this result for N–tuple integrals, where $N > 3$.

17.5 Numerical integration by MATLAB

The numerical integration in MATLAB is done by the commands `quad`, `quad8`, `quadl`, `dblquad` and `polyint`. Note, however, that the command `quad8` is obsolete with the command `quadl` being its recommended replacement in Release 12 and higher of MATLAB.

The command `quad` evaluates integrals of scalar functions by low order method, while the commands `quad8` and `quadl` perform this task with a high order method. For example, the command

```
>> J = quad('f',a,b)
```

approximates the integral

$$J = \int_a^b f(x)\mathrm{d}x \tag{17.18}$$

with a relative error of order 10^{-3} using an adaptive recursive Simpson's rule. Here `'f'` is a string containing the name of the integrand f, or a string expression.

Example 17.5 The integral

$$J = \int_{-1}^{2} \frac{\mathrm{d}x}{2 + x^3}$$

may be computed by `quad` in the following three ways:

(i) using a string expression,

```
>> J = quad('1./(2 + x.^3)',-1,2)
```

(ii) using an inline object by the command `inline`,

```
>> f = inline('1./(2 + x.^3)'),
   J =  quad(f,-1,2)
```

(iii) using an M–file,

```
      J = quad(@fun,-1,2)
```

where `fun.m` is the M–file

```
      function y = fun(x)
      y = 1./(2 + x.^3)
```

All three approaches give

```
>> J = 1.2383
```

in `format short`.

The command

```
>> J = quad('f',a,b,tol)
```

computes the integral (17.18) with a relative error `tol` > 0. Further on,

```
>> J = quad('f',a,b,tol,trace)
```

integrates to a relative error `tol` and for `trace` $\neq 0$ traces the function evaluation with a point plot of the integrand f.

The command `quad8` is used in the same manner. Here the integral is approximated by an adaptive recursive Newton–Cotes rule. The recommended replacement `quadl` of `quad8` is applied similarly to `quad`. It evaluates the integral by using a Lobatto quadrature with a relative error of order 10^{-6} (often the actual error is much less). For more details use `help quadl`.

Example 17.6 Consider the computation of the integral

$$J = \int_0^1 x^9 \mathrm{d}x = \left.\frac{x^{10}}{10}\right|_0^1 = 0.1$$

in MATLAB. The use of the command

```
>> J1 = quad('x.^9',0,1)
```

gives

```
J1 = 0.10000012317954
```

which is a result with a relative error of order 10^{-6}. At the same time the more accurate command

```
>> J = quadl('x.^9',0,1)
```

produces the exact result

```
J = 0.10000000000000.
```

The integration functions in MATLAB make it possible to compute the integral

$$F(y) := \int_a^y f(x)\mathrm{d}x$$

as a function of y, see the next example.

Example 17.7 To solve the integral

$$F(y) = \int_1^y \frac{\exp(x)\sin(x)}{1+x^2}\mathrm{d}x$$

one has to create two M–files, namely

```
function f = ff(x)
f = exp(x).*\sin(x)./(1 + x.^2)
```

and

```
function f = F(y)
f = quadl('ff',1,y,err)
```

Here `err` is a prescribed relative error, for example `err = 1.0e-07` (we recall that this is the MATLAB notation for 10^{-7}). Now the command

```
>> fplot('F',1,4)
```

will plot the graph of the function F for $y \in [1, 4]$.

The command `dblquad` evaluates double integrals

$$D := \int_{x_1}^{x_2} \left(\int_{y_1}^{y_2} f(x, y)\,\mathrm{d}y \right) \mathrm{d}x$$

over the rectangle

$$R := \{(x, y) : x_1 \le x \le x_2,\ y_1 \le y \le y_2\}.$$

The syntax is

```
>> D = dblquad('f',x1,x2,y1,y2)
```

which guarantees a relative error of order 10^{-6}. At the same time the command

```
>> D = dblquad('f',x1,x2,y1,y2,tol)
```

uses a tolerance `tol` > 0 instead of 10^{-6}. In both cases the underlying scalar integration scheme is `quad`. The command

```
>> D = dblquad('f',x1,x2,y1,y2,tol,@quadl)
```

uses `quadl` instead of `quad` and is considered as better (when `tol` $\leq 10^{-6}$) then the command based on `quad`. This more accurate command may of course be slower.

Example 17.8 Consider the integral

$$D = \int_0^{\pi/2} \left(\int_0^1 x \cos xy \, \mathrm{d}y \right) \mathrm{d}x = 1$$

(prove this result!). The command

```
>> D1 = dblquad('x*\cos(x*y)',0,pi/2,0,1)
```

gives

```
D1 = 0.99999998799285
```

(a result with relative error of order 10^{-8}). At the same time

```
>> D = dblquad('x*\cos(x*y)',0,pi/2,0,1,10^(-15),@quadl)
      = 1.00000000000000.
```

This time the result is exact to working precision.

Analytical integration of polynomials is done by the command `polyint`. The reader is advised to use `help polyint` for more details.

17.6 Problems and exercises

Exercise 17.1 The integral

$$J_n = \int_0^1 f_n(x) \mathrm{d}x := \int_0^1 \frac{\mathrm{d}x}{1 + x^n}$$

may be computed using the definition of the integral in the sense of Newton as

$$J_n = F_n(1) - F_n(0)$$

for each integer $n \geq 0$. Here F_n is a primitive of the function f_n,

$$F_n'(x) = f_n(x) = \frac{1}{1+x^n}.$$

For example, for small n we have

$$\begin{aligned}
J_0 &= \int_0^1 \frac{\mathrm{d}x}{2} = \frac{1}{2}, \\
J_1 &= \int_0^1 \frac{\mathrm{d}x}{1+x} = \ln(1+x)|_0^1 = \ln 2 - \ln 1 = \ln 2, \\
J_2 &= \int_0^1 \frac{\mathrm{d}x}{1+x^2} = \mathrm{arctg} x|_0^1 = \frac{\pi}{4}.
\end{aligned}$$

For $n \geq 3$, however, it is better to use a handbook or the MATHEMATICA site

`http://integrals.wolfram.com/index.jsp`

(if, of course, integration is not one of our hobbies). When n increases the expressions become highly unpleasant. For $n = 10$ we shall hardly find something in the handbook (except, maybe, a recurrence formula). An alternative way is to compute symbolically the primitive for $n = 10$, and, if we are patient enough, for $n = 25$, using e.g. the interactive system MATHEMATICA.

The reader must already be aware that this is not the right way to solve this very simple numerical problem.

This example should convince us that even when a closed form solution exists for some definite integrals, it is better to solve them numerically.

Exercise 17.2 Find an estimate for the global error of the quadrature formula (17.16).

Exercise 17.3 Find an asymptotic expression (for large N) for the coefficient C_N in the error bound for the Gauss integration scheme using the Stirling formula

$$N! \approx \left(\frac{N}{e}\right)^N \sqrt{2\pi N}.$$

Chapter 18

Myths in numerical analysis

18.1 Introductory remarks

In this chapter we consider a number of myths, which are popular among users of modern computational systems and may be very misleading. We also propose a useful euristic rule.

The invention of the digital computer in the middle of XX century led to a significant change of the viewpoint on numerical methods and algorithms. Moreover, it became clear that many classical computational schemes are not suitable for implementation in finite arithmetic and in particular in floating-point machine arithmetic.

In order to organize a reliable computational procedure one has to take into account the main three factors determining the accuracy of the computed solution, listed below.

1. The properties of the machine arithmetic and in particular the rounding unit $\mathbf{u}$.

2. The properties of the computational problem and in particular its sensitivity.

3. The properties of the numerical algorithm and in particular its numerical stability.

Unfortunately, the accounting of these factors is not a common practice, especially among the users and developers of mathematical software, who are not numerical analysts. Moreover, many myths are wide spread among this category of users. These myths deserve a special consideration. Analyses of such misconceptions have been presented in [27, 11].

18.2 Thirteen myths

There are sustainable myths[1], or misconceptions, in computational practice some of which are popular even among experienced users of modern computer software. Below we describe some ot them and consider a number of instructive examples.

Myth 1 *Large errors in the computed solution are due to the total effect of a great number of small rounding errors done at each arithmetic operation performed in machine arithmetic.*

This is rarely true or is true only partially. Moreover, when performing a large number of arithmetic operations there is a tendency of their mutual compensation. If there is a large error in the computed result then most probably this is due to the performance of a small number of critical operations (or even of a single such operation), when a numerical disaster had occured. Such an operation may be the catastrophic cancellation of true digits during a subtraction of close numbers.

Example 18.1 Consider the computation of the Ouler constant $e \approx 2.72$ by the formula

$$e \approx e_n := \left(1 + \frac{1}{a_n}\right)^{a_n}, \quad a_n := 10^n,$$

for n sufficiently large. The justification of this (obviuously bad) way to compute e is that $e_n \to e$ for $n \to \infty$. In machine arithmetic with rounding unit $\mathbf{u} \approx 10^{-16}$ we obtain the good approximation e_8 with relative error

$$\frac{|e - e_8|}{e} = 10^{-8}$$

and the the catastrophic result $e_{17} = 1$ with relative error

$$\frac{|e - e_{17}|}{e} = 1 - \frac{1}{e} \approx 0.6321.$$

The reason is not in the computing of the high power with exponent 10^{17} but in the machine summation $1 + 1/a_{17} = 1 + 10^{-17}$ which produces the result 1,

$$\mathrm{d}(1 + 10^{-17}) = 1.$$

Here $\mathrm{d}(x) = \widehat{x}$ is the value of x rounded in machine arithmetic.

[1]This is not a definable concept.

There is also a very “brave” myth which reveals simple ignorance.

Myth 2 *Rounding errors are not important because they are small (a variant: because the are completely compensated).*

The believers in this myth at least have heard about rounding errors. Because some users have not. The latter ones are maybe the happier part of users. But their happiness cannot last long.

The next myth seems almost like a true statement [11].

Myth 3 *A short computation free of cancellation, overflow and underflow, should be accurate.*

The next example shows why this statement is a myth.

Example 18.2 Consider the following algorithm, which transforms $x \geq 0$ into $y \in \mathbb{R}$ by the recursive formulae

$$x_{k+1} = x_k^{1/2},\ k = 1, 2, \ldots, n,$$

and

$$\begin{aligned} x_{k+1} &= x_k^2,\ k = n+1, n+2, \ldots, 2n, \\ y &:= x_{2n+1}. \end{aligned}$$

The theoretical result is $y = x$. In practice, even for moderate values of n in the range of 50 to 100, at all available computer platforms and for all computing environments (we write this in March, 2005), the computed result will be

$$\widehat{y} = 0,\ 0 \leq x < 1,$$

and

$$\widehat{y} = 1,\ x \geq 1.$$

Thus we can achieve an arbitrarily large relative error in the result.

The next myth deserves a special attention.

Myth 4 *Subtraction of close numbers in machine arithmetic is dangerous because the relative error of subtraction is large.*

Something is true here: the relative error of subtracion is large (and even unbounded for arbitrary close numbers).

Indeed, consider the subtraction $a - b$, where

$$A = a(1 + \delta_a), \quad B = b(1 + \delta_b)$$

are two approximate close numbers ($a \neq b$), known with relative errors δ_a, δ_b, respectively, where

$$|\delta_a|, |\delta_b| \leq \delta, \quad \delta > 0.$$

Then we have the following estimate for the relative error

$$\left| \frac{A - B}{a - b} - 1 \right| = \frac{|a\delta_a - b\delta_b|}{|a - b|} \leq \delta \frac{|a| + |b|}{|a - b|}$$

and this error is reachable for, e.g. $a, b > 0$ and $\delta_a = -\delta_b = \delta$.

But subtraction of close numbers is usually done *exactly* in machine arithmetic (e.g. when $A/2 \leq B \leq 2A$) and hence the subtraction itself does not introduce any error. What happens then? What happens is that if the close numbers are approximate (which is the typical case in computer computations) then the left-most true digits are cancelled and the possible inaccuracies in the right-most digits become important. So the useful information is lost even when the subtraction itself is exact.

Of course, if the above close numbers were exact, then the computed result would also be exact. Moreover, in many situations catastrophic cancellation may be harmless. For instance, the machine operation $C + (A - B)$ is OK when

$$1 \gg \frac{|A - B|}{|C|}, \; C \neq 0.$$

So the following statement is also a myth.

Myth 5 *Cancellation in the subtraction of near numbers is always very dangerous.*

Consider now the following myths.

Myth 6 *Increasing the precision of the arithmetics (i.e., decreasing the rounding unit) always increases the accuracy of the computed result.*

Sometimes this is not true, as shown below. The sum

$$10^{65.5} + 1006 - 10^{77} + 999 - 10^{65.5} + 10^{77} = 2005$$

will be computed as 0 on most computers in single, double and extended precision.

It is true, however, that *decreasing the rounding unit decreases the known bounds on the error* of the computations, since in many of them the rounding unit $\mathbf{u}$ (or the quantity $\mathbf{u}^p$ for some $p > 0$) is a multiplier.

Myth 7 *Rounding errors are always harmful.*

Not true again. Sometimes rounding errors can (and do!) help in certain computational procedures. For example, the QR algorithm cannot start (theoretically) for certain matrices but due to rounding errors it actually starts.

Myth 8 *The final result cannot be more accurate than the intermediate results, i.e. errors do not cancel.*

A counterexample to this myth is given in [11].

Many myths are connected with the solution of linear and nonlinear equations

$$f(x) = 0,\ \ x = [x_1, x_2, \ldots, x_n]^\top \in \mathbb{R}^n,$$

where the function $f : \mathbb{R}^n \to \mathbb{R}^n$ is contiunuous.

Let $\widehat{x}$ be the solution computed in machine arithmetic. Then the quantity

$$r(\widehat{x}) := \|f(\widehat{x})\|$$

is the *residual* corresponding to $\widehat{x}$. Since the residual is scale dependent, it is preferable to work with some scaled quantity, e.g.

$$\rho(\widehat{x}) := \frac{\|f(\widehat{x})\|}{\|f\|},$$

where $\|f\|$ is the supremum of $\|f(\xi)\|$ when ξ varies over a compact containing the solution x.

The continuity of the function r and the fact that $r(x) = 0$ for the exact solution x, are the basis of many myths, some of them considered below.

Myth 9 *The accuracy of the computed solution $\widehat{x}$ can be checked by the size of the residual $r(\widehat{x})$ by the rule "the smaller the residual, the better the approximation".*

There is a close variant to Myth 9.

Myth 10 *If we have two approximate solutions then the better one corresponds to the smaller residual.*

Myth 9 and 10 are equvalent and they are equally untrue. That these myths fail for nonlinear equations is almost obvious as the next example shows.

Example 18.3 The scalar equation

$$f(x) := x^3 - 23.001x^2 + 143.022x - 121.021 = 0.$$

has a single real root $x = 1$. Let

$$\widehat{x}_1 = 0.99, \quad \widehat{x}_2 = 11.00$$

be two approximations to the solution. By the way, only the first one may be considered as an approximation. Computing the residuals

$$r(\widehat{x}_k) := |f(\widehat{x}_k)|$$

we have

$$r(\widehat{x}_1) = 1.0022, \quad r(\widehat{x}_2) = 0.1.$$

Thus the bad approximation $\widehat{x}_2$ with a relative error of 1000 percent has a 10 times less residual than the good approximation $\widehat{x}_1$ with relative error of 1 percent!

But maybe Myths 9 and 10 fail only for nonlinear equations? Unfortunately, not. They are false for linear vector equations as well!

Example 18.4 Consider the linear algebraic equation

$$Ax = b, \tag{18.1}$$

where

$$A = \begin{bmatrix} 0.2161 & 0.1441 \\ 1.2969 & 0.8648 \end{bmatrix}, \quad b = \begin{bmatrix} 0.1440 \\ 0.8642 \end{bmatrix}.$$

The approximate solution

$$\widehat{x} = \begin{bmatrix} 0.9911 \\ 0.4870 \end{bmatrix}$$

has small residual

$$r(\widehat{x}) = \|A\widehat{x} - b\| = 0.1414 \times 10^{-7}$$

and according to Myth 9 should be close to the exact one. But the exact solution is

$$x = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

and there is no true digit in $\widehat{x}$. This should be expected since the relative error here is

$$\frac{\|\widehat{x} - x\|}{\|x\|} = 0.643.$$

This example[2] is remarkable because *all* approximate solutions, whose first three decimal digits coincide with these of x, have larger residuals than $r(\widehat{x})$!

This phenomenon for linear equations is explained in [27] and can be observed even for $n = 2$. Briefly, it may be observed in equations with ill-conditioned matrix A, for which the condition number

$$\operatorname{cond}(A) = \|A\|\,\|A^{-1}\|$$

is large. However, this observation is often true for nonlinear equations as well.

Note that for linear equations sometimes the residual is taken in a normalized form as

$$\rho^0(\widehat{x}) := \frac{\|A\widehat{x} - b\|}{\|A\|\,\|\widehat{x}\|}.$$

Example 18.5 Consider the system (18.1) with

$$A = \begin{bmatrix} \varepsilon^3 & 1 \\ 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

which has a solution

$$x = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

where $\varepsilon > 0$ is a small parameter. Let

$$y = \begin{bmatrix} 0 \\ 1+\varepsilon \end{bmatrix}, \; z = \begin{bmatrix} 1/\varepsilon \\ 1 \end{bmatrix}$$

be two approximate solutions with relative errors

$$e_y = \frac{\|y - x\|}{\|x\|} = \varepsilon \ll 1, \; e_z = \frac{\|z - x\|}{\|x\|} = \frac{1}{\varepsilon} \gg 1.$$

At the same time the unscaled residual for y is $r(y) = \varepsilon\sqrt{2}$, while this residual for z is $r(z) = \varepsilon^2$ (the use of scaled residuals gives similar results). We see that for $\varepsilon \to 0$ the bad solution z has a relative error $1/\varepsilon$ tending to ∞ but its residual ε^2 is arbitrarily smaller than the residual $\varepsilon\sqrt{2}$ of the good solution y with relative error ε.

[2]due to W. Kahan

So the check by residuals is completely misleading even for linear systems with two equations. At this point we advise the reader to explain geometrically the observed phenomenon.

The theoretical explanation of the above phenomenon is as follows. In the general case, for a non-singular A, $Ax = b$ and arbitrary $\widehat{x} \neq x$, we have (in the 2-norm)

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

and these inequalities are reachable. Indeed, we have

$$A\widehat{x} - b = A\widehat{x} - Ax = A(\widehat{x} - x).$$

Let v be the normed singular vector ($\|v\| = 1$) corresponding to the maximum singular value $\|A\|$ of A. Then we have $\|Av\| = \|A\|$. Now we may consider the approximation $\widehat{x} = x + kv$, $k \neq 0$. For this choice of $\widehat{x}$ it is fulfilled

$$\|A\widehat{x} - b\| = \|A(kv)\| = |k|\,\|Av\| = |k|\,\|A\|$$

and

$$\|\widehat{x} - x\| = \|kv\| = |k|\,\|v\| = |k|.$$

Hence

$$\frac{\|A\widehat{x} - b\|}{\|\widehat{x} - x\|} = \frac{|k|\,\|A\|}{|k|} = \|A\|.$$

In a similar way, if v is the singular vector, corresponding to the minimum singular value $\|A^{-1}\|^{-1}$ of A, we may show that for $\widehat{x} = x + kv$, $k \neq 0$, the equality

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

is valid. In fact, we have used the reachable inequalities

$$\frac{1}{\|A^{-1}\|} \leq \frac{\|Ay\|}{\|y\|} \leq \|A\|, \; y \neq 0.$$

Denote by $\widehat{x}_1$ and $\widehat{x}_2$ the vectors, for which

$$\frac{1}{\|A^{-1}\|} = \frac{\|A\widehat{x}_1 - b\|}{\|\widehat{x}_1 - x\|}, \quad \frac{\|A\widehat{x}_2 - b\|}{\|\widehat{x}_2 - x\|} = \|A\|.$$

Then we have

$$\frac{e_1}{e_2} = \mathrm{cond}(A)\frac{r_1}{r_2},$$

where

$$r_k := \|A\widehat{x}_k - b\|.$$

This clearly explains why the bettter approximation can have larger residual.

So there is a bit of irony in Myths 9 and 10: the check of the accuracy of the computed solution by the size of the residual can be successfull if the equation is well-conditioned. But then the computed solution is probaly good enough so there is nothing to check. But if the equation is ill-conditioned and there is a danger of large errors in $\widehat{x}$ then the check based on the residual can be misleading.

The bad thing is that 80 percent of experienced computer users beleive in such myths. And many books on numerical analysis do not worn them.

There are some very sophisticated myths that may even be usefull sometimes. But not always – otherwise they would not be myths.

Myth 11 *A reliable way to check the accuracy of $\widehat{x}$ is to repeat the computations with double, or some other extended precision.*

Myth 11 even has a procedural variant.

Myth 12 *If, after a repeated computation with extended precision, the first several digits in the approximate solutions computed in both ways coincide, then these digits are true.*

Why statements 11 and 12 are myths is explained in detail in [27] (see also the example after Myth 6). It is true that if we work with a very small rounding unit **u** we can achieve an arbitrary number of true digits in the computed result. And it is also true that to achieve this goal we shall need an arbitrary large computing time.

Experienced computer users know that if small changes in the data lead to large changes in the result (i.e. if the computational problem is very sensitive) then the computed solution may be contaminated with large errors. Sometimes this correct observation is reversed assuming that *if, for a given set of small perturbations, the result is slightly changed, then the accuracy of the solution is satisfatory.* Thus we come to the next myth. And the reason is in the words "a given". If there was "any" instead, everything would be OK. But it is impossible to make an experiment including all possible initial data. At least because of the lack of time.

Myth 13 *If the check of the result by a repeated computation with slightly perturbed data gives a slightly perturbed result, this guarantees that the* *computational procedure is reliable and that the solution is computed with a good accuracy.*

A counterexample to this myth is given in [27]. The reason is that in very sensitive problems there are "insensitive directions", or hyperplanes (in nonlinear problems they are manifolds) along which large changes of data cause small changes in the result. At the same time a small perturbation of the data along other directions (e.g. orthogonal to the insensitive direction) can change the result dramatically.

Example 18.6 Consider the system (18.1) with

$$A = \begin{bmatrix} a+1 & a \\ a & a-1 \end{bmatrix},$$

where $a > 1$ is large. The matrix A is nonsingular since $\det(A) = -1$. Let

$$b = \begin{bmatrix} 2a \\ 2a \end{bmatrix}.$$

Then the solution is

$$x = \begin{bmatrix} 2a \\ -2a \end{bmatrix}.$$

So if we change a to $a+\delta$, where δ is arbitrary, then the relative change in both the data b and the result x will be $|\delta|/|a|$. Based on this observation only, the system looks very well conditioned with a relative condition number of order 1. But if we choose a perturbation

$$b + \delta b = \begin{bmatrix} 2a+1 \\ 2a-1 \end{bmatrix}, \quad \delta b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

the solution becomes

$$x + \delta x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \delta x = \begin{bmatrix} 1-2a \\ 1+2a \end{bmatrix}$$

So a small relative change of order $1/(2a)$ in the data caused a large relative change of order $2a$ in the result – an amplification of order $4a^2$. This indicates an ill-conditioning of the problem. The things become clear after computing the condition number of A, namely

$$\mathrm{cond}_2(A) = 4a^2 + 2 + O(a^{-2}), \ a \to \infty.$$

18.3 A useful euristic rule

Now we can ask whether it is possible to save something of Myths 11–13? The answer is yes. Or almost yes. And the next statement is not a myth but a useful euristics.

An euristics *If a check of the result by several sets of randomly chosen small perturbations in the initial data and by several machine arithmetics with different rounding units shows small perturbation in the computed result, then with a high degree of reliability we can expect that the computed solution is close to the exact one.*

Of course, we can find a counterexample to this statement as well – that is why it is an euristics and not a theorem. But the reader will hardly find such an example in his or her computational practice.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide.* SIAM, Philadelphia, PA, third edition, 1999.

[2] N. Bahvalov, *Numerical Methods: Analysis, Algebra, Ordinary Differential Equations.* Nauka, Moscow, 1975. (in Russian).

[3] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. "SLICOT - a subroutine library in systems and control theory", *Appl. Comput. Contr., Sign., Circ.*, 1:499–532, 1999.

[4] M. Coleman, *An Introduction to Partial Differential Equations with MATLAB.* Chapman & Hall/CRC Press, 2004, ISBN 1-58488-373-1.

[5] J. Demmel, "On condition numbers and the distance to the nearest ill-posed problem", *Numer. Math.*, 51:251–289, 1987.

[6] D. Duffy, *Advanced Engineering Mathematics with MATLAB.* Chapman & Hall/CRC Press, 2003, ISBN 1-58488-349-9.

[7] J. Eaton, *GNUN Octave. Version 2.0.13.* Network Theory Ltd., Bristol, 1997, ISBN 0-954-1617-2-6.

[8] G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations.* Prentice Hall, Englewood Cliffs, N.J., 1977.

[9] P. Fuhrmann, *A Polynomial Approach to Linear Algebra.* Springer-Verlag, Berlin, 1996.

[10] G. Golub and C. Van Loan, *Matrix Computations.* The Johns Hopkins University Press, Baltimore, third edition, 1996.

[11] N. Higham, *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, USA, second edition, 2002.

[12] N. Higham, M. Konstantinov, V. Mehrmann, and P. Petkov, "The sensitivity of computational control problems", *IEEE Control Syst. Magazine*, 24:28–43, 2004.

[13] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985.* IEEE, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

[14] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parametric and State Estimation, Robust Control and Robotics.* Springer-Verlag, Berlin, 2001, ISBN 1-85233-219-0.

[15] B. Kagström and A. Ruhe, "An Algorithm for numerical computation of the Jordan normal form of a complex matrix", *ACM Trans. Math. Software*, 6:398–419 (1980); Algorithm 560 JNF. *ibid*, 510–523.

[16] M. Konstantinov, D. Gu, V. Mehrmann, and P. Petkov, *Perturbation Theory for Matrix Equations.* North-Holland, Amsterdam, 2003, ISBN 0-444-51315-9.

[17] M. Konstantinov and P. Petkov. "Effects of finite arithmetic in numerical computations", *Topics Contemp. Diff. Geometry, Complex Anal. Math. Physics (St. Dimiev and K. Sekigawa, Eds.)*, World Sci. Publ. Co., New Jersey, 2007, pp. 146–157, ISBN 10 981-270-790-5.

[18] M. Konstantinov, P. Petkov, and D. Gu, "Improved perturbation bounds for general quadratic matrix equations", *Numer. Func. Anal. Optim.*, 20:717–736, 1999.

[19] M. Konstantinov, P. Petkov, and Z. Gancheva, "Thirteen myths in numerical analysis", Proc. 34-th Spring Conf. of UBM, Borovetz, 2005, Publ. House Bulg. Acad. Sci, Sofia, 2005.

[20] M. Konstantinov, P. Petkov, and N. Hristov, *Matrix Computations by the Interactive System SYSLAB.* Univ. of Arch., Civil Engr. and Geodesy, Sofia, 2002 (in Bulgarian).

[21] The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Mass, 01760. *MATLAB Version 6.5.0.180913a (R13)*, 2002.

[22] E. Pärt-Enander, A. Sjöberg, B. Melin, and P. Isaksson, *The MATLAB Handbook*, Addison-Wesley, Harlow (England), 1996, ISBN 0-201-87757-0.

[23] P. Petkov, N. Christov, and M. Konstantinov, *Computational Methods for Linear Control Systems.* Prentice-Hall, Hemel Hempstead, 1991, ISBN 0-13-161803-2.

[24] P. Petkov and N. Hristov, *Analysis and Design of Linear Control Systems using SYSLAB.* Tehnika, Sofia, 1993, ISBN 954-03-0277-3 (in Bulgarian).

[25] R. Piessens, E. de Doncker-Kapenga, C. Uberhuber, and D. Kahane, *QUADPACK: A Subroutine Package for Automatic Integration.* Springer-Verlag, New York, 1983.

[26] POLYX, *The Polynomial Toolbox, Version 2.5* Polyx Ltd, Prague, Czech Republic, 2002.

[27] N. Vulchanov and M. Konstantinov. *Modern Mathematical Methods for Computer Calculations. Part I: Foundations of Numerical Computations. Numerical Differentiation and Integration.* Studies in Math. Sci., Bulgarian Inst. Anal. Res., vol. 1, Sofia, 1996, ISBN 954-8949-01-6 (in Bulgarian)

[28] J. Wilkinson, *The Algebraic Eigenvalue Problem.* Oxford University Press, Oxford, 1965.