Refresher course in maths and a project on numerical modeling done in twos

Course number: MAP 502

Master of Science and Technology STEEM
Ecole Polytechnique, Autumn 2021

Samuel Amstutz[1] **and Thomas Wick**[1,2]
[1] Ecole Polytechnique, Centre de Mathématiques Appliquées,
91128 Palaiseau, France
[2] Leibniz University Hannover, Institute of Applied Mathematics,
Welfengarten 1, 30167 Hannover, Germany
`https://thomaswick.org`

Organization of this class

Class number	Date	Topic	Chapter	lecturer
1	24/09	parts I, II	1+2+3+4+5+6+9	W+A
2	01/10	part II,III	8+10+11	W+A
3	08/10	part III	12	W+A
4	15/10	part IV	13+14	W+A
5	22/10	part IV	16	W+A
6	05/11	mid-term exam		W+A
7	12/11	projects		W+A
8	19/11	projects		W+A
9	26/11	projects		W+A
10	17/12	presentations (exam)		W+A

Preface

These lecture notes have been written since September 2016 for various purposes. Originally, they served as handout for a numerical modeling project within the STEEM (Energy Environment: Science Technology and Management Master) program. Later in 2018, this class was redesigned as a refresher class in maths plus the numerical modeling project. Third, Part I-III of these notes serve currently as preparation for admission in Numerics 1 for foreign Master students at the Leibniz University Hannover, Germany. Consequently, by nature, these notes address an audience that is international and interdisciplinary with participants not necessarily having a previous Bachelor degree in mathematics.

Content-wise, we strive the motivation for scientific computing, floating point numbers, linear algebra (pure and numerics), analysis (some pure elements and numerics), up to Fourier series. Moreover, for the sake of a state-of-the-art preparation of the students for their further studies and career, we dedicate our emphasis to differential equations in the final part. Parts that are not relevant for the mid-term evaluation in the STEEM program are indicated as *complement*. However, some of these parts might become relevant for the later numerical projects. In these projects, the tasks are threefold: modeling, algorithmic design, and implementation. Moreover, we consider it as important to draw interpretations of the obtained results and provide measures how to build confidence into numerical findings such intuition, error analysis, convergence analysis, and comparison to manufactured solutions. These projects are partially purely academic, but modern directions such as regression problems and neural networks can be chosen as well.

Specifically, with respect to the lastly mentioned applications and the impact of mathematics reaching out in science, technical developments, economics, and the society, it is our sincere aim to not only carry over the beauty of mathematics, but also to show its usefulness in other fields.

In case of questions, please do not hesitate to contact us!

Paris and Hannover
December 2021

Samuel Amstutz
Thomas Wick

Contents

I Introduction 11

1 Guiding questions in numerical modeling 13
- 1.1 What is numerical modeling? 13
- 1.2 Concepts in numerical mathematics 14
  - 1.2.1 Definitions 14
  - 1.2.2 Examples 15
- 1.3 First related mathematical notions 17
  - 1.3.1 Well-posedness 17
  - 1.3.2 Measuring errors 17
  - 1.3.3 Illustration of three errors (experimental, theory, numerical) with the clothesline problem 18
  - 1.3.4 Measuring distances: metrics and norms 18
  - 1.3.5 Application in numerical mathematics 20
  - 1.3.6 Notation for the complexity and convergence orders: Landau symbols 20
- 1.4 Differential equations as an example and guiding questions 21
  - 1.4.1 Definitions of differential equations 21
  - 1.4.2 Some important questions and tasks with regard to numerical concepts 22

2 Basic ingredients: sets and numbers 23
- 2.1 Sets 23
- 2.2 Classical sets of numbers 24
- 2.3 Real numbers and their usage in computers: floating-point number system 24
  - 2.3.1 Round-off errors and machine precision 24
  - 2.3.2 Influence of machine precision in numerical mathematics 25
  - 2.3.3 Stability and condition number 26
- 2.4 Complex numbers: a brief reminder 28

3 Exercises 31

II Linear algebra and related numerical notions 33

4 Linear systems and matrices 35
- 4.1 Gaussian elimination for solving linear systems 35
  - 4.1.1 General definitions 35
  - 4.1.2 Elementary row operations, row echelon systems 36
  - 4.1.3 The Gaussian elimination method 36
- 4.2 Vectors and matrices 37
  - 4.2.1 Basic definitions 37
  - 4.2.2 Operations on matrices 38
  - 4.2.3 Matrix of a linear system 41
  - 4.2.4 Invertible matrices 41

4.3 Linear subspaces of $\mathbb{K}^n$ . . . 42
4.3.1 Definitions and first properties . . . 42
4.3.2 Bases . . . 43
4.3.3 Dimension . . . 43
4.4 Determinant . . . 43
4.4.1 Expansion with respect to a row or a column . . . 44
4.4.2 Properties . . . 45
4.4.3 Determinant of a product, characterization of invertible matrices . . . 45
4.5 LU decomposition (complement) . . . 46
4.5.1 Matrix representation of Gaussian elimination . . . 46
4.5.2 LU decomposition for diagonal-dominant matrices . . . 55
4.5.3 Case of Hermitian positive definite matrices: Cholesky decomposition . . . 55
4.6 Canonical inner product, Euclidean norm, matrix norm . . . 55
4.6.1 Canonical inner product and Euclidean norm . . . 55
4.6.2 Matrix norm . . . 56

5 Diagonalization of square matrices and applications 57
5.1 Eigenvalues, eigenvectors, diagonalization . . . 57
5.1.1 Eigenvalues and eigenvectors . . . 57
5.1.2 Diagonalization . . . 58
5.1.3 Case of Hermitian matrices . . . 58
5.2 Applications . . . 58
5.3 The power method (complement) . . . 59

6 Vector spaces and elements of topology 61
6.1 Vector spaces . . . 61
6.1.1 General definition . . . 61
6.1.2 Vector spaces of finite dimension . . . 62
6.2 Basic topology . . . 62
6.2.1 Norms and balls . . . 62
6.2.2 Open sets, closed sets (complement) . . . 63
6.2.3 Closure, interior (complement) . . . 64
6.2.4 Converging sequences (complement) . . . 64
6.2.5 Equivalent norms . . . 64
6.3 Special normed spaces (complement) . . . 64
6.3.1 Inner product spaces . . . 64
6.3.2 Banach and Hilbert spaces . . . 65
6.3.3 Complement on matrix norms . . . 66

7 Linear equation systems and iterative solvers (complement) 67
7.1 Stability analysis of linear equation systems . . . 67
7.1.1 Motivation . . . 67
7.1.2 Stability analysis . . . 68
7.2 Basic iterative solvers: Richardson, Jacobi, Gauss-Seidel, conjugate gradients . . . 70
7.2.1 Fixed-point solvers: Richardson, Jacobi, Gauss-Seidel . . . 70
7.2.2 Conjugate gradients (a Krylov space method) . . . 73
7.2.3 Preconditioning . . . 78
7.2.4 Comments on other Krylov space methods such as GMRES and BiCGStab . . . 80

8 Convex sets and systems of linear inequalities **83**
8.1 Convex sets: definition and first properties . . . 83
8.2 Convex combinations and convex hull (complement) . . . 84
8.2.1 Convex combinations . . . 84
8.2.2 Convex hull . . . 84
8.3 Projection . . . 86
8.4 Cones . . . 87
8.4.1 Definitions . . . 87
8.4.2 Normal cone (complement) . . . 88
8.4.3 Polar cones (complement) . . . 88
8.5 Systems of linear inequalities, introduction to linear programming . . . 89
8.5.1 Linear inequality systems: Fourier-Motzkin elimination . . . 89
8.5.2 Conical hull and Farkas lemma (complement) . . . 90
8.5.3 Introduction to linear programming (complement) . . . 91

9 Exercises **93**

III Functions and related numerical notions **99**

10 Functions of one or several variables **101**
10.1 Basic concepts . . . 101
10.2 Differentiation . . . 101
10.2.1 Partial derivatives . . . 101
10.2.2 Fréchet derivative, Jacobian matrix . . . 102
10.2.3 Chain rule . . . 102
10.2.4 Schwarz' theorem . . . 103
10.2.5 Multiindex notation (complement) . . . 103
10.3 Classical differential operators . . . 103
10.4 Taylor expansions, Hessian matrix . . . 104
10.5 Convex functions . . . 105
10.5.1 Definitions . . . 105
10.5.2 Characterization . . . 105
10.6 Integration by parts . . . 106
10.6.1 Substitution rule . . . 106
10.6.2 Integration by parts and Green's formulae . . . 107

11 Introduction to nonlinear optimization **109**
11.1 General concepts . . . 109
11.1.1 Brief classification of optimization problems . . . 109
11.1.2 Problem setting . . . 109
11.1.3 Global and local minimizers . . . 109
11.1.4 Optimality conditions . . . 110
11.2 Convex case (complement) . . . 111
11.2.1 Specific aspects of convex optimization . . . 111
11.2.2 Towards constrained optimization . . . 111
11.3 Steepest descent method . . . 112
11.3.1 Descent methods . . . 112
11.3.2 Steepest descent . . . 112
11.3.3 Line search . . . 113
11.3.4 Improving the method (complement) . . . 114
11.4 Gradient descent with projection (complement) . . . 116

12 Solving nonlinear equations **119**
12.1 Introduction to iterative methods 119
12.2 The bisection method 120
12.3 Fixed points 121
12.3.1 Reminder: the (Banach) fixed point theorem 121
12.3.2 Attractive and repulsive fixed points 122
12.3.3 Calculating fixed points 122
12.4 The Newton method 123
12.5 Generalization to higher dimensions (complement) 124
12.5.1 Newton's method: going from $\mathbb{R}$ to higher dimensions 124
12.5.2 A basic algorithm for a residual-based Newton method 125
12.5.3 Example of the basic Newton method 125
12.5.4 Example using a Newton defect-correction scheme including line search 126
12.5.5 Newton's method in higher dimensions and the Newton-Kantorovich theorem . 127
12.5.6 Globalization of Newton's method 131
12.6 Newton's method for a coupled, nonlinear system of ODEs (complement) 134
12.7 Iteration schemes in nonlinear optimization (complement) 135
12.7.1 Linear regression 135
12.7.2 Neural networks 136

13 Interpolation and approximation **139**
13.1 Polynomial interpolation 139
13.1.1 Introduction 139
13.1.2 Existence, uniqueness, expression 139
13.1.3 Interpolation error (complement) 140
13.2 Numerical differentiation 141
13.2.1 Approximation of the first derivative 141
13.2.2 Approximation of the second derivative (complement) 142
13.3 Numerical integration (complement) 142
13.3.1 Goal 142
13.3.2 General principle 143
13.3.3 Elementary quadrature formula 143
13.3.4 Simplest rules: box, mid-point, trapezoidal, Simpson 143
13.3.5 Composite quadrature formula 144
13.3.6 Order of a quadrature formula 144
13.3.7 Interpolatory quadrature 144
13.3.8 Gauss quadrature 145

14 Trigonometric interpolation, Fourier series and Fourier transform **147**
14.1 Trigonometric interpolation: discrete Fourier transform 147
14.2 Fourier series 149
14.2.1 Series 149
14.2.2 Convergence of Fourier series 149
14.2.3 Parseval's equality 150
14.2.4 Applications 150
14.3 Fourier transform 150
14.3.1 General concept 150
14.3.2 A glimpse on the Dirac distribution 151

15 Exercises **153**

IV Differential equations 159

16 Introduction to ODEs (ordinary differential equations) 161
16.1 An introductory example . . . 161
16.2 The model ODE . . . 162
16.3 Second order linear ODE's with constant coefficients . . . 163

17 Finite differences for ODE initial-value problems 165
17.1 Problem statement of an IVP (initial value problem) . . . 165
17.2 Stiff problems . . . 165
17.3 Well-posedness . . . 166
17.4 One-step schemes . . . 166
17.4.1 The Euler method . . . 166
17.4.2 Implicit schemes . . . 167
17.5 Numerical analysis . . . 168
17.5.1 Stability . . . 169
17.5.2 Consistency / local discretization error - convergence order . . . 171
17.5.3 Convergence . . . 171
17.6 Detailed numerical tests . . . 172
17.6.1 Problem statement . . . 172
17.6.2 Discussion of the results for test 1 with a=0.25 . . . 173
17.6.3 Investigating the instability of the forward Euler method: test 3 with a=-10 . . . 174

18 Introduction to PDEs (partial differential equations) (complement) 175
18.1 Laplace equation / Poisson problem . . . 175
18.2 Mass conservation / First-order hyperbolic problem . . . 177
18.3 Elasticity (Lamé-Navier) . . . 179
18.4 The incompressible, isothermal Navier-Stokes equations (fluid mechanics) . . . 180
18.5 Three important linear PDEs . . . 180
18.5.1 Elliptic equations and Poisson problem/Laplace equation . . . 181
18.5.2 Parabolic equations and heat equation . . . 183
18.5.3 Hyperbolic equations and the wave equation . . . 184
18.6 Nonconstant coefficients. Change of the PDE types elliptic, parabolic, hyperbolic . . . 185
18.7 Boundary conditions and initial data . . . 185
18.7.1 Boundary conditions . . . 186
18.7.2 Initial conditions . . . 186
18.7.3 Example: temperature in a room . . . 186
18.8 The general definition of a differential equation . . . 187
18.9 Classifications into linear and nonlinear PDEs . . . 188
18.9.1 Linear mappings . . . 188
18.9.2 Examples . . . 189

19 Discretization and solution of PDEs (complement) 191
19.1 Finite elements for PDE boundary values problems . . . 191
19.1.1 Model problem: 1D Poisson . . . 191
19.1.2 Construction of an exact solution (only possible for simple cases!) . . . 192
19.1.3 Equivalent formulations . . . 192
19.1.4 The weak form: defining a bilinear form and a linear form . . . 196
19.1.5 The Lax-Milgram lemma (complement) . . . 197
19.1.6 Finite elements in 1D . . . 199
19.1.7 Algorithmic details (complement) . . . 210
19.1.8 Quadratic finite elements (complement) . . . 219
19.2 Numerical solution of the arising linear equation systems (complement) . . . 222

19.2.1 Basic iterative solvers . . . 222
19.2.2 Geometric multigrid methods . . . 223
19.3 Numerical tests: computational convergence analysis and solver behavior . . . 233
19.3.1 2D Poisson . . . 233
19.3.2 Numerical test: 3D . . . 234
19.3.3 Checking programming code and convergence analysis for linear and quadratic FEM . . . 234
19.3.4 Convergence analysis for 1D Poisson using linear FEM . . . 236
19.3.5 Iterative solver behavior . . . 237
19.4 Discretization of the heat equation (complement) . . . 238
19.4.1 Problem . . . 238
19.4.2 Mathematical problem statement . . . 238
19.4.3 Temporal discretization . . . 238
19.4.4 Numerical analysis: stability analysis . . . 242
19.5 Computational convergence analysis (complement) . . . 246
19.5.1 Discretization error . . . 246
19.5.2 Computationally-obtained convergence order . . . 248
19.5.3 Spatial discretization error . . . 250
19.5.4 Temporal discretization error for fixed spatial numerical solution . . . 250
19.5.5 Extrapolation to the limit . . . 250
19.5.6 Iteration error . . . 251

20 Exercises **253**

V Projects on numerical modeling in teams **257**

21 Project work **259**
21.1 Idea and formal aspects . . . 259
21.1.1 Choice of your project and contact email addresses . . . 259
21.1.2 Final exam mid December . . . 260
21.2 New projects for 2021 . . . 260
21.2.1 Image compression with SVD (singular valued decomposition) - related to eigenvalues . . . 260
21.2.2 Numerical optimization and application to regression problems . . . 260
21.2.3 Predator-prey systems . . . 263
21.2.4 Neural network approximation of ODEs . . . 264
21.2.5 Neural network for image classification . . . 265
21.2.6 Shape optimization for fluids using FreeFem++ . . . 265
21.3 List of projects 2018-2020 (can still be chosen in 2021) . . . 265
21.3.1 Direct and iterative solution of linear equation systems . . . 265
21.3.2 Numerical methods for eigenvalues . . . 266
21.3.3 Numerical methods for root-finding problems . . . 267
21.3.4 Discrete Fourier transform . . . 267
21.3.5 Numerical solution of ODEs . . . 268
21.3.6 Numerical solution of PDEs . . . 268

Bibliography **269**

Index **272**

Part I

Introduction

Chapter 1

Guiding questions in numerical modeling

In this class, we refresh basic notions of numerical linear algebra and numerical analysis in combination with numerical modeling up to differential equations. The content comprises in brief classical notions of analytical methods and linear algebra. In view of technological applications, our goal is the design and analysis of algorithms that can be used for computer simulations of prototype and complex problems.

1.1 What is numerical modeling?

Numerical modeling is a part of **scientific computing**. The latter comprises three main fields:

1. **Mathematical modeling** and analysis of physical, biological, chemical, economical, financial processes, and so forth;
2. Development of reliable and efficient **numerical methods** and algorithms and their analysis;
3. **Research software development**: Implementation of the derived algorithms

All these steps work in a **feed-back manner** and the different subtasks interact with each other. It is in fact the third above aspect, namely *software and computers*, who helped to establish this third category of science. Thus, a new branch of mathematics, *computational science*, has been established. This kind of mathematics may become experimental like experiments in physics/chemistry/biology.

A key task is the design and analysis of algorithms:

Definition 1.1 (Algorithm)**.** *An algorithm is an instruction for a schematic solution of a mathematical problem statement. The main purpose of an algorithm is to formulate a scheme that can be implemented into a computer to carry out so-called numerical simulations. Direct schemes solve the given problem up to round-off errors (for instance Gaussian elimination). Iterative schemes approximate the solution up to a certain accuracy (for instance Richardson iteration for solving linear equation systems, or fixed-point iterations). Algorithms differ in terms of* ***accuracy, robustness****, and* ***efficiency****.*

An important feedback task is to analyse (rigorously or computationally) these algorithms in order to detect shortcomings and suggest improvements. These can be nowadays on the algorithmic side (classical numerical analysis with convergence proofs) or the computational side (analysis of different discretization levels, parameter variations by just running the code again) or coding improvements (re-organization of, e.g., for-loops in a finite element program), or hardware-specific aspects (e.g., CPU computing, GPU computing).

Computational science allows us to investigate research fields that have partially not been addressable in the past. Why? On the one hand experiments are often too expensive, too far away (Mars, Moon, astronomy in general), the scales are too small (nano-scale for example); or experiments are simply too dangerous. On the other hand, mathematical theory or the explicit solution of an (ambitious) engineering problem in an analytical manner is often impossible!

Mathematics helps to formalise and structure given systems from physics, engineering, economics and other disciplines. Hereupon, established algorithms for the numerical discretization and numerical solution for known parts of the given system shall be further extended for new problems and new applications.

1.2 Concepts in numerical mathematics

1.2.1 Definitions

In [25] seven concepts that are very characteristic of numerical modeling were identified. They will be frequently encountered in the forthcoming chapters.

1. **Approximation**: For most mathematical problems, specifically those of practical relevance, the derivation of analytical solutions is difficult or even impossible. Consequently, we must be content with **approximations**, which are often obtained via numerical procedures and computer simulations.

2. **Convergence**: Whether approximations have something do to with a (unique) limit value is studied with the notion of convergence. It is qualitative in the sense that it tells us when a sequence $(a_n)_{n\in\mathbb{N}}$ admits a limit a. In numerical mathematics this limit is often the (unknown) solution we are aiming for, and the a_n are approximate solutions.

3. **Order of convergence**: The speed (or rate) of convergence tells us how fast a numerical approximation tends to limit values. While in analysis we are often merely interested in the convergence itself, in numerical mathematics we must pay attention to how long it takes until an approximate solution has sufficient accuracy. The longer a simulation takes, the more time and more energy (electricity to run the computer, air conditioning of servers, etc.) are consumed. Therefore, we are heavily interested in developing fast algorithms.

4. **Errors**: Numerical mathematics can be considered as the branch of 'mathematics of errors'. What does this mean? Numerical modeling is not wrong, inexact or non-precise! Since we cut sequences after a finite number of steps or accept sufficiently accurate solutions obtained from a software, we need to say how well this numerical solution approximates the (unknown) exact solution. In other words, we need to determine the errors, which can arise in various forms. Due its importance, we also provide the extra Section 1.3.2

5. **Error estimation**: This is one of the biggest branches in numerical mathematics and the most classical one. We need to derive error formulae to judge the outcome of a numerical simulation and to measure the difference (distance; see Section 1.3.4) between the numerical solution and the (unknown) exact solution in a certain norm or metric (see again Section 1.3.4)

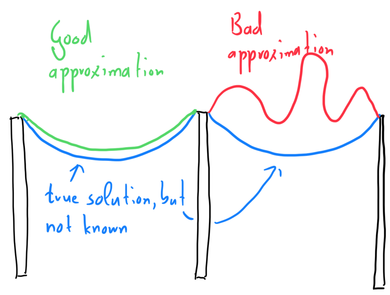
6. **Efficiency**: In general we can say, the higher the convergence order of an algorithm is, the more efficient the algorithm is. Therefore, we obtain faster the numerical solution to a given problem. But numerical efficiency is not automatically related to resource-effective computing. For instance, developing a parallel code using MPI (message passing interface), hardware-optimization (CPU,GPU), software optimizations (ordering in some optimal way for-loops, arithmetic evaluations, etc.) can further reduce computational costs.

7. **Stability**: Lastly, the robustness of algorithms and implementations with respect to parameter (model, material, numerical) variations, boundary conditions, initial conditions, uncertainties must be studied. Stability relates in the broadest sense to the third condition of Hadamard defined in Section 1.3.1.

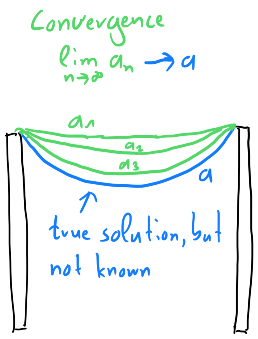
1.2.2 Examples

We illustrate the previous concepts with the help of some examples.

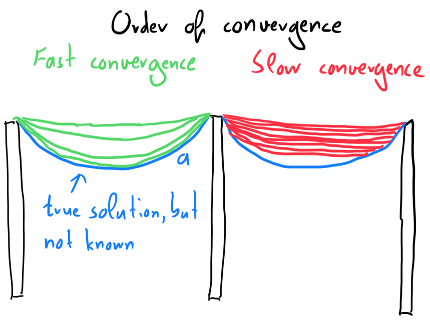
1. **Approximation**: Two approximations of the clothesline problem:



2. **Convergence**: Converging approximations of the clothesline problem:



3. **Order of convergence**: Two different speeds of convergence:



4. **Errors**: We first refer the reader to Section 1.3.3 for a comparison of experimental, theoretical and numerical situations. Second, we sharpen the sense of the influence of different errors. Not all errors are equally important and sometimes, one might try to 'optimize' an error, which has no significant influence. Let's see this in more detail. Let the errors $e_{Model}, e_{Numerics}, e_{Software}$ enter. The total error is defined as

$$e_{Total} = e_{Model} + e_{Numerics} + e_{Software}$$

Let us assume that we have the numbers $e_{Model} = 1000, e_{Numerics} = 0.001, e_{Software} = 4$, the total error is then given by

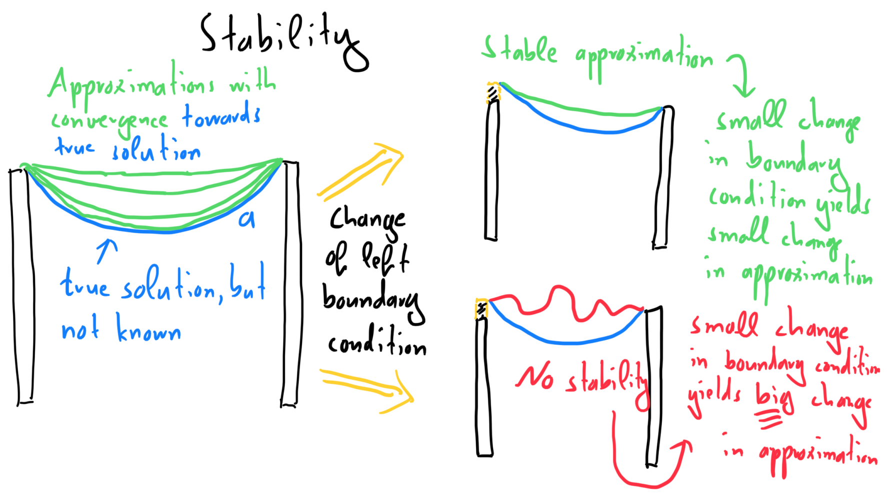
$$e_{Total} = 1000 + 0.001 + 4 = 1004.001.$$

Which error dominates? It is clearly $e_{Model} = 1000$. The relative influence is $e_{Model}/e_{Total} = 0.996$. So, the other two error sources are negligible and would not need further attention in this specific example.

5. **Error estimation**: Error estimation is the process to obtain the concrete numbers $1000, 0.001, 4$ in the previous example. Error estimates can be classified into two categories:
   - **a priori estimates** include the (unknown) exact solution u, such that $\eta := \eta(u)$, and yield qualitative convergence rates for asymptotic limits. They can be derived before (thus a priori) the approximation is known.
   - **a posteriori error estimates** are of the form $\eta := \eta(\tilde{u})$ explicitly employ the approximation $\tilde{u}$ and therefore yield quantitative information with computable majorants (i.e., bounds) and can be further utilized to design **adaptive schemes**.

6. **Efficiency**: is more or less self-explaining. A first answer is to look at CPU or wall time: how many seconds, minutes, weeks, months does a program need to terminate and yield a result? A second answer is to study 'iteration numbers' or arithmetic operations. The latter are often given in terms of the big O notation. Having a linear equation system $Ax = b$ with $A \in \mathbb{R}^{n\times n}$ and $O(n^3)$ complexity means that we need n^3 (cubic in unknowns n) arithmetic operations to calculate the result. For instance, for $n = 100$, we need around 1 000 000 operations. Having another algorithm (yielding the same result of $Ax = b$) with only $O(n)$ operations, means that we only need around 100 operations, which is a great difference. The development of efficient solvers for large linear equations systems is consequently a big branch in numerics and scientific computing.

7. **Stability**: We finally come back to the clothesline problem and change a bit the left boundary condition:


Stability
Approximations with convergence towards true solution
a
true solution, but not known
Change of left boundary condition
Stable approximation
small change in boundary condition yields small change in approximation
No stability
small change in boundary condition yields big change in approximation

1.3 First related mathematical notions

1.3.1 Well-posedness

The concept of well-posedness, as introduced by Hadamard, is very general and in fact very simple:

- The problem under consideration has a solution;
- This solution is unique;
- The solution depends continuously on the problem data.

The first condition is immediately clear. The second condition is also obvious but often difficult to meet - and in fact many physical processes do not have unique solutions. The last condition says that if a variation of the input data (right hand side, boundary values, initial conditions) is small, then also the (unique) solution should only vary a little.

Remark 1.1. *Problems in which one of the three above conditions is violated are **ill-posed**.*

1.3.2 Measuring errors

Going ahead, we implement an algorithm in a software (for instance in Matlab/Octave, Python, Fortran, C++, Java) using a computer. In the end, several error sources need to be addressed.

1. The set of numbers that can be represented in a computer is finite, therefore a numerical calculation is limited by machine precision, which results in **round-off errors**.
2. The memory of a computer (or cluster) is finite and thus functions and equations can only be represented through approximations. Thus, continuous information has to be represented through discrete information, which results in investigating so-called **discretization errors** or more generally **approximation errors**.
3. **Interpolation errors** appear when complicated functions are interpolated to simpler functions.
4. All further simplifications of a numerical algorithm (in order to solve the discrete problem), with the final goal to reduce the computational time, result into **iteration / truncation errors**. One example is the stopping criterion, which decides after how many steps an iteration is stopped. These can be further divided into **linear** and **nonlinear** iteration errors.
5. **Regularization errors** appear when ill-posed models are modified in order to become well-posed.
6. **Homogenization errors** arise when full-scale models are reduced to larger scale (homogenized) such that they are computationally simpler to solve.
7. **Implementation errors**, better known as **bugs**, appear when mistakes are made implementing algorithms into a software. These errors can simply cause the program to abort. Then it is more or less simple to find the place with the help of a debugger. But there are also subtle errors, which cannot be easily found when the program runs nonetheless, but showing strange results or behaviors.
8. **Model errors**: In order to make a ‘quick guess’ of a possible solution and to start the development of an algorithm aimed at addressing at a later stage a difficult problem, often complicated (nonlinear) equations are reduced to simple (in most cases linear) versions, which results in the so-called **model error**.
9. **Data errors and uncertainties**: the data (e.g., input data, boundary conditions, parameters) are obtained from experimental measurements and may be inaccurate themselves.

10. **Inaccurate experimental setups** that yield wrong reference solutions to which numerical solutions should be compared with.

It is very important to understand that we **never can avoid** all these errors. The important aspect is to **control** these errors and to provide answers if these errors are sufficiently big to influence the interpretation of numerical simulations or if they can be assumed to be small. A big branch of numerical mathematics is to derive error estimates that allow to predict about the size of arising errors.

1.3.3 Illustration of three errors (experimental, theory, numerical) with the clothesline problem

You need to understand a bit of French for the words, but the figure might be self-explaining.

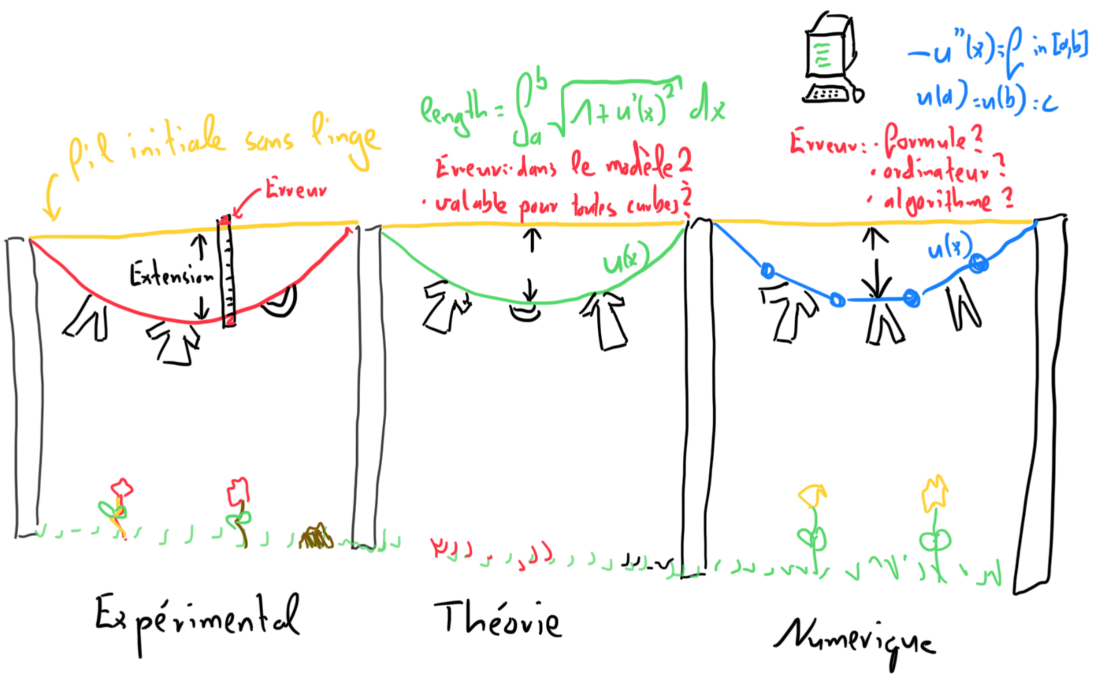


Figure 1.1: Errors in experiments, theory and numerical modeling.

1.3.4 Measuring distances: metrics and norms

To determine the accuracy and quantify the previous errors, we need to frame our problem statements in appropriate spaces that allow us to measure distances. The distance we are interested in is usually between the 'exact' solution u and its numerical approximation $\tilde{u}$, i.e., it is related to the difference

$$u - \tilde{u}.$$

However, this expression is nearly useless since it can be negative (what does a negative distance mean?) and u and $\tilde{u}$ might be vectors. Consequently, we need something like

$$\|u - \tilde{u}\|.$$

In order to define $\|\cdot\|$, we first introduce metric spaces. A metric space is a set X with a metric on it. This metric associates with any pair of elements (i.e., points), say $a, b \in X$, a distance. The concept of distance is more general than that of norm, in that it does not require the structure of vector space (in particular the difference need not be defined).

Definition 1.2 (Metric space). *A metric space is a pair (X, d) where X is a set and d is a metric (or distance) on X. The function d is defined on $X \times X$, where $\times$ denotes the Cartesian product of sets (whose definition is recalled in the next chapter). For all $x, y, z \in X$, it is assumed (axioms defining a distance):*

1. *d is real-valued, finite and nonnegative;*
2. *$d(x, y) = 0$ if and only if $x = y$;*
3. *$d(x, y) = d(y, x)$;*
4. *$d(x, y) \leq d(x, z) + d(z, y)$.*

Example 1.1. *On the real line $\mathbb{R}$ the usual metric is defined by*

$$d(x, y) = |x - y|.$$

Example 1.2. *On the surface of the earth (which is not a vector space but a manifold) we can define the geodesic distance as the length of the shortest path joining two given points.*

In numerical analysis, the framework is usually that of vector spaces and the distances of interest are typically induced by norms.

Definition 1.3 (Normed space). *A normed space X is a vector space with a norm. A norm on a real or complex vector space X is a real-valued function on X whose value at $x \in X$ is denoted by $\|x\|$ and which has the following properties:*

1. $\|x\| \geq 0$
2. $\|x\| = 0 \Leftrightarrow x = 0$
3. $\|\alpha x\| = |\alpha| \|x\|$
4. $\|x + y\| \leq \|x\| + \|y\|$

where $x, y \in X$ and α is any scalar from the underlying field $\mathbb{R}$ or $\mathbb{C}$.

Reminders on vector spaces and complements on norms will be given in chapter 6.

Definition 1.4 (Norm vs. metric). *A norm on X induces a metric d on X, which is defined by*

$$d(x, y) = \|x - y\|.$$

With the help of these definitions it is easy to infer that indeed a norm induces a metric. Therefore, normed spaces are in particular metric spaces. It holds:

Proposition 1.1. *The norm is continuous, that is $x \mapsto \|x\|$ is a continuous mapping of $(X, \|\cdot\|)$ into $\mathbb{R}$.*

Distances, induced or not by a norm, can now be employed to study the accuracy of mathematical models and their numerical approximations. Consequently, we are now able to give a meaning to the distance

$$d(u, \tilde{u})$$

between the 'exact' solution u and its numerical approximation $\tilde{u}$.

We finally add an important refinement when errors are evaluated through a norm:

Definition 1.5. *Let $\tilde{x} \in X$ be an approximation of $x \in X$. With $\|e\| = \|\tilde{x} - x\|$ we denote the* ***absolute error****, and with $\|e\|/\|x\|$ we denote the* ***relative error****.*

In most cases, relative errors are of greater importance. For instance, an error of $200m$ is small if we measure the distance between amphi Painlevé and the moon, but $200m$ are big if we measure the distance between amphi Painlevé and the CMAP corridor.

1.3.5 Application in numerical mathematics

In numerical mathematics, the key task is often to design appropriate norms for measuring the distance between a numerical approximation u_h and the continuous (exact) solution u. These norms give quantitative information about the error for deciding when an approximation is sufficiently close to our sought solution:

$$\|u - u_h\| \to 0 \quad (h \to 0),$$

where h indicates the approximation parameter. In particular, iteration error measurements in certain norms allow to decide when numerical algorithms can be stopped:

$$\|u - u_l\| \to 0 \quad (l \to \infty),$$

where l indicates the iteration index. Very often, we have a mixture of situations in which h, representing typically a discretization parameter, and l, the number of iterations, appear simultaneously.

A natural question is whether different norms yield similar answers. Here, one must distinguish between finite-dimensional vector spaces and infinite-dimensional spaces. The latter results into the branch of functional analysis. Therein, norms are in general not equivalent (a precise definition of this notion is given in chapter 6). Consequently, it does matter which norm is adopted. In finite-dimensional spaces, it can be shown, however, that norms are indeed equivalent. We emphasize that we speak here of the vector space in which the exact solution lives, even if the approximate one is sought in a finite-dimensional subspace. Dealing with two equivalent norms implies that convergence for one norm is equivalent to convergence for the other one. However, of course, both errors are not equal, therefore in quantitative error analysis the choice of the norm matters even for intrinsically finite-dimensional problems.

1.3.6 Notation for the complexity and convergence orders: Landau symbols

Definition 1.6 (Landau symbols)**.** (i) *Let $g(n)$ a function with $g \to \infty$ for $n \to \infty$. Then $f \in O(g)$ if and only if when*

$$\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$$

and $f \in o(g)$ if and only if

$$\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0.$$

(ii) *Let $g(h)$ a function with $g(h) \to 0$ for $h \to 0$. As before, we define $f \in O(g)$ and $f \in o(g)$:*

$$\limsup_{h\to 0} \left|\frac{f(h)}{g(h)}\right| < \infty \quad \Leftrightarrow \quad f \in O(g),$$

and

$$\lim_{h\to 0} \left|\frac{f(h)}{g(h)}\right| = 0 \quad \Leftrightarrow \quad f \in o(g).$$

(iii) *Specifically:*

$$\limsup_{h\to 0} |f(h)| < \infty \quad \Leftrightarrow \quad f \in O(1),$$

and

$$\lim_{h\to 0} |f(h)| = 0 \quad \Leftrightarrow \quad f \in o(1).$$

Often, the notation $f = O(g)$ is used rather than $f \in O(g)$ and similarily $f = o(g)$ rather than $f \in o(g)$.

Example 1.3. *Seven examples:*

1. $\frac{1}{x} = \mathrm{o}(\frac{1}{x^2}) \quad (x \to 0)$,
2. $\frac{1}{x^2} = \mathrm{o}(\frac{1}{x}) \quad (|x| \to \infty)$,

3. *$e^{-x} = \mathrm{o}(x^{-26}) \quad (x \to \infty)$.*
4. *Let $\varepsilon \to 0$ and $h \to 0$. We write*
$$h = o(\varepsilon)$$
when
$$\frac{h}{\varepsilon} \to 0 \quad \text{for } h \to 0, \quad \varepsilon \to 0,$$
which means that h tends faster to 0 than ε.
5. *Let us assume that we have the error estimate (see sections on the numerical analysis)*
$$\|y(t_n) - y_n\|_2 = O(k).$$
Here the O notation means nothing else than
$$\frac{\|y(t_n) - y_n\|_2}{k} \to C \quad \text{for } k \to 0.$$
6. *Let*
$$c_1 k^1 + c_2 k^2 + c_3 k^3 + \dots$$
be a development in powers of k. We can shortly write:
$$c_1 k^1 + O(k^2).$$
7. *Complexity of algorithms, e.g., LR*
$$O(n^3) \quad \text{for} \quad n \to \infty.$$

Here the fraction converges to a constant C (and not necessarily 0!), which illustrates that $O(\cdot)$ convergence is weaker than $o(\cdot)$ convergence. On the other hand, this should not yield the wrong conclusion that $\|y(t_n) - y_n\|_2$ may not tend to zero. Since $k \to 0$, also $\|y(t_n) - y_n\|_2 \to 0$ must hold necessarily.

1.4 Differential equations as an example and guiding questions

As an example of the previously mentioned concepts, we consider differential equations (later in detail in Part IV), which arise in numerous applications in science and engineering and are well-known in continuum mechanics.

1.4.1 Definitions of differential equations

Let us first roughly define the meaning of a differential equation:

Definition 1.7. *A differential equation is a mathematical equation that relates the function with certain of its derivatives.*

Differential equations can be split into two classes:

- **Ordinary differential equations**. An ordinary differential equation (ODE) is an equation (or equation system) involving an unknown function of one independent variable and certain of its derivatives.
- **Partial differential equations**. A partial differential equation (PDE) is an equation (or equation system) involving an unknown function of two or more variables and certain of its partial derivatives.

1.4.2 Some important questions and tasks with regard to numerical concepts

1. What are the underlying physics?

2. What is the mathematical model? Which kind of differential equation are we dealing with? Linear diff. eq.? Coupled system? Nonlinearities? Types of coupling? Order?

3. What discretization scheme is appropriate? Physics-based discretization, i.e., are the physics conserved after discretization in time and/or space?

4. Design of principle algorithms

5. Can we prove that these algorithms really work?

6. Accuracy, efficiency, robustness of algorithms

7. Error-controlled adaptivity or model-order reduction (e.g., with the help of SVD - singular value decomposition; computation of eigenvalues)

8. Linear and/or nonlinear solution schemes? Direct, iterative solution? Multigrid schemes? Computational cost?

9. Analyzing the numerical results: is the numerical solution correct? What does it mean 'correct'? Is it close enough (appropriate norm!) to some known solution? Comparison with experimental results possible? Comparison with analytical or manufactured solutions u and the numerical solution u_{kh}:
$$\|u - u_{kh}\| \to 0 \quad \text{for } h, k \to 0,$$
where k and h are the temporal and spatial discretization parameters, respectively. For an example, we refer to Section 19.3.

10. Advanced postprocessing techniques: quantities of interest rather global norms (see e.g., [30])? Plotting graphical solutions (gnuplot, vtk, ...)? Computational convergence analysis?

The forthcoming sections of these notes aim at refreshing (and perhaps complementing) the mathematical notions needed to address such questions. Applications will mainly be seen through the projects and further courses.

Chapter 2

Basic ingredients: sets and numbers

2.1 Sets

Set theory relies on a series of axioms, which are usually considered as the foundations of mathematics. It includes in particular the definition of natural numbers. This is a very abstract theory, but fortunately it is not needed to know it in details for a practical use of mathematics (rather advanced mathematics existed far before the axiomatic construction). We gather below the main points to be known.

Intuitively, a set is a collection of objects (of any nature) called elements. A set can be finite or not. The empty set, which contains no element, is denoted $\emptyset$. To write the list of the elements of a set we use the notation

$$A = \{1, 2, 3\}.$$

This means that the set A is made of the numbers 1, 2 and 3. The ordering is irrelevant. To say that 1 belongs to A (or 1 is in A) and 4 does not belong to A we write

$$1 \in A, \qquad 4 \notin A.$$

Given two sets A and B, we say that A is included in B, denoted $A \subset B$, if all the elements of A belong to B:

$$A \subset B \Longleftrightarrow \forall x \in A, x \in B.$$

The symbol "$\forall$" means "for all". When $A \subset B$ and $B \subset A$ we say that A and B are equal ($A = B$).

The union of two sets A and B, denoted $A \cup B$, is the set of elements that are either in A or in B:

$$A \cup B = \{x, x \in A \text{ or } x \in B\}.$$

The intersection of two sets A and B, denoted $A \cap B$, is the set of elements that are at the same time in A and in B:

$$A \cap B = \{x, x \in A \text{ and } x \in B\}.$$

The difference of two sets A and B, denoted $A \setminus B$, is the set of elements that are in A but not in B:

$$A \setminus B = \{x, x \in A \text{ and } x \notin B\}.$$

The Cartesian product of two sets A and B, denoted $A \times B$, is the set of pairs (x, y) with $x \in A$ and $y \in B$:

$$A \times B = \{(x, y), x \in A \text{ and } y \in B\}.$$

2.2 Classical sets of numbers

Natural numbers:

$$\mathbb{N} = \{0, 1, 2, \cdots\};$$

$$\mathbb{N}^* = \mathbb{N} \setminus \{0\} = \{1, 2, \cdots\}.$$

Integers:

$$\mathbb{Z} = \{\cdots - 2, -1, 0, 1, 2, \cdots\}.$$

Rational numbers:

$$\mathbb{Q} = \left\{ \frac{p}{q}, p \in \mathbb{Z}, q \in \mathbb{N}^* \right\}.$$

Real numbers: they are the classical numbers we deal with. The set of real numbers is usually denoted by $\mathbb{R}$. It includes the rational numbers but not only. An example is $\sqrt{2}$. The set $\mathbb{R}$ is constructed by "completion" of $\mathbb{Q}$. For further definitions and derivations, we refer to classical textbooks on real-valued analysis or calculus.
Complex numbers:

$$\mathbb{C} = \{x + iy, (x, y) \in \mathbb{R} \times \mathbb{R}\}.$$

In the next sections, we recapitulate real and complex numbers. In addition, we also recall some useful things to know when dealing with numbers with the help of computers.

2.3 Real numbers and their usage in computers: floating-point number system

2.3.1 Round-off errors and machine precision

In numerical mathematics it is important to understand that real numbers cannot be represented in their full length by a computer. They are cut (i.e., rounded) according to the so-called machine precision. The **floating-point system** is based on the IEEE-754 standard set in the year 1985. It consists of floating numbers, zero, Inf and NaN.

Example 2.1. *To warm up, please have a try yourself using for instance MATLAB, Octave, Python and type in some real numbers. For instance* $1/9$ *and see what happens.*

Floating-point numbers have usually the form

$$x = (-1)^s \cdot (0.a_1 a_2 \dots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}, \quad a_1 \neq 0,$$

which exactly means

$$x = (-1)^s (a_1 \beta^{-1} + a_2 \beta^{-2} + \cdots + a_t \beta^{-t}) \beta^e = (-1)^s (a_1 \beta^{t-1} + a_2 \beta^{t-2} + \cdots + a_t \beta^0) \beta^{e-t}.$$

Therein, we have:

1. **sign**: s is either 0 or 1

2. **basis**: $\beta \geq 2$ (being an integer). Computers work with $\beta = 2$ and our usual decimal system has $\beta = 10$.

3. **mantissa**: $m = a_1 \beta^{t-1} + a_2 \beta^{t-2} + \cdots + a_t \beta^0 \in \mathbb{N}$. It has length t, which is the maximum number of digits a_i, with $0 \leq a_i \leq \beta - 1$ that can be stored

4. **exponent**: $e \in \{e_{\min}, \cdots, e_{\max}\} \subset \mathbb{Z}$

An important quantity is the so-called machine precision ϵ, which is defined as

$$\epsilon = \beta^{1-t}$$

and provides the distance between 1 and the closest floating-point number greater than 1. The machine precision depends on the operating system (i.e., your computer) and not on your software.

Example 2.2. *An example of machine (double) precision is $\epsilon = 2.2204e - 16$. This can be obtained for instance in octave with*

```
eps
```

or in python with

```
import numpy
print(numpy.finfo(float).eps)
```

2.3.2 Influence of machine precision in numerical mathematics

The previous explanations need considerations once algorithms are implemented in a software and computer simulations are carried out. Why? Often, we deal with large numbers, e.g., Young's modulus of steel is $E = 190PA = 190 \times 10^9 PA$. Here, we have numbers of order 10^9. For linear equations system due to condition numbers (see below), we also easily deal with very large numbers. Therefore it is important to know what are the biggest numbers the computer can deal with. As **overflow** we denote the situation when the absolute value of a number is greater than the greatest machine number of the computer.

On the other hand, we are concerned with **underflow**, which denotes the situation when a number unequal to zero is rounded to zero. Many numerical algorithms are requested to terminate once the distance between numerical approximation and sought solution is (very) small, e.g., $\|\tilde{x} - x\| \approx 0$. But zero is too small due to machine precision. What to choose else? We define a tolerance TOL such that

$$\|\tilde{x} - x\| < TOL.$$

In order to avoid difficulties due to machine precision we must choose $TOL \gg \epsilon$. For instance:

- we have for machine epsilon $\epsilon \approx 10^{-16}$ in double precision
- Reasonable tolerances are in the range of $TOL = 10^{-8}, ..., 10^{-12}$.

Finally, due to round-off some mathematical laws may become violated on a computer. Due to the finite representation of numbers in the computer, the mathematical law of associativity

$$a + (b + c) = (a + b) + c$$

may be violated; for instance in the case of overflow or underflow.

Example 2.3. *In the exercises, the reader is asked to test the laws of associativity. For the problem with underflow, we have the f*

Example 2.4. *In Section 12.5.3, we provide numerical solutions to Newton's method for computing a root-finding problem. Therein, the tolerance TOL must be chosen. We show in the following the interaction of ϵ and TOL. For a classical choice $TOL = 1e - 12$ we obtain*

```
Iter x            f(x)
5    1.414214e+00 4.751755e-14
```

This means that we need 5 iterations to converge to a root value $|f(x| = 4.75e-14$. For $TOL = 1e-12$

```
Iter x           f(x)
4    1.414214e+00 6.156754e-07
```

This means that we need 4 iterations (the scheme is more efficient!) to converge to a root value $|f(x| = 1.53e-07$ *(the final root is less accurate!). Here, we already see the trade-off between efficiency and accuracy. Now let us choose* $TOL = 1e-16$*, a value of the order of the machine precision. Then:*

```
Iter x           f(x)
...
6502 1.414214e+00 -4.440892e-16
6503 1.414214e+00 4.440892e-16
6504 1.414214e+00 -4.440892e-16
6505 1.414214e+00 4.440892e-16
...
16368 1.414214e+00 -4.440892e-16
16369 1.414214e+00 4.440892e-16
16370 1.414214e+00 -4.440892e-16
16371 1.414214e+00 4.440892e-16
...
```

The computation did not stop (endless loop) because of round-off errors due to machine precision such that the tolerance cannot be met. This shows indeed that we must stay away with TOL *from* ϵ *and should choose* $TOL > \epsilon$.

Example 2.5. *Previously, we wrote that* ϵ *depends on the hardware. Nonetheless, the actual precision in a program can be changed. The standard machine epsilon is double precision. But we can change to* ***single precision****. The value for single precision can be obtained in python with*

```
print(numpy.finfo(numpy.float32).eps)
```

and yields $\epsilon = 1.1920929e-07$*. If we re-do the above calculations, then we even should not obtain the result for* $TOL = 1e-12$*. Indeed*

```
Iter  x           f(x)
...
12270 1.414214e+00 -1.192093e-07
12271 1.414214e+00 -1.192093e-07
12272 1.414214e+00 -1.192093e-07
12273 1.414214e+00 -1.192093e-07
...
```

we obtain an endless loop. Convergence is again achieved for $TOL > \epsilon$ *as for example* $TOL = 1e-5$*:*

```
Iter x           f(x)
4    1.414214e+00 7.152557e-07
```

Whether such low tolerances are sufficiently accurate depends on the problem and on the other hand on the memory (of course single precision needs less memory to store numbers). For instance in machine learning algorithms, often single precision is used.

2.3.3 Stability and condition number

Related to the machine precision is the stability of algorithms. Due to round-off errors, numerical algorithms can yield wrong results. How severe they are can be estimated with the so-called **condition number**.

Consider a function $f : \mathbb{R} \to \mathbb{R}$ and let us evaluate its values in the standard way:

$$x \mapsto f(x).$$

Due to round-off (or other, i.e., discretization, iterations, etc.) errors the function $f(x)$ is not evaluated at x, but at $\tilde{x} = x + \Delta x$. The question is how this input error influences the result, i.e., the output error. We denote by

$$\Delta y = f(x + \Delta x) - f(x)$$

the absolute error in the function f. If $f \in C^1(\mathbb{R})$ (one times continuously differentiable), we obtain with the mean value theorem:

$$\Delta y = f(x + \Delta x) - f(x) = f'(\xi)\Delta x$$

where $\xi \in [x, x+\Delta x]$. We observe that $f'(\xi)$ plays the role of an (absolute) amplification factor between Δy and Δx. However, in applications, the relative error is by far more important. Consequently, a representation for the relative amplification factor is:

$$\frac{\Delta y}{y} \approx f'(x)\frac{\Delta x}{f(x)} = \left(f'(x)\frac{x}{f(x)}\right)\frac{\Delta x}{x}.$$

Definition 2.1 (Condition number)**.** *The number $\kappa_{abs} = |f'(x)|$ is the absolute condition number of the problem $x \mapsto f(x)$. For $xf(x) \neq 0$ the relative condition is defined as*

$$\kappa_{rel} = \left|\frac{f'(x)x}{f(x)}\right|.$$

The condition numbers describe the amplification of input errors of a given problem in relation to the output errors. If

$$\kappa_{rel} \gg 1$$

then the problem is ill-conditioned. If $\kappa_{rel} \approx 1$ the problem is well conditioned.

Example 2.6.

1. *Addition, subtraction: $f(x, y) = x + y$ (here we evaluate in relation to x):*

$$\kappa_{f,x} = \left|\frac{x}{x+y}\right| = \left|\frac{1}{1+\frac{y}{x}}\right|.$$

If $x \approx -y$, then the (relative) condition number of the addition ($x \approx y$ for subtraction) can become arbitrary big. An example is:

$$x = -1.019, \quad y = 1.021 \quad \Rightarrow \quad x + y = 0.002.$$

Let $\tilde{x} = -1.020$ be a perturbed argument. The relative error x is very small: $|1.019-1.020|/|1.019| \leq 0.1\%$. We obtain the perturbed result

$$\tilde{x} + y = 0.001$$

and an error of 100%. The big amplification of the addition of two numbers with the same absolute value is the so-called cancellation.

2. *Multiplication: $f(x, y) = x \cdot y$. This operation is always well-conditioned:*

$$\kappa_{f,x} = \left|y\frac{x}{xy}\right| = 1.$$

3. *Division: $f(x, y) = x/y$. Same conclusion:*

$$\kappa_{f,x} = \left|\frac{1}{y}\frac{x}{\frac{x}{y}}\right| = 1, \quad \kappa_{f,y} = \left|\frac{x}{y^2}\frac{y}{\frac{x}{y}}\right| = 1.$$

4. *Square root: $f(x) = \sqrt{x}$:*

$$\kappa_{f,x} = \left|\frac{1}{2\sqrt{x}}\frac{x}{\sqrt{x}}\right| = \frac{1}{2}.$$

An input error is even reduced.

2.4 Complex numbers: a brief reminder

The set of complex numbers is denoted by $\mathbb{C}$. A complex number is written as

$$z = x + iy$$

where $x = \Re(z), y = \Im(z) \in \mathbb{R}$ are the real part and the imaginary part of z, respectively. The imaginary unit is defined as $i^2 = -1$. The main reason to define complex numbers was for solving the quadratic equation

$$x^2 + px + q = 0 \tag{2.1}$$

with coefficients $p, q \in \mathbb{R}$. For values only in $\mathbb{R}$, this quadratic equation has not always solutions. With complex numbers, we can compute as with real numbers for addition, subtraction, multiplication and division. Let us define

$$\begin{aligned} a &= x + iy \\ b &= u + iv. \end{aligned}$$

Then, the addition gives us:

$$a + b = x + iy + (u + iv) = x + u + i(y + v),$$

with $x, y, u, v \in \mathbb{R}$. The subtraction yields:

$$a - b = x + iy - (u + iv) = x - u + i(y - v).$$

The multiplication yields:

$$a * b = (x + iy) * (u + iv) = xu + ixv + iyu + i^2yv = xu - yv + i(xv + yu).$$

For the division we first introduce the conjugate

$$\bar{a} = x - iy$$

and the squared modulus

$$|a|^2 = a\bar{a} = x^2 + y^2.$$

Then, we can write the reciprocal as

$$\frac{1}{b} = \frac{\bar{b}}{b\bar{b}} = \frac{u - iv}{u^2 + v^2}$$

and the division as

$$\frac{a}{b} = \frac{a\bar{b}}{b\bar{b}} = \frac{1}{u^2 + v^2}((ux + vy) + i(uy - vx)).$$

The modulus (or absolute value) has been defined by

$$r = |a| = \sqrt{x^2 + y^2}.$$

The notation r indicates that this quantity can be identified as the radius of a with respect to the origin in the complex plane. To this end, a can be represented with the help of trigonometric functions:

$$a = r(\cos\phi + i\sin\phi),$$

where ϕ, the argument of a, denotes the angle of 0 to a on the positive real axis. With the help of Euler's formula,

$$e^{i\alpha} = \cos\alpha + i\sin\alpha$$

for $\alpha \in \mathbb{R}$, we obtain the following representation for a complex number a:

$$a = re^{i\phi}.$$

This notation is convenient to handle multiplications, powers and divisions. Indeed trigonometric rules entail

$$e^{i\phi}e^{i\theta} = e^{i(\phi+\theta)}, \qquad (e^{i\phi})^n = e^{in\phi}, \qquad \frac{e^{i\phi}}{e^{i\theta}} = e^{i(\phi-\theta)}.$$

Going back to (2.1), when the discriminant satisfies $\Delta := p^2 - 4q \leq 0$ we have the complex solutions

$$x = \frac{-p \pm i\sqrt{|\Delta|}}{2}.$$

In general a computer stores a complex number as a pair of floating-point numbers.

Chapter 3

Exercises

Concepts in numerical mathematics

Exercise 3.1. *What is the concept for measuring distances? Recapitulate the properties and provide an example of (function) set.*

Exercise 3.2. *Explain the concepts of approximation, stability and efficiency in your own words.*

Exercise 3.3. *We defined different sources of errors. Which errors are specifically related to computer simulations?*

Exercise 3.4. *Let $C[a, b]$ be the space of continuous real-valued functions defined over the closed interval $[a, b]$. Define a metric on $C[a, b]$ and verify the metric properties.*

Sets

Exercise 3.5. *Let A, B, C be three sets. Show that*

1. $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$;
2. $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$;
3. $A \cup B = A \cap B \Rightarrow A = B$.

Real numbers in computers

Exercise 3.6. *In this exercise, we give some ideas for testing how numbers are treated in a computer.*

1. *Test the law of associativity on your computer (e.g., Python, C++, Octave/MATLAB,...) taking* $a = 1.0e + 308, b = 1.1e + 308, c = -1.001e + 308$.
2. *Compute* $((1 + x) - 1)/x$ *for* $x = 1e - 15$ *(something close to machine precision) and observe if there is a cancellation of significant digits.*
3. *Find out the machine precision of your personal computer.*

Complex numbers

Exercise 3.7. *Let $a = 8 + 6i$ and $b = 1 + 2i$ be given. Compute (by hand!)*

1. $a + b$
2. $a - b$

3. $a \cdot b$
4. a/b
5. $\bar{a}$
6. $|a|$
7. a^2
8. *the two square roots of* a *(complex numbers* c *such that* $c^2 = a$*)*
9. *the exponential representation of* $z = a + 2b$
10. z^2 *using the Cartesian (real and imaginary parts) and exponential representations*

Part II

Linear algebra and related numerical notions

Chapter 4

Linear systems and matrices

The numerical solution of large linear systems is one of the two main problems of numerical linear algebra, the other problem being the computation of eigenvalues discussed in chapter 5. Linear systems occur in particular after the approximation (the so-called discretization procedure) of linear differential equations (an introduction to this very wide topic is given in part IV). Even nonlinear problems are usually solved through the construction of a sequence of linear problems, typically by the Newton method (see section 12.4). The precision of the approximation is directly related to the number of unknowns, since it is usually associated with spatial or time resolution. It is common to address linear systems with several thousands (or even millions) of unknowns.

The numerical solution of linear systems is classified in two categories. The first one gathers the so-called direct methods such as Gaussian elimination and LU or Cholesky decompositions. Such methods are exact up to round-off errors made during the calculations. The second category contains the iterative methods as we discuss in Section 7.2. They consist in the construction of a sequence of approximate solutions, converging to an exact solution only when the number of iterations tends to infinity. Some methods are mostly appropriate for systems with specific properties (symmetry, sparsity, block structure, very large systems, systems with indirect knowledge of the coefficients...), or for special computing architectures. The Gaussian elimination method is often considered as the most competitive for general "not too large" systems. The LU decomposition is a kind of reformulation of the Gaussian elimination procedure which is mostly of interest when several systems with the same matrix need to be solved. The Cholesky decomposition is a simplification of the LU decomposition for symmetric (or Hermitian) positive definite matrices. Iterative methods are to be used for very large systems, or when one is interested in a quick approximate solution.

In the following, unless otherwise specified, the letter $\mathbb{K}$ is used to designate either the set $\mathbb{R}$ of real numbers or the set $\mathbb{C}$ of complex numbers.

4.1 Gaussian elimination for solving linear systems

4.1.1 General definitions

Definition 4.1. *A linear system of m equations and n unknowns is a set of equations of form*

$$\left\{\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 & \quad (R_1) \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 & \quad (R_2) \\ \vdots & & \vdots & & & & \vdots & & \vdots & \\ \vdots & & \vdots & & & & \vdots & & \vdots & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m & \quad (R_m) \end{array}\right. . \tag{4.1}$$

The numbers $a_{ij} \in \mathbb{K}$ are the coefficients of the system. The numbers $b_i \in \mathbb{K}$ are the coefficients of the right hand side. The numbers $x_j \in \mathbb{K}$ are the unknowns of the system.

Definition 4.2. *A solution of the system* (4.1) *is an* n*-tuple* $(x_1, ..., x_n)$ *which simultaneously satisfies the* m *equations* $R_1, ..., R_m$.

Definition 4.3. *If all the coefficients* b_i *are equal to zero then the system is said to be homogeneous.*

Definition 4.4. *We say that the system is compatible (or possible) if it admits at least one solution. Otherwise we say that the system is incompatible (or impossible).*

Remark 4.1. *Every homogeneous system is compatible since* $(0, ..., 0)$ *is a solution.*

Definition 4.5. *We say that two systems are equivalent if the have the same set of solutions.*

4.1.2 Elementary row operations, row echelon systems

Proposition 4.1. *The following operations, said to be elementary, transform a linear system into an equivalent one:*

1. *swapping the rows* R_i *and* R_k*, denoted by* $R_i \leftrightarrow R_k$,
2. *multiplying the row* R_i *by some* $\alpha \neq 0$*, denoted by* $R_i \leftarrow \alpha R_i$,
3. *replacing the row* R_i *by* $R_i + \beta R_j$, $j \neq i$*, denoted by* $R_i \leftarrow R_i + \beta R_j$.

Definition 4.6. *A linear system is in row echelon form if the number of leftmost vanishing coefficients in each row is increasing, or possibly constant if this number is* $n + 1$ *(row equal to* 0*).*

There are two types of row echelon systems:

1. the last non-vanishing row is of form
$$\alpha_r x_r + ... + \alpha_n x_n = \beta \qquad (\alpha_r \neq 0), \tag{4.2}$$
2. the last non-vanishing row is of form
$$0 = \beta \qquad (\beta \neq 0).$$

The system is compatible in case 1, incompatible in case 2.

4.1.3 The Gaussian elimination method

The Gaussian elimination method transforms a linear system into a row echelon equivalent one with the help of elementary operations.

There are several steps. Let us describe the first one, which itself consists in two sets of elementary operations.

1. First we possibly swap the row R_1 with another row R_k such that $a_{k1} \neq 0$ (pivoting). We obtain an equivalent linear system of form
$$\left\{\begin{array}{ccccccccccc} a'_{11}x_1 & + & a'_{12}x_2 & + & \cdots & + & a'_{1n}x_n & = & b'_1 & & (R'_1) \\ a'_{21}x_1 & + & a'_{22}x_2 & + & \cdots & + & a'_{2n}x_n & = & b'_2 & & (R'_2) \\ \vdots & & \vdots & & & & \vdots & & \vdots & & \\ \vdots & & \vdots & & & & \vdots & & \vdots & & \\ a'_{m1}x_1 & + & a'_{m2}x_2 & + & \cdots & + & a'_{mn}x_n & = & b'_m & & (R'_m) \end{array}\right. \tag{4.3}$$
with $a'_{11} \neq 0$. This coefficient a'_{11} is called the first pivot.

2. Then we replace each row R_p, $p \geq 2$, by $R_p - \dfrac{a'_{p1}}{a'_{11}} R_1$ (elimination). This leads to an equivalent system of form

$$\left\{\begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \qquad (R_1^{(1)}) \\ & & a_{22}^{(1)}x_2 & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \qquad (R_2^{(1)}) \\ & & \vdots & & & & \vdots & & \vdots \\ & & \vdots & & & & \vdots & & \vdots \\ & & a_{m2}^{(1)}x_2 & + & \cdots & + & a_{mn}^{(1)}x_n & = & b_m^{(1)} \qquad (R_m^{(1)}) \end{array}\right. . \tag{4.4}$$

Step 2 consists in applying the above procedure to rows $R_2^{(1)}, ..., R_m^{(1)}$, obviously choosing the pivot in the second column.

We continue in this way until obtaining a full row echelon system. If this row echelon system is compatible, with last row of form (4.2), then we write the r unknowns $x_1, ..., x_r$ in terms of the other ones, going upwards from the last row to the first one (backward substitutions).

In the numerical implementation of the Gaussian elimination method, small (in absolute value / modulus compared to other coefficients) pivots are undesirable, because they may lead to significant errors in the floating point arithmetic (an illustration in given as exercise). Therefore, pivoting is often performed so as to select the largest available pivot. To improve accuracy, column swapping is generally also performed. This procedure is called total pivoting.

4.2 Vectors and matrices

4.2.1 Basic definitions

Matrices

Definition 4.7. *A matrix with coefficients in $\mathbb{K}$ of type (m, n) is a table with m rows and n columns represented as*

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

The numbers $a_{ij} \in \mathbb{K}$ are the coefficients of the matrix. Sometimes we denote $A = (a_{ij})$.

Definition 4.8. *We denote by $\mathcal{M}_{mn}(\mathbb{K})$, or $\mathbb{K}^{m \times n}$, the set of matrices with coefficients in $\mathbb{K}$ of type (m, n).*

Definition 4.9. *Two matrices $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$ and $B = (b_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$ are equal if they have the same coefficients, i.e., $a_{ij} = b_{ij}$ $\forall (i, j) \in \{1, ..., m\} \times \{1, ..., n\}$.*

Definition 4.10. *A zero matrix, denoted by 0, is a matrix with all coefficients equal to 0. Hence, if $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$, then*

$$A = 0 \Longleftrightarrow a_{ij} = 0 \ \forall (i, j) \in \{1, ..., m\} \times \{1, ..., n\}.$$

Special matrices

a) Row matrix, also called row vector

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \end{pmatrix} \in \mathcal{M}_{1n}(\mathbb{K})$$

b) Column matrix, also called column vector

$$A = \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \end{pmatrix} \in \mathcal{M}_{m1}(\mathbb{K})$$

c) Square matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \in \mathcal{M}_{nn}(\mathbb{K}) =: \mathcal{M}_n(\mathbb{K})$$

Some square matrices have themselves special features.
i) Lower triangular matrix

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix}$$

ii) Upper triangular matrix

$$A = \begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix}$$

iii) Diagonal matrix

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix} = \operatorname{diag}(a_{11}, ..., a_{nn})$$

iv) Identity matrix

$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} = \operatorname{diag}(1, ..., 1) \in \mathcal{M}_n(\mathbb{K})$$

In contrast to vectors and matrices, the elements of $\mathbb{K}$ are called scalars.

Submatrix

Definition 4.11. *A submatrix of a matrix A is a matrix obtained by removing some rows and some columns from A.*

4.2.2 Operations on matrices

Addition, multiplication by a scalar

Definition 4.12. *Let $A = (a_{ij})$ and $B = (b_{ij})$ be two matrices in $\mathcal{M}_{mn}(\mathbb{K})$. The matrix $A + B$ is defined by*

$$A + B = (a_{ij} + b_{ij}) \in \mathcal{M}_{mn}(\mathbb{K}).$$

Definition 4.13. *Let $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$ and $\alpha \in \mathbb{K}$. We define the matrix αA by*

$$\alpha A = (\alpha a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K}).$$

Proposition 4.2. *Let $A, B, C \in \mathcal{M}_{mn}(\mathbb{K})$ and $\alpha \in \mathbb{K}$. We have*

$$\begin{aligned}&A + B = B + A\\&(A + B) + C = A + (B + C)\\&\alpha(A + B) = \alpha A + \alpha B.\end{aligned}$$

Multiplication of matrices

Definition 4.14. *Let $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$ and $B = (b_{ij}) \in \mathcal{M}_{np}(\mathbb{K})$. We define the matrix $C = AB \in \mathcal{M}_{mp}(\mathbb{K})$ by $C = (c_{ij})$ with*

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Remark 4.2.
1. *Be careful with the compatibility of dimensions: the number of columns of the left matrix must be equal to the the number of rows of the second matrix.*
2. *The product AB may be defined whereas the product BA is not. Even if AB and BA are both defined, in general $AB \neq BA$. If $AB = BA$, then we say that A and B commute.*
3. *It may happen that $AB = 0$ while $A \neq 0$ and $B \neq 0$!*

Proposition 4.3. *Let A, B, C be three matrices. We have, under the condition of compatible dimensions, the associativity*

$$(AB)C = A(BC).$$

Proposition 4.4. *Let $A \in \mathcal{M}_{mn}(\mathbb{K})$. We have $AI_n = I_m A = A$.*

Distributivity

Proposition 4.5. *Let A, B, C be three matrices and $\alpha \in \mathbb{K}$. We have, under the condition of compatible dimensions,*

$$\begin{aligned}&A(B + C) = AB + AC\\&(A + B)C = AC + BC\\&\alpha(AB) = (\alpha A)B = A(\alpha B).\end{aligned}$$

Transposition

Definition 4.15. *Let $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$. The transpose of A, denoted by A^T, is the matrix*

$$A^T = (a_{ji}) \in \mathcal{M}_{nm}(\mathbb{K}).$$

Example 4.1.

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}^T = \begin{pmatrix}1 & 3\\2 & 4\end{pmatrix}$$

Definition 4.16. *A matrix $A \in \mathcal{M}_n(\mathbb{R})$ such that $A = A^T$ is called a symmetric matrix. A matrix $A \in \mathcal{M}_n(\mathbb{C})$ such that $A = \bar{A}^T$, where $\bar{A}$ is the complex conjugate matrix of A (obtained by taking the conjugate of each coefficient), is called a Hermitian matrix.*

Proposition 4.6.
1. *Let $A, B \in \mathcal{M}_{mn}(\mathbb{K})$. We have $(A + B)^T = A^T + B^T$.*
2. *Let $A \in \mathcal{M}_{mn}(\mathbb{K})$ and $\alpha \in \mathbb{K}$. We have $(\alpha A)^T = \alpha A^T$.*

	$k=0$	$k=1$	$k=2$	$k=3$
$p=0$	1			
$p=1$	1	1		
$p=2$	1	2	1	
$p=3$	1	3	3	1

Table 4.1: Binomial coefficients C_p^k for $p \leq 3$ (Pascal's triangle)

3. *Let $A \in \mathcal{M}_{mn}(\mathbb{K})$ and $B \in \mathcal{M}_{np}(\mathbb{K})$. We have $(AB)^T = B^T A^T$.*

4. *Let $A \in \mathcal{M}_{mn}(\mathbb{K})$. We have $(A^T)^T = A$.*

Definition 4.17. *Let $A \in \mathcal{M}_n(\mathbb{K})$ be a Hermitian (symmetric if $\mathbb{K} = \mathbb{R}$) matrix. We say that A is positive semi-definite if $\bar{x}^T Ax \geq 0$ for all $x \in \mathbb{K}^n$. We denote $A \geq 0$. We say that A is positive definite if $\bar{x}^T Ax > 0$ for all $x \in \mathbb{K}^n \setminus \{0\}$. We denote $A > 0$.*

Trace of a square matrix

Definition 4.18. *Let $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$. The trace of A, denoted by $\operatorname{tr} A$, is the sum of the diagonal coefficients of A, i.e.*

$$\operatorname{tr} A = \sum_{i=1}^{n} a_{ii}.$$

Proposition 4.7. 1. *Let $A, B \in \mathcal{M}_n(\mathbb{K})$. We have $\operatorname{tr}(A+B) = \operatorname{tr} A + \operatorname{tr} B$.*

2. *Let $A \in \mathcal{M}_n(\mathbb{K})$ and $\alpha \in \mathbb{K}$. We have $\operatorname{tr}(\alpha A) = \alpha \operatorname{tr} A$.*

3. *Let $A \in \mathcal{M}_{mn}(\mathbb{K})$, $B \in \mathcal{M}_{nm}(\mathbb{K})$. We have $\operatorname{tr}(AB) = \operatorname{tr}(BA)$.*

Power of a square matrix

Definition 4.19. *Let $A \in \mathcal{M}_n(\mathbb{K})$ and $p \in \mathbb{N}$. We define*

$$A^p = \underbrace{A \times A \times \cdots \times A}_{p \text{ factors}},$$

with the convention $A^0 = I_n$.

Remark 4.3. *Be careful not to confuse the matrix power with the power of the coefficients. However, if A is diagonal, then $(\operatorname{diag}(\lambda_1, ..., \lambda_n))^p = \operatorname{diag}(\lambda_1^p, ..., \lambda_n^p)$.*

For two commuting matrices, we have the following binomial formula.

Proposition 4.8. *Let $A, B \in \mathcal{M}_n(\mathbb{K})$ be two matrices $\underline{\text{such that } AB = BA}$ and p be a natural number. The binomial formula*

$$(A+B)^p = \sum_{k=0}^{p} C_p^k A^k B^{p-k}$$

holds, where $C_p^k = \binom{p}{k} = \dfrac{p!}{(p-k)!k!}$ are the binomial coefficients.

The binomial formula is formally the same as for scalars. Remind that, due to the property $C_{p+1}^{k+1} = C_p^{k+1} + C_p^k$, the binomial coefficients can be obtained from Pascal's triangle, see Table 4.1.

4.2.3 Matrix of a linear system

Let $A = (a_{ij}) \in \mathcal{M}_{mn}(\mathbb{K})$ and $b = (b_1, ..., b_m)^T \in \mathcal{M}_{m1}(\mathbb{K})$. Solving the equation $Ax = b$ of unknown $x = (x_1, ..., x_n)^T$ is equivalent to solving the linear system

$$\left\{\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}\right. . \tag{4.5}$$

We say that A is the matrix of the system (4.5).

4.2.4 Invertible matrices

Definition and first properties

Definition 4.20. *A matrix $A \in \mathcal{M}_n(\mathbb{K})$ is said to be invertible (or nonsingular, or regular) if there exists $B \in \mathcal{M}_n(\mathbb{K})$ such that $AB = BA = I_n$. In this case, B is unique and it is called the inverse matrix of A, denoted by A^{-1}.*

Proposition 4.9. *Let $A, B \in \mathcal{M}_n(\mathbb{K})$. If $AB = I_n$ <u>or</u> $BA = I_n$, then A is invertible and $B = A^{-1}$.*

Remark 4.4. *If A is invertible then $Ax = b \iff x = A^{-1}b$. The knowledge of A^{-1} permits to straightforwardly solve any linear system whose matrix is A. However, as we will see, numerically computing A^{-1} is usually much harder than solving a linear system of matrix A.*

Calculation of the inverse

Let $A \in \mathcal{M}_n(\mathbb{K})$. In order to know whether A is invertible and, if possible, calculate its inverse, the classical approach when doing the calculations by hand consists in solving the linear system $Ax = y$ of unknown $x = (x_1, ..., x_n)^T$ and right hand side $y = (y_1, ..., y_1)^T$ arbitrary. If this system admits a solution x^* without any condition on y, this means that A is invertible. From $x^* = A^{-1}y$ we infer A^{-1} by identification.

If the computations are done numerically then the above technique does not apply directly. A possible reformulation is to solve the matrix equation $AB = I_n$, of unknown $B \in \mathcal{M}_n(\mathbb{K})$. This is equivalent to solving n linear systems of matrix A and corresponding to each column of B and I_n. Even if LU / Cholesky decompositions are helpful for this, numerically inverting a large matrix is to be avoided unless it is explicitly needed. In particular, **do not invert a matrix in order to solve a linear system!** In addition to CPU time, it often happen that A be sparse and A^{-1} be full and impossible to store in memory.

Example 4.2. *Compute the inverse of the matrix*

$$A = \begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix}.$$

To do so let us solve the linear system

$$\left\{\begin{array}{ll} 3x_1 + x_2 & = y_1 \\ 5x_1 + 2x_2 & = y_2. \end{array}\right.$$

By Gaussian elimination the unique solution is found as

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2y_1 - y_2 \\ -5y_1 + 3y_2 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ -5 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

It follows that A is invertible with inverse

$$A^{-1} = \begin{pmatrix} 2 & -1 \\ -5 & 3 \end{pmatrix}.$$

The equivalent matrix formulation is

$$\begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \qquad \text{often denoted} \qquad \left(\begin{array}{cc|cc} 3 & 1 & 1 & 0 \\ 5 & 2 & 0 & 1 \end{array}\right).$$

After Gaussian elimination this is found equivalent to

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ -5 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \qquad \text{denoted} \qquad \left(\begin{array}{cc|cc} 1 & 0 & 2 & -1 \\ 0 & 1 & -5 & 3 \end{array}\right),$$

from which A^{-1} is readily obtained.

Some algebraic rules

Proposition 4.10. *1. I_n is invertible of inverse I_n.*

2. *If A is invertible then A^{-1} is invertible and $(A^{-1})^{-1} = A$.*
3. *If A, B are invertible then AB is invertible and $(AB)^{-1} = B^{-1}A^{-1}$.*
4. *If A is invertible then A^T is invertible and $(A^T)^{-1} = (A^{-1})^T =: A^{-T}$.*
5. *If $A = \mathrm{diag}(\lambda_1, ..., \lambda_n)$, then A is invertible if and only if all coefficients λ_i do not vanish. In this case $A^{-1} = \mathrm{diag}(\lambda_1^{-1}, ..., \lambda_n^{-1})$.*

4.3 Linear subspaces of $\mathbb{K}^n$

4.3.1 Definitions and first properties

Definition 4.21. *A set $F \subset \mathbb{K}^n$ is a linear subspace of $\mathbb{K}^n$ if*

1. *$0 \in F$,*
2. *$\forall v, w \in F,\ v + w \in F$,*
3. *$\forall v \in F, \forall \lambda \in \mathbb{K},\ \lambda v \in F$.*

Such sets appear naturally when solving underdetermined homogeneous linear systems, as stated below.

Proposition 4.11. *The set of solutions of a linear homogeneous system of the unknown $v = (v_1, ..., v_n) \in \mathbb{K}^n$ (here $v_1, \dots, v_n$ denote the components of the vector v) is a linear subspace of $\mathbb{K}^n$.*

In the following, elements of $\mathbb{K}^n$ are called vectors, whatever they are considered as row vectors or column vectors.

Hereafter, be careful with the number of vectors indexed by m and the corresponding number of components of each vector that is indicated by n.

Definition 4.22. *Let $w, v_1, ..., v_m$ be vectors from $\mathbb{K}^n$. Each of these vectors has n components. We say that w is a linear combination of $v_1, ..., v_m$ if there exist scalars $\lambda_1, ..., \lambda_m$ such that*

$$w = \lambda_1 v_1 + ... + \lambda_m v_m.$$

Proposition 4.12. *Let $v_1, ..., v_m$ be vectors. The set of linear combinations of $v_1, ..., v_m$ is a linear subspace of $\mathbb{K}^n$. It is called the linear subspace spanned by $v_1, ..., v_m$, it is denoted by $Span(v_1, ..., v_m)$.*

4.3.2 Bases

Definition 4.23. *A family $(v_1, ..., v_m)$ of vectors is said to be linearly independent if, for all family $(\lambda_1, ..., \lambda_m)$ of scalars,*

$$\lambda_1 v_1 + ... + \lambda_m v_m = 0 \Longrightarrow \lambda_1 = ... = \lambda_m = 0.$$

Otherwise it is said to be linearly dependent.

Definition 4.24. *Let F be a linear subspace of $\mathbb{K}^n$ and $(v_1, ..., v_m)$ be a family of vectors. If Span$(v_1, ..., v_m) = F$ we say that $(v_1, ..., v_m)$ spans F, or that it is a spanning set of F.*

Example. The vectors $(e_1, ..., e_n)$ defined by $e_i = (e_i^1, ..., e_i^n)$, $e_i^j = 1$ if $i = j$, $e_i^j = 0$ if $i \neq j$ obviously span $\mathbb{K}^n$. For instance for $n = 3$:

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Proposition 4.13. *Let F be a linear subspace of $\mathbb{K}^n$. If $(v_1, ..., v_m)$ spans F then all family of $m + 1$ vectors of F is linearly dependent.*

Corollary 4.14. *A family of $n + 1$ vectors of $\mathbb{K}^n$ is linearly dependent.*

Definition 4.25. *Let F be a linear space of $\mathbb{K}^n$. We say that a family $(v_1, ..., v_m)$ of vectors is a basis of F if it is linearly independent and a spanning set of F.*

Theorem 4.15. *All linear subspace of $\mathbb{K}^n$ different from $\{0\}$ admit a basis.*

Example. The family $(e_1, ..., e_n)$ defined above is a basis of $\mathbb{K}^n$, called canonical basis.

Proposition 4.16. *Let F be a linear subspace of $\mathbb{K}^n$ and $(v_1, ..., v_m)$ be a basis of F. Any vector x of F can be decomposed in a unique way as $x = \lambda_1 v_1 + ... + \lambda_m v_m$ with $\lambda_1, ..., \lambda_m \in \mathbb{K}$. The scalars $\lambda_1, ..., \lambda_m$ are called the coordinates of x in the basis $(v_1, ..., v_m)$.*

4.3.3 Dimension

Here is the "theorem of the dimension".

Theorem 4.17. *Let F be a linear subspace of $\mathbb{K}^n$ different from $\{0\}$. All the bases of F contain the same number of vectors.*

Definition 4.26. *Let F be a linear subspace of $\mathbb{K}^n$ different from $\{0\}$. We call dimension of F, denoted by $\dim F$, the number of vectors of any basis of F. By convention, $\dim\{0\} = 0$.*

Proposition 4.18. *We have $\dim \mathbb{K}^n = n$ and $\dim F \leq n$ for all linear subspace F of $\mathbb{K}^n$.*

Proposition 4.19. *Let F be a linear subspace of $\mathbb{K}^n$ such that $\dim F = m$.*

1. *If p vectors of F are linearly independent then $p \leq m$.*
2. *If m vectors of F are linearly independent then they form a basis of F.*
3. *If p vectors of F span F then $p \geq m$.*
4. *If m vectors of F span F then they form a basis of F.*

Corollary 4.20. *Let F be a linear subspace of $\mathbb{K}^n$ such that $\dim F = n$. Then $F = \mathbb{K}^n$.*

4.4 Determinant

The mathematical construction of the notion of determinant is a lengthy procedure. Here we only recall the main properties of the determinant of a square matrix.

4.4.1 Expansion with respect to a row or a column

The determinant of a square matrix $A \in \mathcal{M}_n(\mathbb{K})$ is a scalar $\det A \in \mathbb{K}$. We adopt a practical definition by induction on n.

We begin with the trivial case $n = 1$.

Definition 4.27. *For $A = (a) \in \mathcal{M}_1(\mathbb{K})$ we set $\det A = a$.*

We now assume that the determinant is defined for all matrix of type $(n-1, n-1)$.

Definition 4.28. *Let $A \in \mathcal{M}_n(\mathbb{K})$ et $i, j \in \{1, ..., n\}$. We call*

- *minor (i,j) of A the scalar*
$$\Delta_{ij}(A) = \det \tilde{A}$$
where the $\tilde{A}$ is the submatrix of A obtained by removing the row i and the column j;

- *cofactor (i,j) of A the scalar*
$$\mathit{Cof}_{ij}(A) = (-1)^{i+j}\Delta_{ij}(A).$$

Definition 4.29. *Let $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$. Given $i_0 \in \{1, ..., n\}$ we call expansion of the determinant of A with respect to the row i_0 the scalar*

$$R_{i_0}(A) = \sum_{j=1}^{n} a_{i_0,j}\,\mathit{Cof}_{i_0,j}(A).$$

Likewise, given $j_0 \in \{1, ..., n\}$ we call expansion of the determinant of A with respect to the column j_0 the scalar

$$C_{j_0}(A) = \sum_{i=1}^{n} a_{i,j_0}\,\mathit{Cof}_{i,j_0}(A).$$

Definition 4.30. *Let $A \in \mathcal{M}_n(\mathbb{K})$. We set*

$$\det A = R_1(A).$$

The definitions 4.27 to 4.30 define unambiguously $\det A$ for any matrix $A \in \mathcal{M}_n(\mathbb{K})$. The following theorem permits to sometimes simplify its calculation.

Theorem 4.21. *Let $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$. For any $i_0, j_0 \in \{1, ..., n\}$ we have*

$$\det A = R_{i_0}(A) = C_{j_0}(A).$$

Notation. If $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$, we usually write

$$\det A = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{vmatrix}.$$

Example 4.3.

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}.$$

We have the minors

$$\Delta_{11}(A) = d,\ \Delta_{12}(A) = c,\ \Delta_{21}(A) = b,\ \Delta_{22}(A) = a,$$

the cofactors

$$\mathit{Cof}_{11}(A) = d,\ \mathit{Cof}_{12}(A) = -c,\ \mathit{Cof}_{21}(A) = -b,\ \mathit{Cof}_{22}(A) = -a,$$

the determinant

$$\det(A) = R_1(A) = R_2(A) = C_1(A) = C_2(A) = ad - bc.$$

4.4.2 Properties

Proposition 4.22. *Let $A \in \mathcal{M}_n(\mathbb{K})$. We denote by $A_1, ..., A_n$ the columns of A and we write $A = (A_1, ..., A_n)$.*

1. *Swapping two columns of A multiplies its determinant by -1.*
2. *If C is an arbitrary column vector then*
$$\det(A_1, ..., A_{i-1}, A_i + C, A_{i+1}, ..., A_n) \\ = \det(A_1, ..., A_{i-1}, A_i, A_{i+1}, ..., A_n) + \det(A_1, ..., A_{i-1}, C, A_{i+1}, ..., A_n).$$
3. *If $\lambda \in \mathbb{K}$ then*
$$\det(A_1, ..., A_{i-1}, \lambda A_i, A_{i+1}, ..., A_n) = \lambda \det(A_1, ..., A_{i-1}, A_i, A_{i+1}, ..., A_n).$$

Remark 4.5. 1. *If A and B are two matrices then in general $\det(A + B) \neq \det A + \det B$.*

2. *If $\lambda \in \mathbb{K}$ then $\det(\lambda A) = \lambda^n \det A$.*

Corollary 4.23. *Let $A \in \mathcal{M}_n(\mathbb{K})$.*

1. *If a column of A is zero then $\det A = 0$.*
2. *If two columns of A are equal then $\det A = 0$.*
3. *If we add to a column of A a linear combination of the other columns of A then the determinant is unchanged.*
4. *If the columns of A are linearly dependent then $\det A = 0$.*

Proposition 4.24. *Let $A \in \mathcal{M}_n(\mathbb{K})$. We have*
$$\det A = \det A^T.$$

Consequently, the assertions of proposition 4.22 and corollary 4.23 apply also row-wise.

Calculating a determinant via recursive expansions is usually tedious. Only special matrices permit an easy calculation.

Proposition 4.25. *The determinant of a (lower or upper) triangular matrix is equal to the product of its diagonal coefficients.*

In particular, this applies to diagonal matrices. Notably, $\det I_n = 1$.

4.4.3 Determinant of a product, characterization of invertible matrices

The following algebraic property is remarkable.

Theorem 4.26. *Let $A, B \in \mathcal{M}_n(\mathbb{K})$. We have*
$$\det(AB) = \det A \det B.$$

It leads to a very useful characterization of invertible matrices.

Corollary 4.27. *Let $A \in \mathcal{M}_n(\mathbb{K})$. Then A is invertible if and only if $\det A \neq 0$. In this case we have*
$$\det(A^{-1}) = \frac{1}{\det A}.$$

4.5 LU decomposition (complement)

4.5.1 Matrix representation of Gaussian elimination

In this section, we formulate the Gaussian elimination method as a series of matrix operations. These concepts are very useful in computer implementations.

We recall the Gauss elimination principle:

$$\begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & & * \\ * & * & * & & * \\ \vdots & & & \ddots & \vdots \\ * & * & * & \cdots & * \end{pmatrix} \rightarrow \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & & * \\ 0 & * & * & & * \\ \vdots & & & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{pmatrix} \rightarrow \cdots \rightarrow \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & & * \\ 0 & 0 & * & & * \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & * \end{pmatrix}$$

Using the upper triangular matrix $U \in \mathcal{M}_n(\mathbb{K})$ ($U = (u_{ij})_{i,j=1}^n$ with $u_{ij} = 0$ for $i > j$), we can solve the reduced system

$$Ux = \tilde{b},$$

using backward substitution. Here, $\tilde{b}$ is the modification of the right hand side b through the elimination process. Let us consider the backward substitution in more detail:

Listing 4.1: **Backward substitution**
Let $U \in \mathcal{M}_n(\mathbb{K})$ be a regular upper triangular matrix, i.e., $u_{ii} \neq 0$. The solution $x \in \mathbb{K}^n$ of $Ux = b$ is given by:

```
|Set| $x_n = u_{nn}^{-1} b_n$ .
|For| $i$ |from| $n-1$ |until| 1
    $x_i = u_{ii}^{-1}\left(b_i - \sum_{j=i+1}^n u_{ij}x_j\right)$
```

It holds

Proposition 4.28 (Backward substitution)**.** *Let $U \in \mathcal{M}_n(\mathbb{K})$ be an upper triangular matrix with $u_{ii} \neq 0$. Then, the matrix U is regular and the backward substitution requires*

$$N_U(n) = \frac{n^2}{2} + O(n)$$

elementary operations on numbers.

Proof. It holds $\det(U) = \prod u_{ii} \neq 0$. Also the matrix U is regular.

Each step of the backward substitution consists of additions, multiplications and divisions of the diagonal elements. For $u_{ii} \neq 0$ each step is well-defined.

For the computation of x_i, we need $n - i$ multiplications and additions. Furthermore, we have one division per step. This yields

$$n + \sum_{i=1}^{n-1}(n-i) = n + (n-1)n - \frac{(n-1)n}{2} = \frac{n^2}{2} + \frac{n}{2}.$$

□

The transformation of A to a triangular structure is obtained through row-based elimination:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \ddots & a_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix} \to \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(3)} & \ddots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} \to$$

$$\to \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \ddots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix} \to \dots \to A^{(n-1)} =: U.$$

Starting from $A^{(0)} := A$, we obtain successively the matrices $A^{(i)}$ with $A^{(n-1)} =: U$. Therein, in step i, we eliminate the ith column of $A^{(i-1)}$ below the diagonal. This is obtained through subtraction of the $g_k^{(i)}$-fold of the ith row from the kth row. Here, it holds for $k = i+1, \dots, n$

$$g_k^{(i)} := \frac{a_{ki}^{(i-1)}}{a_{ii}^{(i-1)}},$$

and the new row is computed as

$$a_{kj}^{(i)} = a_{kj}^{(i-1)} - g_k^{(i)} a_{ij}^{(i-1)}, \quad k = i+1, \dots, n, \quad j = i, \dots, n.$$

These instructions are well-defined if $a_{ii}^{(i-1)} \neq 0$. This follows not necessarily from the regularity of A and all $A^{(i)}$. First of all, we assume that $a_{ii}^{(i-1)} \neq 0$ holds true, and discuss later the general case. In the ith elimination step, the first $i-1$ rows and columns remain unchanged. The ith elimination step is written in compact form with the help of a matrix-matrix multiplication

$$A^{(i)} = F^{(i)} A^{(i-1)}.$$

Here, the elimination matrix is

$$F^{(i)} := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -g_{i+1}^{(i)} & \ddots & & \\ & & \vdots & & \ddots & \\ & & -g_n^{(i)} & & & 1 \end{pmatrix}, \quad g_k^{(i)} := \frac{a_{ki}^{(i-1)}}{a_{ii}^{(i-1)}}.$$

Therein, all non-specified entries are zero. Multiple applications yield

$$U = A^{(n-1)} = F^{(n-1)} A^{(n-2)} = F^{(n-1)} F^{(n-2)} A^{(n-3)} = \underbrace{F^{(n-1)} \cdots F^{(1)}}_{=:F} A^{(0)} = FA. \quad (4.6)$$

Matrices $F^{(i)}$ with this structure are called Frobenius matrices . It holds

Proposition 4.29 (Frobenius matrix)**.** *Each Frobenius matrix $F^{(i)} \in \mathbb{R}^{n\times n}$ is regular and it holds*

$$F^{(i)} := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -g_{i+1} & \ddots & & \\ & & \vdots & & \ddots & \\ & & -g_n & & & 1 \end{pmatrix} \quad \Rightarrow \quad [F^{(i)}]^{-1} := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & g_{i+1} & \ddots & & \\ & & \vdots & & \ddots & \\ & & g_n & & & 1 \end{pmatrix}$$

For two Frobenius matrices $F^{(i_1)}$ and $F^{(i_2)}$ with $i_1 < i_2$, it holds

$$F^{(i_1)}F^{(i_2)} = F^{(i_1)} + F^{(i_2)} - I = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & -g^{(i_1)}_{i_1+1} & 1 & & & \\ & & \vdots & & \ddots & & \\ & & \vdots & & & 1 & & \\ & & \vdots & & & -g^{(i_2)}_{i_2+1} & \ddots & \\ & & \vdots & & & \vdots & & \ddots \\ & & -g^{(i_1)}_n & & & -g^{(i_2)}_n & & & 1 \end{pmatrix}.$$

Proof. Follows from component-wise calculations. □

Multiplying Frobenius matrices, we should take into account that these are not commutative. It holds

$$F^{(i_2)}F^{(i_1)} \neq F^{(i_1)} + F^{(i_2)} - I \text{ for } i_1 < i_2.$$

Careful observations yield for $i_1 < i_2 < i_3$ the simple generalization:

$$\begin{aligned} F^{(i_1)}F^{(i_2)}F^{(i_3)} &= F^{(i_1)}\big(F^{(i_2)} + F^{(i_3)} - I\big) = F^{(i_1)}F^{(i_2)} + F^{(i_1)}F^{(i_3)} - F^{(i_1)} \\ &= F^{(i_1)} + F^{(i_2)} - I + F^{(i_1)} + F^{(i_3)} - I - F^{(i_1)} \\ &= F^{(i_1)} + F^{(i_2)} + F^{(i_3)} - 2I \end{aligned}$$

We now continue with (4.6). With $F^{-(i)} := [F^{(i)}]^{-1}$ and using Proposition 4.29, we obtain that F as product of two regular matrices is itself regular. If follows

$$A = F^{-1}U = [F^{(n-1)} \cdots F^{(1)}]^{-1}U = \underbrace{F^{-(1)} \cdots F^{-(n-1)}}_{=:L}U$$

The matrix L is with the help of the generalization of Proposition 4.29 a lower triangular matrix with the diagonal entries 1:

$$L = \begin{pmatrix} 1 & & & & & \\ g^{(1)}_2 & 1 & & & & \\ g^{(1)}_3 & g^{(2)}_3 & 1 & & & \\ g^{(1)}_4 & g^{(2)}_4 & g^{(3)}_4 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ g^{(1)}_n & g^{(2)}_n & \cdots & \cdots & g^{(n-1)}_n & 1 \end{pmatrix}.$$

We summarize our results:

Proposition 4.30 (LU decomposition). *Let $A \in \mathcal{M}_n(\mathbb{K})$ be a regular matrix. We assume that all diagonal entries $a_{ii}^{(i-1)}$ arising through the elimination process are nonzero. Then, there exists a unique decomposition $A = LU$ with an upper triangular matrix $U \in \mathcal{M}_n(\mathbb{K})$ and a lower trianglar matrix $L \in \mathcal{M}_n(\mathbb{K})$ with diagonal entries 1. The computational cost to construct the LU decomposition requires*

$$\frac{1}{3}n^3 + O(n^2)$$

elementary operations.

Proof. (i) Uniqueness. Let us assume there exist two LU decompositions

$$A = L_1U_1 = L_2U_2 \quad \leftrightarrow \quad L_2^{-1}L_1 = U_2U_1^{-1}.$$

The product of triangular matrices is again a triangular matrix. The product $L_2^{-1}L_1$ has only values of 1 on the diagonal. It follows

$$L_2^{-1}L_1 = U_2U_1^{-1} = I,$$

and therefore $L_1 = L_2$ and $U_1 = U_2$.

(ii) Feasibility. Each step of the elimination is well-defined as long as we do not need to divide by $a_{ii}^{(i-1)} = 0$. The matrix F is per construction regular and therefore the matrix L exists.

(iii) Computational cost. In the $i$the elimination step

$$A^{(i)} = F^{(i)}A^{(i-1)}$$

we have first of all $n - i$ arithmetic operations for the computation of $g_j^{(i)}$ for $j = i+1, \dots, n$. The matrix-matrix multiplication applies to all elements a_{kl} with $k > i$ and $l > i$. It holds

$$a_{kl}^{(i)} = a_{kl}^{(i-1)} - g_k^{(i)}a_{il}^{(i-1)}, \quad k, l = i+1, \dots, n.$$

Here, we need $(n-i)^2$ arithmetic operations. In total, the computational cost in the $n-1$ steps sums-up to

$$N_{LU}(n) = \sum_{i=1}^{n-1} \{n - i + (n-i)^2\} = \sum_{i=1}^{n-1} \{i + i^2\},$$

and with the well-known summation rules, it follows

$$N_{LU}(n) = \frac{n^3}{3} + \frac{n}{3}.$$

□

The LU decomposition can be used to solve linear equation system:

Listing 4.2: **Solving $Ax = b$ with the LU decomposition**
Let $A \in \mathcal{M}_n(\mathbb{R})$ be a regular matrix which admits an LU decomposition.

```
Construct the LU decomposition  $A = LU$
Forward substitution:   $Ly = b$
Backward substitution:  $Ux = y$
```

The forward substitution is similar to Algorithm 4.1 and requires $O(n^2)$ operations according to Proposition 4.28. This is an interesting observation. The solution process is much cheaper than constructing the LU decomposition itself. For this reason, if possible, the LU decomposition is only constructed when needed, but in many applications it can be used several times (to solve several systems with the same matrix). This should be considered in implementations and reduces significantly the computational cost for large problems.

Remark 4.6 (Practical aspects)**.** *The matrix L is a lower triangular matrix with values 1 on the diagonal. These values are not required to be stored. Similarly, the zero-elements of $A^{(i)}$ below the diagonal are not required to be stored either. Consequently, both matrices L and U can be stored in a common square matrix. In step i, it holds*

$$\tilde{A}^{(i)} = \left(\begin{array}{ccccccc} a_{11} & a_{12} & a_{13} & \cdots & \cdots & \cdots & a_{1n} \\ \boldsymbol{l_{21}} & a_{22}^{(1)} & a_{23}^{(1)} & & & & \vdots \\ \boldsymbol{l_{31}} & \boldsymbol{l_{32}} & a_{33}^{(2)} & & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & & & \vdots \\ \cline{5-7} \vdots & & & \boldsymbol{l_{i+1,i}} & a_{i+1,i+1}^{(i)} & \cdots & a_{i+1,n}^{(i)} \\ \vdots & & & \vdots & & \ddots & \vdots \\ \boldsymbol{l_{n1}} & \boldsymbol{l_{n2}} & \boldsymbol{\cdots} & \boldsymbol{l_{n,i}} & a_{n,i+1}^{(i)} & \cdots & a_{nn}^{(i)} \end{array}\right).$$

The bold entries are the entries of L. The values above the line do not change anymore during the algorithm and form already the respective entries of L and U.

The algorithm reads:

Listing 4.3: **LU decomposition of a matrix A without additional memory**
Let $A = (a_{ij})_{ij} \in \mathcal{M}_n(\mathbb{K})$ be a regular matrix for which the LU decomposition exists

```
|For| $i$ |from| 1 |until| $n$
   |For| $k$ |from| $i+1$ |until| $n$
     $a_{ki} = a_{ki}/a_{ii}$
      |For| $j$ |from| $i+1$ |until| $n$
        $a_{kj} = a_{kj} - a_{ki} \cdot a_{ij}$
```

The element $a_{ii}^{(i-1)}$ is the so-called *Pivot element*. Until now, this element was assumed to be non-zero. For regular matrices this is however in general not the case. Let us consider the following example:

$$A := \begin{pmatrix} 1 & 4 & 2 \\ 2 & 8 & 1 \\ 1 & 2 & 1 \end{pmatrix}.$$

In the first step of the construction, we have $a_{11}^{(0)} = 1$ and it holds

$$A^{(1)} = F^{(1)}A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 2 \\ 2 & 8 & 1 \\ 1 & 2 & 1 \end{pmatrix} = \left(\begin{array}{ccc} 1 & 4 & 2 \\ \cline{2-3} 0 & 0 & -3 \\ 0 & -2 & -1 \end{array}\right)$$

At this step, the algorithm aborts, because we have $a_{22}^{(1)} = 0$. We could have continued the algorithm with the choice $a_{32}^{(i)} = -2$ as a new Pivot element. This approach can be applied in a systematic manner and is called pivoting. In the ith step of the scheme, we search first for an appropriate Pivot element a_{ki} in the ith column with $k \geq i$. The kth and ith rows are switched and the LU decomposition can be continued. Switching of the kth and ith rows is achieved by multiplying with a Pivot matrix:

$$P^{ki} := \begin{pmatrix} 1 &&&&&&&&& \\ & \ddots &&&&&&&& \\ && 1 &&&&&&& \\ &&& 0 & 0 & \dots & 0 & 1 &&& \\ &&& 0 & 1 && & 0 &&& \\ &&& \vdots && \ddots && \vdots &&& \\ &&& 0 &&& 1 & 0 &&& \\ &&& 1 & 0 & \cdots & 0 & 0 &&& \\ &&&&&&&& 1 && \\ &&&&&&&&& \ddots & \\ &&&&&&&&&& 1 \end{pmatrix}$$

It holds $p^{ki}_{jj} = 1$ for $j \neq k$ and $j \neq i$ and $p^{ki}_{ki} = p^{ki}_{ik} = 1$; all other elements are zero. We collect now some properties of P:

Proposition 4.31 (Pivot matrices)**.** *Let $P = P^{ki}$ be the Pivot matrix with $P^{ki}_{jj} = 1$ for $j \neq k, i$ and $P^{ki}_{ki} = P^{ki}_{ik} = 1$. Applying $P^{ki}A$ from left switches the kth and ith row of A. Applying AP^{ki} from right switches kth and ith column. It holds*

$$P^2 = I \text{ hence } P^{-1} = P.$$

Proof. It is left to the reader. □

In step i of the LU decomposition, we search first the Pivot element:

Listing 4.4: **Pivot search** In step i, search index $k \geq i$, such that

```
|Search| Index $k \geq i$, such that
  $|a_{ki}| = \max_{j \geq i} |a_{ji}|$
|Set| $P^{(i)} := P^{ki}$.
```

Afterward, we determine $A^{(i)}$ as

$$A^{(i)} = F^{(i)} P^{(i)} A^{(i-1)}.$$

Pivoting ensures that all elements $g^{(i)}_k = a^{(i-1)}_{ki} / a^{(i-1)}_{ii}$ of $F^{(i)}$ are bounded by 1 in their absolute value. In summary, we obtain

$$U = A^{(n-1)} = F^{(n-1)} P^{(n-1)} \cdots F^{(1)} P^{(1)} A. \tag{4.7}$$

The Pivot matrices are not commutative with A and $F^{(i)}$. A straightforward transit to the LU decomposition is therefore not possible. We define

$$\tilde{F}^{(i)} := P^{(n-1)} \cdots P^{(i+1)} F^{(i)} P^{(i+1)} \cdots P^{(n-1)}.$$

The matrix $\tilde{F}^{(i)}$ is obtained through multiple switches of rows and columns of $F^{(i)}$. Specifically, only rows and columns with $j > i$ are switched. The matrix $\tilde{F}^{(i)}$ has the same pattern as $F^{(i)}$ and in particular only values of 1 on the diagonal. This means that it is again a Frobenius matrix and Proposition 4.29 still holds true. Only the entries in the ith column below the diagonal are permuted. For the inverse, it holds correspondingly

$$\tilde{L}^{(i)} := [\tilde{F}^{(i)}]^{-1} = P^{(n-1)} \cdots P^{(i+1)} L^{(i)} P^{(i+1)} \cdots P^{(n-1)}.$$

The simultaneous switch of rows and columns yields unchanged diagonal elements. The matrix $\tilde{L}^{(i)}$ is again a Frobenius matrix and only the elements in the column $l_{ij}, i > j$ are permuted. We form (4.7) through smart insertion of permutation matrices:

$$U = \tilde{F}^{(n-1)} \tilde{F}^{(n-2)} \cdots \tilde{F}^{(1)} \underbrace{P^{(n-1)} \cdots P^{(1)}}_{=:P} A$$

We illustrate this process with the help of a simple example:

$$\begin{aligned}
U &= F^{(3)}P^{(3)}F^{(2)}P^{(2)}F^{(1)}P^{(1)}A \\
&= F^{(3)}P^{(3)}F^{(2)}\underbrace{P^{(3)}P^{(3)}}_{=I}P^{(2)}F^{(1)}\underbrace{P^{(2)}P^{(3)}P^{(3)}P^{(2)}}_{=I}P^{(1)}A \\
&= \underbrace{F^{(3)}}_{=\tilde F^{(3)}}\underbrace{P^{(3)}F^{(2)}P^{(3)}}_{=\tilde F^{(2)}}\underbrace{P^{(3)}P^{(2)}F^{(1)}P^{(2)}P^{(3)}}_{=\tilde F^{(1)}}\underbrace{P^{(3)}P^{(2)}P^{(1)}}_{=P}A
\end{aligned}$$

With $\tilde L^{(i)} = [\tilde F^{(i)}]^{-1}$ it holds

$$\underbrace{\tilde L^{(1)}\cdots\tilde L^{(n-1)}}_{=:\tilde L}U = PA.$$

Since $\tilde L^{(i)}$ are again Frobenius matrices, it holds further with Proposition 4.29:

$$\begin{aligned}
\tilde L &= \tilde L^{(1)}\cdots\tilde L^{(n-1)} \\
&= \sum_{i=1}^{n-1}\tilde L^{(i)} - (n-2)I \\
&= P^{(n-1)}\Big(L^{(n-1)} + P^{(n-1)}\Big(L^{(n-2)} + \ldots \\
&\qquad + \cdots + P^{(2)}F^{(1)}P^{(2)}\Big)\ldots P^{(n-2)}\Big)P^{(n-1)} - (n-2)I.
\end{aligned}$$

For the construction of the LU decomposition not only do we need to permute the $A^{(i)}$, but we also need to change the $L^{(i)}$ that have been computed so far. We summarize:

Proposition 4.32 (LU decomposition with pivoting). *Let $A \in \mathcal{M}_n(\mathbb{K})$ be a regular matrix. Then, there exists an LU decomposition*

$$PA = LU,$$

where P is a product of Pivot matrices, L is a lower triangular matrix with values 1 on the diagonal and U an upper triangular matrix. The LU decomposition without pivoting $P = I$ is unique in case it exists.

Proof. The proof is left for the reader. □

We recall that pivoting yields a well-defined construction of the LU decomposition. But there is a second advantage. With the help of pivoting, we can increase the numerical stability of Gaussian elimination. Through the choice of a Pivot element a_{ki} with maximal relative value (with respect to the row), we can reduce the risk of cancellation.

Example 4.4 (LU decomposition without pivoting). *Let:*

$$A = \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 1.4 & 1.1 & -0.7 \\ 0.8 & 4.3 & 2.1 \end{pmatrix}, \quad b = \begin{pmatrix} 1.2 \\ -2.1 \\ 0.6 \end{pmatrix},$$

and the solution of $Ax = b$ is given by (five-digit accuracy):

$$x \approx \begin{pmatrix} 0.34995 \\ -0.98023 \\ 2.1595 \end{pmatrix}.$$

For the matrix A it holds $\mathrm{cond}_\infty(A) = \|A\|_\infty\|A^{-1}\|_\infty \approx 7.2\cdot 1.2 \approx 8.7$. This problem is well-conditioned. The amplification factor is 8.7 and we expect an amplification of the error of about one *digit. First, we construct the LU decomposition (three-digit accuracy). The entries of L are in bold letters:*

$$F^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1.4}{2.3} & 1 & 0 \\ -\frac{0.8}{2.3} & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 \\ -0.609 & 1 & 0 \\ -0.348 & 0 & 1 \end{pmatrix},$$

$$[L^{(1)}, A^{(1)}] \approx \left(\begin{array}{ccc} 2.3 & 1.8 & 1.0 \\ \cline{2-3} 0.609 & 0.0038 & -1.31 \\ 0.348 & 3.67 & 1.75 \end{array}\right).$$

In the second step, we obtain

$$F^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{3.67}{0.0038} & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -966 & 1 \end{pmatrix},$$

$$[L^{(2)}L^{(1)}, A^{(2)}] \approx \left(\begin{array}{ccc} 2.3 & 1.8 & 1.0 \\ 0.609 & 0.0038 & -1.31 \\ \cline{3-3} 0.348 & 966 & 1270 \end{array}\right).$$

The LU decomposition is then

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.609 & 1 & 0 \\ 0.348 & 966 & 1 \end{pmatrix}, \quad U := \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 0 & 0.0038 & -1.31 \\ 0 & 0 & 1270 \end{pmatrix}.$$

We solve the linear equation system through forward and backward substitution:

$$A\tilde{x} = L\underbrace{U\tilde{x}}_{=y} = b$$

First, we have

$$y_1 = 1.2, \qquad y_2 = -2.1 - 0.609 \cdot 1.2 \approx -2.83,$$
$$y_3 = 0.6 - 0.348 \cdot 1.2 + 966 \cdot 2.83 \approx 2730.$$

And finally, we obtain

$$\tilde{x}_3 = \frac{2730}{1270} \approx 2.15,$$
$$\tilde{x}_2 = \frac{-2.83 + 1.31 \cdot 2.15}{0.0038} \approx -3.55,$$
$$\tilde{x}_1 = \frac{1.2 + 1.8 \cdot 3.55 - 1 \cdot 2.15}{2.3} \approx 2.37.$$

For the solution $\tilde{x}$ it holds

$$\tilde{x} = \begin{pmatrix} 2.37 \\ -3.55 \\ 2.15 \end{pmatrix}, \quad \frac{\|\tilde{x} - x\|_2}{\|x\|_2} \approx 1.4,$$

which is a relative error of 140%. *Here, we only considered round-off errors and not yet perturbed entries!*

The previous example shows the significance of pivoting. In the second step, we chose as Pivot element 0.0038, which is close to 0. For this reason, we obtained values of different orders in the matrices L and U. This yields instabilities in the computation.

Example 4.5 (LU decomposition with pivoting)**.** *We continue the previous example in step 2 and search now first the Pivot element:*

$$[L^{(1)}, A^{(1)}] = \left(\begin{array}{ccc} 2.3 & 1.8 & 1.0 \\ \cline{2-3} 0.609 & 0.0038 & -1.31 \\ 0.348 & \boxed{3.67} & 1.75 \end{array}\right), \quad P^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

Then:

$$[\tilde{L}^{(1)}, \tilde{A}^{(1)}] = \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 0.348 & 3.67 & 1.75 \\ 0.609 & 0.0038 & -1.31 \end{pmatrix}.$$

It follows

$$F^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{0.0038}{3.67} & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -0.00104 & 1 \end{pmatrix},$$

$$[L^{(2)}\tilde{L}^{(1)}, \tilde{A}^{(2)}] \approx \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 0.348 & 3.67 & 1.75 \\ 0.609 & 0.00104 & -1.31 \end{pmatrix}.$$

We obtain the decomposition $\tilde{L}U = PA$ as

$$\tilde{L} := \begin{pmatrix} 1 & 0 & 0 \\ 0.348 & 1 & 0 \\ 0.609 & 0.00104 & 1 \end{pmatrix}, \quad U := \begin{pmatrix} 2.3 & 1.8 & 1.0 \\ 0 & 3.67 & 1.75 \\ 0 & 0 & -1.31 \end{pmatrix}, \quad P := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The linear equation system, we solve in the form:

$$PAx = \tilde{L}\underbrace{Ux}_{=y} = Pb.$$

For the right hand side, it holds $\tilde{b} = Pb = (1.2, 0.6, -2.1)^T$ and forward substitution in $\tilde{L}y = \tilde{b}$ yields

$$\begin{aligned} y_1 &= 1.2, \\ y_2 &= 0.6 - 0.348 \cdot 1.2 \approx 0.182, \\ y_3 &= -2.1 - 0.609 \cdot 1.2 - 0.00104 \cdot 0.182 \approx -2.83. \end{aligned}$$

As approximation $\tilde{x}$, we obtain

$$\tilde{x} = \begin{pmatrix} 0.350 \\ -0.980 \\ 2.160 \end{pmatrix}$$

with a relative error

$$\frac{\|\tilde{x} - x\|_2}{\|x\|_2} \approx 0.0002,$$

which is now only 0.02% *rather than* 140% *as in the previous example.*

These examples show that the constructed LU decomposition is not really a truly decomposition, but only an approximation of $A \approx LU$ due to round-off errors. Computing LU easily yields the test and we can compute the error $A - LU$.

As we emphasized at the beginning of this section, the LU decomposition is one of the most important schemes for solving linear equation systems. Due to third order computational cost, however, computing time exceeds easily the capacities of modern computers - despite faster and bigger machines! For instance, we obtain as illustration:

Within the discretization of partial differential equations (see Chapter 19) we usually obtain very large linear equation systems with $n \sim 10^6 - 10^9$. The good news are that these systems have a special structure, such as symmetry or a sparse pattern with just a few non-zero entries. This is for instance the case for the finite element discretization of Poisson's problem. Here, in contrast to the previous table, we may have $n = 1\,000\,000$, but thanks to these structural properties, we can solve such systems within one minute.

n	Operations ($\approx \frac{1}{3}n^3$)	Time LU decomposition
100	$300 \cdot 10^3$	30 μs
1 000	$300 \cdot 10^6$	30 ms
10 000	$300 \cdot 10^9$	30 s
100 000	$300 \cdot 10^{12}$	10 h
1 000 000	$300 \cdot 10^{15}$	1 year

Table 4.2: Computational time for constructing the LU decomposition of a matrix $A \in \mathcal{M}_n(\mathbb{K})$ on a computer with 10 GigaFLOPS with optimal load-balancing.

4.5.2 LU decomposition for diagonal-dominant matrices

Proposition 4.32 showed us that the LU decomposition for arbitrary regular matrices with pivoting exists. On the other hand, there are many matrices for which the LU decomposition is stable without pivoting. Examples are positive definite or diagonal dominant matrices:

Definition 4.31 (Diagonal dominant matrices)**.** *A matrix $A \in \mathcal{M}_n(\mathbb{K})$ is called* diagonal dominant*, if*

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \quad i = 1, \dots, n.$$

A diagonal dominant matrix has the largest (absolute) value on the diagonal. For regular matrices the diagonal is moreover always nonzero.

Proposition 4.33 (LU decomposition of diagonal dominant matrices)**.** *Let $A \in \mathcal{M}_n(\mathbb{K})$ be a regular and diagonal dominant matrix. Then, the LU decomposition can be constructed without pivoting and all Pivot elements $a_{ii}^{(i-1)}$ are non-zero.*

4.5.3 Case of Hermitian positive definite matrices: Cholesky decomposition

In the case of a Hermitian positive definite matrix A, a variant of the LU decomposition can be obtained, namely the Cholesky factorization. It has the form

$$A = \bar{C}^T C,$$

where C is an upper triangular matrix with real positive diagonal entries.

4.6 Canonical inner product, Euclidean norm, matrix norm

4.6.1 Canonical inner product and Euclidean norm

The general definition of a norm has been given in Definition 1.3, and the general definition of an inner product will be given in chapter 6. In this chapter dedicated to linear algebra in $\mathbb{K}^n$ we restrict ourselves to the canonical inner product and the associated Euclidean norm which are the most standard ones.

Definition 4.32. *The canonical inner (or scalar) product of two vectors $(x_1, \cdots, x_n) \in \mathbb{K}^n$ and $(y_1, \cdots, y_n) \in \mathbb{K}^n$ is defined by*

$$\langle x, y \rangle := \overline{x_1} y_1 + \cdots + \overline{x_n} y_n.$$

The Euclidean norm of $(x_1, \cdots, x_n) \in \mathbb{K}^n$ is defined by

$$\|x\|_2 := \sqrt{\langle x, x \rangle} = \sqrt{|x_1|^2 + \cdots + |x_n|^2}.$$

Obviously, for real vectors, conjugacy and moduli can be dropped.

We have the classical Cauchy-Schwarz inequality for vectors of $\mathbb{K}^n$:

Proposition 4.34. *For all $x, y, \in \mathbb{K}^n$ we have*

$$|\langle x, y \rangle| \leq \|x\|_2 \|y\|_2.$$

4.6.2 Matrix norm

Definition 4.33. *A matrix norm on $\mathcal{M}_n(\mathbb{K})$ is a norm $\|\cdot\|$ on the vector space $\mathcal{M}_n(\mathbb{K})$ that satisfies*

$$\|AB\| \leq \|A\|\|B\| \qquad \forall A, B \in \mathcal{M}_n(\mathbb{K}).$$

As example we first mention the Frobenius norm, which is associated with an inner product (see chapter 6).

Definition 4.34. *The inner product of $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$ and $B = (b_{ij}) \in \mathcal{M}_n(\mathbb{K})$ is defined by*

$$\langle A, B\rangle := \operatorname{tr}(\overline{A}^T B) = \sum_{i=1}^{n}\sum_{j=1}^{n} \overline{a_{ij}} b_{ij}.$$

The Frobenius norm of $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{K})$ is defined by

$$\|A\|_F := \sqrt{\langle A, A\rangle} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}.$$

There are also the so-called induced norms, which are more appropriate for some applications. These norms are constructed from a vector norm.

Definition 4.35. *Let $\|.\|$ be a vector norm on $\mathbb{K}^n$. The induced norm on $\mathcal{M}_n(\mathbb{K})$ is defined by*

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}. \tag{4.8}$$

It is an easy exercise to check that such norms are indeed matrix norms. However they are not associated with an inner product on matrices, even if the Euclidean vector norm is considered.

Solving the maximization problem (4.8) is in principle not an easy task. However, when the Euclidean vector norm is used, it can be reformulated as an eigenvalue problem.

Proposition 4.35. *The matrix norm induced by the Euclidean norm satisfies*

$$\begin{aligned} \|A\|_2 &= \sqrt{\varrho(\bar{A}^T A)} \\ &= \varrho(A) \text{ if } A \text{ is Hermitian (or real symmetric),} \end{aligned}$$

where ϱ stands for the spectral radius (Definition 5.4 in next chapter). It is called the spectral norm.

By definition an induced norm satisfies

$$\|Ax\| \leq \|A\|\|x\| \qquad \forall x \in \mathbb{K}^n.$$

When a matrix norm and a vector norm satisfy the above relation they are said to be compatible.

Chapter 5

Diagonalization of square matrices and applications

Diagonalization of matrices has many applications. The major purpose is to simplify some given matrix (in case it is diagonalizable) to a diagonal matrix by conserving important properties such as the rank, characteristic polynomial, determinant, trace, and eigenvalues and their algebraic multiplicities. Mathematically, diagonalization is the process to construct a similar matrix. Similar matrices represent the same linear mapping under two different bases.

5.1 Eigenvalues, eigenvectors, diagonalization

5.1.1 Eigenvalues and eigenvectors

Let $A \in \mathcal{M}_n(\mathbb{K})$.

Definition 5.1. *Let $\lambda \in \mathbb{K}$. We say that λ is an eigenvalue of A if there exists a column vector $x \in \mathbb{K}^n$, $x \neq 0$, such that $Ax = \lambda x$. Such an x is called an eigenvector associated with the eigenvalue λ.*

Proposition 5.1. *The function*

$$\begin{array}{rcl} \mathbb{K} & \to & \mathbb{K} \\ \lambda & \mapsto & P_A(\lambda) := \det(A - \lambda I_n) \end{array}$$

is a polynomial of degree n. It is called the characteristic polynomial of A.

Proposition 5.2. *Let $\lambda \in \mathbb{K}$. Then λ is an eigenvalue of A if and only if $P_A(\lambda) = 0$.*

Determining the eigenvalues of a matrix amounts to finding the roots of its characteristic polynomial. However this is not an easy task, since characteristic polynomials have no special property[1]. Explicit formulas exist only for $n \leq 4$. For the numerical approximation of eigenvalues iterative methods are used. They are usually not based on the characteristic polynomial. The simplest method, the power method, is described in section 5.3.

Definition 5.2. *Let λ be an eigenvalue of A. We call eigenspace of A associated with the eigenvalue λ the set*

$$E_A(\lambda) = \{x \in \mathbb{K}^n \textit{ s.t. } Ax = \lambda x\}.$$

It is a linear subspace of $\mathbb{K}^n$.

In other words, $E_A(\lambda)$ is the set of eigenvectors associated with λ and the zero vector.

[1]Given an arbitrary polynomial P one can always construct a matrix, called companion matrix, whose characteristic polynomial is P, up to a multiplicative constant.

5.1.2 Diagonalization

Definition 5.3. *A matrix $A \in \mathcal{M}_n(\mathbb{K})$ is diagonalizable if there exists an invertible matrix $P \in \mathcal{M}_n(\mathbb{K})$ such that $D := P^{-1}AP$ is diagonal.*

Remark 5.1. *The matrix $D := P^{-1}AP$ corresponds to a change of coordinate system (or change of basis) in the map $x \mapsto Ax$. Indeed, if $x = P\tilde{x}$, $y = P\tilde{y}$ and $y = Ax$ then $\tilde{y} = P^{-1}AP\tilde{x}$. Therefore P is called a change of basis matrix (or transition matrix). Its columns contain the coordinates of the new basis vectors in the former (here canonical) basis. The relation $D = P^{-1}AP$ is straightforwardly inverted into $A = PDP^{-1}$.*

Proposition 5.3. *A matrix $A \in \mathcal{M}_n(\mathbb{K})$ is diagonalizable if and only if there exists a basis $\mathcal{B} = (v_1, ..., v_n)$ such that each each v_i is an eigenvector associated with an eigenvalue λ_i of A. In this case, if P is the matrix made columnwise of the vectors $(v_1, ..., v_n)$, then*

$$P^{-1}AP = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} =: \operatorname{diag}(\lambda_1, ..., \lambda_n).$$

Therefore it is to be remembered that the columns of a transition matrix associated with a diagonalization form a basis of eigenvectors.

Remark 5.2. *The diagonal coefficients λ_i need not be distinct.*

Proposition 5.4. *Eigenvectors associated with distinct eigenvalues are linearly independent.*

Corollary 5.5. *If $A \in \mathcal{M}_n(\mathbb{K})$ admits n distinct eigenvalues then A is diagonalizable.*

The fact that a matrix be diagonalizable is related to the number of its eigenvalues and their order of multiplicity[2]. We do not enter into these developments, which are extremely classical and well-documented. We only stress that a real matrix may be diagonalizable on $\mathbb{C}$ while it is not on $\mathbb{R}$, due to a possibly larger number of eigenvalues. The case of symmetric and Hermitian matrices is of special interest.

5.1.3 Case of Hermitian matrices

Theorem 5.6. *If A is Hermitian then A is diagonalizable with the help of real eigenvalues. Moreover the transition matrix P can be chosen such that $P^{-1} = \bar{P}^T$ (this is called a unitary matrix).*

This theorem has a direct consequence on the characterization of Hermitian positive (semi-) definite matrices.

Corollary 5.7. *If $A \in \mathcal{M}_n(\mathbb{K})$ be a Hermitian matrix, then A is positive semi-definite (resp. positive definite) if and only if all its eigenvalues are ≥ 0 (resp. > 0).*

5.2 Applications

The diagonalization has many applications. One of them, the expression of matrix power, turns out to be very important in the analysis of iterative methods.

Proposition 5.8. *If $A = PDP^{-1}$ with $D = \operatorname{diag}(\lambda_1, ..., \lambda_n)$ then, for all $k \in \mathbb{N}$, we have $A^k = PD^kP^{-1}$ with $D^k = \operatorname{diag}(\lambda_1^k, ..., \lambda_n^k)$.*

[2]The notion of order of multiplicity is related, but not equivalent, to the fact that an eigenvalue may appear in several places in the diagonal form.

When k goes to infinity the behavior of A^k is governed be the λ_i^k's. If $|\lambda_i| < 1$ for all i then clearly $A^k \to 0$ when $k \to +\infty$. If $|\lambda_{i_0}| > 1$ for some index i_0 then, calling v_{i_0} an eigenvector associated with λ_{i_0}, we have $A^k v_{i_0} = \lambda^k v_{i_0}$, whose norm goes to infinity when $k \to +\infty$.

Definition 5.4. *The largest modulus of the eigenvalues (on $\mathbb{C}$) of a (real or complex) matrix is called spectral radius.*

A fairly similar reasoning can be made concerning the analysis of first order linear differential systems with constant coefficients, where diagonalization uncouples the system, leading to explicit solutions:

$$y' = Ay \Leftrightarrow (P^{-1}y)' = D(P^{-1}y) \Leftrightarrow \left\{ \begin{array}{l} y = Pz, \ z = (z_1, ..., z_n)^T \\ z_i' = \lambda_i z_i \ \forall i = 1, ..., n \end{array} \right. .$$

Here the signs of the real parts of the eigenvalues govern the asymptotic behavior of the solutions.

5.3 The power method (complement)

There exist many more or less sophisticated methods for the numerical approximation of all or some eigenvalues of a given matrix. The extremal eigenvalues (in modulus or in real part) often carry the most important information, as mentioned in section 5.2. The power method is an elementary method aiming at computing the largest modulus of the eigenvalues.

Let $A \in \mathcal{M}_n(\mathbb{R})$. The power method consists in defining two sequences (x_n) and (y_n) by

- $x_0 \in \mathbb{R}^n$ given,
- for all $k \in \mathbb{N}^*$

$$y_k = Ax_{k-1}, \qquad x_k = \frac{y_k}{\|y_k\|}.$$

Here, $\|.\|$ is an arbitrary norm on $\mathbb{R}^n$, typically the Euclidean norm is used.

Theorem 5.9. *We assume that A is diagonalizable (possibly on $\mathbb{C}$), with basis of eigenvectors $(e_1, ..., e_n)$, chosen to be of unit norm, associated with the eigenvalues $(\lambda_1, ..., \lambda_n)$. We assume further that*

$$|\lambda_1| \le |\lambda_2| \le ... \le |\lambda_{n-1}| < |\lambda_n|.$$

If $x_0 = \sum_{i=1}^{n} \beta_i e_i$ with $\beta_n \neq 0$, then

$$\lim_{k \to +\infty} \|y_k\| = |\lambda_n|.$$

Moreover, if λ_n is real and positive, $e_n \in \mathbb{R}^n$, then $\lim_{k \to +\infty} x_k = \pm e_n$.

Remark 5.3.
1. *There is a priori no way to ensure that x_0 admits a nonzero coordinate along e_n, since by definition e_n is unknown. However, because of round-off errors in a numerical implementation, such a component will automatically appear in the vector x_k after some iterations.*
2. *A variant, the inverse power method, aims at computing the smallest modulus of the eigenvalues.*

Chapter 6

Vector spaces and elements of topology

Here again, $\mathbb{K}$ stands either for $\mathbb{R}$ or for $\mathbb{C}$.

6.1 Vector spaces

6.1.1 General definition

We have defined linear subspaces of $\mathbb{K}^n$ in section 4.3, however there is a more general and axiomatic definition of the concept of vector space.

Definition 6.1. *A vector space on $\mathbb{K}$ is a set X, whose elements are called vectors, equipped with two operations*

$$(x,y) \in X^2 \mapsto x+y \in X,$$

$$(\lambda, x) \in \mathbb{K} \times X \mapsto \lambda x \in X$$

satisfying the following properties:

$$\forall x,y,z \in X, \qquad (x+y)+z = x+(y+z),$$

$$\forall x,y \in X, \qquad x+y = y+x,$$

$$\textit{there exists a vector denoted by } 0 \textit{ such that } 0+x = x \textit{ for all } x \in X,$$

$$\textit{for all } x \in X \textit{ there exists a vector denoted by } -x \textit{ such that } x+(-x) = 0\ ,$$

$$\forall (\lambda,\mu,x) \in \mathbb{K}\times\mathbb{K}\times X, \qquad (\lambda\mu)x + \lambda(\mu x),$$

$$\forall x \in X, \qquad 1x = x,$$

$$\forall (\lambda,x,y) \in \mathbb{K}\times X\times X, \qquad \lambda(x+y) = \lambda x + \lambda y,$$

$$\forall (\lambda,\mu,x) \in \mathbb{K}\times\mathbb{K}\times X, \qquad (\lambda+\mu)x = \lambda x + \mu x.$$

Clearly, $\mathbb{K}^n$ equipped with its standard operations is a vector space on $\mathbb{K}$. This also holds true for all its subspaces. We speak of real vector space when $\mathbb{K} = \mathbb{R}$, and of complex vector space when $\mathbb{K} = \mathbb{C}$.

6.1.2 Vector spaces of finite dimension

Definition 6.2. *Let X be a vector space and $(u_i)_{i\in I}$ be a family (not necessarily finite) of elements of X. A vector $x \in X$ is said to be a linear combination of $(u_i)_{i\in I}$ if there exist a finite subfamily $(u_i)_{i\in J}$ and a family $(\alpha_i)_{i\in J}$ of elements of $\mathbb{K}$ such that*

$$x = \sum_{i\in J} \alpha_i u_i.$$

Definition 6.3. *A vector space X is said to be of finite dimension if there exists a finite family $(u_1, \cdots, u_n)$ such that every vector of X is a linear combination of $(u_1, \cdots, u_n)$.*

Of course, $\mathbb{K}^n$ is of finite dimension. But **function spaces are usually not of finite dimension**. For instance $\mathcal{C}([a,b],\mathbb{R})$, the set of continuous real-valued functions on $[a,b]$ is a vector space but it is not of finite dimension. There are exceptions, however. For instance the space of polynomial functions of degree $\leq n$ is of finite dimension: it is spanned by the monomials $(x^k)_{0\leq k\leq n}$.

6.2 Basic topology

6.2.1 Norms and balls

Definition 6.4. *Let X be a vector space. A norm on X is a function*

$$x \in X \mapsto \|x\| \in \mathbb{R}_+$$

that satisfies:

$$\begin{aligned} \forall x \in X, &\qquad \|x\| = 0 \Leftrightarrow x = 0, \\ \forall (x,\alpha) \in X \times \mathbb{K}, &\qquad \|\alpha x\| = |\alpha|\|x\|, \\ \forall (x,y) \in X^2, &\qquad \|x+y\| \leq \|x\| + \|y\|. \end{aligned} \tag{6.1}$$

A vector space endowed with a norm is called a normed vector space.

On $X = \mathbb{K}$, the canonical norm is the modulus (or absolute value) $|\cdot|$.

The triangle inequality (6.1) implies other inequality

$$\forall (x,y) \in X^2, \qquad |\|x\| - \|y\|| \leq \|x - y\|. \tag{6.2}$$

Due to this property, we say that the norm is a 1-Lipschitz (or non-expansive) function.

When there is no confusion the notation $\|x\|$ will be systematically used for the norm of x on its underlying space.

Example 6.1. *On $\mathbb{K}^n$ the most classical norms are defined, for $x = (x_1, \cdots, x_n)$, by*

$$\|x\|_1 = \sum_{i=1}^n |x_i|, \qquad \|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}, \qquad \|x\|_\infty = \max_{i=1,\cdots,n} |x_i|.$$

For instance, for $x = (-1, 2)$ we obtain $\|x\|_1 = 3$, $\|x\|_2 = \sqrt{5}$, $\|x\|_\infty = 2$.

The norm $\|\cdot\|_2$ has already been mentioned, and it is the most classical norm because it has a noticeable property: it is associated with the canonical inner product of $\mathbb{K}^n$. Actually, to check that the norms from example 6.1 are indeed norms according to Definition 6.4, the only nontrivial task is to check the triangle inequality (6.1) for $\|\cdot\|_2$. For this we use the expression of the norm in terms of the canonical inner product and the Cauchy-Schwarz inequality from Proposition 4.34 to obtain

$$\begin{aligned} \|x+y\|_2^2 = \langle x+y, x+y\rangle = \|x\|_2^2 + \|y\|_2^2 + 2\Re\langle x,y\rangle &\leq \|x\|_2^2 + \|y\|_2^2 + 2|\langle x,y\rangle| \\ &\leq \|x\|_2^2 + \|y\|_2^2 + 2\|x\|_2\|y\|_2 = (\|x\|_2 + \|y\|_2)^2. \end{aligned}$$

Definition 6.5. *Let X be a normed vector space, $z \in X$ and $\rho > 0$. The open ball of center z and radius ρ is the set*

$$B_\rho(z) = \{x \in X \ \text{ s.t. } \ \|x - z\| < \rho\}.$$

The closed ball of center z and radius ρ is the set

$$\bar{B}_\rho(z) = \{x \in x \ \text{ s.t. } \ \|x - z\| \leq \rho\}.$$

When the radius ρ is chosen equal to 1 and the center is fixed at the origin, the balls $B_1(0)$ and $\bar{B}_1(0)$ are called the (open or closed) unit balls. See Fig. 6.1.

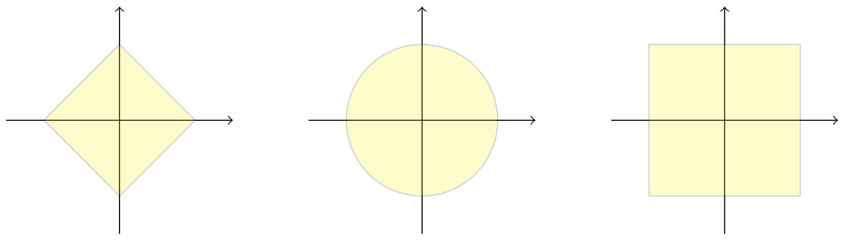


Figure 6.1: Unit balls of $\mathbb{R}^2$ for the norms (from left to right): $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$.

Definition 6.6. *Let X be a normed vector space and $z \in X$. A subset U of X is said to be a neighborhood of z if there exists $\rho > 0$ such that $B_\rho(z) \subset U$.*

6.2.2 Open sets, closed sets (complement)

Definition 6.7. *Let X be a normed vector space. A subset A of X is said to be open if*

$$\forall z \in A \ \exists \rho > 0 \ \text{ s.t. } \ B_\rho(z) \subset A,$$

i.e., A is a neighborhood of all its points.

More generally, a subset $A \subset E \subset X$ is said to be open relatively to the set E if

$$\forall z \in A \ \exists \rho > 0 \ \text{ s.t. } \ B_\rho(z) \cap E \subset A,$$

i.e., A is a relative neighborhood of all its points.

Definition 6.8. *Let X be a normed vector space. A subset A of X is said to be closed if its complementary set $X \setminus A$ is open.*

A subset $A \subset E \subset X$ is said to be closed relatively to E if $E \setminus A$ is open relatively to E.

It is a classical exercise to show that open balls are indeed open sets, and that closed balls are closed sets.

On $\mathbb{R}$ (equipped with the absolute value), all intervals of the form $]a, b[$, $]-\infty, b[$, $]a, +\infty[$ are open, and all intervals of the form $[a, b]$, $]-\infty, b]$, $[a, +\infty[$ are closed. Intervals of the form $[a, b[$ or $]a, b]$ are neither open nor closed. The interval $]-\infty, +\infty[= \mathbb{R}$ is open and closed.

The following proposition is straightforward and also applies to relative open / closed sets.

Proposition 6.1. *An arbitrary union of open sets is an open set. A finite intersection of open sets is an open set.*

An arbitrary intersection of closed sets is a closed set. A finite union of closed sets is a closed set.

6.2.3 Closure, interior (complement)

Definition 6.9. *Let X be a normed vector space and A be a subset of X. The closure of A, denoted by $\bar{A}$, is the subset of X defined by*

$$\forall x \in X, \qquad x \in \bar{A} \Leftrightarrow \forall \rho > 0, B_\rho(x) \cap A \neq \emptyset.$$

The closure of a set A is always closed, in fact it is the smallest (in the sense of inclusion) closed set containing A. If A is closed then $\bar{A} = A$.

Definition 6.10. *Let X be a normed vector space and A be a subset of X. The interior of A, denoted by* $\operatorname{int} A$ *is the subset of X defined by*

$$\forall x \in X, \qquad x \in \operatorname{int} A \Leftrightarrow \exists \rho > 0, B_\rho(x) \subset A.$$

The interior of a set A is always open, in fact it is the largest open set contained in A. If A is open then $\operatorname{int} A = A$.

It is classical to show that $X \setminus \bar{A} = \operatorname{int}(X \setminus A)$ and $X \setminus \operatorname{int} A = \overline{X \setminus A}$. In addition, the closure of an open ball is the closed ball of the same center and radius. The interior of a closed ball is the open ball of the same center and radius.

6.2.4 Converging sequences (complement)

The notion of convergence in normed vector spaces extends that of real-valued sequences, only replacing the absolute value by the norm.

Definition 6.11. *Let X be a normed vector space and (x_n) be a sequence of elements of X. We say that (x_n) is converging to $x \in X$ if*

$$\forall \varepsilon > 0 \ \exists N \in \mathbb{N} \ \ s.t. \ \ n \geq N \Rightarrow \|x_n - x\| < \varepsilon.$$

This can be phrased as: x_n becomes arbitrarily close to x provided that n be large enough.

6.2.5 Equivalent norms

Definition 6.12. *Two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on X are said to be equivalent if there exists $\alpha, \beta > 0$ such that*

$$\alpha\|x\|_a \leq \|x\|_b \leq \beta\|x\|_a \qquad \forall x \in X.$$

Proposition 6.2. *If two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on X are equivalent then a subset of X is open for the norm $\|\cdot\|_a$ if and only if it is open for the norm $\|\cdot\|_b$. Of course, the same holds for closed sets. Likewise, if the norms $\|\cdot\|_a$ and $\|\cdot\|_b$ are equivalent, a sequence of X is converging for the norm $\|\cdot\|_a$ if and only if it converges for the norm $\|\cdot\|_b$.*

Theorem 6.3. *If X is a vector space of finite dimension then all norms on X are equivalent.*

This property is convenient, however it only holds in finite dimensional spaces. Be careful!

6.3 Special normed spaces (complement)

6.3.1 Inner product spaces

Definition 6.13. *Let X be a vector space. A function*

$$(x, y) \in X^2 \mapsto \langle x, y \rangle \in \mathbb{K}$$

is said to be an inner product (or scalar product) if

$$\forall (x, y) \in X^2, \qquad \langle x, y \rangle = \overline{\langle y, x \rangle},$$

$$\forall (x,x',y)\in X^3, \forall \alpha\in\mathbb{K}, \qquad \langle \alpha x + x', y\rangle = \overline{\alpha}\langle x,y\rangle + \langle x',y\rangle,$$
$$\forall x\in X\setminus\{0\}, \qquad \langle x,x\rangle > 0.$$

A vector space endowed with an inner product is called inner product space.

Observe that two above properties imply

$$\langle x, \alpha y + y'\rangle = \alpha\langle x,y\rangle + \langle x,y'\rangle.$$

Therefore one gets the standard linearity of the inner product with respect to the second argument, while one sometimes speaks of antilinearity with respect to the first argument because of the conjugacy. Prescribing the antilinearity with respect to the first argument, hence the linearity with respect to the second one, is a matter of (non-universal) convention. Of course, when $\mathbb{K}=\mathbb{R}$ this discussion is irrelevant.

Typical inner products are those defined in section 4.6 (in the case of matrices, considering $\mathcal{M}_n(\mathbb{K})$ as a vector space).

Theorem 6.4. *If X is an inner product space then it is a normed space for the norm defined by*

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

In addition we have the Cauchy-Schwarz inequality

$$\forall (x,y)\in X^2, \qquad |\langle x,y\rangle| \le \|x\|\|y\|.$$

The norm defined above will be considered as canonical in any inner product space.

If x, y belong to an inner product space then simple algebra yields the expansion formula

$$\|x+y\|^2 = \|x\|^2 + \|y\|^2 + 2\langle x,y\rangle,$$

the parallelogram equality

$$\|x+y\|^2 + \|x-y\|^2 = 2(\|x\|^2 + \|y\|^2),$$

and the polarization identity

$$\langle x,y\rangle = \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right).$$

If A is an arbitrary subset of X then we define the orthogonal of A by

$$A^\perp = \{x\in X \text{ s.t. } \langle x,y\rangle = 0\ \forall y\in A\}.$$

It is always a closed linear subspace of X. When A is a singleton we often write $a^\perp$ for $\{a\}^\perp$.

6.3.2 Banach and Hilbert spaces

Let us first extend the notion of real-valued Cauchy sequence.

Definition 6.14. *Let X be a normed vector space and (x_n) be a sequence of elements of X. We say that (x_n) is a Cauchy sequence if*

$$\forall\varepsilon>0\ \exists N\in\mathbb{N}\ s.t.\ n,m\ge N \Rightarrow \|x_n - x_m\| < \varepsilon.$$

Definition 6.15. *A normed vector space such that every Cauchy sequence is converging is said to be a Banach space. When it is at the same time an inner product space, it is called a Hilbert space.*

Note that a Banach space remains a Banach space when changing the norm to an equivalent one. But in general, the choice of the norm is crucial when defining a Banach space.

Theorem 6.5. *A normed vector space of finite dimension is a Banach space.*

6.3.3 Complement on matrix norms

In order to establish the convergence of sequences of vectors of $\mathbb{K}^n$ the choice of the norm is irrelevant, as explained in section 6.2.5. When such sequences arise from matrix operations it comes the question of the choice of a compatible matrix norm (see section 4.6.2). Optimal convergence results take advantage of this freedom thanks to the following convenient theorem. It highlights the prominent role played by the spectral radius (see Definition 5.4).

Theorem 6.6. *Let A be a square matrix with coefficients in $\mathbb{K}$.*

1. *It holds for every matrix norm $\|\cdot\|$*
$$\varrho(A) \leq \|A\|.$$

2. *For all $\varepsilon > 0$ there exists an induced matrix norm $\|\cdot\|$ such that*
$$\|A\| \leq \varrho(A) + \varepsilon.$$

As example of application we immediately see that the sequence (A^n) goes to 0 as soon as $\rho(A) < 1$, since $\|A^n\| \leq \|A\|^n \leq (\varrho(A) + \varepsilon)^n$ and ε can be taken arbitrarily small. The if and only if statement holds also true by definition of an eigenvalue. We underline that it is not requested that A be diagonalizable.

Chapter 7

Linear equation systems and iterative solvers (complement)

7.1 Stability analysis of linear equation systems

7.1.1 Motivation

Previously, we have shown how to solve linear equation systems. Tacitly we assumed that we deal with 'nice' numbers. However, in practice the computer will solve these systems for us and in most cases we deal with floating point numbers from $\mathbb{R}$. As we discussed in the introduction, we now deal with round-off errors.

The question comes, how such round-off errors may influence the solution of $Ax = b$. As an example, let us consider

$$\begin{pmatrix} 0.988 & 0.960 \\ 0.992 & 0.963 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.084 \\ 0.087 \end{pmatrix}$$

with the solution $(x, y)^T = (3, -3)^T$. We compute the solution with a precision of three correct digits using Gauss elimination:

$$\left(\begin{array}{cc|c} 0.988 & 0.960 & 0.084 \\ 0.992 & 0.963 & 0.087 \end{array}\right) \quad \times 0.988/0.992$$

$$\left(\begin{array}{cc|c} 0.988 & 0.960 & 0.084 \\ 0.988 & 0.959 & 0.0866 \end{array}\right) \quad \downarrow -$$

$$\left(\begin{array}{cc|c} 0.988 & 0.959 & 0.084 \\ 0 & 0.001 & -0.0026 \end{array}\right) \quad .$$

We now have the typical triangular system of A. The right hand side b was accordingly modified. Through backward substitution we obtain

$$\begin{aligned} 0.001y &= -0.0026 \;\Rightarrow\; y = -2.6 \\ 0.988x &= 0.087 - 0.959 \cdot (-2.6) \approx 2.58 \;\Rightarrow\; x = 2.61. \end{aligned}$$

Here, we have $(x, y) = (2.61, -2.60)$. The relative error of the numerical approximation is more than 10%. It seems that the solution of linear equation systems is either ill-conditioned (see the general concept of condition numbers in the chapter 2) or the Gaussian elimination is numerically unstable. The conditioning and stability will be addressed of the following subsection. In addition, in many applications, we deal with very large linear equation systems, where $A \in \mathbb{R}^{n\times n}$ with $n \gg 10^6$ and larger. For this reason, the efficiency and memory requirements play a crucial role and therefore, direct methods (such as Gaussian elimination) are unfortunately not always feasible, and rather iterative methods come into play.

7.1.2 Stability analysis

We are given $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ and consider

$$Ax = b.$$

Through numerical errors, round-off errors, false input data or experimental errors, A and b may be perturbed:

$$\tilde{A}\tilde{x} = \tilde{b}.$$

Here, $\tilde{A} = A + \delta A$ and $\tilde{b} = b + \delta b$ with the perturbations δA and δb, respectively.

We derive now results for the conditioning and error amplification for the solution of linear equation systems. Errors can be in the matrix A as well as in the right hand side b.

We first consider errors in the right hand side vector.

Proposition 7.1 (Perturbation of the right hand side b)**.** *Let $A \in \mathbb{R}^{n\times n}$ be a regular matrix and $b \in \mathbb{R}^n$. With $x \in \mathbb{R}^n$ we denote the solution of $Ax = b$. Let δb be a perturbation of the right hand side $\tilde{b} = b + \delta b$ and $\tilde{x}$ the solution of the perturbed system $A\tilde{x} = \tilde{b}$. Furthermore, we denote with $\|\cdot\|$ a matrix norm that is compatible with the vector norm $\|\cdot\|$. Then, it holds*

$$\frac{\|\delta x\|}{\|x\|} \le \operatorname{cond}(A)\frac{\|\delta b\|}{\|b\|},$$

with the condition number *of the matrix, i.e.,*

$$\operatorname{cond}(A) = \|A\| \cdot \|A^{-1}\|.$$

Proof. Let $\|\cdot\|$ be an arbitrary matrix norm with compatible vector norm $\|\cdot\|$. For the solution $x \in \mathbb{R}^n$ and the perturbed solution $\tilde{x} \in \mathbb{R}^n$, we have

$$\tilde{x} - x = A^{-1}(A\tilde{x} - Ax) = A^{-1}(\tilde{b} - b) = A^{-1}\delta b$$

Thus:

$$\frac{\|\delta x\|}{\|x\|} \le \|A^{-1}\|\frac{\|\delta b\|}{\|x\|}\cdot\frac{\|b\|}{\|b\|} = \|A^{-1}\|\frac{\|\delta b\|}{\|b\|}\cdot\frac{\|Ax\|}{\|x\|} \le \underbrace{\|A\|\cdot\|A^{-1}\|}_{=:\operatorname{cond}(A)}\frac{\|\delta b\|}{\|b\|}.$$

□

Remark 7.1. *The* condition number *of a matrix is very important in numerical linear algebra. Let us also consider the matrix-vector multiplication $y = Ax$. With perturbed input $\tilde{x} = x + \delta x$, we obtain, since obviously* $\operatorname{cond}(A) = \operatorname{cond}(A^{-1})$*:*

$$\frac{\|\delta y\|}{\|y\|} \le \operatorname{cond}(A)\frac{\|\delta x\|}{\|x\|}.$$

Note that by definition the condition number of a matrix depends on the chosen norm. are also equivalent. For symmetric matrices it can be inferred that $\operatorname{cond}_2(A)$*:*

$$\operatorname{cond}_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\max\{|\lambda|,\ \lambda \textit{ eigenvalue of } A\}}{\min\{|\lambda|,\ \lambda \textit{ eigenvalue of } A\}}.$$

In the second step, we consider now a perturbation δA of the matrix A. First, we need to ensure that $\tilde{A} = A + \delta A$ is still regular.

Lemma 7.2. *Let $\|\cdot\|$ be the matrix norm induced by the vector norm. Let $B \in \mathbb{R}^{n\times n}$ be a matrix with $\|B\| < 1$. Then, the matrix $I + B$ is regular and it holds the estimate*

$$\|(I+B)^{-1}\| \le \frac{1}{1 - \|B\|}.$$

Proof. It holds

$$\|(I+B)x\| \geq \|x\| - \|Bx\| \geq (1-\|B\|)\|x\|.$$

Since $1-\|B\|>0$, the mapping $I+B$ is one-to-one. Thus, $I+B$ is regular. Furthermore, we have

$$\begin{aligned}1 = \|I\| = \|(I+B)(I+B)^{-1}\| &= \|(I+B)^{-1} + B(I+B)^{-1}\| \\ &\geq \|(I+B)^{-1}\| - \|B\|\,\|(I+B)^{-1}\| = \|(I+B)^{-1}\|(1-\|B\|) > 0.\end{aligned}$$

□

With this result, we address now the perturbation of a matrix A:

Proposition 7.3 (Perturbation of the matrix A)**.** *Let $A\in\mathbb{R}^{n\times n}$ be a regular matrix, and $b\in\mathbb{R}^n$. Furthermore, let $x\in\mathbb{R}^n$ be the solution of $Ax=b$ and let $\tilde{A}=A+\delta A$ a perturbed matrix with $\|\delta A\|\leq\|A^{-1}\|^{-1}$. For the perturbed solution $\tilde{x}=x+\delta x$ of $\tilde{A}\tilde{x}=b$, it holds*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\operatorname{cond}(A)}{1-\operatorname{cond}(A)\|\delta A\|/\|A\|}\frac{\|\delta A\|}{\|A\|}.$$

Proof. It holds for x and the perturbed solution $\tilde{x}$ and the error $\delta x := \tilde{x}-x$:

$$\begin{aligned}(A+\delta A)\tilde{x} &= b \\ (A+\delta A)x &= b+\delta Ax\end{aligned} \quad\Rightarrow\quad \delta x = -[A+\delta A]^{-1}\delta Ax.$$

According to our assumption $\|A^{-1}\delta A\|\leq\|A^{-1}\|\,\|\delta A\|<1$, it follows with the Lemma 7.2:

$$\begin{aligned}\|\delta x\| \leq \|[I+A^{-1}\delta A]^{-1}A^{-1}\delta A\|\,\|x\| &\leq \frac{\|A^{-1}\|}{1-\|A^{-1}\delta A\|}\|\delta A\|\,\|x\| \\ &\leq \frac{\operatorname{cond}(A)}{1-\|A^{-1}\|\|\delta A\|}\frac{\|\delta A\|}{\|A\|}\|x\|.\end{aligned}$$

We establish the assertion with the expansion $\|A\|/\|A\|$. □

The combination of the two previous perturbation propositions yields the main result:

Theorem 7.4 (Perturbation theorem for linear equation systems)**.** *Let $A\in\mathbb{R}^{n\times n}$ be a regular matrix and $b\in\mathbb{R}^n$ a right hand side vector. Furthermore, let $x\in\mathbb{R}^n$ be the solution of $Ax=b$ with the regular matrix $A\in\mathbb{R}^{n\times n}$. For the solution $\tilde{x}\in\mathbb{R}^n$ of the perturbed system $\tilde{A}\tilde{x}=\tilde{b}$ with $\delta b=\tilde{b}-b$ and $\delta A=\tilde{A}-A$ and the assumption*

$$\|\delta A\| < \frac{1}{\|A^{-1}\|}$$

it holds the estimate

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\operatorname{cond}(A)}{1-\operatorname{cond}(A)\|\delta A\|/\|A\|}\left(\frac{\|\delta b\|}{\|b\|}+\frac{\|\delta A\|}{\|A\|}\right)$$

with the condition number

$$\operatorname{cond}(A) = \|A\|\,\|A^{-1}\|.$$

Proof. We combine the results from Proposition 7.1 and 7.3. To this end, let x be the solution of $Ax=b$, $\tilde{x}$ the perturbed solution $\tilde{A}\tilde{x}=\tilde{b}$ and $\hat{x}$ the solution to the perturbed right hand side $A\hat{x}=\tilde{b}$. Then, we obtain

$$\|x-\tilde{x}\| \leq \|x-\hat{x}\| + \|\hat{x}-\tilde{x}\| \leq \operatorname{cond}(A)\frac{\|\delta b\|}{\|b\|}\|x\| + \frac{\operatorname{cond}(A)}{1-\operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|}}\frac{\|\delta A\|}{\|A\|}\|x\|.$$

Taking into account $\|\delta A\|<\|A^{-1}\|^{-1}$, we have

$$0 \leq \operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|} < \|A\|\,\|A^{-1}\|\,\frac{1}{\|A\|\,\|A^{-1}\|} = 1.$$

Therefore,

$$\operatorname{cond}(A) \le \frac{\operatorname{cond}(A)}{1 - \operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|}},$$

which shows the assertion. □

With this theorem, we re-consider our motivating example from the beginning of this section. We recall

$$A = \begin{pmatrix} 0.988 & 0.959 \\ 0.992 & 0.963 \end{pmatrix}$$
$$\Rightarrow \quad A^{-1} \approx \begin{pmatrix} 8302 & -8267 \\ -8552 & 8517 \end{pmatrix}.$$

Employing the maximum row sum $\|\cdot\|_\infty$ it holds

$$\|A\|_\infty = 1.955, \quad \|A^{-1}\|_\infty \approx 17069, \quad \operatorname{cond}_\infty(A) \approx 33370.$$

Solving linear equation systems is indeed often ill-conditioned. As a consequence, round-off errors may be significantly amplified. This also shows that the difficulty is an intrinsic part of the problem statement, but not necessarily a stability issue with the Gaussian elimination (provided appropriate pivoting is performed!). These are partially good news and the reason why Gaussian elimination is often used for moderate sizes of the matrix A.

7.2 Basic iterative solvers: Richardson, Jacobi, Gauss-Seidel, conjugate gradients

For a moderate number of unknowns a so-called direct solver (LU decomposition, Cholesky) is a good choice to solve $AU = B$. Such methods are always available in most software packages such as Matlab, octave, python, etc:

```
U = sp.sparse.linalg.spsolve(A,B) // in Python
U = A\B // in octave / Matlab
```

For big systems, **iterative solvers** are the methods of choice because they require less memory and less computational cost than direct solvers. We provide a brief introduction in the following section.

7.2.1 Fixed-point solvers: Richardson, Jacobi, Gauss-Seidel

A large class of schemes is based on so-called **fixed point** methods for solving:

$$f(x) = x.$$

We provide in the following a brief introduction that is based on [25]. First, we have

Definition 7.1. *Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ and $C \in \mathbb{R}^{n\times n}$. For an initial guess $x^0 \in \mathbb{R}^n$ we iterate for $k = 1, 2, \ldots$:*

$$x^k = x^{k-1} + C(b - Ax^{k-1}).$$

Please be careful that k does not denote the power, but the current iteration index. Furthermore, we introduce:

$$B := I - CA \quad \text{and} \quad c := Cb.$$

Then:

$$x^k = Bx^{k-1} + c.$$

Thanks to the construction of

$$g(x) = Bx + c = x + C(b - Ax)$$

it is trivial to see that in the limit $k \to \infty$, it holds

$$g(x) = x$$

with the solution

$$Ax = b.$$

Remark 7.2. *Thanks to Banach's fixed point theorem (see section 12.3), we can investigate under which conditions the above scheme will converge. Actually it should hold*

$$\|B\| < 1,$$

for a matrix norm such that

$$\|g(x) - g(y)\| \leq \|B\|\, \|x - y\|.$$

A similar condition appears in the stability analysis of the schemes for solving ODEs, as reported in the numerical tests of Section 17.6 (blow-up, zig-zag solution and converged solution). A big problem (which is also true for the ODE cases) is that different norms may predict different results. For instance it may happen that

$$\|B\|_2 < 1 \quad \textit{but} \quad \|B\|_\infty > 1.$$

Discussions can be found for instance in [25]. In particular, if the spectral radius of B is $\varrho(B) < 1$ then $\|B\| < 1$ for some induced matrix norm and $\varrho(B) > 1$ implies $\|B\| > 1$ for all matrix norm.

We concentrate now on the algorithmic aspects. The two fundamental requirements for the matrix C (defined above) are:

- It should hold $C \approx A^{-1}$ and therefore $\|I - CA\| \ll 1$;
- It should be simple to construct C.

Of course, we easily see that these two requirements are conflicting statements. As always in numerics we need to find a trade-off that is satisfying for the developer and the computer.

Definition 7.2 (Richardson iteration)**.** *The simplest choice of C is the identity matrix, i.e.,*

$$C = I$$

Then, we obtain the Richardson iteration

$$x^k = x^{k-1} + \omega(b - Ax^{k-1})$$

with a relaxation parameter $\omega > 0$.

Further schemes require more work and we need to decompose the matrix A first:

$$A = L + D + U.$$

Here, L is a lower-triangular matrix, D a diagonal matrix, and U an upper-triangular matrix. In more detail:

$$A = \underbrace{\begin{pmatrix} 0 & & \dots & 0 \\ a_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ a_{n1} & \dots & a_{n,n-1} & 0 \end{pmatrix}}_{=:L} + \underbrace{\begin{pmatrix} a_{11} & & \dots & 0 \\ & \ddots & & \\ & & \ddots & \\ 0 & \dots & & a_{nn} \end{pmatrix}}_{=:D} + \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ & \ddots & \ddots & \vdots \\ & & \ddots & a_{n-1,n} \\ 0 & \dots & & 0 \end{pmatrix}}_{=:U}.$$

With this, we can now define two very classical schemes:

Definition 7.3 (Jacobi method)**.** *To solve $Ax = b$ with $A = L + D + U$ let $x^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \ldots$*

$$x^k = x^{k-1} + D^{-1}(b - Ax^{k-1})$$

or in other words $J := -D^{-1}(L + U)$:

$$x^k = Jx^{k-1} + D^{-1}b.$$

Definition 7.4 (Gauss-Seidel method)**.** *To solve $Ax = b$ with $A = L + D + U$ let $x^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \ldots$*

$$x^k = x^{k-1} + (D + L)^{-1}(b - Ax^{k-1})$$

or in other words $H := -(D + L)^{-1}U$:

$$x^k = Hx^{k-1} + (D + L)^{-1}b.$$

To implement these two schemes, we provide the presentation in index-notation:

Theorem 7.5 (Index-notation of the Jacobi and Gauss-Seidel methods)**.** *One step of the Jacobi method and Gauss-Seidel method, respectively, can be carried out in $n^2 + O(n)$ operations. For each step, in index-notation for each entry it holds:*

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{k-1}\right), \quad i = 1, \ldots, n,$$

i.e., (for the Gauss-Seidel method):

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{k} - \sum_{j>i} a_{ij}x_j^{k-1}\right), \quad i = 1, \ldots, n.$$

An alternative class of methods is based on so-called **descent** or **gradient** directions, which further improve the previously introduced methods. So far, we have:

$$x^{k+1} = x^k + d^k, \quad k = 1, 2, 3, \ldots$$

where d^k denotes the **direction** in which we go at each step. For instance:

$$d^k = D^{-1}(b - Ax^k), \quad d^k = (D + L)^{-1}(b - Ax^k)$$

for the Jacobi and Gauss-Seidel methods, respectively. To improve these kind of iterations, we have two possiblities:

- introduce a relaxation (or so-called damping) parameter $\omega^k > 0$ (possibly adapted at each step) such that
$$x^{k+1} = x^k + \omega^k d^k,$$
- and/or to improve the search direction d^k such that we reduce the error as best as possible.

We restrict our attention to positive definite symmetric matrices as they appear naturally many problems. A key point is another view on the problem by regarding it as a minimization problem for which $Ax = b$ is the first-order necessary condition and consequently the sought solution. Imagine for simplicity that we want to minimize $f(x) = \frac{1}{2}ax^2 - bx$. The first-order necessary condition is nothing else than the derivative $f'(x) = ax - b$. We find a possible minimum via $f'(x) = 0$, namely

$$ax - b = 0 \quad \Rightarrow \quad x = a^{-1}b, \quad \text{if } a \neq 0.$$

That is exactly the same how we would solve a linear matrix system $Ax = b$. By regarding it as a minimum problem we understand better the purpose of our derivations: How does minimizing a function $f(x)$ work in terms of an iteration? Well, we try to minimize f at each step k:

$$f(x^0) > f(x^1) > \ldots > f(x^k).$$

This means that the direction d^k (to determine $x^{k+1} = x^k + \omega^k d^k$) should be a descent direction. This idea can be applied for solving linear equation systems. We first define the quadratic form

$$Q(y) = \frac{1}{2}(Ay, y) - (b, y),$$

where $(\cdot, \cdot)$ is the Euclidian scalar product. Then, we can define

Algorithm 7.6 (Descent method - basic idea)**.** *Let $A \in \mathbb{R}^{n\times n}$ be positive definite and $x^0, b \in \mathbb{R}^n$. Then for $k = 0, 1, 2, \ldots$*

- *Compute d^k;*
- *Determine ω^k as minimum of $\omega^k = argmin\, Q(x^k + \omega^k d^k)$;*
- *Update $x^{k+1} = x^k + \omega^k d^k$.*

For instance d^k can be determined via the Jacobi or Gauss-Seidel methods.

Another possibility is the gradient method in which we use the gradient to obtain search directions d^k. This brings us to the gradient method:

Algorithm 7.7 (Gradient descent)**.** *Let $A \in \mathbb{R}^{n\times n}$ positive definite and the right hand side $b \in \mathbb{R}^n$. Let the initial guess be $x^0 \in \mathbb{R}$ and the initial search direction $d^0 = b - Ax^0$. Then $k = 0, 1, 2, \ldots$*

- *Compute the vector $r^k = Ad^k$;*
- *Compute the relaxation*
$$\omega^k = \frac{\|d_k\|_2^2}{(r^k, d^k)};$$
- *Update the solution vector $x^{k+1} = x^k + \omega^k d^k$;*
- *Update the search direction vector $d^{k+1} = d^k - \omega^k r^k$.*

One can show that the gradient method converges to the solution of the linear equation system $Ax = b$ (see for instance [25]).

7.2.2 Conjugate gradients (a Krylov space method)

This section is copy and paste from [25][Section 7.8] and translation from German into English. Gradient descent may converge slowly in most cases because search directions are in general $d^k \not\perp d^{k+2}$.

Formulation of the CG scheme. In order to enhance the performance of gradient descent, the conjugate gradient (CG) scheme was developed. Here, the search directions $\{d^0, \ldots, d^{k-1}\}$ are pairwise orthogonal. The measure of orthogonality is achieved by using the A scalar product:

$$(Ad^r, d^s) = 0 \quad \forall r \neq s.$$

At step k, we seek the approximation $x^k = x^0 + \sum_{i=0}^{k-1} \alpha_i d^i$ as the minimum of all $\alpha = (\alpha_0, \ldots, \alpha_{k-1})$ with respect to $Q(x^k)$:

$$\min_{\alpha\in\mathbb{R}^k} Q\left(x^0 + \sum_{i=0}^{k-1} \alpha_i d^i\right) = \min_{\alpha\in\mathbb{R}^k} \left\{ \frac{1}{2}\left(Ax^0 + \sum_{i=0}^{k-1} \alpha_i A d^i, x^0 + \sum_{i=0}^{k-1} \alpha_i d^i\right) - \left(b, x^0 + \sum_{i=0}^{k-1} \alpha_i d^i\right)\right\}.$$

The stationary point is given by

$$0 \overset{!}{=} \frac{\partial}{\partial \alpha_j} Q(x^k) = \left(Ax^0 + \sum_{i=0}^{k-1} \alpha_i A d^i, d^j \right) - (b, d^j) = -\left(b - Ax^k, d^j \right), \quad j = 0, \dots, k-1.$$

Therefore, the new residual $b - Ax^k$ is perpendicular to all search directions d^j for $j = 0, \dots, k-1$. The resulting linear equation system

$$(b - Ax^k, d^j) = 0 \quad \forall j = 0, \dots, k-1 \tag{7.1}$$

has the feature of "Galerkin orthogonality", similar to FEM schemes.

While constructing the CG method, new search directions should be linearly independent of the current d^j. Otherwise, the space would not become larger and consequently, the approximation cannot be improved.

Definition 7.5 (Krylov space). *We choose an initial approximation $x^0 \in \mathbb{R}^n$ and set $d^0 := b - Ax^0$. The Krylov space $K_k(d^0, A)$ is*

$$K_k(d^0, A) := \operatorname{span}\{d^0, Ad^0, \dots, A^{k-1} d^0\}.$$

Here, A^k means the k-th power of A.

It holds:

Lemma 7.8. *Assume $A^k d^0 \in K_k(d^0, A)$. Then, the solution $x \in \mathbb{R}^n$ of $Ax = b$ is an element of $K_k(d_0, A)$.*

Proof. Let $x^k \in x_0 + K_k(d^0, A)$ be the best approximation, which fulfills the Galerkin equation (7.1) Let $r^k := b - Ax^k$. Since

$$r^k = b - Ax^k = \underbrace{b - Ax^0}_{=d^0} + A \underbrace{(x^0 - x^k)}_{\in K_k(d^0, A)} \in d^0 + AK_k(d^0, A)$$

it holds $r^k \in K_{k+1}(d^0, A)$. Supposing that $K_{k+1}(d^0, A) \subset K_k(d^0, A)$, we obtain $r^k \in K_k(d^0, A)$. The Galerkin equation yields $r^k \perp K_k(d^0, A)$, from which we obtain $r^k = 0$ and $Ax^k = b$. □

If the CG scheme aborts since it cannot find new search directions, the solution is found. Let us assume that the A-orthogonal search directions $\{d^0, d^1, \dots, d^{k-1}\}$ have been found, then we can compute the next CG approximation using the basis representation $x^k = x^0 + \sum \alpha_i d^i$ and employing the Galerkin equation:

$$\left(b - Ax^0 - \sum_{i=0}^{k-1} \alpha_i A d^i, d_j \right) = 0 \quad \Rightarrow \quad (b - Ax^0, d^j) = \alpha_j (Ad^j, d^j) \Rightarrow \quad \alpha_j = \frac{(d^0, d^j)}{(Ad^j, d^j)}.$$

The A-orthogonal basis $\{d^0, \dots, d^{k-1}\}$ of the Krylov space $K_k(d^0, A)$ can be computed with the Gram-Schmidt procedure. However, this procedure has a high computational cost; see e.g., [25]. A better procedure is a two-step recursion formula, which is efficient and stable:

Lemma 7.9 (Two-step recursion formula). *Let $A \in \mathbb{R}^{n \times n}$ symmetric positive definite and $x^0 \in \mathbb{R}^n$ and $d^0 := b - Ax^0$. Then, for $k = 1, 2, \dots,$ the iteration*

$$r^k := b - Ax^k, \quad \beta_{k-1} := -\frac{(r^k, Ad^{k-1})}{(d^{k-1}, Ad^{k-1})}, \quad d^k := r^k - \beta_{k-1} d^{k-1}$$

constructs an A-orthogonal basis with $(Ad^r, d^s) = 0$ for $r \neq s$. Here x^k in step k defines the new Galerkin solution $(b - Ax^k, d^j) = 0$ for $j = 0, \dots, k-1$.

Proof. See [25]. □

We collect now all ingredients to construct the CG scheme. Let x^0 be an initial guess and $d^0 := b - Ax^0$ the resulting defect. Suppose that $K_k := \operatorname{span}\{d^0, \dots, d^{k-1}\}$ and $x^k \in x^0 + K_k$ and $r^k = b - Ax^k$ have been computed. Then, we can compute the next iterate d^k according to Lemma 7.9:

$$\beta_{k-1} = -\frac{(r^k, Ad^{k-1})}{(d^{k-1}, Ad^{k-1})}, \quad d^k = r^k - \beta_{k-1} d^{k-1}. \tag{7.2}$$

For the new coefficient α_k in $x^{k+1} = x^0 + \sum_{i=0}^k \alpha_i d^i$ holds with testing in the Galerkin equation (7.1) with d^k:

$$\Bigg(\underbrace{b - Ax^0}_{=d^0} - \sum_{i=0}^{k} \alpha_i Ad^i, d^k\Bigg) = (b - Ax^0, d^k) - \alpha_k (Ad^k, d^k) = (b - Ax^0 + \underbrace{A(x^0 - x^k)}_{\in K_k}, d^k) - \alpha_k (Ad^k, d^k).$$

That is

$$\alpha_k = \frac{(r^k, d^k)}{(Ad^k, d^k)}, \quad x^{k+1} = x^k + \alpha_k d^k. \tag{7.3}$$

This allows to compute the new residual r^{k+1}:

$$r^{k+1} = b - Ax^{k+1} = b - Ax^k - \alpha_k Ad^k = r^k - \alpha_k Ad^k. \tag{7.4}$$

We summarize (7.2 − 7.4) and formulate the classical CG scheme:

Algorithm 7.10. *Let $A \in \mathbb{R}^{n\times n}$ symmetric positive definite and $x^0 \in \mathbb{R}^n$ and $r^0 = d^0 = b - Ax^0$ be given. Iterate for $k = 0, 1, \dots$:*

1. $\alpha_k = \frac{(r^k, d^k)}{(Ad^k, d^k)}$
2. $x^{k+1} = x^k + \alpha_k d^k$
3. $r^{k+1} = r^k - \alpha_k Ad^k$
4. $\beta_k = \frac{(r^{k+1}, Ad^k)}{(d^k, Ad^k)}$
5. $d^{k+1} = r^{k+1} - \beta_k d^k$

Without round-off errors, the CG scheme yields after (at most) n steps the solution of a n-dimensional problem and is in this sense a direct method rather than an iterative scheme. However, in practice for huge n, the CG scheme is usually stopped earlier, yiedling an approximate solution.

Proposition 7.11 (CG as a direct method)**.** *Let $x^0 \in \mathbb{R}^n$ be any initial guess. Assuming no round-off errors, the CG scheme terminates after (at most) n steps with $x^n = x$. At each step, we have:*

$$Q(x^k) = \min_{\alpha\in\mathbb{R}} Q(x^{k-1} + \alpha d^{k-1}) = \min_{y \in x^0 + K_k} Q(y)$$

i.e.,

$$\|b - Ax^k\|_{A^{-1}} = \min_{y \in x^0 + K_k} \|b - Ay\|_{A^{-1}}$$

with the norm

$$\|x\|_{A^{-1}} = (A^{-1}x, x)^{\frac{1}{2}}.$$

Proof. That the CG scheme is a direct scheme follows from Lemma 7.8.

The iterate is given by

$$Q(x^k) = \min_{y \in x^0 + K_k} Q(y),$$

with is equivalent to (7.1). The ansatz

$$x^k = x^0 + \sum_{k=0}^{k-1} \alpha_k d^{k-1} = x^0 + \underbrace{y^{k-1}}_{\in K_{t-1}} + \alpha_{t-1} d^{k-1}$$

yields

$$(b - Ax^k, d^j) = (b - Ay^{k-1}, d_j) - \alpha_{t-1}(Ad^{k-1}, d^j) = 0 \quad \forall j = 0, \dots, t-1$$

that is

$$(b - Ay^{k-1}, d_j) = 0 \; \forall j = 0, \dots, t-2,$$

and therefore $y^{k-1} = x^{k-1}$ and

$$Q(x^k) = \min_{\alpha \in \mathbb{R}} Q(x^{k-1} + \alpha d^{k-1}).$$

Finally, employing symmetry $A = A^T$, we obtain:

$$\begin{aligned} \|b - Ay\|^2_{A^{-1}} = (A^{-1}[b - Ay], b - Ay) &= (Ay, y) - (A^{-1}b, Ay) - (y, b) \\ &= (Ay, y) - 2(b, y), \end{aligned}$$

i.e., the relationship $\|b - Ay\|^2_{A^{-1}} = 2Q(y)$. □

Remark 7.3 (The CG scheme as iterative scheme)**.** *As previososusly mentioned, in pratice the CG scheme is (always) used as iterative method rather than a direct method. Due to round-off errors the search directions are never* 100% *orthogonal.*

Convergence analysis of the CG scheme We now turn our attention to the convergence analysis, which is a nontrivial task. The key is the following characterization of one iteration $x^k = x^0 + K_k$ by

$$x^k = x^0 + p_{k-1}(A) d^0,$$

where $p_{k-1} \in P_{k-1}$ is a polynomial in A:

$$p_{k-1}(A) = \sum_{i=0}^{k-1} \alpha_i A^i$$

The characterization as minimization of Proposition 7.11 can be written as:

$$\|b - Ax^k\|_{A^{-1}} = \min_{y \in x^0 + K_k} \|b - Ay\|_{A^{-1}} = \min_{q \in P_{k-1}} \|b - Ax^0 - Aq(A)d^0\|_{A^{-1}}.$$

When we employ the $\|\cdot\|_A$ norm, we obtain with $d^0 = b - Ax^0 = A(x - x^0)$

$$\|b - Ax^k\|_{A^{-1}} = \|x - x^k\|_A = \min_{q \in P_{k-1}} \|(x - x^0) - q(A)A(x - x^0)\|_A,$$

that is

$$\|x - x^k\|_A = \min_{q \in P_{k-1}} \|[I - q(A)A](x - x^0)\|_A.$$

In the sense of the best approximation property, we can formulate this task as:

$$p \in P_{k-1}: \quad \|[I - p(A)A](x - x^0)\|_A = \min_{q \in P_{k-1}} \|[I + q(A)A](x - x^0)\|_A. \tag{7.5}$$

The characterization as best approximation is key in the convergence analysis of the CG scheme. Let $q(A)A \in P_k(A)$. We seek a polynomial $q \in P_k$ with $q(0) = 1$, such that

$$\|x^k - x\|_A \le \min_{q\in P_k,\, q(0)=1} \|q(A)\|_A \|x - x^0\|_A. \qquad (7.6)$$

The convergence of the CG method is related to the fact whether we can construct a polynomial $q \in P_k$ with $p(0) = 1$ such that the A norm is as small as possible. First, we have:

Lemma 7.12 (Bounds for matrix polynomials)**.** *Let $A \in \mathbb{R}^{n\times n}$ symmetric positive definite with the eigenvalues $0 < \lambda_1 \le \cdots \le \lambda_n$, and $p \in P_k$ a polynomials with $p(0) = 1$:*

$$\|p(A)\|_A \le M, \quad M := \min_{p\in P_k,\, p(0)=1} \sup_{\lambda\in[\lambda_1,\lambda_n]} |p(\lambda)|.$$

Proof. See [25]. □

Employing the previous result and the error estimate (7.6), we can now derive a convergence result for the CG scheme.

Proposition 7.13 (Convergence of the CG scheme)**.** *Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Let $b \in \mathbb{R}^n$ a right hand side vector and let $x^0 \in \mathbb{R}^n$ be an initial guess. Then:*

$$\|x^k - x\|_A \le 2\left(\frac{1-1/\sqrt{\kappa}}{1+1/\sqrt{\kappa}}\right)^k \|x^0 - x\|_A, \quad k \ge 0,$$

with the spectral condition number $\kappa = \mathrm{cond}_2(A)$ of the matrix A.

Remark 7.4. *We see immediately that a large condition number $\kappa \gg 1$ yields*

$$\frac{1-1/\sqrt{\kappa}}{1+1/\sqrt{\kappa}} \to 1$$

and deteriorates significantly the convergence rate of the CG scheme. This is the key reason why preconditioners of the form $P^{-1} \approx A^{-1}$ are introduced that re-scale the system; see Section 7.2.3:

$$\underbrace{P^{-1}A}_{\approx I} x = P^{-1}b.$$

Computations to substantiate these findings are provided in Section 19.3.5.

Proof of Prop. 7.13. From the previous lemma and the estimate (7.6) it follows

$$\|x^k - x\|_A \le M\|x^0 - x\|_A$$

with

$$M = \min_{q\in P_k,\, q(0)=1} \max_{\lambda\in[\lambda_1,\lambda_n]} |q(\lambda)|.$$

We have to find a sharp estimate of the size of M. That is to say, we seek a polynomial $q \in P_k$ which takes at the origin the value 1, i.e., $q(0) = 1$ and which simultaneously has values near 0 in the maximum norm in the interval $[\lambda_1, \lambda_n]$. To this end, we work with the Tschebyscheff approximation (see e.g., [25] and references therein for the original references). We seek a best approximation $p \in P_k$ of the zero function on $[\lambda_1, \lambda_n]$. Such a polynomial should have the property $p(0) = 1$. For this reason, the trivial solution $p = 0$ is not valid. A Tschebyscheff polynomial reads:

$$T_k = \cos\big(k \arccos(x)\big)$$

and has the property:

$$2^{-k-1} \max_{[-1,1]} |T_k(x)| = \min_{\alpha_0,\ldots,\alpha_{k-1}} \max_{[-1,1]} |x^k + \sum_{i=0}^{k-1} \alpha_i x^k|.$$

We choose now the transformation:

$$x \mapsto \frac{\lambda_n + \lambda_1 - 2t}{\lambda_n - \lambda_1}$$

and obtain with

$$p(t) = T_k\left(\frac{\lambda_n + \lambda_1 - 2t}{\lambda_n - \lambda_1}\right) T_k\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)^{-1}$$

a polynomial of degree k, which is minimal on $[\lambda_1, \lambda_n]$ and can be normalized by

$$p(0) = 1.$$

It holds:

$$\sup_{t\in[\lambda_1,\lambda_n]} |p(t)| = T_k\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)^{-1} = T_k\left(\frac{\kappa+1}{\kappa-1}\right)^{-1} \tag{7.7}$$

with the spectral condition:

$$\kappa := \frac{\lambda_n}{\lambda_1}.$$

We now employ the Tschebyscheff polynomials outside of $[-1, 1]$:

$$T_n(x) = \frac{1}{2}\Big[(x + \sqrt{x^2-1})^n + (x - \sqrt{x^2-1})^n\Big].$$

For $x = \frac{\kappa+1}{\kappa-1}$, it holds:

$$\frac{\kappa+1}{\kappa-1} + \sqrt{\left(\frac{\kappa+1}{\kappa-1}\right)^2 - 1} = \frac{\kappa + 2\sqrt{\kappa} + 1}{\kappa - 1} = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}$$

and therefore

$$\frac{\kappa+1}{\kappa-1} - \sqrt{\left(\frac{\kappa+1}{\kappa-1}\right)^2 - 1} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}.$$

Using this relationship, we can estimate (7.7):

$$T_k\left(\frac{\kappa+1}{\kappa-1}\right) = \frac{1}{2}\Big[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\Big] \geq \frac{1}{2}\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k.$$

It follows that

$$\sup_{t\in[\lambda_1,\lambda_n]} T_k\left(\frac{\kappa+1}{\kappa-1}\right)^{-1} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k = 2\left(\frac{1-\frac{1}{\sqrt{\kappa}}}{1+\frac{1}{\sqrt{\kappa}}}\right)^k.$$

This finishes the proof. □

7.2.3 Preconditioning

This section is copy and paste from [25][Section 7.8.1] and translation from German into English.

The rate of convergence of iterative schemes depends on the condition number of the system matrix. For instance, we recall that for second-order operators (such as Laplace) we have a dependence on the mesh size $O(h^{-2}) = O(N)$ (in 2D). For the CG scheme it holds:

$$\rho_{CG} = \frac{1-\frac{1}{\sqrt{\kappa}}}{1+\frac{1}{\sqrt{\kappa}}} = 1 - \frac{2}{\sqrt{\kappa}} + O\left(\frac{1}{\kappa}\right).$$

Preconditioning reformulates the original system with the goal of obtaining a moderate condition number for the modified system. Let $P \in \mathbb{P}^{n\times n}$ be a matrix with

$$P = KK^T.$$

Then:

$$Ax = b \quad \Leftrightarrow \quad \underbrace{K^{-1}A(K^T)^{-1}}_{=:\tilde{A}}\underbrace{K^T x}_{=:\tilde{x}} = \underbrace{K^{-1}b}_{=:\tilde{b}},$$

which is

$$\tilde{A}\tilde{x} = \tilde{b}.$$

In the case of

$$\mathrm{cond}_2(\tilde{A}) \ll \mathrm{cond}_2(A)$$

and if the application of K^{-1} is cheap, then the consideration of a preconditioned system $\tilde{A}\tilde{x} = \tilde{b}$ yields a much faster solution of the iterative scheme. The condition $P = KK^T$ is necessary such that the matrix $\tilde{A}$ keeps its symmetry.

The preconditioned CG scheme (PCG) can be formulated as:

Algorithm 7.14. *Let $A \in \mathbb{R}^{n\times n}$ symmetric positive definite and $P = KK^T$ a symmetric preconditioner. Choosing an initial guess $x^0 \in \mathbb{R}^n$ yields:*

```
1. $r^0 = b - Ax^0$
2. $Pp^0 = r^0$
3. $d^0 = p^0$
4. For $k = 0, 1, \dots$
    (a) $\alpha_k = \frac{(r^k, d^k)}{(Ad^k, d^k)}$
    (b) $x^{k+1} = x^k + \alpha_k d^k$
    (c) $r^{k+1} = r^k - \alpha_k Ad^k$
    (d) $Pp^{k+1} = r^{k+1}$
    (e) $\beta_k = \frac{(r^{k+1}, p^{k+1})}{(r^k, g^k)}$
    (f) $d^{k+1} = p^{k+1} + \beta_k d^k$
```

At each step, we have as additional cost the application of the preconditioner P. We recall that P allows the decomposition into K and K^T even if they are not explicitely used.

We seek P such that

$$P \approx A^{-1}.$$

On the other hand

$$P \approx I,$$

such that the construction of P is not too costly. Obviously, these are two conflicting requirements. Typical preconditioners are:

- *Jacobi preconditioning*
  We choose $P \approx D^{-1}$, where D is the diagonal part of A. It holds

  $$D = D^{\frac{1}{2}}(D^{\frac{1}{2}})^T,$$

  which means that for $D_{ii} > 0$, this preconditioner is admissible. For the preconditioned matrix, it holds

  $$\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \quad \Rightarrow \quad \tilde{a}_{ii} = 1$$

- *SSOR preconditioning*

The SSOR scheme is a symmetric variant of the SOR method (successive over-relaxation) and is based on the decomposition:

$$P = (D + \omega L)D^{-1}(D + \omega U) = \underbrace{(D^{\frac{1}{2}} + \omega L D^{-\frac{1}{2}})}_{K}\underbrace{(D^{\frac{1}{2}} + \omega D^{-\frac{1}{2}} U)}_{=K^T}.$$

For instance, for the Poisson problem, we can find an optimal ω (which is a non-trivial task) such that

$$\mathrm{cond}_2(\tilde{A}) = \sqrt{\mathrm{cond}_2(A)}$$

can be shown. Here, the convergence improves significantly. The number of necessary steps to achieve a given error reduction by a factor of ϵ improves to

$$t_{CG}(\epsilon) = \frac{\log(\epsilon)}{\log(1 - \kappa^{-\frac{1}{2}})} \approx -\frac{\log(\epsilon)}{\sqrt{\kappa}}, \quad \tilde{t}_{CG}(\epsilon) = \frac{\log(\epsilon)}{\log(1 - \kappa^{-\frac{1}{4}})} \approx \frac{\log(\epsilon)}{\sqrt[4]{\kappa}}.$$

Rather than having 100 steps, we only need 10 steps for instance, in case an optimal ω can be found.

7.2.4 Comments on other Krylov space methods such as GMRES and BiCGStab

For non-symmetric systems the CG method cannot be used anymore. In practice in most cases, one works with the GMRES (generalized minimal residuals) method or the BiCGStab (biconjugate gradient stabilized) scheme. Both also belong to Krylov space methods. Typically, both schemes need to be preconditioned in order to yield satisfactory results.

Let $A \in \mathbb{R}^{n\times n}$ be a regular matrix, but not necessarily symmetric. A symmetric version of the problem

$$Ax = b$$

can be achieved by multiplication with A^T:

$$A^T Ax = A^T b.$$

The matrix $B = A^T A$ is positive definite since

$$(Bx, x) = (A^T Ax, x) = (Ax, Ax) = \|Ax\|_2.$$

In principle, we could now apply the CG scheme to $A^T A$. Instead of one matrix-vector multiplication, we would need two such multiplications per step. However, using $A^T A$, the convergence rate will deteriorate since

$$\kappa(B) = \mathrm{cond}_2(A^T A) = \mathrm{cond}_2(A)^2.$$

For this reason, the CG scheme is not really an option.

The GMRES method, *generalized minimal residual* transfers the idea of the CG scheme to general matrices. Using the Krylov space

$$K_k(d^0, A) = \{d^0, Ad^0, \dots, A^{k-1}d^0\},$$

we first construct an orthonormal basis. Employing the GMRES method, orthogonality will be achieved with respect to the Euclidian scalar product:

$$(d^i, d^j) = \delta_{ij}, \quad i, j = 0, \dots, k-1.$$

To work with the A scalar product does not go anymore because A may not be symmetric.

The approximation $x^k \in x^0 + K_k$ is computed with the help of the Galerkin equation:

$$(b - Ax^k, Ad^j) = 0, \quad j = 0, \ldots, k-1. \tag{7.8}$$

The computational cost is higher because a two-step procedure as in the CG scheme cannot be applied for the GMRES scheme. Therefore, the orthogonal basis is constructed with the help of the Arnoldi procedure.

The weak point of the GMRES method is the increasing computational cost with increasing iteration indices because the orthogonalization needs to be re-done until the first step. In practice, often a restart is employed and only a fixed number of search directions N_o is saved. After N_o iterations, a new Krylov space is constructed.

Chapter 8

Convex sets and systems of linear inequalities

Throughout this chapter, all vector spaces will be considered as real.

8.1 Convex sets: definition and first properties

Definition 8.1. *Let X be a (real) vector space. A set $A \subset X$ is said to be convex if*

$$\forall (x,y) \in A^2, \forall \theta \in [0,1], \qquad \theta x + (1-\theta)y \in A.$$

The interpretation is the following. The set $\{\theta x + (1-\theta)y, \theta \in [0,1]\}$ is the line segment joining x to y. The definition means that this line segment is contained in A as soon as x and y belong to A, see fig. 8.1.

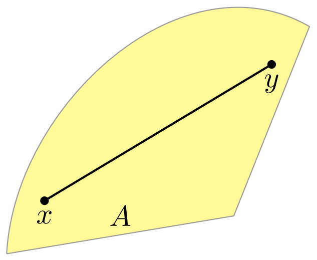


Figure 8.1: Convex set

The following fact is a straightforward consequence of the definition.

Proposition 8.1. *An (arbitrary) intersection of convex sets of X is convex.*

On the contrary, a union of convex sets is not convex in general, see fig. 8.2.

Due to the triangle inequality:

Proposition 8.2. *If X is a normed vector space then all balls (open or closed) of X are convex.*

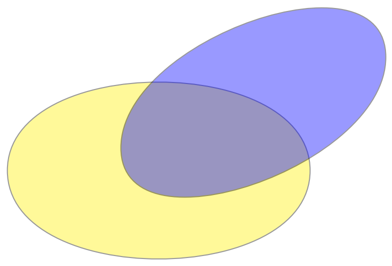


Figure 8.2: Intersection vs union of convex sets

8.2 Convex combinations and convex hull (complement)

8.2.1 Convex combinations

Definition 8.2. *Let X be a vector space. A convex combination of m elements $x_1, ..., x_m \in X$ is an element $y \in X$ that can be decomposed as*

$$y = \sum_{i=1}^{m} \theta_i x_i \qquad \textit{with } (\theta_1, ..., \theta_m) \in (\mathbb{R}_+)^m,\ \sum_{i=1}^{m} \theta_i = 1.$$

In particular a convex combination is a linear combination, but we ask more. In fig. 8.3, the convex combinations of $\{x_1, x_2, x_3\}$ form the triangle.

Proposition 8.3. *If A is convex then any convex combination of elements of A belongs to A.*

Proposition 8.3 is illustrated by fig. 8.3 (take A as the closed triangle).

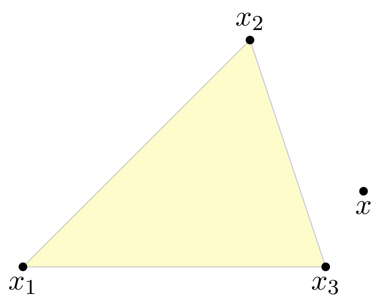


Figure 8.3: x is not a convex combination of $\{x_1, x_2, x_3\}$.

Theorem 8.4 (Carathéodory)**.** *In a vector space of dimension n, all convex combination of m elements, $m > n + 1$, can be written as a convex combination of at most $n + 1$ of these elements.*

Carathéodory's theorem in dimension $n = 2$ is illustrated in fig. 8.4.

8.2.2 Convex hull

Since an intersection of convex sets is a convex set, the following definition makes sense.

Definition 8.3. *Let X be a vector space and A be a subset of X. The convex hull of A, denoted by* conv A*, is the smallest (in the sense of inclusion) convex subset of X containing A. It is the intersection of all convex sets containing A.*

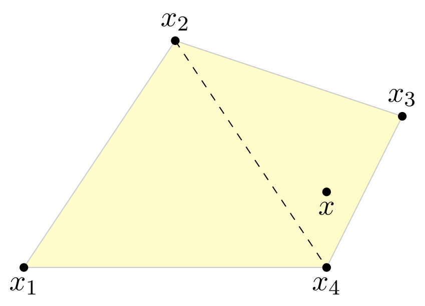


Figure 8.4: Carathéodory's theorem: x is a convex combination of $\{x_1, x_2, x_3, x_4\}$, but it is also a convex combination of $\{x_2, x_3, x_4\}$, or $\{x_1, x_3, x_4\}$.

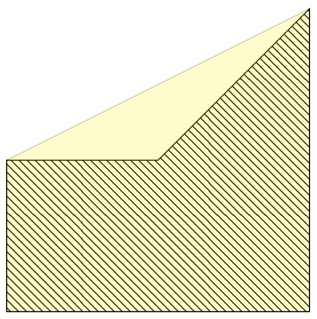


Figure 8.5: A set (hatched) and its convex hull (yellow)

The above construction (see fig. 8.5) can be qualified as *external*. It admits an *internal* counterpart:

Proposition 8.5. *The convex hull of A is the set of all convex combinations of elements of A.*

Proposition 8.5 can be combined with Carathéodory's theorem: if A is a subset of a vector space of dimension n, then the convex hull of A is the set of all convex combinations of at most $n+1$ elements of A, see fig. 8.6.

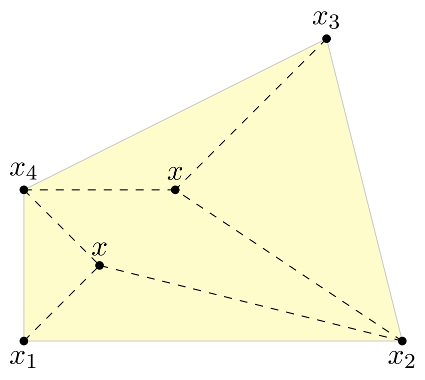


Figure 8.6: The convex hull of $\{x_1, x_2, x_3, x_4\}$ in dimension 2 is the set of all convex combinations of at most 3 vectors taken among $\{x_1, x_2, x_3, x_4\}$.

Proposition 8.6. *Let X, Y be two vector spaces, $A \subset X$, $B \subset Y$. We have*

$$\operatorname{conv}(A \times B) = \operatorname{conv} A \times \operatorname{conv} B.$$

Proposition 8.6 is illustrated in fig. 8.7. A straightforward application is that

$$\operatorname{conv}(\{0,1\}^n) = (\operatorname{conv}\{0,1\})^n = [0,1]^n.$$

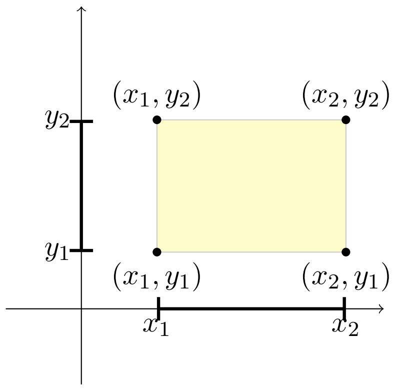


Figure 8.7: The convex hull of $\{x_1, x_2\} \times \{y_1, y_2\} = \{(x_1, y_1), (x_2, y_1), (x_1, y_2), (x_2, y_2)\}$ is $[x_1, x_2] \times [y_1, y_2]$.

8.3 Projection

Theorem 8.7. *Let X be a Hilbert space (typically $\mathbb{R}^n$ equipped with the canonical inner product and the associated Euclidean norm) and A be a nonempty closed convex subset of X. For all $x \in X$ there exists a unique $y \in A$ such that*

$$\|x - y\| = \min_{z \in A} \|x - z\|. \tag{8.1}$$

This element is characterized by the variational inequality (see fig. 8.8)

$$\langle x - y, z - y \rangle \leq 0 \qquad \forall z \in A. \tag{8.2}$$

It is called the projection of x onto A, denoted by $y = P_A(x)$.

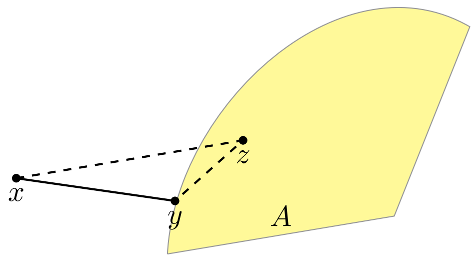


Figure 8.8: Projection onto a convex set

Computing a projection is often difficult (one has to solve an optimization problem), however we discuss below two easy and very useful cases, where the variational inequality (8.2) is easily solved. The first one deals with linear subspaces and corresponds to the standard orthogonal projection, see fig. 8.9..

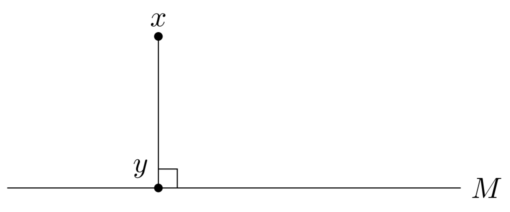


Figure 8.9: Orthogonal projection onto a linear subspace

Proposition 8.8. *Let X be a Hilbert space and M be a closed linear subspace of X (typically $X = \mathbb{R}^n$ and M is a linear subspace of $\mathbb{R}^n$). The projection $y = P_M(x) \in M$ is characterized by*

$$\langle x - y, v \rangle = 0 \qquad \forall v \in M. \tag{8.3}$$

The second case is related to the positive orthant

$$\mathbb{R}^n_+ = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : x_i \geq 0 \ \forall i\},$$

and corresponds to the positive part.

Proposition 8.9. *It holds for all $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$*

$$P_{\mathbb{R}^n_+}(x) = x^+ = (x_1^+, \ldots, x_n^+),$$

with $x_i^+ = \max(0, x_i)$.

8.4 Cones

8.4.1 Definitions

Definition 8.4. *A subset C of a vector space X is said to be a cone if*

$$\forall (\alpha, x) \in \mathbb{R}_+ \times C, \ \alpha x \in C.$$

Cones can be convex or not, see fig. 8.10.

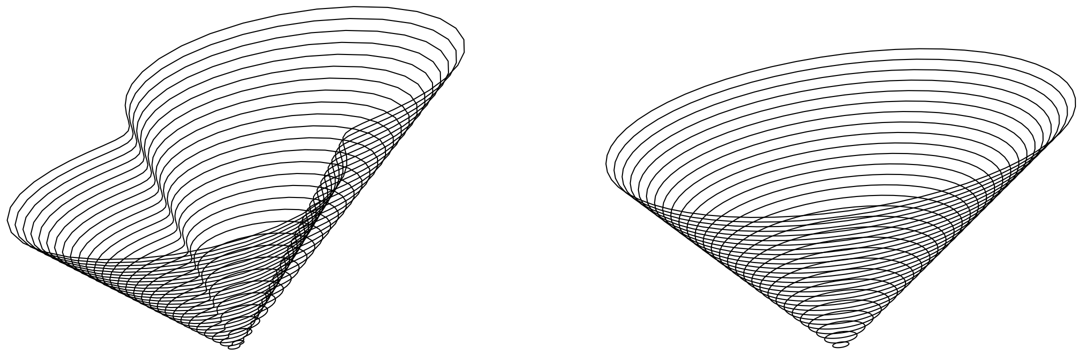


Figure 8.10: Non-convex cone (left) and convex cone (right)

The positive orthant $\mathbb{R}^n_+$ is an example of convex cone.

8.4.2 Normal cone (complement)

Definition 8.5. *Let A be a convex set of a Hilbert space X and let $a \in A$. The normal cone of A at point a is the set*

$$N_A(a) = \{x \in X \ \ s.t. \ \langle x, y - a \rangle \leq 0 \ \forall y \in A\}\,.$$

It generalizes the notion of outward normal, see Fig. 8.11. Clearly, if a is in the interior of A then $N_A(a)$ is reduced to $\{0\}$.

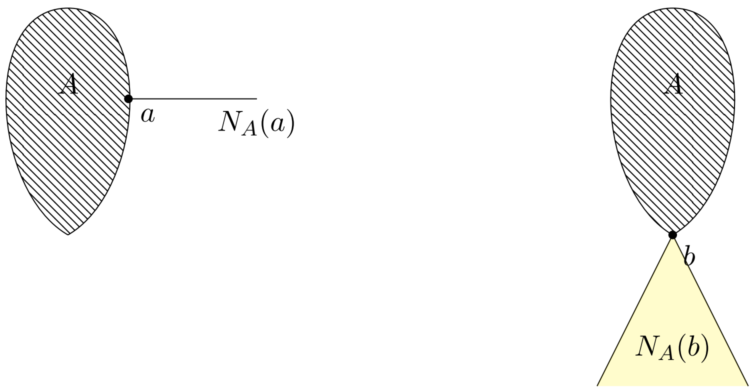


Figure 8.11: Normal cones

8.4.3 Polar cones (complement)

Definition 8.6. *Let C be a cone of a Hilbert space X, with inner product $\langle \cdot, \cdot \rangle$. The positive and negative polar (or dual) cones of C are respectively defined by (see fig. 8.12)*

$$C^+ = \{x \in X \ \ s.t. \ \langle x, y \rangle \geq 0 \ \forall y \in C\}\,, \qquad C^- = \{x \in X \ \ s.t. \ \langle x, y \rangle \leq 0 \ \forall y \in C\}\,.$$

Of course those two cones have similar properties. The negative polar cone is somehow more standard, we will simply call it polar cone. It is clear from the definition that a polar cone is always closed and convex.

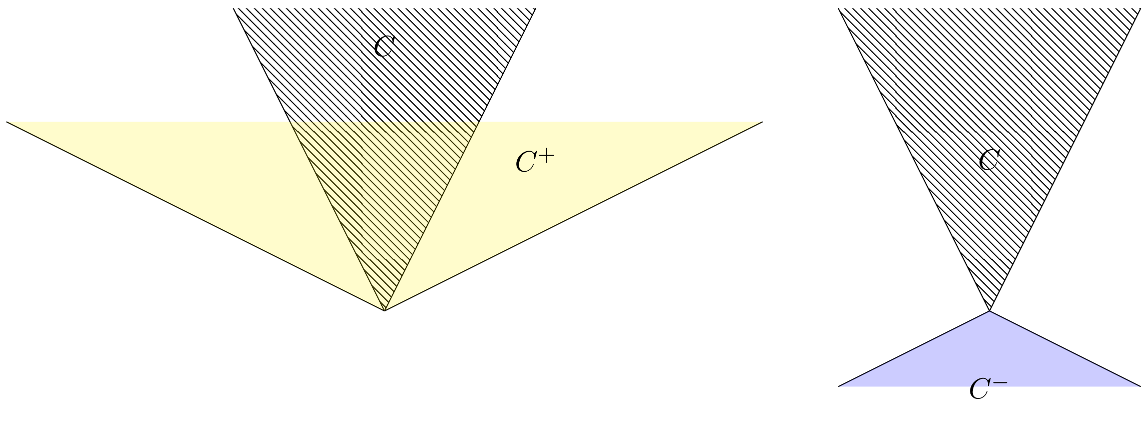


Figure 8.12: Positive and negative polar cones

Example 8.1. *For the positive orthant it is easily checked that*

$$(\mathbb{R}^n_+)^+ = \mathbb{R}^n_+, \qquad (\mathbb{R}^n_+)^- = \mathbb{R}^n_-.$$

Let us now consider reiterated polarity. The definition shows that $C \subset C^{--}$ for every cone C, and even that $\overline{C} \subset C^{--}$ since C^{--} is closed. The following theorem deals with the other inclusion.

Theorem 8.10. *If C is a convex cone of a Hilbert space X then*

$$C^{--} = \overline{C}.$$

8.5 Systems of linear inequalities, introduction to linear programming

8.5.1 Linear inequality systems: Fourier-Motzkin elimination

Here we discuss a practical way of solving linear inequality systems of form $Ax \leq b$, with unknown $x \in \mathbb{R}^n$ and matrix $A \in \mathbb{R}^{m \times n}$. The inequality is meant componentwise. This kind of problem is often known as a feasibility problem: the inequalities are interpreted as constraints and the questions are

1. to know whether it is possible to satisfy all these constraints simultaneously;
2. to describe the feasible set, when it is not empty.

Let

$$C = \{x \in \mathbb{R}^n | Ax \leq b\}.$$

As an intersection of m affine half-spaces (corresponding to each inequality), the set C is convex. Specifically it is called a convex polyhedron. When $b = 0$, C is clearly a cone and we call this a polyhedral cone.

One possibility to compute feasible solutions (in case they exist) is the Fourier-Motzkin algorithm, which is a basic method.

The Fourier-Motzkin elimination works like Gaussian elimination: eliminate from n-variables to $n-1$-variables in an equivalent fashion. This procedure is then recursively applied until 1-variable. Here, we can compute all numbers between the given bounds. By backward substitution we then compute the values of the other variables.

Example 8.2. *Consider the system*

$$\begin{cases} x_1 + x_2 \leq 1 \\ 2x_1 - x_2 \leq 4 \\ x_1 + 2x_2 \leq 8. \end{cases} \tag{8.4}$$

In order to eliminate x_2 we write the equivalent system

$$\begin{cases} x_2 \leq 1 - x_1 \\ x_2 \geq 2x_1 - 4 \\ x_2 \leq 4 - \frac{1}{2}x_1 \end{cases} \Leftrightarrow 2x_1 - 4 \leq x_2 \leq \min(1 - x_1, 4 - \frac{1}{2}x_1).$$

This is possible if and only if

$$\begin{cases} 2x_1 - 4 \leq 1 - x_1 \\ 2x_1 - 4 \leq 4 - \frac{1}{2}x_1 \end{cases} \Leftrightarrow x_1 \leq \frac{5}{3}.$$

We obtain the (nonempty) set of solutions

$$\left\{(x_1, x_2) \mid x_1 \leq \frac{5}{3},\ 2x_1 - 4 \leq x_2 \leq \min(1 - x_1, 4 - \frac{1}{2}x_1)\right\}.$$

See Fig. 8.13.

Remark 8.1. *If we now consider the 'positively homogeneous' system*

$$\begin{cases} x_1 + x_2 \leq 0 \\ 2x_1 - x_2 \leq 0, \end{cases}$$

then we get the set of solutions

$$\{(x_1, x_2) \mid x_1 \leq 0,\ 2x_1 \leq x_2 \leq -x_1\}\,.$$

This is clearly a cone of $\mathbb{R}^2$*, as expected from the structure of the system: a solution remains a solution when it is multiplied by a nonnegative number. See Fig. 8.14*

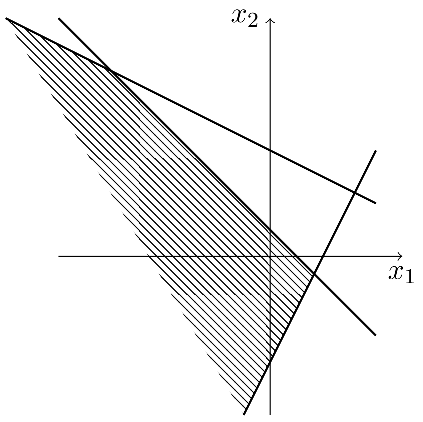


Figure 8.13: Set of solutions in example 8.2 (it is not bounded on the left side).

8.5.2 Conical hull and Farkas lemma (complement)

Here are a few useful results.

Lemma 8.11. *Let* $a_1, \dots, a_m \in \mathbb{R}^n$ *be given vectors. Then, the set*

$$cone(a_1, \dots, a_m) := \{x_1 a_1 + \ldots + x_m a_m | \, x_i \geq 0 \ \forall i = 1, \dots, m\}$$

is a convex cone in $\mathbb{R}^n$*. It is the conical hull of* $a_1, \dots, a_m$*. The vectors* $a_1, \dots, a_m$ *are generators of the cone.*

In the example from Remark 8.1, the vectors $(-1, 1)$ and $(-1, -2)$ are generators of the set of solutions.

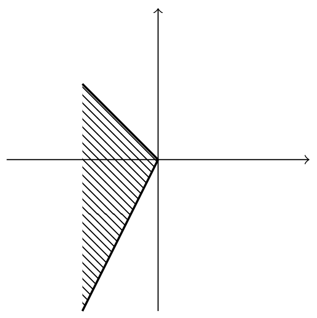


Figure 8.14: Cone of solutions in Remark 8.1

Lemma 8.12. *Let* $A \in \mathbb{R}^{m \times n}$*. Then, the set*

$$K := \{y \in \mathbb{R}^n | y = A^T x, x \geq 0\}$$

(the inequality is meant component-wise) is a non-empty, closed and convex cone.

Lemma 8.13 (Farkas). *Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^n$ be given. Then, the following two statements are equivalent:*

1. *The system $A^T x = b, x \geq 0$ has a solution.*
2. *The inequality $b^T d \geq 0$ holds true for all $d \in \mathbb{R}^n$ with $Ad \geq 0$.*

8.5.3 Introduction to linear programming (complement)

Linear programming deals with optimization problems with linear objective function and linear (or affine) constraints. We provide a brief introduction following [21].

Definition 8.7 (Standard form). *A linear program in standard form is:*

$$\textit{minimize } c^T x, \quad \textit{s.t. } Ax = b, x \geq 0,$$

where $c \in \mathbb{R}^n, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$.

In the above problem the function $x \mapsto c^T x$ is the objective function, the expression $Ax = b$ is the equality constraint and the expression $x \geq 0$ is the inequality constraint.

Other problem statements can be transformed into this form. For instance

$$\text{minimize } c^T x, \quad \text{s.t. } Ax \leq b,$$

is an example in which the constraint $Ax \leq b$ represents a linear inequality system. By using the so-called *slack variable* z inequality constraints can be converted into equalities. Then, we obtain

$$\text{minimize } c^T x, \quad \text{s.t. } Ax + z = b, z \geq 0.$$

However, we do not yet deal with the standard form, because x has no sign. To this end, we must split the vector x into nonnegative and nonpositive parts:

$$x = x^+ - x^-$$

with $x^+ = \max(x, 0)$ and $x^- = \max(-x, 0)$. With these notations, we finally arrive at the following standard form:

$$\text{minimize } \begin{bmatrix} c \\ -c \\ 0 \end{bmatrix}^T \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix} \quad \text{s.t. } \begin{bmatrix} A & -A & I \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix} = b, \quad \begin{bmatrix} x^+ \\ x^- \\ z \end{bmatrix} \geq 0.$$

Remark 8.2. *Further inequality constraints of the form $x \leq u$ or $Ax \geq 0$ can be converted by similar derivations while introducing again slack variables.*

Chapter 9

Exercises

Linear systems

Exercise 9.1. *Solve the following linear systems using the Gauss elimination method.*

$$(a)\begin{cases} 8x-2y+z=3 \\ 12x+y-2z=1 \\ 96x-16y+5z=29, \end{cases}$$

$$(b)\begin{cases} 2x-2y+z-t+u=1 \\ x+2y-z+t-2u=1 \\ 4x-10y+5z-5t+7u=1 \\ 2x-14y+7z-7t+11u=-1, \end{cases}$$

$$(c)\begin{cases} 2x_1+3x_2+5x_3=3 \\ x_1-3x_2-6x_3=2 \\ 3x_1-x_3=4, \end{cases}$$

$$(d)\begin{cases} 2x_1+3x_2-x_3=-1 \\ x_1-5x_2+x_3=6 \\ 3x_1+3x_2+7x_3=0 \\ x_1+2x_2-4x_3=-1. \end{cases}$$

Exercise 9.2. *We consider the system*

$$\begin{cases} 10^{-4}x+y=1 \\ -x+y=2. \end{cases}$$

Solve this system in floating point arithmetic with 3 digits (for the mantissa, in basis 10*)*

1. *without pivoting,*
2. *with row pivoting.*

Conclusion?

Matrix algebra

Exercise 9.3. *Compute the matrix multiplication AB with*

$$A=\begin{pmatrix} 3 & 1 & 2 \\ 4 & 1 & 1 \\ 5 & 2 & 2 \end{pmatrix} \quad \text{and} \quad B=\begin{pmatrix} 7 & 2 & 3 \\ 2 & 2 & 2 \\ 4 & 5 & 6 \end{pmatrix}.$$

Exercise 9.4. *Compute the matrix multiplication AB with*

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 3 & -2 & 4 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & 2 & 11 & 4 \\ -2 & 3 & 0 & 2 \\ 3 & 1 & 4 & 0 \end{pmatrix}.$$

Exercise 9.5. *Consider the matrix*

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

1) Calculate $B \in M_3(\mathbb{R})$ such that $A = 2I + B$.
2) Calculate B^2 and B^3.
3) Deduce an expression of A^n for all $n \in \mathbb{N}$.

Exercise 9.6. *Let m be a real number. When it is possible calculate the inverse of the matrix*

$$A = \begin{pmatrix} 1 & m & -2 \\ 1 & m+1 & m-2 \\ 2 & 2m+1 & 2m-4 \end{pmatrix}.$$

Exercise 9.7. *Consider the matrices*

$$P = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

1) Calculate P^{-1}.
2) Calculate $B = P^{-1}AP$.
3) Show that $A^n = PB^nP^{-1}$for all $n \in \mathbb{N}$. Deduce an expression of A^n.

Linear spaces

Exercise 9.8. *Let (e_1, e_2, e_3) be the canonical basis of $\mathbb{R}^3$. Let a_1, a_2, a_3 be three vectors of $\mathbb{R}^3$ defined by $a_1 = (0, -2, 3)$, $a_2 = (1, 2, 1)$, $a_3 = (3, 0, -4)$.*
1) Show that (a_1, a_2, a_3) is a basis of $\mathbb{R}^3$.
2) Let u be the vector of coordinates $(1, 1, 1)$ in the basis (a_1, a_2, a_3). What are the coordinates of u in the basis (e_1, e_2, e_3)?
3) Let v be the vector of coordinates $(1, 1, 1)$ in the basis (e_1, e_2, e_3). What are the coordinates of v in the basis (a_1, a_2, a_3)?

Determinants

Exercise 9.9. *Calculate the following determinants and give a necessary and sufficient condition on the real numbers a, b, c for d_2 being equal to 0:*

$$d_1 = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{vmatrix}, \qquad d_2 = \begin{vmatrix} 1 & 1 & 1 \\ a & b & c \\ a^2 & b^2 & c^2 \end{vmatrix}.$$

Exercise 9.10. *Calculate the determinant of order n:*

$$\Delta_n = \begin{vmatrix} 1 & 1 & \dots & \dots & 1 \\ 1 & 1 & 0 & \dots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 1 & 0 \\ 1 & 0 & \dots & 0 & 1 \end{vmatrix}.$$

Exercise 9.11. *Let $a \neq b$ be two real numbers. Calculate the determinant of order n*

$$\Delta_n = \begin{vmatrix} a+b & ab & 0 & \dots & 0 \\ 1 & a+b & ab & \ddots & \vdots \\ 0 & 1 & a+b & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & ab \\ 0 & \dots & 0 & 1 & a+b \end{vmatrix}.$$

Indication: show by induction that $\Delta_n = \frac{a^{n+1}-b^{n+1}}{a-b}$.

LU decomposition (complement)

Exercise 9.12. *Let the matrix A and the right hand side vector b given as follows:*

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 4 & 3 & 2 \\ 0 & 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 5 \\ 2 \end{pmatrix}.$$

1. *Construct the LU decomposition of the matrix A.*
2. *Solve $Ax = b$ with the LU decomposition.*
3. *Use the Jacobi and Gauss-Seidel iterative methods and evaluate by hand the first two steps. Take as initial guess vector $x^0 = (0, 0, 0)$.*

Diagonalization

Exercise 9.13. *Let*

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 4 \end{pmatrix}.$$

1) Determine the eigenvalues of A and show that A is diagonalizable on $\mathbb{R}$.
2) Diagonalize A (give the diagonal form and the transition matrix).
3) Calculate A^n for all $n \in \mathbb{N}$.

Exercise 9.14. *1) Calculate the complex eigenvalues of the matrices*

$$A = \begin{pmatrix} 0 & 1 & -6 \\ 1 & 0 & 2 \\ 1 & 0 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix}, \qquad C = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 1 & -1 \\ 0 & 2 & 4 \end{pmatrix}.$$

2) Show that A is diagonalizable on $\mathbb{C}$ and B is diagonalizable on $\mathbb{R}$.
3) What are the spectral radii of A, B and C?

Normed spaces

Exercise 9.15 (complement)**.** *We define on $\mathcal{M}_n(\mathbb{R})$ the map*

$$A = (a_{ij}) \mapsto N(A) = n \max_{ij} |a_{ij}|.$$

Show that this defines a matrix norm.

Exercise 9.16 (complement)**.** *Let $A \in \mathcal{M}_n(\mathbb{R})$ be symmetric positive definite. We define the map*

$$(x, y) \in \mathbb{R}^n \times \mathbb{R}^n \mapsto \langle x, y\rangle_A = \langle Ax, y\rangle,$$

where $\langle\cdot,\cdot\rangle$ is the canonical inner product of $\mathbb{R}^n$.
1. Show that this defines an inner product on $\mathbb{R}^n$.
2. We denote by $\|\cdot\|_A$ the associated norm. Give tight constants α, β, in terms of the eigenvalues of A, such that

$$\alpha\|x\|_2 \le \|x\|_A \le \beta\|x\|_2 \qquad \forall x \in \mathbb{R}^n.$$

Exercise 9.17. *We consider the space $\mathcal{C}([0,1],\mathbb{R})$ of continuous functions from $[0,1]$ into $\mathbb{R}$. Show that the quantities*

$$\|f\|_1 = \int_0^1 |f(t)|dt, \qquad \|f\|_2 = \sqrt{\int_0^1 |f(t)|^2 dt}, \qquad \|f\|_\infty = \sup_{t\in[0,1]} |f(t)|$$

define norms such that

$$\|f\|_1 \le \|f\|_2 \le \|f\|_\infty.$$

Remarks. 1. One can show that these norms are nevertheless not equivalent.
2. Knowing how to prove the triangle inequality for $\|\cdot\|_2$ is not mandatory.

Stability of linear systems and iterative methods (complement)

Exercise 9.18. *We perform a stability analysis of the following linear system. Let*

$$A = \begin{pmatrix} 6 & -2 \\ 11.5 & -3.85 \end{pmatrix}, \quad b = \begin{pmatrix} 10 \\ 17 \end{pmatrix} \quad \text{and} \quad \tilde{A} = \begin{pmatrix} 6 & -2 \\ 11.5 & -3.84 \end{pmatrix}.$$

In this exercise you may ***perform the calculations using a computer*** *and a software such as Python, Matlab, or Octave.*

1. *Compute the solutions x and $\tilde{x}$ of the systems*

$$Ax = b, \quad \tilde{A}\tilde{x} = b.$$

2. *What do you observe by comparing x and $\tilde{x}$ and comparing the matrices A and $\tilde{A}$?*

3. *We know the stability theorem:*

$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \le cond(A)\frac{\|A - \tilde{A}\|}{\|A\|}. \tag{9.1}$$

   Compute the factors

$$\text{cond}_2(A) = \|A^{-1}\|_2\|A\|_2$$

   and

$$\frac{\|A - \tilde{A}\|_2}{\|A\|_2}.$$

4. *Compute the right hand side of the estimate* (9.1) *and interprete the result in view of the relative error of the solutions x und $\tilde{x}$, i.e.,*

$$\frac{\|x - \tilde{x}\|_2}{\|\tilde{x}\|_2}.$$

5. *Is the lastly obtained solution plausible?*

6. **(optional alternative by hand)** *You may replace the spectral norm by the maximal row sum norm, which is defined as* $\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^n |a_{ij}|$ *for* $A \in \mathbb{R}^{n\times n}$*. Then, you can re-do steps No. 1-5 again by hand. You may finally also compare the results using the* $\|\cdot\|_2$ *and* $\|\cdot\|_\infty$*.*

Exercise 9.19. *Implement (in a software) the descent method, Jacobi and Gauss-Seidel approaches for solving*

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ -3 \\ 4 \end{pmatrix}$$

As initial guess take $x^0 = 0$ *and compute for each method* 10 *steps. As information, the exact solution is* $x = (1, 0, 2)^T$*.*

Convex sets

Exercise 9.20. *Show that the simplex*

$$S_n = \left\{ x = (x_1, ..., x_n) \in \mathbb{R}^n_+ \ \ s.t. \ \sum_{i=1}^n x_i = 1 \right\},$$

(see fig. 9.1) is a convex set.

Complement: show that $S_n = \operatorname{conv}\{e_1, ..., e_n\}$*, where* $(e_1, ..., e_n)$ *is the canonical basis of* $\mathbb{R}^n$*.*

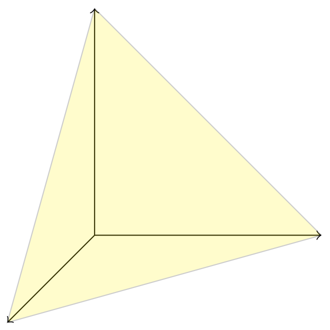


Figure 9.1: Simplex of $\mathbb{R}^3$

Exercise 9.21. *Let* $v_1, ..., v_n$ *be arbitrary vectors of a vector space. Show that*

$$K := \left\{ \sum_{i=1}^n \alpha_i v_i, \ \alpha_i \geq 0 \ \forall i \right\}.$$

is a convex cone.

Complement: Show that

$$K = \operatorname{conv}\left(\bigcup_{i=1}^n h_i \right)$$

with the half-lines

$$h_i = \mathbb{R}_+ v_i = \{t v_i, t \geq 0\}.$$

Exercise 9.22. *Let*

$$M = \left\{ x = (x_1, \dots, x_n) \in \mathbb{R}^n : \sum_{i=1}^n x_i = 0 \right\}.$$

Describe the projection operator onto M*.*

Linear inequalities

Exercise 9.23. *Using the Fourier-Motzkin elimination, find out whether these systems admit solutions:*

$$(a)\ \begin{cases} x_1 + 2x_2 \leq 4 \\ -2x_1 - x_2 \leq 2 \\ x_1 - x_2 \leq 1, \end{cases} \qquad (b)\ \begin{cases} x_1 + 2x_2 \leq 2 \\ -2x_1 - x_2 \leq 4 \\ 4x_1 + x_2 \leq -16. \end{cases}$$

Part III

Functions and related numerical notions

Chapter 10

Functions of one or several variables

10.1 Basic concepts

A real-valued or complex-valued function on a set U is a mapping f which associates to each $x \in U$ a unique number $f(x) \in \mathbb{K}$, $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. We write

$$\begin{array}{rcl} f : U & \to & \mathbb{K} \\ x & \mapsto & f(x). \end{array}$$

The set U is the definition set and $f(U) := \{f(x) \in \mathbb{K} : x \in U\}$ is the image set. The graph of f is the set

$$G(f) := \{(x, f(x)) : x \in U\} \subset U \times \mathbb{K}.$$

We will mainly work in the situation where U is a subset of $\mathbb{R}^d$, hence the variable is represented by a point in $\mathbb{R}^d$, denoted by

$$x = (x_1, \ldots, x_d),$$

actually there are d independent space variables. For instance they can be space and time variables. The canonical inner product in $\mathbb{R}^d$ is denoted by $\langle x, y\rangle = x \cdot y = \sum_{i=1}^d x_i y_i$.

We can similarly consider vector-valued functions, namely functions taking values in an arbitrary vector space. Actually, studying

$$\left\{ \begin{array}{rcl} f : U & \to & \mathbb{R}^m \text{ (or } \mathbb{C}^m) \\ x & \mapsto & f(x) = (f_1(x), \ldots, f_m(x)) \end{array} \right.$$

amounts to studying the m functions $f_1, \ldots, f_m$. This is why the number of variables deserves much more mathematical care than the number of values, at least when this number is finite. We also insist on the fact that we will only consider real variables. Dealing with complex variables is a completely different story!

10.2 Differentiation

10.2.1 Partial derivatives

Let f be a (scalar, vector or matrix-valued) function of a variable $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$. We recall that the partial derivative of f with respect to one of these variables is obtained after differentiating f (i.e. each component or entry of f) with respect to this variable, the other variables being fixed. We frequently use the notations:

$$\frac{\partial f}{\partial x_i} = \partial_{x_i} f \qquad \text{(partial derivative w.r.t. } x_i),$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \partial_{x_i x_j} f \qquad \text{(second partial derivative w.r.t. } x_j \text{ then } x_i),$$

$$\frac{\partial^2 f}{\partial x_i \partial x_i} = \partial_{x_i}^2 f \qquad \text{(second partial derivative w.r.t. } x_i\text{).}$$

10.2.2 Fréchet derivative, Jacobian matrix

There are several notions of differentiability, here we will merely use the most classical notion of Fréchet differentiability. This concept can be defined for functions from an arbitrary real vector space to an arbitrary real or complex vector space, however for simplicity we restrict ourselves here to functions from $\mathbb{R}^n$ to $\mathbb{R}^m$.

Definition 10.1. *Let $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ and $x \in U$. We say that f is Fréchet differentiable at point x if there exists $L \in \mathcal{M}_{mn}(\mathbb{R})$ such that*

$$\lim_{h\to 0} \frac{\|f(x+h) - f(x) - Lh\|}{\|h\|} = 0.$$

In this case the matrix L is unique, it is called the Jacobian matrix of f at x, denoted by $Df(x)$ or $df(x)$ or $Jf(x)$. The linear map $y \in \mathbb{R}^n \mapsto Lx \in \mathbb{R}^m$ is the Fréchet derivative of f at point x.

If the map $x \in U \mapsto Df(x)$ is continuous we say that f is continuously Fréchet differentiable.

Note that due to the equivalence of norms (Theorem 6.3), any norm on $\mathbb{R}^m$ and $\mathbb{R}^n$ can be indifferently chosen.

The Jacobian matrix can be inferred from the partial derivatives.

Proposition 10.1. *1. If $f : x = (x_1, ..., x_n) \in U \subset \mathbb{R}^n \mapsto f(x) = (f_1(x), ..., f_m(x)) \in \mathbb{R}^m$ is Fréchet differentiable at $x \in U$ then*

$$Df(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \dots & \frac{\partial f_m}{\partial x_n}(x) \end{pmatrix}.$$

2. If all partial derivatives of f exist and are continuous functions on U, then f is continuously Fréchet differentiable.

10.2.3 Chain rule

Next, we introduce the chain rule for differentiating compositions of functions. Essentially, it says that the Jacobian matrix of a composition is the product of the Jacobian matrices.

Theorem 10.2. *Let $f : U \subset \mathbb{R}^n \to V \subset \mathbb{R}^m$ and $g : V \to \mathbb{R}^p$ be Fréchet differentiable in a and $b = f(a)$, respectively. Set $h = g \circ f$, i.e. $h(x) = g(f(x))$. Then h is Fréchet differentiable in a with*

$$Dh(a) = Dg(b)Df(a).$$

Here is a special case.

Corollary 10.3. *Let $f : I \subset \mathbb{R} \to V \subset \mathbb{R}^m$ be differentiable in a and $g : V \to \mathbb{R}$ be differentiable in $b = f(a)$. Then, $h := g \circ f$ is differentiable in a with the derivative*

$$h'(a) = Dg(b)f'(a) = \sum_{i=1}^{m} \partial_i g(b)\ f_i'(a).$$

For instance in physics, if a quantity depends on the temperature and the pressure, say $g = g(T, P)$, while T, P both depend on the space position x (say in 1d for simplicity), then we write $h(x) = g(T(x), P(x))$ and

$$\frac{dh}{dx} = \frac{\partial g}{\partial T}\frac{dT}{dx} + \frac{\partial g}{\partial P}\frac{dP}{dx}.$$

10.2.4 Schwarz' theorem

We now provide a result for second-order partial derivatives and when they can be interchanged.

Theorem 10.4 (Schwarz)**.** *Let the function f have in a neighborhood of $a \in \mathbb{R}^n$ the partial derivatives $\partial_i f, \partial_j f$ and $\partial_{ji} f$. Let $\partial_{ji} f$ be continuous in a. Then, there exists $\partial_{ij} f$ and it holds*

$$\partial_{ij} f(a) = \partial_{ji} f(a).$$

10.2.5 Multiindex notation (complement)

For a general description of ODEs and PDEs the multiindex notation is commonly used.

- A multiindex is a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, where each component $\alpha_i \in \mathbb{N}_0$. The order is

$$|\alpha| = \alpha_1 + \ldots + \alpha_n.$$

- For a given multiindex we define the partial derivative:

$$D^\alpha u(x) := \partial_{x_1}^{\alpha_1} \cdots \partial_{x_n}^{\alpha_n} u.$$

  If $k \in \mathbb{N}_0$, we define the set of all partial derivatives of order k:

$$D^k u(x) := \{D^\alpha u(x) : |\alpha| = k\}.$$

Example 10.1. *Let the problem dimension $n = 3$. Then, $\alpha = (\alpha_1, \alpha_2, \alpha_3)$. For instance, let $\alpha = (2, 0, 1)$. Then $|\alpha| = 3$ and $D^\alpha u(x) = \partial_x^2 \partial_z^1 u(x)$.*

10.3 Classical differential operators

Well-known in physics, it is convenient to work with the **nabla-operator** to define derivative expressions:

$$\nabla = \begin{pmatrix} \partial_{x_1} \\ \vdots \\ \partial_{x_d} \end{pmatrix}.$$

Very often in applications, when space and time variables are involved, the classical differential operators apply to the space variables only.

The gradient of a single-valued function is the vector of partial derivatives

$$\nabla u = \begin{pmatrix} \partial_{x_1} u \\ \vdots \\ \partial_{x_d} u \end{pmatrix}.$$

The gradient of a vector-valued function $v : \mathbb{R}^d \to \mathbb{R}^n$ reads:

$$\nabla v = \begin{pmatrix} \partial_{x_1} v_1 & \ldots & \partial_{x_d} v_1 \\ \vdots & & \vdots \\ \partial_{x_1} v_n & \ldots & \partial_{x_d} v_n \end{pmatrix}.$$

It corresponds exactly with the Jacobian matrix! Note the (universal but a little confusing) fact that for single-valued functions the gradient is the transpose of the Jacobian matrix (a row matrix).

The divergence is defined for vector-valued functions $v : \mathbb{R}^d \to \mathbb{R}^d$ by

$$\text{div } v := \nabla \cdot v := \nabla \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \sum_{k=1}^{d} \partial_{x_k} v_k.$$

The divergence for a matrix-valued function $\sigma : \mathbb{R}^d \to \mathcal{M}_d(\mathbb{R})$ is defined as:

$$\nabla \cdot \sigma = (\sum_{j=1}^{d} \frac{\partial \sigma_{ij}}{\partial x_j})_{1 \le i \le d}.$$

The Laplace operator of a scalar-valued function $u : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$\Delta u = \sum_{k=1}^{d} \partial_{x_k x_k} u.$$

For a vector-valued function $u : \mathbb{R}^d \to \mathbb{R}^n$, we define the Laplace operator component-wise as

$$\Delta u = \Delta \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{d} \partial_{x_k x_k} u_1 \\ \vdots \\ \sum_{k=1}^{d} \partial_{x_k x_k} u_n \end{pmatrix}.$$

Let us recall the **cross product** of two vectors $u, v \in \mathbb{R}^3$:

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix}.$$

With the help of the cross product, we can define the **rotational** of $v : \mathbb{R}^3 \to \mathbb{R}^3$ by

$$\operatorname{curl} v = \nabla \times v = \begin{pmatrix} \partial_{x_1} \\ \partial_{x_2} \\ \partial_{x_3} \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \partial_{x_2} v_3 - \partial_{x_3} v_2 \\ \partial_{x_3} v_1 - \partial_{x_1} v_3 \\ \partial_{x_1} v_2 - \partial_{x_2} v_1 \end{pmatrix}.$$

10.4 Taylor expansions, Hessian matrix

If $f : \mathbb{R} \to \mathbb{R}$ is of class $\mathcal{C}^n$ around some point a then we have the classical Taylor-Young formula:

$$f(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j + o(|x-a|^n).$$

The remainder $o(|x-a|^n)$ (Landau notation) is a function such that

$$\lim_{x \to a} \frac{o(|x-a|^n)}{|x-a|^n} = 0.$$

There is also the Taylor-Lagrange formula, if f is of class $\mathcal{C}^{n+1}$:

$$f(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j + \frac{f^{(n+1)}(y)}{(n+1)!}(x-a)^{n+1},$$

for some intermediate point y between a and x.

For functions of several variables we limit ourselves to the Taylor-Young formula of order 2.

Proposition 10.5. *Let $f : U \subset \mathbb{R}^n \to \mathbb{R}$ be a C^2 function. Let $a, x \in U$ points such that the line segment joining a and x is in U. Then*

$$f(x) = f(a) + Df(a)(x-a) + \frac{1}{2}(x-a)^T D^2 f(a)(x-a) + o(\|x-a\|^2),$$

with $Df(a)$ the Jacobian (row) matrix of f at a and $D^2 f(a)$ (or $\nabla^2 f(a)$, or $f''(a)$) the Hessian (square) matrix of f at a given by

$$D^2 f(a) = \begin{pmatrix} \partial_{x_1 x_1} f(a) & \dots & \partial_{x_1 x_n} f(a) \\ & \vdots & \\ \partial_{x_n x_1} f(a) & \dots & \partial_{x_n x_n} f(a) \end{pmatrix}.$$

Remark 10.1. *1. Under the assumptions made the Hessian matrix is symmetric (Schwarz theorem 10.4).*
2. The first order term can be rewritten as

$$Df(a)(x-a) = \langle \nabla f(a), x-a \rangle.$$

10.5 Convex functions

10.5.1 Definitions

A geometrical way to define the convexity of a function is through the notion of epigraph. This point of view is very useful in convex analysis, a branch of mathematics related to optimization. Again we place ourselves in the framework of real vector spaces, mostly $\mathbb{R}^n$.

Definition 10.2. *Let X be a vector space and $f : X \to \mathbb{R}$. The epigraph of f is the set (see fig. 10.1)*

$$\operatorname{epi} f = \{(x,\xi) \in X \times \mathbb{R} \ \ s.t. \ \ \xi \geq f(x)\}.$$

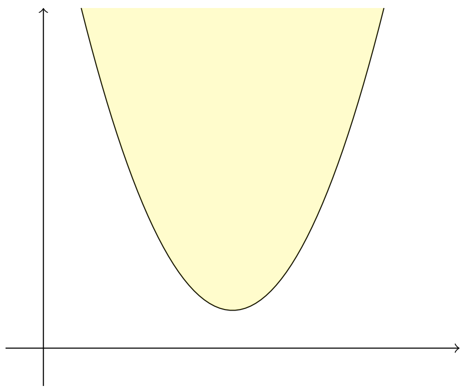


Figure 10.1: Epigraph

Definition 10.3. *Let X be a vector space and $f : X \to \mathbb{R}$. The function f is said to be convex if its epigraph is convex.*

The following characterization is extremely standard, see fig. 10.2.

Proposition 10.6. *A function $f : X \to \mathbb{R}$ is convex if and only if*

$$\forall x, y \in X, \ \forall \theta \in (0,1), \qquad f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y). \tag{10.1}$$

A little refinement is the notion of strict convexity.

Definition 10.4. *A function $f : X \to \mathbb{R}$ is strictly convex if and only if*

$$\forall x, y \in X, x \neq y, \ \forall \theta \in (0,1), \qquad f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y).$$

10.5.2 Characterization

Proposition 10.7. *Let $f : X = \mathbb{R}^n \to \mathbb{R}$ be a Fréchet differentiable function. The following assertions are equivalent:*

(i) f *is convex;*

(ii) $f(y) - f(x) \geq \langle \nabla f(x), y-x \rangle \qquad \forall (x,y) \in X^2$;

(iii) $\langle \nabla f(y) - \nabla f(x), y-x \rangle \geq 0 \qquad \forall (x,y) \in X^2$;

(iv) $D^2 f(x) \geq 0$ *(positive semi-definite)* $\qquad \forall x \in X$, *when* f *is* $\mathcal{C}^2$.

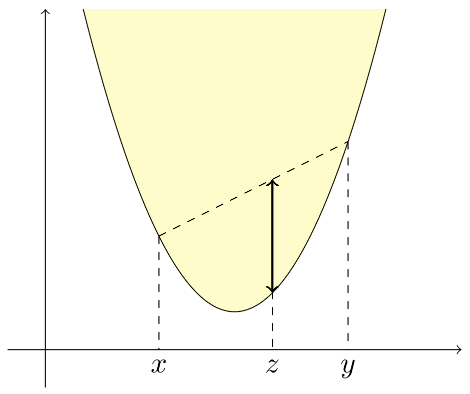


Figure 10.2: Algebraic characterization of convex functions: if $z = \theta x + (1-\theta)y$ for $0 < \theta < 1$ then $f(z) \le \theta f(x) + (1-\theta) f(y)$.

10.6 Integration by parts

10.6.1 Substitution rule

One of the most important formulas in calculus, numerical mathematics, calculus of variations, and continuum mechanics is the substitution rule that allows to transform integrals from one domain to another.

In 1D

Proposition 10.8. *Let $I = [a, b]$ be given. To transform this interval to a new interval, we use a mapping $T : [a, b] \to [\alpha, \beta]$ with $T(a) = \alpha$ and $T(b) = \beta$. If T is of class $\mathcal{C}^1$ we have the transformation rule:*

$$\int_\alpha^\beta f(y)\,dy = \int_{T(a)}^{T(b)} f(y)\,dy = \int_a^b f(T(x))\,T'(x)\,dx.$$

Remark 10.2. *In case that $T(a) = \beta$ and $T(b) = \alpha$ the previous proposition still holds true, but with a negative sign:*

$$\int_\alpha^\beta f(y)\,dy = \int_{T(b)}^{T(a)} f(y)\,dy = -\int_{T(a)}^{T(b)} f(y)\,dy = \int_a^b f(T(x))\,(-T'(x))\,dx.$$

Both cases can be summarized as:

Theorem 10.9. *Let $I = [a, b]$ be given. To transform this interval to a new interval $[\alpha, \beta]$, we employ a monotone mapping T such that $T([a, b]) = [\alpha, \beta]$. If T is of class $\mathcal{C}^1$, it holds:*

$$\int_{T(I)} f(y)\,dy := \int_\alpha^\beta f(y)\,dy = \int_a^b f(T(x))\,|T'(x)|\,dx = \int_I f(T(x))\,|T'(x)|\,dx.$$

Remark 10.3. *We observe the formal relation between the integration increments:*

$$dy = |T'(x)|\,dx.$$

In higher dimensions

We have the following generalization of the substitution rule in arbitrary dimension (also known as **change of variables** under the integral):

Theorem 10.10. *Let $\Omega \subset \mathbb{R}^n$ be an open, measurable, domain. Let the function $T : \Omega \to \mathbb{R}$ be of class C^1, one-to-one (injective) and Lipschitz continuous. Then:*

- *The domain $\hat{\Omega} := T(\Omega)$ is measurable.*
- *The function $f(T(\cdot))|\det T'(\cdot)| : \Omega \to \mathbb{R}$ is (Riemann)-integrable.*
- *For all measurable subdomains $M \subset \Omega$ it holds the substitution rule:*

$$\int_{T(M)} f(y)\, dy = \int_M f(T(x))|\det T'(x)|\, dx,$$

and in particular as well for $M = \Omega$.

Remark 10.4. *The determinant of the Jacobian matrix is sometimes called Jacobian determinant. The formal relation between increments then reads $dy = |\det T'(x)|dx$. This formula is usually stated and proved in the framework of measure theory, thus it involves particular notions of regularity. Here and in the next sections we do not specify the regularity assumptions, which are anyway very mild.*

10.6.2 Integration by parts and Green's formulae

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain with 'smooth' boundary $\partial\Omega$. We denote by n its outward unit normal vector. One of the most important formula in applied mathematics, physics and continuum mechanics is **integration by parts**, which for $u, v : \mathbb{R}^d \to \mathbb{R}$ reads:

$$\int_\Omega \nabla u v\, dx = -\int_\Omega u \nabla v\, dx + \int_{\partial\Omega} uvn\, ds.$$

We can obtain further results, which are very useful, directly based on the integration by parts. For instance the following one is at the basis of the finite element method:

$$\int_\Omega \nabla u \cdot \nabla v\, dx = -\int_\Omega \Delta u\, v\, dx + \int_{\partial\Omega} v\, \partial_n u\, ds.$$

Here, $\partial_n u := \nabla u \cdot n$ is called normal derivative. Very much related and well-known in physics is the divergence formula, valid for $u : \mathbb{R}^d \to \mathbb{R}^d$:

$$\int_\Omega \operatorname{div} u\, dx = \int_{\partial\Omega} u \cdot n\, ds.$$

Chapter 11

Introduction to nonlinear optimization

11.1 General concepts

11.1.1 Brief classification of optimization problems

Optimization aims at finding points that minimize or maximize a given function $f : U \to V$. We distinguish between single objective optimization when $V = \mathbb{R}$ and multi-objective optimization when V is a more general vector space equipped with an order relation. We will place ourselves in the first class, and up to a change of sign we will focus on minimization problems. As to the admissible (or feasible) set U, it is usually either a "smooth enough" subset of a vector space (continuous optimization), or a countable set (discrete optimization). We focus on the first category. Further, we speak of unconstrained optimization if U is the full vector space, and of constrained optimization otherwise. We mainly develop the first one. In the case where f is a convex function and U is a convex set we are in the field of convex optimization.

11.1.2 Problem setting

As said above, we consider an unconstrained continuous optimization problem of form

$$\underset{x\in X}{\text{minimize}}\ f(x) \tag{11.1}$$

where $f : X \to \mathbb{R}$ is an arbitrary function and X is a normed vector space. This function is usually called *cost function*, or *objective function*, or *criterion*. Solving (11.3) not only means finding the value of the minimum (or infimum), but also finding minimizers, if there are some, namely points where the minimum is attained.

11.1.3 Global and local minimizers

Definition 11.1. *We say that $a \in X$ is a global minimizer of $f : X \to \mathbb{R}$ if*

$$f(a) \leq f(x) \qquad \forall x \in X.$$

Definition 11.2. *We say that $a \in X$ is a local minimizer of $f : X \to \mathbb{R}$ if there exists $\varepsilon > 0$ such that*

$$f(a) \leq f(x) \qquad \forall x \in B_\varepsilon(a),$$

where $B_\varepsilon(a)$ is the (open) ball of center a and radius ε. In words, a minimizes f over a neighborhood of a.

An illustration is given in fig. 11.1.

Obviously the problem stated in (11.3) deals with global minima. However, the conditions of optimality we are going to write as well as the output of most algorithms do not allow to distinguish between global and local minima. Finding global minima is often a very difficult task, requiring

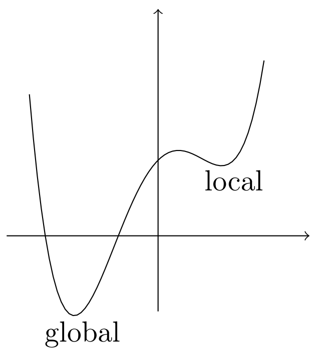


Figure 11.1: Global vs local minimizer

extremely costly numerical methods. We will see later that these bothering facts disappear under convexity assumptions.

More finely, when in definitions 11.1 and 11.2 the inequalities can be made strict for $x \neq a$ we say that a is a strict global / local minimizer. In Figure 11.2 we plot a function which has all these different minima:

$$f(x) = \begin{cases} \sin(4x) + \sin(x) + \frac{1}{4}x^2 & x < x_* \\ \sin(4x_*) + \sin(x_*) + \frac{1}{4}x_*^2 & x_* \leq x \leq x_* + \frac{1}{2} \\ \sin(4(x - \frac{1}{4})) + \sin(x - \frac{1}{4}) + \frac{1}{4}(x - \frac{1}{4})^2 & x > x_* \end{cases}$$

with $x_* \approx 2.72086$.

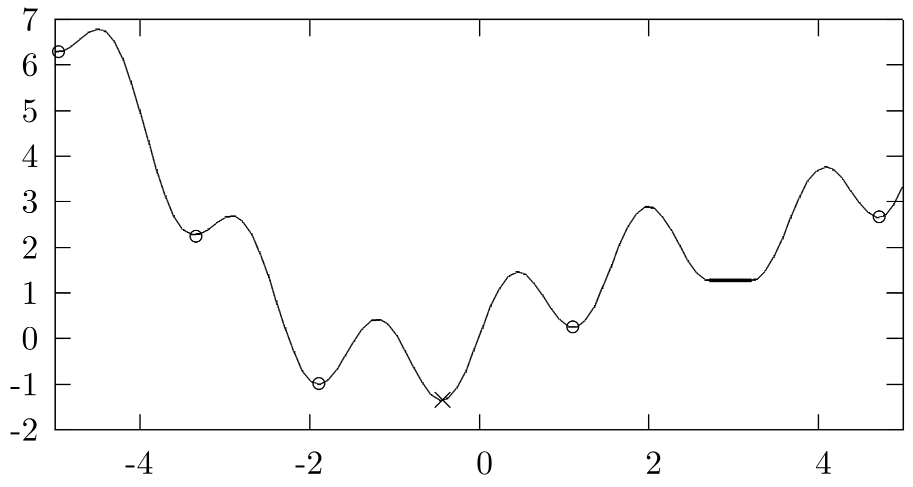


Figure 11.2: Function with different minima. With $\circ$, we denote strict local minima, with $\times$ the global (strict) minimum and the fat region characterizes a non-strict local minimium.

11.1.4 Optimality conditions

For the construction and the justification of numerical algorithms, we introduce the optimality conditions. They are essentially consequences of the Taylor-Young formula (Proposition 10.5).

Theorem 11.1 (Necessary optimality conditions of first and second order)**.**

- *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and $x \in \mathbb{R}^n$ be local minimizer of f. Then it holds the first-order necessary condition*

$$\nabla f(x) = 0.$$

We say that x is a critical or stationary point.

- *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a two times continuously differentiable function. Let $x \in \mathbb{R}^n$ be a local minimizer of f. Then it holds the necessary second order optimality condition:*

$$D^2 f(x) \geq 0$$

(the Hessian is positive semi-definite).

We next state sufficient optimality conditions:

Theorem 11.2 (Sufficient optimality condition of second order)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a two times continuously differentiable function and $x \in \mathbb{R}^n$. If $\nabla f(x) = 0$ (critical point) and $D^2 f(x) > 0$ (the Hessian is positive definite) then x is a* strict local minimizer*.*

Definition 11.3. *A stationary point which neither a local minimizer nor a local maximizer is called local saddle point.*

Recall that, given a symmetric matrix A it holds

- $A > 0 \Leftrightarrow$ all eigenvalues are > 0
- $A < 0 \Leftrightarrow$ all eigenvalues are < 0
- $A \geq 0 \Leftrightarrow$ all eigenvalues are ≥ 0
- $A \leq 0 \Leftrightarrow$ all eigenvalues are ≤ 0.

In dimension is $n = 2$ we have the following criterion. Let

$$A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

The two eigenvalues λ_1, λ_2 satisfy $\lambda_1 \lambda_2 = \det A$ and $\lambda_1 + \lambda_2 = \operatorname{tr} A$, hence:

- $A > 0 \Leftrightarrow \det(A) > 0$ and $\operatorname{tr} A > 0$
- $A < 0 \Leftrightarrow \det(A) > 0$ and $\operatorname{tr} A < 0$
- $A \geq 0 \Leftrightarrow \det(A) \geq 0$ and $\operatorname{tr} A \geq 0$
- $A \leq 0 \Leftrightarrow \det(A) \geq 0$ and $\operatorname{tr} A \leq 0$.

11.2 Convex case (complement)

11.2.1 Specific aspects of convex optimization

A major feature of convexity is that it makes local minimizers automatically global ones.

Proposition 11.3. *Let $f : X \to \mathbb{R}$ be convex. Every local minimizer of f is a global minimizer.*

Proposition 11.4. *Let $f : X \to \mathbb{R}$ be strictly convex. If f admits a global minimizer, then it is unique.*

11.2.2 Towards constrained optimization

The optimality conditions become:

Proposition 11.5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be Fréchet differentiable and convex and $\mathcal{U}$ be a closed convex subset of $\mathbb{R}^n$. Then a point $a \in \mathcal{U}$ is a global minimized of f over $\mathcal{U}$ if and only if*

$$-\nabla f(a) \in N_{\mathcal{U}}(a),$$

where $N_{\mathcal{U}}(a)$ is the normal cone of $\mathcal{U}$ in a.

Describing the normal cone is often a non-trivial task. This is typically achieved through Lagrange multipliers. To be studied in specialized courses...

11.3 Steepest descent method

11.3.1 Descent methods

A general algorithm for the minimization of a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is:

Listing 11.1: **Descent for minimization of continuous functions**
Let $f : \mathbb{R}^n \to \mathbb{R}$ a continuous function.

```
Choose an initial guess $x^0 \in X$
|For| $k$ |from| $1, 2, \dots$
   Determine descent direction $d^k \in \mathbb{R}^n$
   Determine a step length $s_k \in \mathbb{R}$
   Compute new approximation
      $x^k = x^{k-1} + s_k d^k$
   with
      $f(x^k) < f(x^{k-1})$
```

This is a general abstract scheme, which must be specified for the search direction $d^k \in \mathbb{R}^n$ and the step length $s_k \in \mathbb{R}$. A descent direction d^k is a vector d^k such that

$$f(x^{k-1} + sd^k) < f(x^{k-1})$$

as soon as $s > 0$ is small enough. If f is continuously differentiable, it holds the Taylor expansion

$$f(x^{k-1} + sd^k) = f(x^{k-1}) + s\nabla f(x^{k-1}) \cdot d^k + o(s).$$

We infer the (sufficient) condition for d^k to be a descent direction:

$$\nabla f(x^{k-1}) \cdot d^k < 0. \tag{11.2}$$

11.3.2 Steepest descent

A steepest descent direction at a point x is a direction d of unit norm that minimizes the expression $\nabla f(x) \cdot d$. Whereas it heavily depends on the chosen norm, for the Euclidean norm the following holds.

Theorem 11.6 (Optimal descent direction). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. Let $x \in \mathbb{R}^n$ with $\nabla f(x) \neq 0$. The direction of steepest descent in x is uniquely determined as*

$$d = -\frac{\nabla f(x)}{\|\nabla f(x)\|}.$$

Proof. It holds for arbitrary $d \in \mathbb{R}^n$ with $\|d\| = 1$

$$\nabla f(x) \cdot d \geq -\|\nabla f(x)\| \, \|d\| = -\|\nabla f(x)\|.$$

In the case $d = -\nabla f(x)/\|\nabla f(x)\|$ it holds an equality

$$\nabla f(x) \cdot d = -\|\nabla f(x)\|.$$

Therefore, $-\nabla f(x)/\|\nabla f(x)\|$ characterizes a unique normalized direction of steepest descent. $\square$

With this, we obtain in a natural way the *steepest descent or gradient descent* scheme, which is a descent algorithm with optimal descent direction:

Listing 11.2: **Minimization with gradient descent** Let $f : X \to \mathbb{R}$ continuously differentiable.

```
Choose $x^0 \in X$
|For| $k$ |from| $1, 2, \dots$
```

```
  |Abort|, if $\nabla f(x^{k-1}) = 0$
  Determine descent direction $d^k = -\nabla f(x^{k-1})/\|\nabla f(x^{k-1})\|$
  Choose a step length $s_k \in \mathbb{R}$
  Determine new approximation
    $x^k = x^{k-1} + s_k d^k$
  with
    $f(x^k) < f(x^{k-1})$
```

11.3.3 Line search

We have now a more specific scheme, but the step length is still an open question. Let us recall descent schemes for solving linear equation systems. Given a symmetric positive semidefinite matrix A, solving

$$Ax = b$$

is equivalent to the minimization of the quadratic functional (just double-check yourself that $Q'(x) = Ax - b$)

$$Q(x) = \frac{1}{2}(Ax, x) - (b, x).$$

For such a quadratic functions, the optimal step length $s_k \in \mathbb{R}$ with given direction d^k is the minimum of the scalar-valued function

$$h(s) := Q(x^{k-1} + sd^k),$$

which is just obtained through a second order algebraic equation. For a general function, this procedure is not as simple. The step length is rather determined by another numerical procedure known as line search, also used in Newton-type methods (e.g., damped Newton method). The general idea is to make trials in an organized way.

A classical line search technique is the Armijo rule (see e.g., [21]).

Listing 11.3: **Armijo step length rule**
Let $\beta, \gamma \in (0, 1)$ be given. We denote the previous step by $x^{k-1} \in \mathbb{R}^n$ and by $d^k \in \mathbb{R}^n$ the chosen descent direction.

```
Initialize $s_k^{(0)}$
|For| $l$ |from| $0, 1, 2, \dots$
  |Abort|, if $f(x^{k-1} + s_k^{(l)} d^k) < f(x^{k-1}) + \gamma s_k^{(l)} \nabla f(x^{k-1}) \cdot d^k$
  $s_k^{(l+1)} = \beta s_k^{(l)}$
```

The Armijo rule ensures that we really approach the minimum. One typically chooses $\beta = \gamma = 1/2$. It can be shown that this algorithm terminates after a finite number of steps and always finds a suitable step length:

Theorem 11.7 (Armijo step length rule). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable. Let $\gamma \in (0, 1)$ and $d \in \mathbb{R}^n$ a descent direction at a point $x \in X$, just satisfying*

$$\nabla f(x) \cdot d < 0.$$

For any $s_0 > 0$ there exists $m \in \mathbb{N}$ such that

$$f(x + \beta^m s_0 d) - f(x) \leq \gamma \beta^m s_0 \nabla f(x) \cdot d.$$

Proof. It holds

$$\frac{f(x + sd) - f(x)}{s} - \gamma \nabla f(x) \cdot d \to (1 - \gamma) \nabla f(x) \cdot d < 0 \quad (s \to 0).$$

k	x^k	y^k	$\|\nabla f(x^k)\|$	$f(x^k)$
0	-1.	1.5	356	
1	-1.212	1.3675	4.9958	9.5597
2	-1.0898	1.3940	4.7929	6.3382
3	-1.1373	1.3533	4.6039	1.9585
4	-1.0384	1.2768	4.5494	5.7600
5	-1.0836	1.2337	4.3770	1.9861
		$\cdots$		
100	1.0050	1.0100	2.55e-05	0.012493
101	1.0048	1.0101	2.52e-05	0.012492
102	1.0049	1.0099	2.48e-05	0.012491
103	1.0047	1.0099	2.44e-05	0.012491
104	1.0049	1.0097	2.41e-05	0.012490
105	1.0046	1.0098	2.37e-05	0.012489

Table 11.1: Results for the approximation of the Rosenbrock function. The table is taken from [25].

It follows from the continuity of ∇f that there exists an $\epsilon > 0$ such that for all $s < \epsilon$:

$$\frac{f(x+sd) - f(x)}{s} - \gamma \nabla f(x) \cdot d \le 0 \quad \forall |s| < \epsilon.$$

Given $\beta \in (0,1)$ we find a minimal $m \in \mathbb{N}$ such that $s = \beta^m s_0 < \epsilon$ holds true. $\square$

The gradient descent scheme converges slowly, and is therefore not very efficient (see our numerical concepts at the beginning of these lecture notes). However it is very robust and easy to implement, hence it is often used for complicated or high dimensional problems where more sophisticated methods are hardly applicable.

In Figure 11.3, we show the convergence behavior with the help for the so-called Rosenbrock function. This example is a typical test case in numerical optimization:

$$\min_{x,y \in \mathbb{R}} \{f(x,y) = 10(y - x^2)^2 + (1-x)^2\}.$$

We choose as initial guesses $x^0 = (-1, 1.5)$ and $x^0 = (2, -2)$. For the first test case, we summarize the convergence behavior in Table 11.1.

11.3.4 Improving the method (complement)

There are many more methods for increasing the efficiency of the gradient descent scheme. We simply refer to the standard literature [21].

If the governing function $f(x)$ is two-times continuously differentiable, then the Newton scheme, see section 12.4 for a general presentation, can be adopted for the numerical solution of the minimization problem. As previously discussed, the necessary optimality condition is given by

$$F(x) := \nabla f(x) \overset{!}{=} 0.$$

The Newton iteration from an initial guess $x^0 \in X$, reads for $k = 1, 2, 3, \dots$

$$DF(x^k)(x^{k+1} - x^k) = -F(x^k).$$

It holds

$$DF(x^k) = D^2 f(x^k),$$

where $D^2 f(x^k)$ denotes the Hessian of f. In order to apply Newton schemes, the cost functional $f(x)$ must be twice continuously differentiable. In order to apply the classical Newton-Kantorovich theorem (see later) the second order derivative must be even Lipschitz continuous.

Remark 11.1. *Often in practice the strict Newton assumptions cannot be made or it is even clear that these do not hold true. Nonetheless, Newton's method can be used. Indeed, it often still yields satisfactory results. This shows that between mathematical rigorous results and practical applications might be often a gap.*

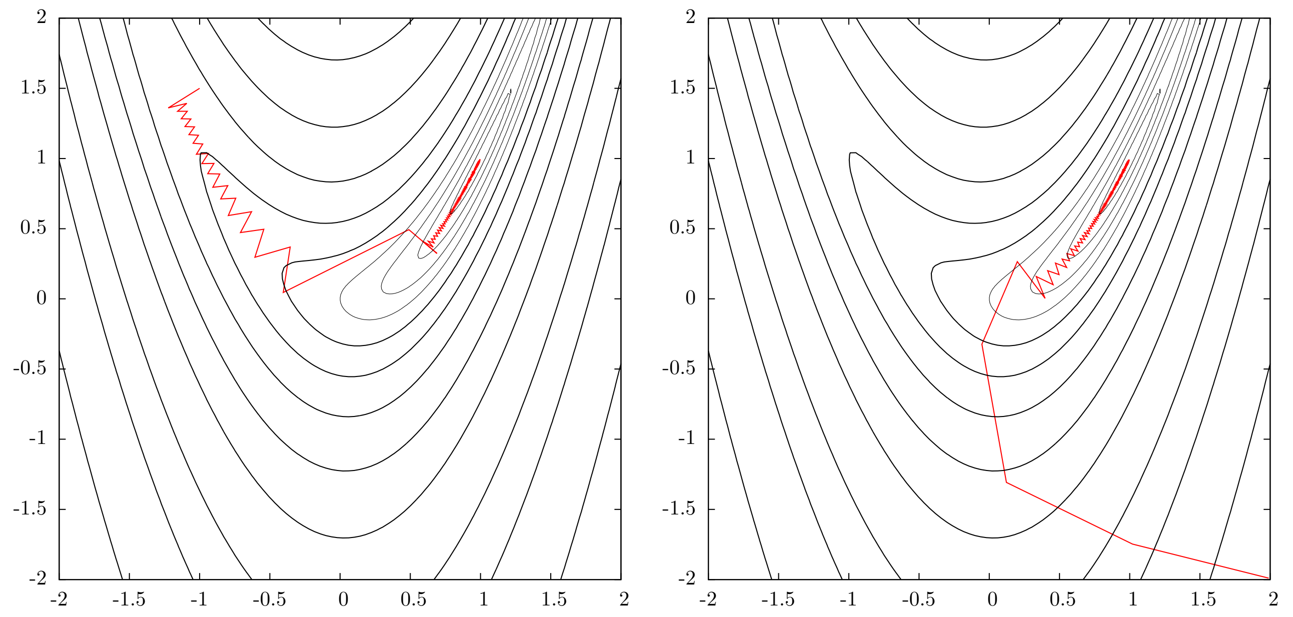


Figure 11.3: Convergence behavior of the gradient scheme for the approximation of the minimum of the Rosenbrock function $f(x,y) = 10(y-x^2)^2 + (1-x)^2$ with different initial values. Figure taken from [25].

k	x^k	y^k	$\|\nabla f(x^k)\|$	$f(x^k)$	k	x^k	y^k	$\|\nabla f(x^k)\|$	$f(x^k)$
0	-1	1.5	18.86	6.5	0	2	-2	496.71	361
1	-1.222	1.444	6.92	4.963	1	1.992	3.966	1.988	0.983
2	-0.104	-1.239	26.08	16.84	2	1.001	0.021	43.91	9.620
3	-0.063	0.002	2.128	1.127	3	1.001	1.002	2.57e-03	1.65e-06
4	0.963	-0.123	45.51	11.03	4	1.000	1.000	7.41e-05	2.74e-11
5	0.965	0.931	7.05e-02	0.00124					
6	0.999	0.999	5.56e-02	1.55e-05					
7	1.000	1.000	9.67e-08	2.34e-15					

Table 11.2: Convergence behavior of Newton's method for the Rosenbrock function. The results are taken from [25].

In Figure 11.4 we plot the convergence behavior of Newton's method for the Rosenbrock function. Our findings are summarized in Table 11.2.

In both cases, the minima could be approximated in only a very few steps. In the values of the table, we also observe that Newton's method is in general not a descent method. The iterates $f(x^k)$ do not monotonically converge to the minimum.

In numerical optimization (but also mildly and highly nonlinear PDE problems) the biggest problem is often the small convergence radius of Newton's method; i.e., the initial guess must be quite close to the solution to expect convergence. Here, we need so-called globalization strategies such as line search method or trust region concepts [21] or see also [9] for PDE problems. Another alternative is a mixture of gradient descent methods and Newton type methods: we first approach a good initial guess by using the robust gradient descent scheme. Once we are in the convergence region, we switch to Newton's method.

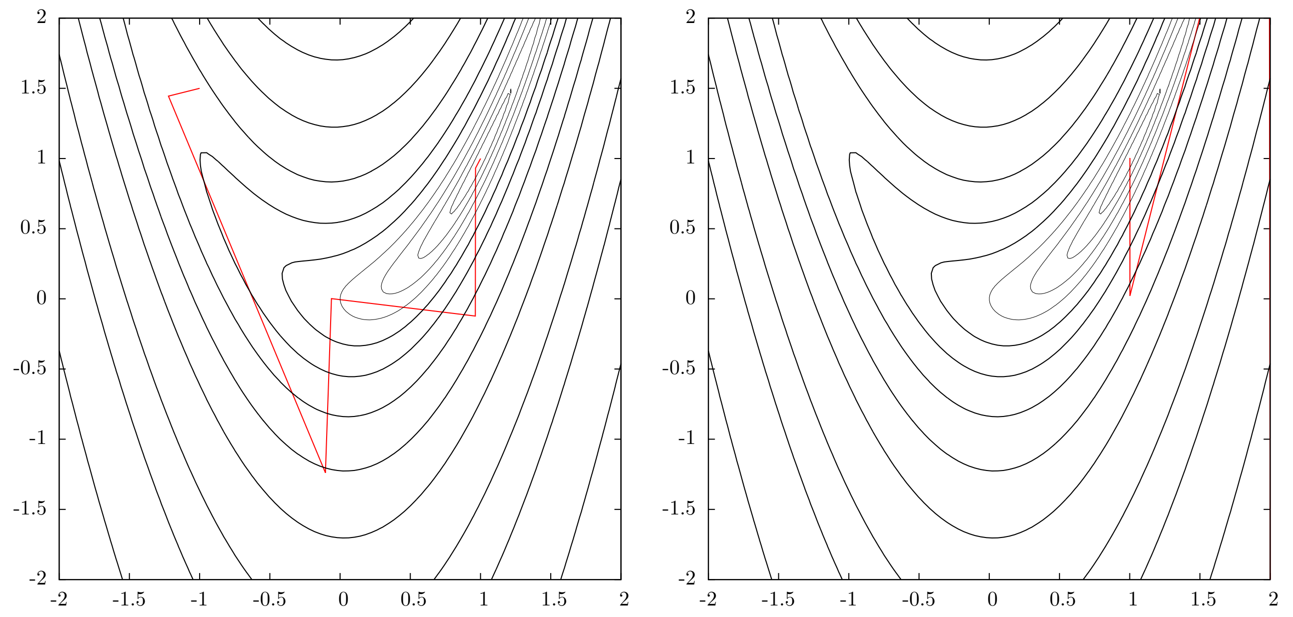


Figure 11.4: Convergence behavior of Newton's method for the approximation of the minimium of the Rosenbrock function $f(x, y) = 10(y - x^2)^2 + (1 - x)^2$. These figures are taken from [25].

11.4 Gradient descent with projection (complement)

We here discuss the problem

$$\underset{x \in \mathcal{U}}{\text{minimize}}\ f(x) \tag{11.3}$$

where $\mathcal{U}$ is a (closed) convex subset of $\mathbb{R}^n$, and $f : \mathbb{R}^n \to \mathbb{R}$ is a two-times continuously differentiable function. For clarity and simplicity we also assume that f is convex, but this assumption is not crucial. One way to cope with the constraint $x \in \mathcal{U}$ is to replace the iteration $x^k = x^{k-1} + s_k d^k$ by

$$x^k = P_{\mathcal{U}}(x^{k-1} + s_k d^k),$$

where $P_{\mathcal{U}}$ is the projection operator onto $\mathcal{U}$, see section 8.3. This method is appropriate if we have at disposal an efficient means to compute the projection. In this case we speak of simple constraints, examples of which have been discussed in section 8.3.

In order to guaranty the descent property

$$f(P_{\mathcal{U}}(x^{k-1} + sd^k)) < f(x^{k-1}) \text{ for small } s > 0,$$

the condition (11.2) turns out not to be sufficient. Luckily,

Proposition 11.8. *The direction $d_k = -\nabla f(x^{k-1})/\|\nabla f(x^{k-1})\|$ is a descent direction, unless x^{k-1} is a minimizer.*

Proof. We have for any $s > 0$

$$f(P_{\mathcal{U}}(x^{k-1} + sd^k)) - f(x^{k-1}) = \nabla f(x^{k-1}) \cdot (P_{\mathcal{U}}(x^{k-1} + sd^k) - x^{k-1}) + O(\|P_{\mathcal{U}}(x^{k-1} + sd^k) - x^{k-1}\|^2).$$

Setting $y_s = x^{k-1} + sd^k$ the first order term rewrites as

$$\begin{aligned}
\nabla f(x^{k-1}) \cdot (P_{\mathcal{U}}(y_s) - x^{k-1}) &= -\frac{\|\nabla f(x^{k-1})\|}{s}(y_s - x^{k-1}) \cdot (P_{\mathcal{U}}(y_s) - x^{k-1}) \\
&= -\frac{\|\nabla f(x^{k-1})\|}{s}\Big((y_s - P_{\mathcal{U}}(y_s)) \cdot (P_{\mathcal{U}}(y_s) - x^{k-1}) + \|P_{\mathcal{U}}(y_s) - x^{k-1}\|^2\Big) \\
&\le -\frac{\|\nabla f(x^{k-1})\|}{s}\|P_{\mathcal{U}}(y_s) - x^{k-1}\|^2,
\end{aligned}$$

using the variational inequality (8.2). It follows that

$$f(P_{\mathcal{U}}(x^{k-1}+sd^k)) - f(x^{k-1}) \leq \|P_{\mathcal{U}}(y_s) - x^{k-1}\|^2 \left(-\frac{\|\nabla f(x^{k-1})\|}{s} + O(1)\right).$$

Moreover, using again (8.2) we obtain

$$P_{\mathcal{U}}(y_s) = x^{k-1} \Leftrightarrow (y_s - x^{k-1}) \cdot (z - x^{k-1}) \leq 0 \ \forall z \in \mathcal{U} \Leftrightarrow -\nabla f(x^{k-1}) \in N_{\mathcal{U}}(x^{k-1}),$$

which is the optimality condition stated in Proposition 11.3. We conclude the desired strict inequality $f(P_{\mathcal{U}}(x^{k-1}+sd^k)) - f(x^{k-1}) < 0$ for small s, unless x^{k-1} is a minimizer. □

Chapter 12

Solving nonlinear equations

Solving nonlinear equations is one of the most-found tasks in numerical mathematics, but also in the mathematical analysis via fixed-point theorems. Examples are

- Root-finding of nonlinear functions, e.g., $f(x) = ax^3 + bx^2 + cx + d$
- Solution of (nonlinear) differential equations
- Solution of nonlinear optimization problems, e.g., classical tasks or in machine learning

Analytical results are again rarely achievable and for this reason, numerical algorithms are used. These are iterations in which an initial guess is chosen and from this guess, further approximations are calculated.

12.1 Introduction to iterative methods

Consider a function $f : I \subset \mathbb{R} \to \mathbb{R}$. We search, if it exists, some point $x \in I$ such that $f(x) = 0$. In general, such an equation can only be solved approximately. Iterative methods consist in the construction of a sequence (x_n) such that $x_n \to x$ when $n \to +\infty$ with $f(x) = 0$. Moreover, the speed of convergence of the sequence is of interest. It can be represented by the notion of order of convergence.

Let (x_n) be a sequence of real numbers that converges to some x, and let $r \geq 1$ be a real number.

Definition 12.1. *(i) The convergence of the sequence (x_n) is said to be of order at least r if there exist $C \geq 0$ and $N \in \mathbb{N}$ such that*

$$|x_{n+1} - x| \leq C|x_n - x|^r \qquad \forall n \geq N.$$

(ii) The convergence is said to be of order r exactly if, in addition, there exist $c > 0$ and $N' \in \mathbb{N}$ such that

$$|x_{n+1} - x| \geq c|x_n - x|^r \qquad \forall n \geq N'.$$

It is immediately seen that, if there exists $\lambda > 0$ such that

$$\frac{|x_{n+1} - x|}{|x_n - x|^r} \to \lambda,$$

then the convergence is of order r exactly.

Obviously, the larger the order of convergence is, the better it is. In the case of $r = 1$, however, the constant $0 < C < 1$ must be bounded. When the convergence is of (exact) order 1, we also speak of linear convergence. For $C := C_n \to 0$ for $n \to \infty$, we have so-called superlinear convergence. When it is of order 2 we speak of quadratic convergence.

12.2 The bisection method

Let $f : [a, b] \to \mathbb{R}$ be a continuous function such that $f(a)f(b) < 0$. The bisection method consists in the construction of three sequences (a_n), (b_n) and (x_n) according to the following procedure

1. Initialization: set $a_0 = a$, $b_0 = b$, $x_0 = (a + b)/2$.
2. Iteration k: set
$$\begin{aligned} a_{k+1} = a_k,\ b_{k+1} = x_k \quad \text{if } f(x_k)f(a_k) < 0, \\ a_{k+1} = x_k,\ b_{k+1} = b_k \quad \text{if } f(x_k)f(b_k) < 0, \end{aligned}$$
$$x_{k+1} = \frac{a_{k+1} + b_{k+1}}{2}.$$
If $f(x_{k+1}) = 0$ then an exact solution is obtained and the iterations stop. Otherwise the iterations are continued.

It is not difficult to establish the following result.

Proposition 12.1 (Bisection)**.** *Let $f \in C[a, b]$ with $f(a)f(b) < 0$. Let a_n, b_n, x_n the sequence constructed through the above algorithm. It holds*

$$x_n \to z, \quad f(z) = 0,$$

and the error estimate.

$$|x_n - z| \leq \frac{b - a}{2^{n+1}}.$$

Proof. (i) For $a < b$, it holds

$$a \leq a_1 \leq \cdots \leq a_n \leq a_{n+1} \leq x_n \leq b_{n+1} \leq b_n \leq \cdots \leq b_1 \leq b.$$

Thus, the scheme can be constructed. The sequence x_n is bounded through a_n und b_n. Therefore (classical result in Analysis), the sequence x_n converges. The algorithm terminates when $f(x_n) = 0$, i.e., $x_n = z$ is the sought root.

(ii) It holds

$$b_{n+1} - a_{n+1} = \begin{cases} x_n - a_n = \frac{b_n - a_n}{2} & f(a_n)f(x_n) < 0 \\ b_n - x_n = \frac{b_n - a_n}{2} & f(a_n)f(x_n) > 0 \end{cases}.$$

In both cases, it follows

$$|b_n - a_n| = \frac{|b_0 - a_0|}{2^n}. \tag{12.1}$$

Due to

$$a_n \leq x_n \leq b_n$$

it follows the convergence of $x_n \to z$ towards $z \in [a_n, b_n] \subset [a, b]$.

(iii) For the limit z it holds thanks to continuity $f(\cdot)$

$$0 \leq f(z)^2 = \lim_{n\to\infty} f(a_n)f(b_n) \leq 0 \quad \Rightarrow \quad f(z) = 0.$$

Therefore, the limit is the sought root. Finally, we can derive the from (12.1) the error estimate:

$$|x_n - z| \leq \frac{|b_n - a_n|}{2} \leq \frac{|b - a|}{2^{n+1}}.$$

□

Remark 12.1 (A priori error estimate). *In Proposition 12.1 we have the a priori error estimate*

$$|x_n - z| \le \frac{b-a}{2^{n+1}}$$

Such an estimate can be evaluated before the computation. For instance, we obtain a guess for the required number of iterations n.

Exercise 12.1 (Bisection). *We consider $I = [0,1]$. The a priori error estimate yields for different $n \in \mathbf{N}$ the bounds*

$$|x_9 - z| < 10^{-3}, \quad |x_{19} - z| < 10^{-6}, \quad |x_{29} - z| < 10^{-9}.$$

This means that in iteration 29 we can expect an accuracy of $TOL = 10^{-9}$. Compare this result to Newton's method (no matter its construction) in which a similar accuracy is obtained in approx. 4 steps. This shows that Newton's method can be more efficient than bisection. More specifically, bisection has a linear order of convergence and Newton's method it quadratically convergent.

Finally, we discuss an a posteriori result for which f must be continuously differentiable.

Proposition 12.2 (A posteriori error estimate). *Let $f \in C^1[a,b]$ with $f(a)f(b) < 0$ and $f(z) = 0$ for $z \in [a,b]$. The derivative is assumed to be bounded:*

$$0 < m \le |f'(x)| \le M < \infty$$

For the root z of f and the sequence $(x_n)_{n\in\mathbb{N}}$, it holds

$$\frac{|f(x_n)|}{M} \le |x_n - z| \le \frac{|f(x_n)|}{m}.$$

Proof. With Taylor expansion, it holds

$$f(x_n) = f(z) + f'(\xi)(x_n - z) = f'(\xi)(x_n - z) \tag{12.2}$$

with an intermediate value $\xi \in [a_n, b_n]$. It follows

$$\frac{|f(x_n)|}{\max|f'(x)|} \le |x_n - z| \le \frac{|f(x_n)|}{\min|f'(x)|}.$$

□

Exercise 12.2. *Given $f(x) = (x-3)^2 - 1$. Take $a = 0$ and $b = 3.5$ and compute the root via bisection. Make first a sketch of the function curve. Check whether bisection can be applied. Compute three steps per hand. Optional: implement the code. Complement: is there only one root? If not, how can be approximate the other(s)?*

Exercise 12.3. *Given $f(x) = (x-3)^2$. Take again $a = 0$ and $b = 3.5$. Can bisection be applied.*

12.3 Fixed points

12.3.1 Reminder: the (Banach) fixed point theorem

Definition 12.2. *A function $F : U \subset \mathbb{R} \to \mathbb{R}$ is said to be Lipschitz continuous with Lipschitz constant $k \ge 0$ if*

$$|F(x) - F(y)| \le k|x - y| \qquad \forall x, y \in U.$$

If F is Lipschitz continuous with Lipschitz constant $k < 1$ then F is said to be contracting.

Definition 12.3. *We say that x is a fixed point of F if $F(x) = x$.*

Note that the equation $f(x) = 0$ can always be equivalently rewritten as $F(x) = x$, for instance with $F(x) = f(x) + x$.

Theorem 12.3 (fixed point)**.** *Let $F : U \to U$ be contracting, with $U \subset \mathbb{R}$ closed (for instance a closed interval $[a, b]$). Then F admits a unique fixed point x^* in U. Moreover, the sequence (x_n) defined by $x_0 \in U$ and $x_{n+1} = F(x_n)$ converges to x^*, and we have*

$$|x_n - x^*| \leq \frac{k^n}{1-k}|x_1 - x_0| \qquad \forall n \in \mathbb{N}.$$

12.3.2 Attractive and repulsive fixed points

Let I be an open interval of $\mathbb{R}$, $F : I \to \mathbb{R}$ be a function of class $\mathcal{C}^1$, and $x \in I$ be a fixed point of F.

With the mean value theorem in analysis, we have

$$\left|\frac{x_{n+1} - x}{x_n - x}\right| = \left|\frac{F(x_{n+1}) - F(x)}{x_n - x}\right| \to |F'(x)| \quad \text{for } n \to \infty$$

From this it follows that the asymptotic convergence rate is linear and the limit is $|F'(x)|$. Specifically, in the case of $|F'(x)| = 0$, we have at least superlinear convergence. From these considerations we obtain

Definition 12.4. *We say that*

- *x is an attractive fixed point if $|F'(x)| < 1$,*
- *x is a repulsive fixed point if $|F'(x)| > 1$.*

This notion gives an information on whether F is contracting or not in a neighborhood of x. It is therefore not surprising to have the following two results.

Proposition 12.4. *Let x be an attractive fixed point of F. There exists $h > 0$ such that $F([x - h, x + h]) \subset [x - h, x + h] \subset I$ and that all sequence defined by $x_0 \in [x - h, x + h]$ and $x_{n+1} = F(x_n)$ converges to x. Moreover, for such a sequence, the convergence is of order at least 1.*

Proposition 12.5. *Let x be a repulsive fixed point of F and (x_n) be a sequence such that $x_{n+1} = F(x_n)$ $\forall n \in \mathbb{N}$. Then*

- *either there exists an integer N such that $x_n = x$ $\forall n \geq N$,*
- *or (x_n) does not converge to x.*

12.3.3 Calculating fixed points

The algorithm to compute fixed-points is simple:

$$x_{n+1} = F(x_n), \quad n = 0, 1, 2, 3, \ldots$$

where x_0 is the initial guess, therein F is the so-called iteration function.

Exercise 12.4. *Take for instance $F(x) = \cos(x)$ as an example for an iteration function.*

Exercise 12.5 (Root-finding problem)**.** *Fixed-point problems are naturally root-finding problems since $x = F(x)$ is equivalent to $x - F(x) = 0$.*

Exercise 12.6. *Given again $f(x) = (x-3)^2 - 1$. Take $x_0 = 0$ as initial guess and design first the fixed point iteration function $F(x)$. The introduce the iteration scheme and a relaxation parameter $\omega > 0$. The latter one is required such that we deal with an attractive fixed point with $|F'(x)| < 1$. Compute for $\omega = 1$ and $\omega = 0.1$ the first three iterations x_1, x_2, x_3. Justify theoretically for which ω the condition $|F'(x)| < 1$ holds true.*

12.4 The Newton method

Let us come back to the root-finding problem

$$f(x) = 0.$$

We assume that f is differentiable. We construct a sequence of iterates $(x_n)_{n\in\mathbb{R}}$ and hopefully reach at some point

$$|f(x_n)| < TOL, \quad \text{where } TOL \text{ is again small, e.g., } TOL = 10^{-10}.$$

One has to start with a Taylor expansion. In our lecture we do this as follows. Let us assume that we are at x_n and can evaluate $f(x_n)$. Now we want to compute this next iterate x_{n+1} with the unknown value $f(x_{n+1})$. Taylor gives us:

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n) + o(x_{n+1} - x_n)^2$$

We assume that $f(x_{n+1}) = 0$ (or very close to zero $f(n_{k+1}) \approx 0$). Then, x_{n+1} is the sought root and neglecting the higher-order terms we obtain:

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n),$$

i.e.,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

If (x_n) converges, the Taylor-Young formula yields $f(x_{n+1}) = o(x_{n+1} - x_n)$, hence a fast convergence of $f(x_n)$ to 0 is expected.

Exercise 12.7. *Given again $f(x) = (x-3)^2 - 1$. Take $x_0 = 0$ as initial guess and use Newton's method to compute the first three iterates x_1, x_2, x_3. Compare with your results the speed of convergence for bisection, fixed-point and Newton.*

Exercise 12.8 (Newton's method as fixed-point iteration)**.** *The previous algorithm can be stated in terms of a fixed-point iteration with the iteration function*

$$F(x) = x - \frac{f(x)}{f'(x)}.$$

Theorem 12.6. *Let I be an open interval and $f : I \to \mathbb{R}$ be a function of class $\mathcal{C}^2$. We assume that there exist positive real numbers m, k such that*

$$|f'(y)| \geq m \qquad \forall y \in I,$$
$$|f''(y)| \leq k \qquad \forall y \in I.$$

Let $x \in I$ be such that $f(x) = 0$. If $0 < h < 2m/k$, $x_0 \in]x-h, x+h[\subset I$ and

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad \forall n \in \mathbb{N},$$

then

1. *$x_n \in]x-h, x+h[\qquad \forall n \in \mathbb{N}$,*
2. *$x_n \to x$,*
3. *the convergence is of order at least 2.*

In the practical use of the Newton method, the constants m and k are usually not known (it is even not always clear that such constants exist). Therefore it is very difficult to ensure that x_0 is taken sufficiently close to x (recall also that x is by definition also unknown) in order to have convergence. This is the major drawback of the Newton method. However, when it converges, the convergence is fast. In contrast, the bisection method guarantees the convergence under the condition $f(a)f(b) < 0$, which is easy to check, but the convergence is slow. Efficient methods often combine several techniques, with Newton-type iterations performed as last stage.

12.5 Generalization to higher dimensions (complement)

The fixed point and Newton methods admit rather straightforward extensions to higher dimensions, i.e., for solving $F(x) = 0$ or $f(x) = x$ with $F, f : \mathbb{R}^n \to \mathbb{R}^n$. In Newton's method, the derivative is replaced by the Jacobian matrix $DF(x)$, hence the iteration becomes

$$x_{n+1} = x_n - DF(x_n)^{-1}F(x_n).$$

It is actually not needed to compute the full inverse of $DF(x_n)$, but only the solution of the system $DF(x_n)h = F(x_n)$ has to be found to obtain $DF(x_n)^{-1}F(x_n)$. Still, for large problems, this may yield a significant computational cost, more or less balanced by the fast convergence of the sequence.

12.5.1 Newton's method: going from $\mathbb{R}$ to higher dimensions

Overview:

- Newton-Raphson (1D), find $x \in \mathbb{R}$ via iterating $k = 0, 1, 2, \dots$ such that $x_k \approx x$ via:

$$\begin{aligned} \text{Find } \delta x \in \mathbb{R}: \quad & f'(x_k)\delta x = -f(x_k), \\ \text{Update:} \quad & x_{k+1} = x_k + \delta x. \end{aligned}$$

- Newton in $\mathbb{R}^n$, find $x \in \mathbb{R}^n$ via iterating $k = 0, 1, 2, \dots$ such that $x_k \approx x$ via:

$$\begin{aligned} \text{Find } \delta x \in \mathbb{R}^n: \quad & F'(x_k)\delta x = -F(x_k), \\ \text{Update:} \quad & x_{k+1} = x_k + \delta x. \end{aligned}$$

  Here we need to solve a linear equation system to compute the update $\delta x \in \mathbb{R}^n$.

Often, Newton's method is formulated in terms of a defect-correction scheme. To illustrate this more clearly we consider the problem $f(x) = y$ instead of $f(x) = 0$.

Definition 12.5 (Defect)**.** *Let $\tilde{x} \in \mathbb{R}$ an approximation of the solution $f(x) = y$. The defect (or similarly the residual) is defined as*

$$d(\tilde{x}) = y - f(\tilde{x}).$$

Definition 12.6 (Newton's method as defect-correction scheme)**.**

$$\begin{aligned} f'(x_k)\delta x = d_k, \quad & d_k := y - f(x_k), \\ x_{k+1} = x_k + \delta x, \quad & k = 0, 1, 2, \dots. \end{aligned}$$

The iteration is finished with the same stopping criterion as for the classical scheme. To compute the update δx we need to invert $f'(x_k)$:

$$\delta x = (f'(x_k))^{-1} d_k.$$

This step seems trivial but is the most critical one if we deal with problems in $\mathbb{R}^n$ with $n > 1$ or in function spaces. Because here, the derivative becomes a matrix. Therefore, the problem results in solving a linear equation system of the type $A\delta x = b$.

Remark 12.2. *This previous form of Newton's method is already very close to the schemes that are used in research. In higher dimensions it applies to deal with problems such as nonlinear PDEs or optimization. The 'only' aspects that are however big research topics are the choice of*

- *good initial Newton guesses;*
- *globalization techniques.*

Two very good books on these topics, including further materials as well, are [9, 21].

12.5.2 A basic algorithm for a residual-based Newton method

In this type of methods, the main criterion is a decrease of the residual in each step. We denote $F(x) = -d(x) = f(x) - y$.

Algorithm 12.7 (Residual-based Newton's method)**.** *Given an initial guess x_0. Iterate for $k = 0, 1, 2, \dots$ such that*

$$\begin{aligned} \textit{Find } \delta x \in \mathbb{R}^n: \quad & F'(x_k)\delta x_k = -F(x_k), \\ \textit{Update:} \quad & x_{k+1} = x_k + \lambda_k \delta x_k, \end{aligned}$$

with $\lambda_k \in (0,1]$ (see the next sections how λ_k can be determined). A full Newton step corresponds to $\lambda_k = 1$. The criterion for convergence is the contraction of the residuals measured in terms of a discrete vector norm:

$$\|F(x_{k+1})\| < \|F(x_k)\|.$$

In order to save some computational cost, close to the solution x^*, intermediate simplified Newton steps can be used. In the case of $\lambda_k = 1$ we observe

$$\theta_k = \frac{\|F(x_{k+1})\|}{\|F(x_k)\|} < 1.$$

If $\theta_k < \theta_{max}$, e.g., $\theta_{max} = 0.1$, then the old Jacobian $F'(x_k)$ is kept and used again in the next step $k+1$. Otherwise, if $\theta_k > \theta_{max}$, the Jacobian will be assembled. Finally the stopping criterion is one of the following (relative preferred!):

$$\begin{aligned} \|F(x_{k+1})\| &\leq TOL_N \quad \textit{(absolute)} \\ \|F(x_{k+1})\| &\leq TOL_N \|F(x_0)\| \quad \textit{(relative)} \end{aligned}$$

If fulfilled, set $x^* := x_{k+1}$ and the (approximate) root x^* of the problem $F(x) = 0$ is found.

12.5.3 Example of the basic Newton method

We illustrate the basic Newton schemes with the help a numerical example in which we want to solve

$$x^2 = 2 \quad \text{in } \mathbb{R}$$

We reformulate this equation as root-finding problem:

$$f(x) = 0,$$

where

$$f(x) = x^2 - 2.$$

For Newton's method we need to compute the first derivative, which is trivial here,

$$f'(x) = 2x,$$

but can become a real challenge for complicated problems. To start Newton's method (or any iterative scheme), we need an initial guess (similar to ODE-IVP). Let us choose $x_0 = 3$. Then we obtain as results:

```
Iter x            f(x)
==============================
0    3.000000e+00 7.000000e+00
1    1.833333e+00 1.361111e+00
2    1.462121e+00 1.377984e-01
3    1.414998e+00 2.220557e-03
4    1.414214e+00 6.156754e-07
5    1.414214e+00 4.751755e-14
==============================
```

We see that Newton's method converges very fast, i.e., quadratically, which means that the correct number of digits roughly doubles at each iteration step.

12.5.4 Example using a Newton defect-correction scheme including line search

In this second example, we present Newton's method as defect correction scheme as introduced in Definition 12.6. Additionally we introduce a line search parameter $\omega \in [0, 1]$. If the initial Newton guess is not good enough we can enlarge the convergence radius of Newton's method by choosing $\omega < 1$. Of course for the given problem here, Newton's method will always converge, because the underlying function is convex. Despite the fact that ω is not really necessary in this example, we can highlight another feature very well. In the optimal case (as seen above in the results), Newton's method will converge quadratically. Any adaptation will deteriorate the performance. Thus, let us choose $\omega = 0.9$. Then we obtain as result only linear convergence:

```
Iter x            f(x)
==============================
0    3.000000e+00 7.000000e+00
1    1.950000e+00 1.802500e+00
2    1.534038e+00 3.532740e-01
3    1.430408e+00 4.606670e-02
4    1.415915e+00 4.816699e-03
5    1.414385e+00 4.840133e-04
6    1.414231e+00 4.842504e-05
7    1.414215e+00 4.842742e-06
8    1.414214e+00 4.842766e-07
9    1.414214e+00 4.842768e-08
10   1.414214e+00 4.842768e-09
11   1.414214e+00 4.842771e-10
12   1.414214e+00 4.842748e-11
==============================
```

And if we choose $\omega = 0.5$ we only obtain (perfect) linear convergence:

```
Iter x            f(x)
==============================
0    3.000000e+00 7.000000e+00
1    2.416667e+00 3.840278e+00
2    2.019397e+00 2.077962e+00
3    1.762146e+00 1.105159e+00
4    1.605355e+00 5.771631e-01
5    1.515474e+00 2.966601e-01
...
32   1.414214e+00 2.286786e-09
33   1.414214e+00 1.143393e-09
34   1.414214e+00 5.716965e-10
35   1.414214e+00 2.858482e-10
36   1.414214e+00 1.429239e-10
37   1.414214e+00 7.146195e-11
==============================
```

In summary, using a full Newton method we only need 5 iterations until a tolerance of $TOL_N = 10^{-10}$ is reached. Having linear convergence with a good convergence factor still 12 iterations are necessary. And a linear scheme does need 37 iterations. This shows nicely the big advantage of Newton's method (i.e., quadratic convergence) in comparison to linear or super-linear convergence.

12.5.5 Newton's method in higher dimensions and the Newton-Kantorovich theorem

In this section, we now study Newton's method in $\mathbb{R}^n$ from the theoretical side including proofs and further extensions such as globalization techniques.

Let $f : D \subset \mathbb{R}^n \to \mathbb{R}^n$. We seek a solution to the root-finding $f(x) = (f_1(x), \dots, f_n(x)) = 0$. We take an initial guess $x^{(0)} \in D$ and iterate $x^{(k)} \to x^{(k+1)}$. The derivation is based on the one-dimensional situation. Around the iteration $x^{(k)}$ we linearize the function $f(\cdot)$ with the help of the Taylor up to the first order. In an arbitrary direction, it holds $w^{(k)} \in \mathbb{R}^n$

$$f(x^{(k)} + w^{(k)}) = f(x^{(k)}) + \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(x^{(k)}) w_j^{(k)} + O(|w^{(k)}|^2).$$

The new iterate $x^{(k+1)} = x^{(k)} + w^{(k)}$ is defined as root of the linearization. Therefore, we seek a solution $w^{(k)} \in \mathbb{R}^n$ of the linear equation system

$$\sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(x^{(k)}) w_j^{(k)} = -f_i(x^{(k)}), \quad i = 1, \dots, n.$$

The Jacobian $Df : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ reads

$$Df = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

and consequently, we obtain in compact notation as linear equation system

$$Df(x^{(k)}) w^{(k)} = -f(x^{(k)}).$$

Consequently, the Newton iteration with initial guess $x^{(0)} \in D$ for $k = 0, 1, 2, \dots$ reads

$$x^{(0)} \in D, \quad Df(x^{(k)}) w^{(k)} = -f(x^{(k)}), \quad x^{(k+1)} = x^{(k)} + w^{(k)}. \tag{12.3}$$

The biggest difference to the one-dimensional case is that we now deal with a matrix for the derivative. Specifically, we need to compute n^2 derivatives. Rather than just dividing, we need to solve a linear equation system with the coefficient matrix $Df(x_k) \in \mathbb{R}^{n \times n}$.

The central convergence results of this Newton method is the Newton-Kantorovich theorem. Let $f : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be a differentiable mapping. With $\|\cdot\|$ we denote the Euclidian vector norm and the induced matrix norm; here the spectral norm. We search a root $z \in D$ such that $f(z) = 0$.

Theorem 12.8 (Newton-Kantorovich)**.** *Let $D \subset \mathbb{R}^n$ be an open and convex set. Moreover, let $f : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be continuously differentiable.*
(1) *Let the Jacobian matrix Df be uniformly Lipschitz-continuous for all $x, y \in D$:*

$$\|Df(x) - Df(y)\| \le L\|x - y\|, \quad x, y \in D, \tag{12.4}$$

with $L < \infty$.
(2) *Let the Jacobian on D be a uniformly bounded inverse*

$$\|Df(x)^{-1}\| \le \beta, \quad x \in D, \tag{12.5}$$

with $\beta < \infty$.

(3) *For the initial guess $x^{(0)} \in D$, assume*

$$q := \alpha\beta L < \frac{1}{2}, \quad \alpha := \|Df(x^{(0)})^{-1} f(x^{(0)})\|. \tag{12.6}$$

(4) *For $r := 2\alpha$ suppose that the closed ball*

$$B_r(x_0) := \{x \in \mathbb{R}^n : \|x - x_0\| \leq r\}$$

is contained in the set D.

Then, the function f has a unique root $z \in B_r(x_0)$ and the Newton iteration

$$Df(x^{(k)})\delta x = -f(x^{(k)}), \quad x^{(k+1)} = x^{(k)} + \delta x, \quad k = 0, 1, 2, \ldots,$$

converges quadratically to the root z. Furthermore, we have the a priori error estimate

$$\|x^{(k)} - z\| \leq 2\alpha q^{2^k - 1}, \quad k = 0, 1, \ldots$$

Proof. We translate the proof from [25], which was based on [24], which is itself close to Kantorovich [19]. Alternative proofs can be found in [9] for instance.

The proof is complicated and we first make an outline:

(i) Derivation of some auxiliary lemma.

(ii) We then show that all iterates $x^{(k)}$ are contained in the ball $B_r(x_0)$. Furthermore, we then obtain the a priori error estimate.

(iii) We show that the iterates $(x^{(k)})_{k\in\mathbb{N}}$ form a Cauchy sequence and have consequently a limit z.

(iv) Next, we establish that this limit z is a root of f

(v) Finally, we show that we have one and only one root in $B_r(x_0)$.

(i) Let $x, y, z \in D$. Since D is convex, it holds for all $x, y \in D$

$$f(x) - f(y) = \int_0^1 Df\big(y + s(x - y)\big)(x - y)\,\mathrm{d}s.$$

We subtract on both sides $Df(z)(x - y)$:

$$f(x) - f(y) - Df(z)(x - y) = \int_0^1 \big(\big(Df(y + s(x - y)\big) - Df(z)\big)\,(x - y)\,\mathrm{d}s$$

With the Lipschitz continuity of the Jacobian Df, it follows

$$\begin{aligned}\|f(y) - f(x) - Df(z)(y - x)\| &\leq L\|y - x\| \int_0^1 \|s(x - z) + (1 - s)(y - z)\|\,ds \\ &\leq \frac{L}{2}\|y - x\|\big(\|x - z\| + \|y - z\|\big).\end{aligned}$$

For the choice $z = x$, we obtain

$$\|f(y) - f(x) - Df(x)(y - x)\| \leq \frac{L}{2}\|y - x\|^2, \quad \forall x, y \in D. \tag{12.7}$$

And for $z = x_0$, it yields

$$\|f(y) - f(x) - D(x_0)(y - x)\| \leq rL\|y - x\|, \quad \forall x, y \in B_r(x_0). \tag{12.8}$$

(ii) Next, we show that all iterates are contained in the ball $B_r(x^{(0)})$. These derivations also yield as byproduct the a priori error estimate. We establish the proof using induction and show that

$$\|x^{(k+1)} - x^{(0)}\| \leq r, \quad \|x^{(k+1)} - x^k\| \leq \alpha q^{2^{(k)}-1}, \quad k = 1, 2, \dots \tag{12.9}$$

Let us begin with $k = 0$. It holds for the Newton iteration $x^{(1)} - x^{(0)} = -Df(x^{(0)})^{-1} f(x^{(0)})$ with the condition (12.6)

$$\|x^{(1)} - x^{(0)}\| = \|Df(x^{(0)})^{-1} f(x^{(0)})\| = \alpha = \frac{r}{2} < r,$$

d.h. $x^{(1)} \in B_r(x^{(0)})$, and it holds the estimate

$$\|x^{(1)} - x^{(0)}\| \leq \alpha = \alpha q^{2^0 - 1}.$$

Induction step from $k \to k+1$. According to the induction assumption, both equations (12.9) are true for $k \geq 0$. Consequently, it holds $x^{(k)} \in B_r(x^{(0)})$ such that the Newton iterate $x^{(k+1)}$ is well-defined. Then, we infer with the condition (12.5), with the estimate (12.7) and the induction assumption (12.9) with the definition of q the following estimates:

$$\begin{aligned}
\|x^{(k+1)} - x^{(k)}\| &= \|Df(x^{(k)})^{-1} f(x^{(k)})\| \\
&\leq \beta\, \|f(x^{(k)})\| \\
&= \beta \| f(x^{(k)}) \underbrace{- f(x^{(k-1)}) - Df(x^{(k-1)})(x^{(k)} - x^{(k-1)})}_{=0} \| \\
&\leq \frac{\beta L}{2} \underbrace{\|x^{(k)} - x^{(k-1)}\|^2}_{\text{Induction}} \leq \frac{\beta L}{2} \left(\alpha q^{2^{(k-1)}-1} \right)^2 \\
&= \frac{\alpha}{2} q^{2^{(k)}-1} < \alpha q^{2^{(k)}-1}
\end{aligned}$$

Hence, we have

$$\begin{aligned}
\|x^{(k+1)} - x^{(0)}\| &\leq \|x^{(k+1)} - x^{(k)}\| + \dots + \|x^{(1)} - x^{(0)}\| \\
&\leq \alpha \big(1 + q + q^3 + q^7 + \dots + q^{2^{(k)}-1}\big) \\
&\leq \frac{\alpha}{1-q} \leq 2\alpha = r.
\end{aligned}$$

Thus, $x^{(k+1)} \in B_r(x^{(0)})$. This shows that induction step from $k \to k+1$, i.e., the two inequalities (12.9) hold true for $k+1$.

(iii) We now show that the iterates $x^{(k)} \in B_r(x^{(0)})$ form a Cauchy sequence. Let $m > 0$. Since $q < \frac{1}{2}$, it holds

$$\begin{aligned}
\|x^{(k)} - x^{k+m}\| &\leq \|x^{(k)} - x^{(k+1)}\| + \dots + \|x^{k+m-1} - x^{k+m}\| \\
&\leq \alpha \big(q^{2^{(k)}-1} + q^{2^{(k+1)}-1} + \dots + q^{2^{m+k-1}-1}\big) \\
&= \alpha q^{2^{(k)}-1} \big(1 + q^{2^{(k)}} + \dots + (q^{2^{(k)}})^{2^{m-1}-1}\big) \\
&\leq 2\alpha q^{2^{(k)}-1}.
\end{aligned} \tag{12.10}$$

This shows that indeed $(x^{(k)}) \subset D$ form a Cauchy sequence because $q < \frac{1}{2}$. In the Banach space $\mathbb{R}^n$, the limit

$$z = \lim_{k \to \infty} x^{(k)} \in \mathbb{R}^n$$

exists. Passing to the limit $k \to \infty$, we obtain with (12.9)

$$\|z - x^{(0)}\| \leq r,$$

such that $z \in B_r(x_0)$. For the limit $m \to \infty$ in (12.10) we establish the error estimate

$$\|x^{(k)} - z\| \le 2\alpha q^{2^{(k)}-1}, \quad k = 0, 1, \dots$$

(iv) It remains to show that $z \in B_r(x^{(0)})$ is a root of f. The Newton iteration rule as well as the condition (12.4) yield

$$\begin{aligned}\|f(x^{(k)})\| &= \|Df(x^{(k)})(x^{(k)} - x^{(k-1)})\| \\ &\le \|Df(x^{(k)}) - Df(x^{(0)}) + Df(x^{(0)})\| \, \|x^{(k+1)} - x^{(k)}\| \\ &\le \Big(L\|x^{(k)} - x^{(0)}\| + \|Df(x^{(0)})\|\Big) \|x^{(k+1)} - x^{(k)}\| \to 0 \quad (k \to \infty).\end{aligned}$$

Hence, it holds

$$f(x^{(k)}) \to 0, \quad k \to \infty.$$

The continuity of f implies $f(z) = 0$.

(v) Finally, we show that the root $z \in B_r(x_0)$ is unique. This is obtained with the help of the contraction property and the formulation of the Newton scheme as fixed-point iteration. Each root of $f(\cdot)$ is a fixed-point of the simplified Newton iteration

$$g(x) := x - Df(x^{(0)})^{-1} f(x)$$

The fixed-point iteration function $g(\cdot)$ is Lipschitz continuous. With (12.8), it holds

$$\begin{aligned}g(x) - g(y) &= x - y - Df(x^{(0)})^{-1} f(x) + Df(x^{(0)})^{-1} f(y) \\ &= Df(x^{(0)})^{-1}(f(y) - f(x) - Df(x^{(0)})(y - x)) \\ &\le \underbrace{\|Df(x^{(0)})^{-1}\|}_{\le \beta} rL\|y - x\| \\ &\le \beta L r \|y - x\|.\end{aligned}$$

With $r = 2\alpha$, it follows $\beta L r \le 2\alpha\beta L \le 2q < 1$. This yields that g is a contraction. Banach's fixed-point theorem yields that we have only one fixed-point and consequently only one root. $\square$

Remark 12.3. *The Newton-Kantorovich theorem differs in several points from well-posedness results for one-dimensional Newton schemes. For instance, the root is not an assumption, but follows within the proof. Moreover, the Newton-Kantorovich theorem only requires a Lipschitz continuous first derivative, but not a twice differentiable function.*

From Thoerem 12.8, the following convergence result can be inferred:

Corollary 12.9. *Let $D \subset \mathbb{R}^n$ be open and $f : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be twice continuously differentiable. We assume that $z \in D$ is a root with regular Jacobian $Df(z)$. Then, Newton's method is locally convergent, i.e., there exists a neighborhood B around z such that the Newton scheme converges for all starting values $x^{(0)} \in B$.*

Example 12.1 (Newton's method in $\mathbb{R}^n$)**.** *We seek the root of the function*

$$f(x_1, x_2) = \begin{pmatrix} 1 - x_1^2 - x_2^2 \\ (x_1 - 2x_2)/(1/2 + x_2) \end{pmatrix}$$

with the Jacobian

$$Df(x) = \begin{pmatrix} -2x_1 & -2x_2 \\ \frac{2}{1+2x_2} & -\frac{4+4x_1}{(1+2x_2)^2} \end{pmatrix}.$$

The roots of f are given by

$$x \approx \pm(0.894427, 0.447214).$$

We start the iteration with $x^{(0)} = (1,1)^T$ and obtain the iterates

$$x^1 \approx \begin{pmatrix} 1.14286 \\ \underline{0.}357143 \end{pmatrix}, \qquad x^2 \approx \begin{pmatrix} \underline{0.}92659 \\ \underline{0.44}2063 \end{pmatrix},$$
$$x^3 \approx \begin{pmatrix} \underline{0.894}935 \\ \underline{0.447}349 \end{pmatrix}, \qquad x^4 \approx \begin{pmatrix} \underline{0.894427} \\ \underline{0.447214} \end{pmatrix}.$$

After only four iterations, the first six digits are exact.

12.5.6 Globalization of Newton's method

Each step of Newton's method consists of the parts

1. Computation of the Jacobian;
2. Solving the linear equation system;
3. Update and monitoring the convergence.

In contrast to the one-dimensional case, the computation of the derivative in $\mathbb{R}^n$ may be cumbersome, since n^2 partial derivatives must be calculated:

$$(Df)_{ij} = \frac{\partial f_i}{\partial x_j}, \quad i,j = 1,\dots,n.$$

This can be a non-neglibible computational cost. In the second step of Newton's method, we must solve a linear equation system. Using for instance LU decomposition (as we discussed before in detail), it requires $O(n^3)$ operations. The solution requires then $O(n^2)$ operations.

As well we can work with simplified Newton methods. If the derivative $f'(x^{(k)})$ is not computed according to the current right hand side, but to some other value $f'(c)$, we still have convergence, but only with a linear order. In the higher-dimensional case, we obtain

Listing 12.1: **(Simplified Newton's method)**
Let $f \in C^1(D)$ be given, an initial guess $x^{(0)} \in D$ and $c \in D$.

```
Compute the Jacobian $Df(c)$
Construct $LU$ decomposition $Df(c) = L(c)R(c)$
Iterate for $k = 1, 2, \dots$
  $L(c)R(c)w^{(k)} = -f(x^{(k-1)})$
  $x^{(k)} = x^{(k-1)} + w^{(k)}$
```

The two expensive steps of Newton's method are only required once. The iteration can be performed efficiently with forward and backward substitution. This can be further improved when the linearization point $c \in \mathbb{R}^n$ is selected in a dynamic way, for instance at $c = x^{(3)}$, $c = x^{(6)}, \dots$ or when the convergence behavior detoriates. As monitor, we can observe the residual of two subsequent iterates

$$\rho_k = \frac{|f(x^{(k)})|}{|f(x^{(k-1)})|}.$$

If $\rho_k > \rho_0$ (bad convergence), then with $c = x^{(k)}$, we perform an update of the Jacobian.

The second challenge is the choice of a good initial guess. Enlarging the convergence radius, i.e., globalization, can be obtained with damping. Here, step *5)* in Algorithm 12.1 is replaced by

$$x^{(k)} = x^{(k-1)} + \omega_k w^{(k)}$$

with a damping parameter $\omega_k \in (0,1]$. For the Newton iteration, this reads

$$x^{(k+1)} = x^{(k)} - \omega_k Df(x^{(k)})^{-1} f(x^{(k)}).$$

Proposition 12.10 (Damped Newton's method). *Let the assumptions from Theorem 12.8 hold true except for the condition on the initial value (12.6). The damped Newton iteration*

$$Df(x^{(k)})\delta x = -f(x^{(k)}), \quad x^{(k+1)} = x^{(k)} + \omega_k \delta x$$

with

$$\omega_k := \min\{1, \frac{1}{\alpha_k \beta L}\}, \quad \alpha_k := \|Df(x^{(k)})^{-1} f(x^{(k)})\|$$

generates a sequence $(x^{(k)})_{k\in\mathbb{N}}$*, for which after* k_* *steps*

$$q_* := \alpha_{k_*}\beta L < \frac{1}{2}$$

is fulfilled. For $k > k_*$ *the squence* $x^{(k)}$ *converges quadratically and we have the a priori error estimate*

$$\|x^{(k)} - z\| \leq \frac{\alpha}{1-q_*} q_*^{2^{(k)}-1}, \quad k \geq k_*.$$

Proof. We show the monotone convergence in the sense of $\|f(x^{(k+1)})\| \leq \|f(x^{(k)})\|$ of the damped Newton iteration. For $x^{(0)}$, the set

$$D_0 := \{x \in D : \ \|f(x)\| \leq \|f(x^{(0)})\|\}$$

is closed and non-empty. We define the damped iteration

$$x_\omega = g_\omega(x) := x - \omega Df(x)^{-1} f(x).$$

For $x \in D_0$, let $\omega_{max} \leq 1$ be the maximal damping parameter such that

$$f(x_\omega) \leq f(x^{(0)}) \quad 0 \leq \omega \leq \omega_{max} \leq 1.$$

We define

$$h(\omega) := f(x_\omega)$$

with $h(0) = f(x)$. For $h(\cdot)$ it holds further

$$h'(\omega) = \frac{d}{d\omega} f(x - \omega Df(x)^{-1} f(x)) = -Df(x_\omega)Df(x)^{-1} f(x).$$

With $h'(0) = -f(x) = -h(0)$, it follows

$$\begin{aligned} f(x_\omega) - f(x) = h(\omega) - h(0) = \int_0^\omega h'(s)\,ds &= -\int_0^\omega Df(x_s)Df(x)^{-1} f(x)\,ds \\ &= -\int_0^\omega \Big\{(Df(x_s) - Df(x))Df(x)^{-1} f(x) + f(x)\Big\}\,ds. \end{aligned}$$

Hence,

$$f(x_\omega) = \left(-\int_0^\omega (Df(x_\omega) - Df(x))Df(x)^{-1}\,ds\right) f(x) + (1-\omega) f(x)$$

and we estimate

$$\begin{aligned} \|f(x_\omega)\| &\leq (1-\omega)\|f(x)\| + \beta L \int_0^\omega \|x_s - x\|\,ds \|f(x)\| \\ &= (1-\omega)\|f(x)\| + \beta L \int_0^\omega s\|Df(x)^{-1} f(x)\|\,ds \|f(x)\| \\ &\leq \left(1 - \omega + \alpha_x \beta L \frac{\omega^2}{2}\right) \|f(x)\|, \quad \alpha_x = \|Df(x)^{-1} f(x)\|. \end{aligned}$$

Now, we determine $0 \le \omega \le \omega_{max} \le 1$ such that the expression

$$\rho(\omega) := \left|1 - \omega + \alpha_x \beta L \frac{\omega^2}{2}\right|$$

is minimal. The minimum is obtained for

$$\omega_{min} = \min\left\{1, \frac{1}{\alpha_x \beta L}\right\}.$$

Then, it holds

$$1 - \omega_{min} + \frac{\omega_{min}^2}{2}\alpha_x \beta L \le 1 - \frac{1}{2\alpha_x \beta L} < 1.$$

For this choice of the damping parameter, we have monotone convergence:

$$\|f(x^{(k+1)})\| \le \left(1 - \frac{1}{2\alpha_k \beta L}\right) \|f(x^{(k)})\|$$

After a finite number of steps, it holds

$$\frac{\alpha_k \beta L}{2} \le \frac{\beta^2 L}{2}\|f(x^{(k)})\| < 1$$

and we arrive at the quadratic convergence radius. □

Remark 12.4 (Globalization). *The damped Newton scheme is a globalization strategy. The condition for the initial guess*

$$x^{(0)} \in D: \quad \underbrace{\|Df(x^{(0)})^{-1} f(x^{(0)})\|}_{=\alpha_0} \beta L < \frac{1}{2}$$

is not required to be necessarily fulfilled. If this product is too big, we can iterate with the damped iteration until $\alpha_k \beta L$ with $\alpha_k = \|Df(x^{(k)})^{-1} f(x^{(k)})\|$ fulfills the condition. As said, globalization strategies yield convergence in a larger region, but therein, however, the convergence is initially usually slow with the rate

$$\rho \le 1 - \frac{1}{2\alpha_k \beta L}.$$

For large α_k, we can choose ρ close to 1.

It is important that the switch to fast convergence be in an automated fashion. Here, this is given since

$$\alpha_k \beta L < \frac{1}{2} \quad \Rightarrow \quad \omega_k = \min\left\{1, \frac{2}{\alpha_k \beta L}\right\} = 1.$$

A difficulty remains however as to the choice of the damping parameter ω_k. Indeed the values

$$\alpha_k = \|Df(x^{(k)})^{-1} f(x^{(k)})\|, \quad \beta = \max_{x \in D} \|Df(x)^{-1}\|,$$

and the Lipschitz constant of $Df(x)$ are in general not computable. A common strategy (see also in numerical optimization [21]) is a line-search technique.

Listing 12.2: **(Line-Search)**
We are given $f \in C^1(D)$, and $x^{(0)} \in D$ and a parameter $\sigma \in (0,1)$ and a maximal number of line search steps $L_{max} \in \mathbb{N}$.

```
For $k = 0, 1, 2, \dots,$ we iterate
    $Df(x^k) w^{(k)} = -f(x^{(k)})$
    Set $\omega_0^k = 1$
    For $l = 0, 1, \dots, L_{max}$ iterate
        $x^{(k+1)} = x^{(k)} + \omega_l w^{(k)}$
        Stop, if $\|f(x^{(k+1)})\| < \|f(x^{(k)})\|$
        $\omega_{l+1}^{(k)} = \sigma \omega_l^{(k)}$
```

Using line search we enforce a monotonic contraction of the residuals. We first try a full Newton step with $\omega = 1$. As long as $\|f(x^{(k+1)})\| > \|f(x^{(k)})\|$, we reduce ω. Common values for σ are $\sigma = \frac{1}{2}$ or $\sigma = \frac{1}{4}$. Further modifications are described in numerical optimization [21] and also so-called error-oriented Newton methods going back to [9] and recently applied in coupled variational inequality systems in the monograph [28].

12.6 Newton's method for a coupled, nonlinear system of ODEs (complement)

We discuss an example motivated from the later Chapter 17. Therein, ordinary differential equations (ODEs) shall be solved at some time point t_n. Often, such systems result into nonlinear problems, which can be solved by methods introduced in the current chapter. We neglect all details concerning ODEs and refer the reader to Chapter 17, but concentrate on all relevant steps for the nonlinear solution.

We look for two unknowns $p(t) : [0,T] \to \mathbb{R}$ and $s(t) : [0,T] \to \mathbb{R}$, which are determined by a coupled nonlinear system of two ODEs:

$$
\begin{aligned}
p'(t) &= r_p p(t) - \alpha s(t)p(t)\\
s'(t) &= r_s s(t) + \mu\alpha p(t)s(t)
\end{aligned}
$$

The derivatives are defined with respect to time, i.e., $p'(t) := \frac{d}{dt}p(t)$ and $s'(t) := \frac{d}{dt}s(t)$. Furthermore, we deal with the parameters $r_p > 0, \alpha > 0, r_s > 0, \mu > 0$, which are independent of the time t.

When we discretize the time-continuous derivatives by an implicit procedure (taking the unknown as well on the right hand side), we obtain

$$
\begin{aligned}
\frac{p^n - p^{n-1}}{\Delta t} &= r_p p^n - \alpha s^n p^n\\
\frac{s^n - s^{n-1}}{\Delta t} &= r_s s^n + \mu\alpha p^n s^n,
\end{aligned}
$$

where $\Delta = t_n - t_{n-1}$ is the time step size (difference between two time points). Since we look simultaneously for the time-discretized p^n and s^n, we easily see that the system is nonlinear due to the couplings $s^n p^n$ (1st equation) and $p^n s^n$ (2nd equation).

One possibility to solve this nonlinear system is to formulate a root-finding problem of the form

$$F(p^n, s^n) = 0$$

with

$$F(p^n, s^n) = \begin{pmatrix} p^n - p^{n-1} - \Delta t[r_p p^n - \alpha s^n p^n] \\ s^n - s^{n-1} - \Delta t[r_s s^n + \mu\alpha p^n s^n] \end{pmatrix}.$$

For Newton's method, we introduce the iteration index j. Formally, we have: set $p^{n,0} := p^{n-1}$ and $s^{n,0} := s^{n-1}$ as initial guess. Then, a defect-correction variant reads:

$$
\begin{aligned}
F'(p^{n,j}, s^{n,j})(w,v)^T &= -F(p^{n,j}, s^{n,j})\\
(p^{n,j+1}, s^{n,j+1})^T &= (p^{n,j}, s^{n,j})^T + (w,v)^T
\end{aligned}
$$

until some stopping criterion, e.g.,

$$\max(\|p^{n,j+1} - p^{n,j}\|_2, \|s^{n,j+1} - s^{n,j}\|_2) < TOL$$

with for instance $TOL = 1e - 10$. The derivative F' is the Jacobian matrix; here $F' \in \mathbb{R}^{2\times 2}$. The crucial point is how the Jacobian reads in detail. Formally, we have

$$F'(p^n, s^n) = \begin{pmatrix} \partial_p F_1, \partial_s F_1 \\ \partial_p F_2, \partial_s F_2 \end{pmatrix}$$

Specifically, for the above system, we obtain

$$\begin{aligned}
\partial_p F_1 &= 1 - 0 - \Delta t[r_p - \alpha s^n]\\
\partial_s F_1 &= 0 - 0 - \Delta t[-\alpha p^n]\\
\partial_p F_2 &= 0 - 0 - \Delta t[\mu\alpha s^n]\\
\partial_s F_2 &= 1 - 0 - \Delta t[r_s + \mu\alpha p^n]
\end{aligned}$$

The full Newton system at a time point t^n for the iteration j then reads:

$$\begin{pmatrix} 1 - \Delta t[r_p - \alpha s^{n,j}] & -\Delta t[-\alpha p^{n,j}] \\ -\Delta t[\mu\alpha s^{n,j}] & 1 - \Delta t[r_s + \mu\alpha p^{n,j}] \end{pmatrix} \begin{pmatrix} w \\ v \end{pmatrix} = - \begin{pmatrix} p^{n,j} - p^{n-1} - \Delta t[r_p p^{n,j} - \alpha s^{n,j} p^{n,j}] \\ s^{n,j} - s^{n-1} - \Delta t[r_s s^{n,j} + \mu\alpha p^{n,j} s^{n,j}] \end{pmatrix}$$

$$\begin{pmatrix} p^{n,j+1} \\ s^{n,j+1} \end{pmatrix} = \begin{pmatrix} p^{n,j} \\ s^{n,j} \end{pmatrix} + \begin{pmatrix} w \\ v \end{pmatrix}.$$

Until

$$\left\| \begin{pmatrix} p^{n,j+1} \\ s^{n,j+1} \end{pmatrix} - \begin{pmatrix} p^{n,j} \\ s^{n,j} \end{pmatrix} \right\|_2 < TOL$$

With this procedure, we then have obtained the solution for the current time point and set

$$p^n := p^{n,j+1}, \qquad s^n := s^{n,j+1}.$$

Afterward, we can proceed to the next time point t_{n+1}; for details we refer to Chapter 17.

We finally notice as well that a simplifed (approximate) Newton scheme is obtained by neglecting the off-diagonal terms. Then, the defect step reads

$$\begin{pmatrix} 1 - \Delta t[r_p - \alpha s^{n,j}] & 0 \\ 0 & 1 - \Delta t[r_s + \mu\alpha p^{n,j}] \end{pmatrix} \begin{pmatrix} w \\ v \end{pmatrix} = - \begin{pmatrix} p^{n,j} - p^{n-1} - \Delta t[r_p p^{n,j} - \alpha s^{n,j} p^{n,j}] \\ s^{n,j} - s^{n-1} - \Delta t[r_s s^{n,j} + \mu\alpha p^{n,j} s^{n,j}] \end{pmatrix}$$

12.7 Iteration schemes in nonlinear optimization (complement)

12.7.1 Linear regression

A typical situation for approximation problems is linear regression. Here, often given data in form of (x_n, p_n) for $n = 1, \dots, N$ is available and a function of the form

$$p(x) = b + mx$$

is sought that best approximates in some mathematical sense (choice of the norm!) these data. The specific goal is to determine the parameters $b \in \mathbb{R}$ and $m \in \mathbb{R}$.

Here, a least-squares approximation is one possibility:

$$S := S(m, b) = \frac{1}{N}\sum_{n=1}^{N}(p_n - (b - mx_n))^2.$$

The function S is the so-called cost (or objective) functional. The goal is solution of the minimization problem:

$$\min_{m,b} S(m, b).$$

We also refer the reader back to chapter 11 for an introduction to nonlinear optimization. For the numerical solution, we present two options.

The first one is the steepest descent method. We just need to calculate the two partial derivatives with respect to m and b:

$$\frac{\partial S}{\partial m} = \frac{2}{N}\sum_{k=1}^{N} -x_k(p_k - (mx_k + b)) \tag{12.11}$$

$$\frac{\partial S}{\partial b} = \frac{2}{N}\sum_{k=1}^{N} -(p_k - (mx_k + b)). \tag{12.12}$$

The final algorithm is then given by: for $l = 1, 2, 3, \dots, N_{max}$, iterate such that

$$\begin{pmatrix} m_{l+1} \\ b_{l+1} \end{pmatrix} = \begin{pmatrix} m_l \\ b_l \end{pmatrix} - \rho \begin{pmatrix} \frac{\partial S(m_l,b_l)}{\partial m} \\ \frac{\partial S(m_l,b_l)}{\partial b} \end{pmatrix}.$$

The second option consists in exploiting the fact the cost function is quadratic, hence the first order optimality condition

$$\nabla S(m, b) = 0$$

is a simple linear system given by (12.11)-(12.12). It can be solved directly.

12.7.2 Neural networks

In the solution of machine learning problems with the area of neural networks, the task is mathematically exactly the same as in the previous subsection, except that linearity is lost. For a concise introduction from a mathematical viewpoint, we refer to [16]. We need to determine weights w and biases b. Determining those two quantities is called training. Let us put both of them into a function $a^{[L]} := a^{[L]}(w, b)$, which is the global activation activation function at the output layer L. Let N pieces of data (training points) and corresponding target outputs be denoted as $(x^n, y^n), n = 1, \dots, N$. Then, a quadratic cost functional (as before) can be defined:

$$C = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\|y^n - a^{[L]}(x^n)\|_2^2.$$

This problem can be solved as in the previous section by gradient descent. However, since usually N is very big (i.e., a big data problem) the computational cost may be prohibitively high. For this reason, a so-called stochastic gradient method is often employed in which the individual gradients running over all training points N is replaced by a single randomly chosen, training point. There is a vast literature on machine learning and neural networks because of its enormous applications in science, technical applications, and our society. The important message in the frame of this class is that the concepts, and numerical methods presented here, are already the key ingredients of algorithms used for machine learning.

Let us now have a look at the activation functions that are classically used. To fix ideas, consider a single layer ($L = 1$). Typically, writing as subscripts the components of vectors, one considers

$$(a^{[L]}(x))_i = S\left(\sum_{j=1}^{M} w_{ij}x_j - b_i\right)$$

with the sigmoid function

$$S(t) = \frac{1}{1 + e^{-t}},$$

see Fig. 12.1. This function approximates the thresholding phenomenon that occurs in natural neurons, while being differentiable (very helpful for optimization!).

In the above notation M is the number of inputs, and i covers the number of outputs.

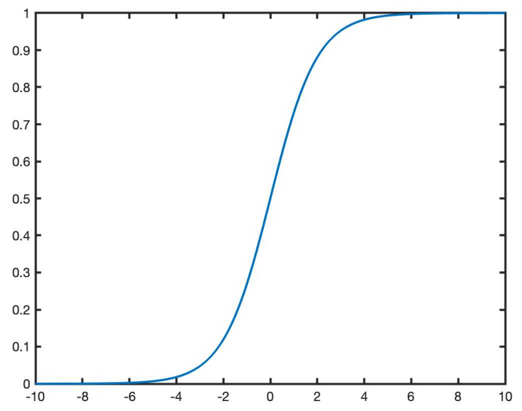


Figure 12.1: Sigmoid function.

We can incorporate intermediate or hidden layers just by composing such operations. The gradient is obtained by the chain rule, and performing Jacobian matrix multiplications in a clever ordering (actually from output to input) is called error back-propagation.

Chapter 13

Interpolation and approximation

In this chapter, we discuss interpolation and approximation. The former consists in constructing a simple function (often a polynomial) through certain given points. With the help of such simple functions further operations such as numerical differentiation and numerical integration can be constructed. Approximation is more general and does not necessarily need to fit in certain given points. One example is linear regression.

13.1 Polynomial interpolation

13.1.1 Introduction

Let $(x_0, ..., x_n)$ and $(y_0, ..., y_n)$ be two families of real numbers, with the so-called nodes x_i's pairwise distinct. We denote by $\mathcal{P}_n$ the set of polynomial functions of degree $\leq n$. We investigate the following issues:

1. Does there exist a polynomial $P \in \mathcal{P}_n$ such that $P(x_i) = y_i \ \forall i = 0, ..., n$? What about uniqueness?

2. How to calculate it?

3. How does it behave in between the points x_i?

13.1.2 Existence, uniqueness, expression

Theorem 13.1. *There exists a unique $P \in \mathcal{P}_n$ such that $P(x_i) = y_i \ \forall i = 0, ..., n$. It is called Lagrange interpolating polynomial.*

Let us write the interpolating polynomial as $P(x) = \sum_{k=0}^{n} a_k x^k$. The coefficients must solve the linear system

$$\begin{cases} a_0 + a_1 x_0 + \cdots + a_n x_0^n &= & y_0 \\ a_0 + a_1 x_1 + \cdots + a_n x_1^n &= & y_1 \\ \vdots & & \vdots \\ a_0 + a_1 x_n + \cdots + a_n x_n^n &= & y_n, \end{cases}$$

which admits the matrix representation

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The determinant of the matrix of this system, called Vandermonde determinant (a particular case was given as exercise), is equal to

$$D_n = \prod_{0 \leqslant i < j \leqslant n} (x_j - x_i) \ \neq 0.$$

Hence this system admits a unique solution. In fact, there is a direct expression of the interpolating polynomial, given by

$$\boxed{P(x) = \sum_{j=0}^{n} y_j \varphi_j(x), \qquad \varphi_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.}$$

Indeed, the functions $\varphi_0, ..., \varphi_n$ belong to $\mathcal{P}_n$ and satisfy $\varphi_i(x_j) = \delta_{ij}$, with $\delta_{ij} = 1$ if $i = j$, $\delta_{ij} = 0$ if $i \neq j$ (Kronecker symbol).

An example is shown in Fig. 13.1.

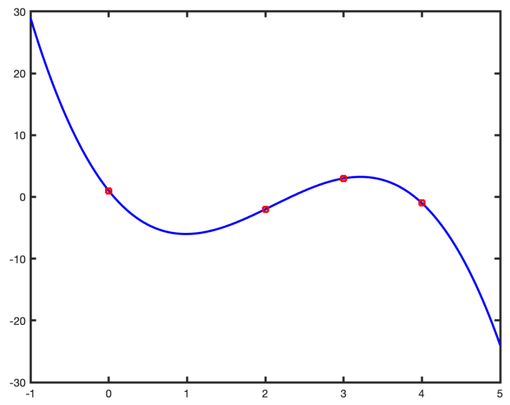


Figure 13.1: Interpolating polynomial of degree 3. The data points are the red squares.

13.1.3 Interpolation error (complement)

For a given function f we now assume that the data points are such that $y_j = f(x_j)$ $\forall j = 0, ..., n$. A natural question is then: how does the Lagrange interpolating polynomial of the families $(x_0, ..., x_n)$ and $(f(x_0), ..., f(x_n))$, called Lagrange interpolating polynomial of f, approximate f? An answer is provided by the following theorem.

Theorem 13.2. *Let f be a function of class $\mathcal{C}^{n+1}$ on $[a, b]$ and P be its Lagrange interpolating polynomial at the points $x_0, x_1, \ldots, x_n \in [a, b]$. Then, for all $x \in [a, b]$, there exists $\xi \in [a, b]$ such that*

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \pi_{n+1}(x), \qquad \pi_{n+1} = \prod_{j=0}^{n} (x - x_j).$$

We obviously always have $|\pi_{n+1}(x)| \leq (b - a)^{n+1}$. If the points are equidistant, i.e. $x_j = a + jh$, $h = (b - a)/n$, then we have the finer estimate $|\pi_{n+1}(x)| \leq n! h^{n+1}/4$. This yields

$$|f(x) - P(x)| \leq \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)| \frac{h^{n+1}}{4(n+1)}.$$

We could expect that $f(x) - P(x) \to 0$ when $n \to +\infty$. However this is not always the case since the derivatives $|f^{(n+1)}(\xi)|$ may increase quickly when n grows. It can even happen that $|f(x) - P(x)| \longrightarrow +\infty$: this is called the Runge phenomenon, due to oscillations of P between the points x_i.

A classical example is the function $f : t \mapsto \frac{1}{1+25t^2}$ on $[-1, 1]$, see figure 13.2.

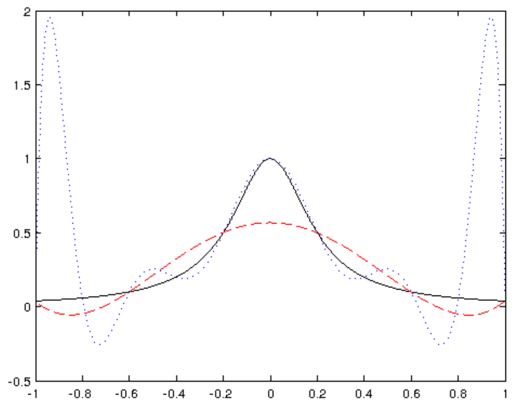


Figure 13.2: Runge phenomenon. The function f is depicted in solid line. The Lagrange interpolating polynomials of f of degree 5 and 10 with equidistant points are depicted in dashed and dotted lines, respectively.

13.2 Numerical differentiation

13.2.1 Approximation of the first derivative

Consider a "sufficiently smooth" function $f : \mathbb{R} \to \mathbb{R}$ and two prescribed points $x_0, x_1 \in \mathbb{R}$ where f takes known values. Hence we can interpolate f linearly at the nodes x_0 and x_1 by

$$p_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1).$$

Now we can approximate the derivative $f'(x)$ through the corresponding derivative of the polynomial

$$\boxed{p_1'(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.}$$

Proposition 13.3 (Difference quotient of the first derivative)**.** *Let $f \in C^2[a,b]$. For two points $x_0, x_1 \in [a,b]$ with $h := x_1 - x_0 > 0$ we have the left-sided and right-sided difference quotients*

$$\begin{aligned}\frac{f(x_1) - f(x_0)}{h} &= f'(x_0) + \frac{1}{2} h f''(x_0) + O(h^2),\\ \frac{f(x_1) - f(x_0)}{h} &= f'(x_1) - \frac{1}{2} h f''(x_0) + O(h^2).\end{aligned}$$

For $f \in C^4[a,b]$, it holds $x_0, x_0 - h, x_0 + h \in [a,b]$ for the central difference quotient

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = f'(x_0) + \frac{1}{4} h^2 f^{(iv)}(x_0) + O(h^3).$$

Proof. (i) With Taylor expansion, it holds

$$f(x_0 \pm h) = f(x_0) \pm h f'(x_0) + \frac{1}{2} h^2 f''(x_0) \pm \frac{1}{6} h^3 f'''(x_0) + \frac{1}{24} h^4 f^{(iv)}(\xi)$$

with some $\xi \in [a,b]$. It follows for the right-sided difference quotient

$$\frac{f(x_0 + h) - f(x_0)}{h} = \frac{h f'(x_0) + \frac{1}{2} h^2 f''(x_0) + O(h^3)}{h} = f'(x_0) + \frac{1}{2} h f''(x_0) + O(h^2).$$

The proof for the left-sided difference quotient is similar.

(ii) For the central difference quotient, the even powers of h cancel due to symmetry:

$$f(x_0+h) - f(x_0-h) = 2f'(x_0) + \frac{1}{3}h^3 f'''(x_0) + O(h^4).$$

This yields the final result. □

We see in the previous proofs that the choice of the nodal values plays a crucial role. If they are chosen in some clever way, we obtain better convergence than expected. This phenomenon is known as super-convergence, which is the case for the central difference quotient. The key point is symmetry. This can be seen as follows: let the derivative in x_0 be constructed through $x_0 - h$ and $x_0 + 2h$ (no symmetry). Then, Taylor around x_0 yields

$$\begin{aligned}
&\frac{f(x_0+2h) - f(x_0-h)}{3h} \\
&\quad = f'(x_0) + \left(2hf''(x_0) + \frac{8}{6}h^2 f'''(x_0)\right) - \left(\frac{1}{2}hf''(x_0) - \frac{1}{6}h^2 f'''(x_0)\right) + O(h^3) \\
&\qquad\qquad f'(x_0) + \frac{3}{2}hf''(x_0) + O(h^2).
\end{aligned}$$

In this case, higher order powers of h do not cancel and we only obtain linear convergence, but not quadratic.

13.2.2 Approximation of the second derivative (complement)

We interpolate f with the help of three points $x_0 - h$, x_0, $x_0 + h$ using a quadratic polynomial:

$$\begin{aligned}
p_2(x) = {} & f(x_0-h)\frac{(x_0-x)(x_0+h-x)}{2h^2} \\
& + f(x_0)\frac{(x-x_0+h)(x_0+h-x)}{h^2} + f(x_0+h)\frac{(x-x_0+h)(x-x_0)}{2h^2}.
\end{aligned}$$

In $x = x_0$ it holds for the first and second derivative:

$$p_2'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h}, \quad \boxed{p_2''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2}.}$$

For the first derivative, we obtain the same result as in the previous subsection.

With Taylor, we obtain for the second derivative

$$p_2''(x_0) = \frac{-2f(x_0) + f(x_0-h) + f(x_0+h)}{h^2} = f''(x_0) + \frac{1}{12}h^2 f^{(iv)}(x_0) + O(h^4).$$

Using p_2 we can also derive a one-sided difference quotient for the second derivative. Thus, we approximate $f''(x_0-h) \approx p_2''(x_0-h)$. With Taylor, we obtain

$$p''(x_0-h) = f''(x_0-h) + hf'''(x_0-h) + O(h^2),$$

which is only an approximation of first order. Again, the choice of good support points is crucial.

13.3 Numerical integration (complement)

13.3.1 Goal

Let $f : [a,b] \to \mathbb{R}$ be a continuous function. Our goal is to calculate

$$\int_a^b f(x)\,dx.$$

In general, we do not know any explicit function F such that $F' = f$. Then we have to content ourselves with an approximation of the integral.

13.3.2 General principle

We first introduce a subdivision $\sigma = (x_0, x_1, \cdots, x_N)$ of $[a, b]$ (i.e., $x_0 = a < x_1 < \cdots < x_N = b$). The stepsize of the subdivision σ is denoted by :

$$h = \max_{0 \leqslant k \leqslant N-1} |x_{k+1} - x_k|.$$

Obviously the integral can be decomposed as

$$\int_a^b f(x)\,dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)\,dx. \tag{13.1}$$

Each integral $\int_{x_k}^{x_{k+1}} f(x)\,dx$ will be approximated by a so-called elementary quadrature formula.

Note that the change of variable

$$x = x_k + \frac{x_{k+1} - x_k}{2}(t+1)\ , \quad t \in [-1, 1],$$

yields

$$\int_{x_k}^{x_{k+1}} f(x)\,dx = \frac{x_{k+1} - x_k}{2} \int_{-1}^{1} g_k(t)\,dt \tag{13.2}$$

with $g_k(t) = f(x_k + \frac{x_{k+1} - x_k}{2}(t+1))$. Therefore it is sufficient to define elementary quadrature formulas for the standard interval $[-1, 1]$.

13.3.3 Elementary quadrature formula

Let $g : [-1, 1] \to \mathbb{R}$ be a continuous function. Our goal is to approximate $\int_{-1}^{1} g(x)\,dx$ by a number $J(g)$.

Definition 13.1. *An elementary quadrature formula is defined by*

- *points $t_0, ..., t_L$ such that $-1 \leq t_0 < t_1 < ... < t_L \leq 1$ called nodes,*
- *coefficients $w_0, w_1, ..., w_L \in \mathbb{R}$ called weights.*

It reads

$$J(g) = \sum_{j=0}^{L} w_j g(t_j).$$

13.3.4 Simplest rules: box, mid-point, trapezoidal, Simpson

Let us work on the interval $[a, b]$. The simplest quadrature rules are the left-sided box rule

$$I^0(f) = (b-a)f(a),$$

the right-sided box rule

$$I^0(f) = (b-a)f(b),$$

the mid-point rule

$$I^0(f) = (b-a)f(\frac{a+b}{2}),$$

the trapezoidal rule

$$I^1(f) = \frac{b-a}{2}(f(a) + f(b)),$$

and the Simpson rule

$$I^2(f) = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b)).$$

13.3.5 Composite quadrature formula

In view of (13.1) and (13.2) it is natural to set the following definition.

Definition 13.2. *The composite quadrature formula associated with the subdivision σ and the elementary quadrature formula $J(g) = \sum_{j=0}^{L} w_j g(t_j)$ is :*

$$I_{\sigma,J}(f) = \sum_{k=0}^{N-1} \frac{x_{k+1} - x_k}{2} \sum_{j=0}^{L} w_j f(\xi_{k,j}) \quad \textit{with} \quad \xi_{k,j} = x_k + \frac{x_{k+1} - x_k}{2}(t_j + 1).$$

Here are classical examples.

	elementary formula $J(g)$	corresponding composite formula $I_{\sigma,J}(f)$
leftpoint formula	$2g(-1)$	$\sum_{k=0}^{N-1}(x_{k+1} - x_k)f(x_k)$
rightpoint formula	$2g(1)$	$\sum_{k=0}^{N-1}(x_{k+1} - x_k)f(x_{k+1})$
midpoint formula	$2g(0)$	$\sum_{k=0}^{N-1}(x_{k+1} - x_k)f(\frac{x_k+x_{k+1}}{2})$
trapezoidal formula	$g(-1) + g(1)$	$\sum_{k=0}^{N-1} \frac{x_{k+1}-x_k}{2}\big(f(x_k) + f(x_{k+1})\big)$

13.3.6 Order of a quadrature formula

Definition 13.3. *An elementary quadrature formula J is said to be of order at least n if it is exact for polynomials in $\mathcal{P}_n$, i.e.,*

$$J(g) = \int_{-1}^{1} g(t)\, dt \qquad \forall g \in \mathcal{P}_n.$$

It is of order exactly n if, in addition, there exists $g \in \mathcal{P}_{n+1}$ such that

$$J(g) \neq \int_{-1}^{1} g(t)\, dt.$$

For the previous examples it holds:

	exact order
left/right-point formula	0
midpoint formula	1
trapezoidal formula	1

It might seem surprising that the midpoint and trapezoidal formulas have the same order. This is due to the fact that the midpoint is at a special location that enables to gain orders for a fixed number of nodes. Such a node is called a Gauss point. We will not enter into this (beautiful and classical) theory.

13.3.7 Interpolatory quadrature

All examples of quadrature formulas we have mentioned so far belong to the class of interpolatory quadrature formulas, which is the standard way of constructing quadrature formulas.

Definition 13.4. *An interpolatory quadrature formula is an elementary quadrature formula such that*

$$J(g) = \int_{-1}^{1} P_g(t)\, dt,$$

where P_g is the Lagrange interpolating polynomial of g at the points $t_0, ..., t_N$.

Then, from

$$P_g(t) = \sum_{j=0}^{L} g(t_j)\varphi_j(t), \qquad \varphi_j(t) = \prod_{k\neq j} \frac{t - t_k}{t_j - t_k},$$

we obtain the expression

$$J(g) = \sum_{j=0}^{L} w_j g(t_j), \qquad w_j = \int_{-1}^{1} \varphi_j(t)\, dt.$$

By construction, the formula is of order at least L. It is easy to prove the reciprocal statement.

Theorem 13.4. *The elementary quadrature formula $J(g) = \sum_{j=0}^{L} w_j g(t_j)$ is of order at least L if and only if it is an interpolatory quadrature formula.*

13.3.8 Gauss quadrature

We finally discuss a class of quadrature rules that are often used in practice. So far, the order (of convergence) of a quadrature rule was determined through the number of support points and the polynomial function that was constructed using these points. The question is whether the quadrature order can be further enhanced or whether our developments are optimal.

A careful revision entails that the mid-point rule has a better convergence order than expected despite the fact that the number of support points is the same as for the box rule. The reason is symmetry (see also the section on difference quotients; super-convergence).

This raises the question whether this is a general principle that an optimal distribution of the support points enhances the convergence order. The answer is positive and leads to the so-called Gauss quadrature.

So far a quadrature rule for $n+1$ support points $x_0, \ldots, x_n \in [a, b]$ had the order $n+1$. This means that polynomials with degree n are exactly integrated. For instance, the box rule integrates constant polynomials exactly (degree $n = 0$). The trapezoidal rule integrates linear polynomials exactly (degree $n = 1$).

As intermediate summary, we note: so far the order of a quadrature rule for $n + 1$ support points is $n + 1$.

An optimal distribution (Gauss quadrature) yields however the much better result:

Proposition 13.5. *An interpolatory quadrature rule with $n + 1$ support points has at most the order $2n + 2$.*

In practice the key question is how such quadrature rules can be constructed. This can be achieved via orthogonalization. More information about the construction can be found in standard numerical analysis books.

Chapter 14

Trigonometric interpolation, Fourier series and Fourier transform

14.1 Trigonometric interpolation: discrete Fourier transform

We address here a similar problem as in section 13.1, but instead of classical algebraic polynomials we consider trigonometric polynomials. Such polynomials are better suited to the approximation of periodic functions, and more generally oscillatory phenomena.

Let a function $f : [0, 2\pi] \to \mathbb{C}$ such that $f(0) = f(2\pi)$ be given. A trigonometric polynomial is a function of form

$$P(x) = \sum_{k=-M}^{M} c_k e^{ikx},$$

for some $M \in \mathbb{N}$ and coefficients $c_{-M}, ..., c_M \in \mathbb{C}$. Note that P is by construction continuous and 2π-periodic on $\mathbb{R}$, this is why we have assumed $f(0) = f(2\pi)$. Obviously, the period 2π is chosen for convenience and can be changed by rescaling.

We now define the equidistant nodes $x_0, ..., x_n \in [0, 2\pi[$ by

$$x_j = \frac{2j\pi}{n+1}.$$

Our goal is to determine P in order to interpolate f at these nodes, i.e., to satisfy

$$P(x_j) = f(x_j) \qquad \forall j = 0, ..., n. \tag{14.1}$$

This yields $n+1$ equations, and since there are $2M+1$ coefficients to fix, it is natural to assume that

$$n = 2M.$$

If n is odd, it is possible to choose $M = [n/2] + 1$ ($[x]$ stands for the greatest integer $\leq x$) and impose an additional condition on the coefficients, see e.g. [23]. For simplicity, we assume that n is even and fix $M = n/2$.

We have the following result.

Proposition 14.1. *The relations* (14.1) *are satisfied if and only if*

$$c_k = \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) e^{-ikx_j}, \qquad k = -M, ..., M. \tag{14.2}$$

Proof. Suppose that (14.1) holds true. Multiplying (14.1) by e^{-imx_j} and summing over j yields

$$\sum_{j=0}^{n} \sum_{k=-M}^{M} c_k e^{i(k-m)x_j} = \sum_{j=0}^{n} e^{-mx_j} f(x_j).$$

We can interchange the summations to obtain

$$\sum_{k=-M}^{M} c_k \sum_{j=0}^{n} e^{i(k-m)x_j} = \sum_{j=0}^{n} e^{-mx_j} f(x_j). \tag{14.3}$$

Now, we remark that

$$\sum_{j=0}^{n} e^{i(k-m)x_j} = \sum_{j=0}^{n} e^{i(k-m)j\frac{2\pi}{n+1}} = \sum_{j=0}^{n} \left(e^{i(k-m)\frac{2\pi}{n+1}}\right)^j .$$

It is a geometric sum, equal to

$$\begin{aligned} &n+1 \qquad \text{if } k=m, \\ &\frac{1-\left(e^{i(k-m)\frac{2\pi}{n+1}}\right)^{n+1}}{1-e^{i(k-m)\frac{2\pi}{n+1}}} = \frac{1-e^{i(k-m)2\pi}}{1-e^{i(k-m)\frac{2\pi}{n+1}}} = 0 \qquad \text{if } k \neq m. \end{aligned}$$

We infer from (14.3) the claimed expression

$$(n+1)c_m = \sum_{j=0}^{n} e^{-mx_j} f(x_j).$$

Suppose now (14.2). We compute

$$P(x_l) = \sum_{k=-M}^{M} c_k e^{ikx_l} = \sum_{k=-M}^{M} \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) e^{-ikx_j} e^{ikx_l} = \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) \sum_{k=-M}^{M} e^{ik(x_l-x_j)}.$$

By the same arguments as previously the inner sum is equal to $2M+1 = n+1$ if $j=l$, 0 else. This entails (14.1). □

The family of coefficients $(c_{-M}, ..., c_M)$ is called the discrete Fourier transform of f relatively to the nodes $x_0, ..., x_n$, and the resulting trigonometric polynomial P is the interpolating trigonometric polynomial of f.

There exist fast algorithms to compute these coefficients. They are termed fast Fourier transform (in short FFT) algorithms. One of them (the Cooley-Tukey algorithm) can be studied as project.

Remark 14.1. *If f is real-valued, we infer from* (14.2) *that $c_{-k} = \overline{c_k}$. The Euler formula entails*

$$P(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \left[a_k \cos(kx) + b_k \sin(kx)\right],$$

with the real coefficients

$$a_k = c_k + c_{-k} = 2\Re(c_k) = \frac{2}{n+1} \sum_{j=0}^{n} f(x_j) \cos(kx_j),$$

$$b_k = i(c_k - c_{-k}) = -2\Im(c_k) = \frac{2}{n+1} \sum_{j=0}^{n} f(x_j) \sin(kx_j).$$

An example of trigonometric interpolation with $n=4$, $M=2$, is shown in Fig. 14.1.

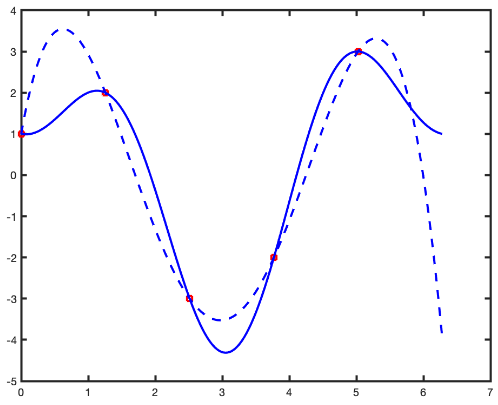


Figure 14.1: Trigonometric interpolation (solid line) vs polynomial interpolation (dashed line). The data points are the red squares. Observe that the trigonometric interpolation extends to a periodic smooth ($\mathcal{C}^\infty$) function, unlike the polynomial interpolation.

14.2 Fourier series

In the above considerations, assume that M (thus also n) goes to infinity. Formally it appears that

$$P(x) \to \sum_{k=-\infty}^{+\infty} c_k e^{ikx}$$

and, recognizing the leftpoint quadrature formula with stepsize $x_{j+1} - x_j = 2\pi/(n+1)$,

$$c_k \to \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx.$$

These ideas admit a strong mathematical background: the Fourier series theory.

14.2.1 Series

Definition 14.1. *A series is an expression of form*

$$S = \sum_{k=0}^{+\infty} a_k, \qquad \textit{meaning exactly} \qquad S = \lim_{n\to+\infty} \sum_{k=0}^{n} a_k.$$

By extension,

$$S = \sum_{k=-\infty}^{+\infty} a_k, \qquad \text{means} \qquad S = \lim_{n\to+\infty} \sum_{k=-n}^{n} a_k.$$

For series of functions, several types of convergence can be considered. Here we will only speak of pointwise convergence, i.e., convergence of the series of numbers when the function is evaluated at a fixed (but arbitrary) point.

14.2.2 Convergence of Fourier series

For the pointwise convergence of Fourier series one has the following result, known as Dirichlet theorem.

Theorem 14.2. *Let $f : \mathbb{R} \to \mathbb{C}$ be a continuous and 2π-periodic function. Let $x \in \mathbb{R}$ such that f admits a left and right derivative at x. We have*

$$f(x) = \sum_{k=-\infty}^{+\infty} c_k e^{ikx} \tag{14.4}$$

with

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}dx. \tag{14.5}$$

The expression (14.4) is called Fourier series of f and the c_k's from (14.5) are called Fourier coefficients of f.

Remark 14.2. *Similarly to remark 14.1, if f is real-valued, we have $c_{-k} = \overline{c_k}$, whereby*

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{+\infty}\left[a_k\cos(kx) + b_k\sin(kx)\right],$$

with

$$a_k = 2\Re(c_k) = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(kx)dx, \qquad b_k = -2\Im(c_k) = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(kx)dx.$$

Remark 14.3. *If f is only piecewise continuous then we have at a discontinuity point x_0*

$$\sum_{k=-\infty}^{+\infty} c_k e^{ikx_0} = \frac{1}{2}\left(\lim_{x\to x_0^-} f(x) + \lim_{x\to x_0^+} f(x)\right).$$

14.2.3 Parseval's equality

Parseval's equality states that the Fourier coefficients carry the 'energy' of the function.

Theorem 14.3. *Let $f : \mathbb{R} \to \mathbb{C}$ be a continuous and 2π-periodic function (square integrable on $[0, 2\pi]$ is actually sufficient) and call c_k its Fourier coefficients. We have*

$$\frac{1}{2\pi}\int_0^{2\pi} |f(x)|^2 dx = \sum_{k=-\infty}^{+\infty} |c_k|^2.$$

14.2.4 Applications

Fourier series are a very important tool in mathematical analysis and signal processing. The main reason of their usefulness is that the basis functions

$$\varphi_k(x) = e^{ikx}$$

are 'eigenfunctions' for the differentiation operator: indeed it holds $\varphi_k' = ik\varphi_k$. Therefore they appear as special solutions of many differential equations. They are also orthogonal, in the sense that

$$\int_0^{2\pi} \varphi_k(x)\varphi_l(x)dx = 0 \qquad \forall k \neq l.$$

It is thus natural to decompose a given function on this 'basis' (to be rigorous the notion of Hilbert basis must be considered, allowing for infinite linear combinations).

14.3 Fourier transform

14.3.1 General concept

Fourier series admit a generalization to non-periodic functions defined over the whole real line: the Fourier transform. In this case the frequencies are no longer discrete, and the series becomes an integral. When the integrals below are well defined [1] one has

$$\boxed{\hat{f}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} f(x)e^{-i\omega x}dx, \qquad f(x) = \int_{-\infty}^{+\infty} \hat{f}(\omega)e^{i\omega x}d\omega.}$$

[1]Basically one needs that $\int_{-\infty}^{+\infty}|f(x)|dx < +\infty$ and $\int_{-\infty}^{+\infty}|\hat{f}(\omega)|d\omega < +\infty$, however, using generalized integrals, $\int_{-\infty}^{+\infty}|f(x)|^2dx < +\infty$ works also.

We call $\hat{f}$ the Fourier transform of f, and the latter equality is known as the Fourier inversion formula. In some cases the above expressions do not have a classical meaning and can be understood in the framework of distribution theory. The Fourier inversion formula holds true for what we call 'tempered distributions'. This is a fairly involved theory that we will not develop. For practical use it is enough to handle the prototype tempered distribution: the so-called Dirac distribution. It is described in the last paragraph.

Let us go back to classical functions and state two properties.

- If f is square integrable over $\mathbb{R}$, i.e. $\int_{-\infty}^{+\infty} |f(x)|^2 dx < +\infty$, then Parseval's equality holds:

$$\frac{1}{2\pi}\int_{-\infty}^{+\infty} |f(x)|^2 dx = \int_{-\infty}^{+\infty} |\hat{f}(\omega)|^2 d\omega.$$

- The Fourier transform of a derivative is formally obtained through integration by parts: if f is differentiable and f, f' admit Fourier transforms, then

$$\widehat{f'}(\omega) = i\omega \hat{f}(\omega).$$

  This property is also recovered by differentiating the inversion formula. Due to this, as for Fourier series, the Fourier transform is very useful to construct special solutions to differential equations.

Remark 14.4. *There is a noticeable symmetry between the definition of the Fourier transform and the inversion formula. Actually, passing to the conjugate in this latter yields*

$$\overline{f(x)} = \int_{-\infty}^{+\infty} \overline{\hat{f}(\omega)} e^{-i\omega x} d\omega = 2\pi \widehat{\overline{\hat{f}}}(x). \tag{14.6}$$

In particular, if f is real-valued as well as its Fourier transform, then $f = 2\pi\hat{\hat{f}}$. Note that placing the normalization coefficient $1/2\pi$ in the definition of the Fourier transform rather than in the inversion formula is a matter of (non-universal) convention. Sometimes, $1/\sqrt{2\pi}$ is placed in front of both.

14.3.2 A glimpse on the Dirac distribution

As mentioned above, it is sometimes convenient to apply the Fourier transform to distributions rather than functions. We will not give a rigorous definition of the concept of distribution, we will content ourselves with the formal description of the most classical one: the Dirac distribution. Intuitively, the Dirac distribution δ is a generalization of a function that is equal to 0 everywhere except at point 0, and with 'integral' equal to 1. More precisely, it satisfies

$$'' \int_{-\infty}^{+\infty} \delta(x)\varphi(x)dx'' = \varphi(0)$$

for all smooth function φ. Indeed, choosing this (so-called test function) φ identically equal to 1 yields $\int_{-\infty}^{+\infty} \delta(x)dx = 1$, while choosing φ arbitrary but vanishing at 0 yields $\delta(x) = 0$ for all $x \neq 0$. An intuitive understanding of this concept is to see the Dirac distribution as the limit of a sequence of nonnegative functions of integral 1 that concentrate at 0, see Fig. 14.2.

We obtain from the definition and the inversion formula in the form (14.6)

$$\hat{\delta} = \frac{1}{2\pi}, \qquad \hat{1} = \delta.$$

One may remark that the function 1 is neither integrable nor square integrable over $\mathbb{R}$ (in fact, it is another example of tempered distribution). Its Fourier transform has been determined so as to satisfy the inversion formula, which is the main goal of the construction.

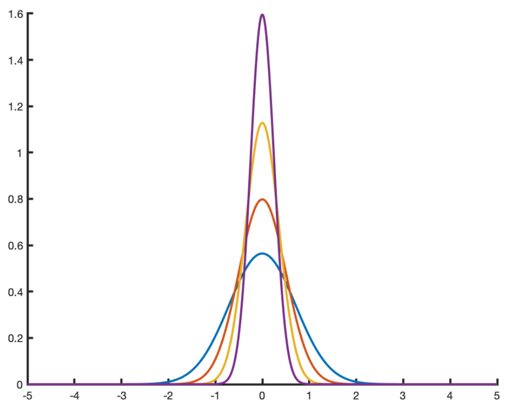


Figure 14.2: Illustration of the Dirac distribution as the limit of a sequence of functions that become thiner and thiner, and higher and higher. Here a Gaussian function with standard deviation going to 0 is considered.

Chapter 15

Exercises

Functions of one or several variables

Exercise 15.1. *Calculate the gradient (Jacobian matrix), the divergence, the rotational and the Laplacian of the following vector fields:*

$$f(x,y,z) = \begin{pmatrix} 3x^2+y+z \\ x^2-y^2-z \\ x+z^2 \end{pmatrix}, \qquad g(x,y,z) = \begin{pmatrix} y^2+z^2 \\ x^2+z^2 \\ x^2+y^2 \end{pmatrix}.$$

Exercise 15.2. *For the following scalar fields calculate the gradient, the Hessian matrix, the critical points and their nature (local minimum/maximum...):*

$$f(x,y) = 3x^2+2y^2+2xy+1,$$

$$g(x,y) = x^2-2y^2+xy-9x.$$

Exercise 15.3 (Integration)**.** *Evaluate*

$$\int_0^1 exp(x)\cdot x\,dx.$$

Evaluate

$$\int_0^2 2xexp(x^2)\,dx.$$

Hint: Use integration by parts and substitution rules.

Exercise 15.4 (Taylor)**.** *Write down the Taylor polynomials of degree* 4 *in* $x_0=0$ *for*

1. $f(x)=\sin(x)$,

and for $x_0=1$ *for*

2. $f(x)=\exp(2x)$.

Exercise 15.5. *Let us define*

$$f(x,y) = \begin{cases} \frac{xy}{x^2+y^2} & (x,y)\neq(0,0) \\ 0 & (x,y)=(0,0). \end{cases}$$

Is this function continuous or discontinuous in $(0,0)$*. Justify your answer.*

Nonlinear equations

Exercise 15.6. *Find the fixed points of the following functions and give their nature (attractive, repulsive). Give a graphical illustration of the convergence (or not) of a sequence of type $x_{n+1} = f(x_n)$.*

1. $f : x \in \mathbb{R}_+^* \mapsto f(x) = \sqrt{x}$,
2. $f : x \in \mathbb{R} \mapsto f(x) = 1 - x^2$.

Exercise 15.7 (complement)**.** *Describe the sequence generated by Newton's method for solving* $\log x = 0$*, with initial point* $x_0 > 0$*. Show that it converges if and only if* $x_0 \in]0, e[$.

Exercise 15.8. *Let the function $f(x, y) = y^2(x-1) + x^2(x+1)$ be given. Compute the stationary points and check which kind of extremum point we deal with.*

Exercise 15.9. *Let $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = \sqrt{x^2+1}$.*

1. *Compute the minimum of $f(x)$.*
2. *That is to solve $f'(x) = 0$.*
3. *Write down Newton's method.*
4. *Perform some steps (per hand) for different initial guesses $x_0 = 0.5$ and $x_0 = 2$ respectively.*
5. *(optional) Implement Newton's method and compute the solution $\hat{x}$ to a tolerance $TOL = 1\mathrm{e}{-10}$.*

Exercise 15.10. *Let $L : \mathbb{R}^3 \to \mathbb{R}$ with*

$$L(x, y, \lambda) = f(x) - \lambda c(x) = -x - 0.5y^2 - \lambda(1 - x^2 - y^2)$$

be given.

1. *Find the stationary point of L. What needs to be done?*
2. *Formulate Newton's method (formally; without implementation) for the previous minimization problem.*

Exercise 15.11 (This exercise is half of a project because implementation is required)**.** *Develop a Newton scheme in $\mathbb{R}^2$ to find the root of the problem:*

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \quad f(x, y) = \Big(2xay^2, 2(x^2+\kappa)ay\Big)^T,$$

where $\kappa = 0.01$ and $a = 5$.

1. *Justify first that integration of f yields $F(x, y) = (x^2 + \kappa)ay^2$. What is the relation between f and F?*
2. *Compute the root of f by hand. Derive the derivative f' and study its properties.*
3. *Finally, design the requested Newton algorithm. As initial guess, take $(x_0, y_0) = (4, -5)$.*
4. *What do you observe with respect to the number of Newton iterations?*
5. *How could we reduce the number of Newton steps?*

Polynomial interpolation

Exercise 15.12. *Consider the points $x_n = n$ for all $n \in \mathbb{N}$. Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function such that $f(x_n) = \frac{1}{2}(1 + (-1)^{n+1})$ for all $n \in \mathbb{N}$. Calculate the Lagrange interpolating polynomial $P \in \mathcal{P}_4$ of f at the points x_0, x_1, x_2, x_3, x_4.*

Exercise 15.13. *Let the following data set be given:*

i	0	1	2
x_i	-1	0	2
y_i	-1	0	1

Construct the piece-wise linear and quadratic interpolation polynomials, respectively.

Exercise 15.14 (complement)**.** *Let $T > 0$. We consider the function defined by $f(x) = \sin(\frac{2\pi}{T}x)$ for all $x \in [0, 1]$.*
Let $n \in \mathbb{N}^\star$. We denote by $P_n \in \mathcal{P}_n$ the Lagrange interpolating polynomial of f at the $n + 1$ points $0, 1/n, 2/n \dots, 1$.

1. *Determine an upper bound of $|f^{(n+1)}(x)|$. Deduce an upper bound of the error $|f(x) - P_n(x)|$ for all $x \in [0, 1]$.*
2. *Let $\epsilon > 0$ and $T > 0$ be given. Provide a condition on n ensuring that the error is less than ϵ for all $x \in [0, 1]$. Make explicit this condition for $T = 2\pi$ and $\epsilon = 0.1$.*

Numerical integration (complement)

Exercise 15.15. *We place ourselves on the reference interval $[-1, 1]$ and we consider the elementary quadrature formula*

$$J(g) = w_0 g(-1) + w_1 g(0) + w_2 g(1).$$

Fix the weights w_0, w_1, w_2 so that the formula is of maximal order (this is the Simpson formula). What is this order?

Exercise 15.16. *Evaluate the integral*

$$\int_0^1 \frac{1}{1+x}\,dx$$

with the mid-point rule, trapezoidal rule and the Simpson rule, respectively.

Trigonometric interpolation, Fourier series, Fourier transform

Exercise 15.17. *Let the function $f : [0, 2\pi] \to \mathbb{R}$ be defined by $f(x) = \sin(2x)$. For $M \in \mathbb{N}$, $n = 2M$, and*

$$x_j = \frac{2j\pi}{n+1} \qquad j = 0, ..., n,$$

let P_M be the interpolating trigonometric polynomial of f associated with these nodes.
1. Calculate P_0 and P_1. Without calculation, determine P_M for any $M \geq 2$.
2. For $f(x) = \sin(mx)$, indicate a relation between the pulsation m and the number of nodes needed to guaranty an exact representation of f by its trigonometric interpolant. This a simplified version of Shannon's theorem.

Exercise 15.18. *Let the function $f : [0, 2\pi] \to \mathbb{R}$ be defined by*

$$f(0) = f(2\pi) = 1,$$
$$f(x) = 0 \text{ otherwise.}$$

Let $M \in \mathbb{N}$, $n = 2M$, and

$$x_j = \frac{2j\pi}{n+1} \qquad j = 0, ..., n.$$

1. Calculate the discrete Fourier transform of f associated with the above nodes.
2. Show that the interpolating trigonometric polynomial of f admits the expression

$$P_M(x) = \frac{1}{2M+1} \frac{\cos(Mx) - \cos((M+1)x)}{1 - \cos x}.$$

Plot this function for $M = 8$ and $M = 16$. What do you observe concerning the minimal value of P_M?
3. (optional) Calculate $P_M(\frac{3\pi}{2M+1})$ and its limit when $M \to +\infty$. What do you conclude?

Exercise 15.19 (complement). *Let the function $f : [0, 2\pi] \to \mathbb{R}$ be defined by*

$$f(x) = x(2\pi - x).$$

1. Calculate the Fourier coefficients c_k, a_k, b_k of f.
2. Deduce the value of

$$\sum_{k=1}^{+\infty} \frac{1}{k^2}.$$

3. (optional) Using Parseval's equality calculate

$$\sum_{k=1}^{+\infty} \frac{1}{k^4}.$$

Exercise 15.20. *Consider the function $r : \mathbb{R} \to \mathbb{R}$ defined by*

$$r(x) = \begin{cases} 1 \text{ if } -1 \leq x \leq 1 \\ 0 \text{ otherwise.} \end{cases}$$

1. Calculate the Fourier transform $\hat{r}$ of r.
2. (optional) Formulate Parseval's equality.
3. (optional) Show that

$$\int_{-\infty}^{\infty} |\omega \hat{r}(\omega)|^2 = +\infty. \tag{15.1}$$

This shows in a quantitative way that the spectrum of r decays slowly at infinity. How do you qualitatively interpret the high frequencies? Interpret (15.1) using Parseval's equality.
4. Provide the expression of the spatial Fourier transform $\hat{u}(t, \omega)$ of the function $u(t, x)$ solution of the heat equation (see also next chapter for a review on differential equations)

$$\begin{cases} \frac{\partial u}{\partial t}(t, x) - \frac{\partial^2 u}{\partial x^2}(t, x) = 0 \qquad \forall (t, x) \in \mathbb{R}_+ \times \mathbb{R} \\ u(0, x) = r(x) \qquad \forall x \in \mathbb{R}. \end{cases}$$

What about the decay of the spectrum? Give a physical interpretation.
5. Same question when r is replaced by the Dirac distribution δ. Optional: give an expression of the function u itself.

Optimization

Exercise 15.21. *We consider the minimization problem:*

$$\text{minimize}_{u \in \mathbb{R}^n} J(u),$$

where $J : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function.

We will specifically work with the 3 following functions defined on $\mathbb{R}^2$:

- *the quadratic function*

$$J_1(x,y) = (x-1)^2 + p(y-1)^2,$$

- *the Rosenbrock function*

$$J_2(x,y) = (x-1)^2 + p(x^2-y)^2,$$

- *the radial function*

$$J_3(r) = e^{-r} + r, \qquad r = \sqrt{x^2+y^2}.$$

In the first 2 cases the positive parameter p can be chosen equal to 10. Of course those are academic examples for which the solution is obvious.

1. *Define the above functions, for instance in Python:*

```
def J(x,y):
     return (x-1)**2+p*(y-1)**2
```

2. *Plot isovalues, for instance:*

```
import numpy as np
import matplotlib.pyplot as plt
h=0.025
xx=np.arange(-1,1.5,h)
yy=np.arange(-0.5,2,h)
X,Y=np.meshgrid(xx,yy)
Z=J(X,Y)
fig, ax=plt.subplots()
CS=ax.contour(X,Y,Z,20)
ax.clabel(CS,inline=1,fontsize=10)
```

3. *Implement the steepest descent method with the following special case of Armijo's line search:*

```
Set $d^k = -\nabla J(u^{k-1})$, $\alpha_0 = 1$
While $J(u^{k-1} + \alpha_i d^k) > J(u^{k-1}) + \frac{1}{2}\alpha_i \nabla J(u^{k-1}) \cdot d^k$
    $\alpha_{i+1} = \alpha_i/2$
end
Set $s_k = \alpha_i$
```

The stopping criterion of the main loop will be the satisfaction of one of the conditions:

- *an optimality criterion: $\|\nabla J(u^k)\| \le \varepsilon$ where $\varepsilon = 10^{-7}\|\nabla J(u^0)\|$,*
- *a maximal number of iterations maxiter = 5000, in order to stop the algorithm if it does not converge.*

Display the obtained minimizers and plot the trajectories.

4. *Play with the initial guess and the algorithmic parameters (such as the stopping criterion and parameters within Armijo's rule). What are your conclusions?*

Part IV

Differential equations

Chapter 16

Introduction to ODEs (ordinary differential equations)

This chapter is on purpose short. We introduce a few examples of ODEs and some analytical properties. For more detailed introductions, we refer the reader to [5] or [10].

16.1 An introductory example

Let us compute the growth of a species (for example human beings) with a very simple (and finally not that realistic) model. But this shows that reality can be represented to some extent at least by simple models, and that continuous comparisons with other data is also necessary. Furthermore, this also shows that mathematical modeling often starts with a simple equation and is then continuously enriched with further terms and coefficients. In the end we arrive at complicated formulae.

To get started, let us assume that the population number is $y = y(t)$ at time t. Furthermore, we assume constant growth g and mortalities rates m, respectively. In a short time frame dt we have a relative increase of

$$\frac{dy}{y} = (g - m)dt$$

of the population. Re-arranging and taking the limit $dt \to 0$ yields:

$$\lim_{dt\to 0} \frac{dy}{dt} = (g - m)y$$

and thus

$$y' = (g - m)y.$$

This is the so-called **Malthusian law of growth**. For this ordinary differential equation (ODE), we can even explicitly compute the solution:

$$y(t) = c\exp((g - m)(t - t_0)).$$

This can be formally obtained through:

$$y' = \frac{dy}{dt} = (g - m)y \tag{16.1}$$

$$\Rightarrow \int \frac{dy}{y} = \int_{t_0}^{t} (g - m)dt \tag{16.2}$$

$$\Rightarrow \log|y| + C = (g - m)(t - t_0) \tag{16.3}$$

$$\Rightarrow y = \exp[C] \cdot \exp[(g - m)(t - t_0)]. \tag{16.4}$$

In order to work with this ODE and to compute the future development of the species we need an initial value at some starting point t_0:

$$y(t_0) = y_0.$$

With this value, we can further work to determine the constant $c = \exp(C)$:

$$y(t_0) = c\exp[(g-m)(t_0 - t_0)] = c = y_0. \tag{16.5}$$

Let us say in the year $t_0 = 2021$ there have been two members of this species: $y(2021) = 2$. Supposing a growth rate of 25 per cent per year yields $g = 0.25$. Let us say $m = 0$ - nobody will die. In the following we compute two estimates of the future evolution of this species: for the year $t = 2024$ and $t = 2032$. We first obtain:

$$y(2024) = 2\exp(0.25 * (2024 - 2021)) = 4.117 \approx 4.$$

Thus, four members of this species exist after three years. Secondly, we want to give a 'long term' estimate for the year $t = 2022$ and calculate:

$$y(2022) = 2\exp(0.25 * (2032 - 2021)) = 31.285 \approx 31.$$

In fact, this species has an increase of 29 members within 11 years. If you translate this to human beings, we observe that the formula works quite well for a short time range but becomes somewhat unrealistic for long-term estimates though.

Remark 16.1. *An improvement of the basic population model is the* ***logistic differential equation*** *that was proposed by Verhulst; see e.g., [5].*

16.2 The model ODE

In the previous example (setting $a = g - m$), we introduced one of the most important ODEs:

$$y' = ay, \quad y(t_0) = y_0. \tag{16.6}$$

This ODE serves often as model problem and important concepts such as existence, uniqueness, stability are usually introduced in terms of this ODE. It is linear with a possibly variable coefficient a. The unique solution can be expressed as

$$y(t) = y_0 \exp(\int_{t_0}^t a(s)ds). \tag{16.7}$$

For a non-homogeneous ODE of form $y' = ay + b$, the general expression of the solution becomes

$$y(t) = \bar{y}(t) + \alpha\exp(\int_{t_0}^t a(s)ds),$$

where $\bar{y}(t)$ is any particular solution and α is a constant. The classical way to find a particular solution is the method of 'variation of the constant', which means searching a solution in the form

$$\bar{y}(t) = \alpha(t)\exp(\int_{t_0}^t a(s)ds).$$

Then one finds $\bar{y}'(t) = a(t)\bar{y}(t) + \alpha'(t)\exp(\int_{t_0}^t a(s)ds)$, whereby α can be obtained through $\alpha'(t) = b(t)\exp(-\int_{t_0}^t a(s)ds)$.

For nonlinear ODEs of form $y' = f(t, y)$ the existence of solutions is much less trivial and often relies on the Cauchy-Lipschitz theorem. We will not enter into these notions. There are also higher order differential equations, which involve second or higher order derivatives of the unknown. Introducing some of these derivatives as additional unknowns, they can be reduced to a system of first order differential equations. Therefore we will subsequently mostly focus our attention to first order ODEs. We only give below a short reminder on exact solutions for second order linear ODE's with constant coefficients.

16.3 Second order linear ODE's with constant coefficients

We consider the differential equation

$$y'' + ay' + by = 0, \tag{16.8}$$

where a, b are fixed real numbers. We call $\mathcal{S}$ the set of solutions.

Definition 16.1. *The algebraic equation*

$$r^2 + ar + b = 0 \tag{16.9}$$

is called characteristic equation of (16.8).

Theorem 16.1. 1. *If the characteristic equation* (16.9) *admits two distinct real solutions α and β, then*

$$\mathcal{S} = \left\{ t \mapsto \lambda e^{\alpha t} + \mu e^{\beta t},\ \lambda, \mu \in \mathbb{R} \right\}.$$

2. *If the characteristic equation* (16.9) *admits a double root α then*

$$\mathcal{S} = \left\{ t \mapsto (\lambda t + \mu) e^{\alpha t},\ \lambda, \mu \in \mathbb{R} \right\}.$$

3. *If the characteristic equation* (16.9) *admits two distinct conjugate complex roots $\rho + i\omega$ and $\rho - i\omega$ then*

$$\mathcal{S} = \left\{ t \mapsto e^{\rho t}[\lambda \cos(\omega t) + \mu \sin(\omega t)],\ \lambda, \mu \in \mathbb{R} \right\}.$$

Examples.
If $a = 0$ and $b = \omega^2 > 0$ then $\mathcal{S} = \{t \mapsto \lambda \cos(\omega t) + \mu \sin(\omega t),\ \lambda, \mu \in \mathbb{R}\}$.
If $a = 0$ and $b = -\omega^2 < 0$ then $\mathcal{S} = \left\{ t \mapsto \lambda e^{\omega t} + \mu e^{-\omega t},\ \lambda, \mu \in \mathbb{R} \right\}$.

Chapter 17

Finite differences for ODE initial-value problems

Finite differences have been very popular in the past and are still used to solve initial-value problems (IVP) and boundary value problems (BVP). Often finite differences are used in combination with other discretization methods as for example finite elements. For example in a time-dependent PDE, we need to discretize in time and space. Using the Rothe method (horizontal method of lines) temporal discretization is often based on finite differences and spatial discretization on finite elements. In these notes we restrict ourselves to single-step methods.

17.1 Problem statement of an IVP (initial value problem)

We consider ODE initial value problems of the form: find a differentiable function $y(t)$ for $0 \le t < T < \infty$ such that

$$\begin{aligned} y'(t) &= f(t, y(t)), \\ y(0) &= y_0. \end{aligned}$$

The second condition is the so-called initial condition. Furthermore, $y'(t) = \frac{d}{dt} y$ is the time derivative with respect to time.

The model problem of an ODE is

$$y' = \lambda y, \quad y(0) = y_0, \quad y_0, \lambda \in \mathbb{R}. \tag{17.1}$$

This ODE has the solution:

$$y(t) = \exp(\lambda t) y_0.$$

Furthermore, Problem (17.1) is an example of an autonomous and linear ODE. It is autonomous because the right hand side does not explicitly depend on the variable t, i.e., $f(t, y) = f(y) = \lambda y$. And this ODE is linear because the coefficient λ is independent of the solution y. One further classification can be made in case the right hand side $f = 0$. Then we say that the ODE is homogeneous. Lastly, the coefficient λ is here a constant: we say that it is a linear ODE with constant coefficient. It is a particular case of the ODE with variable coefficient briefly discussed in (16.6), (16.7).

17.2 Stiff problems

An essential difficulty in developing stable numerical schemes, is associated with **stiffness**, which we shall define in the following. Stiffness is very important in both ODE and (time-dependent) PDE applications.

Stiff problems are characterized as ODE solutions that contain various components with (significant) different evolutions over time. A very illustrative example can be found in [27] in which for instance a solution to an ODE problem may look like:

$$y(t) = \exp(-t) + \exp(-99t).$$

Here, the first term defines the trajectory of the solution whereas the second term requires very small time steps in order to resolve the very rapid decrease of that function.

Such behaviors specifically appear in ODE systems in which we seek $y(t) \in \mathbb{R}^n$ rather than $y(t) \in \mathbb{R}$. Then the following definition can be adopted. An IVP is called **stiff** (along a solution $y(t)$) if the eigenvalues $\lambda(t)$ of the Jacobian $f'(t, y(t))$ of f w.r.t. y yield the stiffness ratio:

$$\kappa(t) := \frac{\max_{\Re\lambda(t)<0} |\Re\lambda(t)|}{\min_{\Re\lambda(t)<0} |\Re\lambda(t)|} \gg 1.$$

17.3 Well-posedness

In ODE lectures, we usually learn first the theorem of Peano, which ensures existence (but not uniqueness) of an ODE solution. In these notes, we consider directly the Cauchy-Lipschitz (sometimes called Picard-Lindelöf) theorem, which establishes existence and uniqueness.

Consider the n-dimensional case:

$$y'(t) = f(t, y(t))$$

where $y(t) = (y_1, \ldots, y_n)^T$ and $f(t, x) = (f_1, \ldots, f_n)^T$. Let an initial point $(t_0, y_0) \in \mathbb{R} \times \mathbb{R}^n$ be given. Furthermore, define

$$D = I \times \Omega \subset \mathbb{R}^1 \times \mathbb{R}^n$$

and let $f(t, y)$ be continuous on D. We seek a solution $y(t)$ on a time interval $I = [t_0, t_0 + T]$ that satisfies $y(t_0) = y_0$.

A crucial aspect for uniqueness is a Lipschitz condition on the right hand side $f(t, y)$: The function $f(t, y)$ on D is said to be (uniformly) Lipschitz continuous if for $L(t) > 0$ it holds

$$\|f(t, x_1) - f(t, x_2)\| \leq L(t)\|x_1 - x_2\|, \quad (t, x_1), (t, x_2) \in D.$$

The function is said to be (locally) Lipschitz continuous if the previous statement holds on every bounded subset of D.

Theorem 17.1 (Cauchy-Lipschitz)**.** *Let $f : D \to \mathbb{R}^n$ be Lipschitz-continuous. Then there exists for each $(t_0, y_0) \in D$ an $\varepsilon > 0$ and a solution $y : I := [t_0 - \varepsilon, t_0 + \varepsilon] \to \mathbb{R}^n$ of the IVP such that*

$$y'(t) = f(t, y(t)), \quad t \in I, \quad y(t_0) = y_0.$$

17.4 One-step schemes

17.4.1 The Euler method

The Euler method is the simplest scheme. It is an explicit scheme and also known as forward Euler method.

We consider again the IVP from above with right hand side that satisfies again a Lipschitz condition. According to Theorem 17.1, there exists a unique solution on some neighborhood of t_0. Here we suppose in addition the existence of a global solution on $[t_0, t_0 + T]$.

For a numerical approximation of the IVP, we first select a sequence of discrete (time) points:

$$t_0 < t_1 < \ldots < t_N = t_0 + T.$$

Furthermore we set

$$I_n = [t_{n-1}, t_n], \quad k_n = t_n - t_{n-1}, \quad k := \max_{1\le n\le N} k_n.$$

The derivation of the Euler method is as follows: approximate the derivative with a forward difference quotient (we sit at the time point t_{n-1} and look forward in time):

$$y'(t_{n-1}) \approx \frac{y(t_n) - y(t_{n-1})}{k_n}.$$

Yet $y'(t_{n-1}) = f(t_{n-1}, y_{n-1}(t_{n-1}))$. Then the ODE can be approximated as:

$$\frac{y(t_n) - y(t_{n-1})}{k_n} \approx f(t_{n-1}, y_{n-1}(t_{n-1})).$$

Setting the approximation $y_n \approx y(t_n)$ we obtain the scheme:

Definition 17.1 (Euler method). *For a given starting point $y_0 := y(0) \in \mathbb{R}^n$, the Euler method generates a sequence $\{y_n\}_{n\in\mathbb{N}}$ through*

$$y_n = y_{n-1} + k_n f(t_{n-1}, y_{n-1}), \quad n = 1, \dots N.$$

17.4.2 Implicit schemes

With the same notation as introduced in Section 17.4.1, we define two further schemes. Beside the Euler method, low-order simple schemes are the implicit Euler and the trapezoidal rule. The main difference is that in general a nonlinear equation system needs to be solved in order to compute the solution. On the other hand we have better numerical stability properties in particular for stiff problems (an analysis will be undertaken in Section 17.5).

The derivation of the implicit (or backward) Euler method is derived as follows: approximate the derivative with a backward difference quotient (we sit at t_n and look back to t_{n-1}):

$$y'(t_n) \approx \frac{y(t_n) - y(t_{n-1})}{k_n}.$$

Consequently, we take the right hand side f at the current time step $y'(t_n) = f(t_n, y_n(t_n))$ and obtain as approximation

$$\frac{y(t_n) - y(t_{n-1})}{k_n} = f(t_n, y_n(t_n)).$$

Consequently, we obtain a scheme in which the right hand side is unknown itself, which leads to a formulation of a nonlinear system:

Definition 17.2 (Implicit Euler). *The implicit Euler scheme is defined as:*

$$\begin{gathered} y_0 := y(0), \\ y_n - k_n f(t_n, y_n) = y_{n-1}, \quad n = 1, \dots, N. \end{gathered}$$

In contrast to the Euler method, the 'right hand side' function f now depends on the unknown solution y_n. Thus the computational cost is (much) higher than for the (forward) Euler method. But on the contrary, the method does not require a time step restriction as we shall see in Section 17.5.1.

To derive the trapezoidal rule (or Crank-Nicolson scheme), we take again the difference quotient on the left hand side but approximate the right hand side through its mean value:

$$\frac{y_n - y_{n-1}}{k_n} = \frac{1}{2}\Big(f(t_n, y_n(t_n)) + f(t_{n-1}, y_{n-1}(t_{n-1}))\Big),$$

which yields:

Definition 17.3 (Trapezoidal rule). *The trapezoidal rule reads:*

$$\begin{aligned} y_0 &:= y(0), \\ y_n &= y_{n-1} + \frac{1}{2}k_n\Big(f(t_n, y_n) + f(t_{n-1}, y_{n-1})\Big), \quad n = 1, \ldots, N. \end{aligned}$$

It can be shown that the trapezoidal rule is of second order, which means that halving the step size k_n leads to an error that is four times smaller. This is illustrated with the help of a numerical example in Section 17.6.

17.5 Numerical analysis

In the previous section, we have constructed algorithms that yield a sequence of discrete solution $\{y_n\}_{n\in\mathbb{N}}$. In the numerical analysis our goal is to derive a convergence result of the form

$$\|y_n - y(t_n)\| \leq Ck^\alpha$$

where α is the order of the scheme. This result will tell us that the discrete solution y^n really approximates the exact solution y and if we come closer to the exact solution at which rate we come closer.

The model problem

To derive error estimates we work with the model scalar problem:

$$y' = \lambda y, \quad y(0) = y_0, \quad y_0, \lambda \in \mathbb{R}. \tag{17.2}$$

Splitting into stability and consistency

For linear numerical schemes the convergence is composed by **stability** and **consistency**. First of all we have from the previous section that y_n is obtained for the forward Euler method as:

$$\begin{aligned} y_n &= (1 + k\lambda)y_{n-1} \\ &= B_E y_{n-1}, \quad B_E := 1 + k\lambda. \end{aligned}$$

Let us write the error at each time point t_n as:

$$e_n := y_n - y(t_n) \quad \text{for } 1 \leq n \leq N.$$

It holds:

$$\begin{aligned} e_n &= y_n - y(t_n) \\ &= B_E y_{n-1} - y(t_n) \\ &= B_E(e_{n-1} + y(t_{n-1})) - y(t_n) \\ &= B_E e_{n-1} + B_E y(t_{n-1}) - y(t_n) \\ &= B_E e_{n-1} + \frac{k(B_E y(t_{n-1}) - y(t_n))}{k} \\ &= B_E e_{n-1} - k\underbrace{\frac{y(t_n) - B_E y(t_{n-1})}{k}}_{=:\eta_{n-1}}. \end{aligned}$$

Therefore, the error can be split into two parts:

Definition 17.4 (Error splitting of the model problem). *The error at step n can be decomposed as*

$$e_n := \underbrace{B_E e_{n-1}}_{\textit{Stability}} - \underbrace{k\eta_{n-1}}_{\textit{Consistency}}. \tag{17.3}$$

The first term, namely the stability, provides an idea how the previous error e_{n-1} is propagated from t_{n-1} to t_n. The second term η_{n-1} is the so-called truncation error (or local discretization error), which arises because the exact solution does not satisfy the numerical scheme and represents the consistency of the numerical scheme. Moreover, η_{n-1} yields the speed of convergence of the numerical scheme.

In fact, for the forward Euler scheme in (17.3), we observe for the truncation error:

$$\begin{aligned}\eta_{n-1} &= \frac{y(t_n) - B_E y(t_{n-1})}{k} = \frac{y(t_n) - (1 + k\lambda)y(t_{n-1})}{k} \\ &= \frac{y(t_n) - y(t_{n-1})}{k} - \lambda y(t_{n-1}) \\ &= \frac{y(t_n) - y(t_{n-1})}{k} - y'(t_{n-1}). \end{aligned} \tag{17.4}$$

Thus,

$$y'(t_{n-1}) = \frac{y(t_n) - y(t_{n-1})}{k} - \eta_{n-1}, \tag{17.5}$$

which is nothing else than the approximation of the first-order derivative with the help of a difference quotient plus the truncation error. We investigate these terms further in Section 17.5.2 and concentrate first on the stability estimates in the very next section.

17.5.1 Stability

The goal of this section is to control the term $B_E = (1 + k\lambda)$. Specifically, we will justify why $|B_E| \leq 1$ should hold. The stability is often related to non-physical oscillations of the numerical solution. Otherwise speaking, a nonstable scheme shows artificial oscillations as for example illustrated in Figure 17.2.

We introduce (absolute) **stability** and **A-stability**. From the model problem

$$y'(t) = \lambda y(t), \quad y(t_0) = y_0,\ \lambda \in \mathbb{C},$$

we know the solution $y(t) = y_0 \exp(\lambda t)$. For $t \to \infty$ the solution is characterized by the sign of $\Re\,\lambda$:

$$\begin{aligned} \Re\,\lambda < 0 \quad &\Rightarrow |y(t)| = |y_0| \exp(t\Re\,\lambda) \to 0, \\ \Re\,\lambda = 0 \quad &\Rightarrow |y(t)| = |y_0| \exp(t\Re\,\lambda) = |y_0|, \\ \Re\,\lambda > 0 \quad &\Rightarrow |y(t)| = |y_0| \exp(t\Re\,\lambda) \to \infty. \end{aligned}$$

In the first two cases it is particularly interesting to know whether a numerical scheme can produce a bounded discrete solution, as the continuous solution is bounded.

Definition 17.5 ((Absolute) stability). *A (one-step) method is absolute stable for $\lambda k \neq 0$ if its application to the model problem produces in the case $\Re\,\lambda \leq 0$ a sequence of bounded discrete solutions:* $\sup_{n\geq 0} |y_n| < \infty$.

To find the stability region, we work with the stability function $R(z)$, $z = \lambda k$, such that $B_E = R(z)$. The region of absolute stability is defined as:

$$SR = \{z \in \mathbb{C} : |R(z)| \leq 1\}.$$

The stability functions to explicit Euler, implicit Euler and trapezoidal rule are given by:

Proposition 17.2. *For the simplest time-stepping schemes forward Euler, backward Euler and the trapezoidal rule, the stability functions $R(z)$ read:*

$$\begin{aligned} R(z) &= 1 + z \qquad \textit{(forward Euler)}, \\ R(z) &= \frac{1}{1 - z} \qquad \textit{(backward Euler)}, \\ R(z) &= \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \qquad \textit{(trapezoidal rule)}. \end{aligned}$$

Let us look at the proof. We take again the model problem $y' = \lambda y$. We recall the discretization of this problem with the forward Euler method:

$$\frac{y_n - y_{n-1}}{k} = \lambda y_{n-1} \tag{17.6}$$
$$\Rightarrow y_n = (y_{n-1} + \lambda k) y_{n-1} \tag{17.7}$$
$$= (1 + \lambda k) y_{n-1} \tag{17.8}$$
$$= (1 + z) y_{n-1} \tag{17.9}$$
$$= R(z) y_{n-1}. \tag{17.10}$$

For the implicit Euler method we obtain:

$$\frac{y_n - y_{n-1}}{k} = \lambda y_n \tag{17.11}$$
$$\Rightarrow y_n = y_{n-1} + \lambda k y_n \tag{17.12}$$
$$\Rightarrow y_n = \frac{1}{1 - \lambda k} y_{n-1} \tag{17.13}$$
$$\Rightarrow y_n = \underbrace{\frac{1}{1 - z}}_{=:R(z)} y_{n-1}. \tag{17.14}$$

The procedure for the trapezoidal rule is again the analogous.

Definition 17.6 (A-stability). *A difference method is A-stable if its stability region is part of the absolute stability region:*

$$\{z \in \mathbb{C} : \Re z \leq 0\} \subset SR.$$

In other words: Let $\{y_n\}_n$ the sequence of solutions of a difference method for solving the ODE model problem. Then, this method is A-stable if for arbitrary $\lambda \in \mathbb{C}^- = \{\lambda : \Re(\lambda) \leq 0\}$ the approximate solutions are bounded for arbitrary, but fixed, step size k. That is to say:

$$|y_{n+1}| \leq |y_n| < \infty \quad \text{for } n = 1, 2, 3, \ldots$$

Remark 17.1. *A-stability is an attractive property since in particular for stiff problems we can compute with arbitrary step sizes k and do not need any step size restriction.*

Proposition 17.3. *The explicit Euler scheme cannot be A-stable.*

Indeed, for the forward Euler scheme, we have $R(z) = 1 + z$. For $|z| \to \infty$ is holds $R(z) \to \infty$ which is a violation of the definition of A-stability.

Remark 17.2. *More generally, explicit schemes can never be A-stable.*

On the contrary:

Proposition 17.4. *The implicit Euler scheme and the trapezoidal rule are A-stable.*

Example 17.1. *We illustrate the previous statements.*

1. *In Proposition 17.2 we have seen that for the forward Euler method it holds:*

$$y_n = R(z) y_{n-1},$$

where $R(z) = 1 + z$. A necessary condition of convergence of the sequence is

$$|R(z)| \leq |1 + z| \leq 1. \tag{17.15}$$

Thus if the absolute value of λ (in $z = \lambda k$) is very big, we must choose a very small time step k in order to achieve $|1 - \lambda k| < 1$. Otherwise the sequence $\{y_n\}_n$ will increase and thus diverge (recall that stability is defined with bounded continuous solutions and consequently the numerical approximation should be bounded, too). In conclusions, the forward Euler scheme is only conditionally stable, i.e., it is stable provided that (17.15) *is fulfilled.*

2. *For the implicit Euler scheme, we see that a large $|\lambda|$ and large k even both help to stabilize the iteration scheme (but be careful, the implicit Euler scheme actually stabilizes sometimes too much. Because it computes contracting sequences also for case where the continuous solution would grow). Thus, no time step restriction is required. Consequently, the implicit Euler scheme is well suited for problems with large parameters/coefficients λ and also for stiff problems.*

Remark 17.3. *The previous example shows that a careful design of the appropriate discretization scheme requires some work: there is no a priori best scheme. Some schemes require time step size restrictions in case of large coefficients (explicit Euler). On the other hand, the implicit Euler scheme does not need step restrictions but may have in certain cases too much damping. Which scheme should be employed for which problem depends finally on the problem itself and must be carefully thought for each problem again.*

17.5.2 Consistency / local discretization error - convergence order

We address now the second 'error source' in (17.3). The consistency determines the precision of the scheme and will finally carry over the local rate of consistency to the global rate of convergence. To determine the consistency we assume sufficient regularity of the exact solution such that we can apply Taylor expansion. The idea is then that discrete schemes contain some dominant terms. The lowest order remainder term determines finally the local consistency of the scheme.

Let us analyze the forward Euler scheme. The truncation error η_{n-1} in (17.5) is given through:

$$y'(t_{n-1}) + \eta_{n-1} = \frac{y(t_n) - y(t_{n-1})}{k}.$$

We need information about the solution at the old time step t^{n-1} in order to eliminate $y(t_n)$. Thus we use Taylor-Lagrange (see section 10.4) and develop $y(t^n)$ at the time point t^{n-1}:

$$y(t^n) = y(t^{n-1}) + y'(t^{n-1})k + \frac{1}{2}y''(\tau^{n-1})k^2.$$

We obtain the difference quotient of forward Euler by the following manipulation:

$$\frac{y(t^n) - y(t^{n-1})}{k} = y'(t^{n-1}) + \frac{1}{2}y''(\tau^{n-1})k.$$

We observe that the first terms correspond to the right hand side of (17.4). Thus the remainder term is

$$\frac{1}{2}y''(\tau^{n-1})k$$

and therefore the truncation error η_{n-1} can be estimated as

$$|\eta_{n-1}| \leq \max_{t\in[0,T]} \frac{1}{2}|y''(t)|k = O(k).$$

17.5.3 Convergence

With the help of the two previous subsections, we can easily show the following error estimates:

Theorem 17.5 (Convergence of implicit/explicit Euler). *We have*

$$\max_{t_n \in I} |y_n - y(t_n)| \leq c(T, y)k = O(k),$$

where $k := \max_n k_n$.

The proof does hold for both schemes, except that when we plug-in the stability estimate one should recall that the backward Euler scheme is unconditionally stable and the forward Euler scheme is only stable when the step size k is sufficiently small. It holds for $1 \leq n \leq N$:

$$\begin{aligned}
|y_n - y(t_n)| = |e_n| &= k|\sum_{k=0}^{n-1} B_E^{n-k}\eta_k| \\
&\leq k\sum_{k=0}^{n-1} |B_E^{n-k}\eta_k| \quad \text{(triangle inequality)} \\
&\leq k\sum_{k=0}^{n-1} |B_E^{n-k}|\,|\eta_k| \\
&\leq k\sum_{k=0}^{n-1} |B_E^{n-k}|\, Ck \quad \text{(consistency)} \\
&\leq k\sum_{k=0}^{n-1} 1\, Ck \quad \text{(stability)} \\
&= kN\, Ck \\
&= T\, Ck, \quad \text{where we used } k = T/N \\
&= C(T,y)k \\
&= O(k).
\end{aligned}$$

In Section 17.6 we demonstrate in terms of a numerical example that the forward Euler scheme will fail when the step size restrication is not satisfied, consequently the scheme is not stable and therefore, the convergence result does not hold true.

Theorem 17.6 (Convergence of trapezoidal rule)**.** *We have*

$$\max_{t_n \in I} |y_n(t_n) - y(t_n)| \leq c(T,y)k^2 = O(k^2).$$

The main message is that the Euler schemes both converge with order $O(k)$ (which is very slow) and the trapezoidal rule converges quadratically, i.e., $O(k^2)$.

17.6 Detailed numerical tests

In this section we demonstrate the previous concepts in terms of a numerical example. The programming code is written in octave (which is the open-source sister of MATLAB).

17.6.1 Problem statement

We solve our ODE model problem numerically. Let $a = g - m$ be $a = 0.25$ (test 1) or $a = -0.25$ (test 2) or $a = -10$ (test 3). The IVP is given by:

$$y' = ay, \quad y(t_0 = 2011) = 2.$$

The end time value is $T = 2014$. The tasks are:

- Use the forward Euler (FE), backward Euler (BE), and trapezoidal rule (CN) for the numerical approximation.
- Observe the accuracy in terms of the discretization error.
- Observe for (stiff) equations with a large negative coefficient $a = -10 \ll 1$ the behavior of the three schemes.

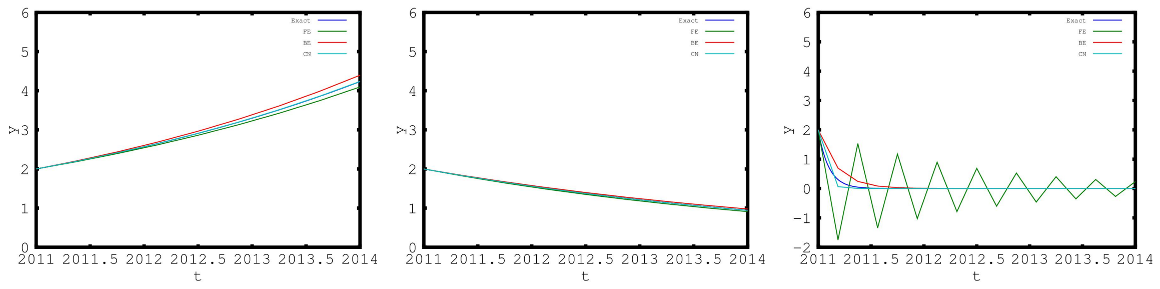


Figure 17.1: Example 17.6.1: on the left, the solution to test 1 is shown. In the middle, test 2 is plotted. On the right, the solution of test 3 with $N = 16$ (number of intervals) is shown. Here, $N = 16$ corresponds to a step size $k = 0.18$ which is slightly below the critical step size for convergence (see Section 17.6.3). Thus we observe the instable behavior of the forward Euler method, but also see slow convergence towards the continuous solution.

17.6.2 Discussion of the results for test 1 with a=0.25

In the following, we present our results for the end time value y_N (corresponding to $T = 2014$) for test case 1 ($a = 0.25$) on three mesh levels:

```
Scheme         #steps(N) k          y_N
===========================================
FE             8         0.37500    4.0961
BE             8         0.37500    4.3959
CN             8         0.37500    4.2363
-------------------------------------------
FE             16        0.18750    4.1624
BE             16        0.18750    4.3115
CN             16        0.18750    4.2346
-------------------------------------------
FE             32        0.093750   4.1975
BE             32        0.093750   4.2720
CN             32        0.093750   4.2341
===========================================
```

- In the second column, i.e., $8, 16, 32$, the number of steps (= number of intervals, i.e., so called mesh cells - speaking in PDE terminology) are given.

- In order to compute numerically the convergence order α with the help of formula (19.44), we work with $k = k_{max} = 0.375$. Then we identify in the third column of the above table that $P(k_{max}) = P(0.375), P(k_{max}/2) = P(0.1875)$ and $P(k_{max}/4) = P(0.09375)$. Formula (19.44) yields $\alpha = 0.9175$ for FE, $\alpha = 1.0921$ for BE, $\alpha = 2.0022$ for CN. This confirms linear convergence in the first two cases, and quadratic convergence in the last one.

- A further observation is that the forward Euler scheme is unstable for $N = 16$ and $a = -10$ and has a zig-zag curve, whereas the other two schemes follow the exact solution and the decreasing exp-function. But for sufficiently small step sizes, the forward Euler scheme is also stable which we know from our A-stability calculations. These step sizes can be explicitely determined for this ODE model problem and shown below.

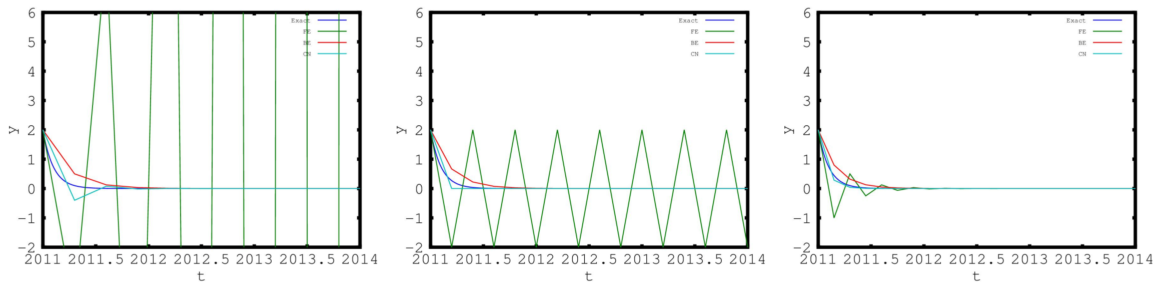


Figure 17.2: Example 17.6.1: tests 3a,3b,3d: Blow-up, constant zig-zag non-convergence, and convergence of the forward Euler method.

17.6.3 Investigating the instability of the forward Euler method: test 3 with a=-10

With the help of Example 17.1 let us understand how to choose stable step sizes k for the forward Euler method. The convergence interval reads:

$$|1+z| \leq 1 \quad \Rightarrow \quad |1+ak| \leq 1$$

In test 3, $a=-10$, which yields:

$$|1+z| \leq 1 \quad \Rightarrow \quad |1-10k| \leq 1$$

Thus, we need to choose a k that fulfills the previous relation. In this case this, $k<0.2$ is calculated. This means that for all $k<0.2$ we should have convergence of the forward Euler method and for $k \geq 0.2$ non-convergence (and in particular no stability!). We perform the following additional tests:

- Test 3a: $N=10$, yielding $k=0.3$;
- Test 3b: $N=15$, yielding $k=0.2$; exactly the boundary of the stability interval;
- Test 3c: $N=16$, yielding $k=0.1875$; from before;
- Test 3d: $N=20$, yielding $k=0.15$.

The results of test 3a,3b,3d are provided in Figure 17.2 and visualize very nicely the theoretically predicted behavior.

Chapter 18

Introduction to PDEs (partial differential equations) (complement)

In this chapter, we introduce notations for partial differential equations (PDE) and their numerical solution using the finite element method. These parts are the abbreviated version of the more extensive lecture notes from the second author [29]; an updated version including some basic python codes can be found here Wi21 (click on the link 'Wi21' to obtain *.pdf)

18.1 Laplace equation / Poisson problem

We first present the classical Poisson problem. This problem has several applications, e.g., Fick's law of diffusion or heat conduction (temporal diffusion) or the deflection of a solid membrane:

Formulation 18.1 (Laplace problem / Poisson problem)**.** *Let Ω be an open set. The* ***Laplace equation*** *reads:*

$$-\Delta u = 0 \quad \textit{in } \Omega.$$

The ***Poisson problem*** *reads:*

$$-\Delta u = f \quad \textit{in } \Omega.$$

Definition 18.1. *A C^2 function (C^2 means two times continuously differentiable) that satisfies the Laplace equation is called* ***harmonic*** *function.*

The physical interpretation is as follows. Let u denote the density of some quantity, for instance concentration or temperature, in equilibrium. If G is any smooth region $G \subset \Omega$, the flux F of the quantity u through the boundary ∂G is zero:

$$\int_{\partial G} F \cdot n \, dx = 0. \tag{18.1}$$

Here F denotes the flux density and n the outer normal vector. Gauss' divergence theorem yields:

Formulation 18.2 (Integral conservation equation)**.** *Extending* (18.1) *to Ω we have*

$$\int_{\partial \Omega} F \cdot n \, dx = 0.$$

Employing Gauss' divergence theorem, it holds

$$\int_{\Omega} \nabla \cdot F \, dx = 0. \tag{18.2}$$

This integral form of a conservation equation is very often the initial representation of physical laws.

The question is now how we come from the integral form to a pointwise differential equation. We first need to assume that the integrand is sufficiently regular. Here, F should be continuously differentiable. Second, we request that (18.2) holds for arbitrary $G \subset \Omega$ as previously mentioned. Then, it can be inferred from results in analysis (calculus) that the integrand is pointwise zero:

Formulation 18.3 (Conservation equation in differential form)**.**

$$\nabla \cdot F = 0 \quad \textit{in } \Omega. \tag{18.3}$$

Constitutive laws Now we need a second assumption (or better a relation) between the flux and the quantity u. Such relations do often come from material properties and are so-called **constitutive laws.** In many situations it is reasonable to assume that the flux F is proportional to the negative gradient $-\nabla u$ of the quantity u. This means that flow goes from regions with a higher concentration to lower concentration regions. For instance, the rate at which energy 'flows' (or diffuses) as heat from a warm body to a colder body is a function of the temperature difference. The larger the temperature difference, the larger the diffusion. We consequently obtain as further relation:

$$F = -\nabla u.$$

Plugging into the Equation (18.3) yields:

$$\nabla \cdot F = \nabla \cdot (-\nabla u) = -\nabla \cdot (\nabla u) = -\Delta u = 0.$$

This is the simplest derivation one can make. Adding more knowledge on the underlying material of the body, a material parameter $a > 0$ can be added:

$$\nabla \cdot F = \nabla \cdot (-a\nabla u) = -\nabla \cdot (a\nabla u) = -a\Delta u = 0.$$

And adding a nonconstant and spatially dependent material further yields:

$$\nabla \cdot F = \nabla \cdot (-a(x)\nabla u) = -\nabla \cdot (a(x)\nabla u) = 0.$$

In this last equation, we do not obtain any more the classical Laplace equation but a diffusion equation in divergence form.

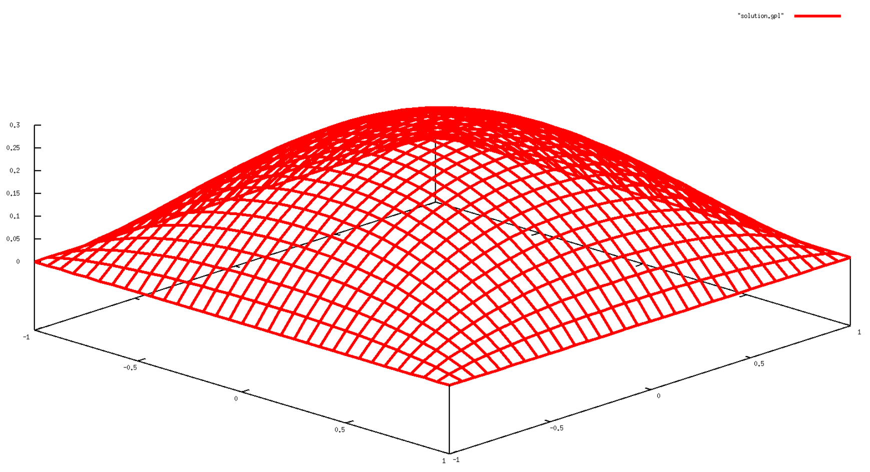


Figure 18.1: Solution of the Poisson problem on $\Omega = (-1, 1)^2$ with the right hand side $f = 1$.

18.2 Mass conservation / First-order hyperbolic problem

In this next example let us again consider some concentration, but now a time dependent situation, which brings us to a first-order hyperbolic equation. The application might be transport of a species/-concentration in some fluid flow (e.g. water) or nutrient transport in blood flow. Let $\Omega \subset \mathbb{R}^n$ be an open domain, $x \in \Omega$ the spatial variable and t the time. Let $\rho(x,t)$ be the density of some quantity (e.g., concentration) and let $v(x,t)$ its velocity. Then the vector field

$$F = \rho v \quad \text{in } \mathbb{R}^n$$

denotes the flux of this quantity. Let G be a subset of Ω. Then we have as in the previous case (Equation (18.1)) the definition:

$$\int_{\partial G} F \cdot n \, dx.$$

But this time we do not assume the 'equilibrium state' but 'flow'. That is to say that the outward flow through the boundary ∂G must coincide with the temporal decrease of the quantity:

$$\int_{\partial G} F \cdot n \, ds = -\frac{d}{dt} \int_G \rho \, dx.$$

We then apply again Gauss' divergence theorem to the left hand side and bring the resulting term to the right hand side. We now encounter a principle difficulty that the domain G may depend on time and consequently integration and differentiation do not commute. Therefore, in a first step we need to transform the integrand of

$$\frac{d}{dt} \int_G \rho \, dx$$

onto a fixed reference configuration $\hat{G}$ in which we can insert the time derivative under the integral sign. Then we perform the calculation and transform lastly everything back to the physical domain G. Let the mapping between $\hat{G}$ and G be denoted by T. Then, it holds:

$$x \in G : x = T(\hat{x}, t), \quad \hat{x} \in \hat{G}.$$

Moreover, $dx = J(\hat{x},t)d\hat{x}$, where $J := \det(\nabla T)$. Using the substitution rule (change of variables) in higher dimensions yields:

$$\frac{d}{dt} \int_G \rho(x,t) \, dx = \frac{d}{dt} \int_{\hat{G}} \rho(T(\hat{x},t),t) J(\hat{x},t) d\hat{x}.$$

We eliminated time dependence on the right hand side integral and thus differentiation and integration commute now:

$$\int_{\hat{G}} \frac{d}{dt}\Big(\rho(T(\hat{x},t),t)J(\hat{x},t)\Big) d\hat{x} = \int_{\hat{G}} \Big(\frac{d}{dt}\rho(T(\hat{x},t),t) \cdot J(\hat{x},t) + \rho(T(\hat{x},t),t)\frac{d}{dt}J(\hat{x},t)\Big) d\hat{x}$$

Here, $\frac{d}{dt}\rho(T(\hat{x},t),t)$ is the material time derivative of a spatial field (see e.g., [17]), which is <u>not</u> the same as the partial time derivative! In the last step, we need the Eulerian expansion formula

$$\frac{d}{dt} J = \nabla \cdot v \, J.$$

Then:

$$
\begin{aligned}
&\int_{\hat G}\Big(\frac{d}{dt}\rho(T(\hat x,t),t)\cdot J(\hat x,t)+\rho(T(\hat x,t),t)\frac{d}{dt}J(\hat x,t)\Big)d\hat x\\
&=\int_{\hat G}\Big(\frac{d}{dt}\rho(T(\hat x,t),t)\cdot J(\hat x,t)+\rho(T(\hat x,t),t)\nabla\cdot v\,J\Big)d\hat x\\
&=\int_{\hat G}\Big(\frac{d}{dt}\rho(T(\hat x,t),t)+\rho(T(\hat x,t),t)\nabla\cdot v\Big)J(\hat x,t)d\hat x\\
&=\int_{G}\Big(\frac{d}{dt}\rho(x,t)+\rho(x,t)\nabla\cdot v\Big)dx\\
&=\int_{G}\Big(\partial_t\rho(x,t)+\nabla\rho(x,t)\,v+\rho(x,t)\nabla\cdot v\Big)dx\\
&=\int_{G}\Big(\partial_t\rho(x,t)+\nabla\cdot(\rho v)\Big)dx.
\end{aligned}
$$

Collecting the previous calculations brings us to:

$$\int_G(\partial_t\rho+\nabla\cdot F)\,dx=\int_G(\partial_t\rho+\nabla\cdot(\rho v)\,dx,\quad \text{where } F=\rho v \text{ as introduced before.}$$

This is the so-called continuity equation (or mass conservation). Since G was arbitrary, we are allowed to write the strong form:

$$\partial_t\rho+\nabla\cdot(\rho v)=0\quad\text{in }\Omega. \tag{18.4}$$

If there are sources or sinks, denoted by f, inside Ω we obtain the more general formulation:

$$\partial_t\rho+\nabla\cdot(\rho v)=f\quad\text{in }\Omega.$$

On the other hand, various simplifications of Equation (18.4) can be made when certain requirements are fulfilled. For instance, if ρ is not spatially varying, we obtain:

$$\partial_t\rho+\rho\nabla\cdot v=0\quad\text{in }\Omega.$$

If furthermore ρ is constant in time, we obtain:

$$\nabla\cdot v=0\quad\text{in }\Omega.$$

This is now the mass conservation law that appears for instance in the incompressible Navier-Stokes equations, which are discussed a little bit later below.

In terms of the density ρ, we have shown in this section:

Theorem 18.4 (Reynolds' transport theorem). *Let $\Phi:=\Phi(x,t)$ be a smooth scalar field and Ω a (moving) domain. It holds:*

$$\frac{d}{dt}\int_\Omega\Phi\,dx=\int_{\partial\Omega}\Phi v\cdot n\,ds+\int_\Omega\frac{\partial\Phi}{\partial t}\,dx$$

The first term on the right hand side represents the ***rate of transport*** *(also known as the* ***outward normal flux****) of the quantity Φv across the boundary surface $\partial\Omega$. This contribution originates from the moving domain Ω. The second contribution is the* ***local time rate of change*** *of the spatial scalar field Φ. If the domain Ω does not move, the first term on the right hand side will of course vanish. Using Gauss' divergence theorem it holds furthermore:*

$$\frac{d}{dt}\int_\Omega\Phi\,dx=\int_\Omega\nabla\cdot(\Phi\,v)\,dx+\int_\Omega\frac{\partial\Phi}{\partial t}\,dx.$$

18.3 Elasticity (Lamé-Navier)

This example is already difficult because a system of nonlinear equations is considered:

Formulation 18.5. *Let* $\hat\Omega_s \subset \mathbb{R}^n, n = 3$ *with the boundary* $\partial\hat\Omega := \hat\Gamma_D \cup \hat\Gamma_N$. *Furthermore, let* $I := (0, T]$ *where* $T > 0$ *is the end time value. The equations for geometrically non-linear elastodynamics in the reference configuration* $\hat\Omega$ *are given as follows: Find vector-valued displacements* $\hat u_s := (\hat u_s^{(x)}, \hat u_s^{(y)}, \hat u_s^{(z)}) :$ $\hat{O}mega_s \times I \to \mathbb{R}^n$ *such that*

$$\begin{aligned}
\hat\rho_s \partial_t^2 \hat u_s - \hat\nabla \cdot (\hat F \hat\Sigma) &= 0 \quad \text{in } \hat\Omega_s \times I, \\
\hat u_s &= 0 \quad \text{on } \hat\Gamma_D \times I, \\
\hat F \hat\Sigma \cdot \hat n_s &= \hat h_s \quad \text{on } \hat\Gamma_N \times I, \\
\hat u_s(0) &= \hat u_0 \quad \text{in } \hat\Omega_s \times \{0\}, \\
\hat v_s(0) &= \hat v_0 \quad \text{in } \hat\Omega_s \times \{0\}.
\end{aligned}$$

We deal with two types of boundary conditions: Dirichlet and Neumann conditions. Furthermore, two initial conditions on the displacements and the velocity are required. The constitutive law is given by the geometrically nonlinear tensors (see e.g., Ciarlet [6]):

$$\hat\Sigma = \hat\Sigma_s(\hat u_s) = 2\mu \hat E + \lambda tr(\hat E) I, \quad \hat E = \frac{1}{2}(\hat F^T \hat F - I). \tag{18.5}$$

Here, μ *and* λ *are the Lamé coefficients for the solid. The solid density is denoted by* $\hat\rho_s$ *and the solid deformation gradient is* $\hat F = \hat I + \hat\nabla \hat u_s$ *where* $\hat I \in \mathbb{R}^{3\times 3}$ *is the identity matrix. Furthermore,* $\hat n_s$ *denotes the normal vector.*

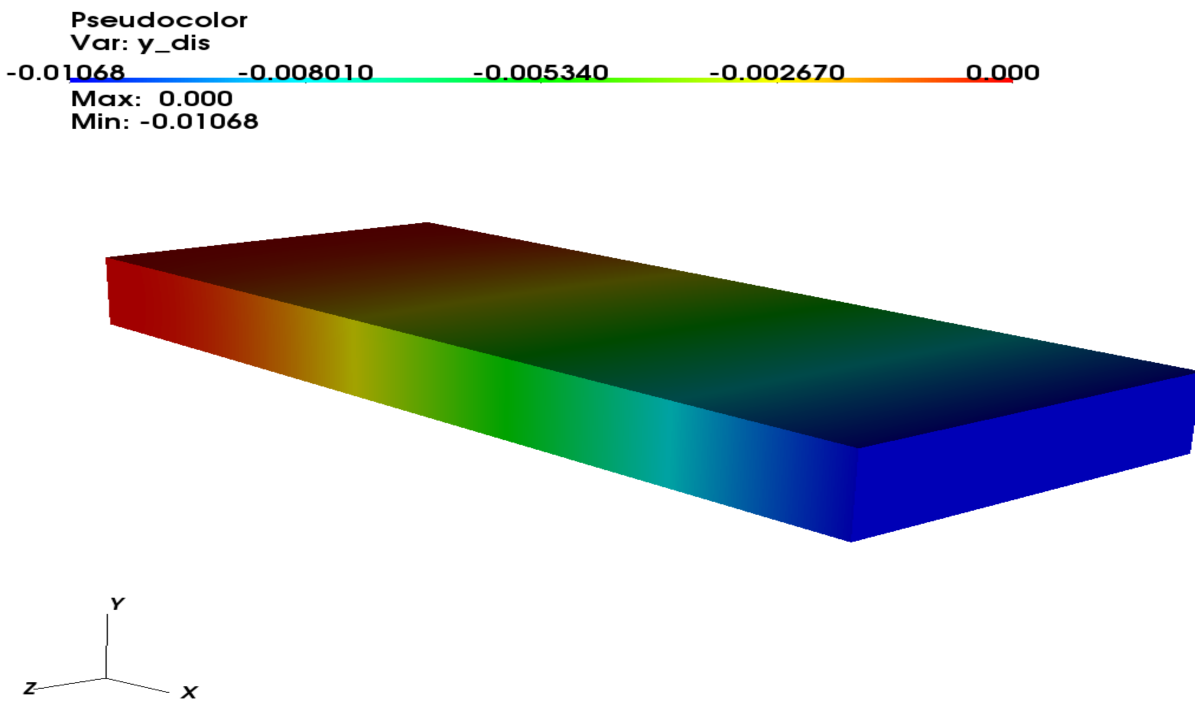


Figure 18.2: Deformed elastic flag.

18.4 The incompressible, isothermal Navier-Stokes equations (fluid mechanics)

Flow equations in general are extremely important and have an incredible amount of possible applications such as for example water (fluids), blood flow, wind, weather forecast, aerodynamics:

Formulation 18.6. *Let $\Omega_f \subset \mathbb{R}^n, n = 3$. Furthermore, let the boundary be split into $\partial\Omega_f := \Gamma_{in} \cup \Gamma_{out} \cup \Gamma_D \cup \Gamma_i$. The isothermal, incompressible (non-linear) Navier-Stokes equations read: Find vector-valued velocities $v_f : \Omega_f \times I \to \mathbb{R}^n$ and a scalar-valued pressure $p_f : \Omega_f \times I \to \mathbb{R}$ such that*

$$\begin{aligned}
\rho_f \partial_t v_f + \rho_f v_f \cdot \nabla v_f - \nabla \cdot \sigma_f(v_f, p_f) &= 0 \quad && \text{in } \Omega_f \times I,\\
\nabla \cdot v_f &= 0 && \text{in } \Omega_f \times I,\\
v_f^D &= v_{in} && \text{on } \Gamma_{in} \times I,\\
v_f &= 0 && \text{on } \Gamma_D \times I,\\
-p_f n_f + \rho_f \nu_f \nabla v_f \cdot n_f &= 0 && \text{on } \Gamma_{out} \times I,\\
v_f &= h_f && \text{on } \Gamma_i \times I,\\
v_f(0) &= v_0 && \text{in } \Omega_f \times \{t = 0\},
\end{aligned}$$

where the (symmetric) Cauchy stress is given by

$$\sigma_f(v_f, p_f) := -p_f I + \rho_f \nu_f (\nabla v_f + \nabla v_f^T),$$

with the density ρ_f and the kinematic viscosity ν_f. The normal vector is denoted by n_f.

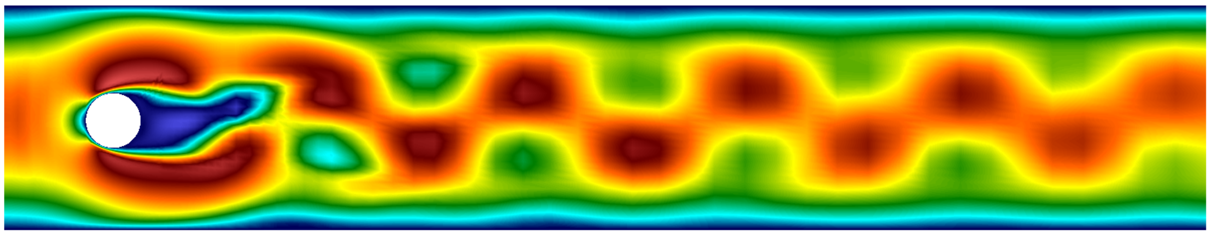


Figure 18.3: Prototype example of a fluid mechanics problem (isothermal, incompressible Navier-Stokes equations): the famous Karman vortex street. The setting is based on the benchmark setting [26] and the code can be found in NonStat Example 1 in [13] www.dopelib.net.

Remark 18.1. *The two Formulations 18.5 and 18.6 are very important in many applications and their coupling results in fluid-structure interaction. Here we notice that fluid flows are usually modeled in Eulerian coordinates and solid deformations in Lagrangian coordinates. In the case of small displacements, the two coordinate systems can be identified, i.e., $\hat{\Omega} \simeq \Omega$. This is the reason why in many basic books - in particular basics of PDE theory or basics of numerical algorithms - the 'hat' notation (or similar notation to distinguish coordinate systems) is not used.*

18.5 Three important linear PDEs

From the previous considerations, we can extract **three important types of linear PDEs**. But we refrain from giving the impression that all differential equations can be classified and then a recipe for solving them applies. This is definitely not true - in particular for nonlinear PDEs. However, often 'new' equations can be related or simplified to these three types. For this reason, it is important to known them.

They read:

- Poisson problem: $-\Delta u = f$ is elliptic: second order in space and no time dependence.
- Heat equation: $\partial_t u - \Delta u = f$ is parabolic: second order in space and first order in time.
- Wave equation: $\partial_t^2 u - \Delta u = f$ is hyperbolic: second order in space and second order in time.

All these three equations have in common that their spatial order is two (the highest derivative with respect to spatial variables that occurs in the equation). With regard to time, there are differences: the heat equation is of first order whereas the wave equation is second order in time.

Let us now consider these three types in a bit more detail.

18.5.1 Elliptic equations and Poisson problem/Laplace equation

The Poisson equation is a boundary-value problem and will be derived from the general definition (very similar to the form before). Elliptic problems are second-order in space and have no time dependence.

Formulation 18.7. *Let $f : \Omega \to \mathbb{R}$ be given. Furthermore, Ω is an open, bounded set of $\mathbb{R}^n$. We seek the unknown function $u : \bar{\Omega} \to \mathbb{R}$ such that*

$$Lu = f \quad \text{in } \Omega, \tag{18.6}$$
$$u = 0 \quad \text{on } \partial\Omega. \tag{18.7}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x)\partial_{x_i}u) + \sum_{i=1}^{n} b_i(x)u\partial_{x_i} + c(x)u, \quad u = u(x), \tag{18.8}$$

with the symmetry assumption $a_{ij} = a_{ji}$ and given coefficient functions a_{ij}, b_i, c. Alternatively we often use the compact notation with derivatives defined in terms of the nabla-operator:

$$Lu := -\nabla \cdot (a\nabla u) + b\nabla u + cu.$$

Finally we notice that the boundary condition (18.7) *is called* ***homogeneous Dirichlet condition****.*

Remark 18.2. *Other possible boundary conditions are introduced in Section 18.7.*

Remark 18.3. *If the coefficient functions a, b, c also depend on the solution u we obtain a nonlinear PDE.*

Consider now the main part

$$\tilde{L}u := \sum_{i,j=1}^{n} a_{ij}(x)\partial_{ij}u$$

and we assume symmetry $a_{ij} = a_{ji}$. The operator $\tilde{L}$ (and also L of course) generates a quadratic form Σ, which is defined as

$$\Sigma(\xi) := \sum_{i,j=1}^{n} a_{ij}(x)\xi_i\xi_j.$$

The properties of the form Σ (and consequently the classification of the underlying PDE) depends on the eigenvalues of the matrix A:

$$A = (a_{ij})_{ij=1}^{n}.$$

Definition 18.2 (Elliptic: sign of all eigenvalues is the same)**.** *At the a given point $x \in \Omega$, the differential operator L is said to be elliptic if all eigenvalues of A are non-zero and have the same sign.*

Equivalently one can say:

Definition 18.3. *A PDE operator L is (uniformly) elliptic if there exists a constant $\theta > 0$ such that*

$$\sum_{i,j=1}^{n} a_{ij}(x)\xi_i\xi_j \geq \theta|\xi|^2,$$

for a.e. $x \in \Omega$ and all $\xi \in \mathbb{R}^n$.

Formulation 18.8 (Poisson problem)**.** *Setting in Formulation 18.7, $a_{ij} = \delta_{ij}$ and $b_i = 0$ and $c = 0$, we obtain the Laplace operator. Let $f : \Omega \to \mathbb{R}$ be given. Furthermore, Ω is an open, bounded set of $\mathbb{R}^n$. We seek the unknown function $u : \bar{\Omega} \to \mathbb{R}$ such that*

$$Lu = f \quad in\ \Omega, \tag{18.9}$$

$$u = 0 \quad on\ \partial\Omega. \tag{18.10}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\nabla \cdot (\nabla u) = -\Delta u.$$

Some important physical laws are related to the Laplace operator (partially taken from [12]):

1. Fick's law of chemical diffusion
2. Fourier's law of heat conduction
3. Ohm's law of electrical conduction
4. Small deformations in elasticity.

In the following we discuss some characteristics of the solution of the Laplace problem, which can be generalized to general elliptic problems. Solutions to Poisson's equation attain their maximum on the boundary and not in the interior of the domain. This property will also carry over to the discrete problem and can be used to **check the correctness of the corresponding numerical solution.**

Definition 18.4 (Harmonic function)**.** *A C^2 function satisfying $\Delta u = 0$ is called a harmonic function.*

Theorem 18.9 (Strong maximum principle for the Laplace problem)**.** *Suppose $u \in C^2(\Omega) \cap C(\bar{\Omega})$ is a harmonic function. Then*

$$\max_{\bar{\Omega}} u = \max_{\partial\Omega} u.$$

Moreover, if Ω is connected and there exists a point $y \in \Omega$ in which

$$u(y) = \max_{\bar{\Omega}} u,$$

then u is constant within Ω. The same holds for $-u$, but then for minima.

Proof. See Evans [12]. □

Remark 18.4 (Maximum principle in practice)**.** *An illustration of the maximum principle in practice for a 1D setting is provided in Figure 19.3, where the maximum values are clearly obtained on the two boundary points of the domain.*

Corollary 18.10. *For finding $u \in C^2(\Omega) \cap C(\bar{\Omega})$, such that*

$$\Delta u = 0 \quad in\ \Omega,$$

$$u = g \quad on\ \partial\Omega,$$

with $g \geq 0$, it holds that

$$u > 0 \quad everywhere\ in\ \Omega \quad if \quad g > 0\ on\ some\ part\ of\ \partial\Omega. \tag{18.11}$$

This property also shows that the Laplace equation has an infinite propagation speed of perturbations: if we change the solution by ε in one point, the solution in the entire domain will (slightly) change.

From the maximum principle we obtain immediately uniqueness of a solution:

Theorem 18.11 (Uniqueness of the Laplace problem)**.** *Let $g \in C(\partial\Omega)$ and $f \in C(\Omega)$. Then there exists at most one solution $u \in C^2(\Omega) \cap C(\bar{\Omega})$ of the boundary-value problem:*

$$-\Delta u = f \quad \text{in } \Omega, \tag{18.12}$$

$$u = g \quad \text{on } \partial\Omega. \tag{18.13}$$

Proof. If u_1 and u_2 both satisfy the equation, we apply the maximum principle to the harmonic functions

$$w = u_1 - u_2, \quad w = -(u_1 - u_2).$$

This yields $u_1 = u_2$. □

18.5.2 Parabolic equations and heat equation

We now study parabolic problems, which contain as the most famous example the heat equation. Parabolic problems are second order in space and first order in time.

In this section, we assume $\Omega \subset \mathbb{R}^n$ to be an open and bounded set. The time interval is given by $I := (0, T]$ for some fixed end time value $T > 0$.

We consider the initial/boundary-value problem (IBVP):

Formulation 18.12. *Let $f : \Omega \times I \to \mathbb{R}$ and $u_0 : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that*[1]

$$\partial_t u + Lu = f \quad \text{in } \Omega \times I, \tag{18.14}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{18.15}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{18.16}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x,t)\partial_{x_i} u) + \sum_{i=1}^{n} b_i(x,t)\partial_{x_i} u + c(x,t)u, \quad u = u(x,t)$$

for given (possibly spatial and time-dependent) coefficient functions a_{ij}, b_i, c.

We have the following:

Definition 18.5. *The PDE operator $\partial_t + L$ is (uniformly) parabolic if there exists a constant $\theta > 0$ such that*

$$\sum_{i,j=1}^{n} a_{ij}(x,t)\xi_i\xi_j \geq \theta|\xi|^2,$$

for all $(x,t) \in \Omega \times I$ and all $\xi \in \mathbb{R}^n$. In particular this operator is elliptic in the spatial variable x for each fixed time $0 \leq t \leq T$.

Definition 18.6 (Parabolic: one eigenvalue zero; all others have the same sign)**.** *For parabolic PDEs one eigenvalue of the space-time coefficient matrix is zero whereas the others have the same sign.*

Formulation 18.13 (Heat equation)**.** *Setting in Formulation 18.12, $a_{ij} = \delta_{ij}$ and $b_i = 0$ and $c = 0$, we obtain the Laplace operator. Let $f : \Omega \to \mathbb{R}$ be given. Furthermore, Ω is an open, bounded set of $\mathbb{R}^n$. We seek the unknown function $u : \bar{\Omega} \to \mathbb{R}$ such that*

$$\partial_t u + Lu = f \quad \text{in } \Omega \times I, \tag{18.17}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{18.18}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{18.19}$$

[1]For the heat equation, we should have better used the notation T, the temperature, for the variable rather than u. But to stay consistant with the entire notation, we still use u.

Here, the linear second-order differential operator is defined by:

$$Lu := -\nabla \cdot (\nabla u) = -\Delta u.$$

The heat equation has an infinite propagation speed for disturbances. If the initial temperature is nonnegative and is positive somewhere, the temperature at any later time is everywhere positive. For the heat equation, a very similar maximum principle as for elliptic problems holds true. This principle is extended to the parabolic boundary. We deal with a diffusions problem and flow points into the direction of the negative gradient. Define $Q = (0,T] \times \Omega$ the space-time cylinder. The parabolic boundary is given by $\Gamma := \bar{Q} - Q$, namely:

$$\Gamma = \Big([0,T] \times \partial\Omega\Big) \cup \Big(\{0\} \times \Omega\Big).$$

This definition contains the bottom and the vertical sides of $Q \times [0,T]$, but not the top. For $f = 0$ the maximum temperature u is obtained on the parabolic boundary. In other words, the maximum is taken at the initial time step or on the boundary of the spatial domain.

Theorem 18.14 (Strong parabolic maximum principle for the heat equation)**.** *Suppose $u \in C_1^2(\Omega \times I) \cap C(\bar{\Omega} \times I)$ is a solution of the heat equation:*

$$\partial_t u = \Delta u \quad \text{in } \Omega \times I = (0,T).$$

Then

$$\max_{\bar{\Omega}\times I} u = \max_{\Gamma} u.$$

In other words: the maximum u (or the minimum) is obtained either at $t = 0$ or on $[0,T] \times \partial\Omega$. A numerical test demonstrating the maximum principle is provided in Section 19.4.4.

Remark 18.5. *As for elliptic problems, the maximum principle can be used to proof uniqueness for the parabolic case.*

18.5.3 Hyperbolic equations and the wave equation

In this section, we shall study hyperbolic problems, which are natural generalizations of the wave equation. As for parabolic problems, let $\Omega \subset \mathbb{R}^n$ to be an open and bounded set. The time interval is given by $I := (0,T]$ for some fixed end time value $T > 0$.

We consider the initial/boundary-value problem:

Formulation 18.15. *Let $f : \Omega \times I \to \mathbb{R}$ and $u_0, v_0 : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that*

$$\partial_t^2 u + Lu = f \quad \text{in } \Omega \times I, \tag{18.20}$$
$$u = 0 \quad \text{on } \partial\Omega \times [0,T], \tag{18.21}$$
$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}, \tag{18.22}$$
$$\partial_t u = v_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{18.23}$$

In the last line, $\partial_t u = v$ can be identified as the velocity. Furthermore, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x,t)\partial_{x_i} u) + \sum_{i=1}^{n} b_i(x,t)\partial_{x_i} u + c(x,t)u, \quad u = u(x,t)$$

for given (possibly spatial and time-dependent) coefficient functions a_{ij}, b_i, c.

Remark 18.6. *The wave equation is often written in terms of a first-order system in which the velocity is introduced and a second-order time derivative is avoided. Then the previous equation reads: Find* $u : \bar{\Omega} \times I \to \mathbb{R}$ *and* $v : \bar{\Omega} \times I \to \mathbb{R}$ *such that*

$$\partial_t v + Lu = f \quad \text{in } \Omega \times I, \tag{18.24}$$

$$\partial_t u = v \quad \text{in } \Omega \times I, \tag{18.25}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{18.26}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}, \tag{18.27}$$

$$v = v_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{18.28}$$

We have the following:

Definition 18.7. *The PDE operator* $\partial_t^2 + L$ *is (uniformly) hyperbolic if there exists a constant* $\theta > 0$ *such that*

$$\sum_{i,j=1}^{n} a_{ij}(x,t)\xi_i\xi_j \geq \theta|\xi|^2,$$

for all $(x,t) \in \Omega \times I$ *and all* $\xi \in \mathbb{R}^n$*. In particular this operator is elliptic in the spatial variable* x *for each fixed time* $0 \leq t \leq T$*.*

Definition 18.8 (Hyperbolic: one eigenvalue has a different sign than all others)**.** *A space-time differential operator is called hyperbolic when one eigenvalue has a different sign than all the others.*

And as before, setting the coefficient functions to trivial values, we obtain the original wave equation. In contrast to parabolic problems, a strong maximum principle does not hold for hyperbolic equations. And consequently, the propagation speed is finite. This means that a disturbance at some point $x \in \Omega$ at some time $t \in I$ is not immediately transported to any point $x \in \Omega$ at any time $t \in I$.

18.6 Nonconstant coefficients. Change of the PDE types elliptic, parabolic, hyperbolic

Even so that we define separate types of PDEs, in many processes there is a mixture of these classes in one single PDE - depending on the size of certain parameters. For instance, the Navier-Stokes equations (only showing here the conservation of momentum):

$$\partial_t v + v \cdot \nabla v - \frac{1}{Re}\Delta v + \nabla p = f$$

for modeling fluid flow, vary between parabolic and hyperbolic type depending on the Reynold's number $Re \sim \nu_f^{-1}$ (respectively a characteristic velocity and a characteristic length). For $Re \to 0$, we obtain a first-order hyperbolic equation:

$$\partial_t v + v \cdot \nabla v + \nabla p = f,$$

while for $Re \to \infty$, we obtain a parabolic type:

$$\partial_t v - \frac{1}{Re}\Delta v + \nabla p = f.$$

Relating this to the general definition of the differential operator L, we can say that the coefficients a_{ij} of the differential operator L are non-constant and change with respect to space and time.

18.7 Boundary conditions and initial data

As seen in the previous sections, all PDEs are complemented with boundary conditions and in time-dependent cases, also with initial conditions. Actually, these are crucial ingredients for solving differential equations. Often, one has the (wrong) impression that only the PDE itself is of importance. But what happens on the boundaries finally yields the 'result' of the computation. And this holds true for analytical (classical) as well as computational solutions.

18.7.1 Boundary conditions

In Section 18.5, for simplicity, we have mainly dealt with homogeneous Dirichlet conditions of the form $u = 0$ on the boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N \cup \partial\Omega_R$. In general three types of conditions are of importance:

- Dirichlet (or essential) boundary conditions: $u = g_D$ on $\partial\Omega_D$; when $g_D = 0$ we say 'homogeneous' boundary condition.
- Neumann (or natural) boundary conditions: $\partial_n u = g_N$ on $\partial\Omega_N$; when $g_N = 0$ we say 'homogeneous' boundary condition.
- Robin (third type) boundary condition: $au + b\partial_n u = g_R$ on $\partial\Omega_R$; when $g_R = 0$ we say 'homogeneous' boundary condition.

On each boundary section, only one of the three conditions can be described. In particular the natural conditions generalize when dealing with more general operators L. Here in this section, we have assumed $L := -\Delta u$.

Remark 18.7. *Going from differential equations to variational formulations, Dirichlet conditions are explicitly built into the function space, for this reason they are called **essential boundary conditions**. Neumann conditions appear explicitly through integration by parts; for this reason they are called **natural boundary conditions**.*

18.7.2 Initial conditions

The number of initial conditions depends on the order of temporal time derivatives. The usual notation is:

$$u = u_0 \quad \text{in } \Omega \times \{t = 0\}.$$

As examples, we have:

- $\partial_t u - ...$: first order in time: one initial condition.
- $\partial_t^2 u - ...$: second order in time: two initial conditions.

18.7.3 Example: temperature in a room

Example 18.1. *Let us compute the heat distribution in our room b302. The room volume is Ω. The window wall is a Dirichlet boundary $\partial_D\Omega$ and the remaining walls are Neumann boundaries $\partial_N\Omega$. Let K be the air viscosity.*

We consider the heat equation: Find $T : \Omega \times I \to \mathbb{R}$ such that

$$\begin{aligned}
\partial_t T + (v\cdot\nabla)T - \nabla\cdot(K\nabla T) &= f && \text{in } \Omega\times I,\\
T &= 18^\circ C && \text{on } \partial_D\Omega\times I,\\
K\nabla T\cdot n &= 0 && \text{on } \partial_N\Omega\times I,\\
T(0) &= 15^\circ C && \text{in } \Omega\times\{0\}.
\end{aligned}$$

The homogeneous Neumann condition means that there is no heat exchange on the respective walls (thus neighboring rooms will have the same room temperature on the respective walls). The nonhomogeneous Dirichlet condition states that there is a given temperature of $18^\circ C$, which is constant in time and space (but this condition may be also non-constant in time and space). Possible heaters in the room can be modeled via the right hand side f. The vector $v : \Omega \to \mathbb{R}^3$ denotes a given flow field yielding a convection of the heat, for instance wind. We can assume $v \approx 0$. Then the above equation is reduced to the original heat equation: $\partial_t T - \nabla\cdot(K\nabla T) = f$.

18.8 The general definition of a differential equation

After the previous examples, let us precise the definition of a differential equation. In order to allow for largest possible generality we first need to introduce some notation. Common is to use the **multiindex notation**.

Definition 18.9 (Evans [12]). *Let $\Omega \subset \mathbb{R}^n$ be open. Here, n denotes the total dimension including time. Furthermore, let $k \geq 1$ an integer that denotes the order of the differential equation. Then, a differential equation can be expressed as: Find $u : \Omega \to \mathbb{R}$ such that*

$$F(D^k u, D^{k-1}u, \dots, D^2 u, Du, u, x) = 0 \quad x \in \Omega,$$

where

$$F : \mathbb{R}^{n^k} \times \mathbb{R}^{n^{k-1}} \times \dots \times \mathbb{R}^{n^2} \times \mathbb{R}^n \times \mathbb{R} \times \Omega \to \mathbb{R}.$$

Example 18.2. *We provide some examples. Let us assume the spatial dimension to be 2 and the temporal dimension is 1. That is for a time-dependent ODE (ordinary differential equation) problem $n = 1$ and for time-dependent PDE cases, $n = 2 + 1 = 3$, and for stationary PDE examples $n = 2$.*

1. *ODE model problem: $F(Du, u) := u' - au = 0$ where $F : \mathbb{R}^1 \times \mathbb{R} \to \mathbb{R}$. Here $k = 1$.*

2. *Laplace operator: $F(D^2u) := -\Delta u = 0$ where $F : \mathbb{R}^4 \to \mathbb{R}$. That is $k = 2$ and lower derivatives of order 1 and 0 do not exist. We notice that in the general form it holds*

$$D^2 u = \begin{pmatrix} \partial_{xx} u & \partial_{yx} u \\ \partial_{xy} u & \partial_{yy} u \end{pmatrix}$$

and specficially now for the Laplacian (consider the sign as well!):

$$D^2 u = \begin{pmatrix} -\partial_{xx} u & 0 \\ 0 & -\partial_{yy} u \end{pmatrix}$$

3. *Heat equation: $F(D^2u, Du) = \partial_t u - \Delta u = 0$ where $F : \mathbb{R}^9 \times \mathbb{R}^3 \to \mathbb{R}$. Here $k = 2$ is the same as before, but a lower-order derivative of order 1 in form of the time derivative does exist. We provide again details on D^2u:*

$$D^2 u = \begin{pmatrix} \partial_{tt} u & \partial_{xt} u & \partial_{yt} u \\ \partial_{tx} u & \partial_{xx} u & \partial_{yx} u \\ \partial_{ty} u & \partial_{xy} u & \partial_{yy} u \end{pmatrix}, \qquad Du = \begin{pmatrix} \partial_t u \\ \partial_x u \\ \partial_y u \end{pmatrix}$$

and for the heat equation:

$$D^2 u = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\partial_{xx} u & 0 \\ 0 & 0 & -\partial_{yy} u \end{pmatrix}, \qquad Du = \begin{pmatrix} \partial_t u \\ 0 \\ 0 \end{pmatrix}$$

4. *Wave equation: $F(D^2u) = \partial_t^2 u - \Delta u = 0$ where $F : \mathbb{R}^9 \to \mathbb{R}$. Here it holds specifically*

$$D^2 u = \begin{pmatrix} \partial_{tt} u & 0 & 0 \\ 0 & -\partial_{xx} u & 0 \\ 0 & 0 & -\partial_{yy} u \end{pmatrix}.$$

We finally compare the coefficient matrices and recall the type classifications. Let us denote the coefficient matrix by A. Then:

$$D^2u = \begin{pmatrix} -\partial_{xx}u & 0 \\ 0 & -\partial_{yy}u \end{pmatrix} \quad \to \quad A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \quad \to \quad \textit{Same sign: elliptic}$$

$$D^2u = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\partial_{xx}u & 0 \\ 0 & 0 & -\partial_{yy}u \end{pmatrix} \quad \to \quad A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \to \quad \textit{One eigenvalue zero; others same sign: parabolic}$$

$$D^2u = \begin{pmatrix} \partial_{tt} & 0 & 0 \\ 0 & -\partial_{xx}u & 0 \\ 0 & 0 & -\partial_{yy}u \end{pmatrix} \quad \to \quad A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \to \quad \textit{One eigenvalue has different sign: hyperbolic}$$

18.9 Classifications into linear and nonlinear PDEs

Based on the previous notation, we now provide classifications of **linear** and **nonlinear** differential equations. Simply speaking: each differential equation, which is not linear is called nonlinear. However, in the nonlinear case a further refined classification can be undertaken.

Definition 18.10 (Evans[12])**.** *Differential equations are divided into linear and nonlinear classes as follows:*

1. *A differential equation is called* ***linear*** *if it is of the form:*

$$\sum_{|\alpha|\le k} a_\alpha(x) D^\alpha u - f(x) = 0.$$

2. *A differential equation is called* ***semi-linear*** *if it is of the form:*

$$\sum_{|\alpha|=k} a_\alpha(x) D^\alpha u + a_0(D^{k-1}u, \dots, D^2u, Du, u, x) = 0.$$

   Here, nonlinearities may appear in all terms of order $|\alpha| < k$, but the highest order $|\alpha| = k$ is fully linear.

3. *A differential equation is called* ***quasi-linear*** *if it is of the form:*

$$\sum_{|\alpha|=k} a_\alpha(D^{k-1}u, \dots, D^2u, Du, u, x) D^\alpha u + a_0(D^{k-1}u, \dots, D^2u, Du, u, x) = 0.$$

   Here, full nonlinearities may appear in all terms of order $|\alpha| < k$, in the highest order $|\alpha| = k$, nonlinear terms appear up to order $|\alpha| < k$.

4. *If none of the previous cases applies, a differential equation is called (fully)* ***nonlinear****.*

Remark 18.8. *In theory as well as practice it holds: the more nonlinear and equation is, the more difficult the analysis becomes and the more challenging is the design of good numerical methods.*

18.9.1 Linear mappings

Let $\{U, \|\cdot\|_U\}$ and $\{V, \|\cdot\|_V\}$ be normed spaces over $\mathbb{R}$.

Definition 18.11 (Linear mappings)**.** *A mapping $T : U \to V$ is called* ***linear*** *or* ***linear operator*** *when*

$$T(u) = T(au_1 + bu_2) = aT(u_1) + bT(u_2),$$

for $u = au_1 + bu_2$ and for $a, b \in \mathbb{R}$.

Example 18.3. *We discuss two examples:*

1. *Let $T(u) = \Delta u$. Then:*

$$T(au_1 + bu_2) = \Delta(au_1 + bu_2) = a\Delta u_1 + b\Delta u_2 = aT(u_1) + bT(u_2).$$

   Thus, T is linear.

2. *Let $T(u) = (u \cdot \nabla)u$. Then:*

$$T(au_1 + bu_2) = ((au_1 + bu_2) \cdot \nabla)(au_1 + bu_2) \neq a(u_1 \cdot \nabla)u_1 + b(u_2 \cdot \nabla)u_2 = aT(u_1) + bT(u_2).$$

   Here, T is nonlinear.

Definition 18.12 (Linear functional)**.** *A mapping $T : U \to V$ with $V = \mathbb{R}$ is called* ***linear functional****.*

Definition 18.13. *A mapping $T : U \to V$ is called* ***continuous*** *when*

$$\lim_{n\to\infty} u_n = u \quad \Rightarrow \quad \lim_{n\to\infty} Tu_n = Tu.$$

Definition 18.14. *A linear operator $T : U \to V$ is called* ***bounded****, when the following estimate holds true:*

$$\|Tu\|_V \leq c\|u\|_U.$$

18.9.2 Examples

How can we now proof in practice, whether a PDE is linear or nonlinear? The simplest way is to go term by term and check the linearity condition from Definition 18.11.

Example 18.4. *We provide again some examples:*

1. *All differential equations from Example 18.2 are* ***linear****. Check term by term Definition 18.11!*

2. *Euler equations (fluid dynamics, special case of Navier-Stokes with zero viscosity). Here, $n = 2 + 1 + 1 = 4$ (in two spatial dimensions) in which we have 2 velocity components, 1 pressure variable, and 1 temporal variable. Let us consider the momentum part of the Euler equations:*

$$\partial_t v_f + v_f \cdot \nabla v_f + \nabla p_f = f.$$

   Here the highest order is $k = 1$ (in the temporal variable as well as the spatial variable). But in front of the spatial derivative, we multiply with the zero-order term v_f. Consequently, the Euler equations are ***quasi-linear*** *because a lower-order term of the solution variable is multiplied with the highest derivative.*

3. *Navier-Stokes momentum equation:*

$$\partial_t v_f - \rho_f \nu_f \Delta v_f + v_f \cdot \nabla v_f + \nabla p_f = f.$$

   Here $k = 2$. But the coefficients in front of the highest order term, the Laplacian, do not depend on v_f. Consequently, the Navier-Stokes equations are neither fully nonlinear nor quasi-linear. Well, we have again the nonlinearity of the first order convection term $v_f \cdot \nabla v_f$. Thus the Navier-Stokes equations are ***semi-linear****.*

4. *A fully* ***nonlinear*** *situation would be:*

$$\partial_t v_f - \rho_f \nu_f (\Delta v_f)^2 + v_f \cdot \nabla v_f + \nabla p_f = f.$$

Example 18.5 (Development of numerical methods for nonlinear equations)**.** *In case you are given a nonlinear IBVP (initial-boundary value problem) and want to start developing numerical methods for this specific PDE, it is often much easier to start with appropriate simplifications in order to build and analyze step-by-step your final method. Let us say you are given the nonlinear time-dependent PDE*

$$\nabla u \partial_t^2 u + u \cdot \nabla u - (\Delta u)^2 = f$$

Then, you could tackle the problem as follows:

1. *Consider the linear equation:*
$$\partial_t^2 u - \Delta u = f$$
which is nothing else than the wave equation.

2. *Add a slight nonlinearity to make the problem semi-linear:*
$$\partial_t^2 u + u \cdot \nabla u - \Delta u = f$$

3. *Add ∇u such that the problem becomes quasi-linear:*
$$\nabla u \partial_t^2 u + u \cdot \nabla u - \Delta u = f$$

4. *Make the problem fully nonlinear by considering $(\Delta u)^2$:*
$$\nabla u \partial_t^2 u + u \cdot \nabla u - (\Delta u)^2 = f.$$

In each step, make sure that the corresponding numerical solution makes sense and that your developments so far are correct. If yes, proceed to the next step.

Remark 18.9. *The contrary to the previous example works as well and should be kept in mind! If you have implemented the full nonlinear PDE and recognize a difficulty, you can reduce the PDE term by term and make it step by step simpler. The same procedure holds true for the mathematical analysis.*

Chapter 19

Discretization and solution of PDEs (complement)

19.1 Finite elements for PDE boundary values problems

In this section, we concentrate on boundary value problems. We change now two things at the same time:

- Considering boundary value problems (BVP) rather than initial-value problems (IVP);
- Using finite elements (FE) rather than finite differences (FD) for discretization.

Of course the other two combinations (IVP solved by FE and BVP solved by FD) are also possible. In fact FD for solving BVP have been very popular in the past and are still important because a quick implementation is possible. On the other hand finite elements have advantages for more general geometries, unstructured grids, complicated PDEs and PDE systems, and allow for a deep mathematical theory. Moreover, the finite element method can work with less smoothness of the underlying functions. A good overview has been compiled in [14]. As in the previous chapter, this part is the abbreviated version of the more extensive lecture notes from the second author [29]; an updated version including some basic python codes can be found here Wi21 (click on the link to obtain *.pdf) These lecture notes are further complemented with additional videos published on youtube:

- PDEs and their classification: `https://www.youtube.com/watch?v=afNZj5URZsc&feature=youtu.be`
- Finite differences: `https://youtu.be/YotrBNLFen0`
- Finite elements: `https://youtu.be/P4lBRuY7pC4`
- Galerkin orthogonality and Céa's lemma: `https://www.youtube.com/watch?v=uPt5PIW1Rfo`
- Lax-Milgram lemma: `https://www.youtube.com/watch?v=w8QjyVug78M`

19.1.1 Model problem: 1D Poisson

We consider the model problem (D)[1]:

$$-u'' = f \quad \text{in } \Omega = (0,1), \tag{19.1}$$

$$u(0) = u(1) = 0. \tag{19.2}$$

Please make yourself clear that this is nothing else than expressing Formulation 18.8 in 1D. Thus, we deal with a second-order elliptic problem.

[1] (D) stands for differential problem. (M) stands for minimization problem. (V) stands for variational problem.

19.1.2 Construction of an exact solution (only possible for simple cases!)

In this section, we construct again an exact solution first. In 1D[2], we can derive the explicit solution of (19.1). First we have

$$u'(\tilde{x}) = -\int_0^{\tilde{x}} f(s)ds + C_1$$

where C_1 is a positive constant. Further integration yields

$$u(x) = \int_0^x u'(\tilde{x})\,d\tilde{x} = -\int_0^x \Big(\int_0^{\tilde{x}} f(s)\,ds\Big)d\tilde{x} + C_1 x + C_2$$

with another integration constant C_2. We have two unknown constants C_1, C_2 but also two given boundary conditions which allows us to determine C_1 and C_2.

$$0 = u(0) = -\int_0^x \Big(\int_0^{\tilde{x}} f(s)\,ds\Big)d\tilde{x} + C_1 \cdot 0 + C_2$$

yields $C_2 = 0$. For C_1, using $u(1) = 0$, we calculate:

$$C_1 = \int_0^1 \Big(\int_0^{\tilde{x}} f(s)\,ds\Big)\,d\tilde{x}.$$

Thus we obtain as final solution:

$$u(x) = -\int_0^x \Big(\int_0^{\tilde{x}} f(s)\,ds\Big)d\tilde{x} + x\left(\int_0^1 \Big(\int_0^{\tilde{x}} f(s)\,ds\Big)\,d\tilde{x}\right).$$

Thus, for a given right hand side f we obtain an explicit expression. For instance $f = 1$ yields:

$$u(x) = \frac{1}{2}(-x^2 + x).$$

It is trivial to double-check that this solution also satisfies the boundary conditions: $u(0) = u(1) = 0$. Furthermore, we see by differentiation that the original equation is obtained:

$$u'(x) = -x + \frac{1}{2} \quad \Rightarrow \quad -u''(x) = 1.$$

Exercise 19.1. *Let $f = -1$.*

- *Compute C_1;*
- *Compute $u(x)$;*
- *Check that $u(x)$ satisfies the boundary conditions;*
- *Check that $u(x)$ satisfies the PDE.*

19.1.3 Equivalent formulations

We first discuss that the solution of (19.1) (D) is also the solution of a minimization problem (M) and a variational problem (V). To formulate these (equivalent) problems, we first introduce the scalar product

$$(v, w) = \int_0^1 v(x)w(x)\,dx.$$

[2] In higher dimensions, we still can construct analytical solutions for simple geometries, but comes easily challenging (impossible!) for more complicated settings and equations.

Furthermore we introduce the linear space

$$V := \{v|\ v \in C[0,1], v' \text{ is piecewise continuous and bounded on } [0,1], v(0) = v(1) = 0\}. \tag{19.3}$$

We also introduce the linear functional $F : V \to \mathbb{R}$ such that

$$F(v) = \frac{1}{2}(v', v') - (f, v).$$

We state

Definition 19.1. *We deal with three (equivalent) problems:*

(D) Find $u \in C^2$ such that $-u'' = f$ with $u(0) = u(1) = 0$;

(M) Find $u \in V$ such that $F(u) \le F(v)$ for all $v \in V$;

(V) Find $u \in V$ such that $(u', v') = (f, v)$ for all $v \in V$.

*In physics, the quantity $F(v)$ stands for the **total potential energy** of the underlying model. Moreover, the first term in $F(v)$ denotes the internal elastic energy and (f, v) the load potential. Therefore, formulation (M) corresponds to the fundamental **principle of minimal potential energy** and the variational problem (V) to the **principle of virtual work** (e.g., [6]). The proofs of their equivalence will be provided in the following.*

Remark 19.1 (Euler-Lagrange equations)**.** *When (V) is derived from (M), we also say that (V) is the **Euler-Lagrange equation** related to (M).*

In the following we show that the three problems $(D), (M), (V)$ are equivalent.

Proposition 19.1. *It holds*

$$(D) \to (V).$$

Proof. We multiply $u'' = f$ with an arbitrary function ϕ (a so-called **test function**) from the space V defined in (19.3). Then we integrate over the interval $\Omega = (0, 1)$ yielding

$$-u'' = f \tag{19.4}$$

$$\Rightarrow -\int_\Omega u''\phi\, dx = \int_\Omega f\phi\, dx \tag{19.5}$$

$$\Rightarrow \int_\Omega u'\phi'\, dx - u'(1)\phi(1) + u'(0)\phi(0) = \int_\Omega f\phi\, dx \tag{19.6}$$

$$\Rightarrow \int_\Omega u'\phi'\, dx = \int_\Omega f\phi\, dx \quad \forall\phi \in V. \tag{19.7}$$

In the second last term, we used integration by parts.

The boundary terms vanish because $\phi \in V$. This shows that

$$\int_\Omega u'\phi'\, dx = \int_\Omega f\phi\, dx$$

is a solution of (V). □

Proposition 19.2. *It holds*

$$(V) \leftrightarrow (M).$$

Proof. We first assume that u is a solution to (V). Let $\phi \in V$ and set $w = \phi - u$ such that $\phi = u + w$ and $w \in V$. We obtain

$$\begin{aligned} F(\phi) = F(u+w) &= \frac{1}{2}(u'+w', u'+w') - (f, u+w) \\ &= \frac{1}{2}(u', u') - (f, u) + (u', w') - (f, w) + \frac{1}{2}(w', w') \geq F(u) \end{aligned}$$

We use now the fact that (V) holds true, namely

$$(u', w') - (f, w) = 0.$$

and also that $(w', w') \geq 0$. Thus, we have shown that u is a solution to (M). We show now that $(M) \to (V)$ holds true as well. For any $\phi \in V$ and $\varepsilon \in \mathbb{R}$ we have

$$F(u) \leq F(u + \varepsilon\phi),$$

because $u + \varepsilon\phi \in V$. We differentiate with respect to ε and show that (V) is a first order necessary condition to (M) with a minimum at $\varepsilon = 0$. To do so, we define

$$g(\varepsilon) := F(u + \varepsilon\phi) = \frac{1}{2}(u', u') + \varepsilon(u', \phi') + \frac{\varepsilon^2}{2}(\phi', \phi') - (f, u) - \varepsilon(f, \phi).$$

Thus

$$g'(\varepsilon) = (u', \phi') + \varepsilon(\phi', \phi') - (f, \phi).$$

A minimum is obtained for $\varepsilon = 0$. Consequently,

$$g'(0) = 0.$$

In detail:

$$(u', \phi') - (f, \phi) = 0,$$

which is nothing else than the solution of (V). □

Proposition 19.3. *The solution u to (V) is unique.*

Proof. Assume that u_1 and u_2 are solutions to (V) with

$$\begin{aligned} (u_1', \phi') &= (f, \phi) \quad \forall \phi \in V, \\ (u_2', \phi') &= (f, \phi) \quad \forall \phi \in V. \end{aligned}$$

We subtract these solutions and obtain:

$$(u_1' - u_2', \phi') = 0.$$

We choose as test function $\phi' = u_1' - u_2'$ and obtain

$$(u_1' - u_2', u_1' - u_2') = 0.$$

Thus:

$$u_1'(x) - u_2'(x) = (u_1 - u_2)'(x) = 0.$$

Since the difference of the derivatives is zero, this means that the difference of the functions themselves is constant:

$$(u_1 - u_2)(x) = const$$

Using the boundary conditions $u(0) = u(1) = 0$, yields

$$(u_1 - u_2)(x) = 0.$$

Thus $u_1 = u_2$. □

Remark 19.2. *The last statements are related to the definiteness of the norm. It holds:*

$$\|u\|_{L^2}^2 = \int_\Omega u^2 \, dx.$$

Thus the results follow directly because

$$\|u\| = 0 \quad \Leftrightarrow \quad u = 0.$$

Proposition 19.4. *It holds*

$$(V) \to (D).$$

Proof. We assume that u is a solution to (V), i.e.,

$$(u', \phi') = (f, \phi) \quad \forall \phi \in V.$$

If we assume sufficient regularity of u (in particular $u \in C^2$), then u'' exists and we can integrate backwards. Moreover, we use that $\phi(0) = \phi(1) = 0$ since $\phi \in V$. Then:

$$(-u'' - f, \phi) = 0 \quad \forall \phi \in V.$$

Since we assumed sufficient regularity for u'' and f the difference is continuous. We can now apply the fundamental principle (see Proposition 19.5):

$$w \in C(\Omega) \quad \Rightarrow \quad \int_\Omega w\phi \, dx = 0 \quad \Rightarrow \quad w \equiv 0.$$

We proof this result later. Before, we obtain

$$(-u'' - f, \phi) = 0 \quad \Rightarrow \quad -u'' - f = 0,$$

which yields the desired expression. Since we know that $(D) \to (V)$ holds true, u has the assumed regularity properties and we have shown the equivalence. □

It remains to state and proof the fundamental lemma of calculus of variations. To this end, we introduce

Definition 19.2 (Continuous functions with compact support)**.** *Let $\Omega \subset \mathbb{R}^n$ be an open domain.*

- *The set of continuous functions from Ω to $\mathbb{R}$ with **compact support** is denoted by $C_c(\Omega)$. Such functions vanish on the boundary.*
- *The set of **smooth functions** (infinitely continuously differentiable) with compact support is denoted by $C_c^\infty(\Omega)$.*

Proposition 19.5 (Fundamental lemma of calculus of variations)**.** *Let $\Omega = [a, b]$ be a compact interval and let $w \in C(\Omega)$. Let $\phi \in C^\infty$ with $\phi(a) = \phi(b) = 0$, i.e., $\phi \in C_c^\infty(\Omega)$. If for all ϕ it holds*

$$\int_\Omega w(x)\phi(x) \, dx = 0,$$

then, $w \equiv 0$ in Ω.

Proof. We perform an indirect proof. We suppose that there exist a point $x_0 \in \Omega$ with $w(x_0) \neq 0$. Without loss of generality, we can assume $w(x_0) > 0$. Since w is continuous, there exists a small (open) neighborhood $\omega \subset \Omega$ with $w(x) > 0$ for all $x \in \omega$; otherwise $w \equiv 0$ in $\Omega \setminus \omega$. Let ϕ now be a positive test function (recall that ϕ can be arbitrary, specifically positive if we wish) in Ω and thus also in ω. Then:

$$\int_\Omega w(x)\phi(x) \, dx = \int_\omega w(x)\phi(x) \, dx.$$

But this is a contradiction to the hypothesis on w. Thus $w(x) = 0$ for all in $x \in \omega$. Extending this result to all open neighborhoods in Ω we arrive at the final result. □

19.1.4 The weak form: defining a bilinear form and a linear form

We continue to consider the variational form in this section and provide more definitions and notation.

We recall the key idea:

- design an appropriate function space,
- multiply the strong form (D) with a test function,
- integrate,
- apply integration by parts on second-order terms.

The last operation 'weakens' the derivative information because rather evaluating 2nd order derivatives, we only need to evaluate a 1st order derivative on the trial function and another 1st order derivative on the test function.

We recall the procedure how to obtain a weak form:

$$
\begin{aligned}
-u'' &= f && (19.8)\\
\Rightarrow -\int_\Omega u''\phi\,dx &= \int_\Omega f\phi\,dx && (19.9)\\
\Rightarrow \int_\Omega u'\phi'\,dx - \int_{\partial\Omega} \partial_n u\phi\,ds &= \int_\Omega f\phi\,dx && (19.10)\\
\Rightarrow \int_\Omega u'\phi'\,dx &= \int_\Omega f\phi\,dx. && (19.11)
\end{aligned}
$$

To summarize we have:

$$\int_\Omega u'\phi'\,dx = \int_\Omega f\phi\,dx \tag{19.12}$$

A common short-hand notation in mathematics is to use parentheses for L^2 scalar products: $\int_\Omega ab\,dx =: (a,b)$:

$$(u',\phi') = (f,\phi) \tag{19.13}$$

A mathematically-correct statement is:

Formulation 19.6. *Find $u \in V$ such that*

$$(u',\phi') = (f,\phi) \quad \forall\phi \in V. \tag{19.14}$$

In the following, a very common notation is introduced. First, we recall concepts known from linear algebra:

Definition 19.3 (Linear form and bilinear form)**.** *If V is a linear space, l is a **linear form** on V if $l : V \to \mathbb{R}$ or in other words $l(v) \in \mathbb{R}$ for $v \in V$. Moreover, l is linear:*

$$l(av + bw) = al(v) + bl(w) \quad \forall a,b \in R, \quad \forall v,w \in V.$$

*A formulation is a **bilinear form** [3] on $V \times V$ if $a : V \times V \to \mathbb{R}$ and $a(v,w) \in \mathbb{R}$ for $v,w \in V$ and $a(v,w)$ is linear in each argument, i.e., for $\alpha,\beta \in R$ and $u,v,w \in V$, it holds*

$$
\begin{aligned}
a(u,\alpha v + \beta w) &= \alpha a(u,v) + \beta a(u,w),\\
a(\alpha u + \beta v, w) &= \alpha a(u,w) + \beta a(v,w).
\end{aligned}
$$

[3] For nonlinear problems we deal with a semi-linear form, which is only linear in one argument.

A bilinear form is said to be ***symmetric*** *if*

$$a(u,v) = a(v,u).$$

A symmetric bilinear form $a(\cdot,\cdot)$ on $V \times V$ defines a ***scalar product*** *on V if*

$$a(v,v) > 0 \quad \forall v \in V, v \neq 0.$$

The associated ***norm*** *is denoted by $\|\cdot\|_a$ and defined by*

$$\|v\|_a := \sqrt{a(v,v)} \quad \text{for } v \in V.$$

Definition 19.4 (Frobenius scalar product)**.** *For vector-valued functions in second-order problems, we need to work with a scalar product defined for matrices because their gradients are matrices. Here the natural form is the Frobenius scalar product, which is defined as:*

$$< A, B >_F = A : B = \sum_i \sum_j a_{ij} b_{ij}.$$

The Frobenius scalar product induces the Frobenius norm:

$$\|A\|_F = \sqrt{< A, B >}.$$

Remark 19.3. *For vector-valued PDEs (such as elasticity, Stokes or Navier-Stokes), we formulate in compact form:*

$$a(u,\phi) = l(\phi)$$

with some $l(\phi)$ and

$$a(u,\phi) = \int_\Omega \nabla u : \nabla \phi \, dx$$

where $\nabla u : \nabla \phi$ is then evaluated in terms of the Frobenius scalar product.

Definition 19.5 (Bilinear and linear forms of the continuous Poisson problem)**.** *For the Poisson problem, we can define:*

$$\begin{aligned} a(u,\phi) &= (u', \phi'), \\ l(\phi) &= (f, \phi). \end{aligned}$$

We shall state the variational form of Poisson's problem in terms of $a(\cdot,\cdot)$ and $l(\cdot)$:

Formulation 19.7 (Variational Poisson problem on the continuous level)**.** *Find $u \in V$ such that*

$$a(u,\phi) = l(\phi) \quad \forall \phi \in V, \tag{19.15}$$

where $a(\cdot,\cdot)$ and $l(\cdot)$ are defined in Definition 19.5. The unknown function u is called the ***trial*** *or (****ansatz****) function whereas ϕ is the so-called* ***test function****.*

19.1.5 The Lax-Milgram lemma (complement)

We present in this section a result that ensures well-posedness (existence, uniqueness, stability) of linear variational problems.

Formulation 19.8 (Abstract model problem)**.** *Let V be a Hilbert space with norm $||\cdot||_V$. Find $u \in V$ such that*

$$a(u,\phi) = l(\phi) \quad \forall \phi \in V.$$

Definition 19.6 (Assumptions)**.** *We suppose:*

1. *$l(\cdot)$ is a bounded linear form:*
$$|l(u)| \leq C\|u\| \quad \text{for all } u \in V.$$

2. *$a(\cdot,\cdot)$ is a bilinear form on $V \times V$ and continuous:*
$$|a(u,v)| \leq \gamma \|u\|_V \|v\|_V, \quad \gamma > 0, \quad \forall u, v \in V.$$

3. *$a(\cdot,\cdot)$ is coercive (or V-elliptic):*
$$a(u,u) \geq \alpha \|u\|_V^2, \quad \alpha > 0, \quad \forall u \in V.$$

Lemma 19.9 (Lax-Milgram). *Let $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ be a continuous, V-elliptic bilinear form. Then, for each $l \in V^*$ the variational problem*
$$a(u,\phi) = l(\phi) \quad \forall \phi \in V$$
has a unique solution $u \in V$. Moreover, we have the stability estimate:
$$\|u\| \leq \frac{1}{\alpha}\|l\|_{V^*}.$$
with
$$\|l\|_{V^*} := \sup_{\varphi \neq 0} \frac{|l(\varphi)|}{||\varphi||_V}.$$

Proof. We recall upfront from before: Riesz: for all $l \in V^*$ there exists $u \in V$ such that
$$(u,\varphi) = \langle l, \varphi \rangle \quad \forall \varphi \in V.$$

We work as follows.
Step 1 (Existence of an operator A)
For all fixed $u \in V$ the mapping $v \mapsto a(u,v)$ is a bounded linear functional on V. Then with Riesz: there exist $Au \in V$ and $f \in V$ such that
$$a(u,\varphi) = (Au,\varphi) \quad \forall \varphi \in V \tag{19.16}$$
$$l(\varphi) = (f,\varphi) \quad \forall \varphi \in V. \tag{19.17}$$

Step 2 (Linearity and boundedness of A)
The mapping $u \mapsto Au$ yields a linear bounded operator $A : V \to V$. It holds for the linearity:
$$(A(\lambda_1 u_1 + \lambda_2 u_2), \varphi)_V = a(\lambda_1 u_1 + \lambda_2 u_2, \varphi) = \ldots = (\lambda_1 A u_1 + \lambda_2 A u_2, \varphi)_V$$

This holds for all $\varphi \in V$ and consequently A is linear. The boundedness (i.e., continuity) we obtain as follows:
$$\|Au\|_V^2 = (Au, Au) = a(u, Au) \leq \gamma \|u\| \|Au\|$$
$$\Rightarrow \|Au\|_V \leq \gamma \|u\| \quad \forall u \in V,$$

showing that A is bounded. For such operators in FA, we the notation $A \in L(V,V)$ is used. To sum-up
$$a(u,\varphi) = l(\varphi) \quad \text{(Variational form)} \tag{19.18}$$
$$Au = f \quad \text{(Operator equation)} \tag{19.19}$$

Step 3 (Existence of a fixed point)
We show that a fixed point exists. Specifically, we establish that
$$v \in V \mapsto T_\delta v := v - \delta(Av - f) \in V$$

with $\delta > 0$, is a contraction on V. Then, it holds

$$T_\delta v = v$$

has a unique solution $u \in V$. This solution is then also solution of (19.18) and (19.19) because of

$$\begin{aligned} T_\delta v - v &= 0 \\ \Leftrightarrow \delta(Av - f) &= 0 \\ \Leftrightarrow Av - f &= 0 \\ \Leftrightarrow Av &= f. \end{aligned}$$

The contradiction now follows from

$$v - \delta(Av - f) = v - \delta Av - \delta f$$

The right term is constant with respect to v and does therefore not play a role in the contraction proof (recall Banach's fixed point theorem in Numerik 1 for iterative schemes for solving $Ax = b$). Then:

$$\|v - \delta Av\|_V^2 = \|v\|^2 - 2\delta a(v, v) + \delta^2 \|Av\|_V^2$$

We now use the coercivity of $a(v, v)$. Then:

$$\|v - \delta Av\|_V^2 \le \|v\|^2 - 2\delta\alpha\|v\|_V^2 + \delta^2\gamma^2\|v\|_V^2 = (1 - 2\delta\alpha + \delta^2\gamma^2)\|v\|_V^2$$

For the choice of

$$0 < \delta < \frac{2\alpha}{\gamma^2}$$

we have a contraction and thus a fixed point. Due to the equivalences of the formulations in this proof (weak form, operator form, fixed point form), we have obtained a unique solution of

$$u \in V : a(u, \varphi) = l(\varphi) \quad \varphi \in V.$$

Step 4
Uniqueness and the stability are the same as in the previous first proof of Lax-Milgram's lemma. □

19.1.6 Finite elements in 1D

In the following we want to concentrate how to compute a discrete solution. As in the previous chapter, this, in principle, allows us to address even more complicated situations and also higher spatial dimensions. The principle of the FEM is rather simple:

- Introduce a mesh $\mathcal{T}_h := \bigcup K_i$ (where K_i denote the single mesh elements) of the given domain $\Omega = (0, 1)$ with mesh size (diameter/length) parameter h
- Define on each mesh element $K_i := [x_i, x_{i+1}], i = 0, \dots, n$ polynomials for trial and test functions. These polynomials must form a basis in a space V_h and they should reflect certain conditions on the mesh edges;
- Use the variational (or weak) form of the given problem and derive a discrete version;
- Evaluate the arising integrals;
- Collect all contributions on all K_i leading to a linear equation system $AU = B$;
- Solve this linear equation system; the solution vector $U = (u_0, \dots, u_n)^T$ contains the discrete solution at the nodal points $x_0, \dots, x_n$;
- Verify the correctness of the solution U.

The mesh

Let us start with the mesh. We introduce nodal points and divide $\Omega = (0,1)$ into

$$x_0 = 0 < x_1 < x_2 < \ldots < x_n < x_{n+1} = 1.$$

In particular, we can work with a uniform mesh in which all nodal points have equidistant distance:

$$x_j = jh, \quad h = \frac{1}{n+1}, \quad 0 \le j \le n+1, \quad h = x_{j+1} - x_j.$$

Remark 19.4. *An important research topic is to organize the points x_j in certain non-uniform ways in order to reduce the discrete error.*

Linear finite elements

In the following we denote P_k the space that contains all polynomials up to order k with coefficients in $\mathbb{R}$:

Definition 19.7.

$$P_k := \{\sum_{i=0}^{k} a_i x^i |\ a_i \in \mathbb{R}\}.$$

In particular we will work with the space of linear polynomials

$$P_1 := \{a_0 + a_1 x |\ a_0, a_1 \in \mathbb{R}\}.$$

A finite element is now a function localized to an element $K_i \in \mathcal{T}_h$ and uniquely defined by the values in the nodal points x_i, x_{i+1}.

We then define the space:

$$V_h^{(1)} = V_h := \{v \in C[0,1] |\ v|_{K_i} \in P_1, K_i := [x_i, x_{i+1}], 0 \le i \le n, v(0) = v(1) = 0\}.$$

The boundary conditions are build into the space through $v(0) = v(1) = 0$. This is an important concept that Dirichlet boundary conditions will not appear explicitly later, but are contained in the function spaces.

All functions inside V_h are so called **shape functions** and can be represented by so-called **hat functions**. Hat functions are specifically linear functions on each element K_i. Attaching them yields a hat in the geometrical sense.

For $j = 1, \ldots, n$ we define:

$$\phi_j(x) = \begin{cases} 0 & \text{if } x \notin [x_{j-1}, x_{j+1}] \\ \frac{x - x_{j-1}}{x_j - x_{j-1}} & \text{if } x \in [x_{j-1}, x_j] \\ \frac{x_{j+1} - x}{x_{j+1} - x_j} & \text{if } x \in [x_j, x_{j+1}] \end{cases} \tag{19.20}$$

with the property

$$\phi_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{19.21}$$

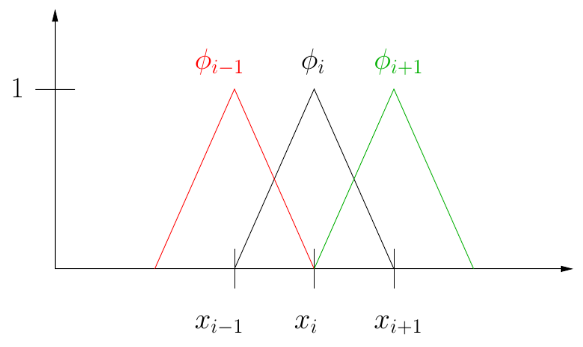


Figure 19.1: Hat functions.

For a uniform step size $h = x_j - x_{j-1} = x_{j+1} - x_j$ we obtain

$$\phi_j(x) = \begin{cases} 0 & \text{if } x \notin [x_{j-1}, x_{j+1}] \\ \frac{x - x_{j-1}}{h} & \text{if } x \in [x_{j-1}, x_j] \\ \frac{x_{j+1} - x}{h} & \text{if } x \in [x_j, x_{j+1}] \end{cases}$$

and for its derivative:

$$\phi_j'(x) = \begin{cases} 0 & \text{if } x \notin [x_{j-1}, x_{j+1}] \\ +\frac{1}{h} & \text{if } x \in [x_{j-1}, x_j] \\ -\frac{1}{h} & \text{if } x \in [x_j, x_{j+1}] \end{cases}$$

Lemma 19.10. *The space V_h is a subspace of $V := C[0,1]$ and has dimension n (because we deal with n basis functions). Thus the such constructed finite element method is a* ***conforming*** *method. Furthermore, for each function $v_h \in V_h$ we have a unique representation:*

$$v_h(x) = \sum_{j=1}^{n} v_{h,j}\phi_j(x) \quad \forall x \in [0,1], \quad v_{h,j} \in \mathbb{R}.$$

Proof. Sketch: The unique representation is clear, because in the nodal points it holds $\phi_j(x_i) = \delta_{ij}$, where δ_{ij} is the Kronecker symbol with $\delta_{ij} = 1$ for $i = j$ and 0 otherwise. □

The function $v_h(x)$ connects the discrete values $v_{h,j} \in \mathbb{R}$ and in particular the values between two support points x_j and x_{j+1} can be evaluated.

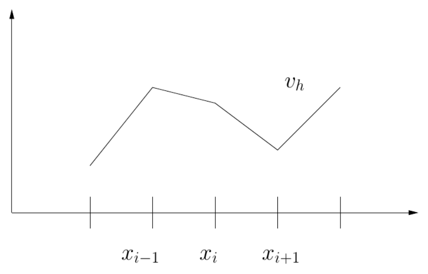


Figure 19.2: The function $v_h \in V_h$.

Remark 19.5. *The finite element method introduced above is a Lagrange method, since the basis functions ϕ_j are defined only through its values at the nodal points without using derivative information (which would result in Hermite polynomials).*

The process to construct the specific form of the shape functions

In the previous construction, we have hidden the process how to find the specific form of $\phi_j(x)$. For 1D it is more or less clear and we would accept the $\phi_j(x)$ really has the form as it has in (19.20) . In $\mathbb{R}^n$ this task is a bit of work. To understand this procedure, we explain the process in detail. Here we first address the defining properties of a finite element (see also Section 19.1.6):

- Intervals $[x_i, x_{i+1}]$;
- A linear polynomial $\phi(x) = a_0 + a_1 x$;
- Nodal values at x_i and x_{i+1} (the so-called degrees of freedom).

Later only these properties are stated and one has to construct the specific $\phi(x)$ such as for example in (19.20). Thus, the main task consists in finding the unknown coefficients a_0 and a_1 of the shape function. The key property is (19.21) (also valid in $\mathbb{R}^n$ in order to have a small support) and therefore we obtain:

$$\begin{aligned}\phi_j(x_j) &= a_0 + a_1 x_j = 1,\\ \phi_j(x_i) &= a_0 + a_1 x_i = 0.\end{aligned}$$

To determine a_0 and a_1 we have to solve a small linear equation system:

$$\begin{pmatrix} 1 & x_j \\ 1 & x_i \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

We obtain

$$a_1 = -\frac{1}{x_i - x_j}$$

and

$$a_0 = \frac{x_i}{x_i - x_j}.$$

Then:

$$\phi_j(x) = a_0 + a_1 x = \frac{x_i - x}{x_i - x_j}.$$

At this stage we have now to distinguish whether $x_j := x_{i-1}$ or $x_j := x_{i+1}$ or $|i - j| > 1$ yielding the three cases in (19.20).

Remark 19.6. *Of course, for higher-order polynomials and higher-order problems in $\mathbb{R}^n$, the matrix system to determining the coefficients becomes larger. However, in all these state-of-the-art FEM software packages, the shape functions are already implemented.*

The discrete weak form

We have set-up a mesh and local polynomial functions with a unique representation, all these developments result in a 'finite element'. Now, we use the variational formulation and derive the discrete counterpart:

Formulation 19.11. *Find $u_h \in V_h$ such that*

$$(u_h', \phi_h') = (f, \phi_h) \quad \forall \phi_h \in V_h. \tag{19.22}$$

Or in the previously introduced compact form:

Formulation 19.12 (Variational Poisson problem on the discrete level)**.** *Find $u_h \in V_h$ such that*

$$a(u_h, \phi_h) = l(\phi_h) \quad \forall \phi_h \in V_h, \tag{19.23}$$

where $a(\cdot,\cdot)$ and $l(\cdot)$ are defined in Definition 19.5 by adding h as subindex for u and ϕ.

Remark 19.7 (Galerkin method)**.** *The process going from V to V_h using the variational formulation is called* ***Galerkin method****. Here, it is not necessary that the bilinear form is symmetric. As further information: not only is Galerkin's method a numerical procedure, but it is also used in analysis when establishing existence of the continuous problem. Here, one starts with a finite dimensional subspace and constructs a sequence of finite dimensional subspaces $V_h \subset V$ (namely passing with $h \to 0$; that is to say: we add more and more basis functions such that $dim(V_h) \to \infty$). The idea of numerics is the same: finally we are interested in small h such that we obtain a discrete solution with sufficient accuracy.*

Remark 19.8 (Ritz method)**.** *If we discretize the minimization problem (M), the above process is called* ***Ritz method****. In particular, the bilinear form of the variational problem is symmetric.*

Remark 19.9 (Ritz-Galerkin method)**.** *For general bilinear forms (i.e., not necessarily symmetric) the discretization procedure is called* ***Ritz-Galerkin method****.*

Remark 19.10 (Petrov-Galerkin method)**.** *In a* ***Petrov-Galerkin method*** *the trial and test spaces can be different.*

In order to proceed we can express $u_h \in V_h$ with the help of the basis functions ϕ_j in $V_h := \{\phi_1, \ldots, \phi_n\}$, thus:

$$u_h = \sum_{j=1}^{n} u_j \phi_j(x), \quad u_j \in \mathbb{R}.$$

Since (19.22) holds for all $\phi_i \in V_h$ for $1 \le i \le n$, it holds in particular for each i:

$$(u_h', \phi_i') = (f, \phi_i) \quad \text{for } 1 \le i \le n. \tag{19.24}$$

We now insert the representation for u_h in (19.24), yielding the **Galerkin equations**:

$$\sum_{j=1}^{n} u_j (\phi_j', \phi_i') = (f, \phi_i) \quad \text{for } 1 \le i \le n. \tag{19.25}$$

We have now extracted the coefficient vector of u_h and only the **shape functions** ϕ_j and ϕ_i (i.e., their derivatives of course) remain in the integral.

This yields a linear equation system of the form

$$AU = B$$

where

$$U = (u_j)_{1 \le j \le n} \in \mathbb{R}^n, \tag{19.26}$$
$$B = \big((f, \phi_i)\big)_{1 \le i \le n} \in \mathbb{R}^n, \tag{19.27}$$
$$A = \Big((\phi_j', \phi_i')\Big)_{1 \le j, i \le n} \in \mathbb{R}^{n \times n}. \tag{19.28}$$

Thus the final solution vector is U which contains the values u_j at the nodal points x_j of the mesh. Here we remark that x_0 and x_{n+1} are not solved in the above system and are determined by the boundary conditions $u(x_0) = u(0) = 0$ and $u(x_{n+1}) = u(1) = 0$.

Evaluation of the integrals

It remains to determine the specific entries of the system matrix (also called stiffness matrix) A and the right hand side vector B. Since the basis functions have only little support on two neighboring elements (in fact that is one of the key features of the FEM) the resulting matrix A is sparse, i.e., it contains only a few number of entries that are not equal to zero.

Let us now evaluate the integrals that form the entries a_{ij} of the matrix $A = (a_{ij})_{1\leq i,j\leq n}$. For the diagonal elements we calculate:

$$\begin{aligned} a_{ii} &= \int_\Omega \varphi_i'(x)\varphi_i'(x)\,dx = \int_{x_{i-1}}^{x_{i+1}} \varphi_i'(x)\varphi_i'(x)\,dx \\ &= \int_{x_{i-1}}^{x_i} \frac{1}{h^2}\,dx + \int_{x_i}^{x_{i+1}} \Big(-\frac{1}{h}\Big)^2\,dx \\ &= \frac{h}{h^2} + \frac{h}{h^2} \\ &= \frac{2}{h}. \end{aligned}$$

For the right off-diagonal we have:

$$a_{i,i+1} = \int_\Omega \varphi_{i+1}'(x)\varphi_i'(x)\,dx = \int_{x_i}^{x_{i+1}} \frac{1}{h}\cdot\Big(-\frac{1}{h}\Big)\,dx = -\frac{1}{h}.$$

It is trivial to see that $a_{i,i+1} = a_{i-1,i}$.

In compact form we summarize:

$$a_{ij} = \int_\Omega \phi_j'(x)\phi_i'(x)\,dx = \begin{cases} -\frac{1}{x_{j+1}-x_j} & \text{if j=i-1} \\ \frac{1}{x_j-x_{j-1}} + \frac{1}{x_{j+1}-x_j} & \text{if j=i} \\ -\frac{1}{x_j-x_{j-1}} & \text{if j=i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{19.29}$$

For a uniform mesh (as we assume in this section) we can simplify the previous calculation since we know that $h = h_j = x_{j+1} - x_j$:

$$a_{ij} = \int_\Omega \phi_j'(x)\phi_i'(x)\,dx = \begin{cases} -\frac{1}{h} & \text{if j=i-1} \\ \frac{2}{h} & \text{if j=i} \\ -\frac{1}{h} & \text{if j=i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{19.30}$$

The resulting matrix A for the 'inner' points $x_1, \ldots, x_n$ reads then:

$$A = h^{-1}\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Remark 19.11. *Since the boundary values at $x_0 = 0$ and $x_{n+1} = 1$ are known to be $u(0) = u(1) = 0$, they are not assembled in the matrix A. We could have considered all support points $x_0, x_1, \ldots, x_n, x_{n+1}$ we would have obtained:*

$$A = h^{-1}\begin{pmatrix} 1 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{(n+2)\times(n+2)}.$$

Here the entries $a_{00} = a_{n+1,n+1} = 1$ (and not 2) because at the boundary only the half of the two corresponding test functions do exist. Furthermore, working with this matrix A in the solution process below, we have to modify the entries $a_{0,1} = a_{1,0} = a_{n,n+1} = a_{n+1,n}$ from the value -1 to 0 such that $u_{h,0} = u_{h,n+1} = 0$ can follow.

It remains to evaluate the integrals of the right hand side vector b. Here the main difficulty is that the given right hand side f may be complicated. Of course it always holds for the entries of b:

$$b_i = \int_\Omega f(x)\phi_i(x)\,dx \quad \text{for } 1 \le i \le n.$$

The right hand side now depends explicitly on the values for f. Let us assume a constant $f = 1$ (thus the original problem would be $-u'' = 1$), then:

$$b_i = \int_\Omega 1 \cdot \phi_i(x)\,dx = 1 \cdot \int_\Omega \phi_i(x)\,dx = 1 \cdot h \quad \text{for } 1 \le i \le n.$$

Namely

$$B = (h, \dots, h)^T.$$

For non-uniform step sizes we obtain:

$$\frac{1}{h_i}\int_{K_i} \phi_i(x)\,dx, \quad \text{and} \quad \frac{1}{h_{i+1}}\int_{K_{i+i}} \phi_i(x)\,dx$$

with $h_i = x_i - x_{i-1}, h_{i+1} = x_{i+1} - x_i$ and $K_i = [x_{i-1}, x_i]$ and $K_{i+1} = [x_i, x_{i+1}]$.

Exercise 19.2. *Derive the system matrix A by assuming non-uniform step sizes $h_j = x_j - x_{j+1}$.*

Exercise 19.3. *Derive the system matrix A for a P_2 discretization, namely working with quadratic basis functions. Sketch how the basis functions look like.*

Definition of a finite element

We briefly summarize the key ingredients that define a **finite element**. A finite element is a triple (K, P_K, Σ) where

- K is an element, i.e., a geometric object (in 1D an interval);
- $P_k(K)$ is a finite dimensional linear space of polynomials defined on K;
- Σ, not introduced so far, is a set of degrees of freedom (DoF), e.g., the values of the polynomial at the vertices of K.

These three ingredients yield a uniquely determined polynomial on an element K.

Properties of the system matrix A

We first notice:

Remark 19.12. *The system matrix A has a factor $\frac{1}{h}$ whereas in the corresponding matrix A, using a finite difference scheme, we have a factor $\frac{1}{h^2}$. The reason is that we integrate the weak form using finite elements and one h is hidden in the right hand side vector b. Division by h we yield the same system for finite elements as for finite differences.*

Next, we show that A is symmetric and positive definite. It holds

$$(\phi_i', \phi_j') = (\phi_j', \phi_i').$$

Using the representation

$$u(x) = \sum_{j=1}^n u_{j,h}\phi_{j,h}(x),$$

we obtain

$$\sum_{i,j=1}^{n} u_i(\phi_i', \phi_j')u_j = (\sum_{i,j=1}^{n} u_i\phi_i', \sum_{i,j=1}^{n} u_j\phi_j') = (u', u') \geq 0.$$

We have

$$(u', u') = 0$$

only if $u' \equiv 0$ since $u(0) = 0$ only for $u \equiv 0$ or $u_{j,h} = 0$ for $j = 1, \ldots, n$. We recall that a symmetric matrix $B \in \mathbb{R}^{n\times n}$ is said to be positive definite if

$$\xi \cdot B\xi = \sum_{i,j=1}^{n} \xi_i b_{ij} \xi_j > 0 \quad \forall \xi \in \mathbb{R}^n, \xi \neq 0.$$

Finally, it holds:

Proposition 19.13. *A symmetric positive matrix B is positive definite if and only if the eigenvalues of B are strictly positive. Furthermore, a positive definite matrix is regular and consequently, the linear system to which B is associated with has a unique solution.*

Proof. Results from linear algebra. □

Numerical test: 1D Poisson

Implementation in octave We design a numerical test implemented in octave [11].

Formulation 19.14. *Let $\Omega = (0, 1)$*

$$\begin{cases} \textit{Find} \quad u \in \mathcal{C}^2(\Omega) \quad \textit{such that} \\ -u''(x) = f \quad \textit{in } \Omega \\ u(0) = 0, \\ u(1) = 0. \end{cases} \tag{19.31}$$

where $f = -1$. To discretize the problem, we work with linear finite elements. To this end, we introduce the function space V_h that consists of piece-wise linear polynomials and also contains the Dirichlet boundary conditions.

The variational form then reads:

$$\int_0^1 u'(x)\varphi'(x)\,dx = \int_0^1 f\varphi(x)\,dx.$$

The evaluation of these integrals has been performed in the previous sections of this chapter.

We now choose $n = 4$. Thus we have five support points

$$x_0, x_1, x_2, x_3, x_4$$

and $h = 1/4 = 0.25$.

With these settings we obtain for the matrix A:

```
16     0     0     0     0
0     32   -16     0     0
0    -16    32   -16     0
0      0   -16    32     0
0      0     0     0    16
```

and for the right hand side vector b:

```
0
-1
-1
-1
0
```

We use the famous backslash solution operator (in fact an LU decomposition):

```
U = A\b
```

and obtain as solution $U = (U_0, U_1, U_2, U_3, U_4)$:

```
 0.00000
-0.09375
-0.12500
-0.09375
 0.00000
```

A plot of the discrete solution is provided in Figure 19.3.

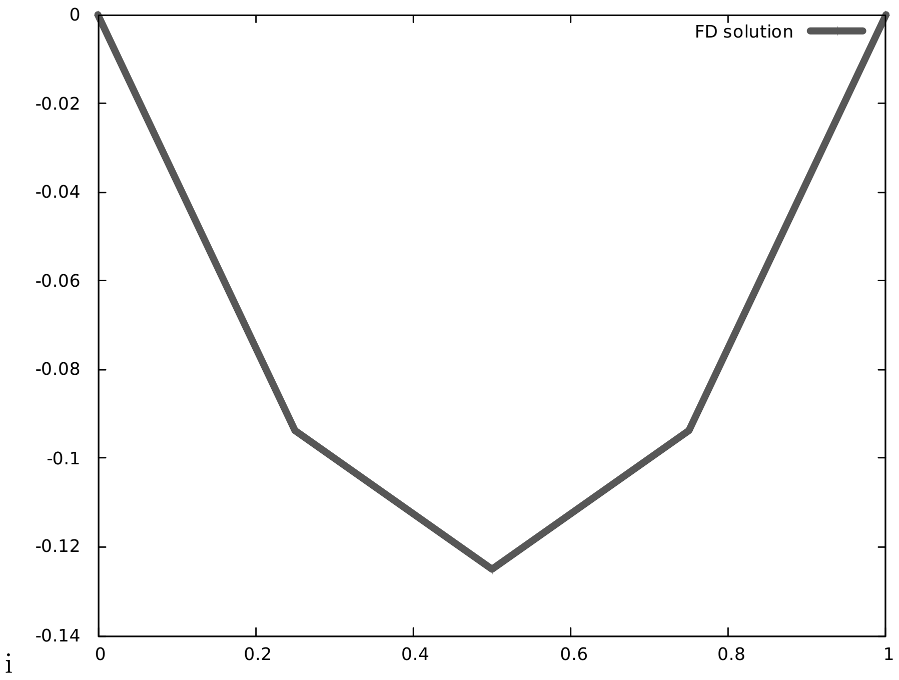


Figure 19.3: Solution of the 1D Poisson problem with $f = -1$ using five support points with $h = 0.25$. We observe that the approximation is rather coarse. A smaller h would yield more support points and more solution values and therefore a more accurate solution.

Implementation in python The goal of the computation is to recover the exact solution with respect to the chosen parameters and domain specifications:

$$u(x) = -\frac{1}{2}(-x^2 + x) \quad \text{on } \Omega.$$

It is trivial to check that this exact solution satisfies the PDE and also the boundary conditions.

As numerical results we obtain the solution curve displayed in Figure 19.1.6. Visually we observe excellent agreement between the numerical and the exact solution.

To quantify this statement we could compute the error in certain norms. The most common for this setting are the L^2 norm (measuring the solution itself) and the H^1 norm (measuring the gradients of the solution):

$$\|u - u_h\|_{L^2} \to 0? \quad \text{for } h \to 0, \|u - u_h\|_{H^1} \to 0? \quad \text{for } h \to 0. \tag{19.32}$$

These findings are shown in the next subsection.

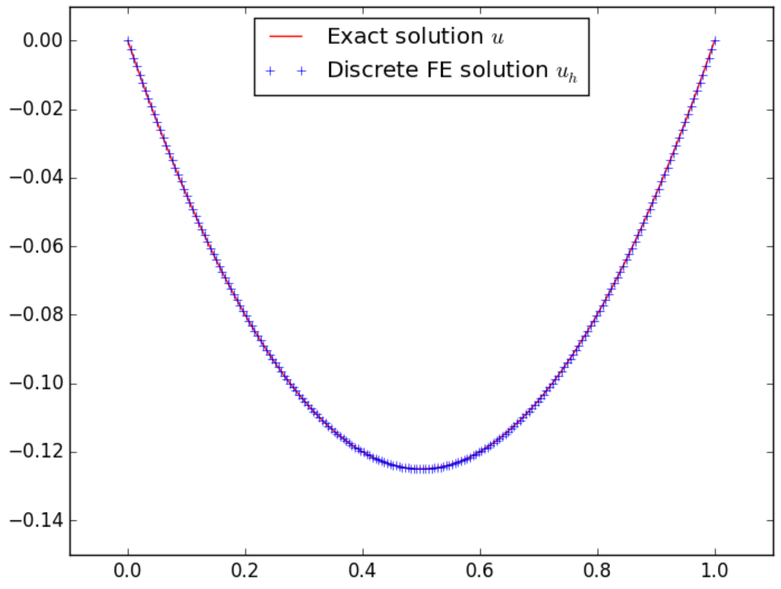


Figure 19.4: Visual comparison between the exact and the manufactured solution. Please do not forget: the 'Picture norm' (thus comparing two graphs as we do here) is not a rigorous mathematical justification that a result is correct. It is rather a strong indication about the solution behavior and very helpful to get an idea about the correctness. The mathematical justification would be to compare the two solutions in a certain error measure (i.e., a norm) as we have done in Section 17.6.

Implementation in deal.II (C++) Thirdly, we adopt deal.II [1, 2] an open source C++ finite element code to carry out this computation. Even that in such packages many things are hidden and performed in the background, deal.II leaves enough room to **see** the important points:

- Creating a domain Ω and decomposition of this domain into elements;
- Setting up the vectors u, b and the matrix A with their correct length;
- Assembling (not yet solving!) the system $Au = b$. We compute locally on each mesh element the matrix entries and right hand side entries on a master cell and insert the respective values at their global places in the matrix A;
- In the assembly, the integrals are evaluated using numerical quadrature in order to allow for more general expressions when for instance the material parameter a is not constant to 1 or when higher-order polynomials need to be evaluated;
- Solution of the linear system $Au = b$ (the implementation of the specific method is hidden though;
- Postprocessing of the solution: here writing the solution data $U_j, 1 \le j \le n$ into a file that can be read from a graphic visualization program such as gnuplot, paraview, visit.

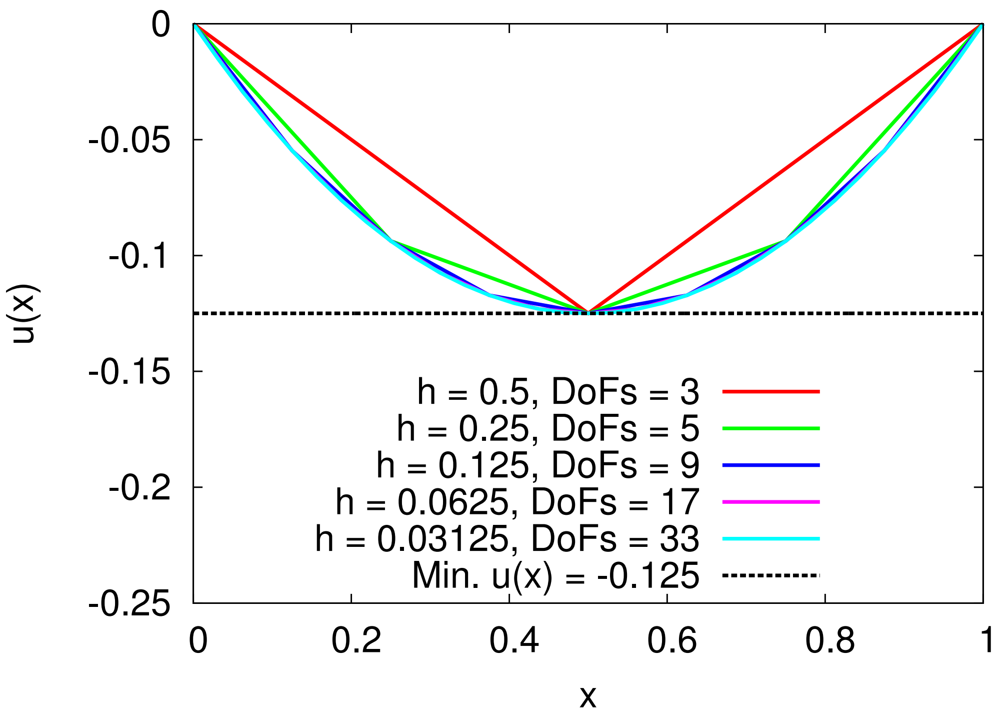


Figure 19.5: Solution of the 1D Poisson problem with $f = -1$ using finite elements with various mesh sizes h. DoFs is the abbreviation for degrees of freedom; here the number of support points x_j. The dimension of the discrete space is $DoFs$. For instance for $h = 0.5$, we have 3 DoFs and two basis functions, thus $dim(V_h) = 3$. The numerical solutions are computed with an adaptation of step-3 in deal.II [1, 2]. Please notice that the Picture norm is not a proof in the strict mathematical sense: to show that the purple, and blue lines come closer and closer must be confirmed by error estimates as presented for instance in [29] accompanied by numerical simulations in Section 19.3. Of course, for this 1D Poisson problem, we easily observe a limit case, but for more complicated equations it is often not visible whether the solutions do converge.

```
Level  Elements   DoFs
======================
1        2          3
2        4          5
3        8          9
4       16         17
5       32         33
======================
```

19.1.7 Algorithmic details (complement)

We provide some algorithmic details for implementing finite elements in a computer code. Of course, for constant coefficients, a ‘nice’ domain Ω with uniform step lengths, we can immediately build the matrix, right hand side vector and compute the solution. In practice, we are interested in more general formulations. Again, we explain all details in 1D. Further remarks and details (in particular for higher dimensions) can be found in the standard literature for finite elements, e.g., [7].

Assembling integrals on each cell K

In practice, one does not proceed as in Section 19.1.6 since this only makes sense for very simple problems.

A more general procedure is based on an element-wise consideration, which is motivated by:

$$a(u,\phi) = \int_\Omega u'\,\phi'\,dx = \sum_{K\in\mathcal{T}_h} \int_K u'\,\phi'\,dx.$$

The basic procedure is:

Algorithm 19.15 (Basic assembling - robust, but partly inefficient)**.** *Let $K_s, s = 0,\dots n$ be an element and let i and j be the indices of the degrees of freedom (namely the basis functions). The basic algorithm to compute all entries of the system matrix and right hand side vector is:*

```
for all elements $K_s$ with $s = 0,\dots,n$
  for all DoFs $i$ with $i = 0,\dots,n+1$
    for all DoFs $j$ with $j = 0,\dots,n+1$
      $a_{ij} += \int_{K_s} \phi_i'(x)\,\phi_j'(x)\,dx$
```

Here $+=$ means that entries with the <u>*same indices*</u> *i,j are summed. For the right hand side, we have*

```
for all elements $K_s$ with $s = 0,\dots,n$
  for all DoFs $i$ with $i = 0,\dots,n+1$
    $b_i += \int_{K_s} f(x)\phi_i(x)\,dx.$
```

Again $+=$ means that only the b_i with the same i are summed.

Remark 19.13. *This algorithm is a bit inefficient since a lot of zeros are added. Knowing in advance the polynomial degree of the shape functions allows to add an if-condition to assemble only non-zero entries.*

Example in $\mathbb{R}^5$

We illustrate the previous algorithm for a concrete example. Let us compute 1D Poisson on four support points $x_i, i = 0,1,2,3,4$ that are equidistantly distributed yielding a uniform mesh size $h = x_j - x_{j-1}$. The discrete space V_h is given by:

$$V_h = \{\phi_0,\phi_1,\phi_2,\phi_3,\phi_4\}, \quad dim(V_h) = 5.$$

The number of cells is $\#K = 4$.

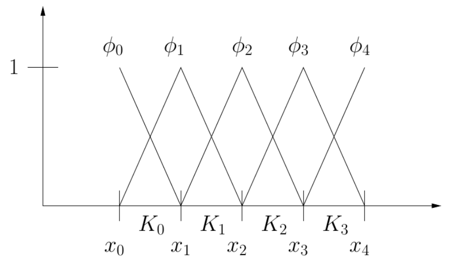


Figure 19.6: Basis functions ϕ_i, elements K_s and nodes x_s used in Section 19.1.7.

It holds furthermore:

$$U \in \mathbb{R}^5, \quad A \in \mathbb{R}^{5\times 5}, \quad b \in \mathbb{R}^5.$$

We start with $s = 0$, namely K_0:

$$a_{00}^{s=0} = a_{00} = \int_{K_0} \phi_0'\phi_0' = \frac{1}{h}, \quad a_{01}^{s=0} = a_{01} = \int_{K_0} \phi_0'\phi_1' = -\frac{1}{h},$$
$$a_{02}^{s=0} = a_{02} = \int_{K_0} \phi_0'\phi_2' = 0, \quad a_{03}^{s=0} = a_{03} = \int_{K_0} \phi_0'\phi_3' = 0, \quad a_{04}^{s=0} = a_{04} = \int_{K_0} \phi_0'\phi_4' = 0.$$

- Similarly, we evaluate $a_{1j}, a_{2j}, a_{3j}, a_{4j}, j = 0, \dots 4$.
- Next, we increment $s = 1$ and work on cell K_1. Here we again evaluate all a_{ij} and sum them to the previously obtained values on K_0. Therefore the $+=$ in the above algorithm.
- We also see that we add a lot of zeros when $|i - j| > 1$. For this reason, a good algorithm first design the sparsity pattern and determines the entries of A that are non-zero. This is clear due to the construction of the hat functions and that they only overlap on neighboring elements.

After having assembled the values on all four elements $K_s, s = 1, 2, 3, 4$, we obtain the following system matrix:

$$A = \begin{pmatrix} \sum_s a_{00}^s & \sum_s a_{01}^s & \sum_s a_{02}^s & \sum_s a_{03}^s & \sum_s a_{04}^s \\ \sum_s a_{10}^s & \sum_s a_{11}^s & \sum_s a_{12}^s & \sum_s a_{13}^s & \sum_s a_{14}^s \\ \sum_s a_{20}^s & \sum_s a_{21}^s & \sum_s a_{22}^s & \sum_s a_{23}^s & \sum_s a_{24}^s \\ \sum_s a_{30}^s & \sum_s a_{31}^s & \sum_s a_{32}^s & \sum_s a_{33}^s & \sum_s a_{34}^s \\ \sum_s a_{40}^s & \sum_s a_{41}^s & \sum_s a_{42}^s & \sum_s a_{43}^s & \sum_s a_{44}^s \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

To fix the homogeneous Dirichlet conditions, we can manipulate directly the matrix A or work with a ‘constraint matrix’. We eliminate the entries of the rows and columns of the off-diagonals corresponding to the boundary indices; here $i = 0$ and $i = 4$. Then:

$$A = \frac{1}{h} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Numerical quadrature

As previously stated, the arising integrals may easily become difficult such that a direct integration is not possible anymore:

- Non-constant right hand sides $f(x)$ and non-constant coefficients $\alpha(x)$;
- Higher-order shape functions;
- Non-uniform step sizes, more general domains.

In modern FEM programs, Algorithm 19.15 is complemented by an alterative evaluation of the integrals using numerical quadrature. The general formula reads:

$$\int_\Omega g(x) \approx \sum_{l=0}^{n_q} \omega_l g(q_l)$$

with quadrature weights ω_l and quadrature points q_l. The number of quadrature points is $n_q + 1$.

Remark 19.14. *The support points x_i and q_l do not need to be necessarily the same. For Gauss quadrature, they are indeed different. For lowest-order interpolatory quadrature rules (box, Trapez) they correspond though.*

Lowest-order quadrature rules A numerical quadrature rule is defined as[4]:

Definition 19.8 (Quadrature rule)**.** *Let $f \in C[a,b]$. A numerical quadrature formula to approximate the integral $I(f) := \int_a^b f(x)\,dx$ is given by*

$$I^n(f) := \sum_{i=0}^{n} \alpha_i f(x_i)$$

with $n+1$ support points $x_0, \ldots, x_n$ and $n+1$ quadrature weights $\alpha_0, \ldots, \alpha_n$.

The simplest quadrature rules are:

Definition 19.9 (Box rule)**.** *The box rule is the simplest quadrature rule. In the interval $[a,b]$ the function $f(x)$ is simply approximated with a constant polynomial $p(x) = f(a)$, i.e.,*

$$I^0(f) = (b-a)f(a),$$

This is the left-sided box rule. A corresponding version by taking the right boundary point is

$$I^0(f) = (b-a)f(b).$$

Definition 19.10 (Midpoint rule)**.** *The function is interpolated in the middle of the interval with a constant polynomial:*

$$I^0(f) = (b-a)f\left(\frac{a+b}{2}\right).$$

Definition 19.11 (Trapezoidal rule)**.** *The trapezoidal rule is obtained by integrating a linear polynomial through $(a, f(a))$ and $(b, f(b))$ in order to approximate the function $f(x)$:*

$$I^1(f) = \frac{b-a}{2}(f(a) + f(b)).$$

[4]The definitions and notation of the quadrature rules are copied and translated from [25].

Remark 19.15. *For more details on quadrature rules we refer to further literature such as for example [25]. Specifically, quadrature rules of optimal order are based on Gauss quadrature in which not only the number of support points are important, but also their (optimal) location. In state-of-the-art finite element software packages, Gauss quadrature is used to evaluate integrals.*

Remark 19.16. *Concerning the order of the quadrature rule (namely up to which order polynomials are integrated in an exact way), numerical quadrature rules show similarities to finite differences schemes. For instance, the trapozoidal rules in FD and numerical quadrature have the same name and the same order. The box rules on the other hand correspond to backward and forward difference quotients yielding the backward and forward Euler schemes.*

Remark 19.17. *It is important though that the quadrature rules are in accordance with the polynomials degree of the finite element scheme. If the quadrature rules do not have sufficient order, the convergence is influenced up to non-solvability of the linear equation system.*

Exercise 19.4. *Evaluate*

$$\int_{-3}^{3} (1-x^2)^2 \, dx$$

using numerical quadrature (box-rule or trapezoidal rule) Which order is necessary to integrate in an exact way?

Exercise 19.5. *Evaluate the entries of the stiffness matrix A using the trapezoidal rule (be careful, the task is nearly trivial).*

Exercise 19.6. *Let $\Omega = (0,1)$. Implement a finite element scheme for solving*

$$-\nabla \cdot (\nabla u) = 1 \quad \text{in } \Omega, \tag{19.33}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{19.34}$$

in octave or C++ or python. Then, change the boundary conditions to

$$u = 1 \quad \text{on } \partial\Omega$$

Check your numerical solution by constructing an analytical solution (satisfying the PDE and the boundary conditions) and evaluate

$$|u_h(x_0) - u(x_0)|,$$

where $x_0 = 0.4$ and u_h indicates the numerical FE solution and u the analytical solution.

Exercise 19.7. *Implement a finite element scheme for solving the PDE:*

$$-\varepsilon u''(x) + u'(x) = 1, \quad \text{in } \Omega = (0,1),$$
$$u(0) = u(1) = 0,$$

where ε is a small but positive parameter, e.g., take $\varepsilon = 1, 10^{-2}, 10^{-4}$. What do you observe in the numerical results with respect to ε?

Exercise 19.8. *Let $\Omega = (0,1)^2$. Implement a finite element scheme for solving*

$$-\nabla \cdot (\nabla u) = 1 \quad \text{in } \Omega, \tag{19.35}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{19.36}$$

in octave or C++ or python.

Trapezoidal rule applied to the Poisson problem We continue the above example by choosing the trapezoidal rule, which in addition, should integrate the arising integrals for the Poisson problem exactly:

$$\int_{K_s} g(x) \approx h_s \sum_{l=0}^{n_q} \omega_l g(q_l)$$

where h_s is the length of interval/element K_s, $n_q = 1$ and $\omega_l = 0.5$. This brings us to:

$$\int_{K_s} g(x) \approx h_s \frac{g(q_0) + g(q_1)}{2}.$$

Applied to our matrix entries, we have on an element K_s:

$$a_{ii} = \int_{K_s} \phi_i'(x)\phi_i'(x)\,dx \approx \frac{h_s}{2}\Big(\phi_i'(q_0)\phi_i'(q_0) + \phi_i'(q_1)\phi_i'(q_1)\Big).$$

For the right hand side, we for the case $f = 1$ we can use for instance the mid-point rule:

$$\frac{1}{h_i}\int_{K_i} \phi_i(x)\,dx \approx \frac{1}{h_i} h_i \phi_i\left(\frac{x_i + x_{i-1}}{2}\right) = \phi_i\left(\frac{x_i + x_{i-1}}{2}\right).$$

Remark 19.18. *If $f = f(x)$ with an dependency on x, we should use a quadrature formula that integrates the function $f(x)\phi_i(x)$ as accurate as possible.*

Remark 19.19. *It is important to notice that the order of the quadrature formula must be sufficiently high since otherwise the quadrature error dominates the convergence behavior of the FEM scheme.*

FE loops We have now all ingredients to extend Algorithm 19.15:

Algorithm 19.16 (Assembling using the trapezoidal rule). *Let $K_s, s = 0, \dots n$ be an element and let i and j be the indices of the degrees of freedom (namely the basis functions). The basic algorithm to compute all entries of the system matrix A and right hand side vector b is:*

```
for all elements $K_s$ with $s = 0, \dots, n$
  for all DoFs $i$ with $i = 0, \dots, n+1$
    for all DoFs $j$ with $j = 0, \dots, n+1$
      for all quad points $l$ with $l = 0, \dots, n_q$
        $a_{ij} += \frac{h_s}{2}\phi_i'(q_l)\phi_j'(q_l)$
```

where $n_q = 1$. Here $+=$ means that entries with the same indices are summed. This is necessary because on all cells K_s we assemble again a_{ij}.

```
for all elements $K_s$ with $s = 0, \dots, n$
  for all DoFs $i$ with $i = 0, \dots, n+1$
    for all quad points $l$ with $l = 0, \dots, n_q$
      $b_i += \frac{h_s}{2} f(q_l)\phi_i(q_l)$
```

Remark 19.20. *In practice it is relatively easy to reduce the computational cost when a lot of zeros are computed. We know due to the choice of the finite element, which DoFs are active on a specific element and can assemble only these shape functions. For linear elements, we could easily add an if condition that only these a_{ij} are assembled when $|i - j| \leq 1$. We finally remark that these loops will definitely work, but there is a second aspect that needs to be considered in practice which is explained in Section 19.1.7.*

Details on the evaluation on the master element

In practice, all integrals are transformed onto a **master element** (or so-called **reference element**) and evaluated there. This has the advantage that

- we only need to evaluate once all basis functions;
- numerical integration formulae are only required on the master element;
- independence of the coordinate system. For instance quadrilateral elements in 2D change their form when the coordinate system is rotated.

The price to pay is to compute at each step a deformation gradient and a determinant, which is however easier than evaluating all the integrals.

We consider the (physical) element $K_i^{(h_i)} = [x_i, x_{i+1}], i = 0, \dots n$ and the variable $x \in K_i^{(h_i)}$ with and $h_i = x_{i+1} - x_i$. Without loss of generality, we work in the following with the first element $K_0^{(h_0)} = [x_0, x_1]$ and $h = h_0 = x_1 - x_0$ as shown in Figure 19.7. The generalization to s elements is briefly discussed in Section 19.1.7.

Figure 19.7: The mesh element $K_0^{(h_0)}$.

The element $K_i^{(h_i)}$ is transformed to the master element (i.e., the unit interval with mesh size $h = 1$) $K^{(1)} := [0, 1]$ with the local variable $\xi \in [0, 1]$ as shown in Figure 19.8.

Figure 19.8: The master element $K^{(1)}$.

For the transformations, we work with the substitution rule (change of variables). Here in 1D, and in higher dimensions with the analogon. We define the mapping

$$\begin{aligned} T_h : K^{(1)} &\to K_0^{(h_0)} \\ \xi &\mapsto T_h(\xi) = x = x_0 + \xi \cdot (x_1 - x_0) = x_0 + \xi h. \end{aligned}$$

While function values can be identified in both coordinate systems, i.e.,

$$f(x) = \hat{f}(\xi), \quad \hat{f} \text{ defined in } K^{(1)},$$

derivatives will be complemented by further terms due to the chain rule that we need to employ. Differentiation in the physical coordinates yields

$$\begin{aligned} \frac{d}{dx} : \quad & 1 = (x_1 - x_0) \frac{d\xi}{dx} \\ \Rightarrow \quad & dx = (x_1 - x_0) \cdot d\xi. \end{aligned}$$

The volume (here in 1D: length) change can be represented by the determinant of the Jacobian of the transformation:

$$J := x_1 - x_0 = h.$$

These transformations follow exactly the way as they are known in continuum mechanics. Thus for higher dimensions we refer exemplarily to [17], where transformations between different domains can be constructed.

We now construct the inverse mapping

$$T_h^{-1} : K_0^{(h_0)} \to K^{(1)}$$
$$x \mapsto T_h^{-1}(x) = \xi = \frac{x - x_0}{x_1 - x_0} = \frac{x - x_0}{h},$$

with the derivative

$$\partial_x T_h^{-1}(x) = \xi_x = \frac{d\xi}{dx} = \frac{1}{x_1 - x_0}.$$

A basis function φ_i^h on $K_0^{(h_0)}$ reads:

$$\varphi_i^h(x) := \varphi_i^1(T_h^{-1}(x)) = \varphi_i^1(\xi)$$

and for the derivative we obtain with the chain rule:

$$\partial_x \varphi_i^h(x) = \partial_\xi \varphi_i^1(\xi) \cdot \partial_x T_h^{-1}(x) = \partial_\xi \varphi_i^1(\xi) \cdot \xi_x$$

with $T_h^{-1}(x) = \xi$.

Example 19.1. *We provide two examples. Firstly:*

$$\int_{K_h} f(x)\,\varphi_i^h(x)\,dx \overset{Sub.}{=} \int_{K^{(1)}} f(T_h(\xi)) \cdot \varphi_i^1(\xi) \cdot J \cdot d\xi, \tag{19.37}$$

and secondly,

$$\int_{K_h} \partial_x \varphi_i^h(x) \cdot \partial_x \varphi_j^h(x)\,dx = \int_{K^{(1)}} \big(\partial_\xi \varphi_i^1(\xi)\big) \cdot \xi_x \cdot \big(\partial_\xi \varphi_j^1(\xi)\big) \cdot \xi_x \cdot J\,d\xi. \tag{19.38}$$

We can now apply numerical integration using again the trapezoidal rule and obtain for the two previous integrals:

$$\int_{T_h} f(x)\,\varphi_i^h(x)\,dx \overset{(19.37)}{\approx} \sum_{k=1}^q \omega_k f(F_h(\xi_k))\,\varphi_i^1(\xi_k) \cdot J$$

and for the second example:

$$\int_{T_h} \partial_x \varphi_j^h(x) \partial_x \varphi_i^h(x)\,dx \approx \sum_{k=1}^q \omega_k \big(\partial_\xi \varphi_j^1(\xi_k) \cdot \xi_x\big) \cdot \big(\partial_\xi \varphi_i^1(\xi_k) \cdot \xi_x\big) \cdot J.$$

Remark 19.21. *These final evaluations can again be realized by using Algorithm 19.16, but are now performed on the unit cell $K^{(1)}$.*

Generalization to s elements

We briefly setup the notation to evaluate the integrals for s elements:

- Let n be the index of the end point $x_n = b$ (b is the nodal point of the right boundary). Then, $n-1$ is the number of elements (intervals in 1d), and $n+1$ is the number of the nodal points (degrees of freedom - DoFs) and the number shape functions, respectively (see also Figure 19.6);
- $K_s^{(h_s)} = [x_s, x_{s+1}], s = 0, \dots n-1$.
- $h_s = x_{s+1} - x_s$;

- $T_s : K^{(1)} \to K_s^{(h_s)} : \xi \mapsto T_s(\xi) = x_s + \xi(x_{s+1} - x_s) = x_s + \xi h_s$;
- $T_s^{-1} : K_s^{(h_s)} \to K^{(1)} : x \mapsto T_s^{-1}(x) = \frac{x - x_s}{h_s}$;
- $\nabla T_s^{-1}(x) = \partial_x T_s^{-1}(x) = \frac{1}{h_s}$ (in 1D);
- $\nabla T_s(\xi) = \partial_x T_s(\xi) = (x_s + \xi h_s)' = h_s$ (in 1D);
- $J_s := \det(\nabla T_s(\xi)) = (x_s + \xi h_s)' = h_s$ (in 1D).

Example: Section 19.1.7 continued using Sections 19.1.7 and 19.1.7

We continue Example 19.1.7 and build the system matrix A by using the master element. In the following, we evaluate the Laplacians in weak form:

$$\begin{aligned} a_{ij} = \int_{K_s} \phi_i'(x) \cdot \phi_j'(x)\, dx &= \int_{K^{(1)}} \phi_i'(\xi)\, \partial_x T_s^{-1}(x) \cdot \phi_j'(\xi)\, \partial_x T_s^{-1}(x)\, J_s\, d\xi \\ &= \int_{K^{(1)}} \phi_i'(\xi)\, \frac{1}{h_s} \cdot \phi_j'(\xi)\, \frac{1}{h_s}\, h_s\, d\xi \\ &= \int_{K^{(1)}} \phi_i'(\xi) \cdot \phi_j'(\xi)\, \frac{1}{h_s}\, d\xi. \end{aligned}$$

We now compute **once!** the required integrals on the unit element (i.e., the master element) $K^{(1)}$. Here, $\xi_0 = 0$ and $\xi_1 = 1$ with $h^{(1)} = \xi_1 - \xi_0 = 1$ resulting in two shape functions:

$$\phi_0(\xi) = \frac{\xi_1 - \xi}{h^{(1)}}, \qquad \phi_1(\xi) = \frac{\xi - \xi_0}{h^{(1)}}.$$

The derivatives are given by:

$$\phi_0'(\xi) = \frac{-1}{h^{(1)}} = -1, \qquad \phi_1'(\xi) = \frac{1}{h^{(1)}} = 1.$$

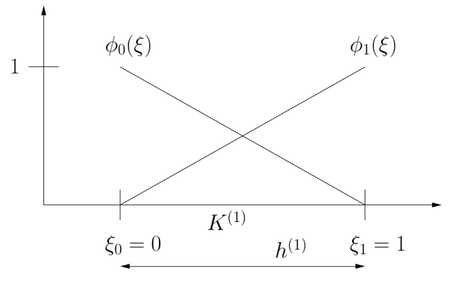


Figure 19.9: The master element $K^{(1)}$ including nodal points and mesh size parameter $h^{(1)}$.

With these calculations, we compute all combinations on the master element required to evaluate the Laplacian in 1d:

$$\begin{aligned} a_{00}^{(1)} &= \int_{K^{(1)}} \phi_0'(\xi) \cdot \phi_0'(\xi)\, d\xi = \int_{K^{(1)}} (-1)^2\, d\xi = 1, \\ a_{01}^{(1)} &= \int_{K^{(1)}} \phi_0'(\xi) \cdot \phi_1'(\xi)\, d\xi = \int_{K^{(1)}} (-1) \cdot 1\, d\xi = -1, \\ a_{10}^{(1)} &= \int_{K^{(1)}} \phi_1'(\xi) \cdot \phi_0'(\xi)\, d\xi = \int_{K^{(1)}} 1 \cdot (-1)\, d\xi = -1, \\ a_{11}^{(1)} &= \int_{K^{(1)}} \phi_1'(\xi) \cdot \phi_1'(\xi)\, d\xi = \int_{K^{(1)}} 1 \cdot 1\, d\xi = 1. \end{aligned}$$

It is clear what happens on each element K_s using Algorithm 19.15:

$$
\begin{aligned}
a_{00}^s &= \int_{K^{(1)}} \phi_0'(\xi)\cdot\phi_0'(\xi)\,\frac{1}{h_s}\,d\xi = \frac{1}{h_s}\int_{K^{(1)}} (-1)^2\,d\xi = \frac{1}{h_s},\\
a_{01}^s &= \int_{K^{(1)}} \phi_0'(\xi)\cdot\phi_1'(\xi)\,\frac{1}{h_s}\,d\xi = \frac{1}{h_s}\int_{K^{(1)}} (-1)\cdot 1\,d\xi = \frac{-1}{h_s},\\
a_{02}^s &= \int_{K^{(1)}} \phi_0'(\xi)\cdot\phi_2'(\xi)\,\frac{1}{h_s}\,d\xi = \frac{1}{h_s}\int_{K^{(1)}} (-1)\cdot 0\,d\xi = 0,\\
a_{03}^s &= \int_{K^{(1)}} \phi_0'(\xi)\cdot\phi_3'(\xi)\,\frac{1}{h_s}\,d\xi = \frac{1}{h_s}\int_{K^{(1)}} (-1)\cdot 0\,d\xi = 0,\\
a_{04}^s &= \int_{K^{(1)}} \phi_0'(\xi)\cdot\phi_4'(\xi)\,\frac{1}{h_s}\,d\xi = \frac{1}{h_s}\int_{K^{(1)}} (-1)\cdot 0\,d\xi = 0.
\end{aligned}
$$

Next, we increase $i \to i+1$ resulting in $i = 1$ and compute:

$$a_{10}^s,\quad a_{11}^s,\quad a_{12}^s,\quad a_{13}^s,\quad a_{14}^s.$$

We proceed until $i = 4$. This procedure is done for all elements K_s with $s = 0,1,2,3$. As stated in Algorithm 19.15, the procedure is however inefficient since a lot of zeros are assembled. Knowing the structure of the matrix (i.e., the sparsity pattern) allows us upfront only to assemble the entries with non-zero entries.

Anyhow, the system matrix is composed by (the same as in Section 19.1.7):

$$
\begin{aligned}
A &= \begin{pmatrix}
\sum_s a_{00}^s & \sum_s a_{01}^s & \sum_s a_{02}^s & \sum_s a_{03}^s & \sum_s a_{04}^s\\
\sum_s a_{10}^s & \sum_s a_{11}^s & \sum_s a_{12}^s & \sum_s a_{13}^s & \sum_s a_{14}^s\\
\sum_s a_{20}^s & \sum_s a_{21}^s & \sum_s a_{22}^s & \sum_s a_{23}^s & \sum_s a_{24}^s\\
\sum_s a_{30}^s & \sum_s a_{31}^s & \sum_s a_{32}^s & \sum_s a_{33}^s & \sum_s a_{34}^s\\
\sum_s a_{40}^s & \sum_s a_{41}^s & \sum_s a_{42}^s & \sum_s a_{43}^s & \sum_s a_{44}^s
\end{pmatrix}\\
&= \begin{pmatrix}
\frac{1}{h_0} & \frac{-1}{h_0} & 0 & 0 & 0\\
\frac{-1}{h_0} & \frac{1}{h_0}+\frac{1}{h_1} & \frac{-1}{h_1} & 0 & 0\\
0 & \frac{-1}{h_1} & \frac{1}{h_1}+\frac{1}{h_2} & \frac{-1}{h_2} & 0\\
0 & 0 & \frac{-1}{h_2} & \frac{1}{h_2}+\frac{1}{h_3} & \frac{-1}{h_3}\\
0 & 0 & 0 & \frac{-1}{h_3} & \frac{1}{h_3}
\end{pmatrix}.
\end{aligned}
$$

This matrix is the same as if we had evaluated all shape functions on the physical elements with a possibly non-uniform step size $h_s, s = 0,1,2,3$.

19.1.8 Quadratic finite elements (complement)

Algorithmic aspects

We explain the idea how to construct higher-order FEM in this section. Specifically, we concentrate on P_2 elements, which are quadratic on each element K_s. First we define the discrete space:

$$V_h = \{v \in C[0,1] | \, v|_{K_j} \in P_2\}$$

The space V_h is composed by the basis functions:

$$V_h = \{\phi_0, \dots, \phi_{n+1}, \phi_{\frac{1}{2}}, \dots, \phi_{n+\frac{1}{2}}\}.$$

The dimension of this space is $\dim(V_h) = 2n+1$. Here, we followed the strategy that we use the elements K_s as in the linear case and add the mid-points in order to construct unique parabolic functions on each K_s. The mid-points represent degrees of freedom as the two edge points. For instance on each $K_j = [x_j, x_{j+1}]$ we have as well $x_{j+\frac{1}{2}} = x_j + \frac{h}{2}$, where $h = x_{j+1} - x_j$. From this construction it is clear (not proven though!) that quadratic FEM have a higher accuracy than linear FEM.

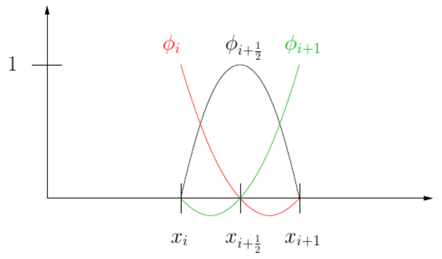


Figure 19.10: Quadratic basis functions

The specific construction of shape functions can be done as shown in 19.1.6 or using directly Lagrange basis polynomials (see lectures on introduction of numerical methods).

Definition 19.12 (P_2 shape functions)**.** *On the master element $K^{(1)}$, we have*

$$\phi_0(\xi) = 1 - 3\xi + 2\xi^2,$$
$$\phi_{\frac{1}{2}}(\xi) = 4\xi - 4\xi^2,$$
$$\phi_1(\xi) = -\xi + 2\xi^2.$$

These basis functions fulfill the property:

$$\phi_i(\xi_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

for $i, j = 0, \frac{1}{2}, 1$. On the master element, a function has therefore the prepresentation:

$$u(\xi) = \sum_{j=0}^{1} u_j \phi_j(\xi) + u_{\frac{1}{2}} \phi_{\frac{1}{2}}(\xi).$$

This definition allows us to construct a global function from V_h:

Proposition 19.17. *The space V_h is a subspace of V. Each function from V_h has a unique representation and is defined by its nodal points:*

$$u_h(x) = \sum_{j=0}^{n+1} u_j \phi_j(x) + \sum_{j=0}^{n} u_{j+\frac{1}{2}} \phi_{\frac{1}{2}}(x).$$

Remark 19.22. *The assembling of A, u and b is done in a similar fashion as shown in detail for linear finite elements.*

Numerical test: 1D Poisson using quadratic FEM

We continue Section 19.1.6.

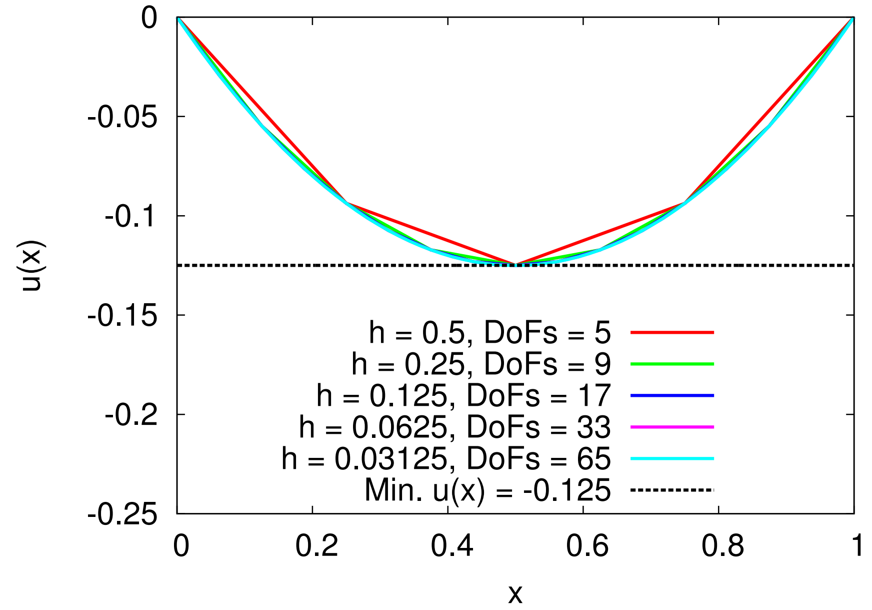


Figure 19.11: Solution of the 1D Poisson problem with $f = -1$ using quadratic finite elements with various mesh sizes h. DoFs is the abbreviation for degrees of freedom; here the number of support points x_j. The dimension of the discrete space is $DoFs$. For instance for $h = 0.5$, we have 2 mesh elements, we have 5 DoFs and three basis functions, thus $dim(V_h) = 5$. The numerical solutions are computed with an adaptation of step-3 in deal.II [1, 2]. Please notice that the Picture norm is not a proof in the strict mathematical sense: to show that the purple, and blue lines come closer and closer must be confirmed by error estimates as presented for instance in [29] accompanied by numerical simulations in Section 19.3. Of course, for this 1D Poisson problem, we easily observe a limit case, but for more complicated equations it is often not visible whether the solutions do converge.

Remark 19.23. *Be careful when plotting the solution using higher-order FEM. Sometimes, the output data is only written into the nodal values values defining the physical elements. In this case, a quadratic function is visually the same as a linear function. Plotting in all degrees of freedom would on the other hand shows also visually that higher-order FEM are employed.*

```
Level  Elements   DoFs
======================
1        2          5
2        4          9
3        8         17
4       16         33
5       32         65
======================
```

Influence of numerical quadrature

We discuss one simple example for a situation where the system matrix will be singular. Since $r' \geq m$ we have convergence of the procedure. Where r' means the degree of the quadrature rule and m is the degree of the FEM polynomials.

We assume the one-dimensional Poisson problem, take quadratic finite elements and use the middle-point rule as integration formula. So we solve

$$-u'' = f, \quad u(0) = (1) = 0$$

and take quadratic basic functions in the interval $[0,1]$ with equidistant step size $h = x_i - x_{i-1}$. There is an additional point required, so we take the middle point of the cells T_i,

$$x_{i-1/2} := \frac{1}{2}(x_{i-1} - x_i)$$

The construction of the quadratic function v satisfies

$$v(x) = \sum_{i=0}^{N} v_i \psi_i(x) + \sum_{i=1}^{N} v_{i-1/2} \psi_{i-1/2}(x)$$

with the following properties

i) $\psi_i(x_k) = \delta_{ik}, \quad \psi_i(x_{k-1/2}) = 0$

ii) $\psi_{i-1/2}(x_k) = 0, \quad \psi_{i-1/2}(x_{k-1/2}) = \delta_{ik}$

Now, we get three quadratic basic functions

$$\psi_i(x) = \begin{cases} \frac{(x-x_{i-1})(x-x_{i-1/2})}{(x_i-x_{i-1})(x_i-x_{i-1/2})}, & x \in T_i \\ \frac{(x-x_{i+1})(x-x_{i+1/2})}{(x_i-x_{i+1})(x_i-x_{i+1/2})}, & x \in T_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

and

$$\psi_{i-1/2}(x) = \begin{cases} \frac{(x-x_{i-1})(x-x_i)}{(x_{i-1/2}-x_i)(x_{i-1/2}-x_{i+1})}, & x \in T_i \\ 0, & \text{otherwise} \end{cases}$$

Next task is constructing the derivatives and give the variational formulation of the one-dimensional Laplace problem. Then solving integrals with middle-point rule,

$$\int_a^b f(x)\,dx \approx (b-a)\, f\left(\frac{a+b}{2}\right) + \frac{1}{24} f''(\xi)(b-a)^3 \tag{19.39}$$

One know that the middle-point rule has degree 1. Using this quadrature formula for quadratic elements fails and the resulting systemmatrix A will have some zeros, so it is singular. For a local element we obtain

$$\partial_x \psi_{i-1}(x) = \frac{2}{h_i^2}(2x - x_{i-1/2} - x_i)$$

$$\partial_x \psi_{i-1/2}(x) = \frac{2}{h_i^2}(x_i + x_{i-1} - 2x)$$

$$\partial_x \psi_i(x) = \frac{2}{h_i^2}(2x - x_{i-1/2} - x_{i-1})$$

and the following integrals

$$
\begin{aligned}
A_{11} &= \int_{x_{i-1}}^{x_i} (\partial_x \psi_{i-1})^2 \, dx \\
A_{22} &= \int_{x_{i-1}}^{x_i} (\partial_x \psi_{i-1/2})^2 \, dx \approx h_i (\partial_x \psi_{i-1/2}(x_{i-1/2}))^2 = 0 \\
A_{33} &= \int_{x_{i-1}}^{x_i} (\partial_x \psi_i)^2 \, dx \\
A_{12} = A_{21} &= \int_{x_{i-1}}^{x_i} \partial_x \psi_{i-1} \cdot \partial_x \psi_{i-1/2} \, dx \approx h_i \left[(\partial_x \psi_{i-1} \cdot \partial_x \psi_{i-1/2})(x_{i-1/2}) \right] = 0 \\
A_{13} = A_{31} &= \int_{x_{i-1}}^{x_i} \partial_x \psi_{i-1} \cdot \partial_x \psi_i \, dx \\
A_{23} = A_{32} &= \int_{x_{i-1}}^{x_i} \partial_x \psi_{i-1/2} \cdot \partial_x \psi_i \, dx \approx h_i \left[(\partial_x \psi_{i-1/2} \cdot \partial_x \psi_i)(x_{i-1/2}) \right] = 0
\end{aligned}
$$

At last we write the results in our matrix,

$$
A = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & 0 & 0 \\ A_{31} & 0 & A_{33} \end{pmatrix} \tag{19.40}
$$

19.2 Numerical solution of the arising linear equation systems (complement)

The previous choices of basis functions $\phi_h \in V_h$ (or better the space V_h itself) have been constructed in such a way that some special properties for the matrix are obtained:

- In order to obtain an accurate representation of the solution u_h we need to work with a great number of basis functions ϕ_h, i.e., the space V_h is large, i.e, $\dim(V_h) = n$ is large!
- On the other hand A should be sparse in order to allow for a fast solution.
- The condition number of the matrix should be not too bad because a bad condition number results in a large number of iterations when using iterative solution methods for solving $Au = b$. The condition number $\chi(A)$ of a matrix A is defined as

  $$\chi(A) = \frac{\lambda_{max}}{\lambda_{min}},$$

  where $\lambda_{max} = \max_j \lambda_j$ and $\lambda_{min} = \min_j \lambda_j$ are the maximal and minimal eigenvalues of the matrix A.

Remark 19.24. *The wishes No. 2 and 3 are again fulfilled by using finite elements rather than other basis functions.*

Remark 19.25. *As other remarks before, the numerical solution of coupled, nonlinear, partial differential equations, which are* ***robust*** *and* ***efficient*** *is an active research topic.*

19.2.1 Basic iterative solvers

We discussed in detail classical iterative solvers in Section 7.2. In fact for the classical Poisson problem, the preconditioned conjugate method is very often used. More ambitious and sophisticated are multigrid methods that are described in detail in the next section. Therein, however, basic iterative methods such as Gauss-Seidel, are used as subsolvers, so-called smoothers. Therefore, understanding of the basic iterative methods remains important.

19.2.2 Geometric multigrid methods

In this section, we follow closely [3][Chapter 10]. Excellent descriptions including historical notes can be found in Braess [4] and the famous book from Hackbusch [15].

Motivation

For elliptic problems, the linear systems are large, sparse and symmetric positiv definite (s.p.d.). These properties already were necessary to use the CG method for the iterative solution. However, the convergence properties depend on the condition number. We recall again that the usual stiffness matrix behaves as (see e.g., [18])

$$\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}} = O(h^{-2}).$$

Example 19.2. *Referring forward to Section 19.2.2, we obtain for instance for* $h = 0.5, 0.25, 0.125$*:*

$$\kappa_{h=0.5}(A) = 8$$
$$\kappa_{h=0.25}(A) = 54$$
$$\kappa_{h=0.125}(A) = 246$$

reflecting that the condition number grows with $O(h^{-2})$*.*

The work amount in terms of arithmetic operations for often-used methods is listed in Table 19.1. Therein, we already see that the CG method may perform extremely well; in particular in 3d.

Table 19.1: Operations for solving $Az = b$ with $A \in \mathbb{R}^{n\times n}$ being large, sparse, and s.p.d.

Scheme	$d = 2$	$d = 3$
Gauss (direct)	n^3	n^3
Banded-Gauss (direct)	n^2	$n^{7/3}$
Jacobi (iterative)	n^2	$n^{5/3}$
Gauss-Seidel (iterative)	n^2	$n^{5/3}$
CG	$n^{3/2}$	$n^{4/3}$
SOR with opt. ω	$n^{3/2}$	$n^{4/3}$
Multigrid	n	n

Multigrid methods can solve linear systems with optimal complexity; namely n. Less than n is not possible since each entry has to be looked up at least once.

The key idea of multigrid is:

- Solve a given PDE on a hierarchy of meshes
- Use on each mesh a (simple) iterative solver, e.g., Jacobi or Gauss-Seidel
- The previous two steps will smooth out the error contributions

Smoothing property of Richardson iteration

In this first section, we explain the motivation why to solve on a hierarchy of meshes using a simple iterative solver. Let

$$Ax = b$$

be given. Richardson iteration (Numerik 1 [25]) then reads:

$$x^{k+1} = x^k + \omega(b - Ax^k)$$

with a relaxation parameter ω. We know that Richardson converges when $0 < \omega < 2/\lambda_{max}(A)$ (see e.g., [25]). Since A is s.p.d. the spectrum of eigenvalues can be ordered as

$$\sigma(A) = \{\lambda_{min}(A) = \lambda_1, \lambda_2, \dots, \lambda_n = \lambda_{max}(A)\}$$

with

$$0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n.$$

We can now write the iteration error $e^{k+1} = x - x^{k+1}$. Specifically:

$$\begin{aligned} e^{k+1} &= x - x^{k+1} \\ &= x - x^k - \omega(b - Ax^k) \quad \text{(Richardson iteration function)} \\ &= (I - \omega A)(x - x^k) \\ &= (I - \omega A)e^k \\ &= Me^k. \end{aligned}$$

The matrix $M = (I - \omega A)$ is the so-called iteration matrix. In Numerik 1, we have analyzed under which conditions for M we obtain convergence. Since we A is s.p.d. we can perform an eigenvalue decomposition. Let (λ_i, z_i) an eigenpair of A, where λ_i is the eigenvalue and z_i the eigenvector. We recall that $Az_i = \lambda_i z_i$ is the eigenvalue equation. We apply this idea to the Me^k:

$$Mz_i = (I - \omega A)z_i = (1 - \omega\lambda_i)z_i.$$

We recall from linear algebra that the $z_i, i = 1, \dots, n$ form a basis of $\mathbb{R}^n$. Consequently, any vector e^k can be written as linear combination with the help of the basis vectors. Thus:

$$Me = M\sum_{i=1}^n a_i z_i = \sum_{i=1}^n a_i(1 - \omega\lambda_i)z_i,$$

where $a_i \in \mathbb{R}$ are the coefficients. Now we have a relation between the error e and the factor $(1 - \omega\lambda_i)$. It is clear that the error depends on the size of $(1 - \omega\lambda_i)$. For the special choice $\omega = 1/\lambda_n < 2/\lambda_n$, we observe the reduction factor:

$$\begin{aligned} 1 - \omega\lambda_i &= 1 - \frac{\lambda_i}{\lambda_n} \quad \to 0 \quad \text{when } i \text{ is large, i.e., } \lambda_i \approx \lambda_n \\ 1 - \omega\lambda_i &= 1 - \frac{\lambda_i}{\lambda_n} \quad \to 1 \quad \text{when } i \text{ is small, i.e., } \lambda_i \ll \lambda_n \end{aligned}$$

We remember from Numerik 1 that the first situation is much better, because we have a fast error reduction. What does this observation mean? First, large eigenvalues with $\lambda_i \approx \lambda_n$ are damped (smoothed) quickly in the iteration error. Second, smaller eigenvalues are not damped quickly. However, smaller eigenvalues become larger on coarser grids (smaller n). And here we are: we use a hierachy of meshes in order to damp all eigenvalues efficiently.

Numerical example of eigenvalues and eigenvectors for 1D Poisson

For Poisson in 1D, we have our well known matrix

$$A = \frac{1}{h}\begin{pmatrix} -1 & 2 & -1 & 0 & 0 \\ & & \ddots & & \\ 0 & -1 & 2 & -1 & 0 \\ & & \ddots & & \\ 0 & 0 & -1 & 2 & -1 \end{pmatrix}.$$

The eigenvalues are

$$\lambda_i = 4\sin^2(i\pi h/2), \quad i = 1, \dots n-1$$

and the corresponding eigenvectors:

$$(z_i)_k = \sin(k/N i\pi), \quad 1 \le i, k < n.$$

Low frequency eigenvectors are obtained for small i. High frequency eigenvectors are for $i \approx n$. Whether an eigenvalue is low frequent or not depends on the mesh size h (and consequently on n). This means:

- High frequency errors with $i > n/2$ are damped quickly with Richardson;
- Low frequency errors with $i \le i/2$ are damped slowly with Richardson.

Consequently, if we transfer low frequency errors to coarser meshes, they become high frequent and we could again damp them with Richardson. The realization requires consequently are hierarchy of meshes.

We demonstrate our previous descriptions with a numerical example on three different (hierarchical) meshes. The starting point is Section 19.1.6. We use octave and specifically the function

```
[V,lambda] = eig(A)
```

We use on $\Omega = (0,1)$ three mesh sizes $h = 0.5, 0.25, 0.125$. Starting on the finest mesh, we obtain:

```
A2 =

     1     0     0     0     0     0     0     0     0
   -64   128   -64     0     0     0     0     0     0
     0   -64   128   -64     0     0     0     0     0
     0     0   -64   128   -64     0     0     0     0
     0     0     0   -64   128   -64     0     0     0
     0     0     0     0   -64   128   -64     0     0
     0     0     0     0     0   -64   128   -64     0
     0     0     0     0     0     0   -64   128   -64
     0     0     0     0     0     0     0     0     1

V =

   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.52743   0.00000
   0.19134  -0.35355   0.46194   0.50000  -0.19134  -0.35355   0.46194   0.48112   0.07816
  -0.35355   0.50000  -0.35355  -0.00000  -0.35355  -0.50000   0.35355   0.42729   0.15511
   0.46194  -0.35355  -0.19134  -0.50000  -0.46194  -0.35355  -0.19134   0.36679   0.22962
  -0.50000  -0.00000   0.50000  -0.00000  -0.50000   0.00000  -0.50000   0.30055   0.30055
   0.46194   0.35355  -0.19134   0.50000  -0.46194   0.35355  -0.19134   0.22962   0.36679
  -0.35355  -0.50000  -0.35355   0.00000  -0.35355   0.50000   0.35355   0.15511   0.42729
   0.19134   0.35355   0.46194  -0.50000  -0.19134   0.35355   0.46194   0.07816   0.48112
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.52743

lambda =
   246.2566          0          0          0          0          0          0          0          0
          0   218.5097          0          0          0          0          0          0          0
          0          0   176.9835          0          0          0          0          0          0
          0          0          0   128.0000          0          0          0          0          0
          0          0          0          0     9.7434          0          0          0          0
          0          0          0          0          0    37.4903          0          0          0
          0          0          0          0          0          0    79.0165          0          0
          0          0          0          0          0          0          0     1.0000          0
          0          0          0          0          0          0          0          0     1.0000
```

We see that the lower eigenvalues $\lambda_i = 79, 37, 9, 1, 1$ for $i \leq n/2$ are relatively small in comparison to $\lambda_n = 246$. Now we coarsen the mesh using $h = 0.25$ and obtain:

```
A1 =

    1    0    0    0    0
  -16   32  -16    0    0
    0  -16   32  -16    0
    0    0  -16   32  -16
    0    0    0    0    1

V =

   0.00000   0.00000   0.00000   0.69531   0.00000
  -0.50000   0.70711   0.50000   0.56348   0.20461
   0.70711   0.00000   0.70711   0.39644   0.39644
  -0.50000  -0.70711   0.50000   0.20461   0.56348
   0.00000   0.00000   0.00000   0.00000   0.69531

lambda =

   54.6274         0         0         0         0
         0   32.0000         0         0         0
         0         0    9.3726         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000
```

We see that the lower eigenvalues $\lambda_i = 1, 1$ for $i < n/2$ are relatively small in comparison to $\lambda_n = 54$. Therefore, a relatively small eigenvalue $\lambda_5 = 79$ in A_2 became a the biggest eigenvalue $\lambda_5 = 54$ on the coarser grid. Now we coarsen again the mesh using $h = 0.5$ and obtain:

```
A0 =

   1   0   0
  -4   8  -4
   0   0   1

V =

   0.00000   0.86824   0.00000
   1.00000   0.49614   0.49614
   0.00000   0.00000   0.86824
```

```
lambda =


   8   0   0
   0   1   0
   0   0   1
```

And again, a small eigenvalue in A_1 became the biggest eigenvalue $\lambda_3 = 8$ on the coarsest mesh. Of course, this behavior of the eigenvalues and eigenvalues is due to the structure of the linear equation system and therefore due to the problem statement; here Poisson.

Variational geometric multigrid

Usually the systematic procedure for multigrid is first explained for a two-grid method. This idea is then extended to *multiple* meshes.

Let $V_h = \{\varphi_1, \ldots, \varphi_n\}$, $dim(V_h) = n$, and we consider the well-known abstract problem: Find $u_h \in V_h$ such that

$$a(u_h, \varphi_h) = l(\varphi_h) \quad \forall \varphi_h \in V_h.$$

We know for $v_h \in V_h$:

$$v_h = \sum_{i=1}^{n} x_i \varphi_i$$

with $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ being the coefficient vector. Consequently, we have the correspondance:

$$x \in \mathbb{R}^n \quad \leftrightarrow \quad v_h \in V_h$$

with the relation $dim(V_h) = n$. We now consider hierarchic refinement and ask that the finite element spaces are nested:

$$V_{2h}^{(1)} \subset V_h^{(1)}$$

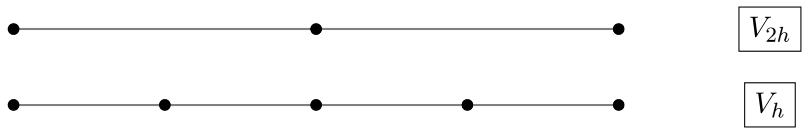


Figure 19.12: Hierarchical refinement and nested FEM spaces $V_{2h} \subset V_h$.

Formulation 19.18 (Two-grid method)**.** *A two-grid method can be formulated as:*

1. *Let $u_h^k \in V_h$ be given. Compute $u_h^{k,1} \in V_h$ by iterating ν times Richardson. This means:*

$$\begin{aligned} \textit{Set } x_h^{k,0} &= x_h^k \\ \textit{Iterate } x_h^{k,i/\nu} &= x_h^{k,(i-1)/\nu} + \omega(b - A_h x_h^{k,(i-1)/\nu}), \quad 1 \le i \le \nu. \end{aligned}$$

   For $i = \nu$, we have obtained $x_h^{k,1} \in \mathbb{R}^{dim(V_h)}$.

2. *Coarse grid correction: we now solve the problem on the coarser grid $\mathcal{T}_{2h}$ with V_{2h}. Find $w \in V_{2h}$ such that*

$$a(u_h^{k,1} + w, v) = l(v) \quad \forall v \in V_{2h}, \tag{19.41}$$

   where $u_h^{k,1}$ is now restricted to the coarser mesh. Set

$$u_h^{k+1} = u_h^{k,1} + w,$$

   where w is now prolongated back to the fine grid.

We explain the second step now in more detail. We define the bases

$$\begin{aligned} V_h &= \{\varphi_1^h, \dots, \varphi_n^h\}, \\ V_{2h} &= \{\varphi_1^{2h}, \dots, \varphi_m^{2h}\}. \end{aligned}$$

Since $V_{2h} \subset V_h$, there exist coefficients r_{ij} to restrict the fine grid function to the coarse grid:

$$\varphi_i^{2h} = \sum_{j=1}^n r_{ij} \varphi_j^h, \quad 1 \le i \le m$$

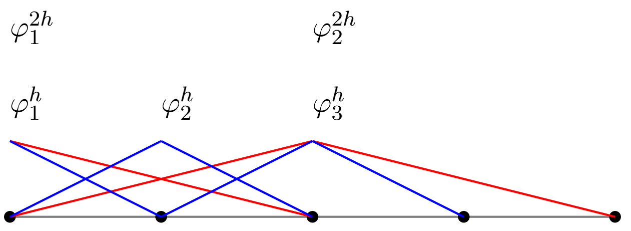


Figure 19.13: Illustration of $\varphi_i^{2h} = \sum_{j=1}^n r_{ij}\varphi_j^h$.

The question is how to compute the restriction operator (matrix) $R = (r_{ij})$? Using the above relation, this can be done explicitly. For instance, observing Figure 19.13, we see

$$\varphi_1^{2h} = r_{11}\varphi_1^h + r_{12}\varphi_2^h + \dots + r_{15}\varphi_5^h$$

In more detail, φ_1^{2h} can be constructed as:

$$\varphi_1^{2h} = 1\varphi_1^h + 0.5\varphi_2^h$$

yielding $r_{11} = 1, r_{12} = 0.5, r_{1j} = 0$ for $j = 3, 4, 5$. Extending to φ_2^{2h} and φ_3^{2h}, we obtain:

$$R_{2h}^h = \begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 1 \end{pmatrix} \quad \in \mathbb{R}^{3\times 5}$$

Remark 19.26. *As usual, in the actual implementation, care has to be taken how the boundary conditions are treated in the FEM code and possibly the indices corresponding to boundary values may be modified.*

We now insert this basis representation into (19.41):

$$\begin{aligned}
a(u_h^{k,1}+w,v) &= l(v) \quad \forall v\in V_{2h}\\
\Leftrightarrow \quad a(w,v) &= l(v) - a(u_h^{k,1},v)\\
\Leftrightarrow \quad a\left(\sum_{j=1}^{N_{2h}} y_j\varphi_j^{2h},\varphi_i^{2h}\right) &= l(\varphi_i^{2h}) - a\left(\sum_{n=1}^{N_h} x_n^{k,1}\varphi_n^h,\varphi_i^{2h}\right), \quad 1\le i\le N_{2h}\\
\Leftrightarrow \quad \sum_{j=1}^{N_{2h}} y_j a\left(\varphi_j^{2h},\varphi_i^{2h}\right) &= l\left(\sum_{m=1}^{N_h} r_{im}\varphi_m^h\right) - a\left(\sum_{n=1}^{N_h} x_n^{k,1}\varphi_n^h, \sum_{m=1}^{N_h} r_{im}\varphi_m^h\right)\\
\Leftrightarrow \quad \sum_{j=1}^{N_{2h}} y_j a\left(\varphi_j^{2h},\varphi_i^{2h}\right) &= \sum_{m=1}^{N_h} r_{im}\left(l\left(\varphi_m^h\right) - a\left(\sum_{n=1}^{N_h} x_n^{k,1}\varphi_n^h,\varphi_m^h\right)\right)\\
\Leftrightarrow \quad A_{2h}y &= R_{2h}^h(b_h - A_h x_h^{k,1})
\end{aligned}$$

The rectangular restriction matrix is determined by $(R_{2h}^h)_{ij} = r_{ij}$.

Algorithm 19.19 (Two grid method (TGM) - algebraic version)**.** *We denote the linear equation systems by $A_h x_h = b_h$ and $A_{2h}x_{2h} = b_{2h}$ with $A_h\in\mathbb{R}^{n\times n}$ and $A_{2h}\in\mathbb{R}^{m\times m}$ with $m<n$. Let the current iterate $x_h^k\in\mathbb{R}^n$ be given. The following method computes the next iterate $x_h^{k+1}\in\mathbb{R}^n$:*

```
$TGM(x_h^k)$
{
$x_h^{k,0} = x_h^k$
$for\ (i = 1,\dots,\nu)$
    $x_h^{k,i/\nu} = x_h^{k,(i-1)/\nu} + \omega(b - A_h x_h^{k,(i-1)/\nu})$   Pre-smoothing
$d_h = b_h - A_h x_h^{k,1}$   Defect on fine grid
$d_{2h} = R_{2h}^h d_h$   Restriction to coarse grid
$x_{2h} = A_{2h}^{-1} d_{2h}$   Coarse grid solve
$y_h = (R_{2h}^h)^T x_{2h}$   Prolongation to fine grid
$x_h^{k,2} = x_h^{k,1} + y_h$   Coarse grid correction applied to fine grid solution
return $x_h^{k,2}$
}
```

We see in this algorithm, the two-grid method can be extended to a multigrid method by applying recursively the idea to the coarse grid solve $y_{2h} = A_{2h}^{-1}d_{2h}$.

Remark 19.27 (Solve on the coarsest grid)**.** *If A_{2h} is sufficiently small (i.e., $\mathcal{T}_{2h}$ sufficiently coarse), then $x_{2h} = A_{2h}^{-1}d_{2h}$ is often solved in practice with a direct solver, e.g, LU.*

Algorithm 19.20 (Geometric multigrid (GMG) method)**.** *Let $j\in\mathbb{N}$ be the mesh level. For instance $j\hat{=}h$, then $j-1\hat{=}2h$, and so forth. Let $\nu_1,\nu_2\in\mathbb{N}$ be given parameters for the number of Richardson iterations. Moreover, let $\gamma\in\mathbb{N}$ be the cycle form parameter (often $\gamma=1$ or $\gamma=2$). The following method computes the new iterate x_j^{k+1} on mesh level j:*

```
$GMG(j, x_j^k, b_j)$
{
$if\ (j == 0)$
  {
    $x_j^k = A_j^{-1} b_j$   Solve on coarsest mesh with direct solve for instance
    $return\ x_j^k$
  }
$x_j^{k,0} = x_j^k$
$for\ (i = 1, \ldots, \nu_1)$
  $x_j^{k,i/\nu} = x_j^{k,(i-1)/\nu} + \omega(b_j - A_j x_j^{k,(i-1)/\nu})$   Pre-smoothing
$d_j = b_j - A_j x_j^{k,1}$   Defect on fine grid
$d_{j-1} = R_{j-1}^j d_j$   Restriction to coarse grid
$x_{j-1} = 0$   Initial value for coarse grid solution
//Coarse grid soluton now with multigrid
$if\ (j == 1)$
  $\bar\gamma = 1$
$else$
  $\bar\gamma = \gamma$
$for\ (i = 1, \ldots, \bar\gamma)$
  $x_{j-1} = GMG(j-1, x_{j-1}, d_{j-1})$
$y_j = (R_{j-1}^j)^T x_{j-1}$   Prolongation to fine grid
$x_j^{k,2} = x_j^{k,1} + y_j$   Coarse grid correction applied to fine grid solution
$for\ (i = 1, \ldots, \nu_2)$
  $x_j^{k,2+i/\nu} = x_j^{k,2+(i-1)/\nu} + \omega(b_j - A_j x_j^{k,2+(i-1)/\nu})$   Post-smoothing
$return\ x_j^{k,3}$
}
```

The choice of γ gives the GMG method specific names. For instance $\gamma = 1$ results in a so-called V cycle. The choice $\gamma = 2$ is known as W cycle. Here, we do one V cycle from j to 0 and back to $j-1$ and do a second V cycle from $j-1$ to 0 back to j.

Usage of MG in practice

In practice, multigrid solvers can be used as solvers for $Ax = b$ themselves. Or, which happens more often, multigrid methods are used a preconditioner (Section 7.2.3) for other iterative methods, e.g., CG or GMRES. The performance of the latter is shown in Section 19.3.5 in which CG is used as linear solver, preconditioned with MG and SOR (symmetric overrelaxation) as smoother.

Convergence analysis of the W cycle

For full proofs, we refer to [3, 4, 15]. Still following [3], we concentrate on the W cycle because the proofs are relatively simple. The proofs of the V cycle are more involved requiring sufficient smoothing steps and H^2 regularity.

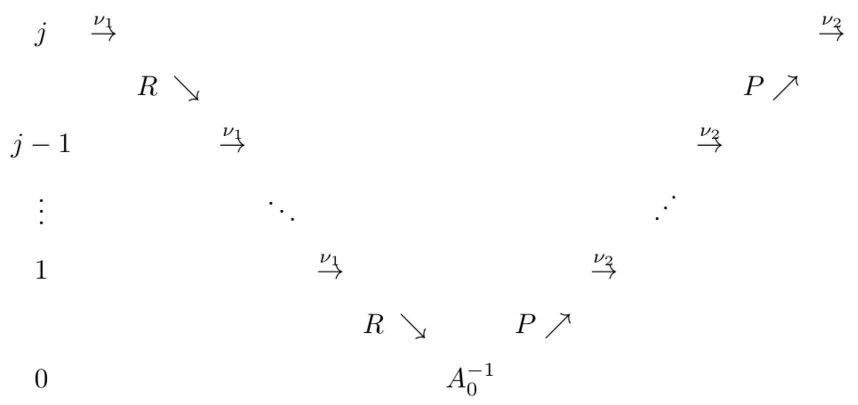


Figure 19.14: The V cycle.

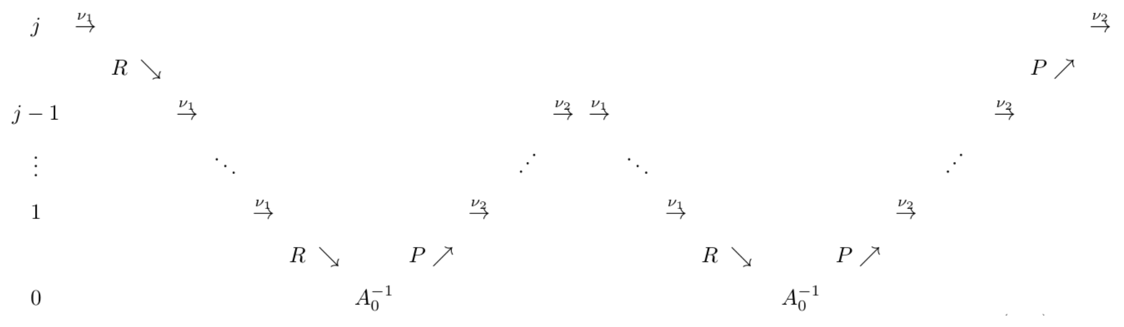


Figure 19.15: The W cycle.

Let j be the mesh index and k the iteration index. The goal is to derive an estimation with the multigrid convergence rate ρ_j on $j+1$ meshes:

$$\|u^{j,k+1} - u_j\| \leq \rho_j \|u^{j,k} - u_j\|$$

where $u_j \in V_j$ is the discrete solution on mesh level j and $u^{j,k+1}$ is the $k+1$-th iterate. Specifically, the multigrid method is convergent when $\rho_j < 1$ and ρ_j is independent of the mesh size h. Then, the computational cost would not increase while refining the mesh. The factor ρ_j is the convergence rate. Of course, $\rho_j \ll 1$, the faster the convergence.

Error recursion for the smoothing step

We have:

$$e^{k,1} = x_h - x_h^{k,1} = (I_h - \omega A_h)^{\nu_1}(x_h - x_h^k) = S_h^{\nu_1} e^k$$

Coarse grid correction step

Then:

$$\begin{aligned}
e^{k+1} &= x_h - x_h^{k+1} \\
&= x_h - \Big(x_h^{k,1} + (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h (b_h - A_h x_h^{k,1}) \Big) \\
&= e^{k,1} - (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h A_h e^{k,1} \\
&= (I_h - (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h A_h) e^{k,1}
\end{aligned}$$

With this, we obtain for the iteration matrix:

$$e^{k+1} = (I_h - (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h A_h) S_h^{\nu_1} e^k = (A_h^{-1} - (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h) A_h S_h^{\nu_1} e^k$$

This allows us to split the error into two parts:

$$\|e^{k+1}\| \le \underbrace{\|(A_h^{-1} - (R_{2h}^h)^T A_{2h}^{-1} R_{2h}^h)\|}_{\text{Approximation}} \underbrace{\|A_h S_h^{\nu_1}\|}_{\text{Smoothing}} \|e^k\|.$$

The appropriate norm is the Euclidian norm as it will turn out.

Norms We recall that A is s.p.d. (only necessary for $s=1$ in the following). Thus we define (similar to the CG method):

$$\|x\|_s := \sqrt{(}x, A^s x), \quad s = 0,1,2.$$

In detail:

$$\begin{aligned} s=0: &\quad \|x\|_0 = \sqrt{(}(x,x)) \quad \text{Euclidian norm}\\ s=1: &\quad \|x\|_1 = \sqrt{(}(x,Ax)) \quad \text{Energy norm}\\ s=2: &\quad \|x\|_2 = \sqrt{(}(Ax,Ax)) \quad \text{Defect norm} \end{aligned}$$

These definitions can be further extended to $s \in \mathbb{R}$ by using a diagonalization of A.

The Sobolev norms $\|\cdot\|_{L^2}$ and $\|\cdot\|_{H^1}$ can be related to $\|\cdot\|_0$ and $\|\cdot\|_1$, respecively. We have:

$$\|x\|_1^2 = (x, Ax) = a(u_h, u_h)$$

As in the Lax-Milgram lemma, cf. Section 19.1.5, we obtain:

$$\alpha \|u_h\|_{H^1} \le \|x\|_1^2 = a(u_h, u_h) \le \gamma \|u_h\|_{H^1}.$$

As seen in the stability estimate of the proof of the Lax-Milgram lemma, these estimates are independent of the basis of V_h and only use approximation properties.

Lemma 19.21. *Let $\{\varphi_1, \ldots, \varphi_N\}$ be the Lagrange basis of V_h on a family of uniform and shape regular triangulations. Let x be the coefficient vector of $v_h \in V_h$. Then, there exist $c_1, c_2 > 0$ independent of h, but dependent on the overall mesh $\mathcal{T}_h$ such that*

$$c_1 h^{n/2} \|x\|_0 \le \|v\|_{L^2} \le c_2 h^{n/2} \|x\|_0.$$

Here n is the space dimension.

Approximation properties

Lemma 19.22 (Approximation property). *Let $A_h x_h = b_h$ the discrete problem on the finest grid. Moreover, let $x_h^{k,1}$ denote the iterate after smoothing and $x_h^{k,2}$ denotes the iterate after the coarse grid correction. If the mesh is uniform and shape-regular, the variational problem is H^2 regular and we have*

$$\|x_h - x_h^{k,2}\|_0 \le c h^{2-n} \|x_h - x_h^{k,1}\|_2$$

As before, n is the space dimension. We see that this estimate is not robust w.r.t. to n.

Smoothing properties

Lemma 19.23 (Smoothing property). *Let A_h be s.p.d. The Richardson iteration*

$$x_h^{k+1} = x_h^k + \omega(b_h - A_h x_h^k)$$

with $\omega = (\lambda_{max}(A))^{-1}$ yields the following estimate:

$$\|x_h - x_h^\nu\|_2 \le \frac{\lambda_{max}(A_h)}{\nu} \|x_h - x_h^0\|_0.$$

Two- and multigrid convergence results

Theorem 19.24. *With all assumptions from before, the two-grid method with ν steps of Richardson iterations as smoother, satisfies*

$$\|x_h - x_h^{k+1}\|_0 \leq \frac{c}{\nu}\|x_h - x_h^0\|_0$$

To this end, for sufficiently large ν, the two-grid method converges independendly of the mesh size h.

Lemma 19.25 (Recursion; [3] or [4][Chapter V.3)**.** *Let ρ_1 be the convergence rate of the two-grid method and ρ_l the convergence rate of a multigrid method with the cycle parameter γ and $l \geq 2$ levels. Then, the following recursion holds true*

$$\rho_l \leq \rho_1 + (1+\rho_1)\rho_{l-1}^{\gamma}$$

Theorem 19.26 (Convergence ρ_l rate W cycle)**.** *If $\rho_1 \leq \frac{1}{5}$ for the two-grid method, then it holds for the W cycle of the multigrid method*

$$\rho_l \leq \frac{5}{3}\rho_1 \leq \frac{1}{3} \quad \textit{for } l = 2, 3, \ldots$$

Lemma 19.27 (Optimal complexity of the multigrid method; [3])**.** *Let N_l the number of unknown on level l using P_1 finite elements. Then, the amount of arithemetic operations A_l on level l is*

$$A_l = O(N_l).$$

19.3 Numerical tests: computational convergence analysis and solver behavior

We finish this chapter with several numerical tests in 1D, 2D and 3D.

19.3.1 2D Poisson

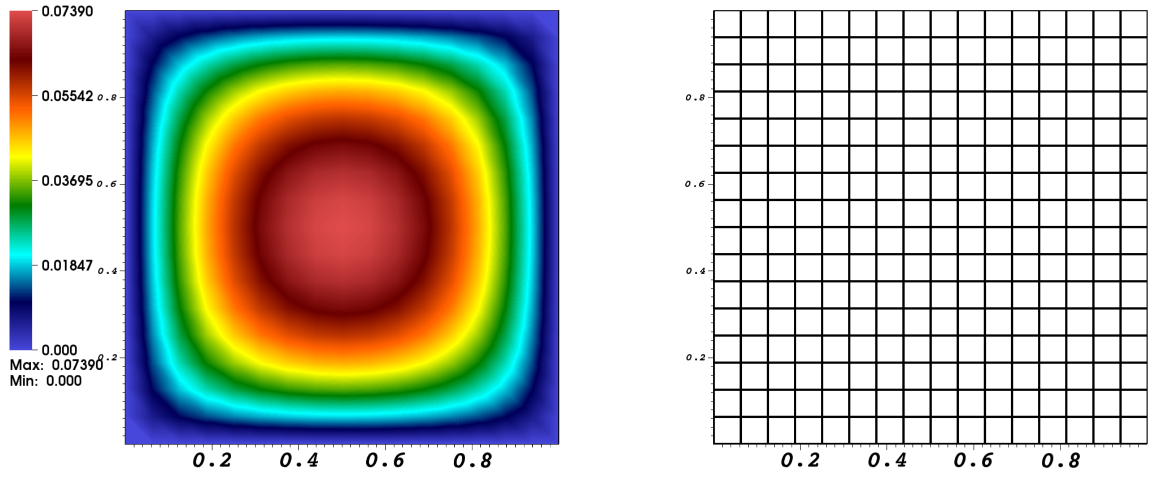


Figure 19.16: The graphical solution (left) and the corresponding mesh (right).

We compute Poisson in 2D on $\Omega = (0,1)^2$ and homogeneous Dirichlet conditions on $\partial\Omega$. The force is $f = 1$. We use a quadrilateral mesh using Q_1 elements (in an isoparametric FEM framework). The number of mesh elements is 256 and the number of DoFs is 289. We need 26 CG iterations (see Section 7.2.2 for the development of the CG scheme), without preconditioning, for the linear solver to converge.

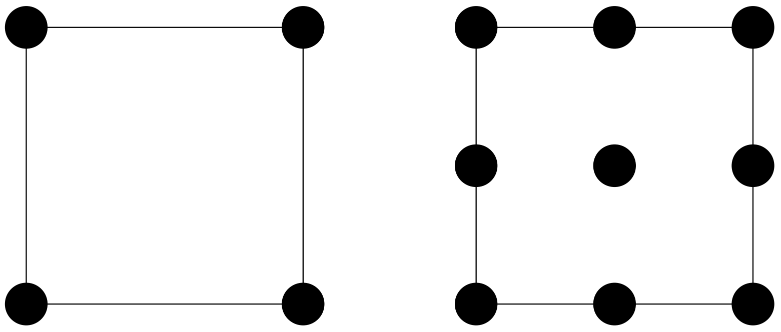


Figure 19.17: Q_1 (bilinear) and Q_2 (biquadratic) elements on a quadrilateral in 2D.

19.3.2 Numerical test: 3D

```
Number of elements:   4096
Number of DoFs:       4913
Number of iterations: 25 CG steps without preconditioning.
```

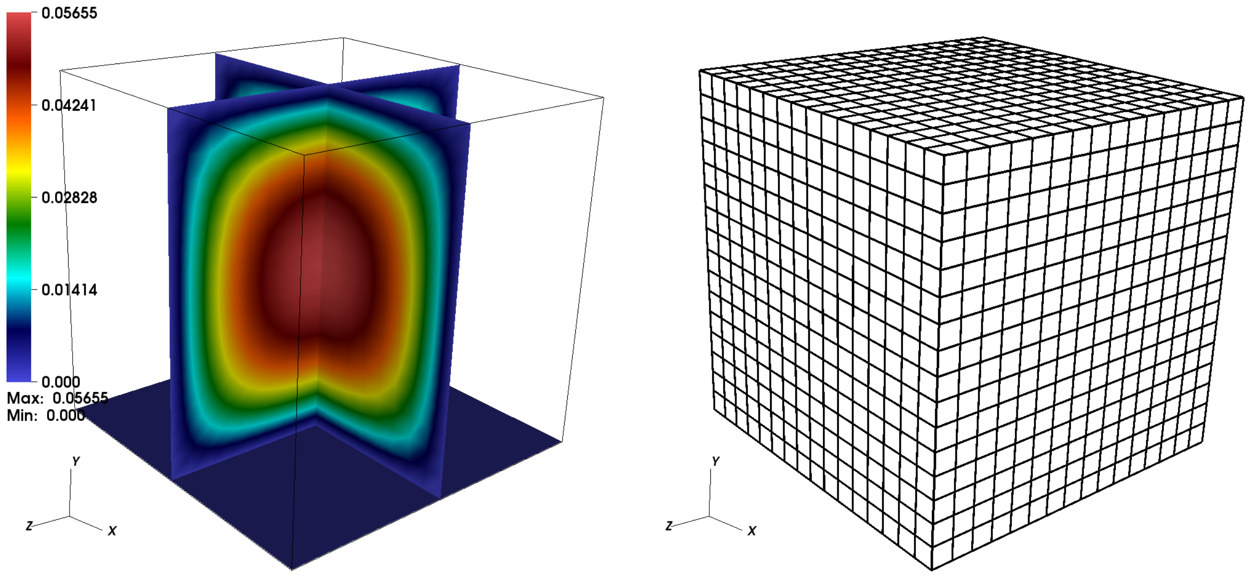


Figure 19.18: The graphical solution (left) and the corresponding mesh (right).

19.3.3 Checking programming code and convergence analysis for linear and quadratic FEM

We present a general algorithm and present afterwards 2D results.

Algorithm 19.28. *Given a PDE problem. E.g. $-\Delta u = f$ in Ω and $u = 0$ on the boundary $\partial\Omega$.*

1. *Construct by hand a solution u that fulfills the boundary conditions.*
2. *Insert u into the PDE to determine f.*
3. *Use that f in the finite element simulation to compute numerically u_h.*

4. *Compare $u - u_h$ in relevant norms (e.g., L^2, H^1).*

5. *Check whether the desired h powers can be obtained for small h.*

We demonstrate the previous algorithm for a $2D$ case in $\Omega = (0,\pi)^2$:

$$\begin{aligned} -\Delta u(x,y) &= f \quad \text{in } \Omega, \\ u(x,y) &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

and we constuct $u(x,y) = \sin(x)\sin(y)$, which fulfills the boundary conditions (trivial to check! But please do it!). Next, we compute the right hand side f:

$$-\Delta u = -(\partial_{xx}u + \partial_{yy}u) = 2sin(x)sin(y) = f(x,y).$$

2D Poisson: Linear FEM We then use f in the above program from Section 19.3.1 and evaluate the L^2 and H^1 norms using linear FEM. The results are:

```
Level    Elements    DoFs        h                L2 err         H1 err
=============================================================================
2        16          25          1.11072          0.0955104      0.510388
3        64          81          0.55536          0.0238811      0.252645
4        256         289         0.27768          0.00597095     0.126015
5        1024        1089        0.13884          0.00149279     0.0629697
6        4096        4225        0.06942          0.0003732      0.0314801
7        16384       16641       0.03471          9.33001e-05    0.0157395
8        65536       66049       0.017355         2.3325e-05     0.00786965
9        262144      263169      0.00867751       5.83126e-06    0.00393482
10       1048576     1050625     0.00433875       1.45782e-06    0.00196741
11       4194304     4198401     0.00216938       3.64448e-07    0.000983703
=============================================================================
```

In this table we observe that we have quadratic convergence in the L^2 norm and linear convergence in the H^1 norm. For a precise (heuristic) computation, we also refer to Chapter 19.5 in which a formula for computing the convergence orders α is derived.

2D Poisson: Quadratic FEM We use again f in the above program from Section 19.3.1 and evaluate the L^2 and H^1 norms using quadratic FEM. The results are:

```
Level    Elements    DoFs        h                L2 err         H1 err
=============================================================================
2        16          81          1.11072          0.00505661     0.0511714
3        64          289         0.55536          0.000643595    0.0127748
4        256         1089        0.27768          8.07932e-05    0.00319225
5        1024        4225        0.13884          1.01098e-05    0.000797969
6        4096        16641       0.06942          1.26405e-06    0.000199486
7        16384       66049       0.03471          1.58017e-07    4.98712e-05
8        65536       263169      0.017355         1.97524e-08    1.24678e-05
9        262144      1050625     0.00867751       2.46907e-09    3.11694e-06
10       1048576     4198401     0.00433875       3.08687e-10    7.79235e-07
11       4194304     16785409    0.00216938       6.14696e-11    1.94809e-07
=============================================================================
```

In this table we observe that we have cubic convergence $O(h^3)$ in the L^2 norm and quadratic convergence $O(h^2)$ in the H^1 norm. This confirms the theory; see for instance [4]. For a precise (heuristic) computation, we also refer to Chapter 19.5 in which a formula for computing the convergence orders α is derived.

We next plot the DoFs versus the errors in Figure 19.19 in order to highlight the convergence orders. For the relationship between h and DoFs versus the errors in different dimensions, we refer again to Chapter 19.5.

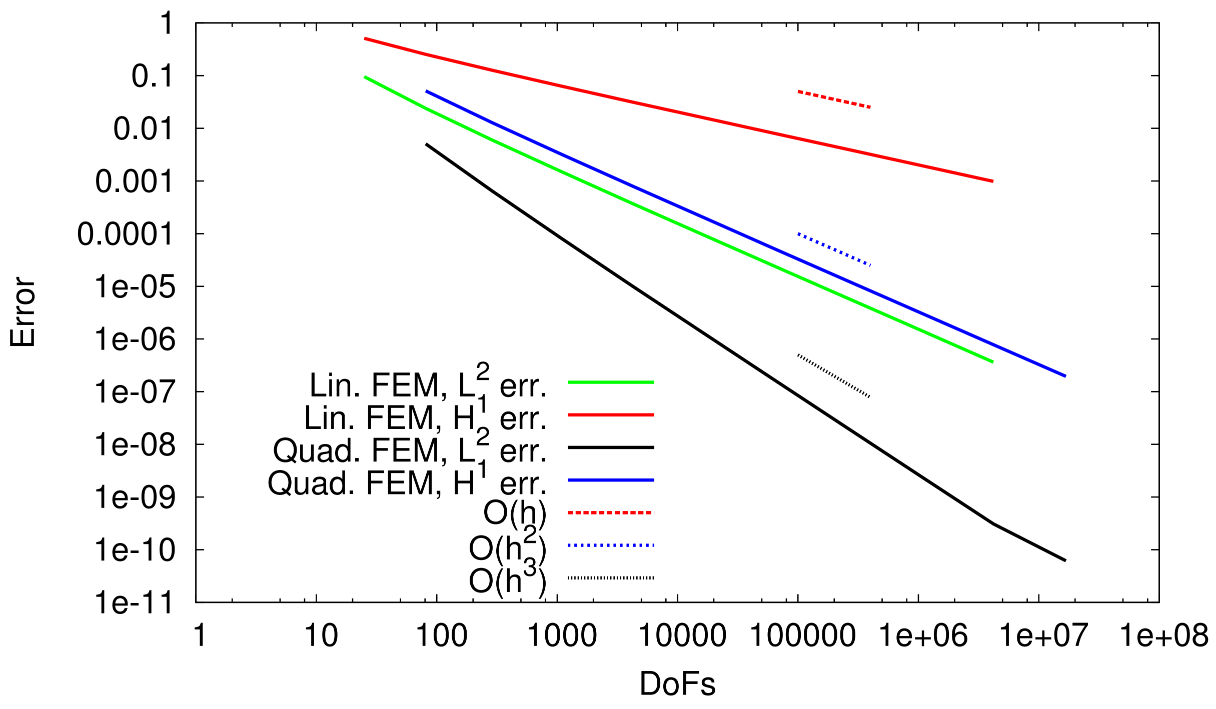


Figure 19.19: Plotting the DoFs versus the various errors for the 2D Poisson test using linear and quadratic FEM. We confirm numerically the theory: we observe $\|u-u_h\|_{L^2} = O(h^{r+1})$ and $\|u-u_h\|_{H^1} = O(h^r)$ for $r = 1, 2$, where r is the FEM degree.

19.3.4 Convergence analysis for 1D Poisson using linear FEM

We continue Section 19.1.6. As manufactured solution we use

$$u(x) = \frac{1}{2}(-x^2 + x),$$

on $\Omega = (0, 1)$ with $u(0) = u(1) = 0$, which we derived in Section 19.1.2. In the middle point, we have $u(0.5) = -0.125$ (in theory and simulations). In the following table, we plot the L^2 and H^1 error norms, i.e., $\|u - u_h\|_X$ with $X = L^2$ and $X = H^1$, respectively.

```
Level    Elements    DoFs         h               L2 err          H1 err
==============================================================================
1        2           3            0.5             0.0228218       0.146131
2        4           5            0.25            0.00570544      0.072394
3        8           9            0.125           0.00142636      0.0361126
4        16          17           0.0625          0.00035659      0.0180457
5        32          33           0.03125         8.91476e-05     0.00902154
6        64          65           0.015625        2.22869e-05     0.0045106
7        128         129          0.0078125       5.57172e-06     0.00225528
8        256         257          0.00390625      1.39293e-06     0.00112764
9        512         513          0.00195312      3.48233e-07     0.000563819
10       1024        1025         0.000976562     8.70582e-08     0.000281909
==============================================================================
```

19.3.5 Iterative solver behavior

We continue the numerical tests from Section 19.3 and consider again the Poisson problem in 2D and 3D on the unit square and with force $f = 1$ and homogeneous Dirichlet conditions. We use as solvers:

- CG
- PCG with SSOR preconditioning and $\omega = 1.2$
- PCG with geometric multigrid preconditioning and SOR smoother
- GMRES
- GMRES with SSOR preconditioning and $\omega = 1.2$
- BiCGStab
- BiCGStab with SSOR preconditioning and $\omega = 1.2$

The tolerance is chosen as $TOL = 1.0e-12$. We also run on different mesh levels in order to show the dependency on n.

```
Dimension   Elements   DoFs(N) CG   PCG  PCG-GMG  GMRES P-GMRES  BiCGStab P-BiCGStab
====================================================================================
2            16         25      3        5
2            64         81     10        6
2           256        289     23   19   6         23   18       16       12
2          1024       1089     47   33   6         83   35       33       21
2          4096       4225     94   60   6        420   78       66       44
2         16384      16641    186        6
2         65536      66049    370        6
2        262144     263169    731        5
2       1048576    1050625   1400        5
2       4194304    4198401   2780        5
------------------------------------------------------------------------------------
3          4096       4913     25   19             25   21       16       11
3         32768      35937     51   32             77   38       40       23
3        262144     274625     98   57            307   83       69       46
====================================================================================
```

Four main messages:

- GMRES performs worst.
- PCG performs better than pure CG, but less significant than one would wish.
- BiCGStab performs very well - a bit surprising.
- Geometric multigrid preconditioning yields mesh-independent constant iteration numbers of the outer CG iteration

19.4 Discretization of the heat equation (complement)

In order to further apply our numerical concepts developed in the previous sections we consider the heat equation and perform a numerical discretization. The heat equation is a time-dependent PDE problem and requires discretization in space and time. Often in the literature, temporal discretization is based on FD whereas spatial discretization is based on the FEM.

The Rothe method, also known as as horizontal method of lines, is a strategy to discretize a PDE in time t and space (x, y, z). In particular the PDE is first discretized in time with a finite difference scheme (thus we first need the methodology developed in Chapter 17). In the second step, the problem is discretized in space with the finite element method developed in Chapter 19.1.

19.4.1 Problem

Compute the heat distribution in a room. Let us first develop a model. We assume that we work on a macroscopic level and can assume continuum mechanics [17]. We work with first principle laws: namely conservation of mass, momentum, angular momentum and energy. This would yield four different equations but actually we will have much less since certain assumptions can be made.

Furthermore, we need to gather initial and boundary conditions:

- What is the temperature in the room at the beginning?
- What is the temperature at the walls?

To illustrate the concepts we work in 1D (one spatial dimension) in the following.

19.4.2 Mathematical problem statement

Let Ω be an open, bounded subset of $\mathbb{R}^d, d = 1$ and $I := (0, T]$ where $T > 0$ is the end time value. The IBVP (initial boundary-value problem) reads:

Formulation 19.29. *Find $u := u(x, t) : \Omega \times I \to \mathbb{R}$ such that*

$$\begin{aligned} \rho\partial_t u - \nabla \cdot (\alpha \nabla u) &= f \quad \text{in } \Omega \times I, \\ u &= a \quad \text{on } \partial\Omega \times [0, T], \\ u(0) &= g \quad \text{in } \Omega \times t = 0, \end{aligned}$$

where $f : \Omega \times I \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$ and $\alpha \in \mathbb{R}$ and ρ are material parameters, and $a \geq$ is a Dirichlet boundary condition. More precisely, g is the initial temperature and a is the wall temperature, and f is some heat source.

19.4.3 Temporal discretization

We first discretize in time. Here we use finite differences and more specifically we introduce the so-called **One-Step-θ** scheme, which allows for a compact notation for three major finite difference schemes: forward Euler ($\theta = 0$), backward Euler ($\theta = 1$), and trapezoidal rule ($\theta = 0.5$) well known from numerical methods for ODEs.

Definition 19.13 (Choice of θ)**.** *By the choice of θ, we obtain the following time-stepping schemes:*

- $\theta = 0$*: 1st order explicit Euler time stepping;*
- $\theta = 0.5$*: 2nd order Crank-Nicolson (trapezoidal rule) time stepping;*
- $\theta = 0.5 + k_n$*: 2nd order shifted Crank-Nicolson which is shifted by the time step size $k_n = t^n - t^{n-1}$ towards the implicit side;*
- $\theta = 1$*: 1st order implicit Euler time stepping.*

Remark 19.28. *We notice that for problems with large time steps ($k \geq 0.5$), we need to normalize/non-dimensionalize the equations in order to get the characteristic time step size that can be used for the shifted variant. Otherwise, you have $\theta > 1$, which is senseless. As alternative, one can use the Rannacher time-stepping by adding backward Euler steps on a regular basis.*

To have good stability properties (namely A-stability) of the time-stepping scheme is important for temporal discretization of partial differential equations. Often (as here for the heat equation) we deal with second-order operators in space, such PDE-problems are generically (very) stiff with the order $O(h^{-2})$, where h is the spatial discretization parameter.

After these preliminary considerations, let us now discretize in time the above problem. We first create a time grid of the time domain $I = [0, T]$ with N_T intervals and a time step size $k = \frac{T}{N_T}$:

$$\begin{aligned}
&\partial_t u - \nabla \cdot (\alpha \nabla u) = f \\
\Rightarrow \quad & \frac{u^n - u^{n-1}}{k} - \theta \nabla \cdot (\alpha \nabla u^n) - (1-\theta) \nabla \cdot (\alpha \nabla u^{n-1}) = \theta f^n + (1-\theta) f^{n-1} \\
\Rightarrow \quad & u^n - k\theta \nabla \cdot (\alpha \nabla u^n) = u^{n-1} + k(1-\theta) \nabla \cdot (\alpha \nabla u^{n-1}) + k\theta f^n + k(1-\theta) f^{n-1}
\end{aligned}$$

where $u^n := u(t^n)$ and $u^{n-1} := u(t^{n-1})$ and $f^n := f(t^n)$ and $f^{n-1} := f(t^{n-1})$.

Spatial discretization

We take the temporally discretized problem and use a Galerkin finite element scheme to discretize in space. That is we multiply with a test function and integrate. We then obtain: Find $u_h \in \{a + V_h\}$ such that for $n = 1, 2, \dots N_T$:

$$(u_h^n, \varphi_h) - k\theta(\alpha \nabla u_h^n, \nabla \varphi_h) = (u^{n-1}, \varphi_h) + k(1-\theta)(\alpha \nabla u^{n-1}, \nabla \varphi_h) + k\theta(f^n, \varphi_h) + k(1-\theta)(f^{n-1}, \varphi_h)$$

for all $\varphi_h \in V_h$.

Definition 19.14 (Specification of the problem data)**.** *For simplicity let us assume that there is no heat source $f = 0$ and the material coefficients are specified by $\alpha = 1$ and $\rho = 1$.*

Furthermore, let us work with the explicit Euler scheme $\theta = 0$. Then we obtain:

$$(u_h^n, \varphi_h) = (u_h^{n-1}, \varphi_h) - k(\nabla u_h^{n-1}, \nabla \varphi_h)$$

The structure of this equation, namely

$$y^n = y^{n-1} + kf(t^{n-1}, y^{n-1}),$$

is the same as we know from ODEs. This means that we seek at each time t^n a finite element solution u_h^n on the spatial domain Ω. With the help of the basis functions $\varphi_{h,j}$ we can represent the spatial solution:

$$u_h^n(x) := \sum_{j=1}^{N_x} u_{h,j} \varphi_{h,j}, \quad u_{h,j} \in \mathbb{R}.$$

Then:

$$\sum_{j=1}^{N_x} u_{h,j}^n (\varphi_{h,j}, \varphi_{h,i}) = (u^{n-1}, \varphi_{h,i}) - k(\nabla u_h^{n-1}, \nabla \varphi_{h,i})$$

which results in a linear equation system: $MU = B$. Here

$$\begin{aligned}
(M_{ij})_{i,j=1}^{N_x} &:= (\varphi_{h,j}, \varphi_{h,i}), \\
(U_j)_{j=1}^{N_x} &:= (u_{h,1}, \dots, u_{h,N}) \\
(B_i)_{i=1}^{N_x} &:= (u_h^{n-1}, \varphi_{h,i}) - k(\nabla u_h^{n-1}, \nabla \varphi_{h,i}).
\end{aligned}$$

Furthermore the old time step solution can be written as well in matrix form:

$$(u^{n-1}, \varphi_{h,i}) = \sum_{j=1}^{N_x} u_{h,j}^{n-1} \underbrace{(\varphi_{h,j}, \varphi_{h,i})}_{=:M}$$

and similarly for the Laplacian term:

$$(\nabla u^{n-1}, \nabla\varphi_{h,i}) = \sum_{j=1}^{N_x} u_{h,j}^{n-1} \underbrace{(\nabla\varphi_{h,j}, \nabla\varphi_{h,i})}_{=:K}.$$

The mass matrix M (also known as the Gramian matrix) is always the same for fixed h and can be explicitely computed.

Proposition 19.30. *Formally we arrive at: Given u_h^{n-1}, find u_h^n, such that*

$$Mu_h^n = Mu_h^{n-1} - kKu_h^{n-1} \quad \Rightarrow \quad u_h^n = u_h^{n-1} - kM^{-1}Ku_h^{n-1}.$$

for $n = 1, \ldots, N_T$.

Remark 19.29. *We emphasize that the forward Euler scheme is not recommended to be used as time stepping scheme for solving PDEs. The reason is that the stiffness matrix is of order $\frac{1}{h^2}$ and one is in general interested in $h \to 0$. Thus the coefficients become very large for $h \to 0$ resulting in a stiff system. For stiff systems, as we learned before, one should better use implicit schemes, otherwise the time step size k has to be chosen too small in order to obtain stable numerical results; see the numerical analysis in Section 19.4.4.*

Evaluation of the integrals (1D in space)

We need to evaluate the integrals for the stiffness matrix K:

$$K = (K_{ij})_{ij=1}^{N_x} = \int_\Omega \varphi_{h,j}'(x)\varphi_{h,i}'(x)\,dx = \int_{x_{j-1}}^{x_{j+1}} \varphi_{h,j}'(x)\varphi_{h,i}'(x)\,dx$$

resulting in

$$A = h^{-1}\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix}$$

Be careful and do not forget the material parameter α in case it is not $\alpha = 1$.

For the mass matrix M we obtain:

$$M = (M_{ij})_{ij=1}^{N_x} = \int_\Omega \varphi_{h,j}(x)\varphi_{h,i}(x)\,dx = \int_{x_{j-1}}^{x_{j+1}} \varphi_{h,j}(x)\varphi_{h,i}(x)\,dx.$$

Specifically on the diagonal, we calculate:

$$\begin{aligned} M_{ii} &= \int_\Omega \varphi_{h,i}(x)\varphi_{h,i}(x)\,dx = \int_{x_{i-1}}^{x_i} \Big(\frac{x - x_{i-1}}{h}\Big)^2 dx + \int_{x_i}^{x_{i+1}} \Big(\frac{x_{i+1} - x}{h}\Big)^2 dx \\ &= \frac{1}{h^2}\int_{x_{i-1}}^{x_i} (x - x_{i-1})^2\,dx + \frac{1}{h^2}\int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2\,dx = \frac{h}{3} + \frac{h}{3} \\ &= \frac{2h}{3}. \end{aligned}$$

For the right off-diagonal, we have

$$\begin{aligned}
m_{i,i+1} &= \int_\Omega \varphi_{h,i+1}(x)\varphi_{h,i}(x)\,dx = \int_{x_i}^{x_{i+1}} \frac{x-x_i}{h}\cdot\frac{x_{i+1}-x}{h}\,dx = \int_{x_i}^{x_{i+1}} \frac{x-x_i}{h}\cdot\frac{x_i-x+h}{h}\,dx\\
&= \ldots\\
&= -\frac{h}{3}+\frac{h^2}{2h} = \frac{h}{6}.
\end{aligned}$$

It is trivial to see that $m_{i,i+1} = m_{i-1,i}$. Summarizing all entries results in

$$M = \frac{h}{6}\begin{pmatrix} 4 & 1 & & & 0\\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1\\ 0 & & & 1 & 4 \end{pmatrix}.$$

Final algorithms

We first setup a sequence of discrete (time) points:

$$0 = t_0 < t_1 < \ldots < t_N = T.$$

Furthermore we set

$$I_n = [t_{n-1}, t_n], \quad k_n = t_n - t_{n-1}, \quad k := \max_{1\le n\le N} k_n.$$

Forward (explicit) Euler:

Algorithm 19.31. *Given the initial condition g, solve for $n = 1, 2, 3, \ldots, N_T$*

$$Mu_h^n = Mu_h^{n-1} - kKu_h^{n-1} \quad\Rightarrow\quad u_h^n = u_h^{n-1} - kM^{-1}Ku_h^{n-1},$$

where $u_h^n, u_h^{n-1} \in \mathbb{R}^{N_x}$.

Algorithm 19.32. *The backward (implicit) Euler scheme reads:*

$$Mu_h^n + kKu_h^n = Mu_h^{n-1}$$

An example of a pseudo C++ code is

```
final_loop()
{
 // Initalization of model parameters, material parameters, and so forth
 set_runtime_parameters();

 // Make a mesh - decompose the domain into elements
 create_mesh();

 apply_initial_conditions();

 // Timestep loop
  do
    {
      std::cout << "Timestep " << timestep_number << " (" << time_stepping_scheme
                << ")" <<    ": " << time << " (" << timestep << ")"
                << "\n=================================================================="
                << std::endl;
      std::cout << std::endl;

      // Solve for next time step: Assign previous time step solution u^{n-1}
      old_timestep_solution = solution;

      // Assemble FEM matrix A and right hand side f
      assemble_system ();

      // Solve the linear equation system Ax=b
      solve ();

      // Update time
      time += timestep;

      // Write solution into a file or similar
      output_results (timestep_number);

      // Increment n->n+1
      ++timestep_number;
    }
  while (timestep_number <= max_no_timesteps);
}
```

19.4.4 Numerical analysis: stability analysis

The heat equation is a good example of a stiff problem (for the definition we refer back to Section 17.2) and therefore, implicit methods are preferred in order to avoid very, very small time steps in order to obtain stability. We substantiate this claim in the current section.

The model problem is

$$\begin{aligned} \partial_t u - \Delta u &= 0 \quad \text{in } \Omega \times I \\ u &= 0 \quad \text{on } \partial\Omega \times I, \\ u(0) &= u^0 \quad \text{in } \Omega \times \{0\}. \end{aligned}$$

Spatial discretization yields:

$$\begin{aligned} (\partial_t u_h, \phi_h) + (\nabla u_h, \nabla \phi_h) &= 0, \\ (u_h^0, \phi_h) &= (u^0, \phi_h). \end{aligned}$$

Backward Euler time discretization yields:

$$(u_h^n, \phi_h) + k(\nabla u_h^n, \nabla \phi_h) = (u_h^{n-1}, \phi_h)$$

It holds

Proposition 19.33. *A basic stability estimate for the above problem is:*

$$\|u_h^n\| \le \|u_h^0\| \le \|u^0\| \tag{19.42}$$

Proof. For the stability estimate we proceed as earlier and make a special choice for a test function: $\phi_h := u_h^n$. We then obtain:

$$\|u_h^n\|_{L^2}^2 + k|\nabla u_h^n|_{H^1}^2 = (u_h^{n-1}, u_h^n) \le \frac{1}{2}\|u_h^{n-1}\|_{L^2}^2 + \frac{1}{2}\|u_h^n\|_{L^2}^2$$

which yields:

$$\frac{1}{2}\|u_h^n\|_{L^2}^2 - \frac{1}{2}\|u_h^{n-1}\|_{L^2}^2 + k|\nabla u_h^n|_{H^1}^2 \le 0, \quad \forall n = 1, \ldots, N.$$

Of course it further holds:

$$\begin{aligned} &\frac{1}{2}\|u_h^n\|_{L^2}^2 - \frac{1}{2}\|u_h^{n-1}\|_{L^2}^2 \le 0, \quad \forall n = 1, \ldots, N, \\ \Leftrightarrow \quad &\frac{1}{2}\|u_h^n\|_{L^2}^2 \le \frac{1}{2}\|u_h^{n-1}\|_{L^2}^2 \end{aligned}$$

Taking the square root and applying the result back to $n = 0$ yields

$$\frac{1}{2}\|u_h^n\|_{L^2} \le \frac{1}{2}\|u_h^0\|_{L^2}$$

It remains to show the second estimate:

$$\|u_h^0\| \le \|u^0\|$$

In the initial condition we choose the test function $\varphi := u_h^0$ and obtain:

$$(u_h^0, u_h^0) = (u^0, u_h^0) \le \frac{1}{2}\|u^0\| + \frac{1}{2}\|u_h^0\|$$

which shows the assertion. □

Stability analysis for backward Euler

A more detailed stability analysis that brings in the spatial discretization properties is based on the matrix notation and is as follows. We recapitulate and start from (see Section 19.4.3):

$$Mu_h^n + kKu_h^n = Mu_h^{n-1}$$

Then:

$$\begin{aligned} Mu_h^n + kKu_h^n &= Mu_h^{n-1} \\ \Rightarrow u_h^n + kM^{-1}Ku_h^n &= u_h^{n-1} \\ \Rightarrow (I + kM^{-1}K)u_h^n &= u_h^{n-1} \end{aligned}$$

Compare to the ODE notation:

$$(1 + z)y^n = y^{n-1}$$

Thus, the role of $z = k\lambda$ is taken by $kM^{-1}K$ thus $\lambda \sim M^{-1}K$. We know for the condition numbers

$$\kappa(M) = O(1), \qquad \kappa(K) = O(h^{-2}), \quad h \to 0.$$

We proceed similar to ODEs, and obtain

$$u_h^n = (I + kM^{-1}K)^{-1}u_h^{n-1}$$

The goal is now to show that

$$(I + kM^{-1}K)^{-1} \le 1.$$

To this end, let $\mu_j, j = 1, \ldots, M = dim(V_h)$ be the eigenvalues of the matrix $M^{-1}K$ ranging from the smallest eigenvalue $\mu_1 \sim O(1)$ to the largest eigenvalue $\mu_M \sim O(h^{-2})$; for a proof, we refer the reader to [18][Section 7.7]. Then:

$$|(I + kM^{-1}K)^{-1}| = \max_j \frac{1}{1 + k\mu_j} = \frac{1}{1 + k\mu_M}.$$

To have a stable scheme, we must have:

$$\frac{1}{1+k\mu_M} < 1.$$

We clearly have then again from (19.42):

$$\|u_h^n\| \leq \|u_h^{n-1}\| \quad \Rightarrow \quad \|u_h^n\| \leq \|u_h^0\|$$

Furthermore, we see that the largest eigenvalue μ_M will increase as $O(h^{-2})$ when h tends to zero. Using the backward Euler or Crank-Nicolson scheme, this will be not a problem and both schemes are **unconditionally stable.**

Stability analysis for explicit Euler

Here, the stability condition reads:

$$u_h^n = (I - kM^{-1}K)u_h^{n-1}$$

yielding

$$|(I - kM^{-1}K)^{-1}| = |\max_j(1 - k\mu_j)| = |1 - k\mu_M|.$$

As before $\mu_M = O(h^{-2})$. To establish a stability estimate of the form $\|u_h^n\| \leq \|u_h^{n-1}\|$, it is required that

$$|1 - k\mu_M| \leq 1.$$

Resolving this estimate, we obtain

$$k\mu_M \leq 2 \quad \Leftrightarrow \quad k \leq \frac{2}{\mu_M} = O(h^2),$$

thus

$$k \leq Ch^2, \quad C > 0. \tag{19.43}$$

Here, we only have **conditional stability in time**, namely the time step size k depends on the spatial discretization parameter h. Recall that in terms of accuarcy and discretization errors we want to work with small h, then the time step size has to be extremely small in order to have a stable scheme. In practice this is in most cases not attrative at all and for this reason the forward Euler scheme does not play a role despite that the actual solution is cheaper than solving an implicit scheme.

Numerical tests

The series of numerical tests in this section has three goals:

- to show some figures to get an impression about simulation results;
- to show computationally that the stability results are indeed satisfied or violated;
- to show computationally satisfaction of the (discrete) maximum principle.

We consider the heat equation

$$\begin{aligned} \partial_t u - \Delta u &= 0, \quad \text{in } \Omega \times I \\ u &= 0 \quad \text{on } \partial\Omega \times I, \\ u(x,0) &= \sin(x)\sin(y) \quad \text{in } \Omega \times \{0\}. \end{aligned}$$

We compute 5 time steps, i.e., $T = 5s$, with time step size $k = 1s$. The computational domain is $\Omega = (0,\pi)^2$. We use a One-step-theta scheme with $\theta = 0$ (forward Euler) and $\theta = 1$ (backward Euler).

The graphical results are displayed in the Figures 19.20 - 19.23. Due to stability reasons and violation of the condition $k \leq ch^2$, the forward Euler scheme is unstable (Figures 19.22 - 19.23). By reducing the time step size to $k = 0.01s$ (the critical time step value could have been computed - the value $k = 0.01s$ has been found by trial and error simply), we obtain stable results for the forward Euler scheme. These findings are displayed in Figure 19.24. However, to reach $T = 5s$, we need to compute 500 time steps, which is finally more expensive, despite being an explicit scheme, than the implicit Euler method.

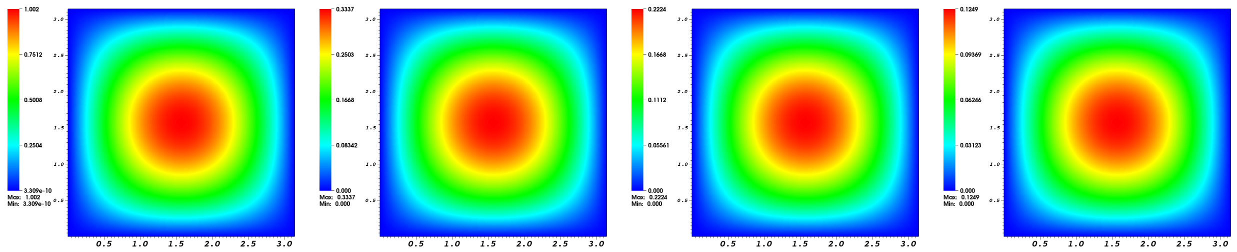


Figure 19.20: Heat equation with $\theta = 1$ (backward Euler) at $T = 0, 1, 2, 5$. The solution is stable and satisfies the parabolic maximum principle. The color scale is adapted at each time to the minimum and maximum values of the solution.

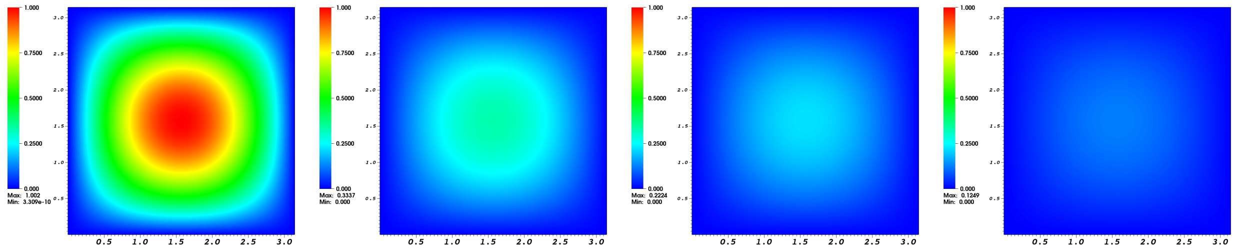


Figure 19.21: Heat equation with $\theta = 1$ (backward Euler) at $T = 0, 1, 2, 5$. The solution is stable and satisfies the parabolic maximum principle. The color scale is fixed between 0 and 1.

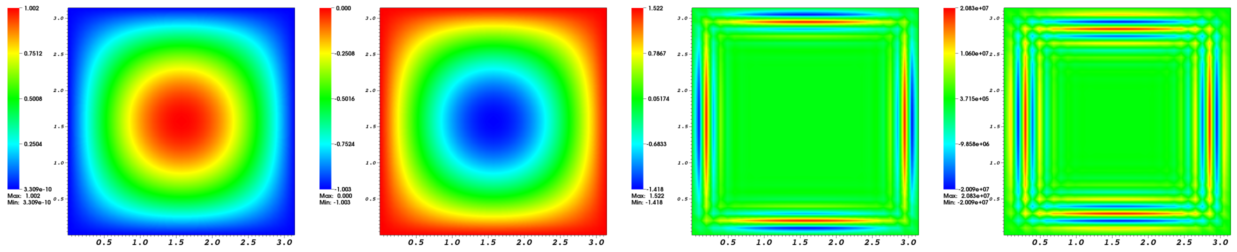


Figure 19.22: Heat equation with $\theta = 0$ (forward Euler) at $T = 0, 1, 2, 5$. The solution is unstable, showing non-physical oscillations, because the time step restriction (19.43) is violated. The color scale is adapted at each time to the minimum and maximum values of the solution.

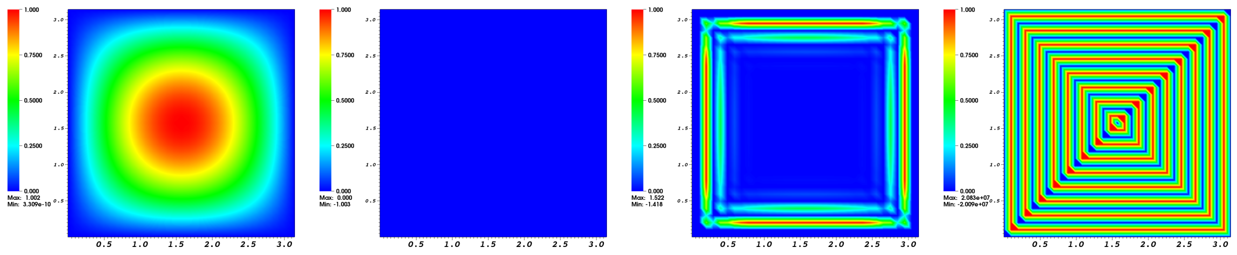


Figure 19.23: Heat equation with $\theta = 0$ (forward Euler) at $T = 0, 1, 2, 5$. The solution is unstable, showing non-physical oscillations, because the time step restriction (19.43) is violated. The color scale is fixed between 0 and 1.

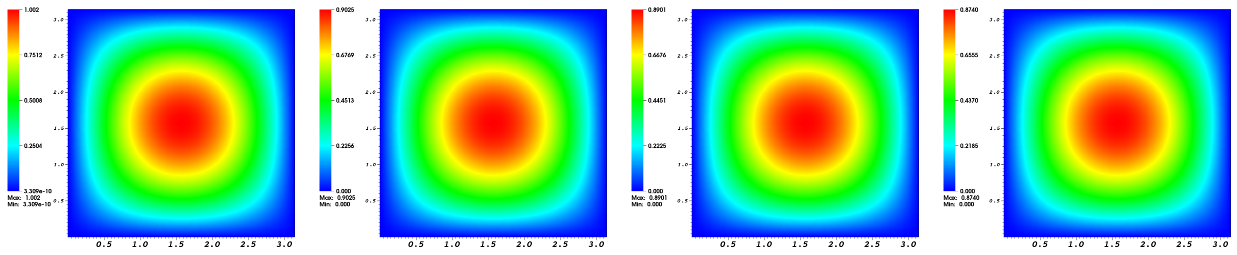


Figure 19.24: Heat equation with $\theta = 0$ at $T = 0, 1, 2, 5$ and time step size $k = 0.01s$. Here the results are stable since the time step size k has been chosen sufficiently small to satisfy the stability estimate (19.43). Of course, to obtain results at the same times T as before, we now need to compute more solutions, i.e., $N = 500$, rather than 5 as in the other tests. The color scale is adapted at each time to the minimum and maximum values of the solution.

19.5 Computational convergence analysis (complement)

We provide some tools to perform a computational convergence analysis. In these notes we faced two situations of 'convergence':

- **Discretization error:** Convergence of the discrete solution u_h towards the (unknown) exact solution u;
- **Iteration error:** Convergence of an iterative scheme to approximate the discrete solution u_h through a sequence of approximate solutions $u_h^{(k)}, k = 1, 2, \ldots$.

In the following we further illustrate the terminologies 'first order convergence', 'convergene of order two', 'quadratic convergence', 'linear convergence', etc.

19.5.1 Discretization error

Before we go into detail, we discuss the relationship between the degrees of freedom (DoFs) N and the mesh size parameter h. In most cases the discretization error is measured in terms of h and all a priori and a posteriori error estimates are stated in a form

$$\|u - u_h\| = O(h^\alpha), \quad \alpha > 0.$$

In some situations it is however better to create convergence plots in terms of DoFs vs. the error. One example is when adaptive schemes are employed with different h. Then it would be not clear to which h the convergence plot should be drawn. But simply counting the total numbers of DoFs is not a problem though.

Relationship between h and N (DoFs)

The relationship of h and N depends on the basis functions (linear, quadratic), whether a Lagrange method (only nodal points) or Hermite-type method (with derivative information) is employed. Moreover, the dimension of the problem plays a role.

We illustrate the relationship for a Lagrange method with linear basis functions in 1D,2D,3D:

Proposition 19.34. *Let d be the dimension of the problem: $d = 1, 2, 3$. It holds*

$$N = \Big(\frac{1}{h} + 1\Big)^d$$

where h is the mesh size parameter (lengh of an element or diameter in higher dimensions for instance), and N the number of DoFs.

Proof. Sketch. No strict mathematical proof. We initialize as follows:

- 1D: 2 values per line;
- 2D: 4 values per quadrilaterals;
- 3D: 8 values per hexahedra.

Of course, for triangles or primsms, we have different values in 2D and 3D. We work on the unit cell with $h = 1$. All other h can be realized by just normalizing h. By simple counting the nodal values, we have in 1D

```
h    N
=======
1    2
1/2  3
1/4  5
1/8  9
1/16 17
1/32 33
...
=======
```

We have in 2D

```
h    N
=======
1    4
1/2  9
1/4  25
1/8  36
1/16 49
1/32 64
...
=======
```

We have in 3D

```
h    N
=======
1    8
1/2  27
1/4  64
...
=======
```

□

Discretization error

With the previous considerations, we have now a relationship between h and N that we can use to display the discretization error.

Proposition 19.35. *In the approximate limit it holds:*

$$N \sim \Big(\frac{1}{h}\Big)^d$$

yielding

$$h \sim \frac{1}{\sqrt[d]{N}}$$

These relationships allow us to replace h in error estimates by N.

Proposition 19.36 (Linear and quadratic convergence in 1D)**.** *When we say a scheme has a linear or quadratic convergence in 1D, (i.e., $d = 1$) respectively, we mean:*

$$O(h) = O\Big(\frac{1}{N}\Big)$$

or

$$O(h^2) = O\Big(\frac{1}{N^2}\Big)$$

In a linear scheme, the error will be divided by a factor of 2 when the mesh size h is divided by 2 and having quadratic convergence the error will decrease by a factor of 4.

Proposition 19.37 (Linear and quadratic convergence in 2D)**.** *When we say a scheme has a linear or quadratic convergence in 2D, (i.e., $d = 2$) respectively, we mean:*

$$O(h) = O\Big(\frac{1}{\sqrt{N}}\Big)$$

or

$$O(h^2) = O\Big(\frac{1}{N}\Big)$$

19.5.2 Computationally-obtained convergence order

In order to calculate the convergence order α from numerical results, we make the following derivation. Let $P(k) \to P$ for $k \to 0$, where k is the discretization parameter (here rather the time step size $k = t_n - t_{n-1}$, but it could also be the spatial discretization parameter h) be a converging process and assume that

$$P(k) - \tilde{P} = O(k^\alpha).$$

Here $\tilde{P}$ is either the exact limit P (in case it is known) or some 'good' approximation to it. Let us assume that three numerical solutions are known (this is the minimum number if the limit P is not known). That is

$$P(k), \quad P(k/2), \quad P(k/4).$$

Then, the convergence order can be calculated via the formal approach $P(k) - \tilde{P} = ck^\alpha$ with the following formula:

Proposition 19.38 (Computationally-obtained convergence order)**.** *Given three numerically-obtained values $P(k), P(k/2)$ and $P(k/4)$, the convergence order can be estimated as:*

$$\alpha = \frac{1}{log(2)} log\Big(\Big| \frac{P(k) - P(k/2)}{P(k/2) - P(k/4)} \Big| \Big). \tag{19.44}$$

The order α is an estimate and heuristic because we assumed a priori a given order, which strictly speaking we have to proof first.

Proof. We assume:

$$\begin{aligned} P(k) - P(k/2) &= O(k^\alpha), \\ P(k/2) - P(k/4) &= O((k/2)^\alpha). \end{aligned}$$

First, we have

$$P(k/2) - P(k/4) = O((k/2)^\alpha) = \frac{1}{2^\alpha} O(k^\alpha)$$

We simply re-arrange:

$$\begin{aligned} P(k/2) - P(k/4) &= \frac{1}{2^\alpha}\Big(P(k) - P(k/2)\Big) \\ \Rightarrow \quad 2^\alpha &= \frac{P(k) - P(k/2)}{P(k/2) - P(k/4)} \\ \Rightarrow \quad \alpha &= \frac{1}{log(2)} \, \frac{P(k) - P(k/2)}{P(k/2) - P(k/4)} \end{aligned}$$

□

Example: Temporal order for FE,BE,CN in ODEs

In the following we present results for the (absolute) end time error of an ODE problem (but it could be any other PDE problem as well) on three mesh levels (different time step sizes k) with three schemes (FE - forward Euler, BE - backward Euler, CN - Crank-Nicolson):

```
Scheme         #steps  k      Error
===========================================
FE err.:       8       0.36   0.13786
BE err.:       8       0.36   0.16188
CN err.:       8       0.36   0.0023295
FE err.:       16      0.18   0.071567
BE err.:       16      0.18   0.077538
CN err.:       16      0.18   0.00058168
FE err.:       32      0.09   0.036483
BE err.:       32      0.09   0.037974
CN err.:       32      0.09   0.00014538
===========================================
```

We monitor that doubling the number of intervals (i.e., halving the step size k) reduces the error in the forward and backward Euler scheme by a factor of 2. This is (almost) linear convergence, which is confirmed by using Formula (19.44) yielding $\alpha = 0.91804$. The CN scheme is much more accurate (for instance using $n = 8$ the error is 0.2% rather than $13 - 16\%$) and we observe that the error is reduced by a factor of 4. Thus quadratic convergence is detected. Here the 'exact' order on these three mesh levels is $\alpha = 1.9967$.

Remark 19.30. *Another example where the previous formula is used can be found in Section 19.3.*

19.5.3 Spatial discretization error

We simply use now h and then

$$\alpha = \frac{1}{log(2)} log\Big(\Big|\frac{P(h) - P(h/2)}{P(h/2) - P(h/4)}\Big|\Big) \tag{19.45}$$

in order to obtain the computational convergence order α. Here,

$$P(h) := u_h, \quad P(h/2) := u_{h/2}, \qquad P(h/4) := u_{h/4},$$

where u is the discrete PDE solution.

19.5.4 Temporal discretization error for fixed spatial numerical solution

If in time-dependent PDEs, no analytical solution can be constructed, the convergence can again be purely numerically determined. However, we have now influence from spatial and temporal components. Often, the spatial error is more influential than the temporal one. This requires that a numerical reference solution must be computed for sufficiently small h and k. If now 'coarse' temporal solutions are adopted (k coarse!), the spatial mesh must be sufficiently fine, i.e., h small!, because otherwise we see significant influences by the spatial components. Then, we perform the procedure from the previous sections: bisection of k and observing the descrease in the corresponding norm or goal functional. That is to say, observe for a discrete PDE solution u:

$$u_{k,h_{fixed,small}}, \quad u_{k/2,h_{fixed,small}}, \quad u_{k/4,h_{fixed,small}}, \quad \ldots, \quad u_{k_{ref},h_{fixed,small}}$$

With this, we can set:

$$\tilde{P} := u_{k_{ref},h_{fixed,small}}, \quad P(k) = u_{k,h_{fixed,small}}, \quad P(k/2) = u_{k/2,h_{fixed,small}}.$$

Of course, using $P(k/4)$ is also an option. But keep in mind that always h must be sufficiently small and fixed while the time step size is varied.

19.5.5 Extrapolation to the limit

In case, there is no manufactured solution, the numerical reference value can be improved by extrapolation:

Formulation 19.39. *Let h_1 and h_2 two mesh sizes with $h_2 < h_1$. The functionals of interest are denoted by J_{h_1} and J_{h_2}. Then, the extrapolation formula reads:*

$$J_{extra} = \frac{(h_1 * h_1 * J_{h_2} - h_2 * h_2 * J_{h_1})}{h_1 * h_1 - h_2 * h_2}$$

The value for J_{extra} can then be used as approximation for $\tilde{P}$ from the previous subsection.
A specific example from [25] realized in octave is:

```
% Octave code for the extrapolation to the limit
format long

h1 = 1/2^3 % Mesh size h1
h2 = 1/2^4 % Mesh size h2

% Richter/Wick, Springer, 2017, page 383
p1 = 0.784969295 % Functional value J on h1
p2 = 0.786079344 % Functional value J on h2
```

```
% Formula MHB, page 335 (Romberg/Richardson)
% See also Quarteroni, Saleri, Ger. 2014, Springer, extrapolation

pMaHB = p2 + (p2 - p1) / ((h1*h1)/(h2*h2) - 1)

% Formula Richter/Wick for extrapolated value
pRiWi = (h1*h1 * p2 - h2*h2 * p1) / (h1*h1 - h2*h2)
```

19.5.6 Iteration error

Iterative schemes are used to approximate the discrete solution u_h. This has a priori nothing to do with the discretization error. The main interest is how fast can we get a good approximation of the discrete solution u_h. One example can be found for solving implicit methods for ODEs in which Newton's method is used to compute the discrete solutions of the backward Euler scheme.

To speak about convergence, we compare two subsequent iterations:

Proposition 19.40. *Let us assume that we have an iterative scheme to compute a root z. The iteration converges with order p when*

$$\|x_k - z\| \leq c\, \|x_{k-1} - z\|^p, \quad k = 1, 2, 3, \ldots$$

with $p \geq 1$ and $c = const$. In more detail:

- *Linear convergence: $c \in (0,1)$ and $p = 1$;*
- *Superlinear convergence: $c := c_k \to 0,\ (k \to \infty)$ and $p = 1$;*
- *Quadratic convergence $c \in \mathbb{R}$ and $p = 2$.*

Cupic and higher convergence are defined as quadratic convergence with the respectice p.

Remark 19.31 (Other characterizations of superlinear and quadratic convergence)**.** *Other (but equivalent) formulations for superlinear and quadratic convergence, respectively, in the case $z \neq x_k$ for all k, are:*

$$\lim_{k\to\infty} \frac{\|x_k - z\|}{\|x_{k-1} - z\|} = 0,$$
$$\limsup_{k\to\infty} \frac{\|x_k - z\|}{\|x_{k-1} - z\|^2} < \infty.$$

Corollary 19.41 (Rule of thumb)**.** *A rule of thumb for quadratic convergence is: the number of correct digits doubles at each step. For instance, a Newton scheme to compute $f(x) = x - \sqrt{2} = 0$ yields the following results:*

```
Iter x            f(x)
==============================
0    3.000000e+00 7.000000e+00
1    1.833333e+00 1.361111e+00
2    1.462121e+00 1.377984e-01
3    1.414998e+00 2.220557e-03
4    1.414214e+00 6.156754e-07
5    1.414214e+00 4.751755e-14
==============================
```

Chapter 20

Exercises

Basic notions on ODEs

Exercise 20.1. *Solve:*

1. $y' + y = te^t, \ y(0) = 1;$
2. $y' - y = -\frac{1}{(t+1)^2}e^t, \ y(0) = 0.$

Exercise 20.2. *Solve:*

1. $y' + y = te^t y, \ y(0) = 1;$
2. *On* $]0, +\infty[, \ ty' + 3y - 2t^5 = 0, \ y(2) = 1.$

Exercise 20.3. *Give the general solutions of:*

1. $y'' - 2y' - 3y = 0;$
2. $y'' + 4y' + 4y = 0;$
3. $y'' - 2y' + 5y = 0.$

Numerical schemes for ODEs

Exercise 20.4. *Given the ordinary differential equation $u'(t) = f(t, u)$, derive basic numerical schemes by taking the forward and backward difference quotient for approximating u', respectively.*

Exercise 20.5. *Derive stability estimates for the Crank-Nicolson scheme.*

Numerical schemes for PDEs (complement)

Exercise 20.6. *Classify the following differential equations with respect to their order, linearity, coupling and whether these are PDE systems:*

1. *Find $u : \Omega \to \mathbb{R}$ such that $-\Delta u + \partial_x u = f$*
2. *Find $u : \Omega \to \mathbb{R}$ such that $\partial_{xx} u + \partial_{tx} u = g$*
3. *Find $u : \Omega \to \mathbb{R}$ such that $-\Delta u + u^2 = f$*
4. *Find $u : \Omega \to \mathbb{R}$ and $\varphi : \Omega \to \mathbb{R}$ such that*

$$\begin{aligned} -\Delta u &= f(\varphi) \\ |\nabla u|^2 - \Delta\varphi &= g(u) \end{aligned}$$

Exercise 20.7 (Weak formulations and energy forms). *Let $\Omega \subset \mathbb{R}^n$ with a sufficiently smooth boundary $\partial\Omega$. Find $\Omega \to \mathbb{R}$ such that*

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{in } \partial\Omega \end{aligned}$$

Derive the energy formulation and derive the so-called weak formulation for this problem.

Exercise 20.8. *Formulate a Galerkin representation for the following problem:*

$$\begin{aligned} -\nabla \cdot (\alpha \nabla u) + \gamma u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{in } \partial\Omega \end{aligned}$$

Exercise 20.9. *Let $\Omega = (0,1)$. Discretize in space and time the following problem:*

$$\partial_t u - \nabla \cdot (\nabla u) = 1 \quad \text{in } \Omega, \tag{20.1}$$

$$u = 0 \quad \text{on } \partial\Omega, \tag{20.2}$$

$$u(0) = 0 \quad \text{in } \Omega \times \{t = 0\}. \tag{20.3}$$

Extended task (starting point of a project): Implement this problem in a software, e.g., python or octave or C++.

Exercise 20.10. *Let Ω be an open, bounded subset of $\mathbb{R}^d, d = 1$ and $I := (0,T]$ where $T > 0$ is the end time value. The IBVP (initial boundary-value problem) reads: Find $u := u(x,t) : \Omega \times I \to \mathbb{R}$ such that*

$$\begin{aligned} \rho \partial_t u - \nabla \cdot (\alpha \nabla u) &= f \quad \text{in } \Omega \times I, \\ u &= a \quad \text{on } \partial\Omega \times [0,T], \\ u(0) &= g \quad \text{in } \Omega \times t = 0, \end{aligned}$$

where $f : \Omega \times I \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$ and $\alpha \in \mathbb{R}$ and $\rho > 0$ are material parameters, and $a \geq 0$ is a Dirichlet boundary condition. More precisely, g is the initial temperature and a is the wall temperature, and f is some heat source.

1. *Using finite differences in time and finite elements in space, implement the heat equation in octave or python (Hint: Try to implement a general One-Step-θ scheme with $\theta \in [0,1]$ for temporal discretization and linear finite elements for spatial discretization).*

2. *Set $\Omega = (-10, 10)$, $f = 0$, $\alpha = 1$, $\rho = 1$, $a = 0$, $T = 1$, and*

$$g = u(0) = \max(0, 1 - x^2).$$

   and carry out simulations for $\theta = 0, 0.5, 1$. What do you observe? Why do you make these observations?

3. *Justify (either mathematically or physically) the correctness of your findings.*

4. *Why do you observe difficulties using $\theta < 0.5$. What is the reason and how can this difficulty be overcome?*

5. *Detecting the order of the temporal scheme: Choose a sufficiently fine spatial discretization (by choosing make the spatial discretization parameter h be sufficiently small) and compute with different time step sizes δt the value of the point $u(x_0, T) := u(x = 0; T = 1)$. Compute the error*

$$|u_{\delta t_l}(x = 0; T = 1) - u_{\delta t_{fine}}(x = 0; T = 1)|, \quad l = l_0, l_0/2, l_0/4, \ldots$$

   How does the error behave with respect to different θ?

Exercise 20.11. *Implement the heat equation with given right hand side:*

$$\begin{aligned}
\partial_t u - \Delta u &= f \quad \textit{in } \Omega = (0,1)^2 \times T = (0,10)\\
u(x,t) &= 0 \quad \textit{on } \partial\Omega \times T\\
u(x,0) &= 0 \quad \textit{in } \Omega = (0,1)^2 \times \{0\}.
\end{aligned}$$

Part V

Projects on numerical modeling in teams

Chapter 21

Project work

The second half of this class consists of numerical projects done in pairs. The idea is to develop algorithms for a given mathematical problem statement. These algorithms then must be implemented and carefully analyzed from a computational point of view. As programming language we encourage to use an open-source package as for instance python, octave, or C++. Matlab (linked to octave as it is known) is also possible in case you have a personal license, but Ecole Polytechnique will not provide you a license because the school supports open-source programming solutions.

Typical guideline questions During the projects, some typical aspects of interests are:

- Why does the algorithm converge/not converge?
- Could the algorithm be more efficient? Why/why not?
- Are there alternative algorithms? Do they converge better?
- Try to explain your observations either mathematically or in a rigorous computational fashion.
- What do you observe? What is your personal interpretation of the results?

These (and possibly more) results shall be presented in a presentation and a written report. For more guideline questions, please also see Section 1.4.2.

21.1 Idea and formal aspects

Please have a careful look on the different projects listed below. They are related to the different topics presented previously (mandatory sections as well as complement sections). In case you have questions, before your choice, please let us know.

Once you made a choice and have built a team (2 or exceptionally 3 persons), you may start working on your project. **Please be careful that until the mid-term exam, you also work still on course materials (exercises!) of the previous chapters.**

21.1.1 Choice of your project and contact email addresses

Projects are in principle to be chosen among those listed below, including the topics from the previous years. It is also possible that you devise your own project, in this case speak with us.

At latest, please make your choice on the day of the mid-term exam such that you can immediately start working afterward. In all cases, please write us

- `samuel.amstutz@polytechnique.edu`
- `thomas.wick@ifam.uni-hannover.de`

an email to indicate:

- Subject line of this email: MAP502: choice of project
- Names of all group members;
- Your project number (Section number from below).

In the meetings No. 7,8,9, we will advise the different groups and provide further help on the specific projects. This will be individual group work and no more lectures.

21.1.2 Final exam mid December

The end-term exam (for each group) consists of:

- A report (word or latex/pdf) of your task, which contains the problem statement, the numerical approach(es), set-up of the numerical example(s), analysis/interpretation of the numerical results;
- A 20 minutes presentation (beamer/PowerPoint);
- Questions from our side to your presentation.

21.2 New projects for 2021

21.2.1 Image compression with SVD (singular valued decomposition) - related to eigenvalues

An interesting application of eigenvalues is the extension to non-quadratic matrices in which singular values must be computed. The first task in this project is to work through and understand the basics of SVD, e.g., with [23][Section 6.5]. The goal is then to study in detail Example 6.9 in [23]. A specific emphasis shall be on the following tasks:

1. Work out the theoretical-algorithmic details of the singular value decomposition (SVD)
2. What exactly happens when using `[U,S,V]=svd(A)` in octave/MATLAB? If you use python, please search for these commands in the python library descriptions.
3. After the implementation: What are the memory requirements?
4. Computational cost (wall clock time)?
5. Accuracy: how many singular values result into an acceptable accuracy?
6. Make a literature research and study another image.

21.2.2 Numerical optimization and application to regression problems

This project elaborates on exercise 15.21, where the steepest descent method was illustrated. Here we go further and explore Newton-type methods as well as some aspects of constrained optimization.

Unconstrained optimization: Newton-type methods

$*$ **Newton's method**

Newton's method in unconstrained optimization consists in solving the first order optimality condition $\nabla J(u) = 0$. With the notation of chapter 11 (listing 11.1) and exercise 15.21 it is defined by the search direction and the stepsize:

$$\left| \begin{array}{l} d^k = -\nabla^2 J(u^{k-1})^{-1} \nabla J(u^{k-1}), \\ s_k = 1, \end{array} \right.$$

where $\nabla^2 J(u)$ is the Hessian matrix of J at point u. Note that in practice we do not compute the inverse of $\nabla^2 J(u^{k-1})$, but 'just' solve the linear system

$$\nabla^2 J(u^{k-1})d^k = -\nabla J(u^{k-1}).$$

Newton's method is not necessarily a descent method (this depends whether $\nabla^2 J(u^{k-1})$ is positive definite or not). Let us recall that convergence is only expected when the initial guess is sufficiently close to the solution. Of course, when J is quadratic, it converges in one iteration.

1. Define for each example from exercise 15.21 or for your own examples the function `ddJ(x,y)` which at point $u = (x, y)$ provides the matrix $\nabla^2 J(u)$.
2. Implement Newton's method, display the obtained solution and plot the trajectory.
3. Test the convergence when the initial point is varying. Show that convergence is not guaranteed even for a convex function.

$\ast$ **Quasi-Newton's methods**

For problems of large dimension, or when the Hessian matrix is, out of reach, we can use quasi-Newton methods which mimick Newton's methods without computing the exact Hessian. They extend the secant method to higher dimension. They have the general expression

$$\left| \begin{array}{l} d^k = -H^{k-1}\nabla J(u^{k-1}), \\ s_k \text{ minimizes } \alpha \mapsto J(u^{k-1} + \alpha d^k), \end{array} \right.$$

where H^{k-1} is some symmetric positive definite approximation of $\nabla^2 J(u^{k-1})^{-1}$. The DFP method starts with some H^0 then uses the construction:

$$\left| \begin{array}{l} \delta^k = u^k - u^{k-1} = s_k d^k, \\ \gamma^k = \nabla J(u^k) - \nabla J(u^{k-1}), \\ H^k = H^{k-1} + \dfrac{\delta^k (\delta^k)^T}{(\delta^k)^T \gamma^k} - \dfrac{H^{k-1}\gamma^k (\gamma^k)^T H^{k-1}}{(\gamma^k)^T H^{k-1}\gamma^k}. \end{array} \right.$$

For the BFGS variant (very popular) we modify the above by:

$$H^k = H^{k-1} + \left(1 + \frac{(\gamma^k)^T H^{k-1}\gamma^k}{(\delta^k)^T\gamma^k}\right)\frac{\delta^k(\delta^k)^T}{(\delta^k)^T\gamma^k} - \frac{\delta^k(\gamma^k)^T H^{k-1} + H^{k-1}\gamma^k(\delta^k)^T}{(\delta^k)^T\gamma^k}.$$

1. Implement the DFP method and display the results and the trajectories, with $H^0 = I$ and Armijo's line search with $\gamma = 0$ (i.e. only check for descent).
2. Test the convergence depending on the initial point.
3. Do the same with BFGS.

$\ast$ **Comparisons**

Compare the running times and recapitulate the pros and cons of each method.

Constrained optimization

We consider the least square approximation problem :

$$\min_{u \in K} J(u) = \frac{1}{2}\|u - p\|^2, \tag{21.1}$$

where $p \in \mathbb{R}^n$ is given, $\|.\|$ is the standard Euclidean norm and K is a polyhedral convex set defined by

$$K = \{u \in \mathbb{R}^n, Au \leq b\}.$$

Therefore, our goal is to compute the projection of p onto K. Note that except in some very specific cases (we have seen the case $A = I$, $b = 0$), we have no formula for this projection.

We will develop a method based on the Lagrangian

$$L(u, \lambda) = J(u) + \lambda \cdot (Au - b).$$

The underlying idea is to penalize the constraint $Au - b \leq 0$ with a weight vector $\lambda \in \mathbb{R}^m_+$ called Lagrange multiplier. We expect that, by a well-chosen Lagrange multiplier λ^*, minimizing $L(u, \lambda^*)$ with respect to u in $\mathbb{R}^n$ will provide a solution to (21.1). Intuitively, for each individual scalar constraint $(Au - b)_i \leq 0$, a two small Lagrange multiplier λ_i would lead to constraint violation, while a two large one would lead to a sub-optimal solution. The question of finding an appropriate Lagrange multiplier will be the purpose of the so-called dual problem. To introduce this we need the concept of saddle point. We say that the pair $(u^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m_+$ is a saddle point of L if:

$$L(u^*, \lambda) \leq L(u^*, \lambda^*) \leq L(u, \lambda^*) \qquad \forall (u, \lambda) \in \mathbb{R}^n \times \mathbb{R}^m_+.$$

$*$ **A bit of theory**

1. Show that if $(u^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m_+$ is a saddle point of L then u^* solves (21.1). Hint: see that $\lambda^* \cdot (Au^* - b) = 0$.
   Note: we will admit the existence and even the uniqueness of a saddle point.

2. Given $\lambda \in \mathbb{R}^m_+$, give an expression of the unique solution u_λ to
$$\min_{u \in \mathbb{R}^n} L(u, \lambda).$$
   Deduce an expression of the dual criterion
$$D(\lambda) = \min_{u \in \mathbb{R}^n} L(u, \lambda).$$

3. The dual problem is
$$\max_{\lambda \in \mathbb{R}^m_+} D(\lambda). \tag{21.2}$$
   How is this related to the notion of saddle point?

4. Uzawa's method is the steepest ascent method with projection applied to the dual problem (21.2). Here the projection is known! Show that this method can be written as
$$u^k = p - A^T \lambda^{k-1},$$
$$\lambda^k = \left(\lambda^{k-1} + \alpha(Au^k - b)\right)^+$$
   (we use a fixed stepsize considering the non-normalized gradient as ascent direction).

It can be proved that if α is such that

$$0 < \alpha < \frac{2}{\|A\|_2^2}$$

then u^k converges to the solution of (21.1). Here, we use the matrix norm $\|A\|_2 = \sqrt{\varrho(A^T A)}$ where $\varrho(B)$ is the spectral radius of B.

$*$ **Implementation**

1. Implement Uzawa's algorithm `projection(A,b,p,alpha)` with stopping criterion:
$$\|u^{k+1} - u^k\| < 10^{-6}.$$

2. Test with
$$K = \{u \in \mathbb{R}^2, \|u\|_\infty \leq 1\}.$$
   Check the convergence result.

Application: regression

$*$ **Isotonic and convex regression**

Let $p = (p_1, ..., p_n) \in \mathbb{R}^n$ and

$$K = \{(u_1, ..., u_n) \in \mathbb{R}^n, u_{i+1} \geq u_i \ \forall 1 \leq i < n\}.$$

1. Construct the corresponding matrix A.
2. Use `projection(A,b,p,alpha)` in this case for some examples of vector p. This is isotonic regression. Provide a graphical output.
3. Do the same with the convex regression:

$$K = \{(u_1, ..., u_n) \in \mathbb{R}^n, u_i \leq \frac{1}{2}(u_{i-1} + u_{i+1}) \ \forall 1 < i < n\}$$

$*$ **Application to statistics**

We want to sell honey glasses and we are interested in the optimal price to maximize both the sales volume and our profit. To this end, we go to different cities in France and offer glasses for different prices and take notes how many glasses were sold. These measurement data are listed in Table 21.1.

No x of sold honey glasses	50	30	25	22	27	30	26	32	28	26	21	16	8	4
Price $p(x)$ per glass	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0

Table 21.1: Data of 14 measurements of different prices and the corresponding number of sold glasses.

On the same graph display the linear regression (see section 12.7.1), the isotonic (decreasing) regression, the convex regression and the concave regression. Be careful that the implementation of convexity / concavity must be adapted due to non-uniform x data.

21.2.3 Predator-prey systems

In this project, we consider in more detail two coupled ODEs, namely predator-prey systems: Find prey $p(t) : [0, T] \to \mathbb{R}$ and predators $s(t) : [0, T] \to \mathbb{R}$ such that

$$\begin{aligned} p'(t) &= r_p p(t) - \alpha s(t)p(t) \quad \text{in } I = (0, T] \\ s'(t) &= r_s s(t) + \mu\alpha p(t)s(t) \quad \text{in } I = (0, T] \\ p(t_0) &= p_0 \\ s(t_0) &= s_0 \end{aligned}$$

where $r_p > 0$ is the prey growth rate, $r_s < 0$ is the predator growth rate, $\alpha \in [0, 1]$ the per-capita predation rate, and μ the conversion rate. A good and recent nice introduction can be found in [22][Chapter 6], which is strongly recommended since useful background information is provided. In Section 6.5.3 of [22], the following values are chosen:

$$r_p = 0.6, \quad r_s = -0.2, \quad \alpha = 0.001, \quad \mu = 0.01.$$

and the initial values $p(t_0) = 5000$ and $s(t_0) = 40$.

We have the following tasks in mind:

1. Implement the Euler method, backward Euler method, and the Crank-Nicolson method in octave or python.
2. Run simulations with the above given ODE system, parameters and initial values for $T = 80$ years.

3. Using the backward Euler method and the Crank-Nicolson method, an implicit system arises. Formulate this system as root finding problem and formulate Newton's method to solve these implicit systems. Hints are given in Section 12.6.

4. Using different (time) step sizes, investigate and analyze the findings (instability of the Euler method) by evaluating the end time value $p(T)$ and $s(T)$. Compare these values for different time step. sizes and determine computationally the convergence order (see Section 19.5)

5. Extend the predator-prey model and include the internal interaction rates σ_p and σ_s, [22][p. 183ff].

6. Run again numerical simulations.

21.2.4 Neural network approximation of ODEs

Implement a neural network to approximate the solution of an ODE problem. The performance shall be compared to a classical implementation using finite differences as we have studied previously in class. As direction use the work presented in [20] (available via open access).

The problem statement is:

$$\begin{aligned} u'(x) &= u(x) - u(x)^2, \quad \text{in } (0,T] \\ u(0) &= 0.5 \end{aligned}$$

with $T = 10$.

The specific tasks are:

1. Read and understand [20]

2. Formulate a root-finding problem of the form $F(x,u) = 0$ and formulate the least-squares problem

$$E(w) = \frac{1}{2}\sum_{i=1}^{N} \|t_i - y_i\|$$

where w is the neural network weight vector, t_i contains given information and y_i the unknown information to be approximated (see [20], Section 3)

3. Please make clear and understand how the weight vector w is trained, how the collocation points $x_1, \dots, x_{N_2}$ are chosen, and how it is possible to determine with this approach a solution of the ODE in the time interval $[0,T]$

4. Implement a neural network approximation for the previous problem statement.

5. Implement a finite difference (e.g., forward Euler) for the previous problem statement.

6. Compare the computational cost: which one is faster?

7. Study in more detail initial weights for the neural network

8. Study the learning rate in more detail.

9. Except the computational cost, what are other advantages and/or shortcomings of using neural networks for solving ODEs.

10. Apply the neural network and also finite differences to a second problem:

$$\begin{aligned} u'(x) &= \sin(x) - \frac{u(x)}{x}, \quad \text{in } (0,T] \\ u(\pi) &= 1 \end{aligned}$$

with $T = 4$ and $T = 12$.

21.2.5 Neural network for image classification

Implement an artificial neural network for the classification of the handwritten digits 0 and 1, for instance taken from the MNIST database. Use a single hidden layer (see section 12.7.2) and a classical steepest descent method for training. The detailed structure of the network and the computation of the gradient (backpropagation) is to be discussed with us. Training is performed with a subset of the database and testing with another subset. Study the accuracy of the classifier with respect to the number of neurons in the hidden layer and the number of training samples. Of course, if time allows, the classification of more digits or other images could be addressed.

21.2.6 Shape optimization for fluids using FreeFem++

This project is intended to students willing to make use of concepts of differential calculus and optimization in infinite dimensional vector spaces. The free software FreeFem++ will be used.

The goal of this project is to design optimal shapes for Navier-Stokes flows using as main building block the code

```
https://github.com/flomnes/optiflow
```

related to the paper [8] (an online version is freely available).

1. Explain in your own words the notion of shape derivative.
2. Explain the main aspects of the algorithm described in [8].
3. Test the algorithm on the built-in examples and study the role of some algorithmic parameters.
4. Implement your own examples, with geometries and physical parameters of your choice.

21.3 List of projects 2018-2020 (can still be chosen in 2021)

21.3.1 Direct and iterative solution of linear equation systems

In this exercise, we develop numerical schemes for solving linear equation systems $Ax = b$. Let $A \in \mathbb{R}^{n\times n}$ with

$$A = \frac{1}{h^2}\begin{pmatrix} h & 0 & \dots & & \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ & & \dots & 0 & h \end{pmatrix} \tag{21.3}$$

and the right hand side $b \in \mathbb{R}^n$ with $b = (1, \dots, 1)^T$ with $h = 1$. This matrix arises in solving for instance the Poisson problem with finite differences or finite elements.

1. Implement Gaussian elimination and solve the above system for $n = 10$ and 100.
2. Implement now numerical fixed-point schemes: the Richardson iteration, Jacobi method and Gauss-Seidel (see Section 7.2 and for some more information on Richardson in Section 19.2.2)
3. Compare iteration numbers and wall clock times for all four schemes. Which scheme is the most efficient one? Hint: please also recall the principal differences between direct and iterative schemes.
4. Change $h = 0.5, 0.25, 0.125$. What do you observe?

5. Extend A to $n = 10^6$ and solve again the problem. How much memory on your computed is needed and what are now the wall clock times? Hint: Consider straightforward implementations of the matrix A (each entry is stored and uses memory) as well as so-called *sparse matrices* in which only non-zero entries are stored.

6. For using Richardson's iteration, the largest eigenvalue λ_{max} must be computed. How can this be done efficiently? Please try to understand that two numerical methods (solving $Ax = b$ with Richardson and computing the eigenvalue) interact. Of course, having λ_{max} at hand, gives a very efficient Richardson approach, but obtaining λ_{max} itself might be very costly. For this reason, a trade-off in the computational cost must be envisaged.

7. Implement the LU decomposition. Can you see any difference in wall clock time / memory consumption for (i) constructing the LU decomposition and (ii) solving $Ax = b$ as $LUx = b$ with the two-step algorithm from Section 4.5.1.

8. **Optional** Implement the Cholesky decomposition and compare whether it is really only half of the computational cost in comparison to the LU decomposition.

9. **Optional** Implement the conjugate gradient (CG) scheme according to Section 7.2.2. This scheme is known to perform very well for the above system.

21.3.2 Numerical methods for eigenvalues

In this exercise, we develop schemes for computing numerically eigenvalue problems. Let $A \in \mathbb{R}^{n\times n}$ with

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \qquad h = \frac{1}{n+1} \tag{21.4}$$

with $h = 1e{-}1, 1e{-}3, 1e{-}6$. See e.g. [23] chap. 9 for an interpretation of this matrix in the framework of finite differences and also [31] for connections with vibrating strings / wave equation.

1. Implement the power method to compute the largest eigenvalue of A.

2. The inverse power method is the power method applied to A^{-1}, without explicitly computing A^{-1} since it is actually enough for that to solve linear systems of matrix A, see details in [23]. Implement the inverse power method to compute the smallest eigenvalue of A. It is also interesting to plot a corresponding eigenvector and compare with the notions of eigenfrequency / eigenmode you may know from physics.

3. To compute all eigenvalues, we adopt the so-called method of QR iterations, see e.g. [23]. In order to construct the orthogonal matrix Q, the Gram-Schmidt procedure (easy to implement, but numerically not stable) or Householder matrices can be used, see e.g. [31]. Further hints and materials will be given for the groups working on this project.

4. (Optional) Implement Lanczos' method for computing the eigenvalues, see e.g. [31].

5. Compare your implemented methods in terms of accuracy, efficiency (number of iterations), and give an interpretation which method you would recommend to use based on your observations.

Hint 1: The power method and QR iterations are already implemented in many scientific computing libraries. The goal **is not** to use such built-in commands, but to implement the methods by yourself.
Hint 2: It can be proven that for the given matrix A the eigenvalues admit the analytical expression

$$\lambda_k = \frac{4}{h^2}\sin^2\frac{k\pi}{2(n+1)}, \qquad k = 1, ..., n.$$

This can be used for comparisons.

21.3.3 Numerical methods for root-finding problems

Develop a nonlinear iteration schemes (fixed-point and Newton) in $\mathbb{R}^2$ to find the root of the problem:

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \quad f(x,y) = \Big(2xay^2, 2(x^2+\kappa)ay\Big)^T, \tag{21.5}$$

where $\kappa = 0.001$ and $a = 3$.

1. Justify first that an antiderivative of f is $F(x,y) = (x^2+\kappa)ay^2$. What is the relation between f and F?

2. Compute the root of f by hand. Derive the derivative f' and study its properties.

3. Finally, design the requested Newton algorithm. As initial guess, take $(x_0, y_0) = (3, 6)$.

4. What do you observe with respect to the number of Newton iterations?

5. How could we possibly reduce the number of Newton steps?

6. Implement as a second example the system

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \quad f(x,y) = \Big(\exp(x^2+y^2) - 3, x + y - \sin(3(x+y))\Big)^T. \tag{21.6}$$

   and solve the root-finding problem $f(x,y) = (0,0)$. How does Newton's method perform here?

21.3.4 Discrete Fourier transform

This project deals with discrete Fourier analysis applied to signal and image processing.

1. Implement the discrete Fourier transform of a set of pairs

$$(x_j, y_j = f(x_j)), \qquad x_j = \frac{2j\pi}{n+1}, \qquad 0 \le j \le n.$$

   Test on easy cases and plot the corresponding interpolating trigonometric polynomial.

2. Apply it to the function of exercise 14.18. What are your conclusions?

3. Apply it to the function of exercise 14.17 and discuss Shannon's theorem. Illustrate the aliasing phenomenon for $f(x) = \sin(x) + \sin(5x)$ with 9 nodes. How many nodes do we need for an exact representation of $f(x) = \cos(x)\cos(5x)$?

4. Improve the implementation using a simple form of Cooley-Tukey's algorithm [31]. Hint: assume that $n+1$ is a power of 2, in particular n is odd ($n = 2M - 1$), see [23], and using periodicity compute c_k for indices $k = 0, ..., n$. Discuss the computer effort in comparison with the basic implementation.

5. Illustrate the effect of truncating the discrete Fourier transform (i.e. omitting high frequencies). Discuss some applications in signal compression.

6. Implement the discrete Fourier transform in two dimensions [31], first in a basic way then optionally in an improved way inspired from question 4.

7. Show the effect of truncating the DFT on greyscale images. Give illustrations in the context of image denoising / compression.

8. Possible continuation: FFT for solving linear PDEs with applications in image denoising. To be discussed with us.

21.3.5 Numerical solution of ODEs

Let the following ODE initial-value problem be given:

$$y'(t) = ay(t), \qquad y(t_0) = y_0 \tag{21.7}$$

on the time interval (t_0, T) where $t_0 = 1$ and $T = 4$. Let the initial value be $y_0 = 2$. Furthermore consider two different test cases with $a = -0.25$ (test 1) and $a = -10$ (test 2).

1. Implement the Euler method, backward Euler method, and the Crank-Nicolson method in octave or python.

2. Recapitulate the stability regions for these three numerical schemes and compute the critical (time) step size for the (forward) Euler scheme for test 1 and test 2.

3. Using the backward Euler method and the Crank-Nicolson method, an implicit system arises. Formulate this system as root finding problem and formulate Newton's method to solve these implicit systems.

4. Using different (time) step sizes, investigate and analyze the findings (instability of the Euler method) by evaluating the end time value $y(T)$. Compare these values for different time step sizes and determine computationally the convergence order (see Section 19.5)

5. When the above system is running, please test your solver(s) at the nonlinear equation:

   $$y' = ay - by^2, \quad y(t_0) = y_0. \tag{21.8}$$

   For population growth, usually $a \gg b$. For instance, one example for the growth of the world population is:

   $$t_0 = 1965, \quad y_0 = 3.34 \times 10^9, \quad a = 0.029, \quad b = 2.695 \times 10^{-12}.$$

   What was the expected world population in the year $t = 2000$? What is the expected world population in $t = 2021$? Are these numerical solutions reasonable?

6. **Optional** In case of interest and time, implement higher order time-stepping schemes such as Runge-Kutta schemes. Determine and compare the accuracy by again evaluating the end time value $y(T)$.

21.3.6 Numerical solution of PDEs

Consider Poisson's problem in 1D and 2D. In 1D, let $\Omega = (0, 1)$. We seek $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} -u''(x) &= f \quad \text{in } \Omega \\ u(0) = u(1) &= 0 \end{aligned}$$

with $f = -1$. In 2D, let $\Omega = (0, 1)^2$. We seek $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

with $f = -1$. The tasks are:

1. Implement the 1D Poisson problem using finite differences. Compare your results to Section 18.1.6 (taken copy and paste from [29][Section 6.7, pages 75ff])

2. Implement the 1D Poisson problem using finite elements. Compare your results to Section 18.3 (taken copy and paste from [29][Section 8.16.4, pages 178ff]) by implementing the manufactured solution and the L^2 error estimate. Compute then the convergence order outlined on page 179 of [29].

3. Implement the 2D Poisson problem using finite elements. Compare your results to Section 18.3 (taken copy and paste from [29][Section 8.16, pages 175ff]).

4. Perform mesh refinement studies and implement an the L^2 norm. Implement then the manufactured solution from Section 18.3.4 (taken from [29][Section 8.16.3]). Run now on refined meshes the problem, and compare your results to the Table in Section 18.3.4 (taken from Section 8.16.3.2 in [29][page 177]).

5. Based on this analysis, what are your personal observations and conclusions?

Bibliography

[1] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The `deal.II` library, version 8.5. *Journal of Numerical Mathematics*, 2017.

[2] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007.

[3] P. Bastian. Lecture notes on scientific computing with partial differential equations. Vorlesungsskriptum, 2014.

[4] D. Braess. *Finite Elemente*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, vierte, überarbeitete und erweiterte edition, 2007.

[5] M. Braun. *Differential equations and their applications*. Springer, 1993.

[6] P. G. Ciarlet. *Mathematical Elasticity. Volume 1: Three Dimensional Elasticity*. North-Holland, 1984.

[7] P. G. Ciarlet. *The finite element method for elliptic problems*. North-Holland, Amsterdam [u.a.], 2. pr. edition, 1987.

[8] C. Dapogny, P. Frey, F. Omnès, and Y. Privat. Geometrical shape optimization in fluid mechanics using FreeFem++. *Struct. Multidiscip. Optim.*, 58(6):2761–2788, 2018.

[9] P. Deuflhard. *Newton Methods for Nonlinear Problems*, volume 35 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, 2011.

[10] P. Deuflhard and F. Bornemann. *Scientific computing with ordinary differential equations*. Texts in applied mathematics. Springer, New York, 2002.

[11] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring. *GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations*. CreateSpace Independent Publishing Platform, 2014. ISBN 1441413006.

[12] L. C. Evans. *Partial differential equations*. American Mathematical Society, 2010.

[13] C. Goll, T. Wick, and W. Wollner. DOpElib: Differential equations and optimization environment; A goal oriented software library for solving pdes and optimization problems with pdes. *Archive of Numerical Software*, 5(2):1–14, 2017.

[14] C. Großmann, H.-G. Roos, and M. Stynes. *Numerical Treatment of Partial Differential Equations*. Springer, 2007.

[15] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer, 1985.

[16] C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *SIAM review*, 61(4):860–891, 2019.

[17] G. Holzapfel. *Nonlinear Solid Mechanics: A continuum approach for engineering.* John Wiley and Sons, LTD, 2000.

[18] C. Johnson. *Numerical solution of partial differential equations by the finite element method.* Cambridge University Press, Campridge, 1987.

[19] L. Kantorovich and G. Akhilov. *Functional analysis in normed spaces.* Fizmatgiz, Moscow, 1959, German translation: Berlin, Akademie-Verlag, 1964.

[20] T. Knoke and T. Wick. Solving differential equations via artificial neural networks: Findings and failures in a model problem. *Examples and Counterexamples*, 1:100035, 2021.

[21] J. Nocedal and S. J. Wright. *Numerical optimization.* Springer Ser. Oper. Res. Financial Engrg., 2006.

[22] A. Quarteroni and P. Gervasio. *A Primer on Mathematical Modelling.* Springer, 2020.

[23] A. Quarteroni, F. Saleri, and P. Gervasio. *Scientific computing with MATLAB and Octave, Fourth Edition.* Texts in computational science and engineering. Springer, 2014.

[24] R. Rannacher. Einführung in die numerische Mathematik (Numerische Mathematik 0). Vorlesungsskriptum, 2000.

[25] T. Richter and T. Wick. *Einführung in die numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele.* Springer, 2017.

[26] M. Schäfer and S. Turek. *Flow Simulation with High-Performance Computer II*, volume 52 of *Notes on Numerical Fluid Mechanics*, chapter Benchmark Computations of laminar flow around a cylinder. Vieweg, Braunschweig Wiesbaden, 1996.

[27] G. Strang. *Computational Science and Engineering.* Wellesley-Cambridge Press, 2007.

[28] T. Wick. *Multiphysics Phase-Field Fracture: Modeling, Adaptive Discretizations, and Solvers.* Radon Series on Computational and Applied Mathematics, de Gruyter, Vol. 28, 2020.

[29] T. Wick. Numerical methods for partial differential equations. Hannover : Institutionelles Repositorium der Leibniz Universität Hannover, DOI: https://doi.org/10.15488/9248, January 2020.

[30] T. Wick. Sergey I. Repin, Stefan A. Sauter: Accuracy of Mathematical Models. *Jahresber. Dtsch. Math. Ver.*, 122:269–274, 2020.

[31] Wikipedia. Wikipedia. `https://www.wikipedia.org/`. Online; accessed XX XX 2021.

Index

A-stability, 170
Accuracy, 13, 22
Algorithm
  Analysis, 13
Algorithms, 22
Ansatz function, 197
Armijo step length rule, 113
Assembling in FEM, 210
Assembling integrals, 210

Ball, 62
Banach spaces, 65
basis, 43
Big O, 20
Bilinear form, 196
  Norm, 197
  Scalar product, 197
bisection, 120
Boundary conditions, 185
Boundary value problem, 191
Bounded operator, 189
Box rule, 212
BVP, 191

Change of variables, 106
closed set, 63
Compact support, 195
compatible norms, 56
complex numbers, 28
Computational convergence, 246
Computational convergence analysis, 233, 246
Computational cost LU decomposition, 48
Computational cost LU solution, 46
Concepts
  In numerics, 14
Condition number
  Matrix, 68, 69
condition number, 26
Conditional stability, 244
cone, 87
Consistency, 168, 171
Constitutive laws, 176
Continuous mapping, 189
Convergence, 171
convergence (sequence), 64
Convergence order
  Computationally obtained, 248
Critical point, 110
Cross product, 104

damping parameter, 72
Defect, 124
Degrees of freedom, 216
descent method, 73
determinant, 43
Diagonal dominant, 55
Difference quotient, 141
Differential equation
  Classification, 188
  General definition, 187
  Linear vs. nonlinear, 188
Differential equations, 21
Differential problem, 192
Differentiation under the integral sign, 178
Diffusion, 175
dimension, 43
Direct solution linear systems, 35
Dirichlet condition, 185
Discrete Fourier transform, 147
Discretization error, 17, 248
Divergence, 103
DoFs, 216
double precision, 25

Efficiency, 13, 22
Eigenvalues
  Poisson, 225
Eigenvectors
  Poisson, 225
Elasticity, 179
Elliptic operator, 182
Elliptic PDE, 181
equivalent norms, 64
Error, 17
  Discretization, 17
  Implementation, 17
  Interpolation, 140
  Model, 17
  Numerical quadrature, 17
  Programming, 17

Regularization, 17
Round-off, 17
Errors in numerical mathematics, 17
Essential boundary conditions, 186
Euler
Backward, 167
Implicit, 167
Euler method, 166
Euler-Lagrange equations, 193
Explicit schemes, 166
Extrapolation to the limit, 250

Fast Fourier transform, 147
FD, 165
FE, 191
FEM
2D simulations, 233
3D simulations, 234
Computational convergence analysis, 234
Quadratic, 219
FFT, 147
Finite differences, 165
Finite element
Definition, 205
Finite elements, 191, 199
Computational convergence analysis, 234
Construction of manufactured solution, 192
Construction of right hand side, 234
Error analysis, 234
Error analysis in simulations, 234
Manufactured solution, 234
Conforming, 201
Evaluation of integrals, 203
Linear, 200
Mesh, 200
Weak form, 202
fixed point, 121
Fixed point methods, 70
floating-point numbers, 24
Fluid flow equations, 180
Forward Euler method, 166
Fourier series, 149
Convergence, 149
Fourier transform, 150
Frobenius matrix, 47
Frobenius scalar product, 197
Fundamental lemma of calculus of variations, 195

Galerkin equations, 203
Galerkin method, 203
Galerkin orthogonalty, 74
Gauss elimination, 46
Gauss quadrature, 145
Gauss-Seidel method, 72
Gaussian elimination, 35
generalized minimal residual, 80
Geometric multigrid, 227
GMRES, *see* generalized minimal residual
Gradient descent, 73
gradient descent
projection, 116
Growth of a species, 161
Guiding questions
In differential equations, 21

Harmonic function, 182
Hat function, 200
Heat equation, 181, 183, 238
Hermitian matrix, 39, 58
Hermitian positive definite matrix, 55
Hessian, 104
Higher-order FEM, 219
Hilbert spaces, 65
Homogeneous Dirichlet condition, 181
Hyperbolic PDE, 181

IBVP, 183, 238, 254
Implicit schemes, 167
induced norm, 56
Integration by parts, 193
Interpolation
Polynomial, 139
Interpolation error, 140
Iteration error, 251
Iterative solution, 70
IVP, 165

Jacobi method, 72
Jacobian, 103
Jacobian matrix, 127

Krylov space, 74

Landau symbols, 20
Laplace equation, 181
Lax-Milgram lemma, 197
Lemma
Lax-Milgram, 197
Line search, 133
linear convergence, 119
Linear form, 196
Linear functional, 189
Linear operator, 188
Linear PDE, 188
Linear systems, 35
Direct solution, 35
LU decomposition, 46

Little o, 20
LU decomposition, 46

machine epsilon, 25
Machine learning, 136
machine precision, 25
Mass matrix, 240
Master element, 215, 217
Mathematical modeling, 13
Matrices, 35, 37
  Invertible, 41
  Properties, 37
matrix norm, 56, 66
Metric, 18
Metric space, 18
metric space, 18
Midpoint rule, 212
Minimization problem, 192
Minimum
  strict, 111
Multigrid, 227
Multiindex notation, 103

Nabla operator, 103
Natural boundary conditions, 186
Navier-Stokes equations, 180
Neumann condition, 185
Neural network, 136
Newton globalization, 133
Newton's method, 123, 134
  Globalization, 131
  Defect-correction, 124
  overview, 124
Newton-Kantorovich theorem, 127
Nonlinear PDE, 188
norm, 55, 62, 66
Normal cone, 88
Normed space, 18
normed space, 19
Norms, 18
numbers, 24
Numerical analysis
  Heat equation, 242
  ODE, 168
Numerical concepts, 14, 22
  Guiding questions, 21
numerical differentiation, 141
  first derivative, 141
  second derivative, 142
Numerical integration, 142, 212
Numerical methods, 13
numerical modeling, 13
Numerical projects, 259
Numerical quadrature, 142, 212
  Interpolatory, 144

ODE, 165
  Newton's method, 134
  Nonlinear coupled system, 134
  System, 134
  Model problem, 162
One-step schemes, 166
One-Step-Theta scheme, 238
One-step-theta scheme, 238
Open set, 63
Optimality conditions, 110
order of convergence, 119
Overflow, 25

Parabolic PDE, 181
Parseval equality, 150
PDE, 191
  Elliptic, 181
  Hyperbolic, 181
  Parabolic, 181
Petrov-Galerkin method, 203
Pivot element, 50
Pivot matrix, 51
Pivoting, 50
Poisson
  1D simulations, 206, 220
  2D simulations, 233
  3D simulations, 234
Poisson equation, 181
Polynomial interpolation, 139
Population, 161
Positive definite matrix, 40
positive orthant, 87
Principle of minimum potential energy, 193
Principle of virtual work, 193
Programming, 259

quadratic convergence, 119
Quadratic FEM, 219
Quadratur, 212
Quadrature formula, 142
  Composite, 144
  Order, 144
Quadrature weights, 212

real numbers, 24
Reference element, 215, 217
Relationship mesh size and DoFs, 247
relaxation parameter, 72
Research software, 13
Reynolds' transport theorem, 178

Richardson, 224
Richardson extrapolation, 250
Ritz method, 203
Ritz-Galerkin method, 203
Robin condition, 185
Robustness, 13, 22

Saddle point, 111
Scalar product, 197
scientific computing, 13
Search direction, 72
set, 23
Shape functions, 200
Shape functions, 203
single precision, 25
Smooth functions, 195
Smoother, 224
    Richardson, 224
    SOR, 230
Software, 13
Software development, 259
Solid mechanics equation, 179
spectral norm, 56
spectral radius, 58, 59, 66
Stability, 168, 169
stability analysis, 242
Stationary point, 110
Stiff PDEs, 239
stiff problem, 242
Stiffness, 165
Stiffness matrix, 240
Symmetric matrix, 39
symmetric matrix, 58

Test function, 193, 197
tolerance, 25
Total potential energy, 193
Trapezoidal rule, 212
Trial function, 197
Trigonometric interpolation, 147
Two grid method, 229

Unconditionally stable, 244
Underflow, 25

V-cycle, 230
Variational multigrid, 227
Variational problem, 192
vector space, 61
vectors, 37

W-cycle, 230
Wave equation, 181, 184
well-posedness, 17