Introduction to Numerical Modeling

Thomas Wick

Centre de Mathématiques Appliquées (CMAP)

École Polytechnique (X)

91128 Palaiseau, France

thomas.wick@polytechnique.edu

Last update: Sunday, November 12, 2017

Foreword

These notes accompany the class

MAP 502 - A project on numerical modeling done in two

given at Ecole Polytechnique.

The specific focus will be on modeling and numerical methods for **differential equations** as they are extremely important in a wide range of applications.

The **goals** of this class are:

- To provide a basic understanding of mathematical modeling in terms of differential equations, their classification, and additional ingredients such as initial and boundary conditions;
- To discretize these equations in time and space in order to prepare them for their computer implementation;
- To illustrate numerical results of such implementations including programming codes;
- To learn how to analyze and interprete the obtained results. Do they make sense? What are typical error sources? What is the computational complexity? How fast do we approach a final solution? What is the rate of convergence?

The prerequisites are lectures in calculus, linear algebra and possibly an introduction to numerics or as well classes on ODE/PDE theory. Moreover, these notes shall serve as a preparation for subsequent project work done in groups of two. These projects will have topics involving ODEs and PDEs and consist of mathematical modeling, numerical analysis and programming of a given problem. Programming shall be preferably undertaken in open source languages octave (the open source sister of MATLAB), python, or C++ (pure or in form of open-source finite element packages deal.II or FreeFem++). Thus, these projects deal with **scientific computing** on a basic level.

I would be more than grateful if you let me know about errors inside these lecture notes via

`thomas.wick@polytechnique.edu`

This is a good place, to express my thankfulness to my students from both years in the STEEM program at Ecole Polytechnique, namely the winter term 2016/2017 and the winter term 2017/2018, since they asked many questions from which I understood which parts of my lecture and lecture notes needed improvements. Moreover, several misprints have been found by them.

Thanks to you'all!

Thomas Wick

(Palaiseau, November 2017)

Contents

1 Literature

1. T. Richter, T. Wick [21]: Introduction to numerical mathematics (in German).
2. G. Strang: Computational Science and Engineering [23].
3. G. Allaire: Introduction to mathematical modelling, numerical simulation, and optimization [2] (in English; original version in French).
4. Ch. Grossmann, H.-G. Roos, M. Stynes: Numerical treatment of partial differential equations [14] (in English; original version in German).
5. M. Hanke-Bourgeois: Foundations of numerical mathematics and scientific computing [15] (in German).
6. G. Allaire, F. Alouges: Analyse variationnelle des équations aux dérivées partielles [3] (in French).
7. L.C. Evans: Partial differential equations [11].

2 Notation

2.1 Domains

We consider open, bounded domains $\Omega \subset \mathbb{R}^d$ where $d = 1, 2, 3$ is the dimension. The boundary is denoted by $\partial\Omega$. The outer normal vector with respect to (w.r.t.) to $\partial\Omega$ is n. We assume that Ω is sufficiently smooth (i.e., a Lipschitz domain or domain with Lipschitz boundary) such that n can be defined. What also works for most of our theory are convex, polyhedral domains with finite corners. For specific definitions of nonsmooth domains, we refer to the literature, e.g., [13].

2.2 Independent variables

A point in $\mathbb{R}^d$ is denoted by

$$x = (x_1, \ldots, x_d).$$

The variable for 'time' is denoted by t. The Euclidian scalar product is denoted by (a, b) or via $a \cdot b$.

2.3 Function, vector and tensor notation

Functions are denoted by

$$u = u(x)$$

if they only depend on the spatial variable $x = (x_1, \ldots, x_d)$. If they depend on time and space, they denoted by

$$u = u(t, x).$$

Usually in physics or engineering vector-valued and tensor-valued quantities are denoted in bold font size or with the help of arrows. Unfortunately in mathematics, this notation is only sometimes adopted. We continue this crime and do not distinguish scaler, vector, and tensor-valued functions. Thus for points in $\mathbb{R}^3$ we write:

$$x := (x, y, z) = \mathbf{x} = \vec{x}.$$

Similar for functions from a space $u : \mathbb{R}^3 \supseteq U \to \mathbb{R}^3$:

$$u := (u_x, u_y, u_z) = \mathbf{u} = \vec{u}.$$

And also similar for tensor-valued functions (which often have a bar under the tensor quantity) as for example the Cauchy stress tensor $\sigma_f \in \mathbb{R}^{3\times 3}$ of a fluid:

$$\sigma_f := \underline{\sigma}_f = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix}.$$

2.4 Partial derivatives

We frequently use:

$$\frac{\partial u}{\partial x} = \partial_x u$$

and

$$\frac{\partial u}{\partial t} = \partial_t u$$

and

$$\frac{\partial^2 u}{\partial t \partial t} = \partial_t^2 u.$$

2.5 Multiindex notation

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 2.1. *Let $\alpha = (2,0,1)$. Then $|\alpha| = 3$ and $D^\alpha u(x) = \partial_x^2 \partial_z^1 u(x)$.*

2.6 Gradient, divergence, Laplace, rotation

The gradient of a single-valued function $v : \mathbb{R}^n \to \mathbb{R}$ reads:
$$\nabla v = \begin{pmatrix} \partial_1 v \\ \vdots \\ \partial_n v \end{pmatrix}.$$
The divergence is defined for vector-valued functions $v : \mathbb{R}^n \to \mathbb{R}^n$:
$$\operatorname{div} v := \nabla \cdot v := \nabla \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \sum_{k=1}^n \partial_k v_k.$$
The divergence for a tensor $\sigma \in \mathbb{R}^{n\times n}$ is defined as:
$$\nabla \cdot \sigma = \Big(\sum_{j=1}^n \frac{\partial \sigma_{ij}}{\partial x_j}\Big)_{1\le i\le n}.$$
The trace of a matrix $A \in \mathbb{R}^{n\times n}$ is defined as
$$tr(A) = \sum_{i=1}^n a_{ii}.$$
The Laplace operator of a two-times continuously differentiable scalar-valued function $u : \mathbb{R}^n \to \mathbb{R}$ is defined as
$$\Delta u = \sum_{k=1}^n \partial_{kk} u.$$

Definition 2.2. *For a vector-valued function $u : \mathbb{R}^n \to \mathbb{R}^m$, we define the Laplace operator component-wise as*
$$\Delta u = \Delta \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^n \partial_{kk} u_1 \\ \vdots \\ \sum_{k=1}^n \partial_{kk} u_m \end{pmatrix}.$$

Let us next introduce the rotation of two vectors $u, v \in \mathbb{R}^3$:
$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix}.$$

2.7 Invariants of a matrix

The principal invariants of a matrix A are the coefficients of the characteristic polynomial $\det(A-\lambda I)$. A matrix $A \in \mathbb{R}^{3\times 3}$ has three principal invariants; namely $i_A = (i_1(A), i_2(A), i_3(A)$ with

$$\det\,(\lambda I - A) = \lambda^3 - i_1(A)\lambda^2 + i_2(A)\lambda - i_3(A).$$

Let $\lambda_1, \lambda_2, \lambda_3$ be the eigenvalues of A, we have

$$\begin{aligned} i_1(A) &= \text{tr}\,(A) = \lambda_1 + \lambda_2 + \lambda_3, \\ i_2(A) &= \frac{1}{2}(\text{tr}\,A)^2 - \text{tr}\,(A)^2 = \lambda_1\lambda_1 + \lambda_1\lambda_3 + \lambda_2\lambda_3, \\ i_3(A) &= \det\,(A) = \lambda_1\lambda_2\lambda_3. \end{aligned}$$

Remark 2.3. *If two different matrices have the same principal invariants, they also have the same eigenvalues.*

Remark 2.4 (Cayley-Hamilton)**.** *Every second-order tensor (i.e., a matrix) satisfies its own characteristic equation:*

$$A^3 - i_1A^2 + i_2A - i_3I = 0.$$

2.8 Normed spaces

Let X be a linear space. The mapping $||\cdot|| : X \to \mathbb{R}$ is a norm if

i)	$\lVert x\rVert \geq 0 \quad \forall x \in X$	(Positivity)
ii)	$\lVert x\rVert = 0 \Leftrightarrow x = 0$	(Definiteness)
iii)	$\lVert \alpha x\rVert = \lvert\alpha\rvert\,\lVert x\rVert, \quad \alpha \in \mathbb{K}$	(Homogeneity)
iv)	$\lVert x+y\rVert \leq \lVert x\rVert + \lVert y\rVert$	(Triangle inequality)

A space X is a normed space when the norm properties are satisfied. If condition ii) is not satisfied, the mapping is called a semi-norm.

2.9 Inequalities

Young's inequality:

$$ab \leq \frac{a^p}{p} + \frac{b^q}{q}, \quad a, b > 0, \quad \frac{1}{p} + \frac{1}{q} = 1.$$

2.10 Little o and big O - the Landau symbols

Definition 2.5 (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 $f \in o(g)$.*

Example 2.6. *1. Let $\varepsilon \to 0$ and $h \to 0$. We write*

$$h = o(\varepsilon)$$

when

$$\frac{h}{\varepsilon} \to 0 \quad \textit{for } h \to 0, \quad \varepsilon \to 0,$$

which means that h tends faster to 0 than ε.

2. *Let us assume that we have the error estimate (see Section 4.5.3)*

$$\|y(t_n) - y_n\|_2 = O(k).$$

Here the O notation means nothing else than

$$\frac{\|y(t_n) - y_n\|_2}{k} \to C \quad \textit{for } k \to 0.$$

Here the fraction converges to a constant C (and not necessarily 0!), which illustrates that O-convergence is weaker than o convergence.

2.11 Taylor expansion

The Taylor series of a function $f \in C^\infty$ developed at a point $a \neq x$ reads:

$$T(f(x)) = \sum_{j=0}^{\infty} \frac{f^{(j)}(a)}{j!}(x-a)^j.$$

3 Motivation, characteristics, and examples of differential equations

3.1 Why numerical simulations?

The key question is why we need numerical simulations? The answer is that in fact in recent decades thanks to the developments of computers, a third pillar of science has been established that sits between experiments and theory:

numerical modeling
id est
scientific computing.

Scientific computing comprises three main fields:

- **Mathematical modeling** and analysis of physical, biological, chemical, economical, financial processes, and so forth;
- Development of reliable and efficient **numerical methods** and algorithms;
- Implementation of these algorithms into a **software**.

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, *numerical mathematics/scientific computing*, has been established. This kind of mathematics is experimental like experiments in physics/chemistry/biology.

Therefore, numerical modeling offers to investigate research fields that have partially not been addressable. 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!

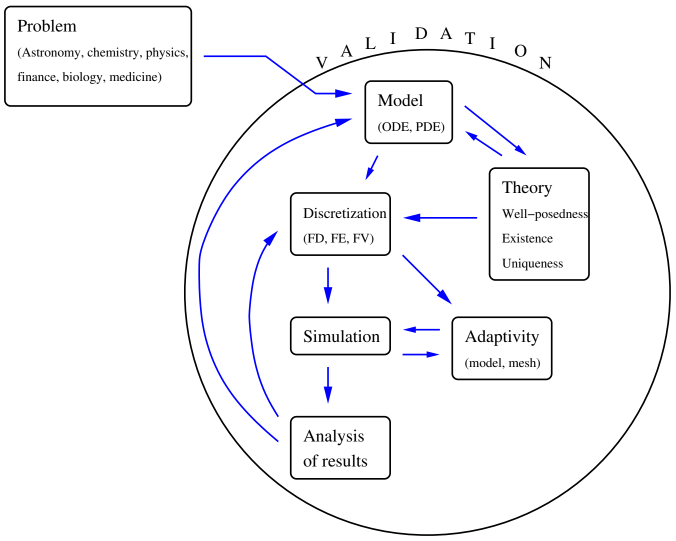

Figure 1: The third pillar of science between theory and experiments: scientific computing.

Thanks to progress in computer technology, scientific computing (or numerical modeling or computational science) plays a major role in nowadays research. However, there is a great danger that research becomes superficial because it sounds easy just to push the button of a computer, run the software/program, and everything is solved. A careful analysis of the underlying algorithms and the obtained solutions is indispensable. Here, it is in particular an advantage to activate imagination and physical sense. If something is weird in scientific computing, it can often already be identified by just looking into the graphical solution and double-checking physical plausibility. On the other hand, a certain depth in theory is helpful as well to judge simulation results. Thus a peculiar challenge working in this field is that an intrinsic interdisciplinary motivation and education must be present by these researchers.

The goal of this lecture is to restrict our focus to **differential equations**. These have endless applications in science and engineering and serve perfectly to provide an idea of numerical modeling. Let us first roughly define the meaning of a differential equation:

Definition 3.1. *A differential equation is a mathematical equation that relates the function with its derivatives.*

Differential equations can be split into two classes:

Definition 3.2 (Ordinary differential equation (ODE))**.** *An ordinary differential equation (ODE) is an equation (or equation system) involving an unknown function of one independent variable and certain of its derivatives.*

Definition 3.3 (Partial differential equation (PDE))**.** *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.*

A solution u of a differential equation is in most cases (except for simple academic test cases in which a manufactured solution can be constructed) computed with the help of discretization schemes generating a sequence of approximate solutions $\{u_h\}_{h\to 0}$. Here h is the so-called discretization parameter. For $h \to 0$ we (hopefully) approach the continuous solution u.

Important questions are

- what kind of discretization scheme shall we use?
- how do we design algorithms to compute u_h?
- how far is u_h away from u in a certain (error) norm?
- the discretized systems (to obtain u_h) are often large with a huge number of unknowns: how do we solve these linear equation systems?
- what is the computational cost?

In order to realize these algorithmic questions, we go ahead and implement them in a software (for instance Matlab/octave, python, fortran, C++, Java) using a computer. Here, three major error sources that need to be addressed:

- The set of numbers is finite and a calculation is limited by machine precision, which results in **round-off errors**.
- 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**.
- All further simplifications of a numerical algorithm (in order to solve the discrete problem), with the final goal to reduce the computational time, are so-called **systematic errors**. One example is the stopping criterion after how many steps an iteration is stopped.

Finally, there are **model errors** such as:

- In order to make a ‘quick guess’ of a possible solution and to start development of an algorithm to address at a later stage a difficult problem, often complicated (nonlinear) differential equations are reduced to simple (in most cases linear) versions, which results in the so-called **model error**.
- **Data errors**: the data (e.g., input data, boundary conditions, parameters) are finally obtained from experimental data and may be inaccurate themselves.

3.2 Concepts in numerical mathematics

In [21], we defined seven concepts that are very characteristic for numerical modeling and will frequently encounter us in the forthcoming chapters:

1. **Approximation**: since analytical solutions are not possible to achieve as we just learned in the previous section, solutions are obtained by **numerical approximations**.

2. **Convergence**: is a qualitative expression that tells us when members a_n of a sequence $(a_n)_{n\in\mathbb{N}}$ are sufficiently close to a limit a. In numerical mathematics this limit is often the solution that we are looking for.

3. **Order of convergence**: While in analysis are often interested in the convergence itself, in numerical mathematics we must pay attention how long it takes until a numerical 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. In order to judge whether a algorithm is fast or not we have to determine the order of convergence.

4. **Errors**: Numerical mathematics can be considered as the branch ‘mathematics of errors’. What does this mean? Numerical modeling is not wrong, inexact or non-precise! Since we cut sequences after a final number of steps or accept sufficiently accurate solutions obtained from our software, we need to say how well this numerical solution the (unknown) exact solution approximates. In other words, we need to determine the error, which can arise in various forms as we discussed in the previous section.

5. **Error estimation**: This is one of the biggest branches in numerical mathematics. We need to derive error formulae to judge the outcome of our numerical simulations and to measure the difference of the numerical solution and the (unknown) exact solution in a certain norm.

6. **Efficiency**: In general we can say, the higher the convergence order of an algorithm is, the more efficient our 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) will definitely yield in less CPU (central processing unit) time a numerical solution. However, if a parallel machine does need less electricity (and thus less money) than a sequential desktop machine/code is a priori unclear.

7. **Stability**: Despite being the last concept, in most developements, this is the very first step to check. How robust is our algorithm against different model and physical parameters? Is the algorithm stable with respect to different input data? This condition relates in the broadest sense to the third condition of Hadamard defined in Section 3.4.

3.3 Which topics these notes do not cover

These notes do not cover other important fields of numerical modeling such as:

- Numerical solution of linear equation systems;
- Interpolation and approximation (however they appear implicitly e.g., by approximating differential equations in terms of difference quotients);
- Numerical integration/quadrature (short remarks exist though);
- Numerical schemes for orthogonalization (such as Gram-Schmidt);
- Numerical computation of eigenvalue problems.

These topics can be found, for instance, in the literature provided at the beginning in Section 1.

3.4 Well-posedness

The concept of well-posedness 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 if a variation of the input data (right hand side, boundary values, initial conditions) vary only a little bit, then also the (unique) solution should only vary a bit.

3.5 Examples of differential equations

3.5.1 Some applications

We start with some examples:

- An ODE example (initial value problem). 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 extend at least by simple models, but 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 further augmented with further terms and coefficients. In the final 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:
  $$\lim_{t\to 0} \frac{dy}{dt} = (g-m)y$$
  and thus
  $$y' = (g-m)y.$$
  For this ordinary differential equation (ODE), we can even explicitely compute the solution:
  $$y(t) = c\exp((g-m)(t-t_0)).$$
  This can be achieved with separation of variables:
  $$y' = \frac{dy}{dt} = (g-m)y \tag{1}$$
  $$\Rightarrow \int \frac{dy}{y} = \int_{t_0}^{t} (g-m)dt \tag{2}$$
  $$\Rightarrow Ln|y| + C = (g-m)(t-t_0) \tag{3}$$
  $$\Rightarrow y = \exp[C]\cdot\exp[(g-m)(t-t_0)] \tag{4}$$
  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 $\exp(C)$:

$$y(t_0) = \exp(C)\exp[(g-m)(t_0-t_0)] = \exp(C) = y_0. \tag{5}$$

Let us say in the year $t_0 = 2011$ there have been two members of this species: $y(2011) = 2$. 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 $t = 2014$ and $t = 2022$. We first obtain:

$$y(2014) = 2\exp(0.25 * (2014 - 2011)) = 4.117 \approx 4.$$

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:

$$y(2022) = 2\exp(0.25 * (2022 - 2011)) = 31.285 \approx 31.$$

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.

- Laplace's equation (boundary value problem) . This equation has several applications, e.g., Fick's law of diffusion or heat conduction (temporal diffusion) or the deflection of a solid membrane:

Formulation 3.4 (Laplace problem / Poisson problem)**.** *Let Ω be an open set. The Laplace problem reads:*

$$-\Delta u = 0 \quad \text{in } \Omega.$$

The Poisson problem reads:

$$-\Delta u = f \quad \text{in } \Omega.$$

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 of u through the boundary ∂G is zero:

$$\int_{\partial G} F \cdot n \, dx = 0. \tag{6}$$

Here F denotes the flux density and n the outer normal vector. Gauss' divergence theorem yields:

$$\int_{\partial G} F \cdot n \, dx = \int_G \nabla \cdot F \, dx = 0.$$

Since this integral relation holds for arbitrary G, we obtain

$$\nabla \cdot F = 0 \quad \text{in } \Omega. \tag{7}$$

Now we need a second assumption (or better a relation) between the flux and the quantity u. In many situations it is reasonable to assume that the flux is proportional to the negative gradient $-\nabla u$ of the quantity u. 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 (7) yields:

$$\nabla \cdot F = \nabla \cdot (-\nabla u) = -\nabla \cdot (\nabla u) = -\Delta u = 0.$$

This is the simplest derivation one can make. Adding a constant (material) parameter would yield:

$$\nabla \cdot F = \nabla \cdot (-a\nabla u) = -\nabla \cdot (a\nabla u) = -a\Delta u = 0.$$

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.

- 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 (6)) 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 referene configuration $\hat{G}$ in which we can insert the time derivative under the integral sign. Then we perform the calculation and transform in 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 expansian 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:

$$\int_G(\partial_t\rho+\nabla\cdot F)\,dx=\int_G(\partial_t\rho+\nabla\cdot(\rho v)\,dx.$$

This is the so-called continuity equation (or mass conservation). Since G was arbitrary, we are allowed to write the strong form:

$$\partial_t\rho+\nabla\cdot(\rho v)=0\quad\text{in }\Omega. \tag{8}$$

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 (8) can be made when certain requirement 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 3.5 (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 comes 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:*

$$\frac{d}{dt}\int_\Omega\Phi\,dx=\int_\Omega\nabla\cdot(\Phi\,v)\,dx+\int_\Omega\frac{\partial\Phi}{\partial t}\,dx.$$

- Elasticity (Lamé-Navier) : elastic deformation of a solid. This example is already difficult because a system of nonlinear equations is considered:

  Formulation 3.6. *Let $\widehat{\Omega}_s \subset \mathbb{R}^n, n = 3$ with the boundary $\partial\widehat{\Omega} := \widehat{\Gamma}_D \cup \widehat{\Gamma}_N$. Furthermore, let $I := (0, T]$ where $T > 0$ is the end time value. The equations for geometrically non-linear elastodynamics in the reference configuration $\widehat{\Omega}$ are given as follows: Find vector-valued displacements $\hat{u}_s := (\hat{u}_s^{(x)}, \hat{u}_s^{(y)}, \hat{u}_s^{(z)}) : \widehat{\Omega}_s \times I \to \mathbb{R}^n$ such that*

$$\begin{aligned}
\hat{\rho}_s \partial_t^2 \hat{u}_s - \widehat{\nabla} \cdot (\widehat{F}\widehat{\Sigma}) - \partial_t \widehat{\nabla} \cdot (\widehat{\Sigma}_v) = 0 \quad &\text{in } \widehat{\Omega}_s \times I,\\
\hat{u}_s = 0 \;\; &\text{on } \widehat{\Gamma}_D \times I,\\
\widehat{F}\widehat{\Sigma} \cdot \hat{n}_s = \hat{h}_s \;\; &\text{on } \widehat{\Gamma}_N \times I,\\
\hat{u}_s(0) = \hat{u}_0 \;\; &\text{in } \widehat{\Omega}_s \times \{0\},\\
\hat{v}_s(0) = \hat{v}_0 \;\; &\text{in } \widehat{\Omega}_s \times \{0\}.
\end{aligned}$$

  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 [7]):

$$\widehat{\Sigma} = \widehat{\Sigma}_s(\hat{u}_s) = 2\mu\widehat{E} + \lambda tr(\widehat{E})I, \quad \widehat{E} = \frac{1}{2}(\widehat{F}^T\widehat{F} - I). \tag{9}$$

  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 $\widehat{F} = \hat{I} + \widehat{\nabla}\hat{u}_s$ where $\hat{I} \in \mathbb{R}^{3\times 3}$ is the identity matrix. Furthermore, $\hat{n}_s$ denotes the normal vector.

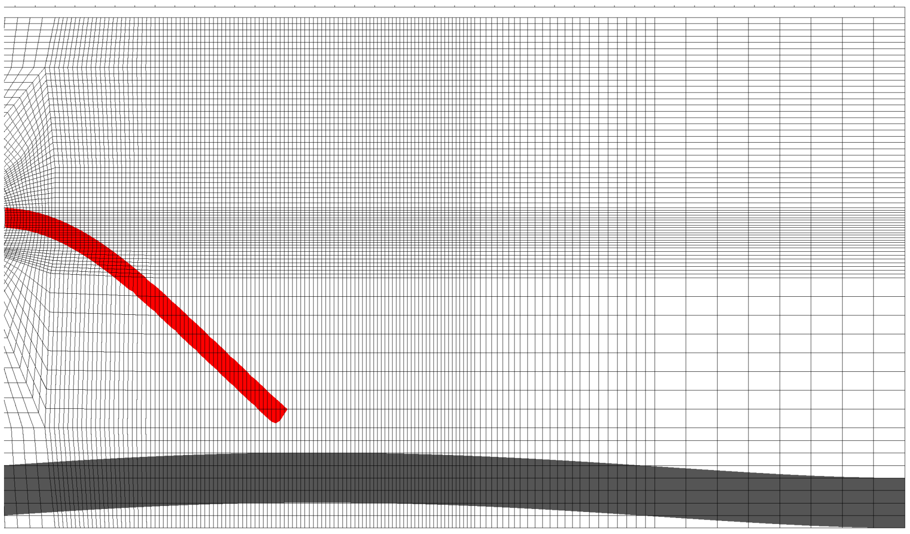

Figure 2: Prototype example of two deforming solids (elasticity). The underlying code is based on deal.II [4] www.dealii.org.

- Navier-Stokes problem (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 3.7. *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*

$$\begin{aligned}
\rho_f \partial_t v_f + \rho_f v_f \cdot \nabla v_f - \nabla \cdot \sigma_f(v_f, p_f) &= 0 \quad \text{in } \Omega_f \times I, && (10)\\
\nabla \cdot v_f &= 0 \quad \text{in } \Omega_f \times I, && (11)\\
v_f^D &= v_{in} \ \text{on } \Gamma_{in} \times I, && (12)\\
v_f &= 0 \ \text{on } \Gamma_D \times I, && (13)\\
-p_f n_f + \rho_f \nu_f \nabla v_f \cdot n_f &= 0 \ \text{on } \Gamma_{out} \times I, && (14)\\
v_f &= h_f \ \text{on } \Gamma_i \times I, && (15)\\
v_f(0) &= v_0 \ \text{in } \Omega_f \times \{t = 0\}, && (16)
\end{aligned}$$

  where the (symmetric) Cauchy stress is given by

$$\sigma_f(v_f, p_f) := -pI + \rho_f \nu_f (\nabla v + \nabla v^T),$$

  with the density ρ_f and the kinematic viscosity ν_f. The normal vector is denoted by n_f.

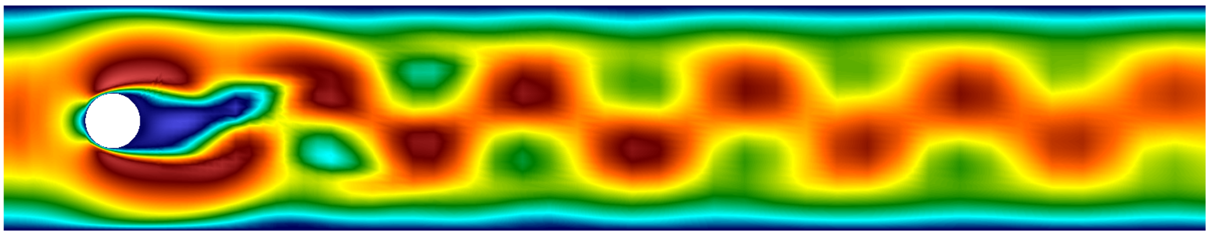

Figure 3: Prototype example of a fluid mechanics problem (isothermal, incompressible Navier-Stokes equations): the famous Karman vortex street. The setting is based on the benchmark setting [22] and the code can be found in NonStat Example 1 in [12] www.dopelib.net.

Remark 3.8. *The two Formulations 3.6 and 3.7 are very important in many applications and their coupling results in fluid-structure interaction. Here we notice that flow fluids 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., $\widehat{\Omega} \simeq \Omega$. 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.*

Exercise 1. *Recapitulate (in case you have had classes on continuum mechanics) the differences between Lagrangian and Eulerian coordinates.*

3.5.2 One important ODE

In the first of the previous examples (setting $a = g - m$), we introduced one of the most important ODEs:

$$y' = ay, \quad y(t_0) = y_0.$$

This ODE serves often as model problem and important concepts such as existence, uniqueness, stability are usually introduced in terms of this ODE.

3.5.3 Three important PDEs

From the previous considerations, we can extract three important types of PDEs. But we refrain to give the impression that all differential equations can be classified and then a receipt for solving them applies. This is definitely not true. However, in particular for PDEs, we are often faced with three outstanding types of equations and often 'new' equations can be related or simplified to these three types.

3.5.3.1 The PDEs 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. The problem is given by:

Formulation 3.9. *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*

$$Lu = f \quad \text{in } \Omega, \tag{17}$$

$$\text{and bc and ic.} \tag{18}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} a_{ij}(x) \frac{\partial^2 u}{\partial x_i \partial x_j}$$

with the symmetry assumption $a_{ij} = a_{ji}$.

The operator L 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:

$$A = (a_{ij})_{ij=1}^{n}.$$

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 (the matrix A is positive definite) and have the same sign: Equivalently one can say:

Definition 3.10. *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 $\xi \in \mathbb{R}^n$.

For parabolic PDEs one eigenvalue of A is zero whereas the others have the same sign. Finally, a hyperbolic equation has one eigenvalue that has a different sign (negative) than all the others (positive).

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 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). In this case the coefficients a_{ij} of the operator L are non-constant and change with respect to space and time.

3.5.3.2 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 3.11. *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*

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, && (19)\\ u &= 0 \quad \text{on } \partial\Omega. && (20) \end{aligned}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x)\partial_{x_i}u) + \sum_{i=1}^{n} b_i(x)u\partial_{x_i} + c(x)u,$$

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 (20) *is often called homogeneous Dirichlet condition.*

Remark 3.12. *If the coefficient functions a, b, c also depend on the solution u we obtain a nonlinear PDE.*

Formulation 3.13 (Poisson problem)**.** *Setting in Formulation 3.11, $a_{ij} = \delta_{ij}$ 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*

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, && (21)\\ u &= 0 \quad \text{on } \partial\Omega. && (22) \end{aligned}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\nabla \cdot (\nabla u) = -\Delta u.$$

In the following we discuss some characteristics of the solution of the Laplace problem, which can be generalized to general elliptic problems.

Theorem 3.14 (Strong maximum principle for the Laplace problem)**.** *Suppose $u \in C^2(\Omega) \cap C(\bar{\Omega})$ is a solution of Laplace problem. Then*

$$\max_{\bar{\Omega}} u = \max_{\partial\Omega} u.$$

From the maximum principle we obtain immediately uniqueness of a solution

Theorem 3.15 (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:*

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, && (23)\\ u &= g \quad \text{on } \partial\Omega. && (24) \end{aligned}$$

3.5.3.3 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:

Formulation 3.16. *Let $f : \Omega \times I \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that*

$$\begin{aligned} \partial_t u + Lu &= f \quad \text{in } \Omega \times I, && (25)\\ u &= 0 \quad \text{on } \partial\Omega \times [0,T], && (26)\\ u &= g \quad \text{on } \Omega \times \{t=0\}. && (27) \end{aligned}$$

Here, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x,t)\partial_{x_i} u) + \sum_{i=1}^{n} b_i(x,t) u \partial_{x_i} + c(x,t)u,$$

for given (possibly spatial and time-dependent) coefficient functions a_{ij}, b_i, c.

We have the following:

Definition 3.17. *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 $0 \leq t \leq T$.

Formulation 3.18 (Heat equation)**.** *Setting in Formulation 3.16, $a_{ij} = \delta_{ij}$ 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*

$$\begin{aligned} \partial_t u + Lu &= f \quad \text{in } \Omega \times I, && (28)\\ u &= 0 \quad \text{on } \partial\Omega \times [0,T], && (29)\\ u &= g \quad \text{on } \Omega \times \{t=0\}. && (30) \end{aligned}$$

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 non-negative 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.

Theorem 3.19 (Strong 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 heat equation. Then*

$$\max_{\bar{\Omega}\times I} u = \max_{\partial\Omega} u.$$

And also as for elliptic problems one can proof uniqueness for the parabolic case in a similar way.

3.5.3.4 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 3.20. *Let $f : \Omega \times I \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that*

$$\begin{aligned} \partial_t^2 u + Lu &= f \quad \text{in } \Omega \times I, && (31)\\ u &= 0 \quad \text{on } \partial\Omega \times [0,T], && (32)\\ u &= g \quad \text{on } \Omega \times \{t=0\}, && (33)\\ \partial_t u &= h \quad \text{on } \Omega \times \{t=0\}. && (34) \end{aligned}$$

In the last line, $\partial_t u = v$ can be identified as the velocity. Furthermore, the linear second-order differential operator is defined by:

$$Lu := -\sum_{i,j=1}^{n} \partial_{x_j}(a_{ij}(x,t)\partial_{x_i}u) + \sum_{i=1}^{n} b_i(x,t)u\partial_{x_i} + c(x,t)u$$

for given (possibly spatial and time-dependent) coefficient functions a_{ij}, b_i, c.

Remark 3.21. *Actually 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*

$$\partial_t v + Lu = f \quad \text{in } \Omega \times I, \tag{35}$$
$$\partial_t u = v \quad \text{in } \Omega \times I, \tag{36}$$
$$u = 0 \quad \text{on } \partial\Omega \times [0,T], \tag{37}$$
$$u = g \quad \text{on } \Omega \times \{t = 0\}, \tag{38}$$
$$v = h \quad \text{on } \Omega \times \{t = 0\}. \tag{39}$$

We have the following:

Definition 3.22. *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 $0 \leq t \leq T$.

And as before, setting the coefficients 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.

3.6 The general definition of a differential equation

After the previous examples, let us precise the definition of a differential equation. The definition holds true for ODEs and PDEs equally. In order to allow for largest possible generality we first need to introduce some notation. Common is to use the multiindex notation as introduced in Section 2.5.

Definition 3.23 (Evans [11])**.** *Let $\Omega \subset \mathbb{R}^n$ be open. 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

$$F : \mathbb{R}^{n^k} \times \mathbb{R}^{n^{k-1}} \times \mathbb{R}^{n^2} \times \mathbb{R}^n \times \mathbb{R} \times \Omega \to \mathbb{R}.$$

Example 3.24. *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 $n = 1$ 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: $F(D^2 u) := -\Delta u = 0$ where $F : \mathbb{R}^4 \to \mathbb{R}$. That is $k = 2$ and lower derivatives of order 1 and 0 do not exist.*
3. *Heat equation: $F(D^2 u, Du) = \partial_t u - \Delta u = 0$ where $F : \mathbb{R}^6 \times \mathbb{R}^3 \to \mathbb{R}$. Here $k = 2$ is the same as before, but a lower-order derivative of order 1 in form of the time derivative does exist.*

4. *Wave equation:* $F(D^2u) = \partial_t^2 u - \Delta u = 0$ *where* $F : \mathbb{R}^6 \to \mathbb{R}$.

We finally, 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 3.25 (Evans[11])**.** *Differential equations are divided into linear and nonlinear 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**.*

Example 3.26. *We provide again some examples:*

1. *All differential equations from Example 3.24 are **linear**.*

2. *Euler equations (fluid dynamics, special case of Navier-Stokes with zero viscosity). Here,* $n = 2+1+1 = 4$ *(in two spatial dimensions). Let us consider the momentum part of the Euler equations:*

$$\partial_t v_f + v_f \cdot \nabla v_f + \nabla p_f = f(x).$$

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:*

$$\partial_t v_f - \rho_f \nu_f \Delta v_f + v_f \cdot \nabla v_f + \nabla p_f = f(x).$$

Here $k = 2$*. But the coefficients in front of the highest order term, do not depend on* v_f*. Consequently, the Navier-Stokes equations are semi-linear or 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:*

$$\partial_t v_f - \rho_f \nu_f (\Delta v_f)^2 + v_f \cdot \nabla v_f + \nabla p_f = f(x).$$

Example 3.27 (Development of numerical methods for nonlinear equations)**.** *In case you are given a nonlinear IBVP 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 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 abstract 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. The contrary works also: if you have implemented the full nonlinear PDE and recognize a 'problem', you can reduce the PDE term by term and make it step by step simpler.

Exercise 2. *We work in $\mathbb{R}^2$. Let $F = I + \nabla u$, where I is the identity matrix in $\mathbb{R}^2$ and u a given deformation.*

1. *Write F component-wise.*
2. *Compute the determinant $J := det(F)$.*

Let us consider now the following problem. Find $v : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^2$ such that:
$$\begin{aligned} -div(J\sigma_f F^T) &= f \quad \text{in } \Omega, \\ v &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$
where $\partial\Omega$ is the boundary of Ω and f a given volume force. Furthermore,
$$\sigma_f = \nu(\nabla v F^{-1}),$$
where ν is a material parameter.

1. *Compute F^{-1}.*
2. *Linearize the PDE-operator $-div(J\sigma_f F^T)$. Hint: Try to get rid of the transformations J and F. Under which assumptions are they small?*
3. *Write the final PDE $-div(A) = f$ component-wise. Here, A is now the linearized part from the task before.*

3.7 Boundary and initial conditions

As seen in the previous sections, all PDEs are complemented with boundary conditions and in the time-dependent case, 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.

3.7.1 Types of boundary conditions

In Section 3.5.3, we have mainly dealt with homogeneous Dirichlet conditions of the form $u = 0$ on the boundary. 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 $L := -\Delta u$.

Example 3.28. *Let us compute the heat distribution in PC 41. 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

$$\begin{aligned}
\partial_t T + (v \cdot \nabla)T - \nabla \cdot (K\nabla T) = f \quad & \text{in } \Omega \times I, \\
T = 18C \quad & \text{on } \partial_D\Omega \times I, \\
K\nabla T \cdot n = 0 \quad & \text{on } \partial_N\Omega \times I, \\
T(0) = 15C \quad & \text{in } \Omega \times \{0\}.
\end{aligned}$$

The homogeneous Neumann condition means that there is no heat exchange on the respective walls (thus neighboring rooms will have the same room temperature). The nonhomogeneous Dirichlet condition states that there is a given temperature of $18C$, 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.

3.7.2 Common traps and errors

We provide a list of traps and challenges:

- If your solution is 'wrong' (why-ever you might know this) then check of course the PDE but also the boundary conditions;
- Do you correctly apply the boundary conditions in your computer program to all boundary parts? Be careful, no explicit prescription means that implicitly Neumann conditions are applied (more details in Section 5);
- Often physical formulae assume that the solution has been computed on an infinite domain. However in a computer program we must cut the domain to a finite dimensional subset. Thus, boundary conditions will influence the result. The question is how much such 'artificial' boundary conditions will influence the findings?

3.7.3 Example: Dependence of the numerical solution on the boundaries

To illustrate the dependence of the numerical solution on the boundaries (or the domain size), let us consider an example from channel fluid flow as sketched in Figure 4

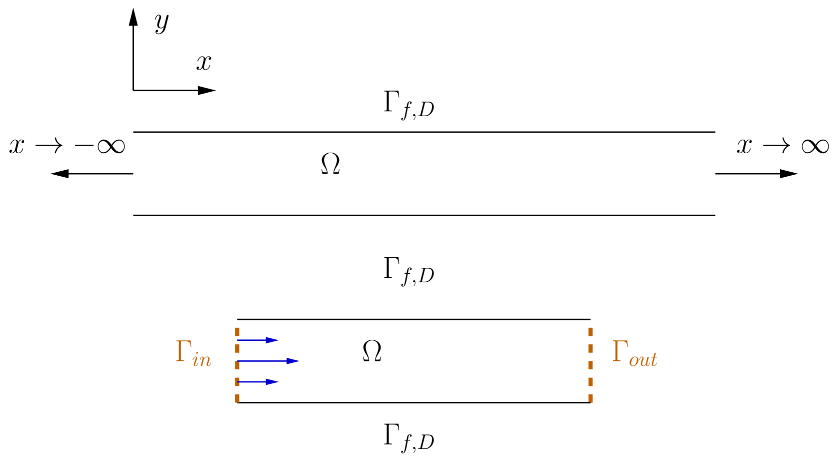

Figure 4: Truncation of an unbounded fluid domain and introducing artificial boundary conditions on Γ_{in} and Γ_{out}.

Moreover, we are interested in evaluating quantities of interest (or mathematically-speaking functionals of interest). In solid mechanics, these are deflections and deformations: what is the displacement of a bridge subject to weather (wind) or forces caused by trucks driving over it. In fluid mechanics, one is often interested in evaluating surface forces such as drag and lift. What have these comments to do with boundary conditions? As previously claimed, boundary conditions and the domain size can have significant influence on quantities of interest - even when these functional evaluations are far away from the boundary. Ideally, the domain should be infinitely large (see again Figure 4) such that boundary conditions do not play a role. However, this is in most case impossible to achieve due to computational cost.

Let us illustrate these considerations with an example: in channel flow with an elastic beam, we want to evaluate the tip-deflection of the beam and also drag and lift forces acting on this beam and the hole. The questions is where to we cut the channel in order to impose artificial boundary conditions. We run numerical simulations for four different channel lengths $5.0m, 2.5m, 1.5m$ and $1.0m$ leading to the results presented in Table 1 in which we clearly see that some of the functionals depend significantly on the length of the channel. For example the relative error of the drag evaluation is about 14%.

Table 1: Dependence of functional evaluations on the length (i.e., on the boundaries) of the channel.

Length	$u_x(A)[10^{-5}]$	$u_y(A)[10^{-4}]$	Drag	Lift
5.0	2.02	8.22	16.7	0.74
2.5	2.27	8.22	15.3	0.74
1.5	2.38	8.23	14.6	0.74
1.0	2.47	7.94	14.3	0.73

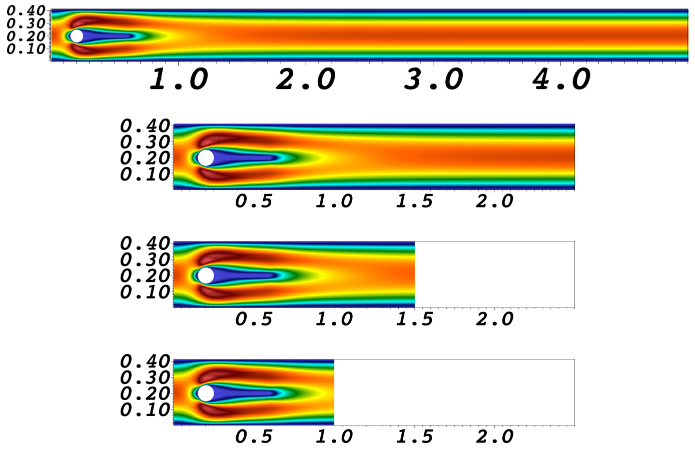

Figure 5: Fluid flow and structural deformations computed for different channel lengths to illustrate dependence of functional evaluations on boundary conditions.

Definition 3.29 (Drag and lift)**.** *Drag and lift are important technical quantities in fluid dynamics. The drag force acts in the opposite direction of the relative motion of the object that is subject to a fluid (or air). The lift force is perpendicular to the drag.*

3.8 Weak/variational solutions versus classical solutions

To solve explicitly a differential equation is only possible for specific special test cases. Thus, a classical solution is in most cases not possible to achieve. Moreover, they require in many cases too much regularity which is difficult to handle as well as often not met by practical applications (e.g., singularities in the domain as corners or slit domains). In order to study well-posedness and simultaneously provide a framework for numerical computations, we work with **weak (or variational)** solutions based on Galerkin techniques. In physics and mechanics this procedure is well-known as the **principle of virtual work**. Such approaches are introduced in Section 5. Thus, we lower our requirements on the smoothness of the solution. For instance error estimates of the form $\|u - u_h\|$ for elliptic problems require lower regularity when working with variational concepts (e.g., finite elements) than using finite differences.

3.9 The challenge of numerical modeling

To finalize this first chapter we provide a list of typical difficulties:

- PDEs are more difficult than ODEs.
- Nonlinear equations are more difficult than linear equations.
- Higher-order PDEs/ODEs are more difficult than low-order PDEs/ODEs.
- Systems are harder than single equations.
- The more coupling terms appear, the more difficult the solution process becomes.
- Interface coupling for two (or more) PDEs is harder than volume coupling.
- Multidomain problems are harder than single domain problems.
- For most PDEs/ODEs an explicit formula of a solution does not exist, and therefore numerical modeling becomes indispensable.
- Correct implementation and debugging code takes a lot of time.

3.9.1 Philosophy of numerical modeling

All the previous types of different equations we have seen in this chapter have different properties and these **properties on the continuous level should be maintained as well as possible after numerical discretization.** Thus it is extremely important to understand the properties of a differential equation (at least to some extend) on the continuous level (ODE/PDE theory) in order to be able to derive and analyze appropriate algorithmic schemes.

4 Finite differences for ODE initial-values problems

Finite differences have been very popular in the past and are still very important 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 (Section 5). 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. Multistep methods are treated elsewhere (e.g., [15]).

4.1 Problem statement of an IVP (initial value problem)

We consider ODE initial values problems of the form:

Formulation 4.1. *Find a differentiable function $y(t)$ for $0 \le t < T < \infty$ such that*

$$y'(t) = f(t, y(t)),$$
$$y(0) = y_0.$$

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

$$y' = \lambda y, \quad y(0) = y_0, \quad y_0, \lambda \in \mathbb{R}. \tag{40}$$

This ODE has the solution:

$$y(t) = \exp(\lambda t) y_0.$$

Furthermore, Problem (40) 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.

4.2 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. The latter situation will be discussed in Section 6.7.

Definition 4.2. *An IVP is called **stiff** (along a solution $y(t)$) if the eigenvalues $\lambda(t)$ of the Jacobian $f'(t, y(t))$ yield the stiffness ratio:*

$$\kappa(t) := \frac{\max_{Re\lambda(t)<0} |Re\lambda(t)|}{\min_{Re\lambda(t)<0} |Re\lambda(t)|} \gg 1.$$

In the above case for the model problem, the eigenvalue corresponds directly to the coefficient λ.

It is however not really true to classify any ODE with large coefficients $|\lambda| \gg 1$ always as a stiff problem. Here, the Lipschitz constant of the problem is already large and thus the discretization error asks for relatively small time steps. Rather stiff problems are characterized as ODE solutions that contain various components with (significant) different evolutions over time; that certainly appears in ODE systems in which we seek $y(t) \in \mathbb{R}^n$ rather than $y(t) \in \mathbb{R}$.

A very illustrative example can be found in [23] 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.

4.3 Well-posedness of solutions

In ODE lectures, we usually learn first the theorem of Peano, which ensures existence (but not uniqueness) of an ODE. In these notes, we consider directly the 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 $f(t,y)$:

Definition 4.3 (Lipschitz condition)**.** *The function $f(t,y)$ on D is said to be (uniformly) Lipschitz continuous if for $L(t) > 0$ it holds*

$$\|f(t,x_1) - f(t,x_2)\| \le L(t)\|x_1 - x_2\|, \quad (t,x_1),(t,x_2) \in D.$$

The function is said to be (locally) Lipschitz continuous if the previous statement holds on every bounded subset of D.

Theorem 4.4 (Picard-Lindelöf)**.** *Let $f : D \to \mathbb{R}^n$ be continuous and Lipschitz. Then there exists for each $(t_0, y_0) \in D$ a $\varepsilon > 0$ and a solution $y : I := [t_0 - \varepsilon, t_0 + \varepsilon] \to \mathbb{R}^n$ of the IVP such that*

$$y'(t) = f(t, y(t)), \quad t \in I, \quad y(t_0) = y_0.$$

Proof. See for example [15]. □

4.4 One-step schemes

4.4.1 The Euler method

The Euler method is the most simplest scheme. It is an explicit scheme and also known as forward Euler method.

We consider again the IVP from before and the right hand side satisfies again a Lipschitz condition. According to Theorem 4.4, there exists a unique solution for all $t \ge 0$.

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):

$$y'(t_{n-1}) \approx \frac{y_n - y_{n-1}}{k_n}$$

Thus: $y'(t_{n-1}) = f(t_{n-1}, y_{n-1}(t_{n-1}))$. Then the ODE can be approximated as:

$$\frac{y_n - y_{n-1}}{k_n} \approx f(t_{n-1}, y_{n-1}(t_{n-1}))$$

Thus we obtain the scheme:

Algorithm 4.5 (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*

$$y_n = y_{n-1} + k_n f(t_{n-1}, y_{n-1}), \quad n = 1, \dots N,$$

where $y_n := y(t_n)$.

Remark 4.6 (Notation)**.** *The chosen notation is not optimal in the sense that y_n denotes the discrete solution obtained by the numerical scheme and $y(t_n)$ the corresponding (unknown) exact solution. However, in the literature one often abbreviates $y_n := y(t_n)$, which would both denote the exact solution. One could add another index y_n^k (k for discretized solution with step size k) to explicitely distinguish the discrete and exact solutions. In these notes, we hope that the reader will not confuse the notation and we still use y_n for the discrete solution and $y(t_n)$ for the exact solution.*

4.4.2 Implicit schemes

With the same notation as introduced in Section 4.4.1, we define two further schemes. Beside the Euler method, low-order simple schemes are 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 4.5).

The derivation of the 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}):

$$y'(t_n) = \frac{y_n - y_{n-1}}{k_n}$$

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

$$\frac{y_n - y_{n-1}}{k_n} = f(t_n, y_n(t_n)).$$

Consequently, we obtain a scheme in which the right hand side is unknown itself, which leads to a formulation of a nonlinear system:

Algorithm 4.7 (Implicit (or backward) Euler)**.** *The implicit Euler scheme is a defined as:*

$$\begin{aligned} y_0 &:= y(0), \\ y_n - k_n f(t_n, y_n) &= y_{n-1}, \quad n = 1, \dots, N \end{aligned}$$

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 4.5.1.

To derive the trapezoidal rule, 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:

Algorithm 4.8 (Trapezoidal rule (Crank-Nicolson))**.** *The trapezoidal rule reads:*

$$\begin{aligned} y_0 &:= y(0), \\ y_n &= y_{n-1} + \frac{1}{2}k_n\Big(f(t_n, y_n) + f(t_{n-1}, y_{n-1}\Big), \quad n = 1, \dots, N. \end{aligned}$$

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 4.6.

4.5 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

$$\|y_n - y(t_n)\| \le Ck^\alpha$$

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. To derive error estimates we work with the model problem (40). 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:

$$e_n := y_n - y(t_n) \quad \text{for } 1 \le n \le N.$$

It holds:

$$\begin{aligned} e_n &= y_n - y(t_n), \\ &= B_E y_{n-1} - y(t_n), \\ &= B_E(e_{n-1} + y(t_{n-1})) - y(t_n), \\ &= B_E e_{n-1} + B_E y(t_{n-1}) - y(t_n), \\ &= B_E e_{n-1} + \frac{k(B_E y(t_{n-1}) - y(t_n))}{k}, \\ &= B_E e_{n-1} - k \underbrace{\frac{y(t_n) - B_E y(t_{n-1})}{k}}_{=:\eta_{n-1}}. \end{aligned}$$

The 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.

Therefore, the error can be split into two parts:

$$e_n := \underbrace{B_E e_{n-1}}_{Stability} - \underbrace{k\eta_{n-1}}_{Consistency} . \tag{41}$$

This error is obtained by plugging-in the exact (but unknown) solution in our numerical scheme. The remainder term is than η_{n-1} and is related to the consistency of the numerical scheme. The first term, namely the stability, provides an idea how the previous error is propagated from t_{n-1} to t_n. The second term, the consistency, yields the speed of convergence and is the previously introduced truncation error. In fact, for the forward Euler scheme we have more precisely in (41):

$$e_n := \underbrace{B_E e_{n-1}}_{Stability} - \underbrace{k\left[\frac{y(t^n) - B_E y(t^{n-1})}{k}\right]}_{Consistency},$$

which yields furthermore:

$$\begin{aligned} \eta_{n-1} &= \frac{y(t_n) - B_E y(t_{n-1})}{k} = \frac{y(t_n) - (1+k\lambda)y(t_{n-1})}{k} && (42) \\ &= \frac{y(t_n) - y(t_{n-1})}{k} - \lambda y(t_{n-1}) && (43) \\ &= \frac{y(t_n) - y(t_{n-1})}{k} - y'(t_{n-1}). && (44) \end{aligned}$$

We investigate these terms further in Section 4.5.2 and concentrate first on the stability estimates in the very next section.

4.5.1 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 7.

We recapitulate (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 $Re\,\lambda$:

$$\begin{aligned} Re\,\lambda < 0 &\quad \Rightarrow |y(t)| = |y_0| \exp(Re\,\lambda) \to 0, \\ Re\,\lambda = 0 &\quad \Rightarrow |y(t)| = |y_0| \exp(Re\,\lambda) = |y_0|, \\ Re\,\lambda > 0 &\quad \Rightarrow |y(t)| = |y_0| \exp(Re\,\lambda) \to \infty. \end{aligned}$$

For a *good* numerical scheme, the first case is particularly interesting whether such a scheme can produce a bounded discrete solution when the continuous solution has this property.

Definition 4.9 ((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 $R(z)$ where $z = \lambda k$. The region of absolute stability is defined as:*

$$SR = \{z = \lambda k \in \mathbb{C} : |R(z)| \leq 1\}.$$

Remark 4.10. *Recall that $R(z) := B_E$.*

The stability functions to explicit, implicit Euler and trapezoidal rule are given by:

Proposition 4.11. *For the simplest time-stepping schemes forward Euler, backward Euler and the trapezoidal rule, the stability functions $R(z)$ read:*

$$\begin{aligned} R(z) &= 1 + z, \\ R(z) &= \frac{1}{1-z}, \\ R(z) &= \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}. \end{aligned}$$

Proof. We take again the model problem $y' = \lambda y$. Let us discretize this problem with the forward Euler method:

$$\begin{aligned} \frac{y_n - y_{n-1}}{k} &= \lambda y_{n-1} && (45) \\ \Rightarrow y_n &= (y_{n-1} + \lambda k) y_{n-1} && (46) \\ &= (1 + \lambda k) y_{n-1} && (47) \\ &= (1 + z) y_{n-1} && (48) \\ &= R(z) y_{n-1}. && (49) \end{aligned}$$

For the implicit Euler method we obtain:

$$\begin{aligned} \frac{y_n - y_{n-1}}{k} &= \lambda y_n && (50) \\ \Rightarrow y_n &= (y_{n-1} + \lambda k) y_n && (51) \\ \Rightarrow y_n &= \frac{1}{1 - ak} y_n && (52) \\ \Rightarrow y_n &= \underbrace{\frac{1}{1-z}}_{=:R(z)} y_n. && (53) \end{aligned}$$

The procedure for the trapezoidal rule is again the analogous. □

Definition 4.12 (A-stability)**.** *A difference method is A-stable if its stability region is part of the absolute stability region:*

$$\{z \in \mathbb{C} : Re\, z \leq 0\} \subset SR,$$

here Re denotes the real part of the complex number z. A brief introduction to complex numbers can be found in any calculus lecture dealing with those or also in the book [21].

In other words:

Definition 4.13 (A-stability)**.** *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 (or even contractive) 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 4.14. *A-stability is attractive since in particular for stiff problems we can compute with arbitrary step sizes k and do not need any step size restriction.*

Proposition 4.15. *The explicit Euler scheme cannot be A-stable.*

Proof. For the forward Euler scheme, it is $R(z) = 1 + z$. For $|z| \to \infty$ is holds $R(z) \to \infty$ which is a violation of the definition of A-stability. □

Remark 4.16. *More generally, explicit schemes can never be A-stable.*

Example 4.17. *We illustrate the previous statements.*

1. *In Proposition 4.11 we have seen that for the forward Euler method it holds:*

   $$y_n = R(z) y_{n-1},$$

   where $R(z) = 1 + z$. Thus, according to Definition 4.12 and 4.13, we obtain convergence when the sequence $\{y_n\}$ is contracting:

   $$|R(z)| \leq |1 + z| \leq 1. \tag{54}$$

   Thus if the 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 respect to decreasing parts of functions! Thus, the continuous solution is bounded 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 (54) *is fulfilled.*

2. *For the implicit Euler scheme, we see that a large λ and large k even both help to stabilize the iteration scheme (but be careful, the implicit Euler scheme, stabilizes actually 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 stiff problems with large parameters/coefficients λ.*

Remark 4.18. *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.*

4.5.2 Consistency / local discretization error - convergence order

We address now the second 'error source' in (41). 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 all Taylor terms of combined to the discrete scheme. The lowest order remainder term determines finally the local consistency of the scheme.

We briefly formally recall Taylor expansion. For a function $f(x)$ we develop at a point $a \neq x$ the Taylor series:

$$T(f(x)) = \sum_{j=0}^{\infty} \frac{f^{(j)}(a)}{j!}(x-a)^j.$$

Let us continue with the forward Euler scheme and the truncation error (44). Using the forward Euler scheme, we need information about the solution at the old time step t^{n-1}. Thus we need to 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 (44). 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

$$\|\eta_{n-1}\| \leq \max_{t\in[0,T]} \frac{1}{2}\|y''(t)\|k = O(k)$$

Therefore, the convergence order is k (namely linear convergence speed).

4.5.3 Convergence

With the help of the two previous subsections, we can easily show the following error estimates:

Theorem 4.19 (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 $k := \max_n k_n$.

Proof. 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 4.6 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 4.20 (Convergence of trapezoidal rule)**.** *We have*

$$\max_{t\in I} |y_n(t) - y(t)| \le c(T,y)k^2 = O(k^2),$$

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., $O(k^2)$.

Let us justify the convergence order for the forward Euler scheme in more detail now.

Theorem 4.21. *Let $I := [0,T]$ the time interval and $f : I \times \mathbb{R}^d \to \mathbb{R}^d$ continuously differentiable and globally Lipschitz-continuous with respect to y:*

$$\|f(t,y) - f(t,z)\|_2 \le L\|y - z\|_2$$

for all $t \in I$ and $y, z \in \mathbb{R}^d$. Let y be the solution to

$$y' = f(t,y), \quad y(0) = y_0.$$

Furthermore let $y_n, n = 1, \dots, n$ the approximations obtained with the Euler scheme at the nodal points $t_n \in I$. Then it holds

$$\|y(t_n) - y_n\|_2 \le \frac{(1+Lk)^n - 1}{2L}\|y''\|\, k \le \frac{e^{LT} - 1}{2L}\|y''\|\, k = c(T,y)k = O(k),$$

for $n = 0, \dots, N$.

Proof. The proof follows [15], but consists in working out the steps shown at the beginning of Section 4.5, Section 4.5.1, and Section 4.5.2. ☐

4.6 Detailed numerical tests

In this section we demonstrate our algorithmic developments in terms of a numerical example. The programming code is written in octave (which is the open-source sister of MATLAB) presented in Section 11.1.

4.6.1 Problem statement and discussion of results

Example 4.22. *Solve our ODE model problem numerically. Let $a = g - m$ be $a = 0.25$ or $a = -0.25$ or $a = -10$ (three test scenarios). The IVP is given by:*

$$y' = ay, \quad y(2011) = 2.$$

Use the forward Euler (FE), backward Euler (BE), and trapezoidal rule (CN) for the numerical approximation. Please observe the accuracy in terms of the discretization error and for (stiff) equations with a large negative coefficient $a = -10 \ll 1$ the behavior of the three schemes.

In order to calculate the convergence order α from numerical results, we make the following derivation. Let $P(k) \to P$ for $k \to 0$ 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 4.23 (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:*

$$\alpha = \frac{1}{log(2)} log\Big(\Big|\frac{P(k) - P(k/2)}{P(k/2) - P(k/4)}\Big|\Big). \tag{55}$$

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:

$$P(k) - P(k/2) = O(k^\alpha),$$
$$P(k/2) - P(k/4) = O((k/2)^\alpha).$$

First, we have

$$P(k/2) - P(k/4) = O((k/2)^\alpha) = \frac{1}{2^\alpha}O(k^\alpha)$$

We simply re-arrange:

$$\begin{aligned} P(k/2) - P(k/4) &= \frac{1}{2^\alpha}\Big(P(k) - P(k/2)\Big) \\ \Rightarrow \quad 2^\alpha &= \frac{P(k) - P(k/2)}{P(k/2) - P(k/4)} \\ \Rightarrow \quad \alpha &= \frac{1}{log(\alpha)}\,\frac{P(k) - P(k/2)}{P(k/2) - P(k/4)} \end{aligned}$$

□

In the following we present our results for the (absolute) error for test case 1 ($a = 0.25$) on such three mesh levels:

```
Scheme          #steps  k     Absolute 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
```

- The absolute error at the end time T in the forth column is computed as

  $$e_N = |y(T) - y_N|,$$

  where $y(T)$ is the exact solution and y_N the numerical approximation at the end time value at the final numerical step N.

- 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 the column after, the errors are provided.

- In order to compute numerically the convergence order α with the help of formula (55), we work with $k = k_{max} = 0.36$. Then we identify in the above table that $P(k_{max}) = P(0.36) = |y(T) - y_8|, P(k_{max}/2) = P(0.18) = |y(T) - y_{16}|$ and $P(k_{max}/4) = P(0.09) = |y(T) - y_{32}|$.

- 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 (55) yielding $\alpha = 0.91804$. The trapezoidal rule 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 $\alpha = 1.9967$.
- 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.

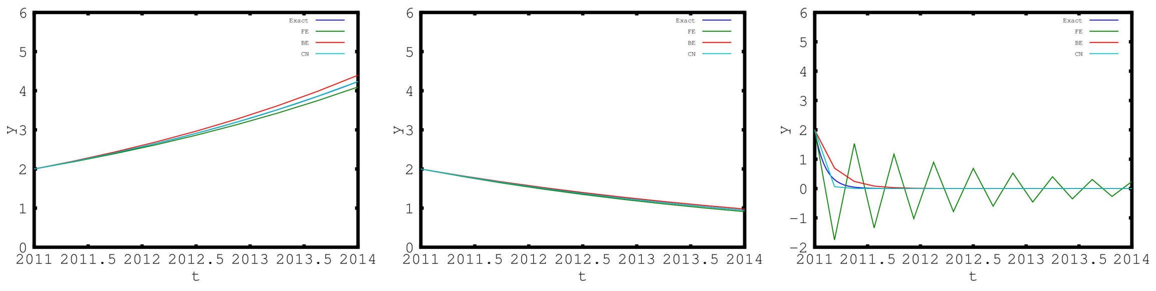

Figure 6: Example 4.22: on the left, the solution to test 1 is shown. In the middle, test 2 is plotted. On the right, the solution of test 3 with $n = 16$ (number of intervals) is shown. Here, $n = 16$ corresponds to a step size $k = 0.18$ which is slightly below the critical step size for convergence (see Section 4.6.2). Thus we observe the instable behavior of the forward Euler method, but also see slow convergence towards the continuous solution.

4.6.2 Treatment of the instability of the forward Euler method

With the help of Example 4.17 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 $k = 0.18$; from before;
- Test 3d: $n = 20$, yielding $k = 0.15$.

The results of test 3a,3b,3d are provided in Figure 7 and visualize very nicely the theoretically predicted behavior.

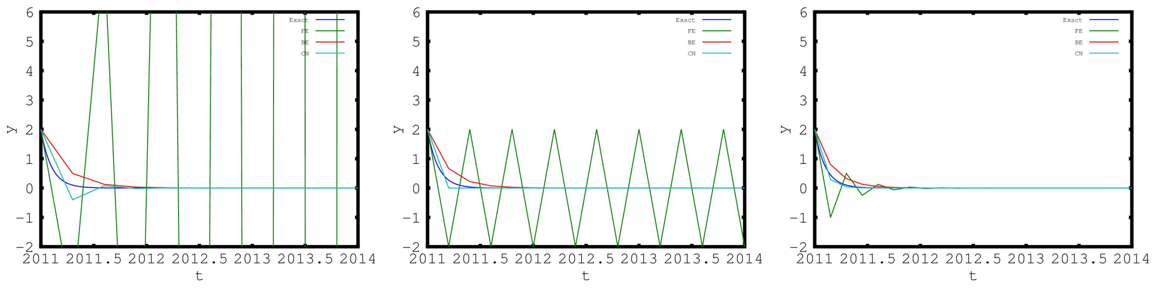

Figure 7: Example 4.22: tests 3a,3b,3d: Blow-up, constant zig-zag non-convergence, and convergence of the forward Euler method.

Exercise 3. *Consider the ODE-IVP:*

$$y'(t) = ay(t), \quad y(t_0) = 7,$$

where $t_0 = 5$ and $T = 11$.

1. *Implement the backward Euler scheme. and set the model parameter to $a = 2$*
2. *Run a second test with $a = -2$.*
3. *Implement the forward Euler scheme. What is the critical step size (Hint: Determine the stability interval).*
4. *Implement the Crank-Nicolson scheme.*
5. *Perform a computational analysis and detect the convergence order.*

4.7 Runge-Kutta methods

A drawback of the Euler methods and to some extend of the trapezoidal rule as well is the low order and slow convergence. This has been justified both theoretically and in terms of the previous example.

In this section, we briefly motivate the idea how to obtain higher-order methods. Our model problem is as before:

$$\begin{aligned} y'(t) &= f(t,y), \quad t \in I, \\ y(t_0) &= y_0. \end{aligned}$$

The explicit Euler methods reads:

$$y^{n+1} = y^n + kf(t^n, y^n), \quad k = t^{n+1} - t^n.$$

The basic idea to obtain higher-order methods is pretty simple: replace $f(\cdot)$ by a general function $F(\cdot)$:

$$y^{n+1} = y^n + kF(k, t^n, y^n, y^{n+1}), \quad k = t^{n+1} - t^n.$$

The question is how $F(\cdot)$ does look like? Well, we use again Taylor and assume that $f \in C^\infty$:

$$y(t^{n+1}) = y(t^n) + y'(t^n)k + \frac{1}{2}y''(t^n)k^2 + \cdots = \sum_{j=1}^{s} \frac{1}{j!} y^{(j)}(t^n)k^j.$$

We now use the ODE:

$$y' = f(t,y) \quad \to y'' = f'(t,y) \quad \to y''' = f''(t,y), \ldots$$

Then the Taylor expansion can be written as:

$$y(t^{n+1}) = y(t^n) + k \underbrace{\sum_{j=1}^{s} \frac{1}{j!} y^{(j-1)}(t^n) k^{j-1}}_{=:F(k,t^n,y^n,y^{n+1})}.$$

In principle we would be finished now. There is however a very practical difficulty; namely the higher-order derivatives of $f(t, y)$. For complicated functions they become tedious to be evaluated - especially if we think to use a finite difference scheme as time stepping scheme for a PDE (see Section 6). Thus we want to get rid of the $f(t, y)$ derivatives. The idea is to replace $f^{(j-1)}$ be difference quotients. This will lead to a recursive iteration. The ansatz is:

$$F(k, t^n, y^n, y^{n+1}) := \sum_{j=1}^{s} b_j k_j$$

with

$$k_j := f(t^n + c_j k, \eta_j), \quad \sum_{j=1}^{s} b_j = 1.$$

The trick is to determine b_j, c_j and η_j, where η_j can be further derived as (using the previously mentioned recursion and difference quotients):

$$\eta_j = y^n + k \sum_{\nu=1}^{s} a_{j\nu} \underbrace{f(t^n + c_\nu k, \eta_\nu)}_{=k_\nu}, \quad \sum_{\nu=1}^{s} a_{j\nu} = c_j, \quad 1, \ldots, s.$$

Let us pause for a moment and denote the different coefficients we have introduced so far:

- η_j: stages
- s: stage number
- c_j: nodes
- b_j: weights
- $a_{j\nu}$ are the entries of the so-called Runge-Kutta matrix.

Remark 4.24 (Explicit Runge-Kutta schemes)**.** *If $a_{j\nu} = 0$ for all $\nu \geq j$, we obtain explicite Runge-Kutta schemes; otherwise the scheme will be implicit.*

The previous coefficients will be determined in such a way that we obtain (hopefully) the optimal order $m = s$:

$$\sum_{j=1}^{s} b_j k_j = \sum_{j=1}^{s} \frac{k^{j-1}}{j!} f^{(j-1)}(t^n) + O(k^m).$$

To determine the consistency order we proceed as for the Euler schemes and plug the exact solution into the numerical scheme. The local truncation error is then given as:

$$\tau^n := \frac{y(t^{n+1}) - y(t^n)}{k} - F(k, t^n, y(t^n), y(t^{n+1})).$$

Example 4.25. *Let us illustrate the previous developments for two cases (the reader can find many more in the cited literature):*

1. *$s = 1$ (explicit Euler): $c_1 = a_{11} = 0$ and $b_1 = 1$ yielding $k_1 = f(t^n, y^n)$.*

2. *$s = 4$ (classical fourth-order Runge-Kutta):*

$$y^{n+1} = y^n + \frac{1}{6}k[k_1 + k_2 + k_3 + k_4].$$

Here we four stages ($s = 4$) and therefore the consistency order $m = 4$. Furthermore:

$$\begin{aligned} k_1 &= f(t^n, y^n), \\ k_2 &= f(t^{n+0.5}, y^n + \frac{1}{2}kk_1), \\ k_3 &= f(t^{n+0.5}, y^n + \frac{1}{2}kk_2), \\ k_4 &= f(t^{n+1}, y^n + kk_3). \end{aligned}$$

One can also derive a tableau of the coefficients as follows:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

Example 4.26. *Using the previous tableau we can express some schemes as follows:*

1. *Explicit Euler:*

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

2. *Implicit Euler:*

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

3. *Runge method:*

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline 0 & 0 & 1 \end{array}$$

4. *Runge-Kutta fourth order (RK4):*

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Exercise 4. *Implement the Runge-Kutta method of order $s = 4$ for the above problem given in Example 4.22. Afterwards, carry out the convergence analysis and verify whether the expected order is correct.*

5 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].

5.1 Problem statement

We introduce the concepts of the finite element method (FEM) for a 1-dimensional (1D) model problem. The key interest from the mathematical point of view is a very rich foundation of well-posedness, convergence analysis and error estimates for finite elements. We however to not tackle these aspects in this lecture and refer to the literature [5, 8, 14, 18]. Rather we introduce its practical purpose in solving problems. This is justified to some extend since finally the FEM was introduced in the 50' by engineers in aeroelasticity working for Boing. The underlying mechanism is very old and is well-known in physics and mechanics and the so-called **principle of virtual work.** An example of a 'recent' reference is [7].

We consider the model problem:

$$-u'' = f \quad \text{in } (0,1), \tag{56}$$

$$u(0) = u(1) = 0. \tag{57}$$

Please make yourself clear that this is nothing else than expressing Formulation 3.13 in 1D. Thus, we deal with a second-order elliptic problem.

In 1D, we can derive explicitly the solution of (56). 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:

$$u'(x) = -x + 1 \quad \Rightarrow \quad -u''(x) = 1.$$

Exercise 5. *Let $f = -1$.*

1. *Compute C_1;*
2. *Compute $u(x)$;*
3. *Check that $u(x)$ satisfies the boundary conditions;*
4. *Check that $u(x)$ satisfies the PDE.*

5.2 Finite elements in 1D

In the following we want to concentrate how to compute a solution with a numerical algorithm implemented in a code running on a computer. 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;
- Derive a variational (or weak) form of the given problem. This weak form is represented in terms of integrals;
- 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.

5.2.1 The mesh

Let us start with the mesh. We introduce nodal points (very similar to the previous chapter ODEs with 'time points'). Thus we divide $\Omega = (0, 1)$ into

$$x_0 = 0 < x_1 < x_2 < \dots < x_n < x_{n+1} = 1$$

yielding the **mesh elements**

$$K_j := [x_j, x_{j+1}].$$

Thus we create a uniform mesh since 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 5.1. *An important research topic is to organize the points x_j in certain non-uniform ways in order to reduce certain error measures. Thus, uniform meshes are the most easiest but in many applications they are not appropriate for a given problem.*

5.2.2 Linear finite elements

In the following we denote by P_k the space that contains all polynomials up to order k with coefficients in $\mathbb{R}$:

$$P_k := \{\sum_{i=1}^{k} a_i x^i |\; a_i \in \mathbb{R}\}.$$

In particular we will work with the space

$$P_1 := \{a_0 + a_1 x |\; a_0, a_1 \in \mathbb{R}\}.$$

A finite element is now a function localized to an element K_j. The ensemble of all such polynomials should be globally continuous.

Remark 5.2. *The global continuity can be weakened and is again another important research topic.*

We then define the space:

$$V_h := \{v \in C[0,1] |\; v|_{K_i} \in P_1, K_i := [x_i, x_{i+1}], 0 \le i \le n, v(0) = v(1) = 0\}.$$

Here $K_i \in \mathcal{T}_h$ are the so-called elements (or mesh cells). 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 can be represented by so-called hat functions. For $j = 1, \dots, n$ we define:

$$\phi_j(x) = \begin{cases} 0 & \text{if } \; x \notin [x_{j-1}, x_j] \\ \frac{x - x_{j-1}}{x_j - x_{j-1}} & \text{if } \; x \in [x_{j-1}, x_j] \\ \frac{x_{j+1} - x}{x_{j+1} - x_j} & \text{if } \; x \in [x_j, x_{j+1}] \end{cases}$$

with the property

$$\phi_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

For a uniform step size $h = x_j - x_{j-1} = x_{j+1} - x_j$ we obtain

$$\phi_j(x) = \begin{cases} 0 & \text{if } \; x \notin [x_{j-1}, x_j] \\ \frac{x - x_{j-1}}{h} & \text{if } \; x \in [x_{j-1}, x_j] \\ \frac{x_{j+1} - x}{h} & \text{if } \; x \in [x_j, x_{j+1}] \end{cases}$$

and for its derivative:

$$\phi_j'(x) = \begin{cases} 0 & \text{if } \; x \notin [x_{j-1}, x_j] \\ +\frac{1}{h} & \text{if } \; x \in [x_{j-1}, x_j] \\ -\frac{1}{h} & \text{if } \; x \in [x_j, x_{j+1}] \end{cases}$$

Lemma 5.3. *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. $\square$

Thus 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.

Remark 5.4. *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).*

5.2.3 The weak form

We have set-up a mesh and local polynomial functions with a unique representation, all this developments result in a 'finite element'. Now, we derive a variational form (or also called weak form).

The key idea is simple: multiply the strong form with a test function, integrate the resulting form, and 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.

Let us look into this procedure in more detail now:

$$-u'' = f \tag{58}$$

$$\Rightarrow -\int_\Omega u''\phi\,dx = \int_\Omega f\phi\,dx \tag{59}$$

$$\Rightarrow \int_\Omega u'\phi'\,dx - \int_{\partial\Omega} \partial_n u\phi\,ds = \int_\Omega f\phi\,dx \tag{60}$$

$$\Rightarrow \int_\Omega u'\phi'\,dx = \int_\Omega f\phi\,dx \tag{61}$$

Remark 5.5. *Please make yourself two things clear:*

- *How does $\int_{\partial\Omega} \partial_n u\phi\,ds$ look in 1D?*
- *Why does this term vanish?*
- *Where do we find the original boundary conditions $u(0) = u(1) = 0$?*

To summarize we have: Find $u \in V$ such that

$$\int_\Omega u'\phi'\,dx = \int_\Omega f\phi\,dx \quad \forall \phi \in V. \tag{62}$$

Definition 5.6 (Notation)**.** *A common short-hand notation in mathematics is to use parentheses for L^2 scalar products:*

$$\int_\Omega ab\,dx =: (a, b),$$

such that we can write

$$(u', \phi') = (f, \phi). \tag{63}$$

We have been a bit sloppy and add now the discretization parameter h and formulate the mathematically correct statement now:

Formulation 5.7. *Find $u_h \in V_h$ such that*

$$(u_h', \phi_h') = (f, \phi_h) \quad \forall \phi_h \in V_h. \tag{64}$$

More generally in applied mathematics, we assign a bilinear form $a(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and a linear form $l(\cdot) : \mathbb{R}^n \to \mathbb{R}$ such that

$$a(u_h, \phi_h) = l(\phi_h), \tag{65}$$

where

$$a(u_h, \phi_h) = (u_h', \phi_h'), \quad l(\phi_h) = (f, \phi_h).$$

In order to proceed we can express $u_h \in V_h$ with the help of the basis functions ϕ_j in $V_h := \{\phi_1, \dots, \phi_n\}$, thus:

$$u_h = \sum_{j=1}^{n} u_j \phi_j(x), \quad u_j \in \mathbb{R}.$$

Since (64) holds for all $\phi_i \in V_h$ for $1 \le i \le n$, it holds in particular for each i:

$$(u_h', \phi_i') = (f, \phi_i) \quad \text{for } 1 \le i \le n \tag{66}$$

We now insert the representation for u_h in (66):

$$\sum_{j=1}^{n} u_j (\phi_j', \phi_i') = (f, \phi_i) \quad \text{for } 1 \le i \le n \tag{67}$$

This yields a linear equation equation system of the form

$$AU = B$$

where

$$U = (u_j)_{1 \le j \le n} \in \mathbb{R}^n, \tag{68}$$
$$B = \big((f, \phi_i)\big)_{1 \le i \le n} \in \mathbb{R}^n, \tag{69}$$
$$A = \Big((\phi_j', \phi_i')\Big)_{1 \le j, i \le n} \in \mathbb{R}^{n \times n}. \tag{70}$$

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 $u(x_{n+1}) = u(1) = 0$.

5.2.4 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 $A = (a_{ij})_{1 \le i,j \le n}$. For the diagonal elements we calculate:

$$a_{ii} = \int_\Omega \varphi_i'(x)\varphi_i'(x)\,dx = \int_{x_{i-1}}^{x_{i+1}} \varphi_i'(x)\varphi_i'(x)\,dx \tag{71}$$
$$= \int_{x_{i-1}}^{x_i} \frac{1}{h^2}\,dx + \int_{x_i}^{x_{i+1}} \Big(-\frac{1}{h}\Big)^2\,dx \tag{72}$$
$$= \frac{h}{h^2} + \frac{h}{h^2} \tag{73}$$
$$= \frac{2}{h}. \tag{74}$$

For the right off-diagonal we have:

$$a_{i,i+1} = \int_\Omega \varphi_{i+1}'(x)\varphi_i'(x)\,dx = \int_{x_i}^{x_{i+1}} \frac{1}{h} \cdot \Big(-\frac{1}{h}\Big)\,dx = -\frac{1}{h}. \tag{75}$$

It is trivial to see that $a_{i,i+1} = a_{i-1,i}$.

In compact form we summarize:

$$a_{ij} = \int_\Omega \phi_j'(x)\phi_i'(x)\,dx = \begin{cases} -\frac{1}{x_{j+1}-x_j} & \text{if } j = i-1 \\ \frac{1}{x_j - x_{j-1}} + \frac{1}{x_{j+1}-x_j} & \text{if } j = i \\ -\frac{1}{x_j-x_{j-1}} & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases}. \tag{76}$$

For a uniform mesh (as we assume in this section) we can simplify the previous calculation since we know that $h = h_j = x_{j+1} - x_j$:

$$a_{ij} = \int_\Omega \phi_j'(x)\phi_i'(x)\,dx = \begin{cases} -\frac{1}{h} & \text{if } j = i-1 \\ \frac{2}{h} & \text{if } j = i \\ -\frac{1}{h} & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases}. \tag{77}$$

The resulting matrix A for the 'inner' points $x_0, \ldots, x_n$ 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 5.8. *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:

$$b_i = \int_\Omega f(x)\phi_i(x)\,dx \quad \forall 1 \le i \le n.$$

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:

$$b_i = \int_\Omega 1 \cdot \phi_i(x)\,dx = 1 \cdot \int_\Omega \phi_i(x)\,dx = 1 \cdot h \quad \forall 1 \le i \le n.$$

Namely

$$B = (h, \ldots, h)^T.$$

In this section, we could evaluate the integrals exactly by making several assumptions and simplifications such as

- Linear trial and test functions;
- Uniform mesh sizes h;
- Constant parameters (no jumps etc.).

Thus in a programming code one does even not need to evaluate the integrals in some automated way but only creates the matrix with the given entries directly. However, in a general finite element program, the integrals are evaluated in terms of numerical quadrature, which we explain in the following.

5.2.5 Numerical quadrature to evaluate more complicated integrals

A numerical quadrature rule is defined as[1]:

Definition 5.9 (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 $x_0, \dots, x_n$ and $n+1$ quadrature weights $\alpha_0, \dots, \alpha_n$.

The simplest quadrature rules are:

Definition 5.10 (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 5.11 (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 5.12 (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 $f(x)$:*

$$I^1(f) = \frac{b-a}{2}(f(a) + f(b)).$$

Remark 5.13. *For more details on quadrature rules we refer to further literature such as for example [21]. 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 5.14. *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.*

[1]The definitions and notation of the quadrature rules are copied and translated from [21].

Exercise 6. *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 7. *Evaluate the entries of the stiffness matrix A using the trapezoidal rule (be careful, the task is nearly trivial).*

Exercise 8. *Let $\Omega = (0,1)$. Implement a finite element scheme for solving*

$$-\nabla \cdot (\nabla u) = 1 \quad \text{in } \Omega, \tag{78}$$
$$u = 0 \quad \text{on } \partial\Omega \tag{79}$$

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 9. *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 10. *Let $\Omega = (0,1)^2$. Implement a finite element scheme for solving*

$$-\nabla \cdot (\nabla u) = 1 \quad \text{in } \Omega, \tag{80}$$
$$u = 0 \quad \text{on } \partial\Omega \tag{81}$$

in octave or C++ or python.

5.2.6 Solving the linear system: Overview

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 5.15. *The wishes No. 2 and 3 are again fulfilled by using finite elements rather than other basis functions.*

Remark 5.16. *As other remarks before, the numerical solution of coupled, nonlinear, partial differential equations, which are* ***robust*** *and* ***efficient*** *is an active research topic.*

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 octave
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.

5.2.7 Solving the linear system: Iterative solvers

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

$$f(x) = x$$

We provide in the following a brief introduction that is based on [21]. First, we have

Definition 5.17. *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 $k = 1, 2, \dots$:*

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

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 5.18. *Thanks to Banach's fixed point theorem (see again [21]), we can investigate under which conditions the above scheme will converge. Recall our discussions about the stability of the schemes for solving ODEs in which we found that $|B| < 1$ should hold. Please also recall the corresponding numerical tests of Section 11.1 (blow-up, zig-zag solution and converged solution). Here the situation is very similar and we must ensure that*

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

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

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

For this reason, one often works with the spectral norm spr(B). More details can be found for instance in [21].

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 5.19 (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 important schemes:

Definition 5.20 (Jacobi method)**.** *To solve $Ax = b$ with $A = L + D + R$ let $x^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \dots$*

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

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

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

Definition 5.21 (Gauß-Seidel method)**.** *To solve $Ax = b$ with $A = L+D+R$ let $x^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \dots$*

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

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

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

To implement these two schemes, we provide the presentation in index-notation:

Theorem 5.22 (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:*

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{k-1}\right), \quad i = 1, \dots, n,$$

i.e., (for the Gauss-Seidel method):

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^k - \sum_{j>i} a_{ij}x_j^{k-1}\right), \quad i = 1, \dots, n.$$

An alternative class of methods is based on so-called **descent** or **gradient** methods, which further improve the previously introduced methods. So far, we have:

$$x^{k+1} = x^k + d^k, \quad k = 1, 2, 3, \dots$$

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:

- Introducing 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 matrices as they appear in the discretization of elliptic PDEs studied previously in this section. 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:

$$f(x^0) > f(x^1) > \dots > f(x^k)$$

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 to solving linear equation systems. We first define the quadratic form

$$Q(y) = \frac{1}{2}(Ay, y)_2 - (b, y)_2,$$

where $(\cdot, \cdot)$ is the Euclidian scalar product. Then, we can define

Algorithm 5.23 (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 $k = 0, 1, 2, \dots$*

- *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 5.24 (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 $k = 0, 1, 2, \dots$*

- *Compute the vector $r^k = Ad^k$;*
- *Compute the relaxation*
$$\omega^k = \frac{\|d_k\|_2^2}{(r^k, d^k)_2}$$

- *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 [21]).*

Exercise 11. *Implement the descent method, Jacobi and Gauss-Seidel approaches for solving*

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ -3 \\ 4 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}.$$

As initial guess take $x^0 = 0$ *and compute for each method* 10 *steps.*

Remark 5.25. *Other important methods that we have not studied in these lecture notes are the conjugate gradient method (CG), preconditioned methods, Krylov subspace methods (such as CG, GMRES - generalized minimal residual method) and multigrid methods.*

5.3 Example

In this section, we substantiate the previous developments in terms of a numerical example including programming code presented in Section 11.2.

5.3.1 Problem statement

Solve:

$$-\frac{d}{dx}(\alpha u'(x)) = f \quad \text{in } \Omega, \tag{82}$$

$$u(0) = u(1) = a \tag{83}$$

with a positive material parameter α and right hand side f. The domain of interest is $\Omega := (x_0, x_N)$ and the boundary conditions are given in (83).

In case that α is a constant that does not depend on $x \in \Omega$, the above problem can be written as

$$-\alpha u''(x) = f.$$

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.

5.3.2 Choice of domain, boundary and material parameters

We now specify all ‘free’ parameters. For the domain we choose $x_0 = 0$ and $x_N = 1$ resulting in $\Omega = (0,1)$. As boundary condition we choose $a = 0$. For the material parameter we take $\alpha = 1$ and for the right hand $f = -1$. Furthermore we need to discretize Ω by creating a mesh consisting of a finite number of elements.

5.3.3 The goal of our computation

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.

5.3.4 Numerical results

As numerical results we obtain the solution curve displayed in Figure 5.3.4. 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):

$$\|u - u_h\|_{L^2} \to 0? \quad \text{for } h \to 0, \|u - u_h\|_{H^1} \to 0? \quad \text{for } h \to 0. \tag{84}$$

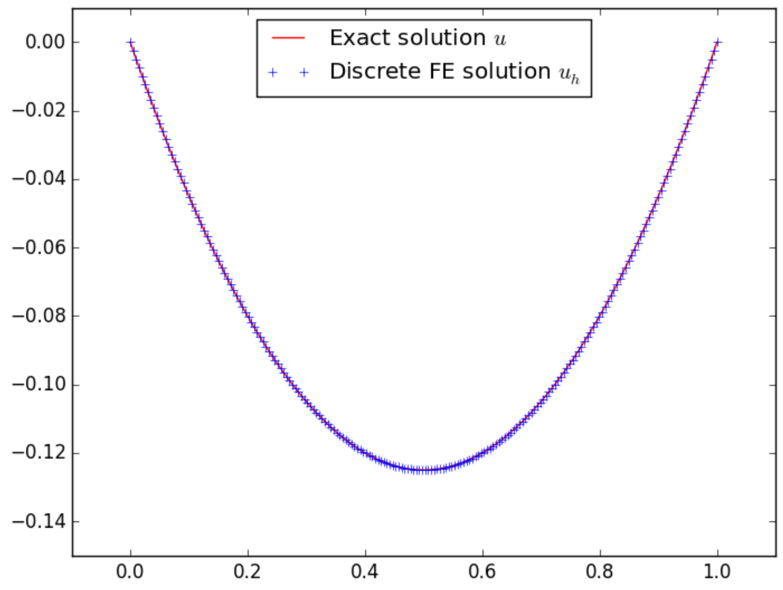

Figure 8: Visual comparison between the exact and the manufactured solution. Please do not forget: the 'picture norm' (thus comparing two graphs as we do here) is not a rigorous mathematical justification that a result is correct. It is rather a strong indication about the solution behavior and very 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 4.6.

6 Combining FD and FE within the Rothe method: Numerical discretization of the heat equation

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 4). In the second step, the problem is discretized in space with the finite element method developed in Chapter 5.

6.1 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 [10, 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.

6.2 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 6.1. *Find $u := u(x, t) : \Omega \times I \to \mathbb{R}$ such that*

$$\rho\partial_t u - \nabla \cdot (\alpha \nabla u) = f \quad \text{in } \Omega \times I, \tag{85}$$

$$u = a \quad \text{on } \partial\Omega \times [0, T], \tag{86}$$

$$u(0) = g \quad \text{in } \Omega \times t = 0, \tag{87}$$

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.

6.3 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 Crank-Nicolson ($\theta = 0.5$) as we already studied in Section 4.

Definition 6.2 (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 towards the implicit side;*
- $\theta = 1$*: 1st order implicit Euler time stepping.*

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 $\delta t = \frac{T}{N_T}$:

$$\partial_t u - \nabla\cdot(\alpha\nabla u) = f \tag{88}$$

$$\Rightarrow \quad \frac{u^n - u^{n-1}}{\delta t} - \theta\nabla\cdot(\alpha\nabla u^n) - (1-\theta)\nabla\cdot(\alpha\nabla u^{n-1}) = \theta f^n + (1-\theta)f^{n-1} \tag{89}$$

$$\Rightarrow \quad u^n - \delta t\theta\nabla\cdot(\alpha\nabla u^n) = u^{n-1} + \delta t(1-\theta)\nabla\cdot(\alpha\nabla u^{n-1}) + \delta t\theta f^n + \delta t(1-\theta)f^{n-1} \tag{90}$$

where $u^n := u(t^n)$ and $u^{n-1} := u(t^{n-1})$ and $f^n := f(t^n)$ and $f^{n-1} := f(t^{n-1})$.

6.4 Spatial discretization

We take the temporally discretized problem and use a Galerkin finite element scheme to discretize in space as explained in Section 5. That is we multiply with a test function and integrate. We then obtain: Find $u_h \in \{a + V_h\}$ such that for $n = 1, 2, \dots N_T$:

$$(u_h^n, \varphi_h) - \delta t\theta(\alpha\nabla u_h^n, \nabla\varphi_h) = (u^{n-1}, \varphi_h) + \delta t(1-\theta)(\alpha\nabla u^{n-1}, \nabla\varphi_h) + \delta t\theta(f^n, \varphi_h) + \delta t(1-\theta)(f^{n-1}, \varphi_h) \tag{91}$$

for all $\varphi_h \in V_h$.

Definition 6.3 (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:

$$(u_h^n, \varphi_h) = (u_h^{n-1}, \varphi_h) - \delta t(\nabla u_h^{n-1}, \nabla\varphi_h) \tag{92}$$

The structure of this equation, namely $y^n = y^{n-1} + f(t^{n-1}, y^{n-1})$, is the same as we have had in Section 4. On the other hand, the single terms in the weak form are akin to our derivations in Section 5.2.3. 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:

$$\sum_{j=1}^{N_x} u_{h,j}(\varphi_{h,j}, \varphi_{h,i}) = (u^{n-1}, \varphi_{h,i}) + \delta t(\nabla u_h^{n-1}, \nabla\varphi_{h,i})$$

which results in a linear equation system: $MU = B$. Here

$$\begin{aligned}(M_{ij})_{i,j=1}^{N_x} &:= (\varphi_{h,j}, \varphi_{h,i}),\\ (U_j)_{j=1}^{N_x} &:= (u_{h,1}, \dots, u_{h,N})\\ (B_i)_{i=1}^{N_x} &:= (u_h^{n-1}, \varphi_{h,i}) + \delta t(\nabla u_h^{n-1}, \nabla\varphi_{h,i})\end{aligned}$$

Here, the mass matrix M (also known as the Gramian matrix) is always the same for fixed h and can be explicitely computed. Formally we arrive at

$$Mu_h^n = Mu_h^{n-1} - \delta tKu_h^{n-1} \quad \Rightarrow \quad u_h^n = u_h^{n-1} - \delta tM^{-1}Ku_h^{n-1}.$$

Remark 6.4. *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 δt has to be chosen too small in order to obtain stable numerical results; see the numerical analysis in Section 6.7.*

6.5 Evaluation of the integrals

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

$$\begin{aligned} m_{i,i+1} &= \int_\Omega \varphi_{h,i+1}(x)\varphi_{h,i}(x)\,dx = \int_{x_i}^{x_{i+1}} \frac{x - x_i}{h}\cdot\frac{x_{i+1} - x}{h}\,dx = \int_{x_i}^{x_{i+1}} \frac{x - x_i}{h}\cdot\frac{x_i - x + h}{h}\,dx \\ &= \dots \\ &= -\frac{h}{3} + \frac{h^2}{2h} = \frac{h}{6}. \end{aligned}$$

It is trivial to see that $m_{i,i+1} = m_{i-1,i}$. Summarizing all entries results in

$$M = \frac{h}{6}\begin{pmatrix} 4 & 1 & & & 0 \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ 0 & & & 1 & 4 \end{pmatrix}.$$

6.6 Final algorithms

Forward Euler: Given the initial condition g, solve for $n = 1, 2, 3, \dots, N_T$

$$Mu_h^n = Mu_h^{n-1} - \delta t K u_h^{n-1} \quad\Rightarrow\quad u_h^n = u_h^{n-1} - \delta t M^{-1} K u_h^{n-1},$$

where $u_h^n, u_h^{n-1} \in \mathbb{R}^{N_x}$.

The backward Euler scheme reads:

$$Mu_h^n + \delta t K u_h^n = Mu_h^{n-1}$$

Exercise 12. *Let $\Omega = (0,1)$. Implement a finite element scheme for solving*

$$\begin{aligned} \partial_t u - \nabla \cdot (\nabla u) &= 1 \quad \text{in } \Omega, &(93)\\ u &= 0 \quad \text{on } \partial\Omega, &(94)\\ u(0) &= 0 \quad \text{in } \Omega \times \{t = 0\}. &(95) \end{aligned}$$

in octave or C++ or python.

Exercise 13. *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*

$$\begin{aligned} \rho\partial_t u - \nabla \cdot (\alpha\nabla u) &= f \quad \text{in } \Omega \times I, &(96)\\ u &= a \quad \text{on } \partial\Omega \times [0,T], &(97)\\ u(0) &= g \quad \text{in } \Omega \times t = 0, &(98) \end{aligned}$$

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?

6.7 Numerical analysis: stability analysis

The heat equation is a good example of a stiff problem (for the definition we refer back to Section 4.2). We first ask the question under which conditions the numerical solution is stable? A basic stability estimate is:

$$\|u_h^n\| \leq \|u_h^0\| \leq \|u^0\|$$

which can be derived by plugging the test function $\varphi := u_h^n$ into the weak formulation and applying Young's inequality for the first left inequality. For the right estimate, we work with the initial condition $(u_h^0, \phi_h) = (u^0, \phi_h)$ and using as test function u_h^0 and estimating with Young's inequality. A better estimate is very similar to our ODE theory. We multiply in

$$Mu_h^n + \delta t K u_h^n = M u_h^{n-1}$$

by M^{-1} (assuming that M is regular, which is the case since M is the mass matrix):

$$u_h^n + \delta t M^{-1} K u_h^n = u_h^{n-1}.$$

Then
$$(I + \delta t M^{-1}K)u_h^n = u_h^{n-1} \quad \Rightarrow \quad u_h^n = (I + \delta t M^{-1}K)^{-1} u_h^{n-1} \tag{99}$$
Before continuing, we compare this expression to our ODE chapter. Let μ_j be the eigenvalues of the matrix $M^{-1}K$ ranging from $O(1)$ to $O(h^{-2})$; for a proof, we refer the reader to [19][Section 7.7]. Then:

$$|(I + \delta t M^{-1}K)^{-1}| = \max_j \frac{1}{1 + \delta t \mu_j}.$$

To have a stable scheme, we must have:
$$\max_j \frac{1}{1 + \delta t \mu_j} < 1.$$
We clearly have then again from (99):

$$\|u_h^n\| \le \|u_h^{n-1}\| \quad \Rightarrow \quad \|u_h^n\| \le \|u_h^0\|$$

Furthermore, we see that the largest μ_j 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.** But this property renders the forward Euler infeasible. Here, the stability condition reads:

$$u_h^n = (I - \delta t M^{-1}K)u_h^{n-1}$$

yielding
$$|(I - \delta t M^{-1}K)^{-1}| = |\max_j (1 - \delta t \mu_j)| = |1 - \delta t \mu_M|.$$
As before $\mu_M = O(h^{-2})$. To establish a stability estimate of the form $\|u_h^n\| \le \|u_h^{n-1}\|$, it is required that

$$|1 - \delta t \mu_M| \le 1.$$

Resolving this estimate, we obtain

$$\delta t \mu_M \le 2 \quad \Leftrightarrow \quad \delta t \le \frac{2}{\mu_M} = O(h^2),$$

thus
$$\delta \le C h^2, \quad C > 0.$$
Here, we only have **conditional stability in time** as we have already seen in Chapter 4. Moreover, the time step size 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 praxis this is in most cases not attrative at all and for this reason the forward Euler scheme does not play a role.

7 Towards nonlinear problems

In this final section, we want to give a brief idea how to tackle nonlinear problems. A lot of interesting situations are nonlinear and a basic course in numerics should teach (at least to give a flavor) how to tackle such problems. On the other hand, nonlinear solvers have been already required by solving IVP using implicit (backward Euler, Crank-Nicolson) time stepping schemes. Thus, there are several reasons to introduce nonlinear techniques in this class.

7.1 Preface

Let us first recall some examples of nonlinear differential equations (in fact exclusively PDEs):

1. Nonlinear solid mechanics: either geometrically nonlinear or through the stress-strain law; e.g., [7, 17];
2. Fluid mechanics: Navier-Stokes equations, e.g., [24];
3. Non-Newtonian fluid flow (e.g., p-Stokes), e.g., [16];
4. Fluid-structure interaction (nonlinear coupled PDE systems), e.g., [25];
5. Multiphase flow in porous media (nonlinear coupled PDE systems), e.g., [6];
6. In general every situation in which different physical phenomena interact;
7. Numerical optimization, e.g., [1, 20].

To tackle such problems, a nonlinear solver is indispensable, but for coupled PDEs systems, further decisions have to be made. But we only list them without providing (unfortunately) any further details in these lecture notes. These are:

- A robust and efficient nonlinear solver (a brief introduction with some details is provided in Section 7.3);
- A robust and efficient linear solver;
- For PDE systems: the way of coupling: partitioned (sequentially) or monolithic
- Combination of different coordinate systems: for instance when solids (described in Lagrangian coordinates) interact with fluid dynamics (described in Eulerian coordinates)

7.2 Simple linearization methods

Let us consider the Navier-Stokes equations:

$$\begin{aligned} \partial_t v + v \cdot \nabla v - \nabla \cdot (\nabla v) + \nabla p &= f, \\ \nabla \cdot v &= 0. \end{aligned}$$

The nonlinear term is the convection term:

$$v \cdot \nabla v.$$

Rather using Newton (explained in the following sections), one often tries to approximate nonlinearities by some other means. For instance, we could use

$$\tilde{v} \cdot \nabla v,$$

where $\tilde{v}$ is some good approximation to v. For instance, one choice is

$$\tilde{v} := v^{n-1}$$

namely taking the old time step solution. This linearization is also known as **Oseen linearization**. Another, and indeed simplest way, is the so-called **Stokes linearization** in which the entire convection term is evaluated at the previous time step t^{n-1}:

$$v^{n-1} \cdot \nabla v^{n-1}.$$

Such time-lagging methods are often used in software and literature to approximate nonlinear system and to linearize them. On the other hand, when we want to treat everything fully implicitly, we need to adopt Newton's method that we discuss in the following.

7.3 Newton's method

There exist different approaches to numerically solve nonlinear problems: the most known are fixed-point approaches and Newton's method. In particular the latter approach can be divided into several variants; general developments are addressed in [9] or in terms of optimization (but as well applicable for classical PDE problems) in [20].

7.3.1 Classical Newton's method

Let $f \in C^1[a,b]$ with at least one point $f(x) = 0$, and $x_0 \in [a,b]$ be a so-called initial guess. The task is to find $x \in \mathbb{R}$ such that

$$f(x) = 0.$$

In most cases it is impossible to calculate x explicitly. Rather we construct a sequence of iterates $(x_k)_{k\in\mathbb{R}}$ and hopefully reach at some point

$$|f(x_k)| < TOL, \quad \text{where } TOL \text{ is small, e.g., } TOL = 10^{-10}.$$

What is true for all Newton derivations in the literature is that one has to start with a Taylor expansion. In our lecture we do this as follows. Let us assume that we are at x_k and can evaluate $f(x_k)$. Now we want to compute this next iterate x_{k+1} with the unknown value $f(x_{k+1})$. Taylor gives us:

$$f(x_{k+1}) = f(x_k) + f'(x_k)(x_{k+1} - x_k) + o(x_{k+1} - x_k)^2$$

We assume that $f(x_{k+1}) = 0$ (or very close to zero $f(x_{k+1}) \approx 0$). Then, x_{k+1} is the sought root and neglecting the higher-order terms we obtain:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k).$$

Thus:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots. \tag{100}$$

This iteration is possible as long as $f'(x_k) \neq 0$.

Remark 7.1 (Relation to Section 5.2.7)**.** *We see that Newton's method can be written as*

$$x_{k+1} = x_k + d_k, \quad k = 0, 1, 2, \ldots,$$

where the search direction is

$$d_k = -\frac{f(x_k)}{f'(x_k)}.$$

The iteration (100) terminates if a stopping criterion

$$\frac{|x_{k+1} - x_k|}{|x_k|} < TOL, \quad \text{or} \quad |x_{k+1} - x_k| < TOL, \tag{101}$$

or

$$|f(x_{k+1})| < TOL. \tag{102}$$

is fulfilled. All these TOL do not need to be the same, but sufficiently small.

Remark 7.2. *Newton's method belongs to fix-point iteration schemes with the iteraction function:*

$$F(x) := x - \frac{f(x)}{f'(x)}. \tag{103}$$

For a fixed point $\hat{x} = F(\hat{x})$ it holds: $f(\hat{x}) = 0$. Compare again to Section 5.2.7.

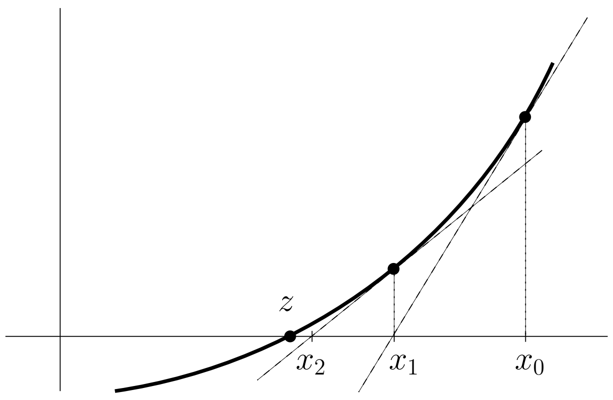

Figure 9: Geometrical interpretation of Newton's method. Figure taken from [21].

The main results is given by:

Theorem 7.3 (Newton's method)**.** *The function $f \in C^2[a,b]$ has a root $\hat{x}$ in the interval $[a,b]$ and*

$$m := \min_{a \le x \le b} |f'(x)| > 0, \quad M := \max_{a \le x \le b} |f''(x)|.$$

Let $\rho > 0$ such that

$$q := \frac{M}{2m}\rho < 1, \quad K_\rho(\hat{x}) := \{x \in \mathbb{R} : |x - \hat{x}| \le \rho\} \subset [a,b].$$

Then, for any starting point $x_0 \in K_\rho(\hat{x})$, the sequence of iterations $x_k \in K_\rho(\hat{x})$ converges to the root $\hat{x}$. Furthermore, we have the a priori estimate

$$|x_k - \hat{x}| \le \frac{2m}{M} q^{2^k}, \quad k \in \mathbb{N},$$

and a posteriori estimate

$$|x_k - \hat{x}| \le \frac{1}{m}|f(x_k)| \le \frac{M}{2m}|x_k - x_{k+1}|^2, \quad k \in \mathbb{N}.$$

Often (and in particular for higher-dimensional problems such as fluid-structure interaction), Newton's method is formulated in terms of a defect-correction scheme.

Definition 7.4 (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 7.5 (Newton's method as defect-correction scheme)**.**

$$\begin{aligned} f'(x_k)\delta x &= d_k, \quad d_k := y - f(x_k), \\ x_{k+1} &= x_k + \delta x, \quad k = 0, 1, 2, \ldots. \end{aligned}$$

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 $A\delta x = b$ and computing the inverse matrix A^{-1} is an expensive operation. $\diamond$

Remark 7.6. *This previous forms of Newton's method are already very close to the schemes that are used in research. One simply extends from $\mathbb{R}^1$ to higher dimensional cases 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, 20].

Exercise 14. *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. *Implement Newton's method and compute the solution $\hat{x}$ to a tolerance $TOL = 1e - 10$.*

Exercise 15. *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 minimum of L. What needs to be done?*
2. *Formulate Newton's method (formally; without implementation) for the previous minimization problem.*

7.3.2 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 digits doubles at each iteration step.

7.3.3 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 7.5 and implemented in octave in Section 11.4. 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
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.

Exercise 16. *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?*

7.4 Solving IVPs with implicit schemes

In Section 4.4.2, we introduced implicit schemes to solve ODEs. In specific cases, one can explicitely arrange the terms in such a way that an implicit schemes is obtained. This was the procedure for the ODE model problem in Section 11.1. However, such an explicit representation is not always possible. Rather we have to solve a nonlinear problem inside the time-stepping process. This nonlinear problem will be formulated in terms of a Newton scheme.

We recall:

$$y'(t) = f(t, y(t))$$

which reads after backward Euler discretization:

$$\frac{y_n - y_{n-1}}{k_n} = f(t_n, y_n(t_n)).$$

After re-arranging some terms we obtain the root-finding problem:

$$g(y_n) := y_n - y_{n-1} - k_n f(t_n, y_n(t_n)) = 0.$$

Taylor expansion yields (where the Newton iteration index is denoted by 'l'):

$$0 = g(y_n^l) + g'(y_n^l)(y_n^{l+1} - y_n^l) + o((y_n^{l+1} - y_n^l)^2),$$

where the Newton matrix (the Jacobian) is given by

$$g'(y_n^l) := I - k_n f'_y(t_n, y_n^l).$$

The derivative f'_y is with respect to the unknown y_n.

In defect correction notation, we have

Algorithm 7.7. *Given a starting value, e.g.,* y_n^0 *we have for* $l = 1, 2, 3, \ldots$*:*

$$g'(y_n^l)\delta y = -g(y_n^l), \tag{104}$$
$$y_n^{l+1} = y_n^l + \delta y \tag{105}$$

Stop if $\|g(y_n^{l+1})\| < TOL_N$ *with e.g.,* $TOL_N = 10^{-8}$.

The difficulty here is to understand the mechanism of the two different indices n and l. As before, we want to compute for a new time point t^{n+1} a solution y_{n+1}. However, in most cases y_{n+1} cannot be constructed explicitely as done for example in Section 5.3. Rather we need to approximate y_{n+1} by another numerical scheme. Here Newton's method is one efficient possibility, in which we start with a rough approximation for y_{n+1}, which is denoted as initial Newton guess $y_{n+1}^0 \approx y_{n+1}$. In order to improve this initial guess, we start a Newton iteration labeled by l and (hope!) to find better approximations

$$y_{n+1}^1, y_{n+1}^2, \ldots$$

for the sought value y_{n+1}. Mathematically speaking, we hope that

$$|y_{n+1}^l - y_{n+1}| \to 0, \quad \text{for } l \to \infty.$$

To make this procedure clear:

- we are interested in the true value $y(t^{n+1})$, which is however in most cases not possible to be computed in an explicit way.
- Thus, we use a finite difference scheme (e.g., backward Euler) to approximate $y(t^{n+1})$ by y_{n+1}.
- However, in many cases this finite difference scheme (precisely speaking: implicit schemes) is still not enough to obtain directly y_{n+1}. Therefore, we need a second scheme (here Newton) to approximate y_{n+1} by y^l_{n+1}.

Remark 7.8. *To avoid two numerical schemes is one reason, why explicit schemes are nevertheless very popular in practice. Here the second scheme can be omitted.*

Remark 7.9. *On the other hand, this example is characteristic for many numerical algorithms: very often, we have several algorithms that interact with each other. One important question is how to order these algorithms such that the overall numerical scheme is robust and efficient.*

8 Computational convergence analysis

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.

8.1 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.

8.1.1 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 8.1. *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
...
```

□

8.1.2 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 8.2. *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 8.3 (Linear and quadratice 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. For an illustration, we refer the reader to Section 4.6.

Proposition 8.4 (Linear and quadratice 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

$$O(h^2) = O\Big(\frac{1}{N}\Big)$$

8.2 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 is given in Section 7.4 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 8.5. *Let us assume that we have an iterative scheme to compute a root z. The iteration converges with order p when*

$$|x_k - z| \leq c|x_{k-1} - z|^p$$

with $p \geq 1$ and $c = const$. In more detail:

- *Linear convergence: $c \in (0,1)$ and $p = 1$;*
- *Superlinear convergence: $c \to 0$ 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.

Corollary 8.6 (Rule of thumb)**.** *A rule of thumb for quadratic convergence is: the number of correct digits doubles at each step. From our previous section, we have the example:*

```
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
```

9 Concluding summary

9.1 Summary

The purpose of these notes has been on an introduction to numerical modeling, or more generally-speaking, an introduction to scientific computing. Here we restricted our focus to differential equations. The three key aspects of scientific computing are:

- Mathematical modeling of a given problem and its classification;
- Development and analysis of numerical algorithms;
- Implementation and debugging of programming code;

and they are linked by a fourth aspect:

- Interpretation of the obtained results and justification of their correctness.

We illustrated all these four steps in terms of ODEs and PDEs. The choice of differential equations has been made because they cover a wide range of possible applications in various fields such as continuum mechanics, fluid mechanics, solid mechanics, as well as engineering, physics, chemistry, biology, economics, and financial mathematics. Most of the algorithmic developments have been complemented with programming code in octave/MATLAB and python such that the concepts can be immediately used by the reader.

9.2 The principles of work, study and research tasks

The intention of this section is provide some help for the work on the student projects done in groups by two.

The basic principle is:

WhatHowWhy?

- **What** are we doing (Problem statement)?
- **How** are we doing this (How are we going to solve that specific problem)?
- **Why** is the solution of this problem really important?

This can be further refined as:

WhatHowWhatHowWhy?

- **What** are we doing (Problem statement)?
- **How** are we doing this (How are we going to solve that specific problem)?
- **What** are our findings (results)?
- **How** are we going to interpret these results? (Is the code correct? Do the results correspond to my physical and mathematical intuition? If I have doubts, how can I verify the correctness of my numerical model and code?)
- **Why** is the solution of this problem really important?

Example 9.1. *Let us take as an example the little numerical projects for this class and identify the above steps:*

- ***What** are we doing (Problem statement)?*
  ***Answer:** We are solving an ODE or PDE. Let us say an ODE for the following points.*

- ***How*** *are we doing this (How are we going to solve that specific problem)?*
  Answer: *We apply for example the forward Euler scheme. But the 'How' might include more. Often we do not understand a given problem in its full complexity. (Actually this arises quite often!) In order to tackle the final problem, we should start with a simplification. For example, we make a nonlinear problem linear. Or a 2D problem is reduced to 1D. And so forth. To be able to make simplifications and to find examples to 'illustrate' a problem in some other way is in many cases the key to really understand this problem.*

- ***What*** *are our findings (results)?*
  Answer: *Take an initial value and end time value. Choose a step size and compute. Display your solution as a plot for example or compute the error to an exact solution if available (see Section 5.3).*

- ***How*** *are we going to interpret these results? (Is the code correct? Do the results correspond to my physical and mathematical intuition? If I have doubts, how can I verify the correctness of my numerical model and code?)*
  Answer: *For forward Euler and too big step sizes we obtain oscillations in the solution. It is now important to distinguish if these oscillations are on purpose (maybe we had some sinus-oscillation as initial condition) or a numerical artefact. Our stability brings us to the conclusion that in this case, we really had a numerical artefact that needs to be avoided. Specifically our step size k was too big and we learned in Thomas' lecture that the forward Euler scheme is unstable if the step sizes are too big.*

- ***Why*** *is the solution of this problem really important?*
  Answer 1: *Because Thomas gave this exercise to pass this class!*
  Answer 2: *In research or industry or private sector or your own company: Who is interested in the solution of your results? Why did you tackle this task? What can you solve now what nobody before could solve? Or why did your new algorithm provide new insight into already known problems? These are some questions that should answer the* ***Why****.*

9.3 Hints for presenting results in a presentation/talk

We collect some hints to make good presentations using, for instance, beamer latex or PowerPoint:

1. Rule of thumb: one minute per slide. For a 20 minutes presentation a good mean value is to have 20 slides.
2. A big risk of beamer/Power-Point presentations is that we rush over the slides. You are the expert and know everything! But your audience is not. Give the audience time to read the slide!
3. Taylor your talk to the audience!
4. Put a title on each slide!
5. In plots and figures: make lines, the legend, and axes big enough such that people from the back can still read your plots. A good **no go, bad** example are my figures 6 and 7 in which the legend is by far too small. Do not do that! A good example is Figure 9.
6. Tables with numbers: if you create columns of numbers do not put all of them. If you have many, you may mark the most important in a different color to highlight them. For instance, I used dots ... in Section 7.3.3 to neglect Newton iterates, which are of less importance and do not contribute to a further understanding.
7. Structure your talk clearly: what is the goal, how did you solve it, what are your results, what is your interpretation? See Section 9.2.
8. Do not glue to your slides. Despite you have everything there, try to explain things in a free speech.
9. Less is more:
   a) Do not put too much text on each slide!

   b) Do not write full sentences, but use bullet points!
   c) Do not use too fancy graphics charts, plots, etc, where everything is moving, etc.
   d) Do not try to pack everything you have learned into one presentation. Everybody believes that you know many, many things. The idea of a presentation is to present in a clear way a piece of your work!

10. As always: the first sentence is the most difficult one: practice for yourself some nice welcome words to get smoothly started.

11. Practice your final presentation ahead of time either alone, with friends or colleagues.

12. Just be authentic during the presentation.

13. If you easily become nervous avoid coffee before the presentation.

10 Quiz at the beginning

10.1 Questions

1. Which programming languages do you know?
2. Which programming languages have you used so far?
3. What is a derivative of a function $f : \mathbb{R} \to \mathbb{R}$? Give an example.
4. What is a partial derivative? Give an example.
5. Differentiate $f(x, y) = x^2 + y^3$ with respect to y.
6. Differentiate $f(x, y) = x^2y^3$ with respect to x and y.
7. What is the Hessian matrix?
8. What does Δu for a function $u : \mathbb{R}^2 \to \mathbb{R}$ mean?
9. How is the divergence operator defined and what is its physical meaning?
10. What is $\nabla \cdot v$ for a function $v : \mathbb{R}^2 \to \mathbb{R}^2$?
11. What is a norm? Why do we need a norm? Can you define the defining properties of a norm?
12. In case you have heard about 'numerical modeling' or 'scientific computing': what is your understanding of these terminologies?
13. Have you had classes on differential equations so far?
14. Did you get to know differential equations in some other classes, e.g., engineering classes? If yes, can you give an example?
15. What is a sequence of numbers (e.g., give an example) and what is convergence?
16. Do you know how to solve a linear equation system $Ax = b$ with $A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^n$?
17. Solve (by hand) the following linear equation system:
$$\begin{aligned} x + 2y + z &= 4 \\ -2x + y + 3z &= 7 \\ 2x - 3y - 2z &= -10 \end{aligned}$$
Formulate first A, x and b and compute then the solution of $Ax = b$.
18. Evaluate $\int_{-2}^{3} x^3 \, dx$
19. What is the substitution rule (in 1D and higher dimensions)?
20. What is integration by parts (in 1D and higher dimensions)?
21. Evaluate the concrete points x_j for $n = 4$ using the following definition:
$$x_j = jh, \quad h = \frac{1}{n+1}, \quad 0 \le j \le n+1, \quad h = x_{j+1} - x_j.$$
22. What is a Taylor expansion? Why is Taylor useful?
23. What is a metric space? Give an example?
24. How is the space $C[a, b]$ defined?

25. Define the triangle inequality. How is Young's inequality defined?
26. What is the Cauchy-Schwarz inequality?
27. Do you know what a Cauchy sequence is?
28. What is a complex number? Give an example.

10.2 Exercises

Exercise 17. *Implement in your preferred programming language the phrase 'Hello MAP 502', which is then printed on the screen.*

Exercise 18. *Implement the calculation* $1+5$ *in your preferred programming language and print the result on the screen.*

Exercise 19. *Evaluate*

$$\int_0^{\pi/4} x^3 \cdot \cos(x)\, dx$$

Exercise 20. *What is the limit of the sequence*

$$(1+\frac{x}{n})^n?$$

11 Programming codes

11.1 Code of Example 4.6 in Octave

We first define the three numerical schemes. These are defined in terms of octave-functions and which must be in the same directory where you later call these functions and run octave. Thus, the content of the next three functions is written into files with the names:

```
euler_forward_2.m  euler_backward_model_problem.m   trapezoidal_model_problem.m
```

Inside these m-files we have the following statements. Forward Euler:

```
function [x,y] = euler_forward_2(f,xinit,yinit,xfinal,n)
% Forward Euler scheme
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016

h = (xfinal - xinit)/n;

x = [xinit zeros(1,n)];
y = [yinit zeros(1,n)];

for k = 1:n
  x(k+1) = x(k)+h;
          y(k+1) = y(k) + h*f(x(k),y(k));
end
end
```

Backward Euler (here the right hand side is also unknown). For simplicity we explicitly manipulate the scheme for this specific ODE. In the general case, the right hand side is solved in terms of a nonlinear solver.

```
function [x,y] = euler_backward_model_problem(xinit,yinit,xfinal,n,a)
% Backward Euler scheme
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016

% Calculuate step size
h = (xfinal - xinit)/n;

% Initialize x and y as column vectors
x = [xinit zeros(1,n)];
y = [yinit zeros(1,n)];

% Implement method
for k = 1:n
  x(k+1) = x(k)+h;
          y(k+1) = 1./(1.0-h.*a) .*y(k);
end
end
```

And finally the trapezoidal rule (Crank-Nicolson):

```
function [x,y] = trapezoidal_model_problem(xinit,yinit,xfinal,n,a)
% Trapezoidal / Crank-Nicolson
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016

% Calculuate step size
h = (xfinal - xinit)/n;
```

```
% Initialize x and y as column vectors
x = [xinit zeros(1,n)];
y = [yinit zeros(1,n)];

% Implement method
for k = 1:n
  x(k+1) = x(k)+h;
          y(k+1) = 1./(1 - 0.5.*h.*a) .*(1 + 0.5.*h.*a).*y(k);
end
end
```

In order to have a re-usable code in which we can easily change further aspects, we do not work in the command line but define an octave-script; here with the name

```
step_1.m
```

that contains initialization and function calls, visualization, and error estimation:

```
% Script file: step_1.m
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016

% We solve three test scenarios
% Test 1: coeff_a = 0.25 as in the lecture notes.
% Test 2: coeff_a = -0.1
% Test 3: coeff_a = -10.0 to demonstrate
%    numerical instabilities using explizit schemes

% Define mathematical model, i.e., ODE
% coeff_a = g - m
coeff_a = 0.25;
f=@(t,y) coeff_a.*y;

% Initialization
t0 = 2011;
tN   = 2014;
num_steps = 16;
y0 = 2;

% Implement exact solution, which is
% available in this example
g=@(t) y0 .* exp(coeff_a .* (t-t0));
te=[2011:0.01:2014];
ye=[g(te)];

% Call forward and backward Euler methods
[t1,y1] = euler_forward_2(f,t0,y0,tN,num_steps);
[t2,y2] = euler_backward_model_problem(t0,y0,tN,num_steps,coeff_a);
[t3,y3] = trapezoidal_model_problem(t0,y0,tN,num_steps,coeff_a);

% Plot solution
plot (te,ye,t1,y1,t2,y2,t3,y3)
xlabel('t')
ylabel('y')
  legend('Exact','FE','BE','CN')
axis([2011 2014 0 6])
```

```
% Estimate relative discretization error
errorFE=['FE err.: ' num2str((ye(end)-y1(end))/ye(end))]
errorBE=['BE err.: ' num2str((ye(end)-y2(end))/ye(end))]
errorCN=['CN err.: ' num2str((ye(end)-y3(end))/ye(end))]

% Estimate absolute discretization error
errorABS_FE=['FE err.: ' num2str((ye(end)-y1(end)))]
errorABS_BE=['BE err.: ' num2str((ye(end)-y2(end)))]
errorABS_CN=['CN err.: ' num2str((ye(end)-y3(end)))]
```

Exercise 21. *Implement the backward Euler scheme and trapezoidal rule in a more general way that avoids the explicit manipulation with the right hand side $f(t,y) = ay(t)$. Hint: A possibility is (assuming that the function f is smooth enough) to formulate the above schemes as fixed-point equations and then either use a fixed-point method or Newton's (see Section 7.4) method to solve them. Apply these ideas to Example 3.*

11.2 Code of Example 5.3 in python

The programming code is based on python version 2.7. In order to perform scientific computing with python some additional packages might have to be installed first (on some computers they are pre-installed though):

- numpy
- scipy
- matplotlib

The full code to obtain the previous results is:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Thomas Wick
# Ecole Polytechnique
# MAP 502
# Winter 2016/2017

# Execution of *.py files
# Possiblity 1: in terminal
#  terminal> python3 file.py  # here file.py = poisson.py

# Possiblity 2: executing in an python environment
# such as spyder.

# This program has been tested with python
# version 2.7 on a SUSE linux system version 13.2

# Problem statement
# Numerical solution using finite
# elements of the Poisson problem:
#
# Find u such that
#
#      -d/dx (k d/dx) u = f in (0,1)
#       u(0) = u(1) = 0
#
#      with f=-1
```

```
#########################################################
# Load packages (need to be installed first if
# not yet done - but is not difficult)
import numpy as np
import matplotlib.pyplot as plt # for plot functons
plt.switch_backend('tkAgg')  # necessary for OS SUSE 13.1 version,
# otherwise, the plt.show() function will not display any window


import scipy as sp
from scipy import sparse
from scipy.sparse import linalg

import scipy.linalg # for the linear solution with Cholesky


#########################################################
# Setup of discretization and material parameters
N = 200 # Number of points in the discretization
x = np.linspace(0,1,N) # Initialize nodal points of the mesh

# Spatial discretization parameter (step length) : h_i=x_{i+1}-x_i
hx = np.zeros(len(x))
hx[:len(x)-1]=x[1:]-x[:len(x)-1]

h = max(hx) # Maximal element diameter

# Sanity check
#print h

# Material parameter
k= 1.0 * np.ones(len(x))


##########################################################
# Construct and plot exact solution and right hand side f

Nddl = N-2
xddl = x.copy()
xddl = xddl[1:len(x)-1]

# Initialize and construct exact solution
uexact = np.zeros(len(x))
uexact[1:len(x)-1] = 0.5 * (xddl * xddl - xddl)

# Construct right hand side vector
f = (-1) * np.ones(len(x)-2)


##########################################################
# Build system matrix and right hand side
Kjj = k[:N-2]/hx[:N-2] + k[1:N-1]/hx[1:N-1] # the diagonal of Kh
Kjjp = - k[1:N-2]/hx[1:N-2] # the two off-diagionals of Kh
```

```
# System matrix (also commenly known as stiffness matrix)
# First argument: the entries
# Second argument: in which diagonals shall the entries be sorted
Ah =sp.sparse.diags([Kjjp,Kjj,Kjjp],[-1,0,1],shape=(Nddl,Nddl))

# Right hand side vector
Fh= (hx[:N-2] + hx[1:N-1]) * 0.5 * f


#########################################################
# Solve system: Ah uh = Fh => uh = Ah^{-1}Fh
utilde = sp.sparse.linalg.spsolve(Ah,Fh)

# Add boundary conditions to uh
# The first and last entry are known to be zero.
# Therefore, we include them explicitely:
uh = np.zeros(len(uexact))
uh[1:len(x)-1]=utilde


#########################################################
# Plot solution
plt.clf()
plt.plot(x,uexact,'r', label='Exact solution $u$')
plt.plot(x,uh,'b+',label='Discrete FE solution $u_h$')
plt.legend(loc='best')
plt.xlim([-0.1,1.1])
plt.ylim([-0.15,0.01])
plt.show()
```

Exercise 22. *In order to obtain more quantitative results, the following extensions of the above code could be done:*

- *Computational error analysis (which error norm?) to study the error between the exact and numerical solution on a sequence of refined meshes;*
- *Implementing high-order finite elements and carrying out again error analysis.*

11.3 Code for a straightforward implementation of Newton's method (Example 7.3.2)

The programming code for standard Newton's method is as follows:

```
% Script file: step_2.m
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016
% We seek a root of a nonlinear problem

format long

% Define nonlinear function of which
% we want to find the root
f = @(x) x^2 - 2;

% Define derivative of the nonlinear function
df = @(x) 2*x;
```

```
% We need an initial guess
x = 3;

% Define function
y = f(x);

% Set iteration counter
it_counter = 0;
printf('%i %e %e\n',it_counter,x,y);

% Set stopping tolerance
TOL = 1e-10;

% Perform Newton loop
while abs(y) > TOL
   dy = df(x);
   x = x - y/dy;
   y = f(x);
   it_counter = it_counter + 1;
   printf('%i %e %e\n',it_counter,x,y);
end
```

11.4 Code for a Newton defect-correction scheme including line search (Example 7.3.3)

As second implementation we present Newton's method as defect correction scheme as introduced in Definition 7.5. 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 $\omega < 1$. The octave code reads:

```
% Script file: step_3.m
% Author: Thomas Wick, CMAP, Ecole Polytechnique, 2016
% We seek a root of a nonlinear problem

format long

% Define nonlinear function of which
% we want to find the root
f = @(x) x^2 - 2;

% Define derivative of the nonlinear function
df = @(x) 2*x;

% We need an initial guess
x = 3;

% Define function
y = f(x);

% Set iteration counter
it_counter = 0;
printf('%i %e %e\n',it_counter,x,y);

% Set stopping tolerance
TOL = 1e-10;
```

```
% Line search parameter
% omega \in [0,1]
% where omega = 1 corresponds to full Newton
omega = 1.0;

% Newtons method as defect correction
% and line search parameter omega
while abs(y) > TOL
   dy = df(x);
   % Defect step
   dx = - y/dy;
   % Correction step
  x = x + omega * dx;
   y = f(x);
   it_counter = it_counter + 1;
   printf('%i %e %e\n',it_counter,x,y);
end
```

References

[1] G. ALLAIRE, *Analyse numerique et optimisation*, Editions de l'Ecole Polytechnique, 2005.

[2] ——, *An Introduction to Mathematical Modelling and Numerical Simulation*, Oxford University Press, 2007.

[3] G. ALLAIRE AND F. ALOUGES, *Analyse variationnelle des équations aux dÃ©rivÃ©es partielles.* MAP 431: Lecture notes at Ecole Polytechnique, 2016.

[4] W. BANGERTH, R. HARTMANN, AND G. KANSCHAT, *deal.II – a general purpose object oriented finite element library*, ACM Trans. Math. Softw., 33 (2007), pp. 24/1–24/27.

[5] G. F. CAREY AND J. T. ODEN, *Finite Elements. Volume III. Compuational Aspects*, The Texas Finite Element Series, Prentice-Hall, Inc., Englewood Cliffs, 1984.

[6] G. CHAVENT AND J. JAFFRÉ, *Mathematical models and finite elements for reservoir simulation: single phase, multiphase and multicomponent flows through porous media*, vol. 17, Elsevier, 1986.

[7] P. G. CIARLET, *Mathematical Elasticity. Volume 1: Three Dimensional Elasticity*, North-Holland, 1984.

[8] P. G. CIARLET, *The finite element method for elliptic problems*, North-Holland, Amsterdam [u.a.], 2. pr. ed., 1987.

[9] P. DEUFLHARD, *Newton Methods for Nonlinear Problems*, vol. 35 of Springer Series in Computational Mathematics, Springer Berlin Heidelberg, 2011.

[10] C. ECK, H. GARCKE, AND P. KNABNER, *Mathematische Modellierung*, Springer, 2008.

[11] L. C. EVANS, *Partial differential equations*, American Mathematical Society, 2010.

[12] 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 (2017), pp. 1–14.

[13] P. GRISVARD, *Elliptic Problems in Nonsmooth Domains*, vol. 24, Pitman Advanced Publishing Program, Boston, 1985.

[14] C. GROSSMANN, H.-G. ROOS, AND M. STYNES, *Numerical Treatment of Partial Differential Equations*, Springer, 2007.

[15] M. HANKE-BOURGEOIS, *Grundlagen der numerischen Mathematik und des Wissenschaftlichen Rechnens*, Vieweg-Teubner Verlag, 2009.

[16] A. HIRN, *Finite Element Approximation of Problems in Non-Newtionian Fluid Mechanics*, PhD thesis, University of Heidelberg, 2012.

[17] G. HOLZAPFEL, *Nonlinear Solid Mechanics: A continuum approach for engineering*, John Wiley and Sons, LTD, 2000.

[18] T. HUGHES, *The finite element method*, Dover Publications, 2000.

[19] C. JOHNSON, *Numerical solution of partial differential equations by the finite element method*, Cambridge University Press, Campridge, 1987.

[20] J. NOCEDAL AND S. J. WRIGHT, *Numerical optimization*, Springer Ser. Oper. Res. Financial Engrg., 2006.

[21] T. RICHTER AND T. WICK, *Einfuehrung in die numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele; book in production*, Springer, 2017.

[22] M. Schäfer and S. Turek, *Flow Simulation with High-Performance Computer II*, vol. 52 of Notes on Numerical Fluid Mechanics, Vieweg, Braunschweig Wiesbaden, 1996, ch. Benchmark Computations of laminar flow around a cylinder.

[23] G. Strang, *Computational Science and Engineering*, Wellesley-Cambridge Press, 2007.

[24] R. Temam, *Navier-Stokes Equations: Theory and Numerical Analysis*, AMS Chelsea Publishing, Providence, Rhode Island, 2001.

[25] T. Wick, *Modeling, discretization, optimization, and simulation of fluid-structure interaction.* Lecture notes at Heidelberg University, TU Munich, and JKU Linz available on https://www-m17.ma.tum.de/Lehrstuhl/LehreSoSe15NMFSIEn, 2015.

Index