PeC^3 Spring School on

Introduction to Numerical Modeling with Differential Equations

Thomas Wick[1] and Peter Bastian[2]

[1]Leibniz Universität Hannover
Institut für Angewandte Mathematik
Welfengarten 1, 30167 Hannover
email: thomas.wick@ifam.uni-hannover.de

[2]Universität Heidelberg
Interdisziplinäres Zentrum für Wiss. Rechnen
Im Neuenheimer Feld 205, D-69120 Heidelberg
email: Peter.Bastian@iwr.uni-heidelberg.de

Universidad Nacional Agraria La Molina,Lima, Perú, October 23–31, 2019

UNIVERSIDAD NACIONAL AGRARIA LA MOLINA
DAAD Deutscher Akademischer Austausch Dienst German Academic Exchange Service
BMZ Federal Ministry for Economic Cooperation and Development
PeC^3 Peruvian Competence Center of Scientific Computing

Acknowledgments

This spring school is part of the DAAD (German academic exchange service) program

PeC^3 : Peruvian Competence Center of Scientific Computing

https://www.pec3.org/

- The support from the DAAD and the Federal Ministry for Economic Cooperation and Development is gratefully acknowledged.
- Moreover, we highly appreciate the excellent support of the local organizers, specifically **Dandy Rueda Castillo** and **Edgar, Santisteban León** and **Aldo Alcides, Mendoza Uribe**

Structure of this spring school

1. Four days
2. Nine lectures à 90 minutes
3. Four practical excercises (one per day) in C++ ranging from 90 - 180 minutes per day (including explanations and summary of the main findings per day)
4. This spring school is accompanied by a three-day pre-course with an introduction to C++ and `hdnum`. The git repository for `hdnum` can be accessed here:

```
git clone https://parcomp-git.iwr.uni-heidelberg.de/Teaching/hdnum
```

Key literature related to this spring school

1. K. Eriksson, D. Estep, P. Hansbo, C. Johnson; Computational Differential Equations, Vol. 1, Cambridge University Press, 538 pages, 1996; online pdf-version available from the year 2009 http://www.csc.kth.se/~jjan/private/cde.pdf
2. A. Quarteroni, F. Saleri, P. Gervasio; Scientific Computing with MATLAB and octave, Springer 2014
3. R. Rannacher; Numerik 1: Numerik gewöhnlicher Differentialgleichungen, Heidelberg University Publishing, 2017 (in german)
4. P. Bastian; Vorlesung Numerik, lecture notes, Heidelberg University, 2017
5. P. Bastian; Lecture Notes Scientific Computing with Partial Differential Equations, Universität Heidelberg, WS 2017/2018. https://conan.iwr.uni-heidelberg.de/data/teaching/finiteelements_ws2017/num2.pdf
6. T. Wick; Numerical methods for partial differential equations; Lecture notes, Leibniz Universität Hannover, 2018, online available http://www.thomaswick.org/links/lecture_notes_Numerics_PDEs_Oct_12_2019.pdf
7. T. Wick; Introduction to Numerical Modeling, lecture notes, MAP 502, Ecole Polytechnique, 2018, online available http://www.thomaswick.org/map_502_winter_2018_engl.html
8. O. Klein; Lecture Notes Object-Oriented Programming for Scientific Computing, 2018, https://conan.iwr.uni-heidelberg.de/teaching/oopfsc_ss2018/
9. O. Klein; Lecture Notes Einführung in die Numerik, 2018, https://conan.iwr.uni-heidelberg.de/teaching/numerik0_ss2018/

Contents

Contents

The Modern Scientific Method

- *Experiment*: Observe and measure some phenomenon. Nowadays the amount of data acquired may be abundant.
- *Theory*: Try to explain observations by a model. Here we consider mathematical models in terms of differential equations.
- *Scientific Computing*: Often the parametrization and prediction of the model is achieved with the help of computers.
- Compare measurements and predictions to improve your model and/or observations.

- Today often *data-driven* "models" obtained with machine-learning are used. A disadvantage of these models is that they do not help us to *understand* how the observed system works.

Why Differential Equations?

- Differential equations are ubiquitous in science and technology
  - Solid mechanics: stability of bridges and buildings
  - Fluid mechanics: drag and lift of an airfoil
  - Material science: phase diagram of a substance
  - Protein folding: How do molecules bind
  - Mathematical biology: How does cancer grow
  - Hydrology: movement of contamination in groundwater
  - Weather and climate prediction
- Numerical methods are often the only way to solve these equations in practical situations
- Approach was enabled by the digital computer (although already envisioned by Leibniz 300 years ago!)
- Most of the time on supercomputers is spent solving differential equations

Historical Perspective

- Sir Isaac Newton (1642-1727) and Gottfried Wilhelm Leibniz (1646-1716) invent calculus
- Newton described motion of the planets by differential equations
- Leibniz developed also mechanical calculating machines
- Euler (1707-1783) and Lagrange (1736-1813) develop *variational calculus*
- Euler finds equations for inviscid fluid flow (1757)

Newton

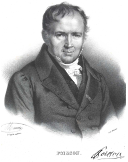
Leibniz

Euler

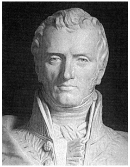
Lagrange

Equations of Mathematical Physics

- PDEs ubiquitous in physics
- E.g. to express conservation of mass, momentum and energy in quantitative form
- Poisson (electrostatics, gravity) $\sim$1800
- Navier-Stokes (viscous flow) 1822/1845
- Maxwell (electrodynamics) 1864
- Einstein (general relativity) 1915
- Schrödinger (quantum mechanics) 1926

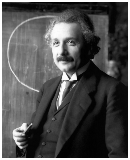


Poisson


Navier


Stokes


Maxwell


Einstein


Schrödinger

Calculating Machines and Computers

- 1671: Leibniz builds a machine to do $+, -, \cdot, /$
- 1831: Charles Babbage designs the steam-powered *Analytical Engine*, a programmable calculator including conditions and loops (not completed)
- 1921: Lewis Fry Richardson proposes to predict the weather based on differential equations using 64000 human computers
- 1937: Alan Turing founds *computability theory* based on the *Turing machine*
- World War II pushed the development of computers: firing tables (ENIAC 1943-46), deciphering codes (Colossus, 1943)
- 1945: John von Neumann proposes the stored-program computer (based on others' ideas)
- 1965: Gordon Moore predicts doubling of # transistors on a chip every 12 month (it is more like 18)
- Today: *You* will witness the end of Moore's law

Milestone Algorithms

- ca. 1670: Newton introduces a method to solve polynomial equations
- 1823: Carl Friedrich Gauß mentions an iterative method to solve (least squares) linear systems of equations arising in the triangulation of Hannover
- 1943: Richard Courant publishes an early version of the Finite Element Method based on the Ritz-Galerkin principle
- 1965: Cooley and Tukey publish the Fast Fourier Transform algorithm
- 1977: Wolfgang Hackbusch and Achi Brandt independently introduce the multigrid method
- 1987: Leslie Greengard and Vladimir Rokhlin Jr. introduce the fast multipole method
- 1999: Wolfgang Hackbusch introduces H-matrices allowing sparse approximation of non-sparse matrices

Sequences

- By $\{\xi_n : n \in \mathbb{N}\}$ we denote a *sequence* of real numbers
- Examples:

$$\xi_n = \frac{1}{n}, \qquad \xi_n = \frac{2n}{n+1}, \qquad \xi_{n+1} = \frac{1}{2}\left(\xi_n + \frac{a}{\xi_n}\right)$$

- A sequence $\{\xi_n\}$ *converges* to the limit $\xi \in \mathbb{R}$ if for any $\epsilon > 0$ there exists $N_\epsilon \in \mathbb{N}$ such that

$$|\xi - \xi_n| < \epsilon \qquad \text{for all } i \geq N_\epsilon$$

- A sequence $\{\xi_n\}$ *is a Cauchy sequence* if for any $\epsilon > 0$ there exists $N_\epsilon \in \mathbb{N}$ such that

$$|\xi_m - \xi_n| < \epsilon \qquad \text{for all } m, n \geq N_\epsilon$$

- Every convergent sequence is a Cauchy sequence
- Completeness *axiom*: Every Cauchy sequence converges in $\mathbb{R}$ (this is what makes $\mathbb{R}$ different from $\mathbb{Q}$)

Functions and Continuity

- Let $I = [a, b]$ be a closed interval (might also be open or half open) $y : I \to \mathbb{R}$ denotes a *function* y mapping

$$x \in I \to y(x) \in \mathbb{R}$$

- y is said to be *continuous in* x if

$$\{x_n\} \to x \quad \Rightarrow \quad \{y(x_n)\} \to y(x)$$

- Equivalently: for any $\epsilon > 0$ there exists $\delta_\epsilon(x) > 0$ such that

$$|y(x) - y(z)| < \epsilon \qquad \text{for all } 0 < |x - z| < \delta_\epsilon(x)$$

- If y is continuous in all $x \in I$ then u is a continuous function
- If δ_ϵ is independent of x then y is *uniformly* continuous
- The set $C(I)$ of all continuous functions on I is closed under addition and scalar multiplication, thus it forms a *vector space*
- For I closed y is bounded and $(C(I), \|.\|_\infty)$ is a normed vector space

Differentiable Functions I

- $y'(x)$ is called the *derivative of* y *in* x if for *all* sequences $\{x_n\}$ converging to x the limit
$$\lim_{x_n \to x} \frac{y(x) - y(x_n)}{x - x_n}$$
exists and is the same
- Clearly, continuity of y is a necessary condition for the derivative to exist
- Then both, nominator and denominator, converge to zero
- If y is differentiable in all $x \in I$, y is *differentiable* and the function y' with values $y'(x)$ is called the *derivative of* y
- Later we will use the equivalent notation $\frac{dy}{dx}(x) = y'(x)$
- We may write $y' = Dy$. D maps a differentiable function to its derivative

Differentiable Functions II

- And we may differentiate again: $y'' = Dy' = DDy = D^2 y$
- And in general $y^{(m)} = D^m y$ is the *m'th derivative of y*
- It is convenient to set $D^0 y = y$
- $D : \mathcal{D} \to \mathcal{R}$ maps a function to a function and is called "differentiation operator"
- What are its domain $\mathcal{D}$ and range $\mathcal{R}$?
- $\mathcal{D} = C^1(I) = \{y \in C(I) : y \textit{ is differentiable}\} \subseteq C(I)$
- $\mathcal{R} = C(I) = C^0(I)$
- In general we denote by $C^m(I)$ the space of m times continuously differentiable functions
- D is a linear operator:

$$D(y_1 + y_2) = Dy_1 + Dy_2, \quad D(cy) = cDy, c \in \mathbb{R}$$

What is a Differential Equation?

- A *differential equation* (DE) relates values of derivatives of one or more functions over a domain (here interval) $I \subseteq \mathbb{R}$
- An example is

$$\Psi(y'(x), y(x), x) = 0 \quad \text{for all } x \in I \tag{1}$$

- $\Psi : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a function in three arguments
- Only values at *the same point* x are related with each other
- Continuity, differentiability of y relate values at nearby points!
- (1) is a *first order ordinary DE in implicit form*
- Order: highest derivative occuring
- Ordinary DE (ODE): functions in *one* variable are involved
- Partial DE (PDE): functions in *several* variables are involved
- *Explicit form* would read: $y'(x) = f(x, y(x))$ for all $x \in I$

Fundamental Theorem of Calculus

- The simplest differential equation would be the following:

$$y'(x) = f(x) \quad \forall x \in I \tag{2}$$

- Assume $y(a)$ is known, then for any $b > a$ we get

$$\int_a^b y'(x)dx = y(b) - y(a) = \int_a^b f(x)dx \Leftrightarrow y(b) = y(a) + \int_a^b f(x)dx$$

- So, the function $y(x) = y(a) + \int_a^x f(\xi)d\xi$ solves (2)
- In fact, any function $y(x) + c$, $c \in \mathbb{R}$, is also a solution
- So, the solution of (2) is unique up to a constant
- A specific solution is picked by fixing $y(a)$ at some point a
- The *fundamental theorem of calculus* states that the function $\int_a^x f(\xi)d\xi$ is well defined and solves (2)

Two Simple Approximation Methods

We want to solve $y'(x) = f(x)$ in $I = (a, b]$, $y(a) = y_a$

- **Method 1**: For $N \in \mathbb{N}$ set $h = (b-a)/N$ and $x_n^h = a + nh$, $0 \leq i \leq N$. For h sufficiently small,

$$\frac{y(x_{n+1}^h) - y(x_n^h)}{h} \approx y'(x_n^h) = f(x_n^h)$$

  leading to the scheme $y_{n+1}^h = y_n^h + hf(x_n^h)$

- **Method 2**: A good approximation of the integral is the *trapezoidal rule*

$$\int_x^{x+h} f(\xi) d\xi \approx \frac{h}{2}(f(x) + f(x+h))$$

  leading to the scheme $y_{n+1}^h = y_n^h + \frac{h}{2}\left(f(x_n^h) + f(x_{n+1}^h)\right)$

Conceptual Model for Growth

- We wish to model growth of a population, e.g. bacteria in a petri dish over time
- From now on we will denote the independent variable by t because ODE models are often used to model functions of time
- In a *conceptual model* we list properties we consider (un-) important for the model
  - $N(t)$ is the number of individuals at time t
  - As a generalization, let $N(t)$ be a real number
  - We assume the spatial extend to be unimportant
  - Increase in population during an interval Δt is proportional to number $N(t)$ and Δt
  - This last assumption means that *infinite resources* (food, energy) are available for growth

Mathematical Growth Model

- Consider the *change* of the population in a time interval Δt:

$$N(t + \Delta t) = N(t) + \lambda \Delta t N(t) \tag{3}$$

  with a constant $\lambda \in \mathbb{R}$
- Observe similarity of (3) with the numerical method 1 above!
- For a fixed Δt this is a discrete model. For different Δt, different sequences of numbers are obtained
- Rearranging gives

$$\frac{N(t + \Delta t) - N(t)}{\Delta t} = \lambda N(t)$$

- Considering the limit $\Delta t \to 0$ we obtain a linear ordinary differential equation:

$$N'(t) = \lambda N(t) \tag{4}$$

Analytical Solution of Growth Model

- This simple ODE can be solved analytically
- Verify that $Ce^{\lambda t}$ for any $C \in \mathbb{R}$ is a solution
- The initial condition $N(t_0) = N_0$ picks the solution $N_0 e^{\lambda(t-t_0)}$
- Are *all* solutions of the form $Ce^{\lambda t}$?
- Let $N(t)$ be any solution of (4), then:

$$\left(N(t)e^{-\lambda t}\right)' = N'(t)e^{-\lambda t} - \lambda N(t)e^{-\lambda t} = \left(N'(t) - \lambda N(t)\right) e^{-\lambda t} = 0$$

- From $\left(N(t)e^{-\lambda t}\right)' = 0$ we conclude

$$N(t)e^{-\lambda t} = C \Leftrightarrow N(t) = Ce^{\lambda t}$$

- More realistic growth models consider finite resoures, e.g. *logistic growth*

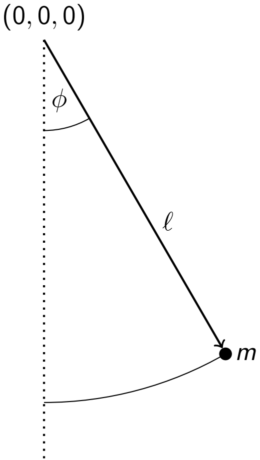
$$N'(t) = \lambda(G - N(t))N(t) \qquad \text{(Bernoulli DE)}$$

Conceptual Model of a Pendulum

We wish to model a pendulum
In a *conceptual model* we list properties we consider (un-) important for the model

- The weight is concentrated in a point of mass m
- The rod of length ℓ is assumed rigid and massless
- The rod is fixed at $(0, 0, 0)$ and movement is in the plane $y = 0$
- Air resistance is neglected

We develop a mathematical model based on Newton's equations of motion



Forces

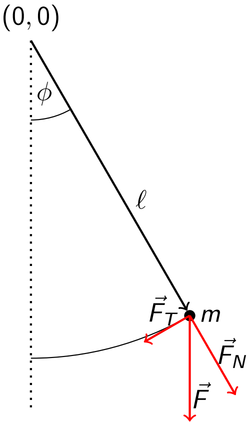
- Movement is along a circle, only force in *tangential direction* is relevant for acceleration
- Tangential force for deflection angle ϕ:

$$\vec{F}_T(\phi) = -\underbrace{m\,g}_{|\vec{F}|}\sin(\phi)\begin{pmatrix}\cos(\phi)\\ \sin(\phi)\end{pmatrix}$$

- For example $\phi = 0$, $\phi = \pi/2$:

$$\vec{F}_T(0) = mg\,0\begin{pmatrix}-1\\ 0\end{pmatrix},\ \vec{F}_T(\pi/2) = mg\begin{pmatrix}0\\ -1\end{pmatrix}$$

- Sign encodes direction



Distance, Velocity, Acceleration

- Distance $s(t)$, velocity $v(t)$, acceleration $a(t)$ satisfy:

$$v(t) = \frac{ds(t)}{dt}, \qquad a(t) = \frac{dv(t)}{dt}.$$

- The distance (including sign) satisfies $s(t) = \ell\phi(t)$.
- Therefore velocity satisfies

$$v(t) = \frac{d\, s(\phi(t))}{dt} = \frac{d\, \ell\phi(t)}{dt} = \ell\frac{d\phi(t)}{dt}$$

- and acceleration satisfies

$$a(t) = \frac{d\, v(\phi(t))}{dt} = \ell\frac{d^2\phi(t)}{dt^2}.$$

Equations of Motion

- Insert into Newton's second law $m\,a(t) = F(t)$ gives:

$$m\ell \frac{d^2\phi(t)}{dt^2} = -mg\sin(\phi(t)) \qquad \forall t > t_0.$$

- The force is scalar as we are only considering the distance travelled where sign encodes direction
- We obtain a second-order nonlinear ordinary differential equation for the deflection angle $\phi(t)$:

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\sin(\phi(t)) \qquad \forall t > t_0. \tag{5}$$

- A unique solution is determined by *two* initial conditions ($t_0 = 0$):

$$\phi(0) = \phi_0, \qquad \frac{d\phi}{dt}(0) = \phi_0'. \tag{6}$$

Solution for Small Deflection Angle

- For *small* deflection angle ϕ observe

$$\sin(\phi) \approx \phi,$$

e.g. $\sin(0.1) = 0.099833417$.

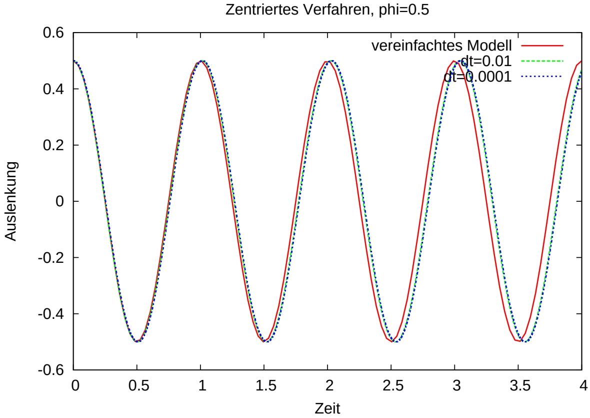
- Using this *approximation* yields the linear ODE

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\phi(t)$$

- Which is solved by $\phi(t) = A\cos(\omega t)$. The constants are fixed by the initial conditions $\phi(0) = \phi_0$, $\frac{d\phi}{dt}(0) = 0$, giving

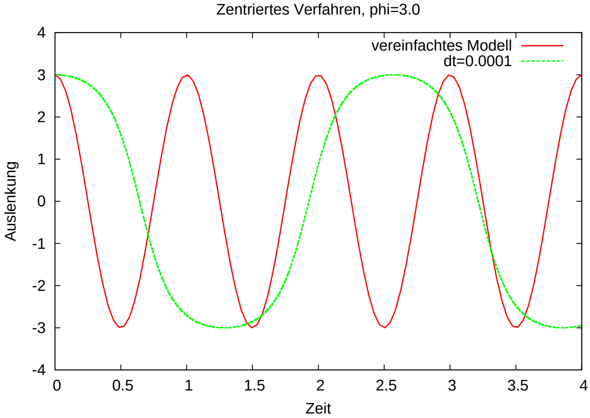
$$\phi(t) = \phi_0 \cos\left(\sqrt{\frac{g}{\ell}}t\right) \tag{7}$$

Model Comparison $\phi_0 = 0.5 \approx 28.6^\circ$



Even for 28.6° the approximation is quite accurate

Model Comparison $\phi_0 = 3.0 \approx 171^\circ$



For such large deformations the approximate model is very inaccurate

Elementary Reactions

- A reaction of three substances A, B, C is denoted by

$$\nu_a A + \nu_b B \quad \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \quad \nu_c C \tag{8}$$

- Forward reaction: ν_a molecules of A react with ν_b molecules of B to give ν_c molecules of C with reaction speed k_1
- The reverse reaction has speed k_2
- ν_a, ν_b, ν_c are the *stoichiometric coefficients*
- Reaction rates $k_j = A_j \exp(-E_j/(RT))$ given by *Arrhenius' law*
- *ODE system*: concentrations $c_i(t)$, $i = A, B, C$ in mol/m^3:

$$\frac{dc_C}{dt}(t) = \nu_c k_1 c_A^{\nu_a}(t) c_B^{\nu_b}(t) - \nu_c k_2 c_C^{\nu_c}(t)$$
$$\frac{dc_A}{dt}(t) = -\nu_a k_1 c_A^{\nu_a}(t) c_B^{\nu_b}(t) + \nu_a k_2 c_C^{\nu_c}(t)$$
$$\frac{dc_B}{dt}(t) = -\nu_b k_1 c_A^{\nu_a}(t) c_B^{\nu_b}(t) + \nu_b k_2 c_C^{\nu_c}(t)$$

- Equilibrium given by: $\frac{k_1}{k_2} = \frac{c_C^{\nu_c}}{c_A^{\nu_a} c_B^{\nu_b}}$ (*mass action law*)

Astrophysical N-body Problem

- Consider N bodies of mass m_i at positions $x_i(t) \in \mathbb{R}^3$
- The *gravitational force* $F_{ij} \in \mathbb{R}^3$ excerted from body j on body i is

$$F_{ij}(x_i, x_j) = G \frac{m_i m_j}{\|x_j - x_i\|^2} \frac{x_j - x_i}{\|x_j - x_i\|}$$

  where G is the *gravitational constant*
- Newton's 2nd law $F_i(t) = m_i a_i(t)$ gives ODE system:

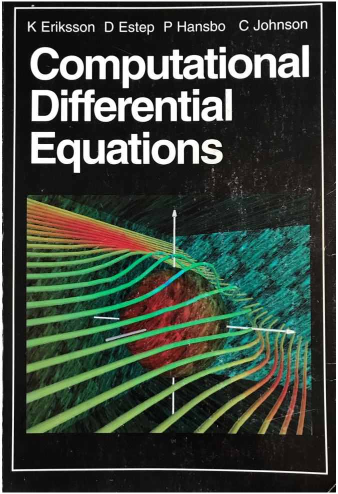
$$\frac{d^2 x_i(t)}{dt^2} = G \sum_{j=1, j\neq i}^{N} m_j \frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|^3} \qquad i = 1, \dots, N$$

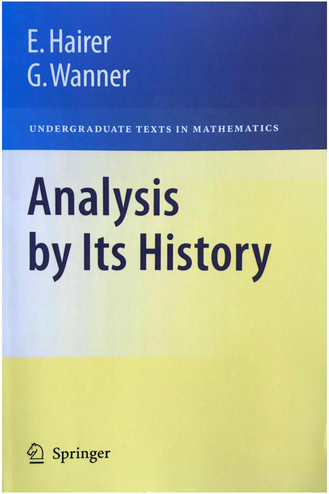
- Introducing velocity $v_i(t) = \frac{dx_i(t)}{dt}$ results in first-order system:

$$\frac{dx_i(t)}{dt} = v_i(t), \quad \frac{dv_i(t)}{dt} = G \sum_{j=1, j\neq i}^{N} m_j \frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|^3} \quad i = 1, \dots, N$$

- These are $6N$ coupled ODEs requiring $6N$ initial conditions

Recommended Reading





http://www.csc.kth.se/~jjan/private/cde.pdf

Summary Lecture 01

- Differential equations as part of the scientific method
- First glimpse on numerical solution
- Derivation of some differential equation models

Contents

Differential equations

We recall from lecture 01:

Definition 1 (Differential equation (DE))

A differential equation is a mathematical equation that relates a (unknown) function with its derivatives.

Differential equations can be split into two classes:

Definition 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. Often: either space x or time t.

Definition 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. Often: x and t or even (x, y, z) and t.

Classifications

- Order of a differential equation
- Single equations and DE systems
- Nonlinear problems:
    - Nonlinearity in the DE
    - The function set is not a vector space yielding a variational inequality
- Coupled problems and coupled DE systems.

Order of differential equations

- The **order** of a differential equation is determined by its highest derivatives
- First order ODE:

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

- Second order ODE:

$$y''(t) = f(y', y, t), \quad y(t_0) = y_0, y'(t_0) = v_0$$

- Higher order, here mth order:

$$y^{(m)}(t) = f(y^{(m-1)}, \dots, y, t)$$

plus m initial conditions.

Reduction of Higher-order Equations to First-order Systems

Higher-order DE can be reduced to low-order DE systems by introducing auxiliary solution functions.

- Given $y^{(m)}(t) = f(y^{(m-1)}, \ldots, y, t)$
- Define

$$y_1(t) = y(t)$$
$$\vdots$$
$$y_m(t) = y^{(m-1)}(t)$$

- This results in the first-order DE system:

$$y_1'(t) = y_2(t)$$
$$\vdots$$
$$y_{m-1}'(t) = y_m(t)$$
$$y_m'(t) = f(t, y_1, \ldots, y_m)$$

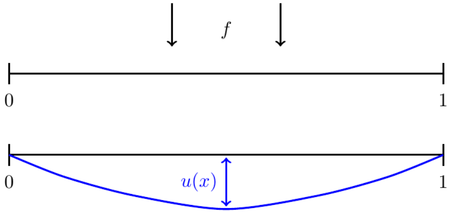
- With this, we can write the original problem in compact form (vector-valued problem!) as

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

Examples

1. $y'(t) = t^{-1}y(t)$: is of 1st order, linear with the solution $y(t) = t$
2. $y'(t) = ty(t)^{-1}$: 1st order nonlinear, singular solution for $y(t) \to 0$.
3. $y'(t) = y(t)^2$: 1st order nonlinear with a singularity at $t = 1$. Only local solution with $y(t) = \sqrt{1 + t^2}$
4. Clothesline problem (membrane deformation) on $\Omega = (0, 1)$: Find a deformation (displacements) u such that

$$-u''(x) = f \quad \text{and } u(0) = u(1) = 0.$$



Single equations vs. systems

Definition 4 (Single equation)

Let d be the dimension. A single DE consists of determining one solution variable, e.g.,

$$u : \Omega \subset \mathbb{R}^d \to \mathbb{R}.$$

Typical examples are Poisson's problem, the heat equation, wave equation, Monge-Ampère equation, Hamiliton-Jacobi equation, p-Laplacian.

Definition 5 (DE system)

A diff. eq. system determines a solution vector

$$u = (u_1, \ldots, u_d) : \Omega^d \to \mathbb{R}^d.$$

For each $u_i, i = 1, \ldots, d$, a DE must be solved. Inside these diff. equ. the solution variables may depend on each other or not. Typical examples of systems are predator-prey systems, linearized elasticity, nonlinear elasto-dynamics, Maxwell's equations.

Implicit differential equations

- So far: $y'(t) = f(y,t)$, which is an explicit representation of a DE
- However, not always, we can resolve with respect to the highest derivative:

$$F(y', y, t) = 0.$$

- Example:

$$F(t, y, y') = (y')^2 + uu' - 3(y')^5.$$

- Of course explicit forms of DE can always be written as implicit forms (e.g., model problem):

$$F(t, y, y') = y'(t) - f(y,t).$$

- Nonlinear numerical methods (fixed-point or Newton) required for the solution!

The model problem

- In general a model problem stands as a characteristic class of similar equations and which can exemplarely analyzed as a prototype problem!
- The model problem for ODEs is defined as:

Formulation 1

Given a model parameter $a \in \mathbb{R}$. Let $I := [t_0, t_0 + T]$ the time interval with the end time value $T > 0$. Find $y : I \to \mathbb{R}$ such that

$$y'(t) = ay(t), \quad y(t_0) = y_0, \quad t \geq t_0$$

The first term is the ODE. Here, $y' = \frac{dy}{dt}$. The second term is the so-called initial condition.

- Important **theoretical** concepts such as existence, uniqueness, stability are usually introduced in terms of this ODE.
- Moreover, important **numerical** concepts such as convergence order, efficiency, accuracy, stability are analyzed for this ODE as well.

Well-posedness: preliminaries

The concept of well-posedness is very general and in fact very simple:

Definition 6 (Hadamard 1923)

1. The problem under consideration has a solution;
2. This solution is unique;
3. 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.

Remark 1

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

Well-posedness: existence, uniqueness, stability

Definition 7 (Lipschitz condition)

Let $D := I \times \Omega \subset \mathbb{R} \times \mathbb{R}^d$. The function $f(t,y)$ on D is said to be (uniformly) Lipschitz continuous if for $L(t) > 0$ it holds

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

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

Theorem 8 (Picard-Lindelöf)

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

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

The proof can be found in classical textbooks on ordinary differential equations.

What is the model problem good for?

- Model problem is a simplified problem to predict growth of a population: human beings, animals, bacteria, virus (see also lecture 01)
- Example:

$$y' = (g-m)y, \quad y(t_0) = y_0$$

with growth g and mortalities rates m

- Exact solution (here possible):

$$y(t) = c\exp((g-m)(t-t_0)).$$

with

$$y(t_0) = \exp(C)\exp[(g-m)(t_0-t_0)] = \exp(C) = y_0 =: c.$$

What is the model problem good for?

- 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.
- Translating 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.
- Solution: construct a better mathematical model! For instance the logistic law (see e.g., M. Braun; Differential equations and their applications, Springer, 1993)

Solving differential equations: numerical discretization

- Analytical solutions are often too difficult or even impossible

$\rightarrow$ Why?

- Integrals are often not possible to be resolved analytically; if yes, it may take very long to write down a analytical solution per hand.
- An alternative could be do to experiments!

$\rightarrow$ However, they are often too expensive, too far away (moon, planets, ...), too smale (nanoscale)

- Three pillars of science:

$$\text{Experiments} \quad \leftrightarrow \quad \text{Scientific computing} \quad \leftrightarrow \quad \text{Theory}$$

Solving differential equations: numerical discretization

- **Numerical discretization**

→ Treat infinite-dimensional problems with the help of finite-dimensional discretizations (computers can only deal with **finite** numbers!)

- Simple example: cut numbers! For instance:
  $x = 3.1445645645608982345002034098430986\ldots$
  $\tilde{x} = 3.14456456456089$

→ Infinite number is x and finite number is $\tilde{x}$.

→ What is the error between $x - \tilde{x}$?

Solving differential equations: numerical discretization

- **Discretization parameter** often denoted by Δt (for temporal discretization) and h for spatial discretization.

$\rightarrow$ Represent a DE with finite numbers! And not infinite numbers!

$\rightarrow$ Allows to solve DE with a computer!

- Paradigm: Design numerical schemes in such a way that physical conservation properties (mass, momentum, energy, ...) are as much as possible conserved after the discretization!

Guiding questions for numerics of DE (1)

- What type of equation are we dealing with?
- What kind of discretization scheme shall we use?
- How do we design algorithms to compute discrete solutions $y^{\Delta t}$ (notation for ODEs) or u_h (notation for PDEs)?
- Can we proof that these algorithms really work?
- Are they robust (stable with respect to parameter variations), accurate, and efficient?
- Can we construct physics-based algorithms that maintain as best as possible conservation laws; in particular when several equations interact (couple)?

Guiding questions for numerics of DE (2)

- How far is u_h (resp. $y^{\Delta t}$) away from u (resp. y) in a certain (error) norm?
  Hint: comparing color figures gives a first impression, but is not science!
- 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?
- How can we achieve more efficient algorithms? Hint: adaptivity and/or parallel computing.
- How can we check that the solution is correct?

Errors: related to numerics and programming

1. The set of numbers is finite and a calculation is limited by machine precision (floating point arithmetics), which results in **round-off errors**. Typical issues are overflow and underflow of numbers.
2. The memory of a computer (or cluster) is finite and thus functions and equations can only be represented through approximations. Thus, continuous information has to be represented through discrete information, which results into the investigation of so-called **discretization errors**.
3. 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**. Mostly, these are so-called **iteration errors**, for instance the stopping criterion after how many steps an iterative method terminates.
4. Finally, **programming errors (code bugs)** are an important error source. Often these can be identified since the output is strange. But there are many, which are hidden and very tedious to detect.

Errors: related to modeling (will not be discussed in this school)

5. In order to make a 'quick guess' of a possible solution and to start the 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**.
6. **Data errors**: the data (e.g., input data, boundary conditions, parameters) are finally obtained from experimental data and may be inaccurate themselves.

Errors: final statements

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

Numerical concepts[1]

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, we 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. In order to judge whether a algorithm is fast or not we have to determine the order of convergence.

[1] Richter/Wick; Springer, 2017 (in german), english translation in `http://www.thomaswick.org/links/lecture_notes_Numerics_PDEs_Oct_12_2019.pdf`

Numerical concepts

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 the (unknown) exact solution by this numerical solution is approximated.* 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.

Numerical concepts

6. **Efficiency**: In general we can say, the higher the convergence order of an algorithm is, the more efficient our algorithm is. 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, whether 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 developments, 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. In practice non-robust or non-stable algorithms exhibit very often non-physical oscillations. For this reason, it is important to have a feeling about the physics whether oscillations in the solution are to be expected or if they are introduced by the numerical algorithm.

Physical oscillations versus numerical oscillations (instabilities)

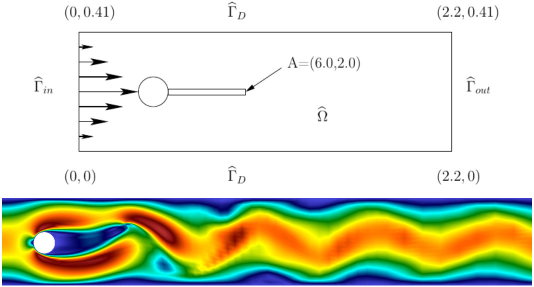
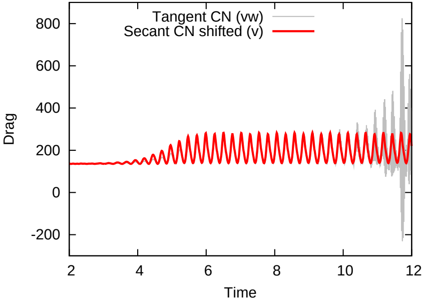


Figure: Fluid flow (Navier-Stokes) interacts with an elastic beam. Due to a non-symmetry of the cylinder, the beam starts oscillating. These oscillations are physical!

Physical oscillations versus numerical oscillations (instabilities)

- Observe the tip of the elastic beam!

$\rightarrow$ Physical oscillations! Shown in red color for a 'good' numerical scheme.



- The grey numerical scheme exhibits at some time around $t \approx 10$ micro-oscillations which are due to numerical instabilites. Finally the grey numerical scheme has a blow-up and yields garbage solutions.

Summary lecture 02

- DE (differential equations) - classifications, examples
- General numerical concepts
- One goal of this spring school is to learn numerical techniques and corresponding programming in order to analyze and implement DE.

Exercise 1 Overview

In this exercise we explore the pendulum in more detail by

- Recapitulating the derivation of the full model and the simplified model
- Deriving two numerical methods for its solution
- Implementing these methods in your own C++
- Evaluating these methods by
  - Looking at stability,
  - Discretization error and
  - Modeling error

Task 1

- Recapitulate the model for the pendulum

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\sin(\phi(t)) \qquad \forall t > t_0.$$

with the two initial conditions

$$\phi(0) = \phi_0, \qquad \frac{d\phi}{dt}(0) = \phi_0'.$$

- For *small* deflection angle ϕ derive the *approximation*

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\phi(t)$$

- Show that it has the general solution $\phi(t) = A\cos(\omega t)$ and determine the constants A, ω from the initial conditions

Full Model, Method 1

- In the first method, begin by rewriting the second order ODE as a first order system

$$\frac{d\phi(t)}{dt} = u(t), \qquad \frac{d^2\phi(t)}{dt^2} = \frac{du(t)}{dt} = -\frac{g}{\ell}\sin(\phi(t)).$$

- Replacing derivatives by difference quotients

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} \approx \frac{d\phi(t)}{dt} = u(t),$$
$$\frac{u(t+\Delta t) - u(t)}{\Delta t} \approx \frac{du(t)}{dt} = -\frac{g}{\ell}\sin(\phi(t)).$$

- yields the *one step* scheme

$$\phi_{n+1} = \phi_n + \Delta t\, u_n \qquad\qquad \phi_0 = \phi(t_0)$$
$$u_{n+1} = u_n - \Delta t\,(g/\ell)\sin(\phi_n) \qquad\qquad u_0 = \phi'(t_0)$$

Where ϕ_n approximates $\phi(n\Delta t)$ for a chosen Δt Rekursion (*Euler*):

Full Model, Method 2

- Now we derive a method that directly approximates the second-order ODE
- It uses a *central difference quotient* for the second derivative

$$\frac{\phi(t+\Delta t) - 2\phi(t) + \phi(t-\Delta t)}{\Delta t^2} \approx \frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\sin(\phi(t)).$$

- Solving for $\phi(t+\Delta t)$ yields the *two step* scheme ($n \geq 2$):

$$\phi_{n+1} = 2\phi_n - \phi_{n-1} - \Delta t^2\,(g/\ell)\,\sin(\phi^n) \tag{9}$$

with the initial condition

$$\phi_0 = \phi(t_0), \qquad \phi_1 = \phi(t_0) + \Delta t\,\phi'(t_0). \tag{10}$$

The starting value ϕ_1 is derived with method 1

Task 2

- Write a C++ program implementing schemes 1 and 2 using a time step Δt that can be entered by the user
- Write the results to a file, where every line contains

$$t_n \qquad \phi_n \qquad u_n$$

- you can visualize the results using `gnuplot` as follows
  `plot "filename" u 1:2`
  where the x-axis uses the first column and the y-axis uses the second column
- You may start from the file `eemodelproblem.cc` available on the cloud `https://cloud.ifam.uni-hannover.de/index.php/s/Cwe4ZqwLRMixS3J`. It solves the problem $u' = \lambda u$ using `hdnum`
- Download the file and put it in the directory `hdnum/examples/num1`. Compile it with
  `g++ -o eemodelproblem -I../.. eemodelproblem.cc`

Task 3: Comparisons

- For method 1: choose an initial deflection angle $\phi_0 = 0.1$ and a time step $\Delta = 0.1$ and compute the solution up to time 4.0. What do you observe?
- Repeat the experiment with successively smaller time steps, say 0.01, 0.001, 0.0001. What do you observe?
- Try to compute the solution for longer times with the small timesteps. What happens?
- Repeat the same experiments with method 2. Is there a difference?
- Compare the solution of the full model and the reduced model for different initial angles $\phi_0 = 0.1, 0.5, 3.0$. Use your favourite method and a timestep Δt that is small enough to avoid any visibly numerical error.
- Recapitulate the concepts stability, discretization error and modeling error in the light of the results of exercise 1.

Contents

Problem Setting

- We consider first-order systems of ODEs in explicit form

$$y'(t) = f(t, y(t)), \qquad t \in (t_0, t_0 + T], \qquad y(t_0) = u_0 \tag{11}$$

  to determine the unknown function $u : [t_0, t_0 + T] \to \mathbb{R}^d$
- In components this reads:

$$\begin{pmatrix} y_1'(t) \\ \vdots \\ y_d'(t) \end{pmatrix} = \begin{pmatrix} f_1(t, y_1(t), \ldots, y_d(t)) \\ \vdots \\ f_d(t, y_1(t), \ldots, y_d(t)) \end{pmatrix}$$

- The right hand side $f : [t_0, t_0 + T] \times \mathbb{R}^d \to \mathbb{R}^d$ is Lipschitz-continuous

$$\|f(t, y) - f(t, w)\| \leq L(t)\|y - w\|$$

- Thus, the system has a unique solution

Taylor's Theorem

An important tool in the derivation and analysis of numerical methods for ODEs is the following theorem

Theorem 9 (Taylor's Theorem with Lagrangian Remainder)

Let $u : I \to \mathbb{R}$ be $(m+1)$-times continuously differentiable. Then, for $t, t+\Delta t \in I$, it holds

$$y(t+\Delta t) = \sum_{k=0}^{m} \frac{y^{(k)}(t)}{k!}\Delta t^k + \frac{y^{(m+1)}(t+\theta\Delta t)}{(m+1)!}\Delta t^{m+1} \quad \theta \in [0,1].$$

As a consequence of Taylor's theorem we have for $m = 1$

$$y(t+\Delta t) = y(t) + y'(t)\Delta t + \frac{y''(t+\xi)}{2}\Delta t^2, \quad 0 \le \xi \le \Delta t$$

Taken component-wise this holds also for vector-valued y

Explicit Euler Method

- Choose N time steps

$$t_0 < t_1 < t_2 < \ldots < t_{N-1} < t_N = t_0 + T, \qquad \Delta t_n = t_{n+1} - t_n$$

- $y_n^{\Delta t}$ denotes the approximation of $y(t_n)$ *computed with step size* Δt
- Take Taylor, use ODE and omit error term to obtain the *explicit Euler approximation*

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \Delta t_n y'(t_n) + \frac{y''(t_n + \xi_n)}{2}\Delta t_n^2 \\ &= y(t_n) + \Delta t_n f'(t_n, y(t_n)) + \frac{y''(t_n + \xi_n)}{2}\Delta t_n^2 \\ \Rightarrow \quad y_{n+1}^{\Delta t} &= y_n^{\Delta t} + \Delta t_n f(t_n, y_n^{\Delta t}) \end{aligned}$$

- Assuming $y_n^{\Delta t} = y(t_n)$ and subtracting we obtain

$$y(t_{n+1}) - y_{n+1}^{\Delta t} = \frac{y''(t_n + \xi_n)}{2}\Delta t_n^2$$

- *The error after one step is* $O(\Delta t^2)$, *how does it propagate?*

Implicit Euler Method

- Using Taylor's theorem slightly differently gives

$$\begin{aligned} y(t_n) = y(t_{n+1} - \Delta t_n) &= y(t_{n+1}) - \Delta t_n y'(t_{n+1}) + \Delta t_n^2 \frac{y''(t_{n+1} - \xi_n)}{2} \\ &= y(t_{n+1}) - \Delta t_n f(t_{n+1}, y(t_{n+1})) + \Delta t_n^2 \frac{y''(t_{n+1} - \xi_n)}{2} \\ \Leftrightarrow y(t_{n+1}) - \Delta t_n f(t_{n+1}, y(t_{n+1})) &= y(t_n) - \Delta t_n^2 \frac{y''(t_{n+1} - \xi_n)}{2} \end{aligned}$$

- Which yields the *implicit Euler approximation*

$$y_{n+1}^{\Delta t} - \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t}) = y_n^{\Delta t} \tag{12}$$

- Need to solve a *nonlinear algebraic equation* to obtain $y_{n+1}^{\Delta t}$ which is computationally much more demanding!
- *Is it worth the effort?*

Local Error in Implicit Euler Method

- We can modify the analysis of the explicit scheme
- From the construction of the scheme we obtain

$$y(t_{n+1}) = y(t_n) + \Delta t_n f(t_{n+1}, y(t_{n+1})) - \Delta t_n^2 \frac{y''(t_{n+1} - \xi_n)}{2}$$

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \Delta t_n f(t_n, y_{n+1}^{\Delta t})$$

- Subtracting, taking norms and using L-continuity gives

$$y(t_{n+1}) - y_{n+1}^{\Delta t} = y(t_n) - y_n^{\Delta t} + \Delta t_n [f(t_{n+1}, y(t_{n+1})) - f(t_n, y_{n+1}^{\Delta t})] - \Delta t_n^2 \frac{y''(t_{n+1} - \xi_n)}{2}$$

$$\|y(t_{n+1}) - y_{n+1}^{\Delta t}\| \leq \|y(t_n) - y_n^{\Delta t}\| + \Delta t_n L(t_{n+1}) \|y(t_{n+1}) - y_{n+1}^{\Delta t}\| + \frac{\Delta t_n^2}{2} \|y''(t_{n+1} - \xi_n)\|$$

$$\|y(t_{n+1}) - y_{n+1}^{\Delta t}\| \leq \frac{1}{1 - \Delta t_n L(t_{n+1})} \left[\|y(t_n) - y_n^{\Delta t}\| + \frac{\Delta t_n^2}{2} \|y''(t_{n+1} - \xi_n)\| \right]$$

- For $\Delta t < 1/L$ the error after one step is also $O(\Delta t^2)$
- This time step restriction is actually superficial

Implicit Trapezoidal Rule

- Another approach follows from integrating the ODE with the trapezoidal rule

$$y(t_{n+1})-y(t_n) = \int_{t_n}^{t_{n+1}} f(\xi, y(\xi))d\xi = \frac{\Delta t_n}{2}[f(t_n, y(t_n))+f(t_{n+1}, y(t_{n+1}))]+O(\Delta t_n^3)$$

- Resulting in the *implicit trapezoidal rule*

$$\begin{aligned} y_{n+1}^{\Delta t} - y_n^{\Delta t} &= \frac{\Delta t_n}{2}[f(t_n, y_n^{\Delta t}) + f(t_{n+1}, y_{n+1}^{\Delta t})] \\ \Leftrightarrow \quad y_{n+1}^{\Delta t} - \frac{\Delta t_n}{2} f(t_{n+1}, y_{n+1}^{\Delta t}) &= y_n^{\Delta t} + \frac{\Delta t_n}{2} f(t_n, y_n^{\Delta t}) \end{aligned}$$

- which has an error $O(\Delta t_n^3)$ after one step
- The computational effort is the same as for the implicit Euler method but it is more accurate

$\Rightarrow$ How to solve algebraic systems efficiently?
$\Rightarrow$ How to construct methods with high accuracy systematically?

Fixed Point Iteration in Implicit Methods

- In implicit Euler we need to solve

$$y_{n+1}^{\Delta t} - \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t}) = y_n^{\Delta t}$$

- Consider the following iteration

$$y_{n+1}^{\Delta t, k+1} = y_n^{\Delta t} + \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t,k}) = g(y_{n+1}^{\Delta t,k})$$

  and observe

$$\|g(y) - g(w)\| = \|y_n^{\Delta t} + \Delta t_n f(t_{n+1}, y) - y_n^{\Delta t} - \Delta t_n f(t_{n+1}, w)\| \leq \Delta t_n L \|y - w\|$$

- Then, according to the Banach fixed point theorem this iteration converges to the unique solution when $q = \Delta t_n L < 1$, resulting in the time step constraint $\Delta t_n < 1/L$
- Consider the linear, autonomous ODE $y' = f(t, y) = Ay$, then $L = \|A\|$ which might be large

Newton Iteration in Implicit Methods

- We rewrite the implicit Euler scheme as

$$F(y_{n+1}^{\Delta t}) = y_{n+1}^{\Delta t} - \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t}) - y_n^{\Delta t} = 0$$

- Newton's method is based on Taylor expansion of f:

$$\begin{aligned} F(y_{n+1}^{\Delta t,k+1}) &= F(y_{n+1}^{\Delta t,k} + \Delta y) = y_{n+1}^{\Delta t,k} + \Delta y - \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t,k} + \Delta y) - y_n^{\Delta t} \\ &\approx y_{n+1}^{\Delta t,k} + \Delta y - \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t,k}) - \Delta t_n \nabla f(t_{n+1}, y_{n+1}^{\Delta t,k})\Delta y - y_n^{\Delta t} = 0 \\ &\Rightarrow (I - \Delta t_n \nabla f(t_{n+1}, y_{n+1}^{\Delta t,k}))\Delta y = y_n^{\Delta t} - y_{n+1}^{\Delta t,k} + \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t,k}) \end{aligned}$$

- For the update Δy a linear system needs to be solved
- Newton's method for computing $y_{n+1}^{\Delta t}$ reads

$$y_{n+1}^{\Delta t,k+1} = y_{n+1}^{\Delta t,k} + (I - \Delta t_n \nabla f(t_{n+1}, y_{n+1}^{\Delta t,k}))^{-1}(y_n^{\Delta t} - y_{n+1}^{\Delta t,k} + \Delta t_n f(t_{n+1}, y_{n+1}^{\Delta t,k}))$$

- For $y' = f(t, y) = Ay$ this would converge in one iteration without a time step restriction

Discussion of Solution Methods

- Fixed point iteration requires a time step restriction $\Delta t_n \leq q/L$ to obtain convergence factor $q < 1$
- It converges from any initial guess (global convergence)
- **But**: implicit methods are typically used to **avoid** time step restrictions
- Newton's method can often handle much larger time steps
- Its convergence is guaranteed if the initial guess is close enough to the solution (local convergence)
- It can require a globalization strategy such as line search
- Combination of both methods is possible

One Step Methods

- All methods discussed so far are called *one step methods* as they are computing an approximation at t_{n+1} from one at t_n
- The *general one step method* has the form

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \Delta t_n F(\Delta t_n, t_n, y_n^{\Delta t}, y_{n+1}^{\Delta t})$$

- All methods discussed so far can be put in this form
- **Question**: How can we *systematically* construct methods where the error after one step is of the form $O(\Delta t^{p+1})$
- p is called the *order* of the method (not to be confused with the order of the ODE)
- Explicit and implicit Euler are first order methods ($p = 1$)

Taylor Method

- Recall the Taylor expansion:

$$y(t+\Delta t) = \sum_{k=0}^{m} \frac{y^{(k)}(t)}{k!}\Delta t^k + \frac{y^{(m+1)}(\xi)}{(m+1)!}\Delta t^{m+1} \quad \xi \in [t, t+\Delta t]$$

- Differentiating the differential equation $k-1$ times yields:

$$y^{(k)}(t) = \frac{d^{k-1}}{dt^{k-1}} f(t, y(t)) =: f^{(k-1)}(t, y(t)) \qquad (k \geq 1)$$

- The n-step Taylor method reads

$$y(t+\Delta t) = y(t) + \sum_{k=1}^{m} \frac{\Delta t^k}{k!} f^{(k-1)}(t, y(t)) + \frac{\Delta t^{m+1}}{(m+1)!} y^{(m+1)}(\xi)$$

- Omitting the remainder term yields the numerical method

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \sum_{k=1}^{m} \frac{\Delta t^k}{k!} f^{(k-1)}(t_n, y_n^{\Delta t})$$

Why the Taylor Method is Impractical

- We need to compute $f^{(0)}(t_n, y_n^{\Delta t}), f^{(1)}(t_n, y_n^{\Delta t}), f^{(2)}(t_n, y_n^{\Delta t}), \ldots$
- Ok, $k = 1$ is easy:

$$f^{(0)}(t_n, y_n^{\Delta t}) = f(t_n, y_n^{\Delta t})$$

- $k = 2$

$$f_r^{(1)}(t, y(t)) = \frac{d}{dt} f_r(t, y(t)) = \frac{\partial f_r}{\partial t}(t, y(t)) + \sum_{s=1}^{d} \frac{\partial f_r}{\partial y_s}(t, y(t)) \frac{dy_s}{dt}(t)$$

$$f^{(1)}(t, y(t)) = f'(t, y(t)) + \nabla_y f(t, y(t)) f(t, y(t))$$

- $(\nabla_y f(t, y(t)))_{r,s} = \frac{\partial f_r}{\partial y_s}(t, y(t))$ is the Jacobian of f
- We need to compute derivatives of the function f in $d + 1$ variables, i.ee. $d(d + 1)$ derivatives
- For $k = 3$, 2nd derivatives of f need to be computed, together with complicated expressions!

Order and a Way Out

Assuming $y_n^{\Delta t} = y(t_n)$ and subtracting yields:

$$
\begin{aligned}
y(t_n + \Delta t) - y_{n+1}^{\Delta t} = y(t_n) &+ \sum_{k=1}^{m} \frac{\Delta t_n^k}{k!} f^{(k-1)}(t_n, y(t_n)) + \frac{\Delta t_n^{m+1}}{(m+1)!} y^{(m+1)}(\xi) \\
&- y_n^{\Delta t} - \sum_{k=1}^{m} \frac{\Delta t^k}{k!} f^{(k-1)}(t_n, y_n^{\Delta t}) \\
&= \frac{\Delta t_n^{m+1}}{(m+1)!} y^{(m+1)}(\xi)
\end{aligned}
$$

Thus, Taylor's method has the order $p = n$

- **Idea:** It suffices to *approximate* $f^{(k)}(t, y_n^{\Delta t})$ in such a way that the error is at least Δt_n^{m+1}
- This leads to the class of (explicit) **Runge[2]-Kutta[3]** methods

[2]Carl Runge, dt. Mathematiker, 1856-1927, Prof. in Hannover and Göttingen
[3]Wilhelm Kutta,dt. Mathematiker, 1867-1944, Prof. in Aachen and Stuttgart

Example: A second-order Method

- Consider $m = 2$ and recall our method:

$$y(t + \Delta t) = y(t) + \Delta t f(t, y(t)) + \frac{\Delta t^2}{2} f^{(1)}(t, y(t)) + O(\Delta t^3)$$

- If we approximate (assuming $d = 1$!) by using Taylor 2 times:

$$\begin{aligned}
f^{(1)}(t, y(t)) &= \frac{1}{\Delta t}\left[f(t + \Delta t, y(t + \Delta t)) - f(t, y(t))\right] + O(\Delta t) \\
&= \frac{1}{\Delta t}\left[f(t + \Delta t, y(t) + \Delta t f(t, y(t)) + O(\Delta t^2))) - f(t, y(t))\right] + O(\Delta t) \\
&= \frac{1}{\Delta t}\left[f(t + \Delta t, y(t) + \Delta t f(t, y(t))) - f(t, y(t))\right] + O(\Delta t)
\end{aligned}$$

- we get

$$y(t + \Delta t) = y(t) + \frac{\Delta t}{2} f(t, y(t)) + \frac{\Delta t}{2} f(t + \Delta t, y(t) + \Delta t f(t, y(t))) + O(\Delta t^3)$$

- and from that a 2nd order method called *Heun's method*

Explicit Runge-Kutta Methods

- This suggests the following class of methods:

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \Delta t_n \left(b_1 k_1 + \ldots + b_s k_s \right) \quad \text{with}$$

$$k_1 = f(t_n, y_n^{\Delta t}), \qquad k_r = f\left(t_n + c_r \Delta t_n, y_n^{\Delta t} + \Delta t_n \sum_{j=1}^{r-1} a_{rj} k_j \right), \quad r > 1.$$

- s is the *number of stages*
- The k_r can be computed recursively (explicit method)
- The coefficients are collected in a *Butcher tableau*:

$$\begin{array}{c|cccc} 0 & 0 & & \cdots & 0 \\ c_2 & a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & \cdots & b_{s-1} & b_s \end{array} \qquad = \qquad \begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

Systematic Construction of Runge-Kutta Methods I

- Basic idea is comparison of coefficients
- Consider $s = 1$
- Then the method reads

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \Delta t_n b_1 f(t_n, y_n^{\Delta t})$$

  and there is only one parameter to be determined
- From Taylor's expansion we know

$$y(t_{n+1}) = y(t_n) + \Delta t_n f(t_n, y(t_n)) + O(\Delta t_n^2)$$

- By comparison of coefficients we obtain $b_1 = 1$
- *Explicit Euler* is the only RK method of order 1

Systematic Construction of Runge-Kutta Methods II

- For $s = 2$ one obtains (use Taylor expansion)

$$y_{n+1}^{\Delta t} = y_n^{\Delta t} + \Delta t_n \left[(b_1 + b_2) f + \Delta t_n b_2 c_2 \frac{\partial f}{\partial t} + \Delta t_n b_2 a_{21} \frac{\partial f}{\partial y} f \right] (t_n, y_n^{\Delta t}) + O(\Delta$$

$$y(t_{n+1}) = y(t_n) + \Delta t_n \left[f + \frac{\Delta t_n}{2} \frac{\partial f}{\partial t} + \frac{\Delta t_n}{2} \frac{\partial f}{\partial y} f \right] (t_n, y(t_n)) + O(\Delta t_n^3)$$

- By comparison of coefficients we obtain the three conditions

$$b_1 + b_2 = 1 \qquad b_2 c_2 = \frac{1}{2} \qquad b_2 a_{21} = \frac{1}{2}$$

for four unknown coefficients
- Need to solve *under-determined* nonlinear algebraic system
- Two solutions are

$$\begin{array}{c|cc} 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad \text{(Heun)} \qquad \begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} \quad \text{(modified Euler)}$$

Remarks About (Explicit) Runge-Kutta Methods

- With $s = 2$ one can achieve at most order 2
- For $s \leq 4$ the order is $p = s$, for larger s one has $p < s$
- This is true for the scalar case $d = 1$, RK methods can be extended for systems $d > 1$ but the order is in general lower than for scalar equations
- The nonlinear conditions become *very complicated* for larger s. So-called Butcher trees allow a systematic representation of the required derivatives
- Alternatively computer algebra systems are used
- For a given maximum achievable order the number of equations is usually smaller than the number of parameters. This allows for optimization of additional properties, e.g. stability.

Some Example Methods

- $s = 3$, 8 Parameters, 6 conditions

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{3} & & \\ \frac{2}{3} & 0 & \frac{2}{3} & \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array}$$

Heun's 3rd order method

- $s = 4$, 13 Parameters, 11 conditions

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

THE Runge-Kutta method (order 4)

Other Methods

- Implicit Runge-Kutta Methods
  - Diagonally implicit Runge-Kutta Methods: A is lower triangular. Need to solve s nonlinear systems of dimension d
  - Fully implicit Runge-Kutta Methods: A is full. Need to solve on nonlinear system of size $s \cdot d$
  - These methods may have very good stability properties and high order (e.g. $p = 2s$ for Gauß' method)
- Linear Multistep Methods
  - Use several previous values $y_n^{\Delta t}, y_{i-1}^{\Delta t}, \ldots, y_{i-r}^{\Delta t}$ to compute $y_{n+1}^{\Delta t}$
  - May be very efficient in terms of f evaluations for a given order
  - Explicit and implicit variants
- Galerkin's method
  - Approximate solution u in finite-dimensional function space, e.g. (trigonometric) polynomials
  - Use variational principle to determine the approximation
  - Good stability properties, a-posteriori error estimates
- Error control and choice of time step Δt

Contents

Problem statement

Recall:

Formulation 2 (Initial value problem - IVP)
Find a differentiable function $y(t)$ for $0 \leq t < T < \infty$ such that

$$\begin{aligned} y'(t) &= f(t, y(t)), \\ y(0) &= y_0. \end{aligned}$$

We recall from yesterday, the ODE model problem:

$$y' = ay, \quad y(0) = y_0, \quad , a \in \mathbb{R}. \tag{13}$$

This ODE has the (unique) solution:

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

Stiff problems

Definition 10

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

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

Here, $Re(\cdot)$ denotes the real part of a complex number.

Remark 2

Stiff problems arise often and in particular in combination with time-dependent PDEs. Stiff problems require very well designed numerical algorithms (in terms of stability) as we will see in this lecture.

Stiff problems: Examples

- For $y' = ay$, the eigenvalue corresponds to a. This means: $\lambda = a$. For big (negative) a we therefore need to be a bit careful with the design of our numerical schemes.
- Remark: scalar problems cannot really be called stiff when $-a \gg 1$. Here, we already need to use small time step sizes for minimizing the local discretization error.
- As second example, we consider now a system; namely

$$u'(t) = Au(t)$$

with $u(0) = (1, 0, -1)^T$ and

$$A = \begin{pmatrix} -21 & 19 & -20 \\ 19 & -21 & 20 \\ 40 & -40 & -40 \end{pmatrix}$$

- The matrix A does not look too horrible on the first view.

Stiff problems: Examples (cont'd)

- Our definition tells us that we have to compute $f'_u(t,u)$. What is this here? Well,

$$f(t,u) = Au(t) \quad \Rightarrow \quad f'_u(t,u) = A.$$

- The eigenvalues of $f'_u(t,u) = A$ are $\lambda_1 = -2$ and $\lambda_{2,3} = -40 \pm 40i$.
- Recall that $i := \sqrt{-1}$, the imaginary part of a complex number.
- The negative real parts are

$$Re(\lambda_1) = -2, \quad Re(\lambda_{2,3}) = -40$$

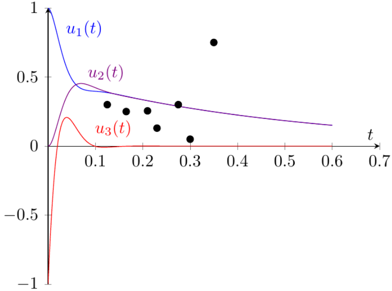
- The stiffness ratio is

$$\kappa(t) = \frac{|-40|}{|-2|} = 20 \gg 1.$$

- Consequently, we consider this as a stiff problem.

Stiff problems: Examples (cont'd)

The solution components evolve as follows:



What do we observe?

- We observe that at the beginning we see (physical!) variations
- For $t \geq 0.1$, we observe $u_1 \approx u_2$ and $u_3 \to 0$
- In black dots, an unstable numerical method is used
- Why this occurs and how this can be repaired, we study in the following

Numerical analysis: preliminaries

- In the previous sections, we have constructed algorithms that yield a sequence of discrete solutions $\{(y^{\Delta t})_k\}_{k \in \mathbb{N}}$.
- Specifically, in lecture 03, we have seen the first steps how the convergence order can be detected.
- In the numerical analysis our goal is to derive a convergence result of the form
$$\|y_n^{\Delta t} - y(t_n)\| \leq Ck^{\alpha}$$
where α is the order of the scheme.
- This result will tell us that the discrete solution $y^{\Delta t}$ really approximates the exact solution y and if we come closer to the exact solution at which rate we come closer.

Numerical analysis: splitting into stability and consistency

- We consider the simplest scheme; namely the forward Euler method
- For the model problem with $f(t, y) = \lambda y$, it holds:

$$\frac{y_{n+1}^{\Delta t} - y_n^{\Delta t}}{\Delta t} = \lambda y_n^{\Delta t}$$

  with $\Delta t = t_{n+1} - t_n$.
- This yields

$$y_{n+1}^{\Delta t} = (1 + \Delta t \lambda) y_n^{\Delta t} = B_E y_n^{\Delta t},$$

  with $B_E := (1 + \Delta t \lambda)$.
- Let us write the error at each time point t_n as:

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

Numerical analysis: splitting into stability and consistency

It holds:

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

Numerical analysis: splitting into stability and consistency

Therefore, the error can be split into two parts:

Definition 11 (Error splitting of the model problem)

The error at step n can be decomposed as

$$e_n := \underbrace{B_E e_{n-1}}_{\textit{Stability}} - \underbrace{\Delta t \eta_{n-1}}_{\textit{Consistency}} . \tag{14}$$

The first term, namely the stability, provides an idea how the previous error e_{n-1} is propagated from t_{n-1} to t_n. The second term η_{n-1} is the so-called truncation error (or local discretization error), which arises because the exact solution does not satisfy the numerical scheme and represents the consistency of the numerical scheme. Moreover, η_{n-1} yields the speed of convergence of the numerical scheme.

Numerical analysis: stability

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 a bounded discrete solution is computed when the continuous solution is bounded.

Numerical analysis: stability

Definition 12 ((Absolute) stability)

A (one-step) method is absolute stable for $\lambda\Delta t \neq 0$ if its application to the model problem produces in the case $Re\,\lambda \leq 0$ a sequence of bounded discrete solutions: $\sup_{n\geq 0} |y_n^{\Delta t}| < \infty$. To find the stability region, we work with the stability function $R(z)$ where $z = \lambda\Delta t$. We define:

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

Example 3

For the forward Euler scheme from before, we simply have: $B(z) := B_E$.

Remark 4

Nonstability often exhibits non-physical oscillations in the discrete solution. For this reason, it is important to have a feeling for the physics in order to decide whether oscillations are physically-wanted or numerical instabilities. See also Exercise 1 (day 1) and the end of lecture 02.

Numerical analysis: stability

Proposition 5

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

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

Numerical analysis: stability

Proof.

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

$$\frac{y_n^{\Delta t} - y_{n-1}^{\Delta t}}{\Delta t} = \lambda y_{n-1}^{\Delta t} \quad \Rightarrow y_n^{\Delta t} = (y_{n-1}^{\Delta t} + \lambda \Delta t) y_{n-1}^{\Delta t} \tag{15}$$

$$= (1 + \lambda \Delta t) y_{n-1}^{\Delta t} = (1 + z) y_{n-1}^{\Delta t} \tag{16}$$

$$= R(z) y_{n-1}^{\Delta t}. \tag{17}$$

For the (implicit) backward Euler method we obtain:

$$\frac{y_n^{\Delta t} - y_{n-1}^{\Delta t}}{\Delta t} = \lambda y_n^{\Delta t} \quad \Rightarrow y_n^{\Delta t} = (y_{n-1}^{\Delta t} + \lambda \Delta t) y_n^{\Delta t} \tag{18}$$

$$\Rightarrow y_n^{\Delta t} = \frac{1}{1 - a\Delta t} y_n^{\Delta t} \Rightarrow y_n^{\Delta t} = \underbrace{\frac{1}{1 - z}}_{=:R(z)} y_n^{\Delta t}. \tag{19}$$

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

Stability domains of forward Euler, backward Euler, and the trapezoidal rule

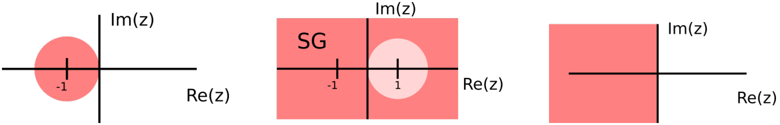


Figure: Stability domains (SG) of the forward Euler scheme, backward Euler scheme and the trapezoidul rule.

Numerical analysis: A-stability

Definition 13 (A-stability)

A difference method is A-stable if its stability region is part of the absolute stability region:

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

Again, *Re* denotes the real part of the complex number z. A brief introduction to complex numbers can be found in any calculus lecture.

Numerical analysis: A-stability, alternative definition

Definition 14 (A-stability)

Let $\{(y^{\Delta t})_k\}_{k\in\mathbb{N}}$ the sequence of solutions of a difference method for solving the ODE model problem. Then, this method is A-stable if for all

$$\lambda \in \mathbb{C}^- = \{\lambda : \, Re(\lambda) \leq 0\}$$

the discrete solutions are bounded (or even contractive) for arbitrary, but fixed, step size Δt. That is to say:

$$|y_{n+1}^{\Delta t}| \leq |y_n^{\Delta t}| < \infty \quad \text{for } n = 1, 2, 3, \ldots$$

Numerical analysis: A-stability

Proposition 6

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. $\square$

Remark 7

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

Proposition 8

The implicit Euler scheme and the trapezoidal rule are A-stable; see also the previous figure.

Numerical analysis: A-stability

We illustrate the previous statements.

1) In Proposition 5 we have seen that for the forward Euler method it holds:

$$y_n^{\Delta t} = R(z) y_{n-1}^{\Delta t},$$

where $R(z) = 1 + z$. Thus, according to Definition 13 and 14, we obtain convergence when the sequence $\{y_n^{\Delta t}\}$ is contracting:

$$|R(z)| \leq |1 + z| \leq 1. \tag{20}$$

- Thus if the value of λ (in $z = \lambda \Delta t$) is very big, we must choose a very small time step Δt in order to achieve $|1 - \lambda \Delta t| < 1$.
- 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 conclusion, the forward Euler scheme is only conditionally stable, i.e., it is stable provided that (20) is fulfilled.

Numerical analysis: A-stability

2) For the implicit Euler scheme, we see that

- a large λ and large Δt 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.

Example

We briefly compute the time step restriction for the DE system from the beginning. We had

$$Re(\lambda_1) = -2, \quad Re(\lambda_{2,3}) = -40.$$

Of course, the larger value is more criticial. Therefore:

$$|1 + \Delta t \lambda| = |1 - 40\Delta t| \leq 1.$$

Then, we obtain:

$$1 - 40\Delta t \leq 1 \Rightarrow -40\Delta t \leq 0 \Rightarrow \Delta t = 0.$$

$$-(1 - 40\Delta t) \leq 1 \Rightarrow 40\Delta t \leq 2 \Rightarrow \Delta t = \frac{1}{20}.$$

The first result is useless. The second finding shows that the critical time step size is $\Delta t = \frac{1}{20}$. In order to obtain a stable numerical result, we need to work with

$$\Delta t < \frac{1}{20}$$

when using the forward Euler scheme.

Stability for higher-order methods: Runge-Kutta

- Consider the Taylor scheme of order R:

$$y_n^{\Delta t} = y_{n-1}^{\Delta t} + \Delta t \sum_{r=1}^{R} \frac{\Delta t^{r-1}}{r!} f^{(r-1)}(t_{n-1}, y_{n-1}^{\Delta t}) = y_{n-1}^{\Delta t} + \Delta t \sum_{r=1}^{R} \frac{\Delta t^{r-1}}{r!} \lambda^r y_{n-1}^{\Delta t}$$

  where we recall that $f(t, y) = \lambda y$.

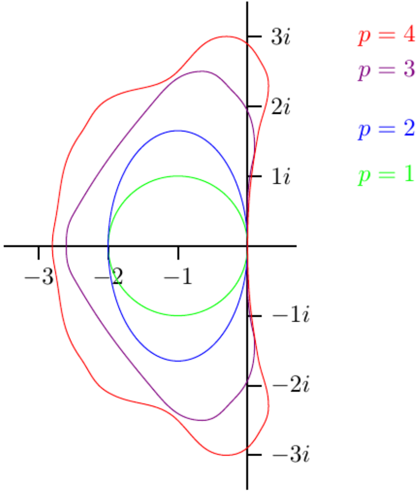
$$|y_n^{\Delta t}| \leq |y_{n-1}^{\Delta t}|$$

- Then, the stability factor $\omega(z)$ is given by:

$$\omega(z) = \sum_{r=0}^{R} \frac{z^r}{r!}, \quad z = \lambda \Delta t$$

- We then obtain (without any proofs!), the stability regions shown on the next slide.

Stability for higher-order methods: Runge-Kutta



Numerical analysis: consistency, local discretization error/truncation error

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

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

- We will obtain the truncation error by plugging the exact solution $y(t)$ into the numerical scheme.
- To this end, we obtain for the forward Euler scheme and let us now specify the truncation error η_{n-1}:

$$y'(t_{n-1}) + \eta_{n-1} = \frac{y(t_n) - y(t_{n-1})}{\Delta t}.$$

- To this end, we need information about the solution at the old time step t_{n-1} in order to eliminate $y(t_n)$.

Numerical analysis (cont'd)

Thus we use Taylor and develop $y(t^n)$ at the time point t^{n-1}:

$$y(t^n) = y(t^{n-1}) + y'(t^{n-1})k + \frac{1}{2}y''(\tau^{n-1})\Delta t^2$$

We obtain the difference quotient of forward Euler by the following manipulation:

$$\frac{y(t^n) - y(t^{n-1})}{\Delta t} = y'(t^{n-1}) + \frac{1}{2}y''(\tau^{n-1})\Delta t.$$

We observe that the first terms correspond to the forward Euler scheme. The remainder term is

$$\frac{1}{2}y''(\tau^{n-1})\Delta t$$

and therefore the truncation error η_{n-1} can be estimated as

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

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

Numerical analysis: convergence

Theorem 15 (Convergence of implicit/explicit Euler)

We have

$$\max_{t_n \in I} |y_n^{\Delta t} - y(t_n^{\Delta t})| \leq c(T, y)\Delta t = O(\Delta t),$$

where $\Delta t := \max_n \Delta t_n$.

Remark 9

The following 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 Δt is sufficiently small.

Numerical analysis: convergence (proof)

It holds for $1 \le n \le N$:

$$
\begin{aligned}
|y_n^{\Delta t} - y(t_n)| = \|e_n\| = \Delta t \|\sum_{k=0}^{n-1} B_E^{n-k} \eta_k\| &\le \Delta t \sum_{k=0}^{n-1} \|B_E^{n-k} \eta_k\| \quad \text{(triangle inequality)} \\
&\le \Delta t \sum_{k=0}^{n-1} \|B_E^{n-k}\| \, \|\eta_k\| \\
&\le \Delta t \sum_{k=0}^{n-1} \|B_E^{n-k}\| \; Ck \quad \text{(consistency)} \\
&\le \Delta t \sum_{k=0}^{n-1} 1 \; C\Delta t \quad \text{(stability)} \\
&= \Delta t N \; C\Delta t \\
&= T \; C\Delta t, \quad \text{where we used } \Delta t = T/N \\
&= C(T, y)\Delta t \\
&= O(\Delta t)
\end{aligned}
$$

What does the convergence order Δt tell us?

- Linear convergence: bisecting Δt will reduce the error by a factor of 2
- Quadratic convergence: bisecting Δt will reduce the error by a factor of 4

Computational convergence analysis

In order to calculate the convergence order α from numerical results, we make the following derivation.

- Let $P(\Delta t) \to P$ for $\Delta t \to 0$ be a converging process and assume that
$$P(\Delta t) - \tilde{P} = O(\Delta t^{\alpha}).$$
- 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 of runs if the limit P is not known). That is
$$P_1 := P(\Delta t), \quad P_2 := P(\Delta t/2), \quad P_3 := P(\Delta t/4).$$
- Then, the convergence order can be calculated via the formal approach $P(\Delta t) - \tilde{P} = c\Delta t^{\alpha}$ with the following formula.

Computational convergence analysis

Proposition 10 (Computationally-obtained convergence order)

Given three numerically-obtained values P_1, P_2 and P_3, the convergence order can be estimated as:

$$\alpha = \frac{1}{log(2)} log\Big(\Big|\frac{P_1 - P_2}{P_2 - P_3}\Big|\Big). \tag{21}$$

The order α is an estimate and heuristic because we assumed a priori a given order, which strictly speaking we have to proof first.

Substantiating our theoretical results: example

We solve our ODE model problem numerically.

Formulation 3

Let $a = g - m$ be $a = 0.25$ (test 1) or $a = -0.25$ (test 2) or $a = -10$ (test 3). The IVP is given by:

$$y' = ay, \quad y(t_0 = 2011) = 2.$$

The end time value is $T = 2014$.

The tasks are:

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

Discussion of the results for test 1 $a = 0.25$

In the following, we present our results for the end time value $y(T = 2014)$ for test case 1 ($a = 0.25$) on three mesh levels:

```
Scheme        #steps (N)  \Delta t    y_N^{Dt} Error (abs.)
=========================================================
FE               8        0.37500     4.0961   0.13786
BE               8        0.37500     4.3959   0.16188
CN               8        0.37500     4.2363   0.0023295
---------------------------------------------------------
FE               16       0.18750     4.1624   0.071567
BE               16       0.18750     4.3115   0.077538
CN               16       0.18750     4.2346   0.00058168
---------------------------------------------------------
FE               32       0.093750    4.1975   0.036483
BE               32       0.093750    4.2720   0.037974
CN               32       0.093750    4.2341   0.00014538
=========================================================
```

Discussion of the results for test 1 $a = 0.25$

- 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 (21), we work with $\Delta t = \Delta t_{max} = 0.375$. Then we identify in the above table that

$$P(\Delta t_{max}) = P(0.375) = |y(T) - y_8^{\Delta t}|,$$
$$P(\Delta t_{max}/2) = P(0.1875) = |y(T) - y_{16}^{\Delta t}|$$
$$P(\Delta t_{max}/4) = P(0.09375) = |y(T) - y_{32}^{\Delta t}|.$$

Discussion of the results for test 1 $a = 0.25$

- We monitor that doubling the number of intervals (i.e., halving the step size Δt) 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 (21) yielding $\alpha = 1.0921$. 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 $\alpha = 2.0022$.
- 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.

Discussion of the results for test 1 $a = 0.25$

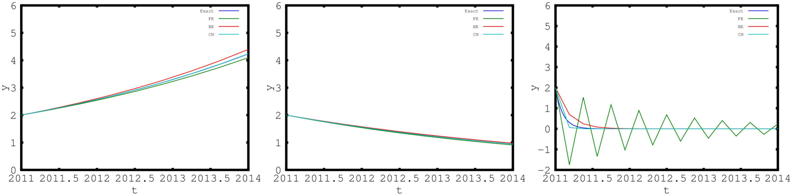


Figure: On the left, the solution to test 1 is shown. In the middle, test 2 is plotted. On the right, the solution of test 3 with $N = 16$ (number of intervals) is shown. Here, $N = 16$ corresponds to a step size $\Delta t = 0.18$ which is slightly below the critical step size for convergence. Thus we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

Discussion of the results for test 3 $a = -10$

The convergence interval for the forward Euler scheme reads:

$$|1 + z| \leq 1 \quad \Rightarrow \quad |1 + a\Delta t| \leq 1$$

In test 3, we are given $a = -10$, yielding:

$$|1 + z| \leq 1 \quad \Rightarrow \quad |1 - 10\Delta t| \leq 1$$

- Thus, we need to choose a Δt that fulfills the previous relation. In this case, we easily calculate $\Delta t < 0.2$.
- This means that for all $\Delta t < 0.2$ we should have convergence of the forward Euler method and for $\Delta t \geq 0.2$ non-convergence (and in particular no stability!).

Discussion of the results for test 3 $a = -10$

We perform the following additional tests:

- Test 3a: $N = 10$, yielding $\Delta t = 0.3$;
- Test 3b: $N = 15$, yielding $\Delta t = 0.2$; exactly the boundary of the stability interval;
- Test 3c: $N = 16$, yielding $\Delta t = 0.1875$; from before;
- Test 3d: $N = 20$, yielding $\Delta t = 0.15$.

Discussion of the results for test 3 $a = -10$

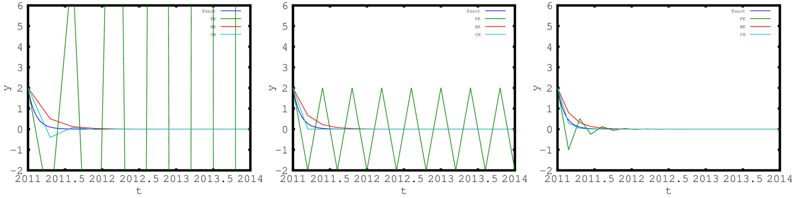


Figure: Tests 3a,3b,3d: Blow-up, constant zig-zag non-convergence, and convergence of the forward Euler method.

Summary of lecture 04

- Numerical analysis of some numerical schemes:
  - forward Euler (first order, explicit, only conditionally stable with time step size, critical for stiff problems),
  - backward Euler (first order, implicit, A-stable),
  - trapezoidal rule (second order, implicit, A-stable)
- Numerical tests demonstrating the theoretical results:
  - Nonstable schemes exhibit oscillations or blow-up of the solution
  - A higher convergence order (trapezoidal rule) has a higher accuracy and converges faster

Exercise 2 Overview

- Recapitulate when explicit Euler produces bounded approximations for the model problem $u' = \lambda u$ and confirm the results in HDNUM
- Learn how an ODE Solver can be implemented in an object-oriented way
- Investigate errors and convergence rates of various explicit and implicit schemes for a linear oscillator problem
- Explore the nonlinear Van der Pol oscillator using explicit and implicit methods.

Task 1

- Consider the linear, scalar model problem

$$u'(t) = \lambda u(t), \quad u(0) = 1, \quad \mathbb{R} \ni \lambda < 0$$

- Derive the explicit Euler scheme
- What is the condition on Δt such that the explicit Euler scheme produces bounded approximations for all $t > 0$
- Confirm your result with the implementation in file `eemodelproblem.cc` provided in the exercise yesterday

Task 2

- We will explain how ODE solvers are implemented in an object-oriented way in HDNUM
- Download the file `linearoscillator.cc` available on the cloud `https://cloud.ifam.uni-hannover.de/index.php/s/Cwe4ZqwLRMixS3J`. It solves the problem

$$u'(t) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} u(t) \text{ in } (0, 20\pi], \qquad u(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  using the methods

#	Scheme	#	Scheme
0	Explicit Euler	4	Implicit Euler
1	Heun 2nd order	5	Implicit midpoint
2	Heun 3rd order	6	Alexander
3	Runge-Kutta 4th order	7	Crouzieux
		8	Gauß 6th order

- and provides errors $e(T)$ and convergence rates for all schemes
- What conclusions can you draw from the tables?

Task 3

- In this exercise we explore the nonlinear Van der Pol oscillator

$$\begin{aligned} u_0'(t) &= -u_1(t) & \quad u_0(0) &= 1 \\ u_1'(t) &= 1000 \cdot (u_0(t) - u_1^3(t)) & \quad u_1(0) &= 2 \end{aligned}$$

  which is an example for a stiff ODE system
- Download an updated version of the file `vanderpol.cc` from the cloud
- Compile and run the following four combinations of methods and timesteps:
  - `RKF45` method is an adaptive embedded Runge Kutta method of 5th order. Run it with tolerances $TOL_1 = 0.2$ and $TOL_2 = 0.001$ using an initial time step $\Delta t = 1/16$
  - The implicit Euler method. Run it with $\Delta t_1 = 1/16$ and $\Delta t_2 = 1/512$
- The output file contains in each line $t_i u_0(t_i) u_1(t_i) \Delta t_i$
- Compare the solutions, especially $u_1(t)$ as well as the time step sizes Δt_i for all four runs. What do you observe?

Contents

A Different Philosophy

- In traditional methods, such as Runge-Kutta, one approximates the unknown function $y(t)$ at temporal values t_n
- In the Galerkin method we approximate $y(t)$ by simple functions such as polynomials
- In this way the approximation is defined at all points in time
- In addition, the ODE is satisfied in an averaged (weak) sense
- Also the error $e(t) = y(t) - y^{\Delta t}(t)$ is defined at all times
- This allows the use of more sophisticated mathematical methods for their analysis
- The presentation follows chapters 6 and 9 from the book "Computational Differential Equations" by Eriksson, Estep, Hansbo and Johnson

Variational (or Weak) Formulation

- Consider the first-order systems of ODEs in $\mathbb{R}^d$ in explicit form

$$y'(t) - f(t, y(t)) = 0, \qquad t \in (t_0, t_0 + T], \qquad y(t_0) = y_0$$

  to determine the unknown function $y : [t_0, t_0 + T] \to \mathbb{R}^d$

- Given a suitable function $\varphi : [t_0, t_0 + T] \to \mathbb{R}^d$ we may multiply and integrate:

$$\int_{t_0}^{t_0+T} (y'(t) - f(t, y(t))) \cdot \varphi(t)\, dt = 0$$

  where "$\cdot$" denotes the Euclidean scalar product.

- Demanding this identity for a sufficiently large class of functions $\varphi \in V_0 = \{v : v(t_0) = 0\}$, we may hope this fixes (uniquely) a function $y \in Y = \{w : w(t_0) = y_0\}$
- This function is called a *variational (or weak) solution of the ODE*
- Under suitable conditions the weak and strong solution coincide
- The *function* $R[y]$, $R[y](t) = y'(t) - f(t, y(t))$ is called *residual*

Galerkin Method

- The function space V_0 is infinite-dimensional, e.g. all continuous functions (with zero initial value)
- **Idea**: Replace V_0 and Y by *finite-dimensional* counter parts!

Example: Use global polynomials. Let us fix $d = 1$

- Define the following classes of polynomials:

$$\begin{aligned}
\mathcal{P}^q &= \text{polynomials of degree } q \\
\mathcal{P}_0^q &= \{p \in \mathcal{P}^q : p(t_0) = 0\} \\
Y^q &= \{p \in \mathcal{P}^q : p = y_0 + v, v \in \mathcal{P}_0^q\} =: y_0 + \mathcal{P}_0^q
\end{aligned}$$

  and note: $\mathcal{P}^q$ is a vector space of dimension $q+1$, $\mathcal{P}_0^q$ is a proper subspace of dimension q, Y^q is called an affine space
- Then the global Galerkin method reads: Find $y^{\Delta t}(t) \in Y^q$ such that:

$$\int_{t_0}^{t_0+T} ((y^{\Delta t})'(t) - f(t, y^{\Delta t}(t)))\varphi(t)\, dt = 0 \qquad \forall \varphi \in \mathcal{P}_0^q$$

Galerkin Method. Example continued

- How to make this method practical?
- Choose a basis representation:

$$\mathcal{P}^q = \text{span}\{1, t-t_0, \ldots, (t-t_0)^q\}, \qquad \mathcal{P}_0^q = \text{span}\{t-t_0, \ldots, (t-t_0)^q\}$$

- Make the ansatz $y^{\Delta t}(t) = y_0 + \sum_{j=1}^q \xi_j (t-t_0)^j$ and insert:

$$\int_{t_0}^{t_0+T} ((y^{\Delta t})'(t) - f(t, y^{\Delta t}(t)))\varphi(t)\, dt = 0 \quad \forall \varphi \in \mathcal{P}_0^q$$

$$\int_{t_0}^{t_0+T} \left(\sum_{j=1}^q \xi_j j (t-t_0)^{j-1} - f\left(t, y_0 + \sum_{j=1}^q \xi_j (t-t_0)^j \right) \right) (t-t_0)^i\, dt = 0 \quad 1 \le i \le q$$

$$\sum_{j=1}^q \xi_j j \frac{T^{i+j}}{i+j} - \int_{t_0}^{t_0+T} f\left(t, y_0 + \sum_{j=1}^q \xi_j (t-t_0)^j \right) (t-t_0)^i\, dt = 0 \quad 1 \le i \le q$$

- Need to solve q coupled nonlinear equations for the coefficients ξ_j

Some Choices

The accuracy of the method can be controlled by

- Increasing the polynomial degree (called *p*-method)
    - Algebraic problem might become ill-conditioned
    - Remedied by choosing an appropriate basis
    - Needs sufficient regularity of the solution of the ODE
- Using piecewise polynomials of degree q (called *h*-method)
    - We will follow this approach below
- Combination of both (called *hp*-method)
- Using of trigonometric polynomials (spectral method)

- *Error control*: What is the error in the computed solution $y^{\Delta t}(t)$?
- *Adaptivity*: How to choose q and Δt_n to control the error?

Piecewise Polynomial Functions

- As before we treat $d = 1$, extend to arbitrary d by making each component a polynomial
- Choose N time steps as before

$$t_0 < t_1 < t_2 < \ldots < t_{N-1} < t_N = t_0 + T, \qquad \Delta t_n = t_{n+1} - t_n,$$
$$I = (t_0, t_0 + T), \quad I_n = (t_n, t_{n+1}), \quad \mathcal{T}_N = \{I_n : 0 \le i < N\}.$$

- Continuous piecewise polynomials of degree q are

$$V_N^q = \{v \in C^0(\bar{I}) : v|_{I_n} \in \mathcal{P}^q, 0 \le i < N\}$$

- Continuous piecewise polynomials of degree q with zero initial value

$$V_{N,0}^q = \{v \in V_N^q : v(t_0) = 0\} \subset V_N^q$$

- Discontinuous piecewise polynomials:

$$W_N^q = \{v \in L^2(I)) : v|_{I_n} \in \mathcal{P}^q, 0 \le i < N\}$$

- By $v_n = v|_{I_n}$ we denote the piece on interval I_n

cG(q) Method

Find $y^{\Delta t}(t) \in y_0 + V_{N,0}^q$ such that

$$\int_{t_0}^{t_0+T} ((y^{\Delta t})'(t) - f(t, y^{\Delta t}(t)))\varphi(t)\, dt = 0 \qquad \forall \varphi \in W_N^{q-1}$$

- Note the use of test functions $\mathcal{P}^{q-1}$ instead of $\mathcal{P}_0^q$ on I_n
- The choice of *discontinuous* test functions is essential, since it allows to solve the problem sequentially! The discrete solution $y^{\Delta t}$ can be determined as follows
- Consider $I_0 = (t_0, t_1]$, restricting the test functions $\varphi_n = 0$, $i > 1$:

  Find $y_0^{\Delta t}(t) \in y_0 + \mathcal{P}_0^q$: $\displaystyle\int_{I_0} ((y_0^{\Delta t})'(t) - f(t, y_0^{\Delta t}(t)))\varphi(t)\, dt = 0 \qquad \forall \varphi \in \mathcal{P}^{q-1}$
- Considering $I_n = (t_n, t_{n+1}]$, $i > 0$, assume $y_{i-1}^{\Delta t}(t)$ is available:

  Find $y_n^{\Delta t}(t) \in y_{i-1}^{\Delta t}(t_n) + \mathcal{P}_0^q$: $\displaystyle\int_{I_n} ((y_n^{\Delta t})'(t) - f(t, y_n^{\Delta t}(t)))\varphi(t)\, dt = 0 \qquad \forall \varphi \in \mathcal{P}^{q-1}$
- The value at the end of interval I_{i-1} is used as initial value in I_n

dG(q) Method

- Now we approximate $y^{\Delta t}$ in W_N^q, i.e. $y^{\Delta t}$ might be discontinuous at t_n
- For $v \in W_N^q$ introduce the notation

$$v_n^+ = v_n(t_n), \quad v_n^- = v_{i-1}(t_n), \quad v_0^- = v_0 \text{ (v_0 a given number)}$$

and the *jump*

$$[v]_n = v_n^+ - v_n^-, \qquad 0 \le i < N$$

- Then the dG(q) methods reads: Find $y^{\Delta t}(t) \in W_N^q$ such that

$$\sum_{i=0}^{N-1} \left\{ \int_{I_n} ((y^{\Delta t})'(t) - f(t, y^{\Delta t}(t)))\varphi(t)\, dt + [y^{\Delta t}]_n \varphi_n^+ \right\} = 0 \qquad \forall \varphi \in W_N^q$$

- Note that both, the solution and the test functions, are in W_N^q
- Of course this needs some explanation

dG(q) Method: Sequential Solution

- Without the jump term the solutions in the intervals I_n would be completely independent of each other, with the jump term we get
- In the interval $I_0 = (t_0, t_1]$: Find $y_0^{\Delta t}(t) \in \mathcal{P}^q$:

$$\int_{I_0} ((y_0^{\Delta t})'(t) - f(t, y_0^{\Delta t}(t)))\varphi(t)\,dt + (y_0^{\Delta t}(t_0) - y_0)\varphi_0(t_0) = 0 \qquad \forall \varphi \in \mathcal{P}^q$$

- In interval $I_n = (t_n, t_{n+1}]$, $i > 0$: Find $y_n^{\Delta t}(t) \in \mathcal{P}^q$:

$$\int_{I_n} ((y_n^{\Delta t})'(t) - f(t, y_n^{\Delta t}(t)))\varphi(t)\,dt + (y_n^{\Delta t}(t_n) - y_{i-1}^{\Delta t}(t_n))\varphi_n(t_n) = 0 \qquad \forall \varphi \in \mathcal{P}^q$$

- Thus, the $y_n^{\Delta t}$ are determined sequentially by solving $q+1$ nonlinear algebraic equations in each interval
- The method can be extended to the vector-valued case

dG(q) Method: Jump Term Explained

- Consider the simple problem

$$y'(t) = 0 \text{ in } (t_0, t_0 + T], \qquad y(t_0) = y_0$$

which has the constant solution $y(t) = y_0$

- Using the weak formulation we obtain using integration by parts:

$$\sum_{i=0}^{N-1} \int_{I_n} (y_n^{\Delta t})'(t)\varphi_n(t)\,dt = \sum_{i=0}^{N-1} \left\{ -\int_{I_n} y_n^{\Delta t}(t)\varphi_n'(t)\,dt + y_n^{\Delta t}(t_{n+1})\varphi(t_{n+1}) - y_n^{\Delta t}(t_n)\varphi(t_n) \right\} = 0$$

- With a small change we obtain the correct solution:

$$\sum_{i=0}^{N-1} \left\{ -\int_{I_n} y_n^{\Delta t}(t)\varphi_n'(t)\,dt + y_n^{\Delta t}(t_{n+1})\varphi(t_{n+1}) - y_{i-1}^{\Delta t}(t_n)\varphi(t_n) \right\} = 0$$

(Observe that $y_n^{\Delta t}(t) = y_{i-1}^{\Delta t}(t_n)$ solves the problem in each interval)

- The change may be expressed as

$$\sum_{i=0}^{N-1} \int_{I_n} (y_n^{\Delta t})'(t)\varphi_n(t)\,dt + y_0^{\Delta t}(t_0)\varphi_0(t_0) - y_0\varphi_0(t_0) + \sum_{i=1}^{N-1} (y_n^{\Delta t}(t_n)\varphi_n(t_n) - y_{i-1}^{\Delta t}(t_n)\varphi_n(t_n)) = 0$$

- This is exactly the jump term in the formulation

Example: cG(1) Method

- Recall: Find $y_n^{\Delta t}(t) \in y_{i-1}^{\Delta t}(t_n) + \mathcal{P}_0^1$:

$$\int_{I_n} ((y_n^{\Delta t})'(t) - f(t, y_n^{\Delta t}(t)))\varphi(t)\, dt = 0 \qquad \forall \varphi \in \mathcal{P}^0$$

- $\mathcal{P}^0 = \text{span}\{1\}$, for $y_n^{\Delta t}(t) \in y_{i-1}^{\Delta t}(t_n) + \mathcal{P}_0^1$ make the Ansatz

$$y_n^{\Delta t}(t) = \underbrace{y_n^{\Delta t}}_{=y_{i-1}^{\Delta t}(t_n)} \underbrace{\frac{t_{n+1} - t}{\Delta t_n}}_{\psi_n^0} + y_{n+1}^{\Delta t} \underbrace{\frac{t - t_n}{\Delta t_n}}_{\psi_n^1}$$

- Inserting the Ansatz into the formulation and $\varphi = 1$:

$$\int_{I_n} y_n^{\Delta t}\left(-\frac{1}{\Delta t_n}\right) + y_{n+1}^{\Delta t}\frac{1}{\Delta t_n} - f(t, y_n^{\Delta t}\psi_n^0(t) + y_{n+1}^{\Delta t}\psi_n^1(t)))\, dt = 0$$

$$\Leftrightarrow \quad y_{n+1}^{\Delta t} - y_n^{\Delta t} - \int_{I_n} f(t, y_n^{\Delta t}\psi_n^0(t) + y_{n+1}^{\Delta t}\psi_n^1(t)))\, dt = 0$$

- Using 2nd order quadrature yields the implicit trapezoidal rule or the implicit midpoint rule

Example: dG(0) Method

- Recall: Find $y^{\Delta t}(t) \in W_N^0$ such that

$$\sum_{i=0}^{N-1} \left\{ \int_{I_n} ((y^{\Delta t})'(t) - f(t, y^{\Delta t}(t)))\varphi(t)\, dt + [y^{\Delta t}]_n \varphi_n^+ \right\} = 0 \qquad \forall \varphi \in W_N^0$$

- Choose the basis and the Ansatz

$$\psi_n(t) = \begin{cases} 1 & t \in I_n \\ 0 & \text{else} \end{cases}, \qquad y^{\Delta t}(t) = \sum_{i=0}^{N-1} y_{n+1}^{\Delta t} \psi_n(t)$$

observe that due to $I_n = (t_n, t_{n+1}]$ one may interpret $y_{n+1}^{\Delta t}$ as the value at the end of the time interval I_n

- Inserting in formulation gives

$$-\int_{I_n} f(t, y_{n+1}^{\Delta t}), dt + y_{n+1}^{\Delta t} - y_n^{\Delta t} = 0$$

- Which yields the implicit Euler method upon quadrature:

$$y_{n+1}^{\Delta t} - y_n^{\Delta t} - \Delta t_n f(t, y_{n+1}^{\Delta t}), dt = 0$$

Error Control and Adaptivity

- **Error control**: Stop the computation when

$$J(y - y^{\Delta t}) \leq TOL$$

  where J is some functional of the error $e = y - y^{\Delta t}$ and TOL is a user given tolerance
- **Adaptivity**: Choose $\mathcal{T}_N$ such that $J(y - y^{\Delta t}) \leq TOL$ is achieved with N as small as possible
- An example would be $J(e) = e(T)$
- Adaptive time step control for traditional methods tries to estimate the leading order term of the truncation error
- Galerkin methods allow a much more rigorous and flexible approach that achieves error control

Dual Problem in A Posteriori Error Estimation

- We restrict ourselves to the *linear* ODE

$$y'(t) + a(t)y(t) = f(t),\ t \in (0, T], \qquad y(0) = y_0$$

- We will have a glimpse on the *dual weighted residual* (DWR) method [4] to estimate the error

$$e(t) = y(t) - y^{\Delta t}(t), \qquad \text{in particular } e(T)$$

- DWR is based on a so-called *dual problem* which reads in this case

$$-\varphi'(t) + a(t)\varphi(t) = 0,\ t \in [0, T), \qquad \varphi(T) = e(T)$$

- Note, that this problem runs *backward* in time with the error $e(T)$ given as initial value
- With the change of variables $t(\tilde{t}) = T - \tilde{t}$ and the identification of $\tilde{\varphi}(\tilde{t}) = \varphi(t(\tilde{t})) = \varphi(T - \tilde{t})$ we obtain an equation for $\tilde{\varphi}$:

$$\tilde{\varphi}(\tilde{t}) + a(T - \tilde{t})\tilde{\varphi}(\tilde{t}) = 0,\ t \in (0, T], \qquad \tilde{\varphi}(0) = e(T)$$

[4] Becker, Rannacher; 1996/2001

Error Representation

- Using the dual problem one obtains

$$
\begin{aligned}
0 &= \int_0^T e(t) \underbrace{\left(-\varphi(t) + a(t)\varphi(t)\right)}_{=0} dt \\
&= \int_0^T e'(t)\varphi(t) + e(t)a(t)\varphi(t)\,dt - e(T)\underbrace{\varphi(T)}_{=e(T)} + \underbrace{e(0)}_{=0}\varphi(0) \\
\Leftrightarrow \quad e^2(T) &= \int_0^T (e'(t) + a(t)e(t))\varphi(t)\,dt = \int_0^T (e'(t) + a(t)e(t))\varphi(t)\,dt \\
&= \int_0^T (y'(t) - (y^{\Delta t})'(t) + a(t)y(t) - a(t)y^{\Delta t}(t))\varphi(t)\,dt \\
\Rightarrow \quad \underbrace{e^2(T)}_{\text{error at } T} &= \int_0^T \underbrace{(f(t) - (y^{\Delta t})'(t) - a(t)y^{\Delta t}(t))}_{\text{residual } R[y^{\Delta t}]} \underbrace{\varphi(t)}_{\text{solution of dual problem}} dt
\end{aligned}
$$

  the last step is due to y being the solution of the ODE $y' + ay = f$
- This is an exact representation of the error in terms of the computable residual $R[y^{\Delta t}]$ and the solution of the dual problem

A Posteriori Error Estimate

- From the exact error representation one may proceed in different ways to produce an error estimate
- One uses Cauchy Schwarz inequality:

$$e^2(T) = \int_0^T R[u](t)\varphi(t)\,dt = \sum_{i=0}^{N-1} \int_{I_n} R[u](t)\varphi(t)\,dt$$
$$\leq \sum_{i=0}^{N-1} \left(\int_{I_n} R^2[u](t)\,dt\right)^{\frac{1}{2}} \left(\int_{I_n} \varphi^2(t)\,dt\right)^{\frac{1}{2}} = \sum_{i=0}^{N-1} \|R[y^{\Delta t}]\|_{0,I_n} \|\varphi\|_{0,I_n}$$

- This is interpreted as follows:
  - The fully computable term $\|R[y^{\Delta t}]\|_{0,I_n}$ measures the error contribution in interval I_n
  - The term $\|\varphi\|_{0,I_n}$ gives the weight of this contribution in the final result
  - This explains the name DWR
- The solution of the dual problem can be approximated (including the initial condition $e(T)$) as it determines only *the relative importance* of the residual contribution

Towards an Adaptive Time-stepping Scheme

How to do error control and adaptivity with the formula

$$|e(T)| \leq \left(\sum_{i=0}^{N-1} \|R[y^{\Delta t}]\|_{0,I_n} \|\varphi\|_{0,I_n} \right)^{\frac{1}{2}} \quad ? \tag{22}$$

1. Choose Δt_0. Compute solutions $y_0^{\Delta t}$ and $y_1^{\Delta t}$ with mesh size Δt_0 and $\Delta t_1 = \Delta t_0/2$. From this estimate the error at final time:
$$\begin{aligned} |\tilde{e}(T)| = |y(T) - y_0^{\Delta t}(T)| &= |y(T) - y_1^{\Delta t}(T) + y_1^{\Delta t}(T) - y_0^{\Delta t}(T)| \\ &\leq |y(T) - y_1^{\Delta t}(T)| + |y_1^{\Delta t}(T) - y_0^{\Delta t}(T)| \\ &\leq \alpha |y(T) - y_0^{\Delta t}(T)| + |y_1^{\Delta t}(T) - y_0^{\Delta t}(T)| \\ \Leftrightarrow |\tilde{e}(T)| &\leq \frac{1}{1-\alpha} |y_1^{\Delta t}(T) - y_0^{\Delta t}(T)| \end{aligned}$$
2. Given an estimate $\tilde{e}(T)$ of $e(T)$ solve the dual problem
3. Compute estimate for $|e(T)|$ using (22), if $|e(T)| \leq TOL$ STOP
4. Halfen the intervals I_n giving the largest error contribution
5. Recompute $y^{\Delta t}$ on new mesh, recompute estimate $\tilde{e}(T)$, goto 2

Galerkin Orthogonality

- We first need a further result called **Galerkin orthogonality**
- We may define a piecewise continuous solution $y_n(t)$

$$\int_{I_n} (y_n'(t) + a(t)y_n(t))\varphi_n(t)\,dt = \int_{I_n} f(t)\varphi_n(t)\,dt \quad \forall \varphi_n \in V(I_n) \quad y_n(t_n) = y_{i-1}(t_n$$

- and the discrete solution in, say cG(q)

$$\int_{I_n} ((y_n^{\Delta t})')(t) + a(t)y_n^{\Delta t}(t))\varphi_n(t)\,dt = \int_{I_n} f(t)\varphi_n(t)\,dt \quad \forall \varphi_n \in \mathcal{P}^{q-1} \quad y_n^{\Delta t}(t_n) =$$

- Subtracting and summing over all intervals gives the Galerkin orthogonality relation

$$\sum_{i=0}^{N-1} \int_{I_n} (e'(t) + a(t)e(t))\varphi(t)\,dt = \sum_{i=0}^{N-1} \int_{I_n} R[u](t)\varphi(t)\,dt = 0 \quad \forall \varphi \in \mathcal{P}^{q-1}$$

together with $e_n(t_n) = e_{i-1}(t_n)$

Another A Posteriori Error Estimate

- The second approach uses an analytical estimate to avoid the solution of a dual problem
- Again we start from the error relation

$$e^2(T) = \sum_{i=0}^{N-1} \int_{I_n} R[y^{\Delta t}](t)\varphi(t)\,dt$$

- Using the L^2-projection $\pi\varphi$ of the dual solution to piecewise polynomials we get

$$\begin{aligned} e^2(T) &= \sum_{i=0}^{N-1} \left\{ \int_{I_n} R[y^{\Delta t}](t)\varphi(t)\,dt - \int_{I_n} R[y^{\Delta t}](t)\pi\varphi(t)\,dt \right\} \\ &= \sum_{i=0}^{N-1} \int_{I_n} R[y^{\Delta t}](t)(\varphi(t) - \pi\varphi(t))\,dt \end{aligned}$$

- For the L^2-projection one has the L^1-estimate

$$\int_{I_n} |\varphi(t) - (\pi\varphi)(t)|\,dt \leq \Delta t_n \int_{I_n} |\varphi'(t)|\,dt$$

Another A Posteriori Error Estimate, ctd.

- and with that we may estimate

$$e^2(T) = \sum_{i=0}^{N-1} \left\{ \|R[y^{\Delta t}]\|_{\infty,I_n} \Delta t_n \int_{I_n} |\varphi'(t)|\, dt \right\} \leq \max_{0 \leq i < N} (\Delta t_n \|R[y^{\Delta t}]\|_{\infty,I_n}) \int_0^T |\varphi'(t)|\, d$$

- We have an analytical solution for φ from which one obtains

$$|\varphi(t)| \leq |e(T)| \exp(\mathcal{A}T), \qquad \forall 0 \leq t \leq T, \qquad |a(t)| \leq \mathcal{A}$$

- Introduce the stability factor $S(T) = \int_0^T |\varphi'(t)|\, dt / |e(T)|$ and
  - If $|a(t)| \leq \mathcal{A}$ then $S(T) \leq \exp(\mathcal{A})$
  - If $a(t) \geq 0$ then $S(T) \leq 1$
- To obtain the final estimate

$$|e(T)| \leq \max_{0 \leq i < N} (\Delta t_n \|R[y^{\Delta t}]\|_{\infty,I_n})$$

What Else ?

There are many more important aspects we could not treat in this first part on ODEs:

- More in-depth treatment of adaptive methods
- E.g. embedded Runge-Kutta methods, extrapolation method
- More in-depth treatment of stiff ODEs
- E.g. different definitions of stiffness, Padé-table, rational approximations of the exponential function
- Implicit higher order Runge-Kutta methods, collocation method to derive them
- Linear multistep methods
- Symplectic methods

Contents

Definition of a PDE

Definition 16 (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.

Example 11

Often, we have x, y, z as spatial independent variables and t as a temporal variable. These are four independent variables.

Laplace equation / Poisson's equation

Formulation 4 (Laplace problem / Poisson problem)

Let Ω be an open set. The **Laplace equation** reads:

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

The **Poisson problem** reads:

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

Definition 17

A C^2 function (C^2 means two times continuously differentiable) that satisfies the Laplace equation is called **harmonic** function.

Notation

We frequently use:

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

and

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

and

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

and

$$\frac{\partial^2 u}{\partial x \partial y} = \partial_{xy} u$$

Nabla operators

Well-known in physics, it is convenient to work with the **nabla-operator** to define derivative expressions. 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 gradient of a vector-valued function $v : \mathbb{R}^n \to \mathbb{R}^m$ is called **Jacobian matrix** and reads:

$$\nabla v = \begin{pmatrix} \partial_1 v_1 & \dots & \partial_n v_1 \\ \vdots & & \vdots \\ \partial_1 v_m & \dots & \partial_n v_m \end{pmatrix}.$$

Nabla operators

The divergence is defined for vector-valued functions $v : \mathbb{R}^n \to \mathbb{R}^n$:

$$\text{div } v := \nabla \cdot v := \nabla \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \sum_{k=1}^{n} \partial_k v_k.$$

The divergence for a tensor $\sigma \in \mathbb{R}^{n\times n}$ is defined as:

$$\nabla \cdot \sigma = \Big(\sum_{j=1}^{n} \frac{\partial \sigma_{ij}}{\partial x_j}\Big)_{1\leq i\leq n}.$$

The trace of a matrix $A \in \mathbb{R}^{n\times n}$ is defined as

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

Nabla operators

Definition 18 (Laplace operator)

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 19

For a vector-valued function $u : \mathbb{R}^n \to \mathbb{R}^m$, we define the Laplace operator component-wise as

$$\Delta u = \Delta \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n} \partial_{kk} u_1 \\ \vdots \\ \sum_{k=1}^{n} \partial_{kk} u_m \end{pmatrix}.$$

Physical interpretation / mathematical modeling of the Laplace operator

The physical interpretation is as follows. Let u denote the density of some quantity, for instance concentration or temperature, in equilibrium. If G is any smooth region $G \subset \Omega$, the flux F of the quantity u through the boundary ∂G is zero:

$$\int_{\partial G} F \cdot n \, dx = 0. \tag{23}$$

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

$$\nabla \cdot F = 0 \quad \text{in } \Omega. \tag{24}$$

Physical interpretation

- Now we need a second assumption (or better a relation) between the flux and the quantity u. Such relations do often come from material properties and are so-called **constitutive laws.**
- In many situations it is reasonable to assume that the flux F is proportional to the negative gradient $-\nabla u$ of the quantity u. This means that flow goes from regions with a higher concentration to lower concentration regions.
- For instance, the rate at which energy 'flows' (or diffuses) as heat from a warm body to a colder body is a function of the temperature difference. The larger the temperature difference, the larger the diffusion.
- We consequently obtain as further relation:

$$F = -\nabla u.$$

Physical interpretation

- Plugging into the Equation (24) yields:

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

This is the simplest derivation one can make. Adding more knowledge on the underlying material of the body, a material parameter $a > 0$ can be added:

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

And adding a nonconstant and spatially dependent material further yields:

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

In this last equation, we do not obtain any more the classical Laplace equation but a diffusion equation in divergence form.

Other fields using Poisson's equation

Some important physical laws are related to the Laplace operator (partially taken from L. Evans; Partial Differential Equations, AMS, 2010):

1. Fick's law of chemical diffusion
2. Fourier's law of heat conduction
3. Ohm's law of electrical conduction
4. Small deformations in elasticity (recall the clothesline problem)

Three important linear PDEs

- 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.

Elliptic PDEs: prototype Laplacian

Formulation 5

Let $f : \Omega \to \mathbb{R}$ be given. Furthermore, Ω is an open, bounded set of $\mathbb{R}^d$. We seek the unknown function $u : \bar{\Omega} \to \mathbb{R}$ such that

$$Lu = f \quad \text{in } \Omega, \tag{25}$$

$$u = 0 \quad \text{on } \partial\Omega. \tag{26}$$

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

$$Lu := -\sum_{i,j=1}^{d} \partial_{x_j}(a_{ij}(x)\partial_{x_i} u) + \sum_{i=1}^{d} b_i(x)\partial_{x_i} u + c(x)u, \quad u = u(x), \tag{27}$$

with the symmetry assumption $a_{ij} = a_{ji}$ and given coefficient functions a_{ij}, b_i, c. Moreover, we assume that A is positive definite (in order words: the eigenvalues are positive).

Elliptic PDEs: prototype Laplacian

Formulation 6

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 (26) is called **homogeneous Dirichlet condition**.

Elliptic PDEs: prototype Laplacian

Theorem 20 (Strong maximum principle for the Laplace problem)
Suppose $u \in C^2(\Omega) \cap C(\bar{\Omega})$ is a harmonic function. Then

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

Moreover, if Ω is connected and there exists a point $y \in \Omega$ in which

$$u(y) = \max_{\bar{\Omega}} u,$$

then u is constant within Ω. The same holds for $-u$, but then for minima.

Remark 12
The maximum principle has a discrete version and it allows a very first check whether a numerically-computed discrete solution of the Poisson problem is correct.

Time-dependent PDEs

- Depend on space and time

$\rightarrow$ We need an analysis and **discretization** in space and time

- The discretization in time is often based on finite difference methods as we learned in the lectures 01 - 04.
- Current mathematical-numerical research also concentrates on Galerkin space/time discretizations in which the temporal part is treated as described in lecture 05.

Parabolic PDEs: prototype heat equation

Formulation 7

Let $f : \Omega \times I \to \mathbb{R}$ and $u_0 : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that[5]

$$\partial_t u + Lu = f \quad \text{in } \Omega \times I, \tag{28}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{29}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{30}$$

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

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

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

Parabolic PDEs: prototype heat equation

Formulation 8 (Heat equation)

Setting in Formulation 7, $a_{ij} = \delta_{ij}$ and $b_i = 0$ and $c = 0$, we obtain the Laplace operator. Let $f : \Omega \to \mathbb{R}$ be given. Furthermore, Ω is an open, bounded set of $\mathbb{R}^n$. We seek the unknown function $u : \bar{\Omega} \to \mathbb{R}$ such that

$$\partial_t u + Lu = f \quad \text{in } \Omega \times I, \tag{31}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{32}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{33}$$

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

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

Hyperbolic PDEs: prototype wave equation

Formulation 9

Let $f : \Omega \times I \to \mathbb{R}$ and $u_0, v_0 : \Omega \to \mathbb{R}$ be given. We seek the unknown function $u : \bar{\Omega} \times I \to \mathbb{R}$ such that

$$\partial_t^2 u + Lu = f \quad \text{in } \Omega \times I, \tag{34}$$

$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{35}$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}, \tag{36}$$

$$\partial_t u = v_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{37}$$

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

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

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

Hyperbolic PDEs: prototype wave equation

Remark 13

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

$$\partial_t v + Lu = f \quad \text{in } \Omega \times I, \tag{38}$$
$$\partial_t u = v \quad \text{in } \Omega \times I, \tag{39}$$
$$u = 0 \quad \text{on } \partial\Omega \times [0, T], \tag{40}$$
$$u = u_0 \quad \text{on } \Omega \times \{t = 0\}, \tag{41}$$
$$v = v_0 \quad \text{on } \Omega \times \{t = 0\}. \tag{42}$$

where $Lu := -\Delta u$.

Remarks to boundary data and initial values

Boundary data:

- 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.

$\Rightarrow$ In practical real-life applications, boundary conditions are often unknown and a potential major error source.

Initial data:

- The number of initial values depends as for ODEs on the order of the time derivative
- For the heat equation, we need one initial condition $u(t_0) := u_0$
- For the wave equation, we need two initial conditions $u(t_0) := u_0, \partial_t u(t_0) := v_0$

Example temperature in a room

We consider the heat equation: Find $T : \Omega \times I \to \mathbb{R}$ such that

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

- The homogeneous Neumann condition means that there is no heat exchange on the respective walls (thus neighboring rooms will have the same room temperature on the respective walls).
- 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.
- The vector $v : \Omega \to \mathbb{R}^3$ denotes a given flow field yielding a convection of the heat, for instance wind. We can assume $v \approx 0$. Then the above equation is reduced to the original heat equation: $\partial_t T - \nabla \cdot (K \nabla T) = f$.

Brief step into numerics

- We deviate a bit and give a brief hint on the numerical discretization of time-dependent PDEs
- We recall that we need to discretize in time and space
- Three possibilities:
  1. First space, then time (method of lines)
  2. First time, then space (Rothe method)
  3. Everything together: full finite difference discretization (not recommended!) or full Galerkin approach (space-time; very elegent, but difficult to implement)
- We concentrate briefly on the second approach: Rothe method.
- Why? We can use the methods that have been presented in the lectures 02-04

Brief step into numerics: temporal discretization of the heat equation

- Let $a \in \mathbb{R}$ be a parameter that is (for simplicity) independent of the space. Let the heat equation be given: Find $u(x,t) : \Omega \times I \to \mathbb{R}$ such that

$$\begin{aligned} \partial_t u - a\Delta u &= f \quad \text{in } \Omega \times I, \\ u &= 0 \quad \text{on } \partial\Omega \times [0,T], \\ u &= u_0 \quad \text{on } \Omega \times \{t = 0\}. \end{aligned}$$

- Formal correspondance to ODEs in terms of the time t:

$$\partial_t u - a\Delta u = f$$

can be written as

$$\underbrace{\partial_t u}_{\approx y'} = \underbrace{a\Delta u + f}_{\approx f(t,y)}$$

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

Brief step into numerics (cont'd)

- Perform temporal discretization using the forward Euler scheme:

$$\frac{u_{n+1}^{\Delta t} - u_n^{\Delta t}}{\Delta t} - a\Delta u_n^{\Delta t} = f_n$$

  with $\Delta t = t_{n+1} - t_n$.

- We immediately obtain:

$$\underbrace{u_{n+1}^{\Delta t}}_{\text{unknown}} = \underbrace{u_n^{\Delta t} + \Delta t a \Delta u_n^{\Delta t} + \Delta t f_n}_{\text{known}}$$

  which is from the structure and numerical properties very known to us.

- We can **explicitly** compute the current solution $u_{n+1}^{\Delta t}$.

Brief step into numerics (cont'd)

Second example.

- Temporal discretization based on the backward Euler scheme:

$$\frac{u_{n+1}^{\Delta t} - u_n^{\Delta t}}{\Delta t} - a\Delta u_{n+1}^{\Delta t} = f_{n+1}$$

- Then, we obtain the implicit system:

$$u_{n+1}^{\Delta t} - \Delta t a \Delta u_{n+1}^{\Delta t} = u_n^{\Delta t} + \Delta t f_{n+1}$$

- This system is **implicit** because the Laplacian $\Delta u_{n+1}^{\Delta t}$ must be resolved.
- Therefore, we cannot ‘explicitly’ solve for $u_{n+1}^{\Delta t}$.

Brief step into numerics (cont'd)

Third example.

- Temporal discretization based on the trapezoidal rule (Crank-Nicolson):

$$\frac{u_{n+1}^{\Delta t} - u_n^{\Delta t}}{\Delta t} - \frac{1}{2}[a\Delta u_{n+1}^{\Delta t} + a\Delta u_n^{\Delta t}] = \frac{1}{2}[f_{n+1} + f_n]$$

- Then, we obtain the implicit system:

$$u_{n+1}^{\Delta t} - \frac{1}{2}\Delta t a\Delta u_{n+1}^{\Delta t} = u_n^{\Delta t} + \frac{1}{2}\Delta t a\Delta u_n^{\Delta t} + \Delta t\frac{1}{2}[f_{n+1} + f_n]$$

- Compare again to lecture 03 or 04 for similar terminological structures in pure ODE problems.
- Other time integration schemes (e.g., Runge-Kutta) may be used as well

Brief step into numerics (cont'd)

- As in the ODE lectures, it now depends on the character of the PDE which time-discretization scheme is best suited.
- For instance: the **heat equation is a dissipative** equation, which can be dealt with a dissipative time-discretization scheme (e.g., backward Euler).
- Of course higher-order methods yield better accuracy as seen in lecture 04 and exercise 2.
- The **wave equation conserves energy**.
- Here, a dissipative time-discretization scheme should not be used!. Also explicit schemes are not well suited because of numerical instabilities.

→ The reason is because the spatial Laplacian 'becomes big' due to the spatial discretization as we will later see.

- Consequently: from the three presented schemes, the only 'good' option is the trapezoidal rule.
- It now remains to discuss spatial discretization, which is our topic in the upcoming lecture 07-09.

Further classifications (we recall from our ODE studies)

- Order of a differential equation
- Single equations and PDE systems
- Nonlinear problems:
  - Nonlinearity in the PDE
  - The function set is not a vector space yielding a variational inequality
- Coupled problems and coupled PDE systems.

Further classifications: examples

- p-Laplace equation: Find $u : \Omega \to \mathbb{R}$:

$$-\nabla \cdot (|\nabla u|^{p-2} \nabla u)^{p/2} = f \tag{43}$$

  Properties: nonlinear (quasilinear), stationary, scalar-valued.

- Find $u : \Omega \to \mathbb{R}$:

$$-\Delta u + u^2 = f \tag{44}$$

  Properties: nonlinear (semilinear), stationary, scalar-valued.

- Incompressible, isothermal Navier-Stokes equations: Find $v : \Omega \to \mathbb{R}^n$ and $p : \Omega \to \mathbb{R}$

$$\partial_t v + (v \cdot \nabla) v - \frac{1}{Re} \Delta v + \nabla p = f, \quad \nabla \cdot v = 0 \tag{45}$$

  with Re being the Reynolds' number. For $Re \to \infty$ we obtain the Euler equations. Properties: nonlinear (semilinear), nonstationary, vector-valued, PDE system.

Further classifications: examples

- A volume-coupled problem: Find $u : \Omega \to \mathbb{R}$ and $\varphi : \Omega \to \mathbb{R}$

$$-\Delta u = f(\varphi), \tag{46}$$

$$|\nabla u|^2 - \Delta \varphi = g(u) \tag{47}$$

  Properties: nonlinear, coupled problem via right hand sides, stationary.

- An interface-coupled problem: Let Ω_1 and Ω_2 with $\Omega_1 \cap \Omega_2 = 0$ and $\bar{\Omega}_1 \cap \bar{\Omega}_2 = \Gamma$ and $\bar{\Omega}_1 \cup \bar{\Omega}_2 = \Omega$. Find $u_1 : \Omega_1 \to \mathbb{R}$ and $u_2 : \Omega_2 \to \mathbb{R}$:

$$-\Delta u_1 = f_1 \quad \text{in } \Omega_1, \tag{48}$$

$$-\Delta u_2 = f_2 \quad \text{in } \Omega_2, \tag{49}$$

$$u_1 = u_2 \quad \text{on } \Gamma, \tag{50}$$

$$\partial_n u_1 = \partial_n u_2 \quad \text{on } \Gamma \tag{51}$$

  Properties: linear, coupled problem via interface conditions, stationary.

Advanced examples (to give an outlook): elasticity

This example is already difficult because a system of nonlinear equations is considered:

Formulation 10

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

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

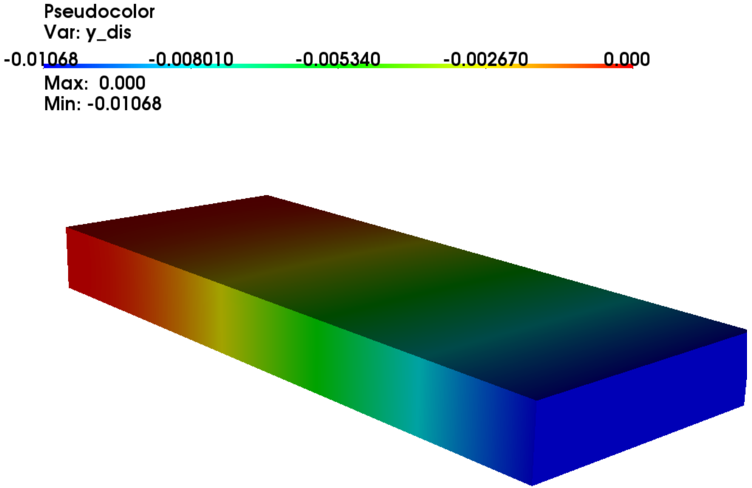
Advanced examples (to give an outlook): elasticity

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 1984):

$$\widehat{\Sigma} = \widehat{\Sigma}_s(\hat{u}_s) = 2\mu \widehat{E} + \lambda tr(\widehat{E}) I, \quad \widehat{E} = \frac{1}{2}(\widehat{F}^T \widehat{F} - I). \tag{52}$$

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.

Advanced examples (to give an outlook): elasticity



Advanced examples (to give an outlook): incompressible flow - Navier-Stokes equations

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

Advanced examples (to give an outlook): incompressible flow - Navier-Stokes equations

Formulation 11

Let $\Omega_f \subset \mathbb{R}^n, n = 3$. Furthermore, let the boundary be split into $\partial\Omega_f := \Gamma_{in} \cup \Gamma_{out} \cup \Gamma_D \cup \Gamma_i$. The isothermal, incompressible (non-linear) Navier-Stokes equations read: Find vector-valued velocities $v_f : \Omega_f \times I \to \mathbb{R}^n$ and a scalar-valued pressure $p_f : \Omega_f \times I \to \mathbb{R}$ such that

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

Advanced examples (to give an outlook): incompressible flow - Navier-Stokes equations

Here the (symmetric) Cauchy stress is given by

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

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

Advanced examples (to give an outlook): incompressible flow - Navier-Stokes equations

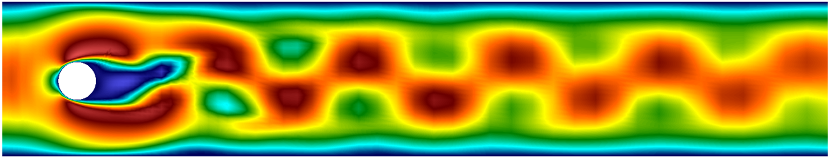


Figure: Prototype example of a fluid mechanics problem (isothermal, incompressible Navier-Stokes equations): the famous Karman vortex street. The setting is based on the benchmark setting Schaefer/Turek et al. 1996 and the code can be found in NonStat Example 1 in DOpElib `www.dopelib.net`.

Summary lecture 06

- Different types of PDEs
- Modeling and physical explanations
- Three-important PDEs
  - Poisson, heat, wave
  - Temporal discretization and the relation to the ODE lectures
- Classifications of the order, linear/nonlinear, PDE systems
- Various further (advanced) examples

Contents

Recall model problem in 1D: Poisson's problem

Find $u : \Omega \to \mathbb{R}$ such that

$$-u'' = f \quad \text{in } \Omega = (0,1), \tag{53}$$

$$u(0) = u(1) = 0. \tag{54}$$

Equivalent formulations

We first introduce the scalar product on $\Omega = (0, 1)$:

$$(v, w) = \int_0^1 v(x)w(x)\, dx.$$

Furthermore we introduce the linear space

$$V := \{v|\ v \in C[0, 1], v' \text{ is piecewise continuous and bounded on } [0, 1], v(0) = v(1) = 0\}. \tag{55}$$

We also introduce the linear functional $F : V \to \mathbb{R}$ such that

$$F(v) = \frac{1}{2}(v', v') - (f, v).$$

Equivalent formulations

Definition 21

We deal with three (equivalent) problems:

(D) Find $u \in C^2$ such that $-u'' = f$ with $u(0) = u(1) = 0$;

(M) Find $u \in V$ such that $F(u) \leq F(v)$ for all $v \in V$;

(V) Find $u \in V$ such that $(u', v') = (f, v)$ for all $v \in V$.

- In physics, the quantity $F(v)$ stands for the **total potential energy** of the underlying model.
- Moreover, the first term in $F(v)$ denotes the internal elastic energy and (f, v) the load potential.
- Therefore, formulation (M) corresponds to the fundamental **principle of minimal potential energy** and the variational problem (V) to the **principle of virtual work** (e.g., Ciarlet 1984).
- The proofs of their equivalence will be provided in the following.

Equivalent formulations

Proposition 14

It holds

$$(D) \to (V).$$

Proof.

We multiply $-u'' = f$ with a function ϕ (a so-called **test function**) from the space V defined in (55). Then we integrate over the interval $\Omega = (0,1)$ yielding

$$-u'' = f \tag{56}$$

$$\Rightarrow -\int_\Omega u''\phi \, dx = \int_\Omega f\phi \, dx \tag{57}$$

$$\Rightarrow \int_\Omega u'\phi' \, dx - u'(1)\phi(1) + u'(0)\phi(0) = \int_\Omega f\phi \, dx \tag{58}$$

$$\Rightarrow \int_\Omega u'\phi' \, dx = \int_\Omega f\phi \, dx \quad \forall \phi \in V. \tag{59}$$

Equivalent formulations

In the second last term, we used integration by parts.
The boundary terms vanish because $\phi \in V$. This shows that

$$\int_\Omega u' \phi' \, dx = \int_\Omega f \phi \, dx$$

is a solution of (V).

Remark 15

The technique used in this proof is of paramount importance since the integration by parts is THE standard trick in the finite element method.

Equivalent formulations

Proposition 16

It holds

$$(V) \leftrightarrow (M).$$

Proof.

We first assume that u is a solution to (V). Let $\phi \in V$ and set $w = \phi - u$ such that $\phi = u + w$ and $w \in V$. We obtain

$$\begin{aligned} F(\phi) = F(u+w) &= \frac{1}{2}(u' + w', u' + w') - (f, u + w) \\ &= \frac{1}{2}(u', u') - (f, u) + (u', w') - (f, w) + \frac{1}{2}(w', w') \geq F(u) \end{aligned}$$

We use now the fact that (V) holds true, namely

$$(u', w') - (f, w) = 0.$$

Equivalent formulations

And also that $(w', w') \geq 0$. Thus, we have shown that u is a solution to (M). We show now that $(M) \to (V)$ holds true as well. For any $\phi \in V$ and $\varepsilon \in \mathbb{R}$ we have

$$F(u) \leq F(u + \varepsilon\phi),$$

because $u + \varepsilon\phi \in V$. We differentiate with respect to ε and show that (V) is a first order necessary condition to (M) with a minimum at $\varepsilon = 0$. To do so, we define

$$g(\varepsilon) := F(u + \varepsilon\phi) = \frac{1}{2}(u', u') + \varepsilon(u', \phi') + \frac{\varepsilon^2}{2}(\phi', \phi') - (f, u) - \varepsilon(f, \phi).$$

Equivalent formulations

Thus

$$g'(\varepsilon) = (u', \phi') + \varepsilon(\phi', \phi') - (f, \phi).$$

A minimum is obtained for $\varepsilon = 0$. Consequently,

$$g'(0) = 0.$$

In detail:

$$(u', \phi') - (f, \phi) = 0,$$

which is nothing else than the solution of (V).

Equivalent formulations

Proposition 17

It holds

$$(V) \to (D).$$

Proof. We assume that u is a solution to (V), i.e.,

$$(u', \phi') = (f, \phi) \quad \forall \phi \in V.$$

If we assume sufficient regularity of u (in particular $u \in C^2$), then u'' exists and we can integrate backwards. Moreover, we use that $\phi(0) = \phi(1) = 0$ since $\phi \in V$. Then:

$$(-u'' - f, \phi) = 0 \quad \forall \phi \in V.$$

Equivalent formulations

Since we assumed sufficient regularity for u'' and f the difference is continuous. We can now apply the fundamental principle (see Proposition 18):

$$w \in C(\Omega) \quad \Rightarrow \quad \int_\Omega w\phi \, dx = 0 \quad \Rightarrow \quad w \equiv 0.$$

We proof this result later. Before, we obtain

$$(-u'' - f, \phi) = 0 \quad \Rightarrow \quad -u'' - f = 0,$$

which yields the desired expression. Since we know that $(D) \to (V)$ holds true, u has the assumed regularity properties and we have shown the equivalence.

Fundamental lemma of calculus of variations

Proposition 18

Let $\Omega = [a, b]$ be a compact interval and let $w \in C(\Omega)$. Let $\phi \in C^\infty$ with $\phi(a) = \phi(b) = 0$, i.e., $\phi \in C_c^\infty(\Omega)$. If for all ϕ it holds

$$\int_\Omega w(x)\phi(x)\, dx = 0,$$

then, $w \equiv 0$ in Ω.

Proof.

We perform an indirect proof. We suppose that there exist a point $x_0 \in \Omega$ with $w(x_0) \neq 0$. Without loss of generality, we can assume $w(x_0) > 0$. Since w is continuous, there exists a small (open) neighborhood $\omega \subset \Omega$ with $w(x) > 0$ for all $x \in \omega$; otherwise $w \equiv 0$ in $\Omega \setminus \omega$.

Proof continued.

Let ϕ now be a positive test function (recall that ϕ can be arbitrary, specifically positive if we wish) in Ω and thus also in ω. Then:

$$\int_\Omega w(x)\phi(x)\,dx = \int_\omega w(x)\phi(x)\,dx.$$

But this is a contradiction to the hypothesis on w. Thus $w(x) = 0$ for all in $x \in \omega$. Extending this result to all open neighborhoods in Ω we arrive at the final result.

Remark 19

The general form of the proof can be found in P. Ciarlet; 2013: Linear and nonlinear functional analysis with applications.

Derivation of a weak (variational) form: Step 1

Two-step procedure:

- Step 1: Design a function space V that also includes the correct boundary conditions
- Step 2: Multiply with a test function from V and integrate

For Poisson with homogeneous Dirichlet conditions, we then obtain:

- Take space

$$V := \{v|\ v \in C[0,1], v' \text{ is pc. cont. and bound. on } [0,1], v(0) = v(1) = 0\}$$

from before.

Derivation of a weak (variational) form: Step 2

- Multiply with test function $\phi \in V$ and integrate:

$$-u'' = f \tag{60}$$

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

$$\Rightarrow \int_\Omega u'\phi'\,dx - \int_{\partial\Omega} \partial_n u\phi\,ds = \int_\Omega f\phi\,dx \tag{62}$$

$$\Rightarrow \int_\Omega u'\phi'\,dx = \int_\Omega f\phi\,dx. \tag{63}$$

To summarize we have:

Formulation 12
Find $u \in V$ such that

$$\int_\Omega u'\phi'\,dx = \int_\Omega f\phi\,dx \quad \forall\phi \in V. \tag{64}$$

Derivation of a weak (variational) form

A common short-hand notation in mathematics is to use parentheses for L^2 scalar products: $\int_\Omega ab\,dx =: (a,b)$:

$$(u', \phi') = (f, \phi) \tag{65}$$

A mathematically-correct statement is:

Formulation 13
Find $u \in V$ such that

$$(u', \phi') = (f, \phi) \quad \forall \phi \in V. \tag{66}$$

In the following, we introduce some tools from functional analysis that are required to analyze further the variational form.

Hilbert spaces

Definition 22 (Hilbert space)

A complete space endowed with an inner product is called a **Hilbert space**. The norm is defined by

$$\|u\| := \sqrt{(u,u)}.$$

Example 20

The space $\mathbb{R}^n$ from before has a scalar product and is complete, thus a Hilbert space. The space $\{C(\Omega), \|\cdot\|_{L^2}\}$ has a scalar product, but is not complete, and therefore not a Hilbert space. The space $\{C(\Omega), \|\cdot\|_{C(\Omega)}\}$ is complete, but the norm is not induced by a scalar product and is therefore not a Hilbert space, but only a Banach space.

Hilbert spaces: L_2

Definition 23 (The L^2 space in 1D)

Let $\Omega = (a, b)$ be an interval (recall 1D Poisson). The space of square-integrable functions on Ω is defined by

$$L^2(\Omega) = \{v : \int_\Omega v^2\, dx < \infty\}$$

The space L^2 is a Hilbert space equipped with the scalar product

$$(v, w) = \int_\Omega vw\, dx$$

and the induced norm

$$\|v\|_{L^2} := \sqrt{(v, v)}.$$

Hilbert spaces: L_2

Using Cauchy's inequality

$$|(v, w)| \leq \|v\|_{L^2} \|w\|_{L^2},$$

we observe that the scalar product is well-defined when $v, w \in L^2$. A mathematically very correct definition must include in which sense (Riemann or Lebesgue) the integral exists. In general, all L spaces are defined in the sense of the Lebuesgue integral (see for instance books introducing Lebesgue spaces).

Hilbert spaces: H^1

Definition 24 (The H^1 space in 1D)

We define the $H^1(\Omega)$ space with $\Omega = (a, b)$ as

$$H^1(\Omega) = \{v : \ v \text{ and } v' \text{ belong to } L^2\}$$

This space is equipped with the following scalar product:

$$(v, w)_{H^1} = \int_\Omega (vw + v'w')\, dx$$

and the norm

$$\|v\|_{H^1} := \sqrt{(v, v)_{H^1}}.$$

Hilbert spaces: H_0^1

Definition 25 (The H_0^1 space in 1D)

We define the $H_0^1(\Omega)$ space with $\Omega = (a, b)$ as

$$H_0^1(\Omega) = \{v \in H^1(\Omega) : \; v(a) = v(b) = 0\}.$$

The scalar product is the same as for the H^1 space.

Well-posedness: existence, uniqueness and stability

Formulation 14 (Abstract model problem)

Let V be a Hilbert space with norm $||\cdot||_V$. Find $u \in V$ such that

$$a(u, \phi) = l(\phi) \quad \forall \phi \in V.$$

with

$$a(u, \phi) := (u', \phi'),$$
$$l(\phi) := (f, \phi)$$

Well-posedness: existence, uniqueness and stability

Definition 26 (Assumptions)

We suppose:

1. $l(\cdot)$ is a bounded linear form:
$$|l(u)| \leq C\|u\| \quad \text{for all } u \in V.$$
2. $a(\cdot,\cdot)$ is a bilinear form on $V \times V$ and continuous:
$$|a(u,v)| \leq \gamma\|u\|_V\|v\|_V, \quad \gamma > 0, \quad \forall u, v \in V.$$
3. $a(\cdot,\cdot)$ is coercive (or V-elliptic):
$$a(u,u) \geq \alpha\|u\|_V^2, \quad \alpha > 0, \quad \forall u \in V.$$

The Lax-Milgram lemma

Lemma 27 (Lax-Milgram)

Let $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ be a continuous, V-elliptic bilinear form. Then, for each $l \in V^*$ the variational problem

$$a(u,\phi) = l(\phi) \quad \forall \phi \in V$$

has a unique solution $u \in V$. Moreover, we have the stability estimate:

$$\|u\| \leq \frac{1}{\alpha}\|l\|_{V^*}.$$

with

$$\|l\|_{V^*} := \sup_{\varphi \neq 0} \frac{|l(\varphi)|}{||\varphi||_V}.$$

The proof can be found in P. Ciarlet; 2013.

The energy norm

The continuity and coercivity of the bilinear form yield the **energy norm**:

$$\|v\|_a^2 := a(v, v), \quad v \in V.$$

This norm is aquivalent to the V-norm of the space V, i.e.,

$$c\|v\|_V \leq \|v\|_a \leq C\|v\|_V, \quad \forall v \in V$$

and two positive constants c and C. We can even precisely determine these two constants:

$$\alpha||u||_V^2 \leq a(u, u) \leq \gamma\|u\|_V^2$$

yielding $c = \sqrt{\alpha}$ and $C = \sqrt{\gamma}$. The corresponding scalar product is defined by

$$(v, w)_a := a(v, w).$$

The energy norm: example

For the Poisson problem, the energy norm reads:

- Given $a(v,v) = (v',v') = \int_\Omega (v'(x))^2\,dx$
- Then:

$$\|v\|_a^2 = \int_\Omega (v'(x))^2\,dx.$$

- The energy norm is the 'natural' norm to measure results of Poisson's problem
- For instance: a computational convergence analysis (see lecture 04), could be done with the energy norm
- Moreover, the energy norm measures indeed the 'physical' energy of the given system

Summary of lecture 07

- Equivalent formulations
- Derivation of a weak form from a strong form
- Hilbert spaces
- Well-posedness of linear, stationary PDEs: Lax-Milgram lemma
- Energy norm: natural norm for the Laplace operator

Exercise 3

Let $\alpha \in \mathbb{R}$. We are given the Poisson problem in 1D on the interval $\Omega = (0,1)$:

$$-\alpha u''(x) = f \quad \text{in } \Omega$$
$$u(0) = u(1) = 0$$

and $\alpha = 1$ and the right hand side $f = -a$ with $a > 0$. The code of this example can be found here:

`https://cloud.ifam.uni-hannover.de/index.php/s/Cwe4ZqwLRMixS3J`

with the password that is known to you.

Remark 21

Please be careful that the above form is only correct when α is constant. The general formulation is

$$-\frac{d}{dx}(\alpha u')$$

which reduces to the above one, when α is constant.

Exercise 3

Please work on the following tasks:

1. Please run the code and observe the results using gnuplot.
   Hint: Please work in the optimized compiling mode
2. We play now with three parameters:
   1. Please vary the **discretization parameter** h and use other parameters. What do you observe?
   2. Vary now the **model parameter** α. What do you observe?
   3. Choose now a different **right hand side** f. What do you observe?
3. Check if the maximum principle holds true.
4. We study in this final task the structure of the code. Go into the code and try to understand the different functions and methods that are implemented therein. Please have a specific look into the `assemble_system` method.

Contents

Recalling exercise 3 and goals for today

- Yesterday in the exercise, we computed Poisson in 1D
- The computer solved for us everything and no programming was necessary
- Today we investigate what the computer really did for us
- Also we will see why the computer became 'slow' when the mesh size parameter h is small

$\rightarrow$ Goal: we discuss and implement the spatial discretization using a Galerkin finite element scheme

- FEM = Finite Element Method
- Such a scheme is based on the variational formulation introduced in lecture 07

Finite elements in 1D

In the following we want to concentrate how to compute a **discrete solution** for Poisson's problem using a Galerkin finite element method The principle of the FEM is as follows:

- 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 $K_i := [x_i, x_{i+1}], i = 0, \ldots, n$ polynomials for trial and test functions. These polynomials must form a basis in a space V_h and they should reflect certain conditions on the mesh edges;
- Use the variational form of the given problem and derive a discrete version;

Finite elements in 1D

- Evaluate the arising integrals;
- Collect all contributions on all K_i leading to a linear equation system $Az = b$;
- Solve this linear equation system; the solution vector $z = (z_1, \dots, z_n)^T$ contains the discrete solution at the nodal points $z_1, \dots, z_n$;
- Verify the correctness of the solution z.

The mesh

Let us start with the mesh. We introduce nodal points and divide $\Omega = (0,1)$ into

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

In particular, we can work with a uniform mesh in which all nodal points have equidistant distance:

$$x_j = jh, \quad h = \frac{1}{n+1}, \quad 0 \le j \le n+1, \quad h = x_{j+1} - x_j.$$

Remark 22

An important research topic is to organize the points x_j in certain non-uniform ways in order to reduce the discrete error. This procedure is called adaptivity.

Linear finite elements

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

Definition 28

$$P_k := \{\sum_{i=0}^{k} a_i x^i | \; a_i \in \mathbb{R}\}.$$

In particular we will work with the space of linear polynomials

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

Linear finite elements

- A finite element is now a function localized to an element $K_i \in \mathcal{T}_h$ and uniquely defined by the values in the nodal points x_i, x_{i+1}.
- We then define the space:

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

- This space $V_h^{(1)}$ is a finite-dimensional realization of the space V from lecture 07
- The boundary conditions are build into the space through $v(0) = v(1) = 0$. This is an important concept that Dirichlet boundary conditions will not appear explicitly later, but are contained in the function spaces.
- All functions inside V_h are so called **shape functions** and can be represented by so-called **hat functions**. Hat functions are specifically linear functions on each element K_i. Attaching them yields a hat in the geometrical sense.

Hat functions

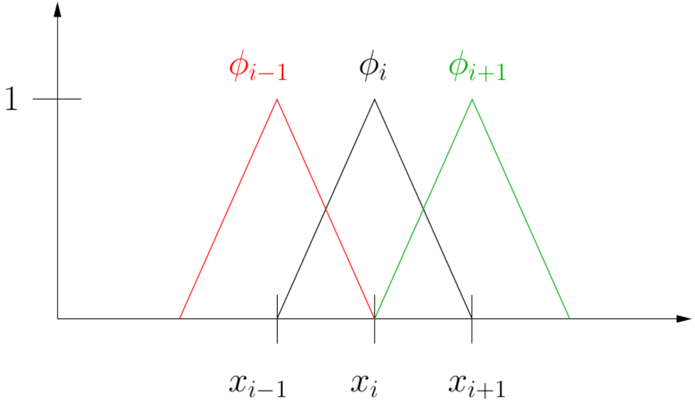


Figure: Hat functions. Linear finite elements in 1D.

Construction of hat functions

For $j = 1, \ldots, n$ we define:

$$\phi_j(x) = \begin{cases} 0 & \text{if } x \notin [x_{j-1}, x_{j+1}] \\ \frac{x - x_{j-1}}{x_j - x_{j-1}} & \text{if } x \in [x_{j-1}, x_j] \\ \frac{x_{j+1} - x}{x_{j+1} - x_j} & \text{if } x \in [x_j, x_{j+1}] \end{cases} \tag{67}$$

with the property

$$\phi_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{68}$$

Construction of hat functions

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

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

and for its derivative:

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

Conforming finite elements

Lemma 29

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:*

$$v_h(x) = \sum_{j=1}^{n} z_j \phi_j(x) \quad \forall x \in [0,1], \quad z_j \in \mathbb{R}.$$

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. $\square$

Lagrange finite elements

Remark 23

The finite element method introduced above is a Lagrange method, since the basis functions ϕ_j are defined only through its values at the nodal points without using derivative information (which would result in Hermite polynomials).

The process to construct the specific form of the shape functions

- In the previous construction, we have hidden the process how to find the specific form of $\phi_j(x)$. For 1D it is more or less clear and we would accept the $\phi_j(x)$ really has the form as previously described.
- In $\mathbb{R}^n$ this task is a bit of work. To understand this procedure, we explain the process in detail. Here we first address the defining properties of a finite element:
  - Intervals $[x_i, x_{i+1}]$;
  - A linear polynomial $\phi(x) = a_0 + a_1 x$;
  - Nodal values at x_i and x_{i+1} (the so-called degrees of freedom).
- The main task consists in finding the unknown coefficients a_0 and a_1 of the shape function.

The process to construct the specific form of the shape functions

The key property is (68) (also valid in $\mathbb{R}^n$ in order to have a small support) and therefore we obtain:

$$\phi_j(x_j) = a_0 + a_1 x_j = 1,$$
$$\phi_j(x_i) = a_0 + a_1 x_i = 0.$$

To determine a_0 and a_1 we have to solve a small linear equation system:

$$\begin{pmatrix} 1 & x_j \\ 1 & x_i \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

We obtain

$$a_1 = -\frac{1}{x_i - x_j}$$

and

$$a_0 = \frac{x_i}{x_i - x_j}.$$

The process to construct the specific form of the shape functions

Then:

$$\phi_j(x) = a_0 + a_1 x = \frac{x_i - x}{x_i - x_j}.$$

At this stage we have now to distinguish whether $x_j := x_{i-1}$ or $x_j := x_{i+1}$ or $|i-j| > 1$ yielding the three cases in (67).

Remark 24

Of course, for higher-order polynomials and higher-order problems in $\mathbb{R}^n$, the matrix system to determining the coefficients becomes larger. However, in all these state-of-the-art FEM software packages, the shape functions are already implemented.

Remark 25

A very practical and detailed derivation of finite elements in different dimensions can be found in the book of Schwarz 1989.

The discrete weak form

Now, we use the **variational formulation** (lecture 07) and derive the discrete counterpart:

Formulation 15

Find $u_h \in V_h$ such that

$$(u_h', \phi_h') = (f, \phi_h) \quad \forall \phi_h \in V_h. \tag{69}$$

Or in the previously introduced compact form:

Formulation 16 (Variational Poisson problem on the discrete level)

Find $u_h \in V_h$ such that

$$a(u_h, \phi_h) = l(\phi_h) \quad \forall \phi_h \in V_h, \tag{70}$$

where $a(\cdot,\cdot)$ and $l(\cdot)$ are defined as

$$a(u_h, \phi_h) := (u_h', \phi_h'), \quad \text{and} \quad l(\phi_h) := (f, \phi_h)$$

Galerkin, Ritz, etc.

Remark 26 (Galerkin method)

The process going from V to V_h using the variational formulation is called **Galerkin method**. Here, it is not necessary that the bilinear form is symmetric. As further information: not only is Galerkin's method a numerical procedure, but it is also used in analysis when establishing existence of the continuous problem. Here, one starts with a finite dimensional subspace and constructs a sequence of finite dimensional subspaces $V_h \subset V$ (namely passing with $h \to 0$; that is to say: we add more and more basis functions such that $dim(V_h) \to \infty$). The idea of numerics is the same: finally we are interested in small h such that we obtain a discrete solution with sufficient accuracy.

Galerkin, Ritz, etc.

Remark 27 (Ritz method)

If we discretize the minimization problem (M), the above process is called **Ritz method**. In particular, the bilinear form of the variational problem is symmetric.

Remark 28 (Ritz-Galerkin method)

For general bilinear forms (i.e., not necessarily symmetric) the discretization procedure is called **Ritz-Galerkin method**.

Remark 29 (Petrov-Galerkin method)

In a **Petrov-Galerkin method** the trial and test spaces can be different.

Constructing the solution of the discrete system

We recall and plan:

- Variational form in space V: infinite-dimensional problem

$\rightarrow$ Cannot be solved with the computer!

- Discrete counterpart $V_h \subset V$: finite-dimensional problem
- In this finite dimensional space, any vector $v_h \in V_h$ can be represented using a linear combination of the basis functions
- What do we need to find our solution $u_h \in V_h$?

$\rightarrow$ Insert representation of u_h into discrete problem and determine the solution coefficients.

- These solution coefficients are exactly the values (because we work with Lagrange finite elements) in the support points of the mesh that we want to know:

x_1	x_2	$\dots$	x_{n-1}	x_n	mesh points
z_1	z_2	$\dots$	z_{n-1}	z_n	solution coefficients of u_h

Determining the solution coefficients

Realizing the plan from the previous slide:

- We express $u_h \in V_h$ with the help of the basis functions ϕ_j in $V_h := \{\phi_1, \ldots, \phi_n\}$, thus:

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

- Since (69) holds for all $\phi_i \in V_h$ for $1 \leq i \leq n$, it holds in particular for each i:

$$(u_h', \phi_i') = (f, \phi_i) \quad \text{for } 1 \leq i \leq n. \tag{71}$$

Determining the solution coefficients

- We now insert the representation for u_h in (71), yielding the **Galerkin equations**:

$$\underbrace{\sum_{j=1}^{n} z_j}_{=z} \underbrace{(\phi_j', \phi_i')}_{=A} = \underbrace{(f, \phi_i)}_{=b} \quad \text{for } 1 \leq i \leq n. \tag{72}$$

- We have now extracted the coefficient vector $(z_1, z_2, \ldots, z_n)$ (neglecting the index h for convenience) of u_h and only the **shape functions** ϕ_j and ϕ_i (i.e., their derivatives of course) remain in the integral.

The resulting linear equation system

This yields a linear equation system of the form

$$Az = b$$

where

$$z = (z_j)_{1 \leq j \leq n} \in \mathbb{R}^n, \tag{73}$$
$$b = ((f, \phi_i))_{1 \leq i \leq n} \in \mathbb{R}^n, \tag{74}$$
$$A = \left((\phi_j', \phi_i')\right)_{1 \leq j, i \leq n} \in \mathbb{R}^{n \times n}. \tag{75}$$

Thus the final solution vector is z containing the values z_j at the nodal points x_j of the mesh.

Remark 30

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

Evaluating the integrals

What remains is to evaluate the integrals:

$$(\phi_j', \phi_i')$$

and

$$(f, \phi_i)$$

More details in lecture 09.

Remarks on $Az = b$

Remark 31 (Regularity of A)

It remains the question whether A is regular such that A^{-1} exists. With the help of linear algebra arguments this can be shown.

Remark 32

A bottleneck in computational cost (run time of the computer) is the solution of the system $Az = b$. For big A (namely small mesh sizes h), the computational cost becomes huge. More in lecture 09.

Numerical test: 1D Poisson (see Exercise 3)

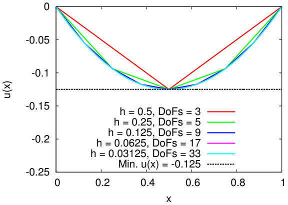


Figure: Solution of the 1D Poisson problem with $f = -1$ using finite elements with various mesh sizes h. DoFs is the abbreviation for degrees of freedom; here the number of support points x_j. The dimension of the discrete space is $DoFs$. For instance for $h = 0.5$, we have 3 DoFs and two basis functions, thus $dim(V_h) = 3$. Please notice that the picture norm is not a proof in the strict mathematical sense: to show that the purple, and blue lines come closer and closer must be confirmed by error estimates as presented. Of course, for this 1D Poisson problem, we easily observe a limit case, but for more complicated equations it is often not visible whether the solutions do converge.

Definition of a finite element

We briefly summarize the key ingredients that define a **finite element**. A finite element is a triple (K, P_K, Σ) where

- K is an element, i.e., a geometric object (in 1D an interval);
- $P_k(K)$ is a finite dimensional linear space of polynomials defined on K;
- Σ, not introduced so far, is a set of degrees of freedom (DoF), e.g., the values of the polynomial at the vertices of K.

These three ingredients yield a uniquely determined polynomial on an element K.

Numerical analysis: outline

- Step 1: Approximation estimates (qualitative; no convergence rates in terms of h powers yet
- Step 2: Interpolation estimates (yielding local h powers)
- Step 3: Convergence results (yielding global h powers)

Galerkin orthogonality: the first approximation result

We have

$$
\begin{aligned}
(u', \phi') &= (f, \phi) \quad \forall \phi \in V, \\
(u_h', \phi_h') &= (f, \phi_h) \quad \forall \phi_h \in V_h.
\end{aligned}
$$

Taking in particular only discrete test functions from $V_h \subset V$ and subtraction of both equations yields:

Proposition 33 (Galerkin orthogonality)

It holds:

$$((u - u_h)', \phi_h) = 0 \quad \forall \phi_h \in V_h,$$

or in the more general notation:

$$a(u - u_h, \phi_h) = 0 \quad \forall \phi_h \in V_h.$$

Galerkin orthogonality: illustration

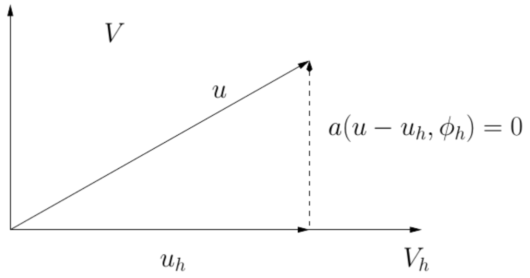


Figure: Illustration of Galerin orthogonality.

- The error measured in the energy norm (defined in lecture 07) stands orthogonal on the discrete space V_h
- For this reason, it holds the best approximation property

Galerkin orthogonality: proof

Proof.
Taking $\phi_h \in V_h$ in both previous equations yields:

$$(u', \phi') - (u_h', \phi_h') = (f, \phi) - (f, \phi_h).$$

Taking both equations in the discrete space V_h means $\phi := \phi_h$ (is no problem since $V_h \subset V$) and with that

$$(f, \phi_h) - (f, \phi_h) = 0.$$

$\square$

Step 1. Céa lemma: a first approximation result

Proposition 34 (Céa lemma)

Let V be a Hilbert space and $V_h \subset V$ a finite dimensional subspace. Let the assumptions of the Lax-Milgram Lemma hold true. Let $u \in V$ and $u_h \in V_h$ be the solutions of the variational problems. Then:

$$\|u - u_h\|_V = \frac{\gamma}{\alpha} \inf_{\phi_h \in V_h} \|u - \phi_h\|_V$$

Step 1. Céa lemma: a first approximation result

Proof.
It holds Galerkin orthogonality:

$$a(u-u_h, w_h) = 0 \quad \forall w_h \in V_h.$$

We choose $w_h := u_h - \phi_h$ and we obtain:

$$\alpha \|u-u_h\|^2 \le a(u-u_h, u-u_h) = a(u-u_h, u-\phi_h) \le \gamma \|u-u_h\| \|u-\phi_h\|.$$

This yields

$$\|u-u_h\| \le \frac{\gamma}{\alpha} \|u-\phi_h\|.$$

Passing to the infimum yields:

$$\|u-u_h\| = \inf_{\phi_h \in V_h} \frac{\gamma}{\alpha} \|u-\phi_h\|.$$

□

Consequences of Céa

Proposition 35

We assume that the hypotheses from before hold true. Furthermore, we assume that $U \subset V$ is dense. We construct an interpolation operator $i_h : U \to V_h$ such that

$$\lim_{h\to 0} \|v - i_h(v)\| = 0 \quad \forall v \in U$$

holds true. Then, for all $u \in V$ and $u_h \in V_h$:

$$\lim_{h\to 0} \|u - u_h\| = 0.$$

This result shows that the Galerkin solution $u_h \in V_h$ converges to the continuous solution u.

Consequences of Céa

Proof.

Let $\varepsilon > 0$. Thanks to the density, for each $u \in V$, it exists a $v \in U$ such that $\|u - v\| \leq \varepsilon$. Moreover, there is an $h_0 > 0$, depending on the choice of ε, such that

$$\|v - i_h(v)\| \leq \varepsilon \quad \forall h \leq h_0.$$

The Céa lemma yields now:

$$\begin{aligned} \|u - u_h\| &\leq C\|u - i_h(v)\| = C(\|u - v + v - i_h(v)\|) \\ &\leq C(\|u - v\| + \|v - i_h(v)\|) \leq C(\varepsilon + \varepsilon) = 2C\varepsilon. \end{aligned}$$

These proofs employs the trick, very often seen for similar calculations, that at an appropriate place the 'right' function, here v is inserted, and the terms are split thanks to the triangular inequality. Afterwards, the two separate terms can be estimated using the assumptions of other known results. □

Step 2. Interpolation estimates in H^1 and L^2 in 1D

First, we need to construct an interpolation operator in order to approximate the continuous solution at certain nodes.

Definition 30 (Interpolation operator)

Let $\Omega = (0,1)$. A P_1 interpolation operator $i_h : H^1 \to V_h$ is defined by

$$(i_h v)(x) = \sum_{j=0}^{n+1} v(x_j)\phi_j(x) \quad \forall v \in H^1.$$

This definition is well-defined since H^1 functions are continuous in 1D and are pointwise defined. The interpolation i_h creates a piece-wise linear function that coincides in the support points x_j with its H^1 function.

Step 2. H^1 and L^2 interpolation estimates in 1D

The convergence of a finite element method in 1D relies on

Lemma 31

Let $i_h : H^1 \to V_h$ be given. Then:

$$\lim_{h\to 0} \|u - i_h u\|_{H^1} = 0.$$

If $u \in H^2$, there is a constant C such that

$$\|u - i_h u\|_{H^1} \leq Ch|u|_{H^2}.$$

Proof.

Since

$$\|u - i_h u\|_{H^1}^2 = \|u - i_h u\|_{L^2}^2 + |u - i_h u|_{H^1}^2,$$

the result follows immediately from the next two lemmas; namely Lemma 32 and Lemma 33. □

Step 2. H^1 and L^2 estimates in 1D

Lemma 32
For a function $u \in H^2$, it exists a constant C (independent of h) such that

$$\|u - i_h u\|_{L^2} \leq Ch^2 \|u''\|_{L^2},$$
$$|u - i_h u|_{H^1} \leq Ch \|u''\|_{L^2}.$$

For the proof see: T. Wick; Numerical methods for partial differential equations: http://www.thomaswick.org/links/lecture_notes_Numerics_PDEs_Oct_12_2019.pdf

Step 2. H^1 and L^2 estimates in 1D

Lemma 33

There exists a constant C (independent of h) such that for all $u \in H^1(\Omega)$, it holds

$$\|i_h u\|_{H^1} \leq C\|u\|_{H^1}$$

and

$$\|u - i_h u\|_{L^2} \leq Ch|u|_{H^1}.$$

Moreover:

$$\lim_{h\to\infty} \|u' - i_h u'\|_{L^2} = 0.$$

For the proof see again: T. Wick; Numerical methods for partial differential equations: `http://www.thomaswick.org/links/lecture_notes_Numerics_PDEs_Oct_12_2019.pdf`

Step 3a. Convergence in H^1

Theorem 34

Let $u \in H_0^1$ and $u_h \in V_h$ be the solutions of the continuous and discrete Poisson problems. Then, the finite element method using linear shape functions converges:

$$\lim_{h \to 0} \|u - u_h\|_{H^1} = 0.$$

Moreover, if $u \in H^2$ (for instance when $f \in L^2$ and in higher dimensions when the domain is sufficiently smooth or polygonal and convex), we have

$$\|u - u_h\|_{H^1} \leq Ch\|u''\|_{L^2} = Ch\|f\|_{L^2}.$$

Thus the convergence in the H^1 norm (the energy norm) is linear and depends continuously on the problem data.

Step 3a. Convergence in H^1

Proof.
The first part is proven by using Lemma 31 applied to Proposition 35, which yields the first part of the assertion. The estimate is based on the Céa lemma:

$$\|u - u_h\|_{H^1} \leq C\|u - \phi_h\| \leq C\|u - i_h u\|_{H^1} \leq Ch|u|_{H^2} = O(h).$$

In the last estimate, we used again Lemma 31. $\square$

Step 3b. Convergence in L^2

Corollary 35

We have

$$\|u - u_h\|_{L^2} \leq Ch\|u''\|_{L^2} = Ch\|f\|_{L^2} = O(h).$$

Proof.

Follows immediately from

$$\|u - u_h\|_{H^1} \leq Ch\|u''\|_{L^2} = Ch\|f\|_{L^2},$$

and then applying the Poincaré inequality to the left hand side term. □

Remark 36

Here the L^2 estimate seems to have order h. It can be shown with the Aubin-Nitsche trick, that L^2 is one order better than H^1.

Numerical tests and computational convergence analysis

Checking programming code and convergence analysis for linear and quadratic FEM:

Algorithm 37

Given a PDE problem. E.g. $-\Delta u = f$ in Ω and $u = 0$ on the boundary $\partial\Omega$.

1. Construct by hand a solution u that fulfills the boundary conditions.
2. Insert u into the PDE to determine f.
3. Use that f in the finite element simulation to compute numerically u_h.
4. Compare $u - u_h$ in relevant norms (e.g., L^2, H^1).
5. Check whether the desired h powers can be obtained for small h.

Example

We demonstrate the previous algorithm for a $2D$ case in $\Omega = (0, \pi)^2$:

$$\begin{aligned} -\Delta u(x,y) &= f \quad \text{in } \Omega, \\ u(x,y) &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

and we constuct $u(x,y) = \sin(x)\sin(y)$, which fulfills the boundary conditions (trivial to check! But please do it!). Next, we compute the right hand side f:

$$-\Delta u = -(\partial_{xx} u + \partial_{yy} u) = 2sin(x)sin(y) = f(x,y).$$

2D Poisson: linear FEM

Level	Elements	DoFs	h	L2 err	H1 err
2	16	25	1.11072	0.0955104	0.510388
3	64	81	0.55536	0.0238811	0.252645
4	256	289	0.27768	0.00597095	0.126015
5	1024	1089	0.13884	0.00149279	0.0629697
6	4096	4225	0.06942	0.0003732	0.0314801
7	16384	16641	0.03471	9.33001e-05	0.0157395
8	65536	66049	0.017355	2.3325e-05	0.00786965
9	262144	263169	0.00867751	5.83126e-06	0.00393482
10	1048576	1050625	0.00433875	1.45782e-06	0.00196741
11	4194304	4198401	0.00216938	3.64448e-07	0.000983703

- The elements are $K_i, i = 0, \dots, n$
- The DOFs represent the number of nodal points $x_i, i = 0, \dots, n+1$

2D Poisson: quadratic FEM

Level	Elements	DoFs	h	L2 err	H1 err
2	16	81	1.11072	0.00505661	0.0511714
3	64	289	0.55536	0.000643595	0.0127748
4	256	1089	0.27768	8.07932e-05	0.00319225
5	1024	4225	0.13884	1.01098e-05	0.000797969
6	4096	16641	0.06942	1.26405e-06	0.000199486
7	16384	66049	0.03471	1.58017e-07	4.98712e-05
8	65536	263169	0.017355	1.97524e-08	1.24678e-05
9	262144	1050625	0.00867751	2.46907e-09	3.11694e-06
10	1048576	4198401	0.00433875	3.08687e-10	7.79235e-07
11	4194304	16785409	0.00216938	6.14696e-11	1.94809e-07

1D Poisson: linear FEM

We continue our studies for the 1D Poisson problem. As manufactured solution we use

$$u(x) = \frac{1}{2}(-x^2 + x),$$

on $\Omega = (0,1)$ with $u(0) = u(1) = 0$. In the middle point, we have $u(0.5) = -0.125$ (in theory and simulations). In the following table, we plot the L^2 and H^1 error norms, i.e., $\|u - u_h\|_X$ with $X = L^2$ and $X = H^1$, respectively.

1D Poisson: linear FEM

Level	Elements	DoFs	h	L2 err	H1 err
1	2	3	0.5	0.0228218	0.146131
2	4	5	0.25	0.00570544	0.072394
3	8	9	0.125	0.00142636	0.0361126
4	16	17	0.0625	0.00035659	0.0180457
5	32	33	0.03125	8.91476e-05	0.00902154
6	64	65	0.015625	2.22869e-05	0.0045106
7	128	129	0.0078125	5.57172e-06	0.00225528
8	256	257	0.00390625	1.39293e-06	0.00112764
9	512	513	0.00195312	3.48233e-07	0.000563819
10	1024	1025	0.000976562	8.70582e-08	0.000281909

1D Poisson: linear FEM

- We compute the convergence order (recall the formula from lecture 03). Setting for instance for the L^2 error we have

$$P(h) = 1.39293e-06$$
$$P(h/2) = 3.48233e-07$$
$$P(h/4) = 8.70582e-08.$$

  Then:

$$\alpha = \frac{1}{log(2)} log\Big(\Big|\frac{1.39293e-06 - 3.48233e-07}{3.48233e-07 - 8.70582e-08}\Big|\Big) = 1.99999696186957,$$

  which is optimal convergence that confirm the a priori error estimates from before.

- The octave code is:

```
alpha = 1 / log(2) * log(abs(1.39293e-06 - 3.48233e-07)
              / abs(3.48233e-07 - 8.70582e-08))
```

- For the H^1 convergence order we obtain:

```
alpha = 1 / log(2) * log(abs(0.00112764 - 0.000563819)
             / abs(0.000563819 - 0.000281909))
      = 1.00000255878430,
```

  which is again optimal convergence and confirms the theory.

Summary of lecture 08

- Finite elements in 1D: idea, construction, examples
- Error estimates allowing for the verification of practical results

Contents

Basic assembling

Algorithm 38 (Basic assembling - robust, but partly inefficient)

Let $K_s, s = 0, \dots n$ be an element and let i and j be the indices of the degrees of freedom (namely the basis functions). The basic algorithm to compute all entries of the system matrix and right hand side vector is:

$$
\begin{aligned}
&\text{for all elements } K_s \text{ with } s = 0, \dots, n\\
&\quad \text{for all DoFs } i \text{ with } i = 0, \dots, n+1\\
&\qquad \text{for all DoFs } j \text{ with } j = 0, \dots, n+1\\
&\qquad\quad a_{ij} = a_{ij} + \int_{K_s} \phi_i'(x)\, \phi_j'(x)\, dx
\end{aligned}
$$

Basic assembling

Algorithm 39

For the right hand side, we have

$$
\begin{aligned}
&\text{for all elements } K_s \text{ with } s = 0, \dots, n \\
&\quad \text{for all DoFs } i \text{ with } i = 0, \dots, n+1 \\
&\qquad b_i = b_i + \int_{K_s} f(x)\phi_i(x)\,dx.
\end{aligned}
$$

Remark 40

This algorithm is a bit inefficient since a lot of zeros are added. Knowing in advance the polynomial degree of the shape functions allows to add an if-condition to assemble only non-zero entries.

Basic assembling

- We illustrate the previous algorithm for a concrete example.
- Let us compute 1D Poisson on four support points $x_i, i = 0, 1, 2, 3, 4$ that are equidistantly distributed yielding a uniform mesh size $h = x_j - x_{j-1}$.
- The discrete space V_h is given by:

$$V_h = \{\phi_0, \phi_1, \phi_2, \phi_3, \phi_4\}, \quad dim(V_h) = 5.$$

- The number of cells is $\#K = 4$.

Basic assembling

It holds furthermore:

$$z \in \mathbb{R}^5, \quad A \in \mathbb{R}^{5\times 5}, \quad b \in \mathbb{R}^5.$$

We start with $s = 0$, namely K_0:

$$a_{00}^{s=0} = a_{00} = \int_{K_0} \phi_0' \phi_0' = \frac{1}{h}, \quad a_{01}^{s=0} = a_{01} = \int_{K_0} \phi_0' \phi_1' = -\frac{1}{h},$$
$$a_{02}^{s=0} = a_{02} = \int_{K_0} \phi_0' \phi_2' = 0,$$
$$a_{03}^{s=0} = a_{03} = \int_{K_0} \phi_0' \phi_3' = 0,$$
$$a_{04}^{s=0} = a_{04} = \int_{K_0} \phi_0' \phi_4' = 0.$$

Basic assembling

- Similarly, we evaluate $a_{1j}, a_{2j}, a_{3j}, a_{4j}, j = 0, \dots 4$.
- Next, we increment $s = 1$ and work on cell K_1. Here we again evaluate all a_{ij} and sum them to the previously obtained values on K_0. Therefore the $+=$ in the above algorithm.
- We also see that we add a lot of zeros when $|i - j| > 1$. For this reason, a good algorithm first design the sparsity pattern and determines the entries of A that are non-zero. This is clear due to the construction of the hat functions and that they only overlap on neighboring elements.

Basic assembling

After having assembled the values on all four elements $K_s, s = 1, 2, 3, 4$, we obtain the following system matrix:

$$A = \begin{pmatrix} \sum_s a^s_{00} & \sum_s a^s_{01} & \sum_s a^s_{02} & \sum_s a^s_{03} & \sum_s a^s_{04} \\ \sum_s a^s_{10} & \sum_s a^s_{11} & \sum_s a^s_{12} & \sum_s a^s_{13} & \sum_s a^s_{14} \\ \sum_s a^s_{20} & \sum_s a^s_{21} & \sum_s a^s_{22} & \sum_s a^s_{23} & \sum_s a^s_{24} \\ \sum_s a^s_{30} & \sum_s a^s_{31} & \sum_s a^s_{32} & \sum_s a^s_{33} & \sum_s a^s_{34} \\ \sum_s a^s_{40} & \sum_s a^s_{41} & \sum_s a^s_{42} & \sum_s a^s_{43} & \sum_s a^s_{44} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

Basic assembling

To fix the homogeneous Dirichlet conditions, we can manipulate directly the matrix A or work with a 'constraint matrix'. We eliminate the entries of the rows and columns of the off-diagonals corresponding to the boundary indices; here $i = 0$ and $i = 4$. Then:

$$A = \frac{1}{h}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Correspondingly, the right hand side in the first and last row has to be changed to

$$b_0 = \frac{1}{h}u(0), \qquad b_4 = \frac{1}{h}u(1)$$

Alternatively, one can eliminate $z_0 = u(0)$ and $z_4 = u(1)$ from the system

Numerical quadrature

As previously stated, the arising integrals may easily become difficult such that a direct integration is not possible anymore:

- Non-constant right hand sides $f(x)$ and non-constant coefficients $\alpha(x)$;
- Higher-order shape functions;
- Non-uniform step sizes, more general domains.

Numerical quadrature

In modern FEM programs, Algorithm 38 is complemented by an alterative evaluation of the integrals using numerical quadrature. The general formula reads:

$$\int_\Omega g(x) \approx \sum_{l=0}^{n_q} \omega_l g(q_l)$$

with quadrature weights ω_l and quadrature points q_l. The number of quadrature points is $n_q + 1$.

Remark 41

The support points x_i and q_l do not need to be necessarily the same. For Gauss quadrature, they are indeed different. For lowest-order interpolatory quadrature rules (box, Trapez) they correspond though.

Numerical quadrature

We continue the above example by choosing the trapezoidal rule, which in addition, should integrate the arising integrals exactly:

$$\int_{K_s} g(x) \approx h_s \sum_{l=0}^{n_q} \omega_l g(q_l)$$

where h_s is the length of interval/element K_s, $n_q = 1$ and $\omega_l = 0.5$. This brings us to:

$$\int_{K_s} g(x) \approx h_s \frac{g(q_0) + g(q_1)}{2}.$$

Applied to our matrix entries, we have on an element K_s:

$$a_{ii} = \int_{K_s} \phi_i'(x)\phi_i'(x)\,dx \approx \frac{h_s}{2}\Big(\phi_i'(q_0)\phi_i'(q_0) + \phi_i'(q_1)\phi_i'(q_1)\Big).$$

Numerical quadrature

For the right hand side, we for the case $f = 1$ we can use for instance the mid-point rule:

$$\frac{1}{h_i} \int_{K_i} \phi_i(x)\, dx \approx \frac{1}{h_i} h_i \phi_i \left(\frac{x_i + x_{i-1}}{2} \right) = \phi_i \left(\frac{x_i + x_{i-1}}{2} \right).$$

Remark 42

If $f = f(x)$ with an dependency on x, we should use a quadrature formula that integrates the function $f(x)\phi_i(x)$ as accurate as possible.

Remark 43

It is important to notice that the order of the quadrature formula must be sufficiently high since otherwise the quadrature error dominates the convergence behavior of the FEM scheme.

Numerical quadrature

We have now all ingredients to extend Algorithm 38:

Algorithm 44 (Assembling using the trapezoidal rule)

Let $K_s, s = 0, \dots n$ be an element and let i and j be the indices of the degrees of freedom (namely the basis functions). The basic algorithm to compute all entries of the system matrix A and right hand side vector b is:

$$
\begin{aligned}
&\text{for all elements } K_s \text{ with } s = 0, \dots, n \\
&\quad \text{for all DoFs } i \text{ with } i = 0, \dots, n+1 \\
&\quad\quad \text{for all DoFs } j \text{ with } j = 0, \dots, n+1 \\
&\quad\quad\quad \text{for all quad points } l \text{ with } l = 0, \dots, n_q \\
&\quad\quad\quad\quad a_{ij} = a_{ij} + h_s \phi_i'(q_l)\phi_j'(q_l)
\end{aligned}
$$

where $n_q = 1$. Here $+=$ means that entries with the same indices are summed. This is necessary because on all cells K_s we assemble again a_{ij}.

Numerical quadrature

Algorithm 45 (Right-hand side)

for all elements K_s with $s = 0, \dots, n$
  for all DoFs i with $i = 0, \dots, n+1$
    for all quad points l with $l = 0, \dots, n_q$
      $b_i = b_i + h_s f(q_l)\phi_i(q_l)$

Master element

In practice, all integrals are transformed onto a **master element** (or so-called **reference element**) and evaluated there. This has the advantage that

- we only need to evaluate once all basis functions;
- numerical integration formulae are only required on the master element;
- independence of the coordinate system. For instance quadrilateral elements in 2D change their form when the coordinate system is rotated.

The price to pay is to compute at each step a deformation gradient and a determinant, which is however easier than evaluating all the integrals.

Master element

- We consider the (physical) element $K_i^{(h_i)} = [x_i, x_{i+1}], i = 0, \ldots n$ and the variable $x \in K_i^{(h_i)}$ with and $h_i = x_{i+1} - x_i$.
- Without loss of generality, we work in the following with the first element $K_0^{(h_0)} = [x_0, x_1]$ and $h = h_0 = x_1 - x_0$. The generalization to s elements is briefly discussed later.
- The element $K_i^{(h_i)}$ is transformed to the master element (i.e., the unit interval with mesh size $h = 1$) $K^{(1)} := [0, 1]$ with the local variable $\xi \in [0, 1]$.

Master element

For the transformations, we work with the substitution rule. Here in 1D, and in higher dimensions with the analogon. We define the mapping

$$\begin{aligned} T_h : K^{(1)} &\to K_0^{(h_0)} \\ \xi &\mapsto T_h(\xi) = x = x_0 + \xi \cdot (x_1 - x_0) = x_0 + \xi h. \end{aligned}$$

While function values can be identified in both coordinate systems, i.e.,

$$f(x) = \hat{f}(\xi), \quad \hat{f} \text{ defined in } K^{(1)},$$

derivatives will be complemented by further terms due to the chain rule that we need to employ. Differentiation in the physical coordinates yields

$$\begin{aligned} \frac{d}{dx}: \quad & 1 = (x_1 - x_0)\,\frac{d\xi}{dx} \\ \Rightarrow \quad & dx = (x_1 - x_0) \cdot d\xi. \end{aligned}$$

Master element

- The volume (here in 1D: length) change can be represented by the determinant of the Jacobian of the transformation:

$$J := x_1 - x_0 = h.$$

- These transformations follow exactly the way as they are known in continuum mechanics.

Master element

We now construct the inverse mapping

$$T_h^{-1} : K_0^{(h_0)} \to K^{(1)}$$
$$x \mapsto T_h^{-1}(x) = \xi = \frac{x - x_0}{x_1 - x_0} = \frac{x - x_0}{h},$$

with the derivative

$$\partial_x T_h^{-1}(x) = \xi_x = \frac{d\xi}{dx} = \frac{1}{x_1 - x_0}.$$

A basis function φ_i^h on $K_0^{(h_0)}$ reads:

$$\varphi_i^h(x) := \varphi_i^1(T_h^{-1}(x)) = \varphi_i^1(\xi)$$

and for the derivative we obtain with the chain rule:

$$\partial_x \varphi_i^h(x) = \partial_\xi \varphi_i^1(\xi) \cdot \partial_x T_h^{-1}(x) = \partial_\xi \varphi_i^1(\xi) \cdot \xi_x$$

with $T_h^{-1}(x) = \xi$.

Master element

Example 36

We provide two examples. Firstly:

$$\int_{K_h} f(x)\,\varphi_i^h(x)\,dx \overset{\text{Sub.}}{=} \int_{K^{(1)}} f(\,T_h(\xi))\cdot\varphi_i^1(\xi)\cdot J\cdot d\xi, \tag{76}$$

and secondly,

$$\int_{K_h} \partial_x\varphi_i^h(x)\cdot\partial_x\varphi_j^h(x)\,dx = \int_{K^{(1)}} \left(\partial_\xi\varphi_i^1(\xi)\right)\cdot\xi_x\cdot\left(\partial_\xi\varphi_j^1(\xi)\right)\cdot\xi_x\cdot J\,d\xi. \tag{77}$$

Master element

We can now apply numerical integration using again the trapezoidal rule and obtain for the two previous integrals:

$$\int_{T_h} f(x)\,\varphi_i^h(x)\,dx \overset{(76)}{\approx} \sum_{k=1}^{q} \omega_k f(F_h(\xi_k))\,\varphi_i^1(\xi_k) \cdot J$$

and for the second example:

$$\int_{T_h} \partial_x \varphi_j^h(x) \partial_x \varphi_i^h(x)\,dx \approx \sum_{k=1}^{q} \omega_k \left(\partial_\xi \varphi_j^1(\xi_k) \cdot \xi_x\right) \cdot \left(\partial_\xi \varphi_i^1(\xi_k) \cdot \xi_x\right) \cdot J.$$

Remark 46

These final evaluations can again be realized by using Algorithm 44, but are now performed on the unit cell $K^{(1)}$.

Generalization to s elements

We briefly setup the notation to evaluate the integrals for s elements:

- Let n be the index of the end point $x_n = b$ (b is the nodal point of the right boundary). Then, $n-1$ is the number of elements (intervals in 1d), and $n+1$ is the number of the nodal points (degrees of freedom - DoFs) and the number shape functions, respectively:
- $K_s^{(h_s)} = [x_s, x_{s+1}], s = 0, \dots n-1.$
- $h_s = x_{s+1} - x_s;$
- $T_s : K^{(1)} \to K_s^{(h_s)} : \xi \mapsto T_s(\xi) = x_s + \xi(x_{s+1} - x_s) = x_s + \xi h_s;$
- $T_s^{-1} : K_s^{(h_s)} \to K^{(1)} : x \mapsto T_s^{-1}(x) = \frac{x - x_s}{h_s};$
- $\nabla T_s^{-1}(x) = \partial_x T_s^{-1}(x) = \frac{1}{h_s}$ (in 1D);
- $\nabla T_s(\xi) = \partial_x T_s(\xi) = (x_s + \xi h_s)' = h_s$ (in 1D);
- $J_s := \det(\nabla T_s(\xi)) = (x_s + \xi h_s)' = h_s$ (in 1D).

Numerical solution of the arising linear equation systems

Numerical solution of $Az = b$

Numerical solution

We provide some ideas how to solve the arising linear systems

$$Az = b$$

where

$$A \in \mathbb{R}^{n\times n}, \quad z = (z_1, \ldots, z_n)^T \in \mathbb{R}^n, \quad b \in \mathbb{R}^n.$$

when discretizing a PDE using finite differences or finite elements. We notice that to be consistent with the previous notation, we assume that the boundary points x_0 and x_{n+1} are not assembled.

Numerical solution

For a moderate number of degrees of freedom, direct solvers such as Gaussian elimination, LU or Cholesky (for symmetric A) can be used. More efficient schemes for large problems in terms of

- computational cost (CPU run time);
- and memory consumptions

are **iterative solvers.**

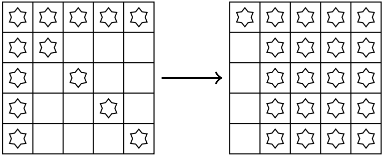
The reason is that Finite Element methods lead to **sparse** linear systems due to the choice of basis functions

Illustrative examples of floating point operations and CPU times are provided in Richter/Wick; 2017 [Pages 68-69, Tables 3.1 and 3.2].

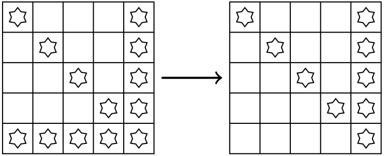
Problem with Direct solvers: Fill-In

Gauss: add multiple of first row to eliminate first column

$\mathcal{O}(n)$ $(n-1)n+1$ non-zeros



better: switch rows and colums; eliminate lower row elements



$\mathcal{O}(n)$ $\mathcal{O}(n)$

Complexity for Various Solvers

Direct solvers	$d = 2$	$d = 3$
Gaussian elimination (GEM)	n^3	n^3
Banded GEM	n^2	$n^{7/3}$
nested dissection ordering GEM	$n^{3/2}$	n^2
Iterative solvers		
Gauss-Seidel, Jacobi	n^2	$n^{5/3}$
conjugate gradient, SOR	$n^{3/2}$	$n^{4/3}$
multigrid	n	n

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

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

$$g(z) = z.$$

We provide in the following a brief introduction. Starting from

$$Az = b$$

we write

$$0 = b - Az$$

and therefore

$$z = \underbrace{z + (b - Az)}_{g(z)}.$$

Introducing a scaling matrix C (in fact C is a preconditioner) and an iteration, we arrive at

$$z^k = z^{k-1} + C(b - Az^{k-1}).$$

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

Summarizing, we have

Definition 37

Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ and $C \in \mathbb{R}^{n\times n}$. To solve

$$Az = b$$

we choose an initial guess $z^0 \in \mathbb{R}^n$ and we iterate for $k = 1, 2, \ldots$:

$$z^k = z^{k-1} + C(b - Az^{k-1}).$$

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:

$$z^k = Bz^{k-1} + c.$$

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

Thanks to the construction of

$$g(z) = Bz + c = z + C(b - Az)$$

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

$$g(z) = z$$

with the solution

$$Az = b$$

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

Definition 38 (Richardson iteration)

The simplest choice of C is the scaled identity matrix, i.e.,

$$C = \omega I.$$

Then, we obtain the Richardson iteration

$$z^k = z^{k-1} + \omega(b - Az^{k-1})$$

with a relaxation parameter $\omega > 0$.

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

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:

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

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

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

Definition 39 (Jacobi method)

To solve $Az = b$ with $A = L + D + R$ let $z^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \ldots$

$$z^k = z^{k-1} + D^{-1}(b - Az^{k-1})$$

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

$$z^k = Jz^{k-1} + D^{-1}b.$$

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

Definition 40 (Gauß-Seidel method)

To solve $Az = b$ with $A = L + D + R$ let $z^0 \in \mathbb{R}^n$ be an initial guess. We iterate for $k = 1, 2, \ldots$

$$z^k = z^{k-1} + (D + L)^{-1}(b - Az^{k-1})$$

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

$$z^k = Hz^{k-1} + (D + L)^{-1}b.$$

Fixed-point schemes: Richardson, Jacobi, Gauss-Seidel

Theorem 41 (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 full A). For each step, in index-notation for each entry it holds:

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

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

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

However, for sparse matrices the work is only $O(n)$!

Gradient descent

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:

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

where d^k denotes the **direction** in which we go at each step. For instance:

$$d^k = D^{-1}(b - Az^k), \quad d^k = (D + L)^{-1}(b - Az^k)$$

for the Jacobi and Gauss-Seidel methods, respectively.

Gradient descent

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

$$z^{k+1} = z^k + \omega^k d^k,$$

  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 without first-order terms studied previously in this section.
- A key point is another view on the problem by regarding it as a minimization problem for which $Az = b$ is the first-order necessary condition and consequently the sought solution.
- Imagine for simplicity that we want to minimize $f(z) = \frac{1}{2}az^2 - bz$. The first-order necessary condition is nothing else than the derivative $f'(z) = az - b$.

Gradient descent

We find a possible minimum via $f'(z) = 0$, namely

$$az - b = 0 \quad \Rightarrow \quad z = a^{-1}b, \quad \text{if } a \neq 0.$$

That is exactly the same how we would solve a linear matrix system $Az = b$. By regarding it as a minimum problem we understand better the purpose of our derivations: How does minimizing a function $f(z)$ work in terms of an iteration? Well, we try to minimize f at each step k:

$$f(z^0) > f(z^1) > \ldots > f(z^k)$$

This means that the direction d^k (to determine $z^{k+1} = z^k + \omega^k d^k$) should be a descent direction.

Gradient descent

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.

Gradient descent

Algorithm 47 (Descent method - basic idea)

Let $A \in \mathbb{R}^{n \times n}$ be positive definite and $z^0, b \in \mathbb{R}^n$. Then for $k = 0, 1, 2, \ldots$

- Compute d^k;
- Determine ω^k as minimum of $\omega^k = argmin\, Q(z^k + \omega^k d^k)$;
- Update $z^{k+1} = z^k + \omega^k d^k$.

For instance d^k can be determined via the Jacobi or Gauss-Seidel methods.

Gradient descent

Algorithm 48 (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 $z^0 \in \mathbb{R}$ and the initial search direction $d^0 = b - Az^0$. Then $k = 0, 1, 2, \ldots$

- Compute the vector $r^k = Ad^k$;
- Compute the relaxation

$$\omega^k = \frac{\|d_k\|_2^2}{(r^k, d^k)_2}$$

- Update the solution vector $z^{k+1} = z^k + \omega^k d^k$.
- 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 $Az = b$.

CG: conjugate gradients

In order to enhance the performance of gradient descent, the conjugate gradient (CG) scheme was developed. Here, the search directions $\{d^0, \dots, d^{k-1}\}$ are pairwise orthogonal. The measure of orthogonality is achievend by using the A scalar product:

$$(Ad^r, d^s) = 0 \quad \forall r \neq s$$

At step k, we seek the approximation $z^k = z^0 + \sum_{i=0}^{k-1} \alpha_i d^i$ as the minimum of all $\alpha = (\alpha_0, \dots, \alpha_{k-1})$ with respect to $Q(z^k)$:

$$\min_{\alpha \in \mathbb{R}^k} Q\left(z^0 + \sum_{i=0}^{k-1} \alpha_i d^i\right) =$$

$$\min_{\alpha \in \mathbb{R}^k} \left\{ \frac{1}{2} \left(Az^0 + \sum_{i=0}^{k-1} \alpha_i A d^i, z^0 + \sum_{i=0}^{k-1} \alpha_i d^i \right) - \left(b, z^0 + \sum_{i=0}^{k-1} \alpha_i d^i \right) \right\}$$

CG: conjugate gradients

The stationary point is given by

$$0 \overset{!}{=} \frac{\partial}{\partial \alpha_j} Q(z^k) = \left(Az^0 + \sum_{i=0}^{k-1} \alpha_i A d^i, d^j \right) - (b, d^j) = -\left(b - Az^k, d^j\right), \quad j = 0, \dots, k-1.$$

Therefore, the new residual $b - Az^k$ is perpendicular to all search directions d^j for $j = 0, \dots, k-1$. The resulting linear equation system

$$(b - Az^k, d^j) = 0 \quad \forall j = 0, \dots, k-1 \tag{78}$$

has the feature of Galerkin orthogonality, which we know as property of FEM schemes.

CG: conjugate gradients

While constructing the CG method, new search directions should be linearly independent of the current d^j. Otherwise, the space would not become larger and consequently, the approximation cannot be improved.

Definition 42 (Krylov space)

We choose an initial approximation $z^0 \in \mathbb{R}^n$ with $d^0 := b - Az^0$ the Krylov space $K_k(d^0, A)$ such that

$$K_k(d^0, A) := \mathrm{span}\{d^0, Ad^0, \dots, A^{k-1}d^0\}.$$

Here, A^k means the k-th power of A.

CG: conjugate gradients

Algorithm 49

Let $A \in \mathbb{R}^{n\times n}$ symmetric positive definite and $z^0 \in \mathbb{R}^n$ and $r^0 = d^0 = b - Az^0$ be given. Iterate for $k = 0, 1, \ldots$:

1. $\alpha_k = \frac{(r^k, d^k)}{(Ad^k, d^k)}$
2. $z^{k+1} = z^k + \alpha_k d^k$
3. $r^{k+1} = r^k - \alpha_k A d^k$
4. $\beta_k = \frac{(r^{k+1}, Ad^k)}{(d^k, Ad^k)}$
5. $d^{k+1} = r^{k+1} - \beta_k d^k$

Without round-off errors, the CG scheme yields after (at most) n steps the solution of a n-dimensional problem and is in this sense a direct method rather than an iterative scheme. However, in practice for huge n, the CG scheme is usually stopped earlier, yiedling an approximate solution.

CG: conjugate gradients

Proposition 50 (Convergence of the CG scheme)

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Let $b \in \mathbb{R}^n$ a right hand side vector and let $z^0 \in \mathbb{R}^n$ be an initial guess. Then:

$$\|z^k - z\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|z^0 - z\|_A, \quad k \geq 0,$$

with the spectral condition $\kappa = \mathrm{cond}_2(A)$ of the matrix A.

CG: conjugate gradients

Remark 51

We see immediately that a large condition number $\kappa \to \infty$ yields

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \to 1$$

and deteriorates significantly the convergence rate of the CG scheme. This is the key reason why preconditioners of the form $P^{-1} \approx A^{-1}$ are introduced that re-scale the system:

$$\underbrace{P^{-1}A}_{\approx I} z = P^{-1}b.$$

Computations to substantiate these findings are provided later.

Preconditioners

Preconditioning reformulates the original system with the goal of obtaining a moderate condition number for the modified system. Let $P \in \mathbb{P}^{n\times n}$ be a matrix with

$$P = KK^T.$$

Then:

$$Az = b \quad \Leftrightarrow \quad \underbrace{K^{-1}A(K^T)^{-1}}_{=:\tilde{A}}\underbrace{K^Tz}_{=:\tilde{z}} = \underbrace{K^{-1}b}_{=:\tilde{b}},$$

which is

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

In the case of

$$\mathrm{cond}_2(\tilde{A}) \ll \mathrm{cond}_2(A)$$

and if the application of K^{-1} is cheap, then the consideration of a preconditioned system $\tilde{A}\tilde{z} = \tilde{b}$ yields a much faster solution of the iterative scheme. The condition $P = KK^T$ is necessary such that the matrix $\tilde{A}$ keeps its symmetry.

Preconditioners

We seek P such that

$$P \approx A^{-1}.$$

On the other hand

$$P \approx I,$$

such that the construction of P is not too costly. Obviously, these are two conflicting requirements.

Preconditioners

The preconditioned CG scheme (PCG) can be formulated as:

Algorithm 52

Let $A \in \mathbb{R}^{n\times n}$ symmetric positive definite and $P = KK^T$ a symmetric preconditioner. Choosing an initial guess $z^0 \in \mathbb{R}^n$ yields:

1. $r^0 = b - Az^0$
2. $Pp^0 = r^0$
3. $d^0 = p^0$
4. For $k = 0, 1, \dots$
    1. $\alpha_k = \frac{(r^k,d^k)}{(Ad^k,d^k)}$
    2. $z^{k+1} = z^k + \alpha_k d^k$
    3. $r^{k+1} = r^k - \alpha_k Ad^k$
    4. $Pp^{k+1} = r^{k+1}$
    5. $\beta_k = \frac{(r^{k+1},p^{k+1})}{(r^k,g^k)}$
    6. $d^{k+1} = p^{k+1} + \beta_k d^k$

Numerical tests

Poisson problem in 2D and 3D on the unit square with force $f = 1$.
We use as solvers:

- CG
- PCG with SSOR preconditioning and $\omega = 1.2$
- GMRES
- GMRES with SSOR preconditioning and $\omega = 1.2$
- BiCGStab
- BiCGStab with SSOR preconditioning and $\omega = 1.2$

Numerical tests

The tolerance is chosen as $TOL = 1.0e-12$. We also run on different mesh levels in order to show the dependency on n.

Dimension	Elements	DoFs	CG	PCG	GMRES	GMRES prec.	BiCGStab	BiCGStab prec.
2	256	289	23	19	23	18	16	12
2	1024	1089	47	33	83	35	33	21
2	4096	4225	94	60	420	78	66	44
3	4096	4913	25	19	25	21	16	11
3	32768	35937	51	32	77	38	40	23
3	262144	274625	98	57	307	83	69	46

Summary of lecture 09

- Finite elements in 1D on a practical level
- Numerical integration
- Master element
- Numerical solution: direct and iterative
- Some numerical examples showing the performance of various solvers

Exercise 4

This exercise is a continuation of Exercise 3. We are again given the following problem: Let $\alpha \in \mathbb{R}$ and the interval $\Omega = (0, 1)$: Find $u : \Omega \to \mathbb{R}$ such that

$$
\begin{aligned}
-\alpha u''(x) &= f \quad \text{in } \Omega \\
u(0) = u(1) &= 0
\end{aligned}
$$

and $\alpha = 1$ and the right hand side $f = -a$ with $a > 0$.

Exercise 4: Tasks

1. Implement P_2 finite elements to solve the above problem. Please first recapitulate quadratic shape functions for yourself by hand.
2. Go into the code and implement the necessary modifications.
3. Implement a numerical quadrature rule in order to evaluate locally the integrals.
4. Check your code using your 'physical intuition'. This means, does the code deliver results that are 'similar' to those from yesterday? Hint: On purpose we do not perform a rigorous computational convergence analysis in this exercise because in 1D the finite element method is actually 'too simple' and would yield for point-wise errors exactly zero.

Exercise 4: Hints to quadratic finite elements

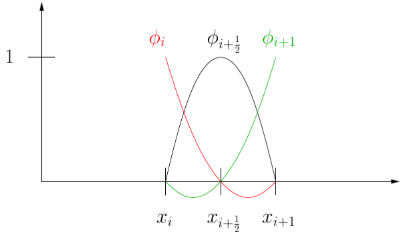
- First we define the discrete space:

$$V_h = \{v \in C[0,1] |\ v|_{K_j} \in P_2\}$$

The space V_h is composed by the basis functions:

$$V_h = \{\phi_0, \ldots, \phi_{n+1}, \phi_{\frac{1}{2}}, \ldots, \phi_{n+\frac{1}{2}}\}.$$

- The dimension of this space is $\dim(V_h) = 2n + 1$.
- The mid-points represent degrees of freedom as the two edge points. For instance on each $K_j = [x_j, x_{j+1}]$ we have as well $x_{j+\frac{1}{2}} = x_j + \frac{h}{2}$, where $h = x_{j+1} - x_j$.



Exercise 4: Hints to quadratic finite elements

Definition 43 (P_2 shape functions)

On the element $K^{(1)}$ (unit element), we have

$$\begin{aligned}\phi_0(\xi) &= 1 - 3\xi + 2\xi^2,\\ \phi_{\frac{1}{2}}(\xi) &= 4\xi - 4\xi^2,\\ \phi_1(\xi) &= -\xi + 2\xi^2.\end{aligned}$$

These basis functions fulfill the property:

$$\phi_i(\xi_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

for $i, j = 0, \frac{1}{2}, 1$. On the master element, a function has therefore the presentation:

$$u(\xi) = \sum_{j=0}^{1} u_j \phi_j(\xi) + u_{\frac{1}{2}} \phi_{\frac{1}{2}}(\xi).$$

Exercise 4: Hints to quadratic finite elements

Using these three shape functions we can now evaluate

$$A_{i,j} = \int_0^1 \phi_i' \phi_j' \, dx$$

and

$$b_j = \int_0^1 (-a)\phi_j \, dx$$

with the Simpson rule to obtain the **local** stiffness matrix

$$A = \frac{1}{h} \begin{pmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7 \end{pmatrix}$$

and the **local** right hand side

$$b = \frac{h}{6}(-a, -4a - a)^T$$

Conclusions

- Numerical methods for ODEs (Classes 1-4): finite differences of low- and higher-order
- Galerkin weak formulation for ODEs (Class 5)
- Numerical methods for PDEs (Classes 6-9) based on Galerkin finite elements
- We touched the three ingredients of scientific computing:
  1. Mathematical modeling
  2. Design and analysis of numerical schemes
  3. Implementation and software design of the developed algorithms
- Specifically, we performed computational analyses by substantiating the theory with the help of numerical tests.
- Use existing open-source software! There are many packages available, e.g.
  - `www.dealii.org`
  - `www.dune-project.org`

Online materials

The materials presented in this spring school are collected here:

`https://cloud.ifam.uni-hannover.de/index.php/s/Cwe4ZqwLRMixS3J`

with the password that is known to you.

Upcoming: Cusco

The End

Thanks for the active participating in this PeC^3 School on Numerical Modelling with Differential Equations !

Final questions ?

Peter Bastian
https://conan.iwr.uni-heidelberg.de/
Peter.Bastian@iwr.uni-heidelberg.de
Thomas Wick
https://www.ifam.uni-hannover.de/wick
thomas.wick@ifam.uni-hannover.de