Numerical Methods for PDEs — TA Summary

Created by Jonas Bachmann, Paul Fischill, Samuel Russo and Nico Graf

Ported to Typst by Valentin Vogt

Last updated on May 09, 2024

About

Theorems, definitions and equations
from the lecture notes come in boxes like this one.

Less important results
that are given for context or completeness look like this.

Tips and practical advice

from the TAs are highlighted like this.

Basics

Theorem 0.3.1.19: Cauchy–Schwarz Inequality
If a is a symmetric positive semi-definite bilinear form, then

$$|a(u,v)| \leq a(u,u)^{\frac{1}{2}} a(v,v)^{\frac{1}{2}} \tag{1}$$

Equation 1.3.4.15 (Cauchy–Schwarz for Integrals)

$$\left| \int_\Omega u(\boldsymbol{x}) v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \right| \leq \left(\int_\Omega |u(\boldsymbol{x})|^2 \, \mathrm{d}\boldsymbol{x} \right)^{\frac{1}{2}} \left(\int_\Omega |v(\boldsymbol{x})|^2 \, \mathrm{d}\boldsymbol{x} \right)^{\frac{1}{2}} = \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \tag{2}$$

Norms

- **Supremum norm:** $\|\boldsymbol{u}\|_\infty = \|\boldsymbol{u}\|_{L^\infty(\Omega)} := \sup_{\boldsymbol{x} \in \Omega} \|\boldsymbol{u}(\boldsymbol{x})\|$
- **L^2 norm:** $\|\boldsymbol{u}\|_2 = \|\boldsymbol{u}\|_{L^2(\Omega)} := \left(\int_\Omega \|\boldsymbol{u}(\boldsymbol{x})\|^2 \, \mathrm{d}x \right)^{\frac{1}{2}}$

Theorem 0.3.2.31: Transformation rule for Integration
Given two domains $\Omega, \widehat{\Omega}$ and a continuous, differentiable mapping $\Phi : \widehat{\Omega} \to \Omega$

$$\int_\Omega f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_{\widehat{\Omega}} f(\Phi(\hat{x})) |\det D\Phi(\hat{x})| \, \mathrm{d}\hat{\boldsymbol{x}} \tag{3}$$

1 Second-Order Scalar Elliptic Boundary Value Problems

1.2 Quadratic Minimization Problems

In the following, let V be a vector space over $\mathbb{R}$.

Linear forms
$\ell : V \to \mathbb{R}$ is a *linear form / linear functional* $\iff$

$$\ell(\alpha u + \beta v) = \alpha \ell(u) + \beta \ell(v) \qquad \forall u, v \in V, \forall \alpha, \beta \in \mathbb{R} \tag{4}$$

Bilinear forms
$a : V \times V \to \mathbb{R}$ is a *bilinear form* $\iff$

$$\begin{aligned} &a(su_1 + u_2, tv_1 + v_2) \\ &\quad = st \cdot a(u_1, v_1) + sa(u_1, v_2) + ta(u_2, v_1) + a(u_2, v_2) \forall u_1, u_2, v_1, v_2 \in V, \forall s, t \in \mathbb{R} \end{aligned} \tag{5}$$

Positive definiteness
A bilinear form $a : V \times V \to \mathbb{R}$ is *positive definite* if

$$u \in V \setminus \{\mathbf{0}\} \iff a(u, u) > 0$$

It is *positive semi-definite* if

$$a(u, u) \geq 0 \quad \forall u \in V$$

Quadratic Functional
A *quadratic functional* $J : V \to \mathbb{R}$ is defined by

$$J(u) := \frac{1}{2} a(u, u) - \ell(u) + c, \quad u \in V \tag{6}$$

where $a : V \times V \to \mathbb{R}$ a symmetric bilinear form, $\ell : V \to \mathbb{R}$ a linear form and $c \in \mathbb{R}$.

Continuity of linear form
A linear form $\ell : V \to \mathbb{R}$ is *continuous / bounded* on V, if

$$\exists C > 0 \quad |\ell(v)| \leq C \|v\| \quad \forall v \in V, \tag{7}$$

where $\|\cdot\|$ is a norm on V.

1.3 Sobolev Spaces

When we solve a minimization problem, we first need to define the space of functions in which we look for the solution. For example, in Physics, we generally want the solution to be continuous. E.g, a function describing the shape of an elastic string should not have jumps.

It turns out that the correct space to describe our minimization problems is the **Sobolev space**. For functions u in a Sobolev space, the bilinear form a in the quadratic functional is well defined (i.e., $a(u, u) < \infty$). Hence the space in which we look for minimizers is determined by the given quadratic functional. To select the space for your problem, follow the guideline … *Choose the largest space such that the problem is well defined.*

Sobolev Spaces
$H_0^1(\Omega)$ is a vector space with norm

$$|v|_{H^1} := \left(\int_\Omega \|\mathbf{grad}\, v\|^2 \, \mathrm{d}\boldsymbol{x} \right)^{\frac{1}{2}}$$

$H^1(\Omega)$ is another vector space with norm

$$\|v\|_{H^1}^2 := \|v\|_{L^2}^2 + |v|_{H^1}^2$$

Note that $|\cdot|_{H^1}$ is not a norm on the space $H^1(\Omega)$, but a seminorm.

Both spaces contain all functions for which the norm is finite (and, in the case of $H_0^1(\Omega)$, which are 0 on $\partial\Omega$).

Alternative notation for norms includes $\|\cdot\|_0$ for $\|\cdot\|_{L^2}$ and $|\cdot|_1$ for $|\cdot|_{H^1}$.

If the quadratic minimization problem is well defined, we get the following lemma for existence and uniqueness of minimizers:

Theorem 1.3.3.6: Existence of minimizers in Hilbert spaces
On a real Hilbert space V with norm $\|\cdot\|_a$ for any $\|\cdot\|_a$-bounded linear functional $\ell : V \to \mathbb{R}$, the quadratic minimization problem

$$\begin{aligned} u_* &= \operatorname*{argmin}_{v \in V} J(v) \\ J(v) &:= \frac{1}{2}\|v\|_a^2 - \ell(v) \end{aligned} \tag{8}$$

has a unique solution.

Note that here, we use the bilinear form to define the norm $\|u\|_a = \sqrt{a(u,u)}$. The main point is that we can see the bilinear form of the quadratic minimization problem as the norm of some Sobolev space. The above theorem guarantees that a solution exists in this space if the linear form is bounded.

For checking boundedness we can often use Cauchy–Schwarz (Equation 1.3.4.15) and Poincaré–Friedrichs (Theorem 1.8.0.20).

1.4 Linear Variational Problem

Definition 1.4.1.6: Linear Variational Problem
Let V be a vector (function) space, $\widehat{V} \subset V$ an affine space, and $V_0 \subset V$ the associated subspace. The equation

$$\text{Find } u \in \widehat{V} \text{ such that } a(u,v) = \ell(v) \quad \forall v \in V_0 \tag{9}$$

is called a (generalized) *linear variational problem*, if

- $a : V \times V_0 \to \mathbb{R}$ is a bilinear form
- $\ell : V_0 \to \mathbb{R}$ is a linear form

Theorem 1.3.3.6 tells us that the minimization problem has a solution, but knowing that a solution exists is of course not enough: We want to find it, but an infinite-dimensional minimization problem is hard to solve. To make it easier, we reformulate the problems in a linear variational form (9), which is quite close to something we can solve numerically. To do this transformation, we use the following equivalence:

Theorem 1.4.1.8: Equivalence of quadratic minimization problem and linear variational problem
For a (generalized) quadratic functional $J(v) = \frac{1}{2}a(v,v) - \ell(v) + c$ on a vector space V and with a symmetric positive definite bilinear form $a : V \times V \to \mathbb{R}$ the following is equivalent:

- The quadratic minimization problem for $J(v)$ has the unique minimizer $u_* \in \widehat{V}$ over the affine subspace $\widehat{V} = g + V_0, g \in V$.
- The linear variational problem

  $$u \in \widehat{V} \quad a(u,v) = \ell(v) \quad \forall v \in V_0$$

  has the unique solution $u_* \in \widehat{V}$.

Note that the trial space $\widehat{V}$, from which we pick a solution, and the test space V_0 can be different. For an example of different trial and test spaces, see Section 1.7.

1.5 Boundary Value Problems

Lemma 1.5.2.1: General product rule
For all $\boldsymbol{j} \in \left(C^1\left(\overline{\Omega}\right)\right)^d, v \in C^1\left(\overline{\Omega}\right)$ holds

$$\operatorname{div}(\boldsymbol{j}v) = v \operatorname{div} \boldsymbol{j} + \boldsymbol{j} \cdot \mathbf{grad}\, v \quad \text{in } \Omega \tag{10}$$

Lemma 1.5.2.4: Gauss' Theorem
Let $\boldsymbol{n} : \partial\Omega \to \mathbb{R}^d$ denote the exterior unit normal vector field on $\partial\Omega$ and dS denote integration over a surface. We have

$$\int_\Omega \operatorname{div} \boldsymbol{j}(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} \boldsymbol{j}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \,\mathrm{d}S(\boldsymbol{x}) \quad \forall \boldsymbol{j} \in \left(C^1_{\mathrm{pw}}\left(\overline{\Omega}\right)\right)^d \tag{11}$$

Lemma 1.5.2.7: Green's first formula
For all vector fields $\boldsymbol{j} \in \left(C^1_{\mathrm{pw}}\left(\overline{\Omega}\right)\right)^d$ and functions $v \in C^1_{\mathrm{pw}}\left(\overline{\Omega}\right)$ holds

$$\int_\Omega \boldsymbol{j} \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} = -\int_\Omega \operatorname{div} \boldsymbol{j}\, v \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \boldsymbol{j} \cdot \boldsymbol{n}\, v \,\mathrm{d}S \tag{12}$$

Lemma 1.5.3.4: Fundamental lemma of the calculus of variations
Let $f \in L^2(\Omega)$ satisfy

$$\int_\Omega f(\boldsymbol{x}) v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = 0 \quad \forall v \in C_0^\infty(\Omega), \tag{13}$$

then $f \equiv 0$.

We have seen that minimizing a quadratic functional is equivalent to solving a linear variational problem (9). The variational problem is called the **weak form**. We can transform it (with extra smoothness requirements) into the problem's **strong form**, an elliptic BVP (PDE with boundary conditions).

Weak to strong

1. Use Lemma 1.5.2.7 to get rid of derivatives on v (e.g. turn $\mathbf{grad}\, u \cdot \mathbf{grad}\, v$ into $-\operatorname{div}(\mathbf{grad}\, u) + \ldots$
2. Use properties of the test space (usually that $v = 0$ on $\partial\Omega$) to get rid of boundary terms
3. Use Lemma 1.5.3.4 to remove the integrals and test functions

1.7 Boundary Conditions

For 2nd-order elliptic BVPs we need boundary conditions to get a unique solution. To be more precise, we need **exactly one** of the following boundary conditions on every part of $\partial\Omega$

Main boundary conditions for 2nd-order elliptic BVPs

- **Dirichlet**: u is fixed to be $g : \partial\Omega \to \mathbb{R}$

$$u = g \quad \text{on } \partial\Omega$$

- **Neumann**: the flux $\boldsymbol{j} = -\kappa(\boldsymbol{x})\ \mathbf{grad}\, u$ through $\partial\Omega$ is fixed with $h : \partial\Omega \to \mathbb{R}$

$$\boldsymbol{j} \cdot \boldsymbol{n} = -h \quad \text{on } \partial\Omega$$

- **Radiation**: flux depends on u with an increasing function $\Psi : \mathbb{R} \to \mathbb{R}$

$$\boldsymbol{j} \cdot \boldsymbol{n} = \Psi(u) \quad \text{on } \partial\Omega$$

In the weak form, Dirichlet conditions have to be imposed directly on the **trial** space. The test space needs to be set to 0 wherever Dirichlet conditions are given ("*Don't test where the solution is known*"). For example, trial and test spaces for a standard Dirichlet problem are

$$V = \{u \in C^1(\Omega) \,|\, u = g \text{ on } \partial\Omega\}$$
$$V_0 = \{v \in C^1(\Omega) \,|\, v = 0 \text{ on } \partial\Omega\}$$

Dirichlet BCs are called **essential boundary conditions**.

Neumann conditions, which are only enforced through some term in the variational equation, are called **natural boundary conditions**.

There are some constraints on the boundary data:

- **Admissible Dirichlet Data**: Dirichlet boundary values need to be continuous.
- **Admissible Neumann Data**: h needs to be in $L^2(\Omega)$ (can be discontinuous)

The following theorem is frequently needed when dealing with integrals over the boundary:

Theorem 1.9.0.19: Multiplicative trace inequality

$$\exists C = C(\Omega) > 0 : \|u\|_{L^2(\partial\Omega)} \leq C\|u\|_{L^2(\Omega)} \cdot \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega) \tag{14}$$

1.8 Second-Order Elliptic Variational Problems

We have seen how we can get from a minimization problem via a variational problem to a BVP. Now we want to move in the opposite direction: from a PDE with boundary conditions to a variational problem.

Strong to weak

1. Test the PDE with (multiply by v) and integrate over Ω
2. Use Lemma 1.5.2.7 to "shift" one derivative from u to v (e.g., from $-\operatorname{div}(\mathbf{grad}\, u)$ to $\mathbf{grad}\, u \cdot \mathbf{grad}\, v + \ldots$)
3. Use Neumann BC on boundary terms ($\mathbf{grad}\, u \cdot n = h$)
4. Pick Sobolev trial/test spaces V, V_0 such that
   - $a(u,u)$ is finite for $u \in V, V_0$
   - boundary conditions are satisfied ($u = g$ in $V \Rightarrow v = 0$ in V_0)

   To fulfill the first condition, we can define the "base" space for both trial and test as $\{v \mid a(v,v) < \infty\}$, which is equal to H^1 for the usual $\Delta u = f$ problem. If there are extra (e.g., boundary) terms in a, try to bound these with the H^1 norm.

For Neumann problems there is a **compatibility condition**. If we choose test function $v \equiv 1$ we get the requirement

$$-\int_{\partial\Omega} h \,\mathrm{d}S = \int_{\Omega} f \,\mathrm{d}\boldsymbol{x}$$

for the existence of solutions. Additionally, the solution of Neumann problems is unique only up to constants. To address this we can use the constrained function space

$$H^1_*(\Omega) := \left\{ v \in H^1(\Omega) : \int_{\Omega} v \,\mathrm{d}\boldsymbol{x} = 0 \right\}$$

Theorem 1.8.0.20: Second Poincaré–Friedrichs inequality
If $\Omega \subset \mathbb{R}^d$ is bounded and connected, then

$$\exists C = C(\Omega) > 0 : \|u\|_0 \le C \operatorname{diam}(\Omega) \|\mathbf{grad}\, u\|_0 \quad \forall u \in H^1_*(\Omega) \tag{15}$$

This theorem tells us that (under some conditions), the L^2 norm of functions from this space is bounded by the H^1-seminorm.

2 Finite Element Method

2.2 Galerkin Discretization

The idea of Galerkin discretization is to replace the infinite-dimensional function space V_0 by a finite-dimensional subspace $V_{0,h} \subset V_0$.

Theorem 2.2.1.5: Unique solution of discrete variational problems
If the bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ is symmetric and positive definite and the linear form $\ell : V_0 \to \mathbb{R}$ is continuous (7) w.r.t. $\|\|_a$, then the discrete variational problem

$$u_h \in V_{0,h} : a(u_h, v_h) = \ell(v_h), \quad \forall v_h \in V_{0,h} \tag{16}$$

has a unique *Galerkin* solution $u_h \in V_{0,h}$ satisfying the energy estimate

$$\|u\|_a \leq \sup_{v_h \in V_{0,h}} \frac{|\ell(v_h)|}{\|v_h\|_a}$$

Recall the definition of a basis (here, superscripts are indices and not to be confused with exponents): $\{b^1, ..., b^N\} \subset V$ is a basis, if for every $v \in V$ there are unique coefficients μ_i such that $v = \sum_{i=1}^N \mu_i b^i$ and $N = \dim V$. Now we can expand $u_h = \mu_1 b^1 + \cdots + \mu_N b^N$ and our goal is to find the coefficients μ_i.

Galerkin Discretization
Linear discrete variational problem (16) $\xrightarrow{\text{Choice of basis } \mathfrak{B}_h}$ Linear system of equations $\boldsymbol{A}\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}$

$$\textbf{Galerkin Matrix}: \quad \boldsymbol{A} = \left[a\left(b^k, b^j\right)\right]_{j,k=1}^N \in \mathbb{R}^{N,N}$$

$$\textbf{RHS vector}: \quad \vec{\boldsymbol{\varphi}} = \left[\ell\left(b^j\right)\right]_{j=1}^N \in \mathbb{R}^N$$

$$\textbf{coefficient vector}: \quad \vec{\boldsymbol{\mu}} = [\mu_1, ..., \mu_N]^\top \in \mathbb{R}^N$$

Note that $\boldsymbol{A}_{j,k} = a\left(b_h^k, b_h^j\right) \neq a\left(b_h^j, b_h^k\right)$ if a is not symmetric (note the order of arguments k, j). Of course the bilinear form a determines some properties of the Galerkin matrix. If a is symmetric and/or positive definite, the Galerkin matrix $\boldsymbol{A}$ will have the same properties.

The choice of $V_{0,h}$ alone determines the quality of the solution u_h. While mathematically the choice of basis $\mathfrak{B}_h$ does not matter, for solving the equation numerically, the choice is crucial: The basis determines properties of $\boldsymbol{A}$ like sparsity, so it influences how stable and efficient the numerical solution is.

Computing the energy norm in code

Sometimes, problems ask you to compute $\|u_h\|_a$, i.e., the energy norm of a discrete solution.

$$\|u_h\|_a = \sqrt{a(u_h, u_h)} = a\left(\sum_{i=0}^N \mu_i b_i^h, \sum_{j=0}^N \mu_j b_j^h\right) = \sum_{i=0}^N \sum_{j=0}^N \mu_i \mu_j a\left(b_i^h, b_j^h\right) = \vec{\mu}^\top \boldsymbol{A} \vec{\mu}$$

So it can be computed as `sqrt(mu.transpose() * A * mu)`.

2.3 Linear FEM in 1D

In FEM, the goal is to approximate u by piecewise polynomial functions.

Mesh in one dimension
Let $\Omega = [a, b]$, we equip it with $M + 1$ **nodes** resulting in the set of nodes:

$$\mathcal{V}(\mathcal{M}) = \{a = x_0 < x1 < \cdots < x_M = b\}$$

The nodes define intervals, which build up the mesh:

$$\mathcal{M} = \{]x_{j-1}, x_j[: 1 \leq j \leq M\}$$

The intervals $[x_{j-1}, x_j]$ are the **cells** of the mesh. We define local cell size $h_j = |x_j - x_{j-1}|$ and global mesh width $h_{\mathcal{M}} = \max_j h_j$.

A simple space for continuous, $\mathcal{M}$-piecewise polynomial functions in $H_0^1(]a, b[)$ consists of linear functions on each cell:

$$V_{0,h} = S^0_{1,0}(\mathcal{M}) = \left\{v \in C^0([a, b]) : v|_{[x_{i-1}, x_i]} \text{ is linear}, i = 1, ..., M, v(a) = v(b) = 0\right\} \quad (17)$$

$$\to N = \dim S^0_{1,0}(\mathcal{M}) = M - 1$$

The 0-superscript stands for C^0: functions are globally continuous. The 1-subscript denotes local degree 1 polynomial and the 0-subscript denotes value 0 on the boundary. The $\mathcal{S}$ stands for $\mathcal{S}$calar functions.

Common basis functions are the 1D tent functions:

$$b_h^j(x) = \begin{cases} (x - x_{j-1})/h_j & \text{if } x_{j-1} \leq x \leq x_j \\ (x_j - x)/h_{j+1} & \text{if } x_j \leq x \leq x_{j+1} \\ 0 & \text{else} \end{cases} \quad (18)$$

$$\to b_h^j(x_i) = \delta_{ij} \quad (19)$$

A basis satisfying condition (19) is called a **cardinal** basis. Another key property of tent functions is that their support just comprises two adjacent cells:

$$\operatorname{supp}\left(b_h^j\right) = [x_{j-1}, x_{j+1}]$$

The support of a function $f : \Omega \to \mathbb{R}$ is defined as $\overline{\{x \in \Omega : f(x) \neq 0\}}$, the set of inputs for which the function is nonzero. Since polynomials are easy to differentiate and integrate, computing (bi)linear forms a and ℓ for them is quite easy.

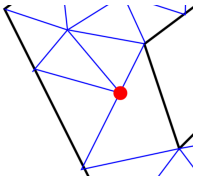
2.4 Linear FEM in 2D

Mesh in two dimension
Meshes in 2D rely on **triangulations**. A **triangulation** $\mathcal{M}$ of Ω satisfies:

- $\mathcal{M} = \{K_i\}$, where K_i are open triangles
- $i \neq j \rightarrow K_i \cap K_j = \varnothing$
- $\bigcup_{i=1}^{M} \overline{K_i} = \overline{\Omega}$
- $i \neq j \rightarrow \overline{K_i} \cap \overline{K_j}$ is either $\varnothing$, an edge from both triangles or a vertex from both

Again, the vertices are called **nodes** and the triangles are the **cells**.

This definition does not allow for hanging nodes because of point 4. Hanging nodes are those which lie on the edge of a triangle:



We define a space of piecewise-linear functions analogously to (17):

$$V_{0,h} = S_1^0(\mathcal{M}) = \left\{ v \in C^0\left(\overline{\Omega}\right) : v_K(\boldsymbol{x}) = \alpha_K + \boldsymbol{\beta}_K \cdot \boldsymbol{x}, \alpha_K \in \mathbb{R}, \boldsymbol{\beta}_K \in \mathbb{R}^2, \boldsymbol{x} \in K \right\} \tag{20}$$

$$\rightarrow \dim S_1^0(\mathcal{M}) = \#\mathcal{V}(\mathcal{M})$$

And $S_{1,0}^0(\mathcal{M})$ additionally requires functions to be zero on $\partial\Omega$, with

$$\dim S_{1,0}^0(\mathcal{M}) = \#\{\boldsymbol{x} \in \mathcal{V}(\mathcal{M}) : \boldsymbol{x} \notin \partial\Omega\}$$

Similarly, the 1D tent functions can be extended to 2D by requiring the cardinal property. This property is already enough since three points fully define a plane — in other words, knowing the values in three points (the vertices of a triangle) fully defines a linear function. Cardinal bases will produce sparse Galerkin matrices, as the support of a basis function only covers the triangles adjacent to its node, which can only interact with neighboring basis functions.

Computation of Galerkin Matrix
Often, bilinear forms are defined by integration over the whole domain. But we have seen that the support of basis functions is only local. We can exploit this by performing integration only over the cells.

$$\boldsymbol{A}_{ij} = a\left(b_h^j, b_h^i\right) = \sum_{K \in \operatorname{supp}\left(b_h^j\right) \cap \operatorname{supp}\left(b_h^i\right)} a|_K\left(b_h^j, b_h^i\right) \tag{21}$$

where $a|_K$ is the local bilinear form over cell K.

Cell oriented assembly
To further take advantage of (21), cell oriented assembly can be performed. Go through all cells and compute $a|_K\left(b_h^j, b_h^i\right)$ for all pairs of basis functions associated with cell K (element matrix) and add the values to the entry at (i, j) of $\boldsymbol{A}$.

The same procedure can be applied to calculating the right hand side vector φ, just that only one basis function is involved as the RHS comes from a linear functional.

2.5 Building Blocks of General Finite Element Methods

The first building block are meshes, see Section 2.3 and Section 2.4. Next we need to choose a space of functions.

Definition 2.5.2.2: Multivariate Polynomials

Space of d-variate (taking inputs in $\mathbb{R}^d$) polynomials with total degree p:

$$\mathcal{P}_p(\mathbb{R}^d) = \left\{ \boldsymbol{x} \in \mathbb{R}^d \to \sum_{\boldsymbol{\alpha} \in \mathbb{N}_0^d, |\boldsymbol{\alpha}| \leq p} c_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}, c_{\boldsymbol{\alpha}} \in \mathbb{R} \right\}$$

with $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_d), \boldsymbol{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdot \cdots \cdot x_d^{\alpha_d}$ and $|\boldsymbol{\alpha}| = \alpha_1 + \cdots + \alpha_d$

As an example, $\mathcal{P}_2(\mathbb{R}^2) = \text{Span}\{1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\}$

Lemma 2.5.2.5: Dimension of spaces of Polynomials

$$\dim \mathcal{P}_p(\mathbb{R}^d) = \binom{d+p}{p}, \quad p \in \mathbb{N}_0, d \in \mathbb{N}$$

which in the limit of $p \to \infty$ behaves like $\mathcal{O}(p^d)$.

Definition 2.5.2.7: Tensor Product Polynomials

The space of tensor product polynomials of degree p in each coordinate is

$$\begin{aligned} \mathcal{Q}_p(\mathbb{R}^d) &= \left\{ \boldsymbol{x} \in \mathbb{R}^d \to \sum_{l_1=0}^{p} \cdots \sum_{l_d=0}^{p} c_{l_1,\ldots,l_d} x_1^{l_1} \cdot \cdots \cdot x_d^{l_d}, \quad \text{where } c_{l_1,\ldots,l_d} \in \mathbb{R} \right\} \\ &= \text{Span}\left\{ \boldsymbol{x} \to p_1(x_1) \cdot \cdots \cdot p_d(x_d), p_i \in \mathcal{P}_p(\mathbb{R}) \right\} \end{aligned}$$

As an example, $\mathcal{Q}_2(\mathbb{R}^2) = \text{Span } \{1, x_1, x_2, x_1 x_2, x_1^2, x_1^2 x_2, x_1^2 x_2^2, x_1 x_2^2, x_2^2\}$

Lemma 2.5.2.8: Dimension of spaces of tensor product polynomials

$$\dim \mathcal{Q}_p(\mathbb{R}^d) = (p+1)^d, \quad p \in \mathbb{N}_0, d \in \mathbb{N}$$

Finally we need **locally** supported basis functions. This basis $\mathfrak{B}_h = b_h^1, \cdots, b_h^N$ should satisfy the following constraints:

1. $\mathfrak{B}_h$ is a basis of V_h, hence $\dim \mathfrak{B}_h = \dim V_h$.
2. Each b_h^i is associated with a single mesh entity (cell/edge/face/vertex).
3. Each b_h^i is only locally supported, i.e., only nonzero on cells adjacent to the associated entity.

2.6 Lagrangian Finite Element Methods

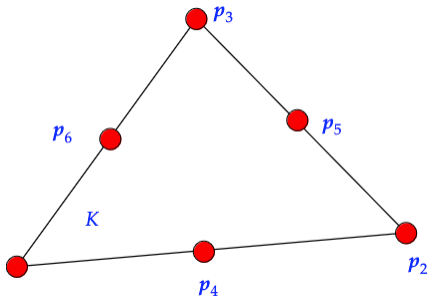
Remember Eqs. (17) and (20) as two examples of finite element spaces. They are examples of the general Lagrangian FE spaces. First we introduce them for *simplicial* meshes, i.e., those consisting of triangles (2D) or tetrahedra (3D).

Definition 2.6.1.1: Simplicial Lagrangian finite element spaces

$$\mathcal{S}_p^0(\mathcal{M}) = \left\{ v \in C^0\left(\overline{\Omega}\right) : v_K \in \mathcal{P}_p(K), \forall K \in \mathcal{M} \right\} \tag{22}$$

This space is well suited for triangular meshes, as the local dimension $\binom{d+p}{p} = \binom{2+p}{p}$ (in 2D) is the same as the number of interpolation nodes of a triangle. The local basis functions of $\mathcal{S}_1^0$ are the barycentric coordinate functions. In $\mathcal{S}_2^0$, the local basis functions are combinations of barycentric coordinate functions:

$$\begin{aligned} b_K^1 &= (2\lambda_1 - 1)\lambda_1, \quad b_K^4 = 4\lambda_1\lambda_2, \\ b_K^2 &= (2\lambda_2 - 1)\lambda_2, \quad b_K^5 = 4\lambda_2\lambda_3, \\ b_K^3 &= (2\lambda_3 - 1)\lambda_3, \quad b_K^6 = 4\lambda_1\lambda_3 \end{aligned}$$



where the local basis functions 1-3 are associated with vertices and 4-6 with edges.

Analogously, the following space is well suited for quadrilaterals, as its local dimension $(p+1)^2$ is the same as the amount of vertices and interpolation points on quads.

Definition 2.6.2.5: Tensor product Lagrangian finite element spaces

$$\mathcal{S}_p^0(\mathcal{M}) = \left\{ v \in C^0\left(\overline{\Omega}\right) : v|_K \in \mathcal{Q}_p(K), \forall K \in \mathcal{M} \right\} \tag{23}$$

Note that the choice of local polynomial space is the only difference, $\mathcal{Q}_p(K)$ instead of $\mathcal{P}_p(K)$. Of course these spaces can be mixed: on mixed (so-called hybrid) meshes, we use (22) on triangles and (23) on quadrilaterals.

2.7 Implementation of Finite Element Methods

Remember the principle of cell-oriented assembly. The goal is to rely mostly on local computations. To perform cell-oriented assembly, a map from local to global indices is needed. In LehrFEM++ this is the job of the dofhandler (`lf::assemble::DofHandler`). It provides the following main methods:

- `NumDofs()`, returns the total number of global basis functions, the dimension of the FE space.
- `NumLocalDofs(const lf::mesh::Entity &)`, returns the number of global basis functions **covering** any geometric entity, so counts those associated with the entity and with its sub-entities.
- `GlobalDofIndices(const lf::mesh::Entity &)`, returns an array of indices of the global basis function **covering** the given entity. The order is with decreasing co-dimension of the functions's associated entity, so vertex, edge, cell. Within each co-dimension, the order is according to the local numbering of the entity.
- `NumInteriorDofs(const lf::mesh::Entity &)`, returns the number of global basis functions **associated with** the given entity (so for a triangle, the number of basis functions in the interior of the triangle, not those of the edges or vertices).
- `InteriorGlobalDofIndices(const lf::mesh::Entity &)`, similar to `GlobalDofIndices` but returns only the indices of the global basis functions **associated with** the given entity.
- `Entity(unsigned int dofum)`, returning the entity associated with the global index `dofnum`.

Instead of dimension, in LehrFEM++ the concept of **co-dimension** is used. Instead of going from a point with dimension 0 to a triangle with dimension 2, the co-dimension is the other way around. The highest-dimensional entity has co-dimension 0. This ensures that cells are always of co-dimension 0.

To assemble the Galerkin matrix, `lf::assemble::AssembleMatrixLocally` can be used. To use it, we need element matrix providers, constructs which provide the *local* element matrix for given bilinear forms. Some common bilinear forms are already implemented.

- $\int_K \alpha(\boldsymbol{x})\, \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x}$ is implemented in `lf::fe::DiffusionElementMatrixProvider`
- $\int_K \gamma(\boldsymbol{x})\, u\, v \,\mathrm{d}\boldsymbol{x}$ is implemented in `lf::fe::MassElementMatrixProvider`
- $\int_e \gamma(\boldsymbol{x})\, u\, v \,\mathrm{d}S$ is implemented in `lf::fe::MassEdgeMatrixProvider`. Note the integration over an edge and not a cell.
- $\int_K \alpha(\boldsymbol{x})\, \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} + \int_K \gamma(\boldsymbol{x})\, u\, v \,\mathrm{d}\boldsymbol{x}$ combined is implemented in `lf::uscalfe::ReactionDiffusionElementMatrixProvider`
- $\int_K f(\boldsymbol{x}) v \,\mathrm{d}\boldsymbol{x}$ is implemented by `lf::fe::ScalarLoadElementVectorProvider`
- $\int_e f(\boldsymbol{x}) v \,\mathrm{d}S$ is implemented by `lf::fe::ScalarLoadEdgeVectorProvider`. Note again the integration over an edge.

Note that the last two are actually element vector providers.

The following formula is usually used when computing element matrices by hand:

Lemma 2.7.5.5: Integration of powers of barycentric coordinate functions
For a d-simplex K (line in 1D, triangle in 2D, tetrahedron in 3D) with barycentric coordinate functions $\lambda_1, \ldots, \lambda_{d+1}$

$$\int_K \lambda_1^{\alpha_1} \cdot \cdots \cdot \lambda_{d+1}^{\alpha_{d+1}} \,\mathrm{d}\boldsymbol{x} = d!|K| \frac{\alpha_1! \cdot \cdots \cdot \alpha_{d+1}!}{(\alpha_1 + \cdots + \alpha_{d+1} + d)!}, \quad \alpha_i \in \mathbb{N}, \tag{24}$$

where $|K|$ is the volume of the simplex.

Quadrature rules

$$\int_K f(x)\,\mathrm{d}x \approx \sum_{l=1}^{P_K} w_l^K f\left(\zeta_l^K\right), \quad w_l^K \to \text{weights}, \zeta_l^K \to \text{(quadrature) nodes}$$

Order of a quadrature rule: a quad rule is of order q if

- for a simplex K, it is exact for all polynomials $f \in \mathcal{P}_{p-1}\left(\mathbb{R}^d\right)$
- for a tensor product element K, it is exact for all polynomials $f \in \mathcal{Q}_{p-1}\left(\mathbb{R}^d\right)$

Essential Boundary Conditions

Remember from Section 1.7 that essential boundary conditions are Dirichlet boundary conditions, i.e., $u = g$ on $\partial\Omega$, and can be solved with the offset function trick. This trick can also be used in FEM. Assume that the basis functions are sorted such that all interior ones come first, followed by the ones on the boundary (we are free to choose the order of basis functions). Then, it is possible to write $\boldsymbol{A}$ as follows:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_{0\partial} \\ \boldsymbol{A}_{0\partial}^\top & \boldsymbol{A}_{\partial\partial} \end{bmatrix}$$

where $\boldsymbol{A}_0$ is the Galerkin matrix for $\mathcal{S}_{p,0}^0(\mathcal{M})$, containing the interactions among interior basis functions. $\left(\boldsymbol{A}_{0\partial}\right)_{ij} = a\left(b_h^j, b_h^i\right)$, where b_h^j belongs to the boundary and b_h^i to the interior, so $\boldsymbol{A}_{0\partial}$ contains the interactions of interior with boundary functions. Similarly, $\boldsymbol{A}_{\partial\partial}$ consists only of entries calculated from basis functions of the boundary. Then we want to solve:

$$\begin{bmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_{0\partial} \\ \boldsymbol{A}_{0\partial}^\top & \boldsymbol{A}_{\partial\partial} \end{bmatrix} \begin{bmatrix} \vec{\boldsymbol{\mu}}_0 \\ \vec{\boldsymbol{\mu}}_\partial \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{\varphi}} \\ \vec{\boldsymbol{\varphi}}_\partial \end{bmatrix}$$

where $\vec{\boldsymbol{\mu}}_\partial$ are the coefficients of the basis expansion of g on the boundary, which are known since g is given. We only need to solve for $\vec{\boldsymbol{\mu}}_0$ which results in

$$\boldsymbol{A}_0 \vec{\boldsymbol{\mu}}_0 = \vec{\boldsymbol{\varphi}} - \boldsymbol{A}_{0\partial} \vec{\boldsymbol{\mu}}_\partial$$

This can be done in LehrFEM++ with lf::assemble::FixFlaggedSolutionComponents or lf::assemble::FixFlaggedSolutionCompAlt. Both modify the matrix $\boldsymbol{A}$ and RHS vector $\vec{\boldsymbol{b}}$ such that $\boldsymbol{A}\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{b}}$ has the same solution as the above equation, but do it in slightly different ways (see docs).

Boundary data in LehrFEM

`FixFlaggedSolutionComponents` (in both versions) requires a lambda function `std::pair<bool, double> selector(unsigned int dof_idx)`. It returns wether the dof is to be fixed and if so, the value it should be fixed to.

We get the boundary flags with

```
auto bd_flags = lf::mesh::utils::flagEntitiesOnBoundary(dofh.Mesh(), 2);
```

The second argument is the codim of entities we are interested in. To fix all dofs on the boundary (including those associated with edges and cells), give no second argument.

1. One function g for the whole boundary

To get the function values of g at the boundary nodes, wrap it in a `MeshFunction`:

```
auto mf_g = lf::mesh::utils::MeshFunctionGlobal(g);
auto boundary_val =
  lf::fe::InitEssentialConditionFromFunction(*fe_space, bd_flags, mf_g);
```

This gives a `std::vector<std::pair<bool, scalar_t>>`. To get our selector:

```
auto selector = [&](unsigned int dof_idx) -> std::pair<bool, double> {
  return boundary_val[dof_idx];
};
```

If g has different definitions on different parts of the boundary (e.g., $g = 0$ on Γ_0 and $g = 1$ on Γ_1), try to express g as a lambda function with an if-else statement.

2. Constant value

The selector becomes

```
auto selector = [&](unsigned int dof_idx) -> std::pair<bool, double> {
  if (bd_flags[dof_idx]) {
    return std::make_pair(true, boundary_value);
  } else {
    return std::make_pair(false, 0.0); // value irrelevant
  }
};
```

3. BC only on part of the boundary

Wee need to create our own `bd_flags`. Initialize it with default value `false`:

```
lf::mesh::utils::AllCodimMeshDataSet<bool> bd_flags(mesh_p, false);
```

Then loop over nodes and edges *separately* and set `bd_flags` to `true` for the nodes/edges where the BC should be applied.

```
for (const auto& edge : fe_space->Mesh()->Entities(1)) {
  if (...) bd_flags(*edge) = true;
}
for (const auto& node : fe_space->Mesh()->Entities(2)) {...}
```

2.8 Parametric Finite Element Methods

Lemma 2.7.5.14: Affine transformation of triangles
For any triangle K s.t. $|K| > 0$, there is a unique affine transformation $\Phi_K(\hat{\boldsymbol{x}}) = F_K\hat{\boldsymbol{x}} + \tau_K$, with $K = \Phi_K\left(\widehat{K}\right)$ and $\widehat{K}$ the unit triangle.

This is nice since it allows us to transform integrals on an arbitrary triangle to ones on the unit triangle and perform easy integration there. Additionally, Φ_K can be computed straightforwardly:

$$\text{Let } K \text{ be a triangle with vertices } a^1, a^2, a^3. \text{ Then } \Phi_K(\hat{\boldsymbol{x}}) = \begin{bmatrix} a_1^2 - a_1^1 & a_1^3 - a_1^2 \\ a_2^2 - a_2^1 & a_2^3 - a_2^2 \end{bmatrix} \hat{\boldsymbol{x}} + \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$$

Definition 2.8.1.2: Pullback
Given domains $\Omega, \widehat{\Omega} \subset \mathbb{R}^d$ and a bijective mapping $\Phi : \widehat{\Omega} \to \Omega$, the *pullback* $\Phi^* u$ of a function $u : \Omega \to \mathbb{R}$ is a function on $\widehat{\Omega}$ defined by $\left(\Phi^* u\right)(\hat{\boldsymbol{x}}) := u(\Phi(\hat{\boldsymbol{x}})), \hat{\boldsymbol{x}} \in \widehat{\Omega}$.

An example for this is the affine transformation of triangles (Lemma 2.7.5.14) above. Note that in the following we will use $\hat{x}$ for an element that "lives" in a reference triangle (or quadrilateral). The reference triangle has corners $\{(0,0),(1,0),(0,1)\}$; the reference quad is the unit square.

When dealing with *parametric* finite element methods, we know that

$$\hat{b}^i_{\widehat{K}} = \Phi_k^* b^i_K,$$

where $b^i_K : K \to \mathbb{R}$ are the local basis functions on the concrete element K and $\hat{b}^i_{\widehat{K}}$ are the local basis functions on the reference element $\widehat{K}$ that matches K.

To make life easier, we call everything on the reference element **local** and everything on the concrete element **global**.

Note that all bilinear forms and linear forms in this course consist of integrals, so we will sooner or later need to use quadrature to approximate the integrals. But in the literature, quadrature rules are defined on reference elements. Hence, we want a change of variables in the integrals of the linear and bilinear forms such that we can apply these quadrature rules. This change of variable is given by the pullback function.

For the simple example of a mass matrix we get (by some multidimensional analysis)

$$\left(A_K\right)_{ij} = \int_K b^j_K(x) b^i_K(x)\,\mathrm{d}x = \int_{\widehat{K}} \left(\Phi^*_K b^j_K\right)(\hat{x})\left(\Phi^*_K b^i_K\right)(\hat{x}) \sqrt{\det(D\Phi^\top_K(\hat{x}) D\Phi_K(\hat{x}))}\,\mathrm{d}x, \tag{25}$$

where we have simply applied the pullback to both functions, switched to integration on $\widehat{K}$, and added an integration element. Similarly, for the diffusion matrix we get

$$\begin{aligned} &\int_K \mathbf{grad}_{\boldsymbol{x}} b^j_K(x) \cdot \mathbf{grad}_{\boldsymbol{x}} b^i_K(x)\,\mathrm{d}x \\ &\qquad = \int_{\widehat{K}} \Phi^*_K\left(\mathbf{grad}_{\boldsymbol{x}} b^j_K\right)(\hat{x}) \cdot \Phi^*_K(\mathbf{grad}_{\boldsymbol{x}} b^i_K)(\hat{x}) \sqrt{\det(D\Phi^\top_K(\hat{x}) D\Phi_K(\hat{x}))}\,\mathrm{d}x. \end{aligned} \tag{26}$$

by pulling back the global gradients (denoted by $\mathbf{grad}_{\boldsymbol{x}}$) of the global shape functions.

Why is the integration element $\det\left(D\Phi_K^\top(\hat{x})D\Phi_K(\hat{x})\right)$ more complicated than what we know from Theorem 0.3.2.31? It is needed when Ω and $\widehat{\Omega}$ do not live in the same space. If, for example, $\Omega \subset \mathbb{R}^3$ describes a 2D plane "living" in 3D space and $\widehat{\Omega} \subset \mathbb{R}^2$, we will have $D\Phi_K \in \mathbb{R}^{3\times 2}$ and hence $\det(\Phi_K)$ is not defined, so the full term is used. In the simpler case of square Φ, this simplifies to $\det(\Phi_K)$. For both cases, the term is provided by `lf::geometry::IntegrationElement`.

The next thing which needs clarification is $\Phi_K^*(\mathbf{grad}_{\boldsymbol{x}} b_K^i)$, pullback of the gradient. As the gradient $\mathbf{grad}_{\boldsymbol{x}} b_K^i(x)$ depends on the shape of K, it would be complicated to compute this term directly.

Therefore we use

Lemma 2.8.3.10: Transformation formula for gradients
For differentiable $u : K \to \mathbb{R}$ and any diffeomorphism $\Phi : \widehat{K} \to K$ we have

$$\left(\mathbf{grad}_{\hat{\boldsymbol{x}}} \Phi^* u\right)(\hat{\boldsymbol{x}}) = (D\Phi(\hat{\boldsymbol{x}}))^\top \underbrace{[(\mathbf{grad}_{\boldsymbol{x}} u)(\Phi(\hat{x}))]}_{(\Phi^* \mathbf{grad}_{\boldsymbol{x}})u}(\hat{\boldsymbol{x}})$$

In words: on the left side we have **local gradient of the pullback** of u, on the right side we have **pullback of the global gradient** of u.

This is exactly what we need since the pullback of the global gradient occurs in (26) and the local gradient of local shape functions is easy to compute.

Now we need to get rid of $(D\Phi(\hat{x}))^\top$ on the right. Let's define $J = D\Phi(\hat{x})$. Since J might not be square, we cannot simply invert it. But we can multiply by $J\left(J^\top J\right)^{-1}$ to get

$$\Phi_K^*(\mathbf{grad}_{\hat{\boldsymbol{x}}} b_K^i) = J\left(J^\top J\right)^{-1}\left(\mathbf{grad}_{\hat{\boldsymbol{x}}} \hat{b}_{\widehat{K}}^i\right)$$

$\hat{b}(\hat{x})$ is a reference basis function on the reference shape — we can compute its gradient easily by hand.

Note that in the script, the case of square J is assumed, such that this simplifies to $J^{-\top}\left(\mathbf{grad}_{\hat{\boldsymbol{x}}} \hat{b}_{\widehat{K}}^i\right)$. The long term only matters when we have (as above) Ω and $\widehat{\Omega}$ not living in the same space. The same remedy applies here, you can in any case use `lf::geometry::JacobianverseGramian` which will return $J^{-\top}$ or $J\left(J^\top J\right)^{-1}$, respectively.

Bilinear Transformation for Quadrilaterals Let $\{\boldsymbol{a^1}, \boldsymbol{a^2}, \boldsymbol{a^3}, \boldsymbol{a^4}\}$ be the ordered corners of a quadrilateral. Then

$$\begin{aligned}\Phi_K(\hat{x}) &= (1-\hat{x}_1)(1-\hat{x}_2)\boldsymbol{a^1} + \hat{x}_1(1-\hat{x}_2)\boldsymbol{a^2} + (1-\hat{x}_1)\hat{x}_2\boldsymbol{a^3} + (1-\hat{x}_1)\hat{x}_2\boldsymbol{a^4} \\ &= \begin{bmatrix} \alpha_1 + \beta_1\hat{x}_1 + \gamma_1\hat{x}_2 + \delta_1\hat{x}_1\hat{x}_2 \\ \alpha_2 + \beta_2\hat{x}_1 + \gamma_2\hat{x}_2 + \delta_2\hat{x}_1\hat{x}_2 \end{bmatrix}\end{aligned}$$

with

$$\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix} = \boldsymbol{a^1}, \begin{bmatrix}\beta_1 \\ \beta_2\end{bmatrix} = \boldsymbol{a^2} - \boldsymbol{a^1}, \begin{bmatrix}\gamma_1 \\ \gamma_2\end{bmatrix} = \boldsymbol{a^4} - \boldsymbol{a^1}, \begin{bmatrix}\delta_1 \\ \delta_2\end{bmatrix} = \boldsymbol{a^4} - \boldsymbol{a^3} - \boldsymbol{a^2} + \boldsymbol{a^1}$$

3 FEM: Convergence and Accuracy

3.1 Abstract Galerkin Error Estimates

The main takeaway here is that the solution given by the Galerkin method is the best one (with respect to the energy norm) in the chosen discrete subspace. This is formalized by the following lemma:

Theorem 3.1.3.7: Cea's Lemma
Under some assumptions that guarantee the existence of a unique solution we have

$$\|u - u_h\|_a = \inf_{v_h \in V_{0,h}} \|u - v_h\|_a, \tag{27}$$

where u is the exact solution and u_h is the Galerkin solution.

Next we want to discuss the types of refinement, i.e., the steps we can take to increase the accuracy of our method.

h-Refinement
Replace the mesh $\mathcal{M}$ (underlying $V_{0,h}$) with a finer mesh $\mathcal{M}'$ (underlying larger discrete trial space ${V'}_{0,N'}$).

p-Refinement
Replace $V_{0,h} := S_p^0(\mathcal{M}), p \in \mathbb{N}$, with $V'_{0,h} := S_{p+1}^0(\mathcal{M})$, yielding a larger space: $V_{0,h} \subset V'_{0,h}$

So h-refinement refines the mesh (smaller and smaller triangles) and p-refinement chooses more powerful basis functions (start with linear, then quadratic, etc.). The h in h-refinement refers to the mesh width:

Definition 3.2.1.4: Mesh width
Given a mesh $\mathcal{M} = \{K\}$, the **mesh width** $h_{\mathcal{M}}$ is defined as

$$h_{\mathcal{M}} := \max\{\text{diam } K : K \in \mathcal{M}\}$$
$$\text{diam } K := \max\{|p - q| : p, q \in K\}$$

3.2 Empirical (Asymptotic) Convergence of Lagrangian FEM

As in NumCSE, there are two types of convergence, algebraic and exponential. We refer to the number of basis functions (dimension of the trial space) as N and we study the behavior of errors as $N \to \infty$. Note that both exercises and theorems are often posed in terms of $h_{\mathcal{M}}$ as $h \to 0$, which is equivalent if we have a fixed polynomial degree p and only do h-refinement.

Definition 3.2.2.1: Types of convergence

$\|u - u_N\|_a = \mathcal{O}(N^{-\alpha}), \alpha > 0$ is called **algebraic** convergence with rate α.

$\|u - u_N\|_a = \mathcal{O}\big(\exp\big(-\gamma N^{\delta}\big)\big), \gamma, \delta > 0$ is called **exponential** convergence.

Determining convergence rates

- **algebraic**

$$\alpha \approx \frac{\log \epsilon_{i-1} - \log \epsilon_i}{\log N_i - \log N_{i-1}} = \frac{\log(\epsilon_i/\epsilon_{i-1})}{\log(h_i/h_{i-1})}$$

- **exponential** In general: complicated (see §3.2.2.5 Lecture Notes)

  If $\delta = 1$ (plain exponential convergence) and we let N increase linearly (e.g., $N_i = i$),

$$\frac{\epsilon_{i+1}}{\epsilon_i} \approx \exp(-\gamma)$$

Note that in the case of h-refinement we get the relation between N and h given by

Equation 3.2.2.1

$$N = \dim S_p^0(\mathcal{M}) \approx p^d h_{\mathcal{M}}^{-d} \Longrightarrow \frac{h_{\mathcal{M}}}{p} \approx N^{-\frac{1}{d}} \tag{28}$$

where p are the dimensions of the local basis functions, e.g., $p = 1$ for linear basis functions, and d is the dimension of the underlying space Ω.

E.g., in the case where we have $\Omega \subset \mathbb{R}^2$ and piecewise linear basis functions, we get $h_{\mathcal{M}}^{-2} \approx N$.

3.3 A Priori (Asymptotic) Finite Element Error Estimates

Since FEM is similar to polynomial interpolation, we can use interpolation error estimates to get error bounds. Here are some results for linear interpolation:

Linear interpolation error 1D

Using the linear interpolant I_1 we want to study the interpolation error $u - I_1 u$. The following interpolation error estimates can be used for sufficiently smooth functions u:

$$\|u - I_1 u\|_{L^\infty(]a,b[)} \leq \frac{1}{4} h_{\mathcal{M}}^2 \|u''\|_{L^\infty(]a,b[)} \tag{29}$$

$$\|u - I_1 u\|_{L^2(]a,b[)} \leq h_{\mathcal{M}}^2 \|u''\|_{L^2(]a,b[)} \tag{30}$$

$$|u - I_1 u|_{H^1(]a,b[)} \leq h_{\mathcal{M}} \|u''\|_{L^2(]a,b[)} \tag{31}$$

Linear interpolation error 2D
In 2D, linear interpolation corresponds to using tent functions: $I_1 u = \sum_{p \in \mathcal{V}(\mathcal{M})} u(p) b^p$, where b^p is the tent function associated with point p.

$$\begin{aligned} \|u - I_1 u\|_{L^2(\Omega)} &\leq C h_{\mathcal{M}}^2 \left\| \|D^2 u\|_F \right\|_{L^2(\Omega)} \\ \|\mathbf{grad}\,(u - I_1 u)\|_{L^2(\Omega)} &\leq C \rho_{\mathcal{M}} h_{\mathcal{M}} \left\| \|D^2 u\|_F \right\|_{L^2(\Omega)} \end{aligned} \tag{32}$$

Here, $D^2 u$ is the Hessian of u, $\|\cdot\|_F$ is the Frobenius norm, and $\rho_{\mathcal{M}}$ is the shape regularity measure of the mesh $\mathcal{M}$, defined as $\rho_{\mathcal{M}} = \max_{K \in \mathcal{M}} \frac{h_{\mathcal{M}}^2}{|K|}$ for a triangular mesh.

To get rid of this cumbersome notation, we can introduce more Sobolev spaces.

Definition 3.3.3.1: Higher order Sobolev spaces/norms
The m-th order Sobolev norm is defined as

$$\|u\|_{H^m(\Omega)}^2 = \sum_{k=0}^{m} \sum_{\boldsymbol{\alpha} \in \mathbb{N}^d, |\boldsymbol{\alpha}| = k} \int_\Omega |D^{\boldsymbol{\alpha}} u|^2 \, \mathrm{d}\boldsymbol{x}, \qquad \text{where } D^{\boldsymbol{\alpha}} u = \frac{\partial^{|\boldsymbol{\alpha}|} u}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} \tag{33}$$

Hence we can define the m-th Sobolev space as

$$H^m(\Omega) = \left\{ v : \Omega \to \mathbb{R} : \|v\|_{H^m(\Omega)} < \infty \right\} \tag{34}$$

Definition 3.3.3.3: Higher order Sobolev semi-norms
The m-th order Sobolev semi-norm is defined as

$$|u|_{H^m(\Omega)}^2 = \sum_{\boldsymbol{\alpha} \in \mathbb{N}^d, |\boldsymbol{\alpha}| = m} \int_\Omega |D^{\boldsymbol{\alpha}} u|^2 \, \mathrm{d}\boldsymbol{x} \tag{35}$$

Remember the multi-index $\boldsymbol{\alpha}$ already seen in Definition 2.5.2.2. Using this new notation, we can rewrite the error bounds from (32) as

$$\begin{aligned} \|u - I_1 u\|_{L^2(\Omega)} &\leq C h_{\mathcal{M}}^2 |u|_{H^2(\Omega)} \\ \|\mathbf{grad}\,(u - I_1 u)\|_{L^2(\Omega)} &\leq C \rho_{\mathcal{M}} h_{\mathcal{M}} |u|_{H^2(\Omega)} \end{aligned}$$

It turns out that these bounds are not sharp. There is a very useful result for Lagrangian finite elements:

Theorem 3.3.5.6: Best approximation error estimates for Lagrangian finite elements
Given a triangular mesh $\mathcal{M}$, if the true solution u is in $H^k(\Omega)$, the best approximation error is bounded by

$$\inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \leq C \left(\frac{h_{\mathcal{M}}}{p} \right)^{\min\{p, k-1\}} \|u\|_{H^k(\Omega)} \tag{36}$$

We might not know the constant C and/or $\|u\|_{H^k(\Omega)}$, but we know p and $h_{\mathcal{M}}$ as they are imposed by the choice of function space and mesh. Remember the concept of refinement: we can adjust these values. And from Eq. (28) we know that $h_{\mathcal{M}}/p \approx N^{-\frac{1}{d}}$. Hence the error displays **algebraic** convergence with rate $\min\{p, k-1\}/d$. What still remains a question is k, the smoothness of the solution u.

Another useful result hidden in the lecture notes is

Equation 3.6.3.10 (L^2 estimate)
Under some assumptions (convex domain, smooth coefficient functions), we have

$$\|u - u_h\|_{L^2(\Omega)} \leq C \frac{h_{\mathcal{M}}}{p} \|u - u_h\|_{H^1(\Omega)} \tag{37}$$

So we gain one order of convergence in the L^2 norm compared to the H^1 norm.

Rules of thumb for converge

If we are using $\mathcal{S}_p^0(\mathcal{M})$ and u is sufficiently smooth (e.g., $u \in C^\infty(\Omega)$), we have

$$\begin{aligned} \|u - u_h\|_{H^1(\Omega)} &= \mathcal{O}(h^p) \\ |u - u_h|_{H^1(\Omega)} &= \mathcal{O}(h^p) \\ \|u - u_h\|_{L^2(\Omega)} &= \mathcal{O}(h^{p+1}) \end{aligned}$$

3.4 Elliptic regularity

Theorem 3.4.0.2: Smooth elliptic lifting theorem
For domains Ω with smooth boundaries $\partial\Omega$, i.e. no corners and sufficiently smooth σ, if

$$u \in H_0^1(\Omega) \quad \text{and} \quad -\operatorname{div}(\sigma \, \mathbf{grad}\, u) \in H^k(\Omega)$$

or

$$u \in H^1(\Omega), -\operatorname{div}(\sigma \, \mathbf{grad}\, u) \in H^k(\Omega) \quad \text{and} \quad \mathbf{grad}\, u \cdot \boldsymbol{n} = 0 \quad \text{on } \partial\Omega$$

holds, then $u \in H^{k+2}(\Omega)$ and

$$\|u\|_{H^{k+2}(\Omega)} \leq C \|\operatorname{div}(\sigma \, \mathbf{grad}\, u)\|_{H^k(\Omega)}$$

This tells us that when solving the typical PDE $-\operatorname{div}(\sigma\ \mathbf{grad}\, u) = f$ and the source term f is in $H^k(\Omega)$, the solution u will be in $H^{k+2}(\Omega)$ (of course under the right assumptions).

The theorem requires smooth domains, but our meshes will have corners, so what can be done there? As long as the domain and all cells are convex, something similar still holds:

Theorem 3.4.0.10: Elliptic lifting on convex domains
If $\Omega \subset \mathbb{R}^d$ is convex, $u \in H_0^1(\Omega)$ and $\Delta u \in L^2(\Omega)$, then $u \in H^2(\Omega)$.

If we are solving the Laplace equation on a convex domain, we just need to check if $f \in L^2(\Omega)$, since $-\Delta u = f$.

Finding k in Theorem 3.3.5.6

- Often, the exercise gives you u which is in $C^\infty(\Omega)$, which means u is infinitely smooth. In this case, $k = \infty$.
- Sometimes, you may use elliptic regularity results like Theorem 3.4.0.10 to find $k = 2$, for example.

3.5 Variational Crimes

What are variational crimes?

A variational crime is committed when instead of the true variational problem

$$u_h \in V_{0,h} : a(u_h, v_h) = \ell(v_h), \forall v_h \in V_{0,h},$$

we solve a different, "perturbed" variational problem

$$\tilde{u}_h \in V_{0,h} : a_h(\tilde{u}_h, v_h) = \ell_h(v_h), \forall v_h \in V_{0,h}$$

with modified (bi-)linear forms a_h, ℓ_h. With computers, the use of quadrature and approximation of boundaries result in such a crime and are unavoidable. As Hiptmair likes to say, "we are all sinners".

So the only distinction we can make is between acceptable and unacceptable "crimes". Crimes which do not affect the type and rate of convergence are acceptable.

So how to not temper with the convergence?

- if $\|u - u_h\|_1 \in \mathcal{O}(h_{\mathcal{M}}^p)$, use a quadrature rule of order at least $2p - 1$
- if $V_{0,h} = \mathcal{S}_p^0(\mathcal{M})$ then approximate the boundary with polynomials of degree p

3.6 FEM: Duality Techniques for Error Estimation

Theorem 3.6.1.7: Duality estimate for linear functional output
Given a functional $F : V_0 \to \mathbb{R}$ the dual solution g_F solves

$$g_F \in V_0 : a(g_F, v) = F(v), \forall v \in V_0$$

and we get the estimate

$$|F(u) - F(u_h)| \leq C\|u - u_h\|_a \inf_{v_h \in V_{0,h}} \|g_F - v_h\|_a$$

Why is this useful? If g_F can be approximated well in $V_{0,h}$, then the output error $|F(u) - F(u_h)|$ can converge to 0 much faster than $\|u - u_h\|_a$.

5 Non-Linear Elliptic Boundary Value Problems

5.1 Elastic String Model

We want to derive the general variational equation for an elastic string. For this, one approximates the string as n point masses affected by gravity connected with springs, whose energy behaves according to Hooke's law. Then, one takes the limit $n \to \infty$ to derive a continuous model. The total energy is then just given by the sum of elastic and gravitational energies — given positions $(\mu_0, ..., \mu_n)$ and $x_i = a + hi$, assuming the spring constants are 1:

Total energy of discrete spring system

$$E(\mu) = \frac{1}{2} \sum_{i=0}^{n} \left(\sqrt{h^2 + (\mu_{i+1} - \mu_i)^2} \right)^2 + \sum_{i=1}^{n} m_i \mu_i g$$

Then, the equilibrium position for this model can be found by minimizing this expression over μ. A continuous model is derived by replacing the discrete positions μ_i by a function $u(x_i)$, and the mass by a mass density. Then, performing some manipulations, one obtains

Total energy for the continuous string model

$$J_s(u) = \int_a^b \frac{1}{2} \frac{b-a}{L} \sigma(x) \left(\sqrt{1 + |u'(x)|^2} - \frac{L}{b-a} \right)^2 + g\rho(x)u(x) \, \mathrm{d}x$$

where σ is the spring stiffness, L is the total rest length of the string, and ρ is the mass density. The term $\sqrt{1 + |u'(x)|^2}$ is the length of the spring at position x.

In a similar fashion, a membrane model can be derived by assuming a two-dimensional grid of springs containing points masses and taking the limit $n \to \infty$ springs. The energy then becomes

Total energy for the membrane model

$$\begin{aligned} J_M(u) = \int_\Omega \frac{1}{2L} \sigma(\boldsymbol{x}) \Bigg(& \left(\sqrt{1 + \left| \frac{\partial u}{\partial x_1}(\boldsymbol{x}) \right|^2} - \frac{L}{b-a} \right)^2 \\ & + \left(\sqrt{1 + \left| \frac{\partial u}{\partial x_2}(\boldsymbol{x}) \right|^2} - \frac{L}{b-a} \right)^2 \Bigg) + g\rho(\boldsymbol{x}) u(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \end{aligned}$$

In the limit of a taut membrane, i.e. for $L \ll b - a$, these equations just reduce to the problem of minimizing the familiar functionals seen in earlier chapters:

Elastic string/membrane: taut limit

$$\begin{aligned} J_s(u) &= \frac{1}{2}\int_a^b \hat{\sigma}(x)|u'(x)|^2 + g\rho(x)u(x)\,\mathrm{d}x \\ J_M(u) &= \frac{1}{2}\int_\Omega \hat{\sigma}(\boldsymbol{x})\|\mathbf{grad}\, u(\boldsymbol{x})\|^2 + g\rho(\boldsymbol{x})u(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \end{aligned} \tag{38}$$

5.2 Calculus of Variations

The difference between the equations seen in (38), which hold in the limit of a very stretched string/membrane, and the general equations given above, is that the former are **quadratic** minimization problems, while the latter are **nonlinear** minimization problems. The theory used so far mapped quadratic minimization problems to **linear** variational problems, which were then discretized. The new equations, however, yield nonlinear variational equations. Therefore, more general variational problems need to be derived.

The idea employed is that, for a minimizer u_* of $J(u)$, every perturbation $J(u_* + v)$ would be larger than $J(u_*)$. This means that $f(t) = J(u_* + tv)$ has a minimum at $t = 0$ for every function v:

Theorem 5.2.1.5: Characterization of global minimizers
Assume $u_* \in V_0$ is a global minimizer of $J(u)$, i.e.

$$u_* = \operatorname*{argmin}_{u \in V_0} J(u)$$

Then, if $\varphi_v(t) = J(u_* + tv)$ is differentiable in $t = 0$, we have

$$\frac{\mathrm{d}\varphi_v}{\varphi t}(0) = 0 \quad \forall v \in V_0$$

This means that nonlinear variational equations can be derived by computing this derivative for an arbitrary v. As an example, for the elastic string model introduced in the last sub-chapter, this yields

Variational equations for elastic string model

$$\int_a^b \frac{\sigma(x)}{c}\left(\sqrt{1+|u'(x)|^2} - c\right)\frac{u'(x)v'(x)}{\sqrt{1+|u'(x)|^2}} + g\rho(x)v(x)\,\mathrm{d}x = 0 \quad \forall v \in H_0^1(]a,b[)$$

This can be formulated more generally as a **general variational equation**:

General variational equation
A general, nonlinear variational equation reads

$$u \in \widehat{V} : \quad a(u; v) = 0 \quad \forall v \in V_0$$

Where a is **linear in the second argument** v and $V_0, \widehat{V}$ are function spaces.

5.3 Nonlinear Boundary Value Problems

Similarly to the linear case, nonlinear PDEs can be derived from the variational equations by "stripping" of the derivatives of v by partial integration and employing the fundamental lemma of calculus of variations. As an example, for the string model, this yields for $u(a) = u_a$, $u(b) = u_b$

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\sigma(x)}{c}\left(\sqrt{1+|u'(x)|^2}-c\right)\frac{u'(x)}{\sqrt{1+|u'(x)|^2}}\right) = g\rho(x) \qquad \text{in }]a,b[$$

5.4 Galerkin Discretization of Non-Linear BVPs

The idea of Galerkin discretization for non-linear variational equations is exactly the same as for linear equations, but they yield nonlinear systems of equations instead of linear systems of equations: One restricts u and v to a finite function space $u_h \in \widehat{V}_h$ and $v_h \in V_{0,h}$, and expands the functions in some basis of the space. This, then, leads to nonlinear equations for the basis expansion coefficients. These equations could be solved directly by employing some fixed-point iteration seen in NumCSE.

Nonlinear Galerkin Discretization
Given a variational problem $a(u; v) = 0 \; \forall v \in V_{0,h}$, the Galerkin discretization reads

$$\left(F(\mu)\right)_i = a\left(u_{0,h} + \sum_{j=1}^{N} \mu_j b_h^j; b_h^i\right), \quad i = 1, ..., N$$

where b_h^i are fixed basis functions, μ_i are the basis function coefficients and $u_{0,h}$ contains Dirichlet boundary conditions.

Another option is to already linearize the continuous problem, and then discretize it to derive linear systems of equations. This is done by employing Newton's method in function space: The conventional Newton iteration is given as

$$\xi^{(k+1)} = \xi^{(k)} - \mathrm{D}F\left(\xi^{(k)}\right)^{-1} F\left(\xi^{(k)}\right)$$

Replacing the vector ξ with a function u and the derivative by a function derivative now gives

Equation 5.3.2.5 (Functional Newton iteration)

$$w \in V_0 : a\big(u^{(k)}; v\big) + \mathrm{D}_u a\big(u^{(k)}; v\big) w = 0 \quad \forall v \in V_0$$
$$u^{(k+1)} = u^{(k)} + w$$

Here, the directional derivative is defined as

$$\mathrm{D}_u a\big(u^{(k)}; v\big) w = \lim_{t \to 0} \frac{a(u + tw; v) - a(u; v)}{t}, \quad u^{(k)} \in \widehat{V}, \quad v, w \in V_0$$

Computing the directional derivative

You can simply compute the defining limit by Taylor-expanding $a(u + tw, v)$. If you do this once, you notice that only the first-order term in t remains: the 0th order term cancels out with $a(u, v)$, and the higher-order terms vanish in the limit $t \to 0$.

So a more convenient method is to do Taylor expansion and to keep only the first-order term.

Now, the advantage of this equation for w is that the functional derivative is linear, i.e. $(v, w) \mapsto D_u a\big(u^{(k)}; v\big) w$ is a **bilinear form**. Now, one can employ Galerkin discretization for the linear problem in w, exactly like it was done in Chapter 2 and 3. The final equations then read

Equation 5.3.3.6 (Nonlinear Newton equations for variational problems)

$$w_h \in V_{0,h}^{(k)} : \mathrm{D}_u a\Big(u_h^{(k-1)}; v_h\Big) w_h = -a\Big(u_h^{(k-1)}; v_h\Big) \quad \forall v_h \in V_{0,h}^{(k)}$$
$$u_h^{(k)} = P_h^{(k)}\Big(u_h^{(k-1)} + w_h\Big)$$

Here, different function spaces can be used for each iterations, so a projector $P_h^{(k)}$ needs to be used to project the solution from $V_h^{(k-1)}$ to $V_h^{(k)}$. In all of these equations, the previous iterate $u_h^{(k-1)}$ is kept fixed, a linear system like derived in Chapter 2 is solved to obtain the intermediate w_h, and then a new iterate $u^{(k)}$ is obtained.

9 Second-Order Linear Evolution Problems

9.2 Parabolic Initial-Boundary Value Problems

9.2.1 Heat Equation

In local form the heat equation is given by

$$\frac{\partial}{\partial t}(\rho u) - \operatorname{div}(\kappa(\boldsymbol{x}) \operatorname{\mathbf{grad}} u) = f \quad \text{in } \tilde{\Omega} = \Omega \times]0,T[\tag{39}$$

where u is the temperature, ρ the heat capacity, κ the heat conductivity and f a (time-dependent) heat source/sink. Without the time derivative, this looks very similar to standard PDEs, for which we know the transformation into a nice variational problem.

To solve it we still need boundary conditions. Besides the boundary conditions of the spatial domain — which are now required for all times — one also needs initial conditions over the whole domain at time 0.

$$\begin{aligned} u(\boldsymbol{x},t) &= g(\boldsymbol{x},t) \quad \text{for } (\boldsymbol{x},t) \in \partial\Omega \times]0,T[\\ u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \Omega \end{aligned} \tag{40}$$

Testing with time-independent test functions v and assuming ρ to be time independent as well, we get to

$$\begin{gathered} \int_\Omega \rho(\boldsymbol{x}) \dot{u} v \,\mathrm{d}\boldsymbol{x} + \int_\Omega \kappa(\boldsymbol{x}) \operatorname{\mathbf{grad}} u \cdot \operatorname{\mathbf{grad}} v \,\mathrm{d}\boldsymbol{x} = \int_\Omega f(\boldsymbol{x},t)\, v \,\mathrm{d}\boldsymbol{x} \quad \forall v \in H_0^1(\Omega) \\ u(\boldsymbol{x},0) = u_0(\boldsymbol{x}) \in H_0^1(\Omega) \end{gathered} \tag{41}$$

with the shorthand notation

$$\begin{aligned} m(\dot{u},v) &= \int_\Omega \rho(\boldsymbol{x})\, \dot{u}\, v \,\mathrm{d}\boldsymbol{x} \\ a(u,v) &= \int_\Omega \kappa(\boldsymbol{x}) \operatorname{\mathbf{grad}} u \cdot \operatorname{\mathbf{grad}} v \,\mathrm{d}\boldsymbol{x} \\ \ell(v) &= \int_\Omega f(\boldsymbol{x},t)\, v \,\mathrm{d}\boldsymbol{x} \end{aligned}$$

and the realization that $m(\dot{u},v) = \frac{\mathrm{d}}{\mathrm{d}t} m(u,v)$ as only u depends on time (note that importantly, the domain Ω also stays constant) we can rewrite (41) as

$$\frac{\mathrm{d}}{\mathrm{d}t} m(u,v) + a(u,v) = \ell(v) \tag{42}$$

which looks like something we know how to solve from NumCSE.

9.2.3 Stability

Lemma 9.2.3.8.: Decay of solutions of parabolic evolutions
If $f \equiv 0$, the solution $u(t)$ of the heat equation (41) satisfies

$$\|u(t)\|_m \leq e^{-\gamma t}\|u_0\|_m, \quad \|u(t)\|_a \leq e^{-\gamma t}\|u_0\|_a \quad \forall t \in]0,T[$$

where $\gamma = \operatorname{diam}(\Omega)^{-2}$.

Note that this lemma also tells us that if f is time-independent, the solution $u(t)$ converges exponentially (in time) to the stationary solution (the solution of (42) without the $m(\cdot,\cdot)$ part).

9.2.4 Method of Lines

Now let's look into how we can solve (42). Let's apply the Galerkin discretization. As u is now also time-dependent, we let the coefficients of u (and not the basis) depend on time.

$$u_h(t) = \sum_{i=1}^{N} \mu_i(t) b_h^i$$

Combining this with (42), we get

$$\begin{aligned} \boldsymbol{M}\left\{\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu}(t)\right\} + \boldsymbol{A}\vec{\mu}(t) &= \vec{\varphi}(t) \\ \vec{\mu}(0) &= \vec{\mu}_0 \end{aligned} \tag{43}$$

where $\boldsymbol{M}_{i,j} = m\left(b_h^j, b_h^i\right)$, $\boldsymbol{A}_{i,j} = a\left(b_h^j, b_h^i\right)$ and $[\vec{\varphi}(t)]_i = \ell\left(b_h^i\right)$.

This is now an ODE with respect to time and can be solved by time stepping, learned in NumCSE.

Recall ODEs An ODE is given as

$$\dot{\boldsymbol{u}} = \boldsymbol{f}(t, \boldsymbol{u})$$

and is called linear if $\boldsymbol{f}(t,\boldsymbol{u}) = \boldsymbol{A}(t)\boldsymbol{u}$. There is an evolution operator associated with the ODE, defined as $\Phi^{t_0,t}u_0 = u(t)$. There are some methods to approximate the evaluation operator with a discrete evolution operator Ψ.

- explicit Euler: $\Psi^{t,t+\tau}\boldsymbol{u} = \boldsymbol{u} + \tau\boldsymbol{f}(t,\boldsymbol{u})$
- implicit Euler:$\Psi^{t,t+\tau}\boldsymbol{u} = \boldsymbol{w}, \boldsymbol{w} = \boldsymbol{u} + \tau\boldsymbol{f}(t+\tau,\boldsymbol{w})$
- implicit midpoint: $\Psi^{t,t+\tau}\boldsymbol{u} = \boldsymbol{w}, \boldsymbol{w} = \boldsymbol{u} + \tau\boldsymbol{f}(t+\frac{1}{2}\tau, \frac{1}{2}(\boldsymbol{w}+\boldsymbol{u}))$

Hence we can calculate the time evolution by the sequence

$$\boldsymbol{u}^{(0)} = \boldsymbol{u}_0, \quad \boldsymbol{u}^{(j)} = \Psi^{t_{j-1},t_j}\boldsymbol{u}^{(j-1)}, \quad j = 1,\ldots,M$$

As Ψ is the discrete approximation, the question about the error is immediate. One usually considers

- the error at final time: $\epsilon_M = \left\|\boldsymbol{u}(T) - \boldsymbol{u}^{(M)}\right\|$
- maximum error in the sequence: $\epsilon_\infty = \max_j \left\|\boldsymbol{u}^{(j)} - \boldsymbol{u}\left(t_j\right)\right\|$

Theorem 9.2.6.14: Convergence of single-step methods
Given the above sequence of solutions, obtained by a single step method of order $q \in \mathbb{N}$, then

$$\epsilon_\infty = \max_j \left\| \boldsymbol{u}^{(j)} - \boldsymbol{u}(t_j) \right\| \leq C\tau^q$$

with $\tau = \max_j |t_j - t_{j-1}|$.

Runge–Kutta Single-Step Methods

Definition 7.3.3.1: General Runge–Kutta single-step method
For coefficients $b_i, a_{i,j} \in \mathbb{R}, c_i = \sum_{j=1}^s a_{i,j}$, the discrete evolution operator $\Psi^{s,t}$ of an **s-stage Runge–Kutta single step method** (RK-SSM) for the ODE $\dot{\boldsymbol{u}} = \boldsymbol{f}(t, \boldsymbol{u})$ is defined by

$$\boldsymbol{k}_i = \boldsymbol{f}\left(t + c_i\tau, \boldsymbol{u} + \tau \sum_{j=1}^s a_{i,j}\boldsymbol{k}_j\right), \quad i = 1, ..., s, \quad \Psi^{t,t+\tau}\boldsymbol{u} = \boldsymbol{u} + \tau \sum_{j=1}^s b_j \boldsymbol{k}_j$$

with $\boldsymbol{k}_j$ the increments.

The RK-SSM methods can be written down in compact form (the butcher scheme) as

$$\begin{array}{c|c} \boldsymbol{c} & \mathfrak{A} \\ \hline & \boldsymbol{b} \end{array} \tag{44}$$

where $\boldsymbol{c}$ is a vector containing the coefficients c_i, $\boldsymbol{b}$ the coefficients b_i and $\mathfrak{A}$ a matrix containing the coefficients $a_{i,j}$.

So continuing from (43) with different time stepping schemes, we get

- explicit Euler:

$$\vec{\mu}^{(j)} = \vec{\mu}^{(j-1)} + \tau_j \boldsymbol{M}^{-1}\left(\vec{\varphi}(t_{j-1}) - \boldsymbol{A}\vec{\mu}^{(j-1)}\right)$$

- implicit Euler:

$$\vec{\mu}^{(j)} = \left(\tau_j \boldsymbol{A} + \boldsymbol{M}\right)^{-1}\left(\boldsymbol{M}\vec{\mu}^{(j-1)} + \tau_j \vec{\varphi}(t_{j-1})\right)$$

- implicit midpoint (Crank-Nicolson)

$$\vec{\mu}^{(j)} = \left(\boldsymbol{M} + \frac{1}{2}\boldsymbol{A}\right)^{-1} \tau_j \left(\left(\boldsymbol{M} - \frac{1}{2}\boldsymbol{A}\right)\vec{\mu}^{(j-1)} + \frac{1}{2}\left(\vec{\varphi}(t_j) + \vec{\varphi}(t_{j-1})\right)\right)$$

These all involve solving a linear system of equations each time step. However note, that the matrices to invert stay constant with respect to time, so we can calculate the decomposition only once to save a lot of time.

Using a general RK-SSM method as the time step, we get the following system of equations

$$\boldsymbol{M}\vec{\kappa}_i + \sum_{m=1}^{s} \tau a_{i,m} \boldsymbol{A}\vec{\kappa}_m = \vec{\varphi}\big(t_j + c_i\tau\big) - \boldsymbol{A}\vec{\mu}^{(j)}$$

$$\vec{\mu}^{(j+1)} = \vec{\mu}^{(j)} + \tau \sum_{m=1}^{s} b_m \vec{\kappa}_m$$

With the Kronecker product, this can be rewritten as

$$(\boldsymbol{I}_s \otimes \boldsymbol{M} + \tau \mathfrak{A} \otimes \boldsymbol{A}) \begin{bmatrix} \vec{\kappa}_1 \\ \vdots \\ \vec{\kappa}_s \end{bmatrix} = \begin{bmatrix} \vec{\varphi}\big(t_j + c_1\tau\big) - \boldsymbol{A}\vec{\mu}^{(j)} \\ \vdots \\ \vec{\varphi}\big(t_j + c_s\tau\big) - \boldsymbol{A}\vec{\mu}^{(j)} \end{bmatrix}$$

which can be used to solve for the increments $\vec{\kappa}_i$.

Recall stiff initial value problems:

Stiffness
An initial value problem is called stiff if stability imposes much tighter timestep constraints on explicit single step methods than the accuracy requirements.

To study the stiffness of the method of lines, we first diagonalize it. For (43) let $\vec{\psi}_1, ..., \vec{\psi}_N$ denote the N linearly independent generalized eigenvectors satisfying

$$\boldsymbol{A}\vec{\psi}_i = \lambda_i \boldsymbol{M}\vec{\psi}_i, \quad \left(\vec{\psi}_j\right)^\top \boldsymbol{M}\vec{\psi}_i = \delta_{ij}$$

with positive eigenvalues λ_i. With $\boldsymbol{T} = \left[\vec{\psi}_1, ..., \vec{\psi}_N\right]$ and $\boldsymbol{D} = \operatorname{diag}\,(\lambda_1, ..., \lambda_N)$, this can be rewritten as

$$\boldsymbol{A}\boldsymbol{T} = \boldsymbol{M}\boldsymbol{T}\boldsymbol{D}, \quad \boldsymbol{T}^\top \boldsymbol{M}\boldsymbol{T} = \boldsymbol{I}$$

The existence of eigenvectors with positive eigenvalues is guaranteed, as $\boldsymbol{A}, \boldsymbol{M}$ are positive (semi)definite. Thus with a change of basis to the eigenvector basis, one can diagonalize (43).

$$\vec{\mu}(t) = \sum_k \eta_k(t)\vec{\psi}_k \Leftrightarrow \vec{\mu}(t) = \boldsymbol{T}\vec{\eta}(t) \Leftrightarrow \boldsymbol{T}^\top \boldsymbol{M}\vec{\mu}(t) = \vec{\eta}(t)$$

$$\rightarrow \boldsymbol{M}\boldsymbol{T}\frac{\mathrm{d}}{\mathrm{d}t}\vec{\eta}(t) + \boldsymbol{M}\boldsymbol{T}\boldsymbol{D}\vec{\eta}(t) = \vec{\varphi}(t)$$

$$\rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\vec{\eta}(t) + \boldsymbol{D}\vec{\eta}(t) = \boldsymbol{T}^\top \vec{\varphi}(t)$$

As $\boldsymbol{D}$ is diagonal, this amounts to N decoupled scalar ODEs. On those, we can perform our analysis more easily. In NumCSE you have seen that both Euler and Crank-Nicolson can be rewritten as a RK-SSM with appropriate coefficients, so we can study the stability of the general RK-SSM for the scalar case. For $\dot{u} = -\lambda u$, with the butcher scheme (44) we obtain $\Psi_\lambda^{t,t+\tau} u = S(-\lambda\tau)u$, with the stability function

$$S(z) = 1 + z\boldsymbol{b}^\top (I - z\mathfrak{A})^{-1}\mathbf{1} = \frac{\det\,\left(\boldsymbol{I} - z\mathfrak{A} + z\boldsymbol{b}\mathbf{1}^\top\right)}{\det\,(\boldsymbol{I} - z\mathfrak{A})}$$

Unconditional stability of single step methods A necessary condition for unconditional stability of a single step method, is that the discrete evolution operator Ψ_λ^t applied to the scalar ODE $\dot{u} = -\lambda u$ satisfies

$$\lambda > 0 \to \lim_{j\to\infty} (\Psi_\lambda^\tau)^j u_0 = 0, \quad \forall u_0, \forall \tau > 0$$

Definition 9.2.7.46: L-stability
An RK-SSM satisfying the above condition, is called L-stable if its stability function satisfies

$$|S(z)| < 1, \forall z < 0 \text{ and } S(-\infty) = \lim_{z\to-\infty} S(z) = 0$$

Plugging $-\infty$ int S we obtain $S(-\infty) = 1 - \boldsymbol{b}^\top \mathfrak{A}^{-1}\mathbf{1}$, which is equal to zero if $\boldsymbol{b}$ is equal to the last row of $\mathfrak{A}$.

Theorem 9.2.8.5: Meta-theorem — Convergence of fully discrete evolution
Assume that

- the solution of the parabolic IBVP is "sufficiently smooth"
- its spatial Galerkin finite element discretization relies on degree p Lagrangian finite elements on uniformly shape-regular families of meshes
- time stepping is based on an L-stable single step method of order q,

then we can expect an asymptotic behavior of the total discretization error according to

$$\left(\tau \sum_{j=1}^{M} \left| u(\tau j) - u_h^{(j)} \right|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_{\mathcal{M}}^p + \tau^q)$$

Hence the total error is the spatial error plus the temporal error.

9.3 Linear wave equations

In local form, the (linear) wave equation is given by

$$\rho(\boldsymbol{x})\frac{\partial^2}{\partial t^2}u - \operatorname{div}(\sigma(\boldsymbol{x})\ \mathbf{grad}\, u) = f \quad \text{in } \tilde{\Omega} \tag{45}$$

Note the similarity to the heat equation (39). Since the wave equation is a second order ODE $\ddot{\boldsymbol{u}} = \boldsymbol{f}(\boldsymbol{u})$, two initial conditions are needed. In addition to the initial conditions (40), the initial velocity

$$\frac{\partial}{\partial t}u(\boldsymbol{x}, 0) = v_0(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \Omega$$

is also needed.

We want to use the time stepping schemes we already know. To apply them, the wave function can be converted into a first order ODE:

$$\begin{aligned} \dot{u} &= v \\ \rho\dot{v} &= \operatorname{div}(\sigma(\boldsymbol{x})\ \mathbf{grad}\, u) \end{aligned}$$

Remember from Analysis that the particular wave equation $\frac{\partial^2}{\partial t^2}u - c^2\frac{\partial^2}{\partial x^2}u = 0$ in 1D results in the d'Alembert solution:

$$u(x,t) = \frac{1}{2}(u_0(x+ct) + u_0(x-ct)) + \frac{1}{2c}\int_{x-ct}^{x+ct} v_0(s)\,\mathrm{d}s$$

with u_0 and v_0 the initial conditions. Hence there is again the concept of domain of dependence and domain of influence. This will be important later. Furthermore, in the absence of a source term, as in the simple case above, the solution will stay undamped. This corresponds to *conservation of total energy*.

We can formulate the variational problem:

$$m(\ddot{u}, v) + a(u, v) = 0 \quad \forall v \in V_0 \tag{46}$$

Theorem 9.3.2.16: Energy conservation in wave propagation
If u solves (46), then its energy is conserved, in the sense that

$$\frac{1}{2}m\left(\frac{\partial}{\partial t}u, \frac{\partial}{\partial t}u\right) + \frac{1}{2}a(u,u) \equiv \text{const}$$

where $\frac{1}{2}m\left(\frac{\partial}{\partial t}u, \frac{\partial}{\partial t}u\right)$ can be understood as the 'kinetic' energy and $\frac{1}{2}a(u,u)$ as the 'potential' energy.

9.3.3 Method of Lines

The method of lines gives rise to

$$\boldsymbol{M}\left\{\frac{d^2}{dt^2}\vec{\mu}(t)\right\} + \boldsymbol{A}\vec{\mu}(t) = \vec{\varphi}(t)$$

$$\vec{\mu}(0) = \vec{\mu}_0, \frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu}(0) = \vec{\nu}_0$$

Using $\vec{\nu} = \dot{\vec{\mu}}$, we can rewrite it to be a first order ODE.

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu} = \vec{\nu}$$

$$\boldsymbol{M}\frac{\mathrm{d}}{\mathrm{d}t}\vec{\nu} = \vec{\varphi}(t) - \boldsymbol{A}\vec{\mu}$$

$$\vec{\mu}(0) = \vec{\mu}_0, \vec{\nu}(0) = \vec{\nu}_0$$

Remember that in the case of $\vec{\varphi} \equiv 0$, energy is conserved:

$$E_h(t) = \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu}^\top \boldsymbol{M}\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu} + \frac{1}{2}\vec{\mu}^\top\boldsymbol{A}\vec{\mu} \equiv \text{const}$$

So we would like a time stepping that preserves this. Such time stepping schemes are called *structure preserving*. One such timestepping scheme is the **Crank–Nicolson** one:

$$\boldsymbol{M}\frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\frac{1}{2}\boldsymbol{A}\left(\vec{\mu}^{(j-1)} + \vec{\mu}^{(j+1)}\right) + \frac{1}{2}\left(\vec{\varphi}\left(t_j - \frac{1}{2}\tau\right) + \vec{\varphi}\left(t_j + \frac{1}{2}\tau\right)\right)$$

Another one would be the **Störmer scheme**:

$$\boldsymbol{M}\frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\boldsymbol{A}\vec{\mu}^{(j)} + \vec{\varphi}\big(t_j\big)$$

For both of these *second order* time stepping schemes, we need $\vec{\mu}^{(-1)}$ to get $\vec{\mu}^{(1)}$. Now the question is, where do we get this from? It can be obtained with a special initial step, using a symmetric (first order) difference quotient:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mu}(0) = \vec{\nu}_0 \to \frac{\mu^{(1)} - \mu^{(-1)}}{2\tau} = \vec{\nu}_0$$

And finally there is the **Leapfrog** timestepping. Using the auxiliary variable $\vec{\nu}^{(j+1/2)} = \frac{\vec{\mu}^{(j+1)} - \vec{\mu}^{(j)}}{\tau}$ and inserting this into the Störmer scheme results in

$$\begin{aligned}\boldsymbol{M}\frac{\vec{\nu}^{(j+1)} - \vec{\nu}^{(j)}}{\tau} &= -\boldsymbol{A}\vec{\mu}^{(j)} + \vec{\varphi}\big(t_j\big)\\ \frac{\vec{\mu}^{(j+1)} - \vec{\mu}^{(j)}}{\tau} &= \vec{\nu}^{(j+1/2)}\end{aligned}$$

with the initial step $\vec{\nu}^{(-1/2)} + \vec{\nu}^{(1/2)} = 2\vec{\nu}_0$.

10 Convection-Diffusion Problems

10.1 Heat conduction in a Fluid

Consider a flowing fluid. Then there is the key quantity, the *flow field* $\boldsymbol{v} : \Omega \subset \mathbb{R}^d \to \mathbb{R}^d$, where d is the dimension we consider. The flow field can be understood as $\boldsymbol{v}(\boldsymbol{x}) =$ fluid velocity at point $\boldsymbol{x} \in \Omega$.

Given a flow field $\boldsymbol{v}$, we can consider the autonomous initial value problem

$$\dot{\boldsymbol{y}} = \boldsymbol{v}(\boldsymbol{y}), \quad \boldsymbol{y}_0 = \boldsymbol{x}_0$$

The solution $t \to \boldsymbol{y}(t)$ describes how a particle moves, carried by the fluid, also called *streamline*.

As the domain Ω is usually bounded, we cannot have fluid leaving the domain. This means the fluid velocity must be zero in the normal direction of the domain boundary. Hence

$$\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega$$

Fourier's law in a moving fluid

$$\boldsymbol{j}(\boldsymbol{x}) = -\kappa \, \mathbf{grad}\, u(\boldsymbol{x}) + \boldsymbol{v}(\boldsymbol{x}) \rho u(\boldsymbol{x})$$

with κ the heat conductivity and ρ the volumetric heat capacity. We already know the first part, called diffusive heat flux, from the heat equation. The second part is called convective heat flux. With this new flux, the standard PDE becomes

$$-\operatorname{div}(\kappa \, \mathbf{grad}\, u) + \operatorname{div}(\rho \boldsymbol{v}(\boldsymbol{x}) u) = f \quad \text{in } \Omega \tag{47}$$

Incompressible Fluids

A fluid is called incompressible if its associated flow map (evaluation operator) Φ^t is volume preserving. This is the case iff.

$$\operatorname{div} \boldsymbol{v} \equiv 0 \quad \text{in } \Omega$$

Hence the fluid is incompressible if its flow velocity is divergence-free.

In case of incompressibility, equation (47) can be simplified using the general product rule Lemma 1.5.2.1 and $\operatorname{div} \boldsymbol{v} = 0$

$$-\operatorname{div}(\kappa \, \mathbf{grad}\, u) + \boldsymbol{v}(\boldsymbol{x}) \cdot \mathbf{grad}\, \rho u = f \quad \text{in } \Omega$$

We can also look at the time-dependent heat flow, which is similar to the heat equation (39)

$$\frac{\partial}{\partial t}(\rho u) - \operatorname{div}(\kappa \, \mathbf{grad}\, u) + \operatorname{div}(\rho \, \boldsymbol{v}(\boldsymbol{x}, t) \, \boldsymbol{u}) = f \quad \text{in } \Omega \tag{48}$$

We will see later on how to solve this.

10.2 Stationary Convection-Diffusion Problems

Here we will focus on the convection diffusion equation (47) with constant κ, ρ, incompressible flow $\boldsymbol{v}$ and zero Dirichlet boundary conditions. Hence

$$-\kappa\Delta u + \rho\boldsymbol{v}(\boldsymbol{x}) \cdot \mathbf{grad}\, u = f \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega$$

Non-dimensionalizing the problem results in

$$-\epsilon\Delta u + \boldsymbol{v}(\boldsymbol{x}) \cdot \mathbf{grad}\, u = f \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega$$

with $\|\boldsymbol{v}\|_{L^\infty(\Omega)} = 1$. This results in the following variational form

$$\epsilon \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, w \,\mathrm{d}\boldsymbol{x} + \int_\Omega (\boldsymbol{v} \cdot \mathbf{grad}\, u) w \,\mathrm{d}\boldsymbol{x} = \int_\Omega f(\boldsymbol{x}) w \,\mathrm{d}x \tag{49}$$

with the left-hand side the bilinear form $a(u, w)$. However, a is not symmetric. This also means that it does not induce an energy norm. However, it is still positive definite (see lecture document).

10.2.1 Singular perturbation

A boundary value problem depending on a parameter ϵ is called singularly perturbed, if the limit problem for $\epsilon \to \epsilon_0$ is not compatible with the boundary conditions.

For $\epsilon = 0$, the above PDE is singular perturbed. It cannot satisfy Dirichlet boundary conditions on the *outflow* part of the boundary.

$$\Gamma_{\text{out}} = \boldsymbol{x} \in \partial\Omega : \boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) > 0$$
$$\Gamma_{\text{in}} = \boldsymbol{x} \in \partial\Omega : \boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0$$

10.2.2 Upwinding

When trying to solve Eq. (49) with the Galerkin approach, when ϵ is very close to 0, one can observe huge oscillations in the solution, which is not correct. It comes from the fact that the Galerkin matrix becomes close to singular. So our goal is to get a robust method that can solve Eq. (49) no matter the ϵ.

Consider again Eq. (49) but in $d = 1$ and with zero boundary conditions

$$\epsilon \int_0^1 \frac{\partial u}{\partial x}\frac{\partial w}{\partial x}\,\mathrm{d}x + \int_0^1 \frac{\partial u}{\partial x} w \,\mathrm{d}x = \int_0^1 f(x) w \,\mathrm{d}x$$

To calculate the Galerkin matrix for an equidistant mesh with M cells, we use the global composite trapezoidal rule for the convective term

$$\int_0^1 \psi(x)\,\mathrm{d}x \approx h \sum_{j=0}^{M} \psi(jh)$$

Hence the convective term of the bilinear form will be approximated by

$$\int_0^1 \frac{\partial u_h}{\partial x}\, w_h \,\mathrm{d}x \approx h \sum_{j=1}^{M-1} \frac{\partial u_h}{\partial x}(jh)\, w_h(jh)$$

But $\frac{\partial u_h}{\partial x}(jh)$ is not defined: $u_h \in \mathcal{S}^0_{1,0}$ is piecewise continuous, so its derivative has jumps at the nodes. However, convection transports the information in the direction of $\boldsymbol{v}$ (= 1 in our case). So we use the value of $\frac{\partial u_h}{\partial x}$ from "upwind", i.e., in direction $-\boldsymbol{v}$, which here is the value in the previous cell:

$$\frac{\partial u_h}{\partial x}(jh) = \lim_{\delta \to 0} \frac{\partial u_h}{\partial x}(jh - \delta \boldsymbol{v}) = \left.\frac{\partial u_h}{\partial x}\right|_{]x_{j-1}, x_j[}$$

And generalized in more dimensions

$$\boldsymbol{v}(\boldsymbol{p}) \cdot \mathbf{grad}\, u_h(\boldsymbol{p}) = \lim_{\delta \to 0} \boldsymbol{v}(\boldsymbol{p}) \cdot \mathbf{grad}\, u_h(\boldsymbol{p} - \delta \boldsymbol{v}(\boldsymbol{p}))$$

10.2.2.2 Streamline Diffusion

A totally different idea for fixing the problem of $\epsilon \to 0$ is to add some h-dependent diffusion. I.e., replace $\epsilon \leftarrow \epsilon + c(h)$ with $c(h) > 0$. However, there is smearing in the internal layers. But the solution is smooth along the direction of $\boldsymbol{v}$, so adding diffusion along the velocity should not do any harm.

The method of *Anisotropic diffusion* is born. On cell K, replace $\epsilon \leftarrow \epsilon \boldsymbol{I} + \delta_K \boldsymbol{v}_K \boldsymbol{v}_K^\top$. Here, $\boldsymbol{v}_K$ is the local velocity, obtained by averaging over the vertices, and $\delta_K > 0$ some controlling parameter.

$$\int_\Omega \epsilon \boldsymbol{I} + \delta_K \boldsymbol{v}_K \boldsymbol{v}_K^\top)\ \mathbf{grad}\, u \cdot \mathbf{grad}\, w \,\mathrm{d}\boldsymbol{x} + \int_\Omega (\boldsymbol{v} \cdot \mathbf{grad}\, u) w \,\mathrm{d}\boldsymbol{x} = \int_\Omega f(\boldsymbol{x}) w \,\mathrm{d}\boldsymbol{x}$$

However this affects the solution u, such that it will not be the same as the one from Eq. (49). To get rid of this inconsistency, the anisotropic diffusion can be introduced via a residual term

$$\begin{aligned} &\int_\Omega \epsilon\ \mathbf{grad}\, u \cdot \mathbf{grad}\, w \,\mathrm{d}\boldsymbol{x} + \int_\Omega (\boldsymbol{v} \cdot \mathbf{grad}\, u) w \,\mathrm{d}\boldsymbol{x} \\ &+ \sum_{K \in \mathcal{M}} \delta_K \int_K (-\epsilon \Delta u + \boldsymbol{v} \cdot \mathbf{grad}\, u - f)(\boldsymbol{v} \cdot \mathbf{grad}\, w) = \int_\Omega f(\boldsymbol{x}) w \,\mathrm{d}\boldsymbol{x} \end{aligned}$$

the added term will be zero for the exact solution (strong PDE) and the anisotropic diffusion is still here. The control parameter is usually chosen according to

$$\delta_K = \begin{cases} \epsilon^{-1} h_K^2 & \text{if } \|\boldsymbol{v}\|_{K,\infty} h_K \leq 2\epsilon \\ h_K & \text{if } \|\boldsymbol{v}\|_{K,\infty} h_K > 2\epsilon \end{cases}$$

With this, the $\mathcal{O}(h_{\mathcal{M}}^2)$ convergence of $\|u - u_h\|_{L^2(\Omega)}$ for h-refinement is preserved, while upwind quadrature only achieves $\mathcal{O}(h_{\mathcal{M}})$ convergence.

10.3 Discretization of Time-Dependent Convection-Diffusion IBVPs

Now we will take a look at how time-dependent (also called *transient*) convection-diffusion can be modeled. Assuming the incompressibility condition and non-dimensionalizing, (48) becomes

$$\frac{\partial u}{\partial t} - \epsilon \Delta u + \boldsymbol{v}(\boldsymbol{x}, t) \cdot \mathbf{grad}\, u = f \quad \text{in } \Omega \tag{50}$$

If we solve this with the method of lines (Section 9.3.3) without upwind quadrature, oscillations occur. But with upwinding, we get damping, which is also wrong. Therefore, we need a different method. Again, we need to take care of the limit $\epsilon \to 0$. So let's first look at the pure transport problem ($\epsilon = 0$):

$$\frac{\partial u}{\partial t} + \boldsymbol{v}(\boldsymbol{x}, t) \cdot \mathbf{grad}\, u = f \quad \text{in } \Omega$$

Its solution is given by the *Method of Characteristics*

$$u(\boldsymbol{x}, t) = \begin{cases} u_0(\boldsymbol{x_0}) + \int_0^t f(\boldsymbol{y}(s), s)\,\mathrm{d}s & \text{if } \boldsymbol{y}(s) \in \Omega \qquad \forall 0 < s < t \\ g(\boldsymbol{y}(s_0), s_0) + \int_{s_0}^t f(\boldsymbol{y}(s), s)\,\mathrm{d}s & \text{if } \boldsymbol{y}(s_0) \in \partial\Omega \wedge \boldsymbol{y}(s) \in \Omega \quad \forall s_0 < s < t \end{cases} \tag{51}$$

where $\boldsymbol{y}(t)$ is defined as the solution of $\dot{\boldsymbol{y}}(t) = \boldsymbol{v}(\boldsymbol{y}(t), t)$ and describes a streamline.

We have the initial condition u_0 on Ω and the Dirichlet boundary condition g on the inflow boundary (see (40) for definitions). This method works as follows: We follow a streamline backwards in time starting at $\boldsymbol{x}$.

Then, the solution consists of two parts: The inital value of u at $t = 0$ and the integral of the source function f from 0 to t along the streamline. This is described by the first case for a streamline that stays inside the domain. The second case is for a streamline that leaves the domain. Here, s_0 is the time at which the streamline intersects the boundary, and we use the boundary value g as the initial value at time s_0.

Splitting Methods

Given a general ODE whose right-hand side is the sum of two functions

$$\dot{\boldsymbol{y}} = \boldsymbol{g}(t, \boldsymbol{y}) + \boldsymbol{r}(t, \boldsymbol{y})$$

The Strang Splitting single step method provides a method to solve this.

Strang Splitting
Compute $\boldsymbol{y}^{(j+1)}$ given $\boldsymbol{y}^{(j)}$ according to

$$\begin{aligned} \tilde{\boldsymbol{y}} &= \boldsymbol{z}\left(t_j + \frac{1}{2}\tau\right), && \text{where } \boldsymbol{z}(t) \quad \text{solves } \dot{\boldsymbol{z}} = \boldsymbol{g}(t, \boldsymbol{z}), \boldsymbol{z}(t_j) = \boldsymbol{y}^{(j)} \\ \hat{\boldsymbol{y}} &= \boldsymbol{w}(t_{j+1}), && \text{where } \boldsymbol{w}(t) \quad \text{solves } \dot{\boldsymbol{z}} = \boldsymbol{r}(t, \boldsymbol{w}), \boldsymbol{w}(t_j) = \tilde{\boldsymbol{y}} \\ \boldsymbol{y}^{(j+1)} &= \boldsymbol{z}(t_{j+1}), && \text{where } \boldsymbol{z}(t) \quad \text{solves } \dot{\boldsymbol{z}} = \boldsymbol{g}(t, \boldsymbol{z}), \boldsymbol{z}\left(t_j + \frac{1}{2}\tau\right) = \hat{\boldsymbol{y}} \end{aligned}$$

and $t_{j+1} = t_j + \tau$

Theorem 10.3.3.5: Order of Strang splitting single step method
If the IVP in each sub-step is solved exactly or with a 2nd-order time stepping method, the Strang splitting method is of second order.

We can now apply this to Eq. (50).

$$\begin{array}{ccccc} \frac{\partial}{\partial t}u = & \epsilon\Delta u & & f - \boldsymbol{v}\cdot\mathbf{grad}\,u \\ \updownarrow & \updownarrow & & \updownarrow \\ \dot{\boldsymbol{y}} = & \boldsymbol{g}(\boldsymbol{y}) & & \boldsymbol{r}(\boldsymbol{y}) \end{array}$$

This amounts to once solving pure diffusion

$$\frac{\partial t}{\partial z} - \epsilon\Delta z = 0$$

and once pure transport

$$\frac{\partial t}{\partial w} + \boldsymbol{v}\cdot\mathbf{grad}\,u = f$$

To solve the pure transport problem, we have seen the method of characteristics (51). However, it requires integration along streamlines. We can approximate these integrals by "following" particles along their path defined by the velocity field. This **particle method** works as follows:

1. Pick suitable interpolation nodes $\boldsymbol{p}_i$, the initial particle positions.
2. Solve initial value problems
$$\dot{\boldsymbol{y}}(t) = \boldsymbol{y}(\boldsymbol{v}(t), t) \quad , \quad \boldsymbol{y}(0) = \boldsymbol{p}_i$$
with suitable single step methods.
3. Reconstruct the approximation. E.g., with the composite midpoint rule,
$$u_h^{(j)}\left(\boldsymbol{p}_i^{(j)}\right) = u_0(\boldsymbol{p}_i) + \tau\sum_{l=1}^{j-1} f\left(\frac{1}{2}(\boldsymbol{p}_i^l + \boldsymbol{p}_i^{l-1}), \frac{1}{2}(t_l + t_{l-1})\right)$$

Basically, we have an interpolation problem with nodes that change over time. At each step, we need to add particles at the inflow boundary and remove ones that leave the domain. This means we need to re-mesh in each step, i.e., create a new triangular mesh with the advected nodes/particles.

10.3.4 Semi-Lagrangian Method

Check out this video for an intuitive explanation of the method in 1D.

A method which relies on a *fixed* mesh is the Semi-Lagrangian method.

Definition 10.3.4.2: Material derivative
Given a velocity field $\boldsymbol{v}$, the material derivative of a function f is given by

$$\frac{Du}{D\boldsymbol{v}}(\boldsymbol{x}, t_0) = \lim_{\tau \to 0} \frac{u(\boldsymbol{x}, t_0) - u\big(\Phi^{t_0, t_0-\tau}\boldsymbol{x}, t_0 - \tau\big)}{\tau}$$

By the chain rule we find

$$\frac{Du}{D\boldsymbol{v}}(\boldsymbol{x}, t) = \mathbf{grad}\, u(\boldsymbol{x}, t) \cdot \boldsymbol{v}(\boldsymbol{x}, t) + \frac{\partial}{\partial t} u(\boldsymbol{x}, t)$$

Hence the transient convection-diffusion equation (50) can be rewritten as

$$\frac{Du}{D\boldsymbol{v}} - \epsilon \Delta u = f \quad \text{in } \Omega$$

We will now approximate the material derivative by a backwards difference. It is easy to interpret that expression: the total change of u in a timestep for a particle is the difference of the current $u^{(j)}$ at the particle's current position $\boldsymbol{x}$ minus the value of the previous function $u^{(j-1)}$ at the particle's old position $\Phi^{t_j, t_j-\tau}\boldsymbol{x}$. We get a semi-discretization

$$\frac{u^{(j)}(\boldsymbol{x}) - u^{(j-1)}\big(\Phi^{t_j, t_j-\tau}\boldsymbol{x}\big)}{\tau} - \epsilon \Delta u^{(j)} = f\big(\boldsymbol{x}, t_j\big) \quad \text{in } \Omega$$

with additional initial conditions for $t = t_j$. We can apply the standard Galerkin method to this semi-discretization.

$$\int_\Omega \frac{u^{(j)}(\boldsymbol{x}) - u^{(j-1)}\big(\Phi^{t_j, t_j-\tau}\boldsymbol{x}\big)}{\tau} w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} + \epsilon \int_\Omega \mathbf{grad}\, u^{(j)} \cdot \mathbf{grad}\, w\, \mathrm{d}\boldsymbol{x} = \int_\Omega f\big(\boldsymbol{x}, t_j\big) w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$$

Unfortunately this cannot be implemented as is because the function $u^{(j-1)}\big(\Phi^{t_j, t_j-\tau}\boldsymbol{x}\big)$ is not a finite element function on $\mathcal{M}$.

To get around this, we do two things: Replace $\Phi^{t_j, t_j-\tau}\boldsymbol{x}$ by $\boldsymbol{x} - \tau \boldsymbol{v}\big(\boldsymbol{x}, t_j\big)$ (an explicit Euler step) and replace $u^{(j-1)}\big(\Phi^{t_j, t_j-\tau}\boldsymbol{x}\big)$ by its linear interpolant $I_1\big(u^{(j-1)}\big(\Phi^{t_j, t_j-\tau}\boldsymbol{x}\big)\big)$.

$$\int_\Omega \frac{u^{(j)}(\boldsymbol{x}) - I_1\big\{u^{(j-1)}\big(\boldsymbol{s} - \tau \boldsymbol{v}\big(\boldsymbol{s}, t_j\big)\big)\big\}(\boldsymbol{x})}{\tau} w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} + \epsilon \int_\Omega \mathbf{grad}\, u^{(j)} \cdot \mathbf{grad}\, w\, \mathrm{d}\boldsymbol{x} = \int_\Omega f\big(\boldsymbol{x}, t_j\big) w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$$

Here, $\boldsymbol{s}$ is the variable on which the interpolation operator I_1 acts (in the lecture notes, this variable is written as a dot " $\cdot$ "). The above equation still looks quite complicated, but can be implemented.

11 Numerical Methods for Conservation Laws

11.2 Scalar Conservation Laws in 1D

The goal of this chapter is to solve general conservation laws which are of the form

$$\frac{\partial u}{\partial t}(x,t) + \frac{\partial}{\partial x}(f(u(x,t),x)) = s(u(x,t),x,t). \tag{52}$$

The flux $f : \mathbb{R} \times \Omega \to \mathbb{R}$ can be a general function, which can depend non-linearly on the solution u. Everything in this chapter will be one-dimensional in space and time, so we have $\Omega \subseteq \mathbb{R}$.

We usually consider the special case $s = 0$ and $f = f(u)$, known as the **Cauchy problem**:

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) &= 0 \\ u(x,0) &= u_0(x) \end{aligned} \tag{53}$$

Example: Particle Model

We model cars as particles with position $x_i(t)$. Traffic speed is modeled by

$$\dot{x}_i(t) = v_{\max} \cdot \left(1 - \frac{\Delta_0}{\Delta x_i(t)}\right), \quad \Delta x_i(t) = x_{i+1}(t) - x_i(t),$$

where $v_{\max}$ is the maximum velocity and Δ_0 is the minimal distance between cars (i.e., the car length). Using some basic assumptions, we get a PDE:

$$\frac{\partial u}{\partial t}(x,t) = \frac{\partial}{\partial x} u(1-u).$$

We define the car density $u(x)$ as the number of cars in an infinitesimal interval around x.

11.2.2 Characteristics

We consider the Cauchy problem (53). Then a characteristic curve is defined as

Definition 11.2.2.3: Characteristic curve for 1D scalar conservation law

$\Gamma : [0,T] \to \mathbb{R} \times [0,T]$ with $\Gamma(\tau) := (\gamma(\tau), \tau)$, such that γ satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t}\gamma(\tau) = f'(u(\gamma(\tau),\tau))$$

for $0 \le \tau \le T$.

Generally, characteristic curves are lines along which information propagates. This means $u(x,t)$ will only depend on the initial condition at x_0, namely $u_0(x_0)$, if there is a characteristic curve that starts in the point x_0 and travels to the spacetime point (x,t). One property of characteristic curves is the following:

Lemma 11.2.2.6: Classical solution and characteristic curves
Smooth solutions of (53) are constant along characteristic curves.

For example in the case of linear advection ($\partial_t u + v\partial_x u = 0$) we can use this to solve the equation because $\gamma(\tau) = (x_0 + \tau v)$, which implies $u(x,t) = u_0(x - tv)$.

But this does not work if the solution is not smooth. For example, in the traffic flow model above, the solution has a jump after a certain time and hence this approach breaks down.

11.2.4 Jump conditions and Riemann Problem

The method of characteristics usually only works up to a certain point in time. To get the solution for times after that, we first note that the solution will usually have a discontinuity after the time where the method of characteristics breaks down. So we study how the solution behaves at these jumps (discontinuities).

The setting is as follows: We still study the Cauchy problem (53). We can derive that along jumps, the normal components must be continuous, which leads to the

Definition 11.2.4.2: Rankine–Hugoniot (jump) condition

$$\dot{s}(u_l - u_r) = f(u_l) - f(u_r)$$

where $\dot{s} = \frac{d\gamma}{dt}$ is the time derivative of the discontinuity curve $\Gamma(t) = (\gamma(t), t) \in \mathbb{R} \times [0, T]$.

Note that this is useful because it allows us to compute the jump if we know u_l and u_r.

The Riemann problem is given as

Definition 11.2.5.1: Riemann problem

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0$$

and

$$u_0(x) = \begin{cases} u_l \in \mathbb{R} \text{ if } x < 0 \\ u_r \in \mathbb{R} \text{ if } x > 0 \end{cases}$$

Note that f can still be chosen to be any sufficiently smooth flux function.

Using the Rankine–Hugoniot jump condition we then get the following solution for Riemann problems with a shock:

Lemma 11.2.5.4: Shock solution for Riemann problem
For any two states $u_r, u_l \in \mathbb{R}$ the piecewise constant function

$$u(x,t) := \begin{cases} u_l & \text{for } x < \dot{s}t \\ u_r & \text{for } x > \dot{s}t \end{cases} \qquad \dot{s} := \frac{f(u_l) - f(u_r)}{u_l - u_r}, x \in \mathbb{R}, 0 < t < T$$

is a weak solution to the Riemann problem.

Note that the solution only holds if the equation implies a shock (jump). This is the case if f is convex and $u_l > u_r$ or if f is concave and $u_r > u_l$.

If the jump only exists in the beginning we have a different solution

Lemma 11.2.5.4: Rarefaction solution for Riemann problem
If $f \in C^2(\mathbb{R})$ is strictly $\begin{cases} \text{convex and } u_l < u_r \\ \text{concave and } u_r < u_l \end{cases}$, then

$$u(x,t) := \begin{cases} u_l & \text{for } x < \min\{f'(u_l), f'(u_r)\} \cdot t \\ g\left(\frac{x}{t}\right) & \text{for } \min\{f'(u_l), f'(u_r)\} < \frac{x}{t} < \max\{f'(u_l), f'(u_r)\} \\ u_r & \text{for } x > \max\{f'(u_l), f'(u_r)\} \cdot t \end{cases}$$

is a weak solution to the Riemann problem with $g := (f')^{-1}$.

The question when to choose which of the two solution is answered by

Definition 11.2.6.1: Lax entropy condition
Let u be a weak solution of (53), and a piecewise classical solution in neighborhood of C^2-curve $\Gamma := (\gamma(\tau), \tau), 0 \leq \tau \leq T$, discontinuous across Γ.

u satisfies the *Lax entropy condition* in $(x_0, t_0) \in \Gamma$ iff.

$$f'(u_l) > \underbrace{\frac{f(u_l) - f(u_r)}{u_l - u_r}}_{\dot{s}} > f'(u_r)$$

Now if u satisfies the Lax entropy condition, then we have to pick the shock solution. Otherwise we pick the rarefaction solution.

11.2.7 Properties of Entropy Solutions

The essential properties here are that with the propagation speed $f'(u)$, we find the **domain of dependence** and the **domain of influence**, which is best illustrated by a picture and hence we encourage the reader to look at the lecture document and the illustrations below Theorem 11.2.7.3.

Moreover the second result is that the number of extrema of the solution is non-increasing in time.

11.3 Conservative Finite Volume (FV) Discretization

11.3.1 Finite Difference Methods

Finite difference methods are probably the simplest methods for solving PDEs. We just replace the spacial derivatives by some finite difference quotient for example one of the following

$$\begin{aligned}
\text{Symmetric difference quotient} \quad & \frac{\partial f}{\partial x} \approx \frac{f(x_0+h)-f(x_0-h)}{2h} \\
\text{Backward difference quotient} \quad & \frac{\partial f}{\partial x} \approx \frac{f(x_0)-f(x_0-h)}{h} \\
\text{Forward difference quotient} \quad & \frac{\partial f}{\partial x} \approx \frac{f(x_0+h)-f(x_0)}{h}
\end{aligned}$$

Then we construct a solution by time stepping: given $u(x,t_k)$ we compute $u(x,t_{k+1})$ by using some Runge–Kutta integrator.

With the example of finite difference methods, we observe that by the nature of the problems we are studying in this chapter, the solutions have to be constructed under consideration of the flux direction. That is, in the spirit of characteristic curves we know that information propagates along curves in space time. And if this curve advances, for example, from left to right in space, then we need to use the forward difference quotient, because the backward difference quotient will not contain the information of the flow direction.

11.3.2 Spatially Semi-Discrete Conservation Form

The method we will use in this section of the course is the Finite Volume Method. We use a uniform 1D mesh with N cells and search a piecewise constant solution. The coefficients are the values of u at the cell centers:

$$\mu_i \approx u(x_i,t)$$

we build a mesh by taking intervals around the spatial points in which we approximate the solution u. Now we derive an ODE for $\boldsymbol{\mu}$ by starting from the problem definition and integrating over cell j:

$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x} f(u)$$

$$\underbrace{\frac{\partial}{\partial t}\int_{x_{j-1/2}}^{x_{j+1/2}} u(x,t)\,\mathrm{d}x}_{\approx h\cdot\mu_j} = -\Big(f\Big(u\Big(x_{j+1/2},t\Big)\Big) - f\Big(u\Big(x_{j-1/2},t\Big)\Big)\Big) \tag{54}$$

Because u is piecewise constant, $u\left(x_{j+\frac{1}{2}},t\right)$ is undefined.

An approximation for $f_{j+\frac{1}{2}} := f\left(u\left(x_{j+\frac{1}{2}},t\right)\right)$ is given by the **numerical flux function** F:

General Numerical Flux Function

$$f_{j+\frac{1}{2}} \approx F\left(\mu_{j-m_l+1},\ldots,\mu_{j+m_r}\right)$$

F is called a (m_l+m_r)–point numerical flux function.

A **2-point flux**, which is all we need in this section, is defined as $f_{j+\frac{1}{2}} \approx F(\mu_j, \mu_{j+1})$.

With this, (54) becomes (omitting the time argument of μ_j)

$$\frac{\mathrm{d}\mu_j}{\mathrm{d}t}(t) = -\frac{1}{h}\big(F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)\big) \tag{55}$$

We can simply plug the RHS of (55) into a Runge–Kutta method, so the only thing left to do is to find a suitable flux function F, which should satisfy the following condition:

Definition 11.3.3.5: Consistent numerical flux function
A numerical function $F : \mathbb{R}^{m_l+m_r} \to \mathbb{R}$ is *consistent* with the flux $f : \mathbb{R} \to \mathbb{R}$ if

$$F(u, u, ..., u) = f(u), \qquad \forall u \in \mathbb{R}$$

Theory tells us that a consistent numerical flux will produce a solution with the correct shock speed.

11.3.4 Numerical Flux Functions

This section now treats how to find suitable flux functions. There are several options: the simplest one just averages the inputs of $F(u, w)$, but suffers from the same problem as finite difference methods.

One remedy for this is the *Lax–Friedrichs / Rusanov* Flux which is useful but flattens the edges of jumps (which is due to its construction with additional diffusion).

Equation 11.3.4.16 (Lax–Friedrichs / Rusanov Flux)

$$F_{LF}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2}(w - v) \max_{\min\{v,w\} \le u \le \max\{v,w\}} f'(u)$$

The third term is the artificial diffusion term.

As pointed out before, the direction in which the information flows is crucial, so an important idea to choose the right flux is to respect that. Moreover the flux has to reproduce physical solution in the sense explained above when studying two possible solutions for the Riemann problem.

The final flux we look at, which solves these problems, is the Godunov Flux.

Definition 11.3.4.33: Godunov Flux

$$F_{GD}(v, w) = \begin{cases} \min_{v \le u \le w} f(u) & \text{if } v < w \\ \max_{w \le u \le v} f(u) & \text{if } v \ge w \end{cases}$$

11.3.5 Monotone Schemes

First, we define what it means for a flux to be monotone:

Definition 11.3.5.5: Monotonicity of flux functions
A 2-point numerical flux function $F = F(v, w)$ is *monotone* if
– F is increasing in its first argument, i.e. $F(v, w) \leq F(v + \Delta v, w) \quad \forall w \in \mathbb{R}$
– F is decreasing in its second argument, i.e. $F(v, w) \leq F(v, w + \Delta w) \quad \forall v \in \mathbb{R}$

In Section 11.2.7, we mentioned that the number of extrema of an analytical solution does not increase over time. Here, we show that the two main fluxes (LF and Godunov) we derived in the previous chapter are monotone and therefore both have this property. This is established by the following two lemmas:

Lemma 11.3.5.8: Monotonicity of Lax–Friedrichs and Godunov flux
For any continuously differentiable flux function f the associated Lax–Friedrichs flux and Godunov flux are monotone.

Lemma 11.3.5.13: Non-oscillatory monotone semi-discrete evolutions
If $\boldsymbol{\mu} = \boldsymbol{\mu}(t)$ solves the two-point flux equation (55) with a monotone numerical flux and $\boldsymbol{\mu}(0)$ has finitely many local extrema, then the number of local extrema of $\boldsymbol{\mu}(t)$ cannot be larger than that of $\boldsymbol{\mu}(0)$.

11.4 Timestepping for Finite-Volume Methods

As explained before, once we have chosen the numerical Flux, we just need to apply Runge–Kutta integration. This subsection studies some conditions that have to be considered when applying Runge–Kutta – in particular, constraints on choosing the timestep size τ.

Definition 11.4.2.5: Numerical domain of dependence
Consider the explicit fully discrete evolution $\boldsymbol{\mu}^{(k+1)} := \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mu}^{(k)}\big)$ on a uniform spatio-temporal mesh ($x_j = hj, j \in \mathbb{Z}, t_k = k\tau, k \in \mathbb{N}_0$) with

$$\exists m \in \mathbb{N}_0 : \big(\boldsymbol{\mathcal{H}}(\boldsymbol{\mu})\big)_j = \boldsymbol{\mathcal{H}}\big(\mu_{j-m}, ..., \mu_{j+m}\big), j \in \mathbb{Z}. \tag{56}$$

Then the **numerical domain of dependence** is given by

$$D_h^-\big(x_j, t_k\big) := \{(x_n, t_l) \in \mathbb{R} \times [0, t_k] : j - m(k - l) \leq n \leq j + m(k - l)\}$$

$\boldsymbol{\mathcal{H}}$ is an operator which gives $\boldsymbol{\mu}$ at the next timestep. It incorporates both the numerical flux and the Runge–Kutta method. If we have a first-order time integrator and the numerical flux function is $F = F\big(\mu_{j-m}, ..., \mu_{j+m}\big)$, then $\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{H}}\big(\mu_{j-m}, ..., \mu_{j+m}\big)$.

Example: Let's say we are solving (55) and call the RHS $R(\mu_j)$

$$\frac{\mathrm{d}\mu_j}{\mathrm{d}t}(t) = -\frac{1}{h}\big(F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)\big) = R(\mu_j)$$

and we apply explicit Euler:

$$\begin{aligned}\big(\mathcal{H}(\boldsymbol{\mu})\big)_j &= \mu_j + \tau R(\mu_j) \\ &= \mu_j - \frac{\tau}{h}\big(F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)\big)\end{aligned}$$

If we compare this to (56), we see that $m = 1$ in this case.

The following kind of condition appears over and over in numerical integration and gives an upper bound for the timestep size τ: If this upper bound is respected, the numerical solution is stable.

Definition 11.4.2.11: Courant–Friedrichs–Lewy (CFL) condition
An explicit local fully discrete evolution $\boldsymbol{\mu}^{(t+1)} := \mathcal{H}(\boldsymbol{\mu})$ on uniform spatio-temporal mesh $(x_j = hj, j \in \mathbb{Z}, t_k = k\tau, k \in \mathbb{N})$ satisfies the CFL condition, if the convex hull of its numerical domain of dependence contains the maximal analytical domain of dependence

$$D^-(x_j, t_k) \subset \operatorname{convex}\big(D_h^-(x_j, t_k)\big) \quad \forall j, k.$$

Applied to our problem, this means

Equation 11.4.2.12 (CFL condition for explicit fully discrete evolution)

$$\frac{\tau}{h} \leq \frac{m}{\max\{|\dot{s}_{\min}|, |\dot{s}_{\max}|\}}.$$

11.4.4 Convergence of Fully Discrete FV Method

The **consistency error** is defined as follows:

$$\epsilon := \max_j \Big\{ F\big(u(x_j, t), u(x_{j+1}, t)\big) - f\Big(u\Big(x_{j+\frac{1}{2}}, t\Big)\Big)\Big\},$$

if we assume u to be an exact solution of If $\epsilon = \mathcal{O}(h^q)$, the flux is called *q-th order consistent.* q is then the order of convergence of the FV method.

We get at most order one convergence in the maximum and the L^1 norm. This can be seen by the following fact:

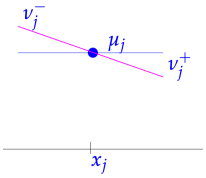
Order barrier for monotone numerical fluxes
Monotone numerical fluxes (Definition 11.3.5.5) are at most first order consistent.

11.5 Higher-Order FV Schemes

For the FV Method to achieve a convergence of order > 1, we need fluxes which are consistent of order > 1. Instead of coming up with completely new fluxes, we use the ones from Section 11.3.4 and only modify the arguments given to them.

We choose the approach to go from piecewise-constant to **discontinuous piecewise-linear** solutions. Instead of approximating $f\left(u\left(x_{j+\frac{1}{2}}\right)\right)$ by $F(\mu_j, \mu_{j+1})$, we use $F\left(\nu_j^+, \nu_{j+1}^-\right)$.

ν_j^+, ν_j^- are the values of the linear approximation of u at the boundaries of cell j (see image). The discrete evolution equation becomes



$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h}\left(F\left(\nu_j^+(t), \nu_{j+1}^-(t)\right) - F\left(\nu_{j-1}^+(t), \nu_j^-(t)\right)\right) \tag{57}$$

where ν_j^+, ν_j^- are the values of the linear approximation of u at the cell boundaries. The linear approximation on cell j is defined by the slope σ_j. So this method boils down to finding formulas for the slopes:

$$\sigma_j = \text{slope}(\mu_{j-1}, \mu_j, \mu_{j+1})$$

11.5.2 Slope limiting

Finding a linear approximation is formally expressed as a **reconstruction operator**:

$$\mathrm{R}_M\vec{\boldsymbol{\mu}} = \text{discontinuous pw. linear approximation of } u$$

Note that this operator as a mapping between vector spaces is not linear. However, we call it a **linear reconstruction** because the result is a (piecewise) linear function. We get the following formulas which we can plug into (57):

$$v_j^+ = \mathrm{R}_M\vec{\boldsymbol{\mu}}\left(x_{j+\frac{1}{2}}^-\right) \text{ and } v_j^- = \mathrm{R}_M\vec{\boldsymbol{\mu}}\left(x_{j-\frac{1}{2}}^+\right)$$

where x^+ denotes the limit from the right and x^- from the left. Note that we always assume $\mathrm{R}_M\vec{\boldsymbol{\mu}}(x_j) = \mu_j$, i.e., the linear reconstruction has the same values as the piecewise constant function at the cell centers.

Given slopes σ_j, we can now define the linear reconstruction operator:

$$\mathrm{R}_M\vec{\boldsymbol{\mu}}(x) = \mu_j + \sigma_j(x - x_j) \quad \text{for } x \in \left]x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}\right[$$

We introduce an important property of the reconstruction operator:

Definition 11.5.2.1: Monotonicity-preserving linear reconstruction
A linear reconstruction operator R_M is called *monotonicity-preserving* if

$$\mu_j \le \mu_{j+1} \implies \mathrm{R}_M\vec{\boldsymbol{\mu}} \text{ is non-decreasing} \in \left]x_j, x_{j+1}\right[$$
$$\mu_j \ge \mu_{j+1} \implies \mathrm{R}_M\vec{\boldsymbol{\mu}} \text{ is non-increasing} \in \left]x_j, x_{j+1}\right[$$

This ensures that no new extrema are introduced by the numerical scheme.

Finally, we present a reconstruction operator which is monotonicity-preserving:

Definition 11.5.2.9: Minmod reconstruction

$$\sigma_j(\mu_{j-1}, \mu_j, \mu_{j+1}) = \text{minmod}\left(\frac{\mu_{j+1}-\mu_j}{x_{j+1}-x_j}, \frac{\mu_j-\mu_{j-1}}{x_j-x_{j-1}}\right)$$

where minmod is defined as

$$\text{minmod}\ (a,b) = \begin{cases} a, & \text{if} \quad ab>0, |a|<|b| \\ b, & \text{if} \quad ab<0, |b|<|a| \\ 0, & \text{otherwise} \end{cases}$$

11.6 Systems of Conservation Laws

Definition 11.6.1.1: Linear system of conservation laws
Let $\boldsymbol{A}(x) : \mathbb{R} \to \mathbb{R}^m \times \mathbb{R}^m$ be a matrix-valued function and $D \subset \mathbb{R} \times \mathbb{R}^+$. Then

$$\frac{\partial \boldsymbol{u}}{\partial t}(x,t) + \frac{\partial}{\partial x}\Big(\boldsymbol{A}(x)\,\boldsymbol{u}(x,t)\Big) = 0 \quad \text{on } D \tag{58}$$

is a linear system of conservation laws for $\boldsymbol{u} : D \to \mathbb{R}^m$.

By diagonalizing $\boldsymbol{A}$, we can get a general solution formula. Let $\boldsymbol{r}_i$ be an eigenvector of $\boldsymbol{A}$ with eigenvalue λ_i: $\boldsymbol{A}\boldsymbol{r}_i = \lambda_i \boldsymbol{r}_i$. We collect the eigenvectors in a matrix $\boldsymbol{R}$. Note that both λ_i and $\boldsymbol{r}_i$ depend on x.

Solution of (58)

$$\boldsymbol{u}(x,t) = \sum_{i=1}^{m} \left(\boldsymbol{R}^{-1}\boldsymbol{u}_0\right)_i (x-\lambda_i t)\boldsymbol{r}_i$$

We have a superposition of states $\boldsymbol{r}_i$ propagating with speed λ_i scaled by $w_i := \left(\boldsymbol{R}^{-1}\boldsymbol{u}_0\right)_i$

We want to solve the Riemann problem

$$\boldsymbol{u}(x,0) = \begin{cases} \boldsymbol{u}_l \ \text{if } x<0 \\ \boldsymbol{u}_r \ \text{if } x>0 \end{cases} \tag{59}$$

for which we can find an explicit solution. Let's illustrate it for the case $m=4$:

$$\boldsymbol{u}(x,t) = \boldsymbol{u}_l \ \text{if } x < \lambda_1 t$$

$$\sum_{i=1}^{k} w_i^r \boldsymbol{r}_i + \sum_{i=k+1}^{m} w_i^l \boldsymbol{r}_i \ \text{if } \lambda_k t < x < \lambda_{k+1} t$$

For the jumps we get a formula similar to the Rankine–Hugoniot condition:

$$\boldsymbol{A}(x)(\boldsymbol{u}_k - \boldsymbol{u}_{k-1}) = \lambda_k(\boldsymbol{u}_k - \boldsymbol{u}_{k-1}),$$

where λ_k takes the role of $\dot{s}$ in the scalar case.

Definition 11.6.2.2: Non-linear system of conservation laws
Given $U \subset \mathbb{R}^m$, $D \subset \mathbb{R} \times \mathbb{R}^+$ and a continuously differentiable flux function $\boldsymbol{F} : U \to \mathbb{R}^m$,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{u})}{\partial x}(x,t) = 0 \quad \text{on } D \tag{60}$$

is a system of conservation laws for $\boldsymbol{u} : D \to U$.

This system is called *constant-coefficient* because $\boldsymbol{F}$ does not depend on time and *translation-invariant* because it does not take x as an argument.

We apply diagonalization again, this time to the Jacobian $D\boldsymbol{F}(\boldsymbol{u})$. Note that here, eigenvectors and eigenvalues do not depend on x anymore, but instead on $\boldsymbol{u}$:

$$D\boldsymbol{F}(\boldsymbol{u})\, \boldsymbol{r}_i(\boldsymbol{u}) = \lambda_i(\boldsymbol{u})\, \boldsymbol{r}_i$$

Here, we also get a Rankine–Hugoniot condition:

Theorem 11.6.2.19: Rankine--Hugoniot condition for systems
If a curve $\Gamma = (\gamma(t), t)$ separates two domains

$$\Omega_l = \{(x,t) \in D : x < \gamma(t)\} \quad \text{and} \quad \Omega_r = \{(x,t) \in D : x > \gamma(t)\}$$

and $\boldsymbol{u}_l, \boldsymbol{u}_r = \boldsymbol{u}|_{\Omega_l}, \boldsymbol{u}|_{\Omega_r} \in C^1$ are strong solutions of (60), then $\boldsymbol{u}$ is a weak solution of (60) if and only if

$$\frac{d\gamma}{dt}(t) \cdot (\boldsymbol{u}_l(\gamma(t),t) - \boldsymbol{u}_r(\gamma(t),t)) = \boldsymbol{F}(\boldsymbol{u}_l(\gamma(t),t)) - \boldsymbol{F}(\boldsymbol{u}_r(\gamma(t),t))$$

For the Riemann problem (59), this simplifies to

$$\dot{s}(\boldsymbol{u}_l - \boldsymbol{u}_r) = \boldsymbol{F}(\boldsymbol{u}_l) - \boldsymbol{F}(\boldsymbol{u}_r), \quad \dot{s} = \frac{d\gamma}{dt}$$

There is also an entropy condition for systems:

Definition 11.6.2.29: Lax entropy condition for systems
1. $\exists k \in \{1, \dots, m\} : \lambda_k(\boldsymbol{u}_l) > \dot{s} > \lambda_k(\boldsymbol{u}_r)$
2. $\forall j > k : \quad \lambda_j(\boldsymbol{u}_l), \lambda_j(\boldsymbol{u}_r) < \dot{s}$
3. $\forall j < k : \quad \lambda_j(\boldsymbol{u}_l), \lambda_j(\boldsymbol{u}_r) > \dot{s}$