Numerical Methods for PDE

ETH Zurich

Janik Schuettler

FS18

Contents

Part I

Summary

1 Overview

1.1 Spaces

$$
\begin{aligned}
L^2(\Omega) &= \left\{ v : \Omega \to \mathbb{R} \text{ integrable} : \int_\Omega |v(x)|^2 \,\mathrm{d}x < \infty \right\} \\
H^1(\Omega) &= \left\{ v : \Omega \to \mathbb{R} \text{ integrable} : \int_\Omega |\,\mathbf{grad}\, v(x)|^2 \,\mathrm{d}x < \infty \right\} \\
H^1_0(\Omega) &= \left\{ v \in H^1(\Omega) : v|_{\partial\Omega} = 0 \right\} \\
H^1_*(\Omega) &= \left\{ v \in H^1(\Omega) : \int_\Omega v(x) \,\mathrm{d}x = 0 \right\} \\
\mathcal{S}^0_1(\mathcal{M}) &= \big\{ v \in C^0(\overline{\Omega}) : \forall K \in \mathcal{M} : v|_K(\mathbf{x}) = \alpha_k + \boldsymbol{\beta}_K \mathbf{x},\ \alpha_K \in \mathbb{R}, \\
&\qquad \boldsymbol{\beta}_K \in \mathbb{R}^2, \mathbf{x} \in K \big\} \\
\mathcal{S}^0_{1,0}(\mathcal{M}) &= \mathcal{S}^0_1(\mathcal{M}) \cap H^1_0(\Omega) \\
\mathcal{S}^0_p(\mathcal{M}) &= \big\{ v \in C^0(\overline{\Omega}) : v|_K \in \mathcal{P}_p(K),\ \forall K \in \mathcal{M} \big\} \\
\mathcal{S}^0_p(\mathcal{M}) &= \big\{ v \in C^0(\overline{\Omega}) : v|_K \in \mathcal{Q}_p(K),\ \forall K \in \mathcal{M} \big\} \\
\mathcal{S}^0_{p,0}(\mathcal{M}) &= \mathcal{S}^0_p(\mathcal{M}) \cap H^1_0(\Omega)
\end{aligned}
$$

1.2 Norms

$$
\begin{aligned}
\|f\|_a &= \sqrt{a(f,f)} \\
\|f\|_{L^2(\Omega)} &= \|f\|_0 = \left(\int_\Omega |f(x)|^2 \,\mathrm{d}x \right)^{1/2} \\
|f|_{H^1(\Omega)} &= |f|_1 = \left(\int_\Omega \|\mathbf{grad}\, f\|^2 \right)^{1/2} = \|\mathbf{grad}\, f\|_{L^2(\Omega)} \\
\|f\|^2_{H^1(\Omega)} &= \|f\|^2_{L^2(\Omega)} + |f|^2_{H^1(\Omega)}
\end{aligned}
$$

On $H^1_*(\Omega)$, $|\cdot|$ is a norm equivalent to $\|\cdot\|_1$.

1.3 Inequalities

$$
\begin{aligned}
&|\|u\| - \|v\|| \le \|u \pm v\| \le \|u\| + \|v\| && \text{(Triangle)} \\
&\|u \pm v\|_a \le \|u\|_a + \|v\|_a
\end{aligned}
$$

$$
\begin{aligned}
&\langle u, v\rangle \le 2\langle u, v\rangle \le \|v\|^2 + \|u\|^2 && \text{(Binomial)} \\
&a(u,v) \le \|u\|_a^2 + \|v\|_a^2
\end{aligned}
$$

$$
\begin{aligned}
&\left| \int_\Omega uv \,\mathrm{d}x \right|^2 \le \int_\Omega |u|^2 \,\mathrm{d}x \int_\Omega |v|^2 \,\mathrm{d}x && \text{(Cauchy-Schwartz)} \\
&|\langle u, v\rangle| \le \|u\| \|v\| \\
&|a(u,v)| \le \|u\|_a \|v\|_a = \sqrt{a(u,u)a(v,v)}
\end{aligned}
$$

$$
\|u_0\|_{L^2(\Omega)} \le \operatorname{diam}(\Omega) \|\mathbf{grad}\, u_0\|_{L^2(\Omega)} \qquad \text{(Poincaré-Friedrich 1)}
$$

$$
\begin{aligned}
&\|u_*\|_{L^2(\Omega)} \le C \operatorname{diam}(\Omega) \|\mathbf{grad}\, u_*\|_{L^2(\Omega)} && \text{(Poincaré-Friedrich 2)} \\
&\|u\|_{L^2(\Omega)} \le C \|\mathbf{grad}\, u\|_{L^2(\Omega)} + D \|u\|_{L^2(\partial\Omega)}
\end{aligned}
$$

$$
\|u\|^2_{L^2(\partial\Omega)} \le C \|u\|_{L^2(\Omega)} \|u\|_{H^1(\Omega)} \qquad \text{(Multiplicative trace)}
$$

$$
\begin{aligned}
|a(u,v)| &\le \sqrt{a(u,u)a(v,v)} \\
|l(u)| &\le \|f\|_0 \|u\|_0 \le C \|u\|_{H^1(\Omega)} \\
\|a+b\| &\le \|a-b\| + 2\|a\| \\
\|a+b\| &\le \|a-b\| + 2\|b\|
\end{aligned}
$$

where $u \in H^1(\Omega)$, $u_0 \in H^1_0(\Omega)$, $u_* \in H^1(\Omega)_*$.

J **bounded from below** for pos. def. bilinear form a on V_0

$$
\exists C > 0 : |l(u)| \le C \|u\|_a \quad \forall u \in V_0
$$

Continuity for linear form l on V_0

$$
\exists C > 0 : |l(v)| \le C \|v\| \quad \forall v \in V_0,
$$

for bilinear form a on V_0

$$
\exists K > 0 : |a(u,v)| \le K \|u\| \|v\| \quad \forall u, v \in V_0,
$$

1.4 Formulas

Multidimensional Taylor expansion

$$
\begin{aligned}
\mathbf{f}(\mathbf{x}+\mathbf{h}) =& \mathbf{f}(\mathbf{x}) + \mathrm{D}f(\mathbf{x})\mathbf{h} + \mathcal{O}(h^2) \quad \text{for } h \to 0 \\
F(x+\delta x, \mathbf{y}+\delta\mathbf{y}) =& F(x,\mathbf{y}) + \partial_x F(x,\mathbf{y})\delta x + \partial_{\mathbf{y}} F(x,\mathbf{y})\delta\mathbf{y} \\
& \frac{1}{2}\partial_x^2 F(x,\mathbf{y})\delta x^2 + \partial_x\partial_{\mathbf{y}} F(x,\mathbf{y})\delta x\delta\mathbf{y} + \\
& \frac{1}{2}\delta\mathbf{y}^T \partial^2_{\mathbf{y}} F(x,\mathbf{y})\delta\mathbf{y} + \mathcal{O}(|\delta x|^3 + \|\delta\mathbf{y}\|^3)
\end{aligned}
$$

Theorem 1.1 (Integration of barycentric coordinate func.) *For any non-degenerate d-simplex K with barycentric coordinate functions $\lambda_1, \ldots, \lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$ for $j = 1, \ldots, d+1$,*

$$
\int_K \lambda_1^{\alpha_1} \ldots \lambda_{d+1}^{\alpha_{d+1}} \,\mathrm{d}x = d!|K| \frac{\alpha_1!\alpha_2! \ldots \alpha_{d+1}!}{(\alpha_1 + \alpha_2 + \ldots + \alpha_{d+1} + d)!},\ \forall \alpha \in \mathbb{N}_0^{d+1}.
$$

Theorem 1.2 (Gauss') *With $\boldsymbol{n} : \partial\Omega \to \mathbb{R}^d$ denoting the exterior unit normal vectorfield on $\partial\Omega$ and $\mathrm{d}S$ indicating integration over a surface, we have*

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

Theorem 1.3 (Green's first formula) *For all vector fields $\boldsymbol{j} \in (C^1_{pw}(\overline{\Omega}))^d$ and functions $v \in C^1_{pw}(\overline{\Omega})$ holds*

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

Conversation of energy for all control volumes V

$$
\int_{\partial V} \mathbf{j} \cdot \mathbf{n} \,\mathrm{d}S = \int_V f \,\mathrm{d}\mathbf{x},
$$

power flux through surface V = heat production inside V,

where $f \in C^0_{pw}(\Omega)$ is a heat source or sink.

Fourier's law

$$
\begin{aligned}
&\mathbf{j}(\mathbf{x}) = -\kappa(\mathbf{x}) \,\mathbf{grad}\, u(\mathbf{x}), && \mathbf{x} \in \Omega, && \text{(Steady)} \\
&\mathbf{j}(\mathbf{x}) = -\kappa(\mathbf{x}) \,\mathbf{grad}\, u(\mathbf{x}) + \mathbf{v}(\mathbf{x})\rho u(\mathbf{x}), && \mathbf{x} \in \Omega, && \text{(Moving)}
\end{aligned}
$$

where $\mathbf{j}$ denotes heat flux, u temperature, and κ heat conductivity. The first term corresponds to diffusive heat flux and the second term to convective heat flux.

2 BVP $\leftrightarrow$ LVP

2nd-order elliptic Dirichlet problem

$$
\begin{aligned}
-\operatorname{div}(\alpha(\mathbf{x}) \,\mathbf{grad}\, u(\mathbf{x})) &= f(\mathbf{x}) && \text{in } \Omega, \\
u(\mathbf{x}) &= g(\mathbf{x}) && \text{on } \partial\Omega
\end{aligned}
$$

Seek $u \in H^1(\Omega)$ with $u|_{\partial\Omega} = g$ such that

$$
\int_\Omega \alpha(\mathbf{x}) \,\mathbf{grad}\, u(\mathbf{x}) \,\mathbf{grad}\, v(\mathbf{x}) \,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) \,\mathrm{d}\mathbf{x}
$$

for all $v \in H_0^1(\Omega)$.

2nd-order elliptic Neumann problem

$$\begin{aligned} -\operatorname{div}(\alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})) &= f(\mathbf{x}) && \text{in } \Omega, \\ \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\cdot\mathbf{n} &= -h(\mathbf{x}) && \text{on } \partial\Omega \end{aligned}$$

Seek $u \in H_*^1(\Omega)$ such that

$$\int_\Omega \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\,\mathbf{grad}\,v(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega} hv\,\mathrm{d}S$$

for all $v \in H_*^1(\Omega)$.

2nd-order elliptic mixed Neumann problem

$$\begin{aligned} -\operatorname{div}(\alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})) &= f(\mathbf{x}) && \text{in } \Omega, \\ u(\mathbf{x}) &= g(\mathbf{x}) && \text{on } \Gamma_0 \subset \partial\Omega, \\ \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\cdot\mathbf{n} &= -h(\mathbf{x}) && \text{on } \partial\Omega\backslash\Gamma_0 \end{aligned}$$

Seek $u \in H^1(\Omega)$ with $u = g$ on Γ_0 such that

$$\int_\Omega \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\,\mathbf{grad}\,v(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega\backslash\Gamma_0} hv\,\mathrm{d}S$$

for all $v \in H^1(\Omega)$ with $v|_{v_0} = 0$.

2nd-order elliptic Radiation problem

$$\begin{aligned} -\operatorname{div}(\alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})) &= f(\mathbf{x}) && \text{in } \Omega, \\ \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\cdot\mathbf{n} &= \Psi(u) && \text{on } \partial\Omega \end{aligned}$$

Seek $u \in H^1(\Omega)$ such that

$$\int_\Omega \alpha(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x})\,\mathbf{grad}\,v(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega} \Psi(u)\,\mathrm{d}S = \int_\Omega f(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

for all $v \in H^1(\Omega)$.

3 Basis Functions and Local Shape Functions

3.1 1D Tent Functions

$$b_N^j(x) = \begin{cases} \frac{x-x_{j-1}}{x_j-x_{j-1}} & x \in [x_{j-1}, x_j] \\ \frac{-x+x_{j+1}}{x_{j+1}-x_j} & x \in [x_j, x_{j+1}] \\ 0 & \text{otherwise} \end{cases}$$

$$\partial_x b_N^j(x) = \begin{cases} \frac{1}{|x_j-x_{j-1}|} & x \in (x_{j-1}, x_j) \\ -\frac{1}{|x_{j+1}-x_j|} & x \in (x_j, x_{j+1}) \\ 0 & \text{otherwise} \end{cases}$$

$$\int_0^1 b_N^j(x) b_N^i(x)\,\mathrm{d}x = \{TODO$$

$$\int_0^1 \partial_x b_N^j(x) \partial_x b_N^i(x)\,\mathrm{d}x = \begin{cases} 0 & |i-j| > 2 \\ -1/h_{j+1} & j = i+1 \\ -1/h_i & j = i-1 \\ 1/h_i + 1/h_{i+1} & 1 \le i = j \le M-1 \end{cases}$$

3.2 2D Tent Functions

TODO

3.3 2D simplicial Linear Local Shape Functions

Unit triangle with vertices $\mathbf{a}^1$, $\mathbf{a}^2$, $\mathbf{a}^3$.

Area formula $|K| = \frac{1}{2}|e_1||e_2|\sin\omega_3$.

$$\begin{aligned} \lambda_1(\mathbf{x}) &= \frac{1}{2|K|}(\mathbf{x}-\mathbf{a}^2)\begin{pmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{pmatrix} = -\frac{|e_1|}{2|K|}(\mathbf{x}-\mathbf{a}^2)\cdot\mathbf{n}^1 \\ \lambda_2(\mathbf{x}) &= \frac{1}{2|K|}(\mathbf{x}-\mathbf{a}^3)\begin{pmatrix} a_2^3 - a_2^1 \\ a_1^1 - a_1^3 \end{pmatrix} = -\frac{|e_2|}{2|K|}(\mathbf{x}-\mathbf{a}^3)\cdot\mathbf{n}^2, \\ \lambda_3(\mathbf{x}) &= \frac{1}{2|K|}(\mathbf{x}-\mathbf{a}^3)\begin{pmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{pmatrix} = -\frac{|e_3|}{2|K|}(\mathbf{x}-\mathbf{a}^1)\cdot\mathbf{n}^3 \end{aligned}$$

Element stiffness matrix $A_{ij} = \int_K \mathbf{grad}\,\lambda_i\,\mathbf{grad}\,\lambda_j\,\mathrm{d}\mathbf{x}$ is given by

$$A = \frac{1}{2}\begin{pmatrix} \cot\omega_3 + \cot\omega_2 & -\cot\omega_3 & -\cot\omega_2 \\ -\cot\omega_3 & \cot\omega_3 + \cot\omega_1 & -\cot\omega_1 \\ -\cot\omega_2 & -\cot\omega_1 & \cot\omega_2 + \cot\omega_1 \end{pmatrix},$$

where ω_i denotes angle of the i-th corner.

The 2D triangular mass matrix $M_{ij} = \int_K \lambda_i\lambda_j\,\mathrm{d}\mathbf{x}$ is given by

$$M = \frac{|K|}{12}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

3.4 2D simplicial Linear Local Shape Functions on Unit Triangle

Unit triangle with vertices $\mathbf{a}^1 = (0,0)^T$, $\mathbf{a}^2 = (1,0)^T$, $\mathbf{a}^3 = (1,1)^T$.

$$\begin{aligned} \lambda_1(\mathbf{x}) &= 1 - x_1, \\ \lambda_2(\mathbf{x}) &= x_1 - x_2, \\ \lambda_3(\mathbf{x}) &= x_2 \end{aligned}$$

The element stiffness matrix A and mass matrix M for the triangle with vertices $\mathbf{a}^1$, $\mathbf{a}^2$, $\mathbf{a}^3$ are (this form generally holds for all right triangles with catheti of same length)

$$A = \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}, \quad M = \frac{1}{24}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

3.5 2D simplicial Quadratic Local Shape Functions

Let λ_i' denote the 2D linear local shape functions.

$$\begin{aligned} \lambda_1 &= (2\lambda_1' - 1)\lambda_1' & \lambda_4 &= 4\lambda_1'\lambda_2' & &\text{for } \mathbf{p}_4 = \frac{\mathbf{a}^1 + \mathbf{a}^2}{2} \\ \lambda_2 &= (2\lambda_2' - 1)\lambda_2' & \lambda_5 &= 4\lambda_2'\lambda_3' & &\text{for } \mathbf{p}_5 = \frac{\mathbf{a}^2 + \mathbf{a}^3}{2} \\ \lambda_3 &= (2\lambda_3' - 1)\lambda_3' & \lambda_6 &= 4\lambda_1'\lambda_3' & &\text{for } \mathbf{p}_6 = \frac{\mathbf{a}^1 + \mathbf{a}^2}{2} \end{aligned}$$

3.6 2D Tensor Product Linear Local Shape Functions

$$\begin{aligned} b_K^1(\mathbf{x}) &= (1-x_1)(1-x_2), & b_K^3(\mathbf{x}) &= x_1x_2 \\ b_K^2(\mathbf{x}) &= x_1(1-x_2), & b_K^4(\mathbf{x}) &= (1-x_1)x_2 \end{aligned}$$

4 Quadrature

Trapezoidal rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}(f(a)+f(b))$$

Composite trapezoidal rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{1}{2}\sum_{l=1}^{M} h_l(f(x_{l-1}) + f(x_l))$$

2D trapezoidal rule for triangle K with vertices $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3$

$$\int_K f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{|K|}{3}\left(f(\mathbf{a}^1) + f(\mathbf{a}^2) + f(\mathbf{a}^3)\right)$$

Composite midpoint rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{l=1}^{M} h_l f\left(\frac{x_l + x_{l-1}}{2}\right)$$

Part II

Theory

5 Second-order Scalar Elliptic Boundary Value Problems

5.1 Equilibrium Models

Configuration space, domains in this course, derivation of potential energy for taut membrane using spring model, continuity requirements for membrane functional.

Taut membrane model: Potential energy functional under vertical loading f is given by

$$J_M(u) = \int_\Omega \frac{1}{2}\sigma(\mathbf{x})\|\mathbf{grad}\, u(\mathbf{x})\|^2 - f(\mathbf{x})u(\mathbf{x})\,\mathrm{d}\mathbf{x}.$$

Typical arrangement, uniformly positive definite matrix, boundary conditions and configuration space.

Electromagnetic field energy

$$J_E(u) = \frac{1}{2}\int_\Omega (\varepsilon(\mathbf{x})\,\mathbf{grad}\, u(\mathbf{x}))\cdot \mathbf{grad}\, u(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

where $\varepsilon : \Omega \to \mathbb{R}^{3,3}$ is the dielectric tensor, symmetric and uniformly positive definite.

Definition 5.1 (Quadratic functional) *on a real vector space V_0 is a mapping $J : V_0 \to \mathbb{R}$ of the form*

$$J(u) = \frac{1}{2}a(u,u) - l(u) + c, \quad u \in V_0,$$

where $a : V_0 \times V_0 \to \mathbb{R}$ is a symmetric bilinear form, $l : V_0 \to \mathbb{R}$ a linear form l, and $c \in \mathbb{R}$.

Definition 5.2 (Quadratic minimization problem) *A minimization problem*

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

*is called a **quadratic minimization problem** if J is a quadratic functional on a real vector space V_0.*

Affine configuration space V can be written as $V = u_0 + V_0$. Then, $u_* = \operatorname{argmin}_{u\in V} J(u) = u_0 + \operatorname{argmin}_{u\in V_0} J(u_0 + u)$

Definition 5.3 (Positive definiteness) *A (symmetric) bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ on a real vector space V_0 is **positive definite** if $u \in V_0\backslash\{0\}$ if and only if $a(u,u) > 0$.*

Theorem 5.4 (Uniqueness of minimizer) *If the bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ is positive definite, then any solution of*

$$u_* = \operatorname*{argmin}_{u\in V_0} J(u), \quad J(u) = \frac{1}{2}a(u,u) - l(u) + c$$

is unique for any linear form $l : V_0 \to \mathbb{R}$.

Definition 5.5 (Energy norm) *A symmetric positive definite bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ induces the **energy norm***

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

Lemma 5.6 (Boundedness/ continuity of linear form) *The quadratic functional J based on a symmetric positive definite bilinear form a is bounded from below on V_0 if and only if*

$$\exists C > 0 : |l(u)| \le C\|u\|_a, \quad \forall u \in V_0,$$

*where $\|\cdot\|_a$ is the energy norm induced by a. Such linear forms are also called **continuous linear forms**.*

Proof, example of non-existence.

5.2 Sobolev spaces

Motivation We would like to find the largest space on which J is still well defined, i.e. $\{\text{functions } v : \Omega \to \mathbb{R} : a(v,v) < \infty\}$.

Definition 5.7 (L^2-Space) *of square-integrable functions on Ω is given by*

$$V_0 := \{v : \Omega \to \mathbb{R}, \textit{integrable} : \int_\Omega |v(\boldsymbol{x})|^2\,\mathrm{d}\boldsymbol{x} < \infty\}$$

and denoted $L^2(\Omega)$. It is a normed space with norm

$$\|v\|_0 := \|v\|_{L^2(\Omega)} := \left(\int_\Omega |v(\boldsymbol{x})|^2\,\mathrm{d}\boldsymbol{x}\right)^{1/2}.$$

Problem with L^2-spaces An arbitrarily small change in energy of u is sufficient to impose any boundary values, i.e. $\|u - \tilde{u}_n\|_0 \to 0$ for $n \to \infty$, where $tildeu_n$ has arbitrary boundary values.

Definition 5.8 (Sobolev space $H^1(\Omega)$) *of integrable functions on Ω with square integrable gradient (that vanishes on the boundary $\partial\Omega$) is given by*

$$H^1(\Omega) := \left\{v : \Omega \to \mathbb{R}, \textit{integrable} : \int_\Omega |\,\boldsymbol{grad}\, v(\boldsymbol{x})|^2\,\mathrm{d}\boldsymbol{x} < \infty\right\}$$
$$H^1_0(\Omega) := \left\{v \in H^1(\Omega) : v = 0 \textit{ on } \partial\Omega\right\}$$

with norm

$$|v|_1 := |v|_{H^1(\Omega)} := \left(\int_\Omega \|\boldsymbol{grad}\, v\|^2\,\mathrm{d}\boldsymbol{x}\right)^{1/2},$$
$$\|v\|^2_{H^1(\Omega)} := \|v\|^2_0 + |v|^2_{H^1(\Omega)}.$$

Theorem 5.9 (First Poincaré-Fridrichs inequality) *If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$ is bounded, then*

$$\|u\|_0 \le \operatorname{diam}(\Omega)\|\boldsymbol{grad}\, u\|_0 \quad \forall u \in H^1_0(\Omega)$$

Corollary 5.10 (Condition for unique minimizer on $H^1_0(\Omega)$) *If $f \in L^2(\Omega)$ then $u \to \int_\Omega f(\boldsymbol{x})u(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ is a continuous linear functional on $H^1_0(\Omega)$.*

Proof, how to work with Sobolev spaces.

Theorem 5.11 (Continuity conditions for $H^1(\Omega)$) *Let Ω be partitioned into sub-domains Ω_1 and Ω_2. A function u that is continuously differentiable in both sub-domains and continuous up to their boundary belongs to $H^1(\Omega)$ if and only if it is continuous on Ω, i.e. for $i = 1, 2$*

$$u \in C^1(\Omega_i), C^0(\overline{\Omega_i}) \implies (u \in H^1(\Omega) \iff u \in C^0(\Omega)).$$

Mental replacement for H^1 $C^1_{pw}(\overline{\Omega}) \subset H^1(\Omega)$

5.3 Variational Formulations

Configurational derivative

$$\mathrm{D}J(u,v) := \lim_{t\to 0}\frac{J(u+tv) - J(u)}{t}, \quad \forall v \in V_0$$

Meta-Theorem 5.1 (Linearity of configurational derivatives) *For "sufficiently smooth" $J : V \to \mathbb{R}$ the configurational derivative $\mathrm{D}J : V \times V_0 \to \mathbb{R}$ is linear in the second argument*

$$\mathrm{D}J(u; \alpha u + \beta w) = \alpha\,\mathrm{D}J(u;v) + \beta\,\mathrm{D}J(u;w) \quad \forall v, w \in V_0,\ \forall \alpha, \beta \in \mathbb{R}.$$

Minimizer satisfies variational equation If $J : V \to \mathbb{R}$ is "sufficiently smooth", the variational equation

$$\mathrm{D}J(u_*, v) = 0 \quad \forall v \in V_0,$$

provides a necessary condition fulfilled by every global minimizer of J.

Theorem 5.12 (QMP $\iff$ LVP) *For a quadratic functional $J(v) = \frac{1}{2}a(v,v) - l(v) + c$ on a vector space V_0 and with a symmetric positive bilinear form $a(\cdot,\cdot)$ are equivalent:*

- *the quadratic minimization problem (QMP) for J has a unique minimizer $u_* \in V_0$,*
- *the linear variational problem (LVP)*

$$u \in V_0 : \quad a(u,v) = l(v) \quad \forall v \in V_0$$

has a unique solution $u_* \in V$.

Needle loading example: $\delta \notin L^2$, no solution exists.

Continuous dependence A small perturbation of data $f \to f + \delta f$ leads to a perturbed solution $u \to u + \delta u$ satisfying

$$|\delta u|_1 \leq \operatorname{diam}(\Omega)\|\delta f\|_0.$$

Summary $\|\cdot\|_a$ determines the (Sobolev) space and "l has to fit this space", i.e. ; has to be bounded w.r.t. $\|\cdot\|_a$, otherwise no equilibrium exists.

5.4 Boundary Value Problems (BVP) for Equilibrium Models

We would like to derive the PDE associated with the LVP.

Lemma 5.13 (General product rule) *For all $\boldsymbol{j} \in (C^1(\overline{\Omega}))^d$, $v \in C^1(\Omega)$ holds* $\operatorname{div}(\boldsymbol{x}v) = v \operatorname{div} \boldsymbol{j} + \boldsymbol{j} \cdot \boldsymbol{grad}\, v$.

Theorem 5.14 (Gauss') *With $\boldsymbol{n} : \partial\Omega \to \mathbb{R}^d$ denoting the exterior unit normal vectorfield on $\partial\Omega$ and $\mathrm{d}S$ indicating integration over a surface, we have*

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

Theorem 5.15 (Green's first formula) *For all vector fields $\boldsymbol{j} \in (C^1_{pw}(\overline{\Omega}))^d$ and functions $v \in C^1_{pw}(\overline{\Omega})$ holds*

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

LVP $\to$ PDE Use Green's theorem and the fundamental lemma of calculus of variations.

Integrated boundary conditions If v does not vanish on the (entire) boundary, set $v \in C_0^\infty(\Omega)$ and eliminate boundary terms. Then switch back to original $v \in V$ and use the resulting PDE to eliminate terms in the previous formulation. This yields a boundary condition.

Extra smoothness for LVP $\to$ PDE The derivation of PDEs from LVPs hinges an extra smoothness of solution u of LVP. Therefore, LVP solutions (weak solutions) are more general than PDE solutions (strong solutions).

Non-pointwise boundary condition For some problems the boundary condition cannot be stated as a pointwise condition, i.e. only in integral-form. Try to insert a constant function, e.g. v=1, and check if the condition can be fulfilled via a pointwise condition. Check exercise 2.6/e for details.

5.5 Diffusion Models: Stationary Heat Conduction

Conversation of energy for all control volumes V

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n}\,\mathrm{d}S = \int_V f\,\mathrm{d}\mathbf{x},$$

power flux through surface V = heat production inside V,

where $f \in C^0_{pw}(\Omega)$ is a heat source or sink.

Fourier's law

$$\mathbf{j}(\mathbf{x}) = -\kappa(\mathbf{x})\,\mathbf{grad}\,u(\mathbf{x}), \quad \mathbf{x} \in \Omega,$$

where $\mathbf{j}$ denotes heat flux, u temperature, and κ heat conductivity.

5.6 Boundary Conditions

Mixing boundary conditions Different boundary conditions may be imposed on different boundaries as long as exactly one boundary conditions is imposed on each boundary.

Dirichlet boundary condition Temperature u is fixed with $g : \partial\Omega \to \mathbb{R}$ prescribed

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

Neumann boundary condition Heat flux $\mathbf{j}$ through $\partial\Omega$ is fixed with $h : \partial\Omega \to \mathbb{R}$ prescribed

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

Radiation boundary condition Heat flux through $\partial\Omega$ depends on (local) temperature with increasing function $\Psi : \mathbb{R} \to \mathbb{R}$

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

Combined with Fourier's law, we get the Robin boundary condition

$$\mathbf{grad}\,u \cdot \mathbf{n} + \gamma u = g \quad \text{on } \partial\Omega.$$

5.7 2nd-order Elliptic Variational Problems: BVP $\to$ LVP

Convert BVP $\to$ LVP

(i) Test PDE with smooth functions. Do not test where the solution is known e.g. on the boundary (instead set $v \equiv 0$ there).

(ii) Integrate over domain

(iii) Perform integration by parts, e.g. by using Green's first formula

(iv) Optionally, incorporate boundary conditions into boundary terms.

(v) Choose suitable function space (Sobolev space).

Example conversions for Dirichlet, Neumann, and Radiation BVPs.

Balance condition for Neumann BVP Neumann BVP need an additional condition to ensure uniqueness: $u \in H^1_*(\Omega)$

Theorem 5.16 (Second Poincaré-Friedrichs inequality) *If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is bounded, then*

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

5.8 Essential and natural Boundary Conditions

- **Essential boundary conditions** are directly imposed on trial space and (in homogeneous form) on test space.
- **Natural boundary conditions** are enforced only through the variational equation.

Admissible Dirichlet data If $g : \partial\Omega \to \mathbb{R}$ is piecewise continuously differentiable (and bounded piecewise derivatives), then it can be extended to an $u_0 \in H^1(\Omega)$ if and only if it is continuous on $\partial\Omega$.

Asmissible Neumann data $h \in L^2(\partial\Omega)$ (and hence $h \in C^0_{pw}(\partial\Omega)$) provides valid Neumann data for the 2nd order elliptic BVP. In particular, h may be discontinuous.

Theorem 5.17 (Multiplicative trace inequality)

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

6 Finite Element Methods (FEM)

6.1 Galerkin Discretization

Basic idea of Galerkin discretization

(i) Replace V_0 ($\dim V_0 = \infty$) in LVP with a finite dimensional subspace $V_{0,N} \subset V_0$ called Galerkin (or discrete) trial space/ test space ($\dim V_{0,N} = N$).

(ii) Introduce (ordered) basis $\mathcal{B}_N = \{b_N^1, \dots, b_N^N\} \subset V_{0,N}$.

(iii) Assemble matrix $A_{ij} = a(b_N^i, b_N^j)$ and right-hand side vector $\varphi_i = l(b_N^i)$.

(iv) Solve LSE $A\mu = \varphi$, where the solution $\mu = (\mu_1, \dots, \mu_N)^T$ are the coefficients of the solution $u_N = \sum_{k=1}^N \mu_k b_N^k$.

LVP discretization Ritz-Galkerin discretization of abstract (linear) variational problem

$$u \in V_0 : a(u,v) = l(v) \quad \forall v \in V_0 \overset{discrete}{\longrightarrow} u_N \in V_{0,N} : a(u_N, v_N) = l(v_N) \quad \forall v_N \in V_{0,N}$$

QMP discretization Ritz-Galerkin discretization of minimization problem for quadratic functional $J : V_0 \to \mathbb{R}$

$$u = \operatorname*{argmin}_{v \in V_0} J(v) \overset{discrete}{\longrightarrow} u_N \operatorname*{argmin}_{v_n \in V_{0,N}} J(v_N)$$

Theorem 6.1 (Existence + uniqueness of sols. of disc. LVP) *If the bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ is symmetric and positive definite and the linear form $l : V_0 \to \mathbb{R}$ is continuous in the sense of it exists $C_l > 0$ such that $|l(u)| \le C_l \|u\|_a$ for all $u \in V_0$, then the discrete variational problem has a unique Galerkin solution $u_N \in V_{0,N}$ that satisfies the stability estimate*

$$\|u_N\|_a \le C_l.$$

Theorem 6.2 (Independence of Galerkin solution of basis) *The choice of the basis $\mathcal{B}$ has no impact on the (set of) Galerkin solutions u_N.*

Lemma 6.3 (Change of basis) *Consider two bases $\mathcal{B} = \{b_N^1, \dots, b_N^N\}$, $\tilde{\mathcal{B}} = \{\tilde{b}_N^1, \dots, \tilde{b}_N^N\}$ related by the (regular) basis transformation matrix $S \in \mathbb{R}^{N,N}$ according to $\tilde{b}_N^j = \sum_{k=1}^N S_{jk} b_N^k$. Then the Galerkin matrices $A, \tilde{A} \in \mathbb{R}^{N,N}$, the righ-hand side vectors $\varphi, \tilde{\varphi}$ and the coefficient vectors $\mu, \tilde{\mu}$, respectively, satisfy*

$$\tilde{A} = SAS^T, \quad \tilde{\varphi} = S\varphi, \quad \tilde{\mu} = S^{-T}\mu.$$

Numerical properties like conditioning and sparsity depend on the choice of basis and are important for efficient solvers.

Derivations of LSE, change of basis.

6.2 1D Linear FEM

Domain $\Omega = [a, b]$

Mesh and properties

- Partition $\chi = \{a = x_0 < x_1 < \dots < x_{M-1} < x_M = b\}$
- Mesh $\mathcal{M} = \{(x_{j-1}, x_j) : 1 \le j \le M\}$
- Equidistant mesh for $x_j = a + jh$, $h = \frac{b-a}{M}$
- Cells $[x_{j-1}, x_j]$ for $j = 1, \dots, M$
- Cell size $h_j := |x_j - x_{j-1}|$ for $j = 1, \dots, M$
- Meshwidth $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$

Test space

$$V_{0,N} = \mathcal{S}_{1,0}^0(\mathcal{M}) := \{v \in C^0([a,b]) : v(a) = v(b) = 0, \; v|_{[x_{i-1}, x_i]} \text{ linear}, i = 1, \dots, M\}$$

Basis $b_N^j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \ne j \end{cases}$, for more info see section 3.1.

Matrix A

$$A = \begin{pmatrix} \frac{1}{h_1} + \frac{1}{h_2} & -\frac{1}{h_2} & 0 & 0 \\ -\frac{1}{h_2} & \frac{1}{h_2} + \frac{1}{h_3} & -\frac{1}{h_3} & \vdots \\ & & \ddots & -\frac{1}{h_{M-1}} \\ 0 & \dots & -\frac{1}{h_{M-1}} & \frac{1}{h_{M-1}} + \frac{1}{h_M} \end{pmatrix}$$

Right-hand side vector φ using composite trapezoidal rule for $1 \le k \le N$

$$\varphi_k = \int_a^b f(x) b_n^k(x)\,\mathrm{d}x \approx \frac{1}{2}(h_k + h_{k+1}) f(x_k).$$

6.3 2D Triangular Linear FEM

2D triangular Meshes

Triangulation Requirements for a proper triangular mesh:

(i) $\mathcal{M} = \{K_i\}_{i=1}^M$, $M \in \mathbb{N}$, K_i = open triangle

(ii) disjoint interiors: $K_i \cap K_j = \emptyset$ for $i \ne j$

(iii) tiling/ partitioning property $\bigcup_{i=1}^M \overline{K}_i = \overline{\Omega}$

(iv) intersection: for $i \ne j$ $\overline{K}_i \cap \overline{K}_j$ is either empty, an edge of both triangles, or a vertex of both triangles.

Data structure for triangles could for example store for each vertex i its coordinates (x_1^i, x_2^i) and the three vertex indices of every triangle K_j.

i	x_1^i	x_2^i	K_j	$i_{j,1}$	$i_{j,2}$	$i_{j,3}$
0	-1	+1	0	0	1	8
1	-1	0	1	1	4	8
2	-1	-1	2	4	7	8
⋮			⋮			

Space of linear FE functions

$$V_{0,N} = \mathcal{S}_1^0(\mathcal{M}) = \big\{v \in C^0(\overline{\Omega}) : \forall K \in \mathcal{M} : v|_K(\mathbf{x}) = \alpha_k + \boldsymbol{\beta}_K \mathbf{x}, \; \alpha_K \in \mathbb{R}, \boldsymbol{\beta}_K \in \mathbb{R}^2, \mathbf{x} \in K\big\} \subset H^1(\Omega)$$

Nodal basis functions

Basis $b_N^{\mathbf{x}}(\mathbf{y}) = \begin{cases} 1 & \mathbf{x} = \mathbf{y} \\ 0 & \mathbf{y} \in \mathcal{V}(\mathcal{M}) \backslash \{\mathbf{x}\} \end{cases}$.

Dimension $\dim \mathcal{S}_1^0(\mathcal{M}) = \#\mathcal{V}(\mathcal{M})$

Discussion on existence of such basis functions

Sparse Galerkin matrix

(Nodes $x^i, x^j \in \mathcal{V}(\mathcal{M})$ not connected by an edge $\iff$ $\mathrm{Vol}(\mathrm{supp}(b_N^i) \cap \mathrm{supp}(b_N^j)) = 0$) $\implies$ $A_{ij} = 0$.

Computation of Galerkin matrix

Assembly We would like to assemble the global Galkerin matrix A from smaller local matrices A_K.

Example of assembly Consider two points x^1, x^j connected by an edge of triangles K_1, K_2. Then $A_{ij} = \int_\Omega \mathbf{grad}\, b_N^j \,\mathbf{grad}\, b_N^i \,\mathrm{d}\mathbf{x} = \int_{K_1} \mathbf{grad}\, b_N^j|_{K_1} \,\mathbf{grad}\, b_N^i|_{K_1} \,\mathrm{d}\mathbf{x} + \int_{K_2} \mathbf{grad}\, b_N^j|_{K_2} \,\mathbf{grad}\, b_N^i|_{K_2} \,\mathrm{d}\mathbf{x}$. Instead of directly computing A, we can assemble A using the matrices A_{K_1} and A_{K_2}.

Barycentric coordinate functions λ_i of triangle K_k are the global basis functions restricted to K_k,

$$\lambda_i = b_N^{\mathrm{dofh}(k,i)}|_{K_k} \qquad i = 1, 2, 3.$$

Theorem 6.4 (Integration of barycentric coordinate func.) *For any non-degenerate d-simplex K with barycentric coordinate functions $\lambda_1, \dots, \lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$ for $j = 1, \dots, d+1$,*

$$\int_K \lambda_1^{\alpha_1} \dots \lambda_{d+1}^{\alpha_{d+1}} \,\mathrm{d}x = d!|K| \frac{\alpha_1! \alpha_2! \dots \alpha_{d+1}!}{(\alpha_1 + \alpha_2 + \dots + \alpha_{d+1} + d)!}, \ \forall \alpha \in \mathbb{N}_0^{d+1}.$$

Data structure dofh maps local to global indices such that

$$\mathrm{dofh}(k,l) = \text{global number of vertex } l \text{ of } k\text{-th cell},$$
$$\mathbf{x}^{\mathrm{dofh}(k,l)} = \mathbf{a}^l \text{ when } \mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3 \text{ are vertices of } K_k,$$

for $l \in \{1,2,3\}$, $k \in \{1, \dots, N\}$, $N = \#\mathcal{V}(\mathcal{M})$.

Algorithm 1 Efficient assembly of Galerkin matrix

```
for all elements i = 1, ..., M do
    Get vertex indices i_j with j = 1, 2, 3 of the i-th element
    Store the element's vertices coordinates x^{i_j}
    Compute element matrix A_{K_i} for the vertices
    for k, l = 1, 2, 3 do
        A_{j_k,j_l} = A_{j_k,j_l} + (A_{K_i})_{k,l}
```

Computation of r.h.s vector

Assembly of r.h.s. vector

$\varphi_j = \int_\Omega f(\mathbf{x}) b_N^j \,\mathrm{d}\mathbf{x} = \sum_{K: x^j \in \overline{K}} \int_K f(\mathbf{x}) b_N^j|_K(\mathbf{x}) \,\mathrm{d}\mathbf{x}$

Quadrature for r.h.s. vector using 2D trapezoidal rule is $(\varphi_K)_j = \int_K f(\mathbf{x}) \lambda_j(\mathbf{x}) \,\mathrm{d}\mathbf{x} \approx \frac{|K|}{3} f(\mathbf{a}^j)$

6.4 Building Blocks for General Finite Element Methods

6.4.1 Meshes/ Triangulations

Definition 6.5 (Mesh) *or triangulation of $\Omega \subset \mathbb{R}^d$ is a finite collection $\{K_i\}_{i=1}^M$, $M \in \mathbb{N}$ of open non-degenerate (curvilinear) polygons ($d = 2$)/ polygons ($d = 3$) such that*

(i) $\overline{\Omega} = \bigcup_i \overline{K}_i$

(ii) $K_i \cap K_j = \emptyset \iff i \neq j$,

(iii) for all $i, j \in \{1, \dots, M\}$ for $i \neq j$, the intersection $\overline{K}_i \cap \overline{K}_j$ is either empty or a vertex, edge, or face of both K_i and K_j.

6.4.2 Polynomials

Definition 6.6 (Multivariate Polynomials) *The space of multivariate (d-variate) polynomials of (total) degree $p \in \mathbb{N}_0$ is defined as*

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

Lemma 6.7 (Dimension of spaces of polynomials)

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

Definition 6.8 (Tensor Product Polynomials) *The space of tensor product polynomials of degree $p \in \mathbb{N}$ in each coordinate direction is defined as*

$$\mathcal{Q}_p(\mathbb{R}^d) = \left\{ \boldsymbol{x} \to \sum_{l_1=0}^p \dots \sum_{l_d=0}^p c_{l_1,\dots,l_d} x_1^{l_1} \dots x_d^{l_d}, \ c_{l_1,\dots,l_d} \in \mathbb{R} \right\}.$$

Lemma 6.9 (Dimension of spaces of tensor prod. polyn.)

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

6.4.3 Basis Functions

Requirements on basis functions for a finite element trial/ test space $V_{0,N}$

(i) $\mathcal{B}_N := \{b_N^1, \dots, b_N^N\}$ is basis of $V_{0,N}$ with $N = \dim V_{0,N}$

(ii) each b_N^i is associated with a single geometric entity (cell/ edge/ face/ vertex) of $\mathcal{M}$

(iii) $\mathrm{supp}(b_N^i) = \bigcup_{K \in \mathcal{M}: \mathbf{p} \subset \overline{K}} \overline{K}$ if b_N^i is associated with cell/ edge/ face/ vertex $\mathbf{p}$

Sparse Galerkin matrix For $\mathrm{supp}(b_N^i) \cap \mathrm{supp}(b_N^j) = \emptyset \implies A_{ij} = 0$ ensures that $nnz(A) = \mathcal{O}(N)$.

Definition 6.10 (Local shape functions) *Given a finite element function space on a mesh $\mathcal{M}$ with global shape functions b_N^i for $i = 1, \dots, N$, the set of local shape functions on $K \in \mathcal{M}$ is given by $\{b_N^j|_K, K \subset \mathrm{supp}(b_N^j)\}$.*

6.5 Lagrangian Finite Element Space

Definition 6.11 (Simplicial Lagrangian finite element space) *Space of p-th degree Lagrangian finite element functions on simplicial mesh $\mathcal{M}$*

$$\mathcal{S}_p^0(\mathcal{M}) := \{v \in C^0(\overline{\Omega}) : v|_K \in \mathcal{P}_p(K) \ \forall K \in \mathcal{M}\}$$

Definition 6.12 (Tensor product Lagrangian finite el. spaces) *Space of p-th degree Lagrangian finite element functions on tensor product mesh $\mathcal{M}$*

$$\mathcal{S}_p^0(\mathcal{M}) := \{v \in C^0(\overline{\Omega}) : v|_K \in \mathcal{Q}_1(K) \ \forall K \in \mathcal{M}\}.$$

Construction of basis functions The general procedure is to show that a cardinal basis of continuous functions exists for a given set of interpolation nodes. First choose a proper set of interpolation nodes. Then choose local shape functions b_K^i and glue them together such that the resulting function is continuous.

Hybrid meshes in 2D use simplicial Lagrangian FE for triangles and tensor product Lagrangian FE for rectangles, i.e.

$$\mathcal{S}_p^0(\mathcal{M}) = \left\{ v \in H^1(\Omega) : v|_K \in \begin{cases} \mathcal{P}_1(\mathbb{R}^2) & K \in \mathcal{M} \text{ is triangle} \\ \mathcal{Q}_1(\mathbb{R}^2) & K \in \mathcal{M} \text{ is rectangle} \end{cases} \right\}.$$

6.6 Implementation of Finite Element Space

6.6.1 Mesh

We have $\mathcal{O}(1)$-access from cells to edges, vertices and neighboring cells.

Numbering convention

Geometric information

6.6.2 Quadrature

Transformed quadrature rule

$$\int_{\hat{K}} f(\hat{\mathbf{x}}) \,\mathrm{d}\hat{\mathbf{x}} \approx \sum_{l=1}^p \hat{\omega}_l f(\hat{\boldsymbol{\xi}}_l) \quad \to \quad \int_K f(\mathbf{x}) \,\mathrm{d}\mathbf{x} \approx \frac{|K|}{|\hat{K}|} \sum_{l=1}^p \omega_l^K f(\boldsymbol{\xi}_l^K)$$

with $\omega_l^K = \hat{\omega}_l$, $\boldsymbol{\xi}_l^K = \Phi_K(\hat{\boldsymbol{\xi}}_l)$.

Definition 6.13 (Order) *A quadrature rule is of **order** $q \in \mathbb{N}$ if*

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

Order is invariant under affine transformation.

6.6.3 Treatment of essential Boundary Condition

Offset function trick for 2nd-order elliptic Dirichlet problem with non-zero data. Seek $u \in H^1(\Omega)$ with $u|_{\partial\Omega} = g$ such that

$$\int_\Omega \alpha(\mathbf{x}) \,\mathbf{grad}\, u(\mathbf{x}) \,\mathbf{grad}\, v(\mathbf{x}) \,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) \,\mathrm{d}\mathbf{x}$$

for all $v \in H_0^1(\Omega)$. This problem is equivalent to: Seek $w \in H_0^1(\Omega)$ with $u|_{\partial\Omega} = g$ such that

$$\int_\Omega \alpha(\mathbf{x}) \operatorname{\mathbf{grad}} w(\mathbf{x}) \operatorname{\mathbf{grad}} v(\mathbf{x}) \,\mathrm{d}\mathbf{x} = \int_\Omega -\kappa(\mathbf{x}) \operatorname{\mathbf{grad}} u_0 \operatorname{\mathbf{grad}} v + f(\mathbf{x}) v(\mathbf{x}) \,\mathrm{d}\mathbf{x}$$

for all $v \in H_0^1(\Omega)$ with offset function $u_0 \in H^1(\Omega)$ satisfying $u_0 = g$ on $\partial\Omega$.

Function u_0 for $V_N = \mathcal{S}_1^0(\mathcal{M})$ and Dirichlet data $g \in C^0(\partial\Omega)$ use

$$u_0 = \sum_{\mathbf{x} \in \mathcal{V}(\mathcal{M}) \cap \partial\Omega} g(\mathbf{x}) b_N^{\mathbf{x}},$$

where $b_N^{\mathbf{x}}$ denotes the tent function associated with $\mathbf{x} \in \mathcal{V}(\mathcal{M})$.

LSE with essential boundary condition is given by

$$\mathbf{A}_0 \boldsymbol{\nu} = \varphi - \mathbf{A}_{0\partial} \boldsymbol{\gamma},$$

where $\mathbf{A}_0 \in \mathbb{R}^{N,N}$ is the Galerkin matrix for space $\mathcal{S}_{1,0}^0(\mathcal{M})$ and $\mathbf{A} = \begin{pmatrix} \mathbf{A}_0 & \mathbf{A}_{0\partial} \\ \mathbf{A}_{\partial 0} & \mathbf{A}_{\partial\partial} \end{pmatrix} \in \mathbb{R}^{M,M}$ the Galerkin matrix for space $\mathcal{S}_1^0(\mathcal{M})$. The solution can be obtained from $u \approx u_N = u_0 + w_N = \sum_{j=N+1}^{M} \gamma_j b_N^j + \sum_{j=1}^{N} \nu_j b_N^j$.

6.7 Parametric Finite Elements

Lemma 6.14 (Presevation of polynomials under affine pullb.) *If $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ is an affine (linear) transformation, then*

$$\Phi^*(\mathcal{P}_p(\mathbb{R}^d)) = \mathcal{P}_p(\mathbb{R}^d) \quad \textit{and} \quad \Phi^*(\mathcal{Q}_p(\mathbb{R}^d)) = \mathcal{Q}_p(\mathbb{R}^d).$$

This section is TODO

7 Finite Differences (FD) and Finite Volume (FV)

7.1 Finite Differences

Replace derivatives with difference quotients in finitely many points, i.e. at the nodes of a mesh/ grid.

$$\begin{aligned}
\frac{\partial u(x)}{\partial x} &\approx \frac{u(x+h) - u(x)}{h} \\
\frac{\partial^2 u(x)}{\partial x^2} &\approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \\
\frac{\partial^2 u}{\partial x_1^2}|_{\mathbf{x}=(a,b)} &\approx \frac{u(a+h,b) - 2u(a,b) + u(a-h,b)}{h^2} \\
\frac{\partial^2 u}{\partial x_2^2}|_{\mathbf{x}=(a,b)} &\approx \frac{u(a,b+b) - 2u(a,b) + u(a,b-h)}{h^2} \\
-\Delta u|_{\mathbf{x}=(a,b)} &\approx \frac{-u(a-h,b) - u(a+h,b)}{h^2} \\
&\quad + \frac{4u(a,b) - u(a,b-h) - u(a,b+a)}{h^2}
\end{aligned}$$

2D discretization We discretize $-\Delta u = f$ on a equidistant tensor-product mesh of $(0,1)^2$ with meshwidth h such that $u(ih, jh) = \mu_{i,j}$ for all $1 \le i, j \le N$. This leads to the system

$$-\Delta u(ih, jh) = \frac{-\mu_{i-1,j} - \mu_{i+1,j} + 4\mu_{i,j} - \mu_{i,j-1} - \mu_{i,j+1}}{h^2} = f(ih, jh).$$

Stencil notation for $-\Delta$ at point (ih, jh) is $\begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}$.

Lexikographic ordering defines $\mu_{(j-1)N+i} := \mu_{i,j}$ and $\varphi_{(j-1)N+i} := f(ih, jh)$.

LSE of $-\Delta u = f$ using lexikographic ordering leads is $A\mu = \varphi$

$$\mathbf{A} = \frac{1}{h^2} \begin{pmatrix} \mathbf{T} & -\mathbf{I} & 0 & \dots \\ -\mathbf{I} & \mathbf{T} & -\mathbf{I} & \\ 0 & & \ddots & \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} 4 & -1 & 0 & \dots \\ -1 & 4 & -1 & \\ 0 & & \ddots & \end{pmatrix}.$$

FD and FE Galkerin Most finite difference schemes correspond to FE Galerkin schemes with numerical quadrature on structured meshes.

7.2 Finite Volume

Control volumes C_i for $i = 1, \dots, \tilde{M}$ are cells of a mesh $\tilde{\mathcal{M}} = \{C_i\}$ covering computational domain Ω. Associate with each cell C_i a nodal value μ_i such that $\mu_i \approx u(\mathbf{p}_i)$, where $\mathbf{p}_i$ is the center of C_i.

Local numerical fluxes J_{ik} for two adjacent cells C_k, C_i with common edge $\Gamma_{ik} := \overline{C}_i \cap \overline{C}_k$ is given by

$$J_{ik} = \Psi(\mu_i, \mu_k) \approx \int_{\Gamma_{ik}} \mathbf{j} \cdot \mathbf{n}_{ik} \,\mathrm{d}S,$$

where Ψ is the numerical flux function

FV LSE Is must hold that $\int_{C_i} f \,\mathrm{d}x = \int_{\partial C_i} \mathbf{j} \cdot \mathbf{n} \,\mathrm{d}S$

$$\sum_{k \in \mathcal{U}_i} J_{ik} = \sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) = |C_i| f(\mathbf{p}_i) = \int_{C_i} f \,\mathrm{d}x \quad \forall i = 1, \dots, \tilde{\mathcal{M}},$$

where $\mathcal{U}_i = \{j : C_i \text{ and } C_j \in \tilde{\mathcal{M}} \text{ share common edge}\}$

Voronoi dual mesh for $\mathcal{V}(\mathcal{M}) = \{\mathbf{p}_1, \dots, \mathbf{p}_M\}$, define the Voronoi dual mesh $\tilde{\mathcal{M}} := \{C_i\}_{i=1}^M$, where C_i are the Voronoi cells

$$C_i := \{\mathbf{x} \in \Omega : |\mathbf{x} - \mathbf{p}_i| < |\mathbf{x} - \mathbf{p}_j| \forall j \ne i\}. \qquad \text{(Voronoi cells)}$$

Voronoi cells are constructed by perpendicular bisectors for edges and circumcenters of triangles for nodes.

Theorem 7.1 (Angle condition for Voronoi dual meshes) *The following angle conditions ensure that the Voronoi cells belonging to adjacent nodes of a triangular mesh have a common edge ($\overline{C}_i \cap \overline{C}_j \ne \emptyset \iff$ nodes i, j are connected by an edge of $\mathcal{M}$):*

(i) sum of angles facing interior edge $\le \pi$,

(ii) angles facing boundary edges $\le \frac{\pi}{2}$.

Definition 7.2 (Delaunay triangulation) *are triangular meshes satisfying the angle conditions from theorem 7.1.*

Barycentric dual mesh edges are the union of likes connecting barycenters and midpoints of edges of $\mathcal{M}$ and nodes are barycenters of triangles.

Definition 7.3 (Barycenter) *of a triangle with vertices $\boldsymbol{a}^1$, $\boldsymbol{a}^2$, $\boldsymbol{a}^3$ is the point $\boldsymbol{p} := \frac{1}{3}(\boldsymbol{a}^1 + \boldsymbol{a}^2 + \boldsymbol{a}^3)$.*

Neumann b.c. $\mathbf{j} \cdot \mathbf{n} = -h$ on $\partial\Omega$

$$\sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) - \int_{\partial C_i \cap \partial\Omega} h \,\mathrm{d}S = |C_i| f(\mathbf{p}_i)$$

Dirichlet b.c. $u = 0$ on $\partial\Omega$ makes coefficients $\mu_k = 0$ if $p_k \in \partial\Omega$. Keep only equations for p_i where $p_i \in \Omega$.

Numerical flux

$$\Psi(\mu_i, \mu_j) := \int_{\Gamma_{ik}} -\operatorname{\mathbf{grad}} I\mu \cdot \mathbf{n} \,\mathrm{d}S, \qquad I\mu = \sum_j \mu_j b_N^j$$

FV equations for concrete numerical flux

$$\sum_{k \in \mathcal{U}_i} \int_{\Gamma_{ik}} \operatorname{\mathbf{grad}} I\mu \cdot \mathbf{n} \,\mathrm{d}S = \mu_i \int_{\partial C_i} \operatorname{\mathbf{grad}} b_N^i \cdot \mathbf{n}_i \,\mathrm{d}S + \sum_{j \in \mathcal{U}_i} \mu_j \sum_{k \in \mathcal{U}_i} \int_{\Gamma_{ik}} \operatorname{\mathbf{grad}} b_N^j \cdot \mathbf{n}_{ik} \,\mathrm{d}S = \int_{C_i} f \,\mathrm{d}x.$$

It follows that the matrix entry is $A_{ij} = -\int_{\partial C_i} \mathbf{grad}\, b_N^j \cdot \mathbf{n}_i \,\mathrm{d}S$.

FV and FE Galerkin The FV discretization and the finite element Galerking discretization spawn the same system matrix for the model problem $-\Delta u = 0$ in Ω and $u = 0$ on $\partial\Omega$.

8 Convergence and Accuracy

8.1 Galerkin Error Estimate

Goal We would like to bound the en energy norm of the discretization error $\|u - u_N\|_a$. Energy norm, because the error in energy $|J(u) - J(u_N)| \leq \frac{1}{2}\|u + u_N\|_a \|u - u_N\|_a$ can be bounded with the energy norm $\|\cdot\|_a$.

Galerkin orthogonality for all $w_N \in V_{0,N}$ it holds that

$$a(u - u_N, w_N) = 0,$$
$$\|u - w_N\|_a^2 = \|u - u_N\|_a^2 + \|u_N - w_N\|_a^2.$$

Theorem 8.1 (Cea's lemma, Optimality of Galerkin sol.) *Assume that the bilinear form $a : V_0 \times V_0 \to \mathbb{R}$ is symmetric positive definite, the right hand side functional $l : V_0 \to \mathbb{R}$ is continuous with respect to the energy norm induced by a, and V_0 equipped with the energy norm $\|\cdot\|_a$ is a Hilbert space. Then the energy norm of the Galerkin discretization error satisfies*

$$\|u - u_N\|_a = \inf_{v_N \in V_{0,N}} \|u - v_N\|_a,$$

i.e. the discretization error of the Galerkin solution is the best approximation error in energy.

Refinement by increasing polynomial degree $\mathcal{S}_p^0(\mathcal{M}) \subset \mathcal{S}_{p+1}^0(\mathcal{M})$ (p-refinement) or mesh refinement (h-refinement).

8.2 A-priori FE Error Estimates

1D error estimate predicts algebraic convergence with rate 1

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

Lemma 8.2 (Local interpolation error estimate) *For any triangle K and $u \in C^2(\overline{K})$ the following holds*

$$\|u - I_1 u\|_{L^2(K)}^2 \leq \frac{3}{8} h_K^4 \|\|\mathrm{D}^2 u\|_F\|_{L^2(K)}^2,$$
$$\|\boldsymbol{grad}(u - I_1 u)\|_{L^2(K)}^2 \leq \frac{3}{24} \frac{h_{\mathcal{M}}^6}{|K|^2} \|\|\mathrm{D}^2 u\|_F\|_{L^2(K)}^2.$$

Shape regularity measure $\rho := h_K^2/|K|$ ρ_k large $\iff$ K "distorted" $\iff$ K has small angles. We define $\rho_{\mathcal{M}} := \max_K \rho_K$.

Lemma 8.3 (Error estimate for p.w. linear interpolation) *For any $u \in C^2(\overline{K})$ and 2D piecewise linear interpolation $I_1 : C^0(\overline{K}) \to \mathcal{S}_1^0(\mathcal{M})$ for a triangular mesh $\mathcal{M}$, holds*

$$\|u - I_1 u\|_{L^2(K)} \leq \sqrt{\frac{3}{8}} h_K^2 \|\|\mathrm{D}^2 u\|_F\|_{L^2(K)},$$
$$\|\boldsymbol{grad}(u - I_1 u)\|_{L^2(K)} \leq \sqrt{\frac{3}{24}} h_{\mathcal{M}} \rho_{\mathcal{M}} \|\|\mathrm{D}^2 u\|_F\|_{L^2(K)}$$
$$= \sqrt{\frac{3}{24}} h_{\mathcal{M}} \rho_{\mathcal{M}} |u|_{H^2(\Omega)},$$

where $h_{\mathcal{M}}$ denotes the mesh width and $\rho_{\mathcal{M}}$ the shape regularity measure of $\mathcal{M}$.

Definition 8.4 (Higher order Sobolev norms) *The m-th order Sobolev (semi-)norm $m \in \mathbb{N}$ for sufficiently smooth $u : \Omega \to \mathbb{R}$ and m-th order Sobolev space are defined as*

$$|u|_{H^m(\Omega)}^2 := \sum_{\boldsymbol{\alpha} \in \mathbb{N}^d, |\boldsymbol{\alpha}| = m} \int_\Omega |\,\mathrm{D}^{\boldsymbol{\alpha}}|^2 \,\mathrm{d}\boldsymbol{x}$$
$$\|u\|_{H^m(\Omega)}^2 := \sum_{k=0}^m \sum_{\boldsymbol{\alpha} \in \mathbb{N}^d, |\boldsymbol{\alpha}| = k} \int_\Omega |\,\mathrm{D}^{\boldsymbol{\alpha}}|^2 \,\mathrm{d}\boldsymbol{x},$$
$$H^m(\Omega) := \{v : \Omega \to \mathbb{R} : \|v\|_{H^m(\Omega)} < \infty\}$$

with $\mathrm{D}^{\boldsymbol{\alpha}} := \frac{\partial^{|\boldsymbol{\alpha}|}}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}}$.

Theorem 8.5 (Sobolev embedding theorem) *For $m > \frac{d}{2}$ it follows that $H^m(\Omega) \subset C^0(\overline{\Omega})$ and there exists $C = C(\Omega) > 0$ such that $\|u\|_\infty \leq C\|u\|_{H^m(\Omega)}$ for all $u \in H^m(\Omega)$.*

8.3 General Approximation Error Estimates (for Lagrangian FE)

Theorem 8.6 (Lagrangian FE best approximation err. estim.) *Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygon/ polyhedral domain equipped with a mesh $\mathcal{M}$ consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{R}$, it exists a constant $C > 0$ depending only on k and the shape regularity measure $\rho_{\mathcal{M}}$ such that*

$$\inf_{v_N \in S_p^0(\mathcal{M})} \|u - v_N\|_{H^1(\Omega)} \leq C \left(\frac{h_{\mathcal{M}}}{\rho}\right)^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)}$$

for all $u \in H^k(\Omega)$.

Approximate gain of accuracy The number of unknowns can be approximated by $N := \dim \mathcal{S}_p^0(\mathcal{M}) \approx h^{-d} p^d = (\frac{p}{h_{\mathcal{M}}})^d$, then

$$\|u - u_N\|_a \leq C N^{-\frac{\min\{p+1,k\}-1}{d}} \|u\|_{H^k(\Omega)}.$$

Therefore, reducing the energy norm of the error by a factor $\rho > 1$ (cost) requires an approximate increase of the problem size by a factor $\rho^{\frac{\min\{p+1,k\}-1}{d}}$ (gain).

8.4 Elliptic Regularity Theory

Corners in the boundary are bad for u as they prevent u from being in $H^2(\Omega)$ (or higher).

Theorem 8.7 (Smooth elliptic lifting theorem) *If $\partial\Omega$ is C^∞-smooth, i.e. possesses a local parameterization by C^∞-functions, and $\sigma \in C^\infty(\overline{\Omega})$, and*

- *if $u \in H_0^1(\Omega)$ and $-\operatorname{div}(\sigma\, \boldsymbol{grad}\, u) \in H^k(\Omega)$ then $u \in H^{k+2}(\Omega)$ for any $k \in \mathbb{N}$,*
- *if $u \in H^1(\Omega)$, $-\operatorname{div}(\sigma\, \boldsymbol{grad}\, u) \in H^k(\Omega)$ and $\boldsymbol{grad}\, u \cdot \boldsymbol{n} = 0$ on $\partial\Omega$ then $u \in H^{k+2}(\Omega)$ for any $k \in \mathbb{N}$.*

Furthermore, for an u satisfying one of the above conditions, there is a constant $C = C(k, \Omega, \sigma)$ such that

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

Theorem 8.8 (Corner singular function decomposition) *Let $\Omega \subset \mathbb{R}^2$ be a polygon with J corners $\boldsymbol{c}^i$. Denote the polar coordinates of the corner $\boldsymbol{c}^i$ by (r_i, φ_i) and the inner angle at the corner $\boldsymbol{c}^i$ by ω_i. Additionally, let $f \in H^l(\Omega)$ with $l \in \mathbb{N}_0$ and $l \neq \lambda_{ik} - 1$, where λ_{ik} are given by the singular exponents $\lambda_{ik} = \frac{k\pi}{\omega_i}$ for $k \in \mathbb{N}$. Then $u \in H_0^1(\Omega)$ with $-\Delta u = f$ in Ω can be decomposed*

$$u = u^0 + \sum_{i=1}^J \Psi(r_i) \sum_{\lambda_{ik} < l+1} \kappa_{ik} s_{ik}(r_i, \varphi_i), \quad \kappa_{ik} \in \mathbb{R},$$

with regular part $u^0 \in H^{l+2}(\Omega)$, cut-off function $\Psi \in C^\infty(\mathbb{R}^+)$ such that $\Psi \equiv 1$ in a neighborhood of 0, and corner singular functions

$$\lambda_{ik} \notin \mathbb{N}: \quad s_{ik}(r, \varphi) = r^{\lambda_{ik}} \sin(\lambda_{ik}\varphi),$$
$$\lambda_{ik} \in \mathbb{N}: \quad s_{ik}(r, \varphi) = r^{\lambda_{ik}} (\ln r) \sin(\lambda_{ik}\varphi).$$

8.5 Variational Crimes

Acceptable variational crimes must not affect (type and rate) of asymptotic convergence when solving a perturbed variational problem.

Numerical quadrature To archive $\|u - u_N\|_1 = \mathcal{O}(h_{\mathcal{M}}^p)$ at best, quadrature rule of order $2 - 1$ is sufficient for right-hand side functional f_N and bilinear form a_N.

Boundary approximation If $V_{0,N} = \mathcal{S}_p^0(\mathcal{M})$, use boundary fitting with polynomials of degree p.

8.6 Duality Techniques

We are now interested in the number $F(u)$ for given functional $F : V_0 \to \mathbb{R}$.

Assumptions on output functional F are linearity and continuous with respect to the energy norm, i.e. there exists a constant $C_f > 0$ such that $|F(v)| \le C_f \|v\|_a$ for all $v \in V_0$.

Output error can be bounded with the discretization error in energy norm by using the assumptions

$$|F(u) - F(u_N)| = |F(u - u_N)| \le C_f \|u - u_N\|_a.$$

Theorem 8.9 (Duality estimate for lin. functional output) *Define the dual solution $g_F \in V_0$ to F as the solution of the dual variational problem $g_F \in V_0 : a(v, g_F) = F(v)$ for all $v \in V_0$. Then*

$$|F(u) - F(u_N)| \le \|u - u_N\|_a \inf_{v_N \in V_{0,N}} \|g_F - v_N\|_a.$$

Theorem 8.10 (Elliptic lifting theorem on convex domains) *If $\Omega \subset \mathbb{R}^2$ convex, $u \in H^1_0(\Omega)$, and $\Delta u \in L^2(\Omega)$ then $u \in H^2(\Omega)$.*

Trick to deal with non-continuous functionals Consider the non-continuous functional $J(u) = \int_{\Gamma_i} \kappa \operatorname{\mathbf{grad}} u \cdot \mathbf{n} \, \mathrm{d}S$. Define the cut-off function $\Psi \in C^0(\overline{\Omega}) \cap H^1(\Omega)$ such that $\Psi|_{\Gamma_i} \equiv 1$ and $\Psi|_{\Gamma_0} = 0$. Define the continuous functional $J^*(u) = \int_\Omega \kappa \operatorname{\mathbf{grad}} u \operatorname{\mathbf{grad}} Psi$. Then (using Green's formula and the given problem) $J(u) = J^*(u)$ for solutions of the problem.

Theorem 8.11 (Error estimate for p.w. linear interpolation) *For any $u \in C^2(\overline{\Omega})$ and 2D piecewise linear interpolation $I_1 : C^0(\overline{\Omega}) \to \mathcal{S}^0_1(\mathcal{M})$ for a triangular mesh $\mathcal{M}$ holds*

$$\|u - I_1 u\|_{L^2(\Omega)} \le \sqrt{\frac{3}{8}} h^2_{\mathcal{M}} \| \|D^2 u\|_F \|_{L^2(\Omega)} = \mathcal{O}(h^2_{\mathcal{M}}),$$

$$\|\boldsymbol{grad}(u - I_1 u)\|_{L^2(\Omega)} \le \sqrt{\frac{3}{24}} \rho_{\mathcal{M}} h_{\mathcal{M}} \| \|D^2 u\|_F \|_{L^2(\Omega)} = \mathcal{O}(h_{\mathcal{M}}).$$

8.7 Discrete Maximum Principle

Theorem 8.12 (Discrete Maximum Principle) *For $u \in C^0(\overline{\Omega}) \cap H^1(\Omega)$ holds the maximum principle*

$$-\operatorname{div}(\kappa \boldsymbol{grad} u) \ge 0 \implies \min_{\boldsymbol{x} \in \partial\Omega} u(\boldsymbol{x}) = \min_{\boldsymbol{x} \in \Omega} u(\boldsymbol{x})$$
$$-\operatorname{div}(\kappa \boldsymbol{grad} u) \le 0 \implies \max_{\boldsymbol{x} \in \partial\Omega} u(\boldsymbol{x}) = \max_{\boldsymbol{x} \in \Omega} u(\boldsymbol{x})$$

Averaging argument Consider $-\Delta u = 0$ in Ω and $u = g$ on $\partial\Omega$. Then the coefficient recursion is given by

$$\mu_{i,j} = \frac{1}{4}(\mu_{i-1,j} + \mu_{i+1,j} + \mu_{i,j-1} + \mu_{i,j+1}),$$

which is an average formula. If μ_{ij} is assumed to be maximal or minimal, it follows that all coefficients are equal $\mu_{i,j} = \mu_{i+1,j} = \mu_{i-1,j} = \dots$ and hence u_N is constant, because averages are smaller or equal than the maximal averaged over elements.

Angle conditions ensuring maximum principle are $\alpha + \beta \le \pi$, where α and β are opposite angles of two triangles connected by an edge, and $\sum_{\mathbf{x} \in \mathcal{V}(\mathcal{M})} b^{\mathbf{x}}_N$.

8.8 Validation and Debugging of Finite Element Codes

Model problem Let $\Omega \subset \mathbb{R}^d$ with $d = 2, 3$ and $\partial\Omega = \overline{\Gamma}_D \cup \overline{\Gamma}_N \cup \overline{\Gamma}_R$. Seek $u \in H^1(\Omega)$ such that $u = g$ on Γ_D and

$$\int_\Omega \boldsymbol{\alpha} \operatorname{\mathbf{grad}} u \operatorname{\mathbf{grad}} u + \gamma u v \, \mathrm{d}\mathbf{x} + \int_{\Gamma_R} \lambda u v \, \mathrm{d}S = \int_\Omega f v \, \mathrm{d}\mathbf{x} + \int_{\Gamma_N} h v \, \mathrm{d}S$$

for all $v \in H^1_{\Gamma_D}(\Omega) = \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}$. This problem has Dirichlet b.c. on Γ_D, Neumann b.c. on Γ_N and Robin b.c. on Γ_R.

Method of manufactured solutions

(i) Pick a simple domain Ω (no boundary approximations).

(ii) Use coefficients $\boldsymbol{\alpha}$, γ given by simple analytic formulas.

(iii) Pick $u \in C^\infty(\overline{\Omega})$ as an exact solution with simple analytic formula (no polynomials) and set f, g, h accordingly.

(iv) Use code: solve manufactured BVP to get u_N.

(v) Use code to compute $\|u - u_N\|$ with overkill quadrature.

(vi) Estimate rate of convergence and match with theoretical predictions. In case of mismatch, simplify the problem, e.g. $\Gamma_D = \partial\Omega$, $g = 0$, $\lambda = 0$, $\boldsymbol{\alpha} = 1$, ... As a last resort, use $u \in \mathcal{P}_p(\mathbb{R}^d)$, this must give $u_N = u$.

Direct testing of (bi-)linear forms

(i) Use simple polynomial/ circular Ω.

(ii) Choose $w \in C^\infty(\overline{\Omega})$ with simple analytic formula (no polynomials).

(iii) Compute $b(w, w)$ exactly.

(iv) Use code to compute $b(I_p w, I_p w)$ on sequence of meshes, where $I_p w$ is the interpolation on mesh nodes with coefficient vector v. This should give

$$b(w, w) - v_k^T \mathbf{B}_k v_k = \mathcal{O}(h_k^p).$$

9 2nd-Order Linear Evolution Problems

Computational domain becomes $\tilde{\Omega} := \Omega \times (0, T) \subset \mathbb{R}^{d+1}$

9.1 Parabolic Initial-Boundary Value Problems (IVBP)

Heat equation

$$\partial_t \int_V \rho u \, \mathrm{d}\mathbf{x} + \int_V \mathbf{j} \cdot \mathbf{n} = \int_V f \, \mathrm{d}\mathbf{x} \quad \forall \text{ control volumes } V$$

where ρ models the energy stored in V.

Boundary conditions Use the same spatial boundary conditions. For temporal boundary condition, the only option is to specify the initial value $u(x, 0) = u_0(x)$

IBVP

$$\begin{aligned} \partial_t(\rho u) - \operatorname{div}(\kappa \operatorname{\mathbf{grad}} u) &= f && \text{in } \tilde{\Omega} := \Omega \times (0, T) \\ u(\mathbf{x}, t) &= g(\mathbf{x}, t) && \text{for } (\mathbf{x}, t) \in \partial\Omega \times (0, T) \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) && \text{for all } \mathbf{x} \in \Omega \end{aligned}$$

Spatial variational formulation for $t \in (0, T) \mapsto u(t) \in V_0$

$$\begin{aligned} \partial_t m(u(t), v) + a(u(t), v) = l(t)(v) \quad \forall v \in V_0 \\ u(0) = u_0 \in V_0 \end{aligned}$$

Theorem 9.1 (Decay of solution of parabolic evolutions) *For $f \equiv 0$, the solution $u(t)$ satisfies*

$$\|u(t)\|_m \le e^{-\gamma t} \|u_0\|_m, \qquad \|u(t)\|_a \le e^{-\gamma t} \|u_0\|_a, \quad \forall t \in (0, T).$$

Parabolic evolutions dissipate energy Without excitation, exponential decay of energy during parabolic evolution.

Spatial semi-discretization by method of lines (MOL) Replace V_0 with finite dimensional $V_{0,N}$ and introduce basis $\mathcal{B} = \{b^1_N, \dots, b^N_N\} \subset V_0$ independent of time. The time dependent solution is given by $u_N(t) = \sum_{i=1}^N \mu_i(t) b^i_N$ with time dependent coefficients $\mu_i : [0, T] \to \mathbb{R}$.

Method of lines ODE

$$\begin{aligned} \mathbf{M} \partial_t \mu(t) + \mathbf{A} \mu(t) = \varphi(t) \quad \text{for } 0 < t < T, \\ \mu(0) = \mu_0 \end{aligned}$$

ODE Find $u : I \subset \mathbb{R} \to \mathbb{R}^N$ such that $\dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t))$ with $\mathbf{f} : I \times \mathbb{R}^N \to \mathbb{R}^N$ Lipschitz continuous.

Evolution operator

$$\Phi : I \times I \times \mathbb{R}^N \to \mathbb{R}^N : (t_0, t, \mathbf{u}_0) \mapsto \Phi^{t,t}\mathbf{u}_0 := \Phi(t_0, t, \mathbf{u}_0) := \mathbf{u}(t) \quad \text{where } \dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}), \quad \mathbf{u}(t_0) = \mathbf{u}_0$$

Properties of evolution operator

- For fixed $t_0 \in I$ and $\mathbf{u}_0 \in \mathbb{R}^N$ the so-called trajectory $t \mapsto \Phi$ supplies the solution for the initial value problem.
- For fixed $s, t \in I$ the mapping $\mathbf{u} \mapsto \Phi^{s,t}\mathbf{u}$ is bijective and in terms of self-mappings of $\mathbb{R}^N$ we have $\Phi^{t,t} = Id|_{\mathbb{R}^N}$ for all $t \in I$ and $\Phi^{s,r} \circ \Phi^{r,t} = \Phi^{s,t}$ for all $s, r, t \in I$.
- Recover ODE by $\partial_t \Phi(t_0, t, u) = f(t, \Phi(t_0, t, u))$.

Definition 9.2 (Discrete evolution operator) *belonging to an ODE on $I \times \mathbb{R}^N$ ia a mapping $\Psi : I \times I \times \mathbb{R}^N \to \mathbb{R}^N$ such that for all $t \in I$ and $\boldsymbol{u} \in \mathbb{R}^N$*

$$\exists q \in \mathbb{N}_0, \tau_0 > 0 : \ \|\Psi^{t,t+\tau}\boldsymbol{u} - \Phi^{t,t+\tau}\boldsymbol{u}\| \le C(t, \boldsymbol{u})\tau^{q+1} \quad \forall |\tau| < \tau_0,$$

*with $C = C(t, \boldsymbol{u})$ depending on t and $\boldsymbol{u}$ continuously. The largest possible q is called the **order** of the discrete evolution.*

Simple single step methods (SSM)

- **Explicit Euler** $\Psi^{t,t+\tau}\mathbf{u} = \mathbf{u} + \tau\mathbf{f}(t, \mathbf{u})$ of order 1
- **Implicit Euler** $\Psi^{t,t+\tau}\mathbf{u} = \mathbf{u} + \tau\mathbf{f}(t, \Psi^{t,t+\tau}\mathbf{u})$ of order 1
- **Implicit Midpoint** $\Psi^{t,t+\tau}\mathbf{u} = \mathbf{w}$, $\mathbf{w} = \mathbf{u} + \tau\mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{w} + \mathbf{u}))$ of order 2

Theorem 9.3 (Order $q \implies$ rate q) *Let $\boldsymbol{u}^{(j)} \in \mathbb{R}^N$, $j = 1, \dots, M$ be a sequence of pointwise approximations of the solution of the initial value problem $\dot{\boldsymbol{u}}(t) = \boldsymbol{f}(t, \boldsymbol{u}(t))$ for all $t \in I$ and $\boldsymbol{u}(t_0) = \boldsymbol{u}_0$ on a time interval $[t_0, T]$ with fixed final time $T > t_0$ generated by a single step method of order $q \in \mathbb{N}$ on a temporal mesh $t_0 < t_1 < \ldots < t_M = T$. If $\boldsymbol{f} \in C^{q+1}(I \times \mathbb{R}^N)$, then*

$$\max_{j=1,\dots,M} \|\boldsymbol{u}^{(j)} - \boldsymbol{u}(t_j)\| \le C\tau^q$$

for $\tau = \max_{j=1,\dots,M} |t_j - t_{j-1}|$ with $C > 0$ independent of t_j.

Collocation SSM provide a general recipe for building discrete evolutions. The idea is to use polynomial approximation on $[t, t+\tau]$ such that the ODE is solved at isolated points.

Theorem 9.4 (Order of collocation SSN) *Provided that $\boldsymbol{f}] \in C^p(I \times \mathbb{R}^N)$, the order of an s-stage collocation single step method agrees with the order of the quadrature on $[0, 1]$ with nodes c_j and weights b_j for $j = 1, \dots, s$.*

Definition 9.5 (General Runge-Kutta method) *For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i = \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ 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*

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

$i = 1, \dots, s$. The $\boldsymbol{k}_i \in V_0$ are called increments.

Notation for RK-SSM $\mathcal{U}_{ij} = a_{ij}$

$$\begin{array}{c|c} \mathbf{c} & \mathcal{U} \\ \hline & \mathbf{b}^T \end{array}$$

Time stepping for MOL ODE We set $t_j := j\tau$

- **Explicit Euler** $\boldsymbol{\mu}^{(j)} = \boldsymbol{\mu}^{(j-1)} + \tau\mathbf{M}^{-1}(\varphi(t_{j-1}) - \mathbf{A}\boldsymbol{\mu}^{(j-1)})$
- **Implicit Euler** $\boldsymbol{\mu}^{(j)} = (\mathbf{M} + \tau\mathbf{A})^{-1}(\mathbf{M}\boldsymbol{\mu}^{(j-1)} + \tau\varphi(t_j))$
- **General RK-SSM** $\boldsymbol{\mu}^{(j+1)} = \boldsymbol{\mu}^{(j)} + \tau\sum_{m=1}^s \boldsymbol{\kappa}_m b_m$ with coefficient vector $\boldsymbol{\kappa}_i \in \mathbb{R}^N$ satisfying the linear system of equations

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

$$\iff (\mathbf{I}_s \otimes \mathbf{M} + \tau\mathcal{U} \otimes \mathbf{A}) \begin{pmatrix} \boldsymbol{\kappa}_1 \\ \vdots \\ \boldsymbol{\kappa}_s \end{pmatrix} = \begin{pmatrix} \varphi(t_j + c_1\tau) - \mathbf{A}\boldsymbol{\mu}^{(j)} \\ \vdots \\ \varphi(t_j + c_s\tau) - \mathbf{A}\boldsymbol{\mu}^{(j)} \end{pmatrix}.$$

Stability analysis based on diagonalization Diagonalization of MOL-ODE leads to equations of the form $\dot{\eta}_i(t) = -\lambda_i\eta_i(t)$ with eigenvectors $\lambda_i > 0$

Stability

- **Explicit Euler** is stable for $|1 - \tau\lambda_i| < 1 \iff \tau \le \frac{2}{\lambda_i}$
- **Implicit Euler** is unconditionally stable $\lim_{n\to\infty} \eta_i^{(n)} = 0$, i.e. for all $\tau > 0$. The (theoretical) constraints are $|\frac{1}{1+\tau\lambda_i}| < 1$ and $\lambda_i > 0$.
- **RK-SSM** are stable for $|S(z)| < 1$ for all $z < 0$, where S is the stability function with $\mathbf{1} = (1, \dots, 1)^T \in \mathbb{R}^s$

$$S(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathcal{U})^{-1}\mathbf{1} = \frac{\det(\mathbf{I} - z\mathcal{U} + z\mathbf{1}\mathbf{B}^T)}{\det(\mathbf{I} - z\mathcal{U})}.$$

Unconditional stability of SSM The discrete evolution $\Psi_\lambda^\tau : \mathbb{R} \to \mathbb{R}$ of the single step method applied to the scalar ODE $\dot{u} = -\lambda u$ must satisfy

$$\lambda > 0 \implies \lim_{n\to\infty} (\Psi_\lambda^\tau)^n u_0 = 0 \quad \forall u_0 \in \mathbb{R}, \forall \tau > 0. \tag{9.1}$$

Theorem 9.6 (Behavior of generalized eigenvalues) *Let $\mathcal{M}$ be a simplicial mesh and $\boldsymbol{A}, \boldsymbol{M}$ denote the Galerkin matrices for the bilinear forms $a(u, v) = \int_\Omega \nabla u \nabla v \, \mathrm{d}\boldsymbol{x}$ and $m(u, v) = \int_\Omega uv \, \mathrm{d}\boldsymbol{x}$, respectively, and $V_{0,N} := \mathcal{S}^0_{o,0}(\mathcal{M})$. Then the smallest and largest generalized eigenvectors $\boldsymbol{A}\mu = \lambda\boldsymbol{M}\mu$, denoted by λ_{min} and λ_{max} satisfy*

$$\frac{1}{\mathrm{diam}(\Omega)} \le \lambda_{min} \le C \quad , \quad Ch_{\mathcal{M}}^{-2} \le \lambda_{max},$$

where the "generic constants" depend only on the polynomial degree p, the domain Ω and the shape regularity measure $\rho_{\mathcal{M}}$.

Definition 9.7 ($L(\pi)$-stability) *A Runge-Kutta single-step method satisfying 9.1 is called $L(\pi)$-stable if its stability function $S(z)$ satisfies*

(i) $|S(z)| < 1$ for all $z < 0$, and

(ii) "$S(-\infty)$" $:= \lim_{z\in\mathbb{R}\to\infty} S(z) = 0$.

$L(\pi)$-stable 2-stage RK-SSM

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

RADAU-3 scheme (order 3)

$$\begin{array}{c|cc} \lambda & \lambda & 0 \\ 1 & 1-\lambda & \lambda \\ \hline & 1-\lambda & \lambda \end{array} \quad \lambda = 1 - \frac{1}{2}\sqrt{2}$$

SDIRK-2 scheme (order 2)

Meta-Theorem 9.1 (Total err = spatial err + temporal err) *Assume that*

- *the solution of the parabolic IBVP is "sufficiently smooth" (both in space and time),*
- *its spacial Galerkin finite element discretization relies on the degree p Lagrangian finite elements on uniformly shape-regular families of meshes,*

- *timestepping is baed on an $L(\pi)$-stable single step method of order q with uniform timestep $\tau > 0$.*

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

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

where $C < 0$ must not depend on $h_{\mathcal{M}}$ and τ.

Reduction of error Assuming sharp estimates, spatial and temporal resolution have to be adjusted in tandem to archive an error reduction.

Potential inefficiency of conditionally stable SSM In order to reduce the error by a fixed factor ρ,

- accuracy requires reduction of τ by a factor $\rho^{1/q}$
- stability requires reduction of τ by a factor $(\rho^{1/p})^2 = \rho^{2/p}$.

For $\frac{1}{q} < \frac{2}{p} \iff 2q > p$, stability enforces smaller timestep than required by accuracy and renders timestepping inefficient.

9.2 Linear Wave Equation

Homogeneous linear wave equation in spatial variational form

$$u \in V(t): \quad \int_\Omega \rho\, \partial_t^2 u\, v \,\mathrm{d}\mathbf{x} + \int_\Omega \sigma \nabla u \nabla v \,\mathrm{d}\mathbf{x} = 0 \quad \forall v \in H_0^1(\Omega)$$
$$u \in V(t): \quad m(\ddot{u}, v) + a(u, v) = 0 \quad \forall v \in H_0^1(\Omega),$$

where $V(t) := \{v : (0,T) \to H^1(\Omega) : v(\mathbf{x}, t) = g(\mathbf{x}, t) \text{ for } \mathbf{x} \in \partial\Omega, 0 < t < T\}$.

Rewriting 2nd order ODEs $\ddot{w} = g(t, w)$ by introducing new function v as $\dot{\mathbf{u}} = \begin{pmatrix} \dot{w} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ g(t,w) \end{pmatrix} = \mathbf{f}(t, \mathbf{u})$.

Initial conditions Boundary conditions are nothing new. For initial conditions we need $u(\mathbf{x}, 0) = u_0(\mathbf{x})$ and $\dot{u}(\mathbf{x}, 0) = v_0(\mathbf{x})$ for $\mathbf{x} \in \Omega$.

d'Alembert solution for the Cauchy problem $\partial_t^2 u = c^2 \partial_x^2 u$ on $\mathbb{R} \times [0, 1]$ is given by $u(x,t) = \frac{1}{2}(u_0(x+ct) + u_0(x-ct)) + \frac{1}{2c}\int_{x-ct}^{x+ct} v_0(s)\,\mathrm{d}s$.

Theorem 9.8 (Domain of dependence for isotropic wave eqn) *Let $\tilde{u} : \tilde{\Omega} \to \mathbb{R}$ be a classical solution of $\partial_t^2 = c\Delta u$. Then*

$$(|\boldsymbol{x} - \boldsymbol{x}_0| \geq \mathbb{R} \implies u(\boldsymbol{x}, 0) = 0, \partial_t u(\boldsymbol{x}, 0) = 0) \implies u(\boldsymbol{x}, t) = 0,$$

if $|\boldsymbol{x} - \boldsymbol{x}_0| \geq R + ct$.

Discretized energies

- **Kinetic energy** $\frac{1}{2}m(\dot{u}, \dot{u}) = \frac{1}{2}\dot{\mu}^T M \dot{\mu}$
- **Elastic energy** $\frac{1}{2}a(u,u) = \frac{1}{2}\mu^T A \mu$
- **Potential energy** $\frac{1}{2}\overline{\mu}^T A \overline{\mu}$?? TODO

Theorem 9.9 (Energy conservation in wave propagation) *If $u : \tilde{\Omega} \to \mathbb{R}$ solves $m(\ddot{u}, v) + a(u, v) = 0$, then we have conservation of energy in the sense that*

$$t \mapsto \frac{1}{2}m(\dot{u}, \dot{u}) + \frac{1}{2}a(u,u) \equiv \textit{const.}$$

Method of lines ODE

$$\mathbf{M}\partial_t^2 \mu(t) + \mathbf{A}\mu(t) = \varphi(t) \quad \text{for } 0 < t < T,$$
$$\begin{pmatrix} \dot{\mu}(t) \\ \mathbf{M}\dot{\nu}(t) \end{pmatrix} = \begin{pmatrix} \nu(t) \\ -\mathbf{A}\mu(t) \end{pmatrix} \quad \text{with } \mu(0) = \mu_0, \nu(0) = \nu_0.$$

Discrete conservation of energy

$$(\mu^{(t)})^T \mathbf{A} \mu^{(t)} + (\nu^{(t)})^T \mathbf{M} \nu^{(t)} = (\mu^{(t-1)})^T \mathbf{A} \mu^{(t-1)} + (\nu^{(t-1)})^T \mathbf{M} \nu^{(t-1)}$$

Conservation of energy for SSM Explicit and implicit Euler do not conserve energy, implicit midpoint rule exhibits perfect energy conservation. Leapfrog method does almost conserve energy.

Crank-Nicolson timestepping is the energy conversing 2nd-order method for the problem $\mathbf{M}\partial_t^T \mu + \mathbf{A}\mu = \varphi$

$$\mathbf{M}\frac{\mu^{(j+1)} - 2\mu^{(j)} + \mu^{(j-1)}}{\tau^2} = -\frac{1}{2}\mathbf{A}(\mu^{(j-1)}\mu^{(j+1)}) + \frac{1}{2}(\varphi(t - \frac{1}{2}\tau) + \varphi(t + \frac{1}{2}\tau))$$

for $j = 1, 2, \ldots$.

Störmer-Verlet timestepping is the 2nd-order method for the homogeneous problem $\mathbf{M}\partial_t^T \mu + \mathbf{A}\mu = 0$

$$\mathbf{M}\frac{\mu^{(j+1)} - 2\mu^{(j)} + \mu^{(j-1)}}{\tau^2} = -\mathbf{A}\mu^{(j)}$$

for $j = 1, 2, \ldots$ and with special initial step $\frac{\mu^{(1)} - \mu^{(-1)}}{2\tau} = \nu_0$.

Leapfrog timestepping is the first-order form of the Störmer scheme

$$\mathbf{M}\frac{\nu^{(j+\frac{1}{2})} - \nu^{(j-\frac{1}{2})}}{\tau} = -\mathbf{A}\mu^{(j)}$$
$$\frac{\mu^{(j+1)} - \mu^{(j)}}{\tau} = \nu^{(j+\frac{1}{2})}$$

for $j = 0, 1, \ldots$ and with initial step $\nu^{(-\frac{1}{2})} + \nu^{(\frac{1}{2})} = 2\nu_0$.

Mass lumping replaces $\mathbf{M}$ with a diagonal matrix. For linear Lagrangian FE by using vertex-based quadrature formula $\int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{|K|}{\#\mathcal{V}(K)} \sum_{\mathbf{p} \in \mathcal{V}(K)} f(\mathbf{p})$.

Stability, CFL-condition Crank-Nicolson timestepping is unconditionally stable, Leapfrog timestepping is only conditionally stable with condition $\tau \leq \frac{2}{\sqrt{\lambda_i}}$, which because $\max_i \lambda_i = \mathcal{O}(h_{\mathcal{M}}^{-2})$ implies the CLF condition

$$\tau \leq C\, h_{\mathcal{M}}. \qquad \text{(CLF-condition)}$$

Leapfrog timestep constraint is okay The leapfrog timestep constraint $\tau \leq \mathcal{O}(h_{\mathcal{M}})$ does not compromise (asymptotic) efficiency, if $p \geq 2$, where p is the degree of spatial Lagrangian finite elements.

CLF and domain of dependence CLF-condition $\iff$ analytical domain of dependence $\subset$ numerical domain of dependence

Leapfrog vs. Crank-Nicolson Given the possibility of mass lumping, the computational effort for leapfrog timestepping becomes substantially smaller than that for the Crank-Nicolson scheme, because the latter will always involve the solution of a $N \times N$ sparse linear system of equations.

10 Convection-Diffusion Problems

10.1 Heat Conduction in a Fluid

Flow field $\mathbf{v} : \Omega \to \mathbb{R}^d$ on bounded domain $\Omega \subset \mathbb{R}^d$.

Assumptions We assume that $\mathbf{v} \in (C^0(\overline{\Omega}))^d$ and no inflow or outflow, i.e. $\mathbf{v}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) = 0$ for all $\mathbf{x} \in \Omega$

Flow map is the evolution operator $\Phi : \mathbb{R} \times \Omega \to \Omega : (t, \mathbf{x}_0) \mapsto y(t)$ for the flow equation $\dot{\mathbf{y}} = \mathbf{v}(\mathbf{y})$. Notation is $\Phi^t \mathbf{x} = \Phi(t, \mathbf{x}) = \Phi^{0,t}\mathbf{x}$.

Fourier's law in moving fluids

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

Convection diffusion equation Together with $\operatorname{div}\mathbf{j} = f$ this corresponds to the linear scalar convection diffusion equation

$$\underbrace{-\operatorname{div}(\kappa\,\mathbf{grad}\, u)}_{\text{2nd order diffusive term}} + \underbrace{\operatorname{div}(\rho \mathbf{v} u)}_{\text{1st order convective term}} = f$$

Incompressible fluids preserve the volumes, i.e. $vol(\Phi^t(V)) = vol(V)$ for all t and $V \subset \Omega$.

Theorem 10.1 (Div-free velocity fields are incompressible) *A stationary fluid flow in Ω is incompressible if and only if its associated velocity field $\boldsymbol{v}$ satisfies* $\operatorname{div} \boldsymbol{v} = 0$ *everywhere in Ω.*

Convection diffusion equation for incompressible fluids simplifies to

$$-\kappa\Delta u + \rho \mathbf{v}\,\mathbf{grad}\, u = f \quad \text{in } \Omega.$$

Time-dependent convection diffusion equation and incompressible fluids

$$\partial_t(\rho u) - \operatorname{div}(\kappa\,\mathbf{grad}\, u) + \operatorname{div}(\rho \mathbf{v} u) = f \quad \text{in } \tilde{\Omega} = \Omega \times [0,T].$$

10.2 Stationary Convection-Diffusion Problem

Model problem

$$-\varepsilon\Delta u + \mathbf{v}\,\mathbf{grad}\, u = f \quad \text{in } \Omega.$$

Bilinear form

$$a(u,w) = \varepsilon \int_\Omega \nabla u \nabla w \,\mathrm{d}\mathbf{x} + \int_\Omega v \nabla u\, w \,\mathrm{d}\mathbf{x}$$

is not symmetric, but well defined on $H_0^1(\Omega)$ and positive definite, because $a(w,w) = \varepsilon \|\nabla w\|^2_{L^2(\Omega)} > 0$ for $w \neq 0$.

Singular perturbed problem is a boundary value problem depending on parameter $\varepsilon \approx \varepsilon_0$ for which the limit of the problem for $\varepsilon \to \varepsilon_0$ is not compatible with the boundary conditions.

Limiting case $\varepsilon = 0$ (i.e. the problem $\mathbf{v}\,\mathbf{grad}\, u = f$) is solved by

$$u(\mathbf{y}(t)) = u(\mathbf{y}(0)) + \int_0^t f(\mathbf{y}(s), t)\,\mathrm{d}s.$$

We choose $\mathbf{y}(0)$ on $\partial\Omega$ where u is known.

Problem for $\varepsilon = 0$ For the pure transport problem $\mathbf{v}\,\mathbf{grad}\, u = f$ Dirichlet boundary conditions can be imposed only on the inflow boundary Γ_{in}, but not on its complement in $\partial\Omega$, the outflow boundary, given by

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

ε-robust numerical methods are those satisfying the discrete maximum principle.

Sign conditions for maximum principle

- positive diagonal entries $\mathbf{A}_{ii} > 0$,
- non-positive off-diagonal entries $\mathbf{A}_{ij} \leq 0$ for $i \neq j$
- diagonal dominance $\sum_j \mathbf{A}_{ij} \geq 0$

Flow information The direction of the flow field $\mathbf{v} : \Omega \to \mathbb{R}^n$ determines the direction of flow of information.

Even/odd decoupling For $\varepsilon \to 0$, the discretization becomes $\mu_{j+1} - \mu_{j-1} = hf(hj)$. Even/odd decoupling leads to a singular system matrix.

Forward vs. backward difference quotients

- Linear system arising from use of backward difference quotient (implicit Euler) results in good solutions, because sign conditions are always fulfilled.
- Linear system arising from use of forward difference quotient (explicit Euler) results in bad solutions, because of strong restrictions ($\frac{2\varepsilon}{h} < 1$) ensuring the sign conditions hold.

Upwind quadrature uses upwind information to evaluate $\partial_x u_N(jh)$ as

$$\partial_x u(jh) := \lim_{\delta \to 0} \partial_x u_N(jh - \delta)$$
$$\mathbf{v}(\mathbf{p})\,\mathbf{grad}\, u_N(\mathbf{p}) := \lim_{\delta \to 0} \mathbf{v}(\mathbf{p})\,\mathbf{grad}\, u_N(\mathbf{p} - \delta\mathbf{v}(\mathbf{p}))$$

1D upwind contribution $\int_0^1 \sum_{l=1}^{M-1} \mu_l \partial_x b_N^l b_N^i \,\mathrm{d}x = h\frac{\mu_i - \mu_{i-1}}{h}$

Upwind quadrature fulfills sign conditions and thus respects the maximum principle.

Upwind quadrature is 1st-order $\|u - u_N^{up}\|_{L^2(\Omega)} = \mathcal{O}(h)$

Artificial diffusion is upwinding by h-dependent strengthening of diffusion

$$(\varepsilon + h/2)\frac{-\mu_{i-1} + 2\mu_i - \mu_{i+1}}{h^2} + \frac{-\mu_{i-1} + \mu_{i+1}}{2h} = f(ih)$$

Smearing of internal layers happens if too much artificial diffusion has been added to the problem. This might lead to a perturbed solution.

Heuristics of streamline upwinding Since the solution is smooth along streamlines, then adding diffusion in the direction of streamlines cannot do much harm.

Anisotropic artificial diffusion

$$\varepsilon \leftarrow \varepsilon\mathbf{I} + \delta_K \mathbf{v}_K \mathbf{v}_K^T \in \mathbb{R}^{2,2},$$

where $\mathbf{v}_K$ is the local velocity and δ_K is a method parameter controlling the strength of anisotropic diffusion.

Anisotropic variational problem Seek $u \in H^1(\Omega)$ such that

$$\int_\Omega \varepsilon\nabla u\nabla w + (\mathbf{v}\nabla u)w \,\mathrm{d}\mathbf{x} + \sum_{K\in\mathcal{M}} \delta_K \int_K (-\varepsilon\Delta u + \mathbf{v}\nabla u - f)(\mathbf{v}\nabla w)\,\mathrm{d}\mathbf{x} = \int_\Omega fw\,\mathrm{d}\mathbf{x}$$

for all $w \in H_0^1(\Omega)$.

Choice of δ_K $\delta_K = \begin{cases} \varepsilon^{-1/2}h_K^2 & \frac{\|\mathbf{v}\|_{K,\infty}h_K}{2\varepsilon} \leq 1, \\ h_K & \frac{\mathbf{v}_{K,\infty}h_K}{2\varepsilon} > 1. \end{cases}$

Streamline and maximum principle Streamline diffusion does not strictly respect the maximum principle

Streamline diffusion is 2nd-order $\|u - u_N^{up}\|_{L^2(\Omega)} = \mathcal{O}(h^2)$

Meta-Theorem 10.1 *No ε-robust linear method that respects the maximum principle strictly can be more than 1st-order convergent in L^2.*

10.3 Transient Convection-Diffusion IBVP

Transient convection-diffusion problem

$$\partial_t(\rho u) + \operatorname{div}(-\kappa\nabla u + \rho \mathbf{v} u) = f(\mathbf{x},t) \quad \text{in } \tilde{\Omega}$$

with boundary and initial conditions, for example $u(\mathbf{x},t) = g(\mathbf{x},t)$ for $\mathbf{x} \in \partial\Omega$ and $u(\mathbf{x},0) = u_0(\mathbf{x})$ for $\mathbf{x} \in \Omega$.

Fluid velocity becomes $\mathbf{v} : \tilde{\Omega} \to \mathbb{R}^d : \mathbf{v} = \mathbf{v}(\mathbf{x},t)$.

Incompressible transient convection-diffusion

$$\partial_t u - \varepsilon\Delta u + \mathbf{v}(\mathbf{x},t)\nabla u = f \quad \text{in } \tilde{\Omega}$$

MOL approach

$$\mathbf{M}\dot{\mu} + \varepsilon\mathbf{A}\mu + \mathbf{B}(t)\mu = \varphi(t)$$

Spacial discretization for method of lines approach to singularly perturbed transient convection-diffusion IBVPs use ε-robustly stable spatial discretization of convective term. The prize one has to pay are spurious damping or smearing due to artificial diffusion.

Temporal discretization use $L(\pi)$-stable timestepping methods. The prize is even more artificial diffusion.

Pure transport equation is the singular perturbed ($\varepsilon \to 0$) problem

$$\partial_t u + \mathbf{v}(\mathbf{x},t)\nabla u = 0 \quad \text{in } \tilde{\Omega}$$

Solution for pure transport equation

$$u(\mathbf{x},t) = \begin{cases} u_0(\mathbf{x}_0) + \int_0^t f(\mathbf{y}(s),s)\,\mathrm{d}s & \mathbf{y}(s) \in \Omega \ \forall 0 < s < t \\ g(\mathbf{y}(s_0),s_0) + \int_{s_0}^t f(\mathbf{y}(s),s)\,\mathrm{d}s & \mathbf{y}(s_0) \in \partial\Omega, \mathbf{y}(s) \in \Omega \end{cases}$$

$\forall s_0 < s < t$ for $(\mathbf{x},t) \in \tilde{\Omega}$ with $u(\mathbf{x},t) = g(\mathbf{x},t)$ on Γ_{in}.

Lagrangian split-step method separates the convection-diffusion equation into pure transport and pure diffusion and solves the problems separately.

$$\partial_t u = \varepsilon\Delta u + f - \mathbf{v}\nabla u \iff \dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}) + \mathbf{h}(\mathbf{y})$$

where $\mathbf{g}(\mathbf{y}) = \varepsilon\Delta u(\mathbf{y})$, $\mathbf{h}(\mathbf{y}) = f(\mathbf{y}) - \mathbf{v}(\mathbf{y})\nabla u(\mathbf{y})$

Strang splitting is a single-step method that computes $\mathbf{y}^{(j)} \approx \mathbf{y}(t_j)$ from $\mathbf{y}^{(j-1)} \approx \mathbf{y}(t_{j-1})$ according to

(i) $\tilde{\mathbf{y}} := \mathbf{z}(t_{j-1} + \frac{1}{2}\tau)$, where $\mathbf{z}(t)$ solves $\dot{\mathbf{z}} = \mathbf{g}(t,\mathbf{z})$, $\mathbf{z}(t_{j-1}) = \mathbf{y}^{(j-1)}$,

(ii) $\hat{\mathbf{y}} := \mathbf{w}(t_j)$, where $\mathbf{w}(t)$ solves $\dot{\mathbf{w}} = \mathbf{h}(t,\mathbf{w})$, $\mathbf{w}(t_{j-1}) = \tilde{\mathbf{y}}$,

(iii) $\mathbf{y}^{(j)} := \mathbf{z}(t_j)$, where $\mathbf{z}(t)$ solves $\dot{\mathbf{z}} = \mathbf{g}(t,\mathbf{z})$, $\mathbf{z}(t_{j-1} + \frac{1}{2}\tau) = \hat{\mathbf{y}}$.

Theorem 10.2 (Strang splitting is 2nd-order) *Assuming exact solutions of the initial value problems of the sub-steps, the Strang splitting single step method is of second order.*

Particle method for pure transport

(i) Pick suitable interpolation nodes $\mathbf{p}_1, \dots, \mathbf{p}_N \in \Omega$

(ii) Particle pushing: solve N initial value problems $\dot{\mathbf{y}} = \mathbf{v}(\mathbf{y}(t),t)$ with $\mathbf{y}(0) = \mathbf{p}_i$ for particle trajectories by means of a suitable single-step method with uniform timestep $\tau = T/M$, $M \in \mathbb{N}$ to create a sequence of solution points $\mathbf{p}_i^{(j)}$ for $j = 1,\dots,M$, $i = 1,\dots,N$.

(iii) Reconstruct approximation $u_N^{(j)} \approx u(\cdot,t_j)$, $t_j = j\tau$ by interpolation. We demand for $i = 1,\dots,N$

$$u_N^{(i)}(\mathbf{p}_i^{(j)}) = u_0(\mathbf{p}_i) + \tau \sum_{l=1}^{j-1} f\left(\frac{\mathbf{p}_i^{(l)} + \mathbf{p}_i^{(l-1)}}{2}, \frac{t_l + t_{l-1}}{2}\right)$$

Particle mesh method (PMM)

(i) MOL half step

(ii) Particle method described above

(iii) Remeshing: $\mathcal{M}_j$ with nodes at new particle positions

(iv) Repeat first step on $\mathcal{M}_j$ with particle temperatures defining initial finite element function

Problems with PMM Bas meshes, remesing is expensive, reassembly of $\mathbf{A}$ and $\mathbf{M}$ necessary, but as a benefit no artificial diffusion.

Material derivative

$$\begin{aligned}\frac{\mathrm{D}f}{\mathrm{D}u}(x,t) &= \lim_{\tau\to 0} \frac{f(x,t) - f(\mathbf{y}(t-\tau),t-\tau)}{\tau} \\ &= \nabla f(x,t)\mathbf{v}(x,t) + \partial_t(x,t)\end{aligned}$$

Convection diffusion eqn with material derivative becomes

$$\frac{\mathrm{D}f}{\mathrm{D}u} - \varepsilon\Delta u = f \qquad \text{in } \tilde{\Omega}$$

Semi-Lagrangian method discretized the material derivative $\frac{\mathrm{D}f}{\mathrm{D}u}$. The resulting problem becomes seek $u_N^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M})$

$$\int_\Omega \frac{u_N^{(j)}(\mathbf{x}) - u_N^{(j-1)}(\Phi^{t,t_j-\tau}\mathbf{x})}{\tau} v_N(\mathbf{x})\,\mathrm{d}\mathbf{x} + \varepsilon \int_\Omega \nabla u_N^{(j)} \nabla v_N\,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x},t_j) v_N(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

for all $v_N \in \mathcal{S}_{1,0}^0(\mathcal{M})$.

Approximations of Φ Replace $\mathbf{x} \mapsto u_N^{(j-1)}(\Phi^{t_j,t_j-\tau\mathbf{x}})$ with its p.w. linear interpolant and use explicit Euler to approximate $\Phi^{t_j,t_j-\tau} \approx \mathbf{x} - \mathbf{v}(\mathbf{x},t)\tau$ (**streamline backtracking**).

$$\int_\Omega \frac{u_N^{(j)}(\mathbf{x}) - I_1(u_N^{(j-1)}(\cdot - \tau\mathbf{v}(\cdot,t_j)))(\mathbf{x})}{\tau} v_N(\mathbf{x})\,\mathrm{d}\mathbf{x} + \varepsilon \int_\Omega \nabla u_N^{(j)} \nabla v_N\,\mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x},t_j) v_N(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

Part III

Implementation

11 Mesh

11.1 Initialization

Types

```
using grid_t = volume2dGrid::hybrid::Grid;
using eth::grid::GridViewTypes view = eth::grid::GridViewTypes::LeafView;
using gridView_t = typename eth::grid::GridView<grid_t::gridTraits_t::template viewTraits_t<view>>;
// the basic grid information is encoded in the grid traits
using gridTraits_t = gridView_t::gridTraits_t;
using gridCreator_t = GridCreator<grid_t,view>;
using gridFactory_t = gridCreator_t::gridFactory_t;
using itsct_t = eth::grid::Intersection<betl2:: volume2dGrid::hybrid::GridTraits>;
```

Initialize grid

```
const std::string basename = "./hex" + std::to_string(5);
betl2::input::gmsh::Input input(basename);
```

Create surface grid

```
const gridFactory_t gridFactory = gridCreator_t()(input);
const gridView_t gridView = gridFactory.getView();
```

11.2 Accessing geometric Information

Geometry object has the following types and methods

- Constant `dimFrom` telling the dimension of the reference element
- Constant `dimTo`, the dimension of the ambient space
- Type `gridTraits_t` of the GridTraits of the mesh of which the entity is part of
- Vector type `globalCoord_t` for the absolute coordinates of points in ambient space
- Vector type `localCoord_t` for relative coordinates in a reference element
- Integer type `size_type` for indices
- Method `size_type numCorners()` telling the number of vertices of the entity
- Method `globalCoord_t mapCorner(int i)` returning the global coordinates of the vertices of the geometric entity
- Method `gridTraits_t::ctype_t volume()` telling the volume/ area of the geometric entity
- Method `globalCoord_t center()` obtaining the global coordinates of the center of gravity of the geometric entity
- integration element?

Access to geometry object through `fespace` element by calling `element->geometry()`

Geometric entity object provides the following methods

- `refElType()` provides information about the geometric type of the mesh entity or, more precisely, about the underlying reference element. The following are available `POINT, SEGMENT, TRIA, QUAD, TETRA, HEXA, PRISM, PYRAMID` and can be accessed through `using triangle_t = eth::base::RefElType::TRIA;`.
- `geometry()` returns a (constant) reference to the geometric information attached to the entity.

For entities with codimension 0, the following methods are available in addition

- `int countSubEntities<codim>()` returns the number of entities of co-dimension `codim` contained in the boundary of the cell.
- `EntityPtr subEntity<codim>(int locidx)` returns a pointer to the entity with local number `locidx` and c-dimension `codim` contained in the boundary of the cell.

11.3 Numbering

Index set object `indexSetRef_t set(gv.indexSet())` provides the following methods

- `index_t index(const Entity &) const` returns the unique index (actually an integer) of any entity passed to it.
- `template <CODIM> index_t subIndex(const Entity<GRID_TRAITS,0> &element,size_type locidx)`, a shortcut to access indices of sub-entities with co-dimension 0 (equivalent to calling `set.index(T.template subEntity<codim>(locidx))`)

11.4 Looping through Grid

Loop over elements in FESpace

```
// alternative 1
// loop over all elements in fespace of codimension 0, i.e. cells
for (auto const element& : fespace) {
  // loop over local shape functions
  for (auto index : fespace.indices(element)) {}
}
// alternative 2
for (auto el_it = fespace.begin(); el_it != fespace.end(); ++el_it) {
  for (auto dof_it = fespace.begin(*el_it); dof_it != fespace.end(*el_it); ++dof_it)
```

```
    auto locIdx = fespace.localIndex (dof_it, *el_it);
    auto glbIdx = fespace.globalIndex(dof_it);
  }
}
// alternative 3
for (const auto element& : gridView.template entities<0>()) {}
```

wie könnte ich hier mit diesem index ein element abfragen?

11.5 Intersections

Intersection object provides the following methods

- `bool boundary()` if false, no other neighbor exists
- `bool neighbor()` true if a neighbor cell exists
- `geometry()` geometry of intersection object
- `inside()` return pointer to ”master element”
- `outside()` returns pointer to neighbor element; well defined value only if this exists
- `indexInInside()` local number of edge corresponding to intersection object in ”master element”
- `indexInOutside()` local number of edge corresponding to intersection object in neighboring element; well defined return value only if this exists

Basic loop over intersections

```
for (const auto& el : gridView.template entities<0>()) {
  for (const auto& inters: gridView.intersections(el)) {
    const auto& inter = *inters;
    const auto& geom = inter.geometry();
    const unsigned int locEdgeId = inter.indexInInside();
  }
}
```

Loop over all boundary edges

```
#include <iostream>
using itsct_t = eth::grid::Intersection<betl2::volume2dGrid:: hybrid::GridTraits>;
std::vector<const itsct_t*> boundary_inters;
for (const auto& el:gridView.template entities<0>()) {
  for (const auto& inters: gridView.intersections(el)) {
    if (inters.boundary())
      boundary_inters.push(&inters);
  }
}
for (const auto& inters: boundary_inters) {
  const auto& el = inters->inside(); // master element
  const auto& geom = inters->geometry();
}
```

12 Basis and Dof Handler

Definition of FEBasisType

```
// APPROX_ORDER = {Constant, Linear, Quadratic, Cubic}
// FEBasisType = Lagrange
template<int APPROX_ORDER,enum FEBasisType FE_TYPE> class FEBasis;
```

Dof handler

```
// fe::FESContinuity = fe::FESContinuity::Continuous
fe::DofHandler<fe::FEBasisType, fe::FESContinuity, eth::grids::utils::GridViewFactory>;
```

Example initialization

```
// define a constant linear finite element basis
using febasis_t = fe::FEBasis<fe::Linear, fe::FEBasisType::Lagrange>;
// define dofhandler type for linear basis functions
using dofHandler_t = fe::DofHandler<febasis_t, fe::FESContinuity::Continuous, gridFactory_t>;
// instantiate dofhandler for the grid
dofHandler_t  dh;
// distribute the degrees of freedom
dh.distributeDofs(gridFactory);
```

12.1 FESpace

FESpace provides the following member functions.

- `begin()` and `end()` return the constant iterator to the beginning and end of the container of cells, i.e. entities of codimension zero. This enables foreach loops over `fe::FESpaces`.

- `begin(e)` and `end(e)` take e, a constant reference to an entity of co-dimension zero (cell) and provide a constant iterator to the beginning and end of the vector of dofs for the element/cell e.
- `dofsOnElements()` returns a constant reference to the container of all dofs managed by the `fe::FESpace`.
- `globalIndex(dIter)` takes a constant dof iterator dIter and returns its global index.
- `localIndex(dIter,e)` takes a constant dof iterator dIter and a constant ref- erence to an entity e of codimension zero and returns the local index of the dof from dIter with respect to the cell e.
- `filter<CODIM>(e, intersectionIndex)` takes e, a constant reference to an entity of codimension zero (cell) and an intersectionIndex of the element (type int), referring to one of the elements intersections (sides). It returns a standard vector containing the local indices (w.r.t. the cell e) of all dofs that are associated with entities of codimension CODIM contained in the intersection corresponding to the intersectionIndex.
- `filterAll(e, intersectionIndex)` takes e, a constant reference to an entity of codimension zero (cell) and an intersectionIndex of the cell e (type int), referring to one of the elements intersections (sides). It returns a standard vector containing pointers to all dofs that are associated with the side corresponding to the intersectionIndex.
- `filterIndices(e, intersectionIndex)` takes e, a constant reference to an en- tity of codimension zero (cell) and an intersectionIndex of the cell e (type int), referring to one of the elements intersections (sides). It returns a standard vector containing the local in- dices (w.r.t. the cell e) of all dofs that are associated with the intersection corresponding to the intersectionIndex.
- `indices(e, intersectionIndex)` takes e, a constant reference to an entity of codimension zero (cell) and an intersectionIndex of the cell e (type int), referring to one of the elements intersections (sides). It returns a standard vector containing the local→global index mappings (w.r.t. the intersection associated with the intersectionIndex) of all dofs that are associated with the intersection corresponding to the intersectionIndex.
- `indices(e)` takes e, a constant reference to an entity of codimension zero (cell), and provides a standard vector filled with its local→global index mappings (as pairs of integer indices, see below).
- `numDofs()` returns the global number of dofs.
- `numElements()` returns the total number of elements.

FESpace elements? TODO that are being looped over also have some functions like calling geometry object, please explain here TODO

Basis functions and their gradients

```
typename FEBASIS::template basisFunction_t<RET>;
typename FEBASIS::template diffBasisFunction_t<RET>;
```

provide the following types and functions

- `static const eth::base::RefElType refElType` geometry type of element for which FEBasis was designed
- `static const int numFunction` number of local shape functions
- `static const int functionDim` number of vector components of return value
- `static const int localDim` dimension of ambient space for reference element
- `using matrix_t`
- `Eval( const matrix_t< localDim, NUM_POINTS > &)` returns the evaluation for the local shape functions and for diffBasisFunction_t returns the gradients of the local shape functions. It takes a matrix with point coordinates with respect to the reference element in its columns. The number of columns has to be passed as a template parameter. It returns a matrix with a rows for each individual local shape functions, with the result (vectors) of the evaluations in the passed points horizontally concatenated in each row.

Point evaluations

```
typedef fe::FEBasis< fe::Quadratic, fe::FEBasisType::Lagrange > qfebasis_t;
// get quadratic basis functions for reference triangle;
typedef typename qfebasis_t::template basisFunction_t< REtria> basisFuncts;
// evaluate them on quadrature points
const auto functEval  = basisFuncts::Eval( xti );
// get the basis functions' gradients for reference triangle;
typedef typename qfebasis_t::template diffBasisFunction_t< REtria> basisFunctGrads;
// evaluate them on quadrature points
const auto gradEval  = basisFunctGrads::Eval( xti );;
```

13 Local Computations and Assembly

Local stiffness matrix

```
const auto& geom = el.geometry();
auto elem_area = geom.volume();
result_t result;
// compute gradients
Eigen::MatrixXd grads(2,3);
grads << (geom.mapCorner(1) - geom.mapCorner(2)),
         (geom.mapCorner(2) - geom.mapCorner(0)),
         (geom.mapCorner(0) - geom.mapCorner(1));
// compute stiffness local matrix
result = grads.transpose()*grads/(4.*elem_area);
```

Local mass matrix

```
const auto& geom = el.geometry();
auto elem_area = geom.volume();
result_t result;
// compute mass local matrix
result.setConstant(elem_area/12.);
result += result.diagonal().asDiagonal();
```

NPDE local and global assemblers Global assemblers

- `NPDE::GalerkinMatrixAssembler` with function `assembleMatrix( const FESPACE_TEST_T& fe_test, const FESPACE_TRIAL_T& fe_tria`
- `NPDE::LoadVectorAssembler` with function `assembleRhs(const FESPACE_TEST_T&  fe_test, const BUILDER_DATA_T&  data )`
- `NPDE::IntersectionGalMatAsse`
- `NPDE::IntersectionLoadVectAsse`

Local triangular assemblers

- `NPDE::AnalyticStiffnessLocalAssembler` corresponds to local bilinear form $a(u,v) = \int_K \nabla u \nabla v \, \mathrm{d}\mathbf{x}$
- `NPDE::AnalyticMassLocalAssembler` corresponds to local bilinear form $a(u,v) = \int_K uv \, \mathrm{d}\mathbf{x}$
- `NPDE::LocalVectorAssembler` corresponds to local linear form $l(v) = \int_K fv \, \mathrm{d}\mathbf{x}$

NPDE stiffness assembly

```
typedef NPDE::AnalyticStiffnessLocalAssembler  localAssembler_t;
// type taking care of assembly of Galerkin matrix
typedef NPDE::GalerkinMatrixAssembler< localAssembler_t > assembler_t;
// instantiate corresponding object
assembler_t Ah;
// compute the Galerkin stiffness matrix in Eigen sparse matrix CRS format
typedef double numeric_t;
typedef Eigen::SparseMatrix< numeric_t > sparseMatrix_t;
const sparseMatrix_t& A = Ah.assembleMatrix( fespace, fespace, 1.0 );
```

14 Quadrature

Quadrature class `template<enum eth::base::RefElType RET, eth::base::signed_tNUM_POINTS> class Quadrature` provides the following member functions

- `getNumPoints()` return the number p of quadrature points (`NUM_POINTS`).
- `getRefEl()` returns the reference element type (`RET`).
- `getPoints()` returns an object of type `Eigen::Matrix<refElDim, NUM_POINTS>` containing the local coordinates of the quadrature nodes as columns, `refElDim` is the dimension of the reference element.
- `getWeights()` returns an object of type `Eigen::Matrix<1, NUM_POINTS>` containing the quadrature weights.
- `getScale()` returns correction scaling factor σ such that the sum of the quadrature weights multiplied with σ is equal to the are of the reference element, $|\hat{K}|$. In other words, only after rescaling with σ we get a valid quadrature formula on $\hat{K}$.
- `getRefDim()` returns the dimension of the reference element, for instance 2 in the case of TRIA or QUAD.

Quadrature for 2D-element types

```
typedef QuadRule<eth::base::RefElType::TRIA, NT> tria_t;
typedef QuadRule<eth::base::RefElType::QUAD, NQ> quad_t;
typedef QuadRuleList<tria_t, quad_t> quadrules_t;
```

Reference type

```
using refEl_t = eth::base::RefElType;
static const refEl_t triaType = eth::base::RefElType::TRIA;
static const refEl_t quadType = eth::base::RefElType::QUAD;
```

Basic idea of quadrature

```
result_t result;
//  Define quadrature for triangles
using quadrule_t = betl2::quad::Quadrature< eth::base::RefElType:TRIA, 7>;
// get points and weights over reference triangle
const auto & wti = quadrule_t::getWeights()*quadrule_t::getScale();
const auto & xti = quadrule_t::getPoints();
// map quadrature points and weights to current triangle
for( int l=0; l < xti.cols(); l++) {
  // evaluate function at current quadrature point
  const double f_eval = f( xti.col(l) );
  result += wti(l) * f_eval;
}
return result;
```

Basic idea of quadrature for transformed triangles

```
const auto& geom = el.geometry();
result_t result;
//  Define quadrature for triangles
using quadrule_t = betl2::quad::Quadrature< eth::base::RefElType:TRIA, 7>;
// get points and weights over reference triangle
const auto & wti = quadrule_t::getWeights()*quadrule_t::getScale();
const auto & xti = quadrule_t::getPoints();
// get determinant of Jacobian of 'reference->actual' element transformation
const auto detJi = geom.template integrationElement< quadrule_t::getNumPoints() >( xti );
// map quadrature points and weights to current triangle
const auto  globwti = detJi.cwiseProduct( wti );
const auto&  globxti = geom.global(xti);
for( int l=0; l < xti.cols(); l++) {
  // evaluate function at current quadrature point
  const double f_eval = f( globxti.col(l) );
  result += globwti(l) * f_eval;
}
return result;
```

15 Boundary Information

16 Solving system

SparseLU

```
// define the (direct) solver
typedef Eigen::SparseLU< Eigen::SparseMatrix<numeric_t> > solver_t;
// instantiate the solver
solver_t solver;
// initialize it and solve
solver.compute( Ah );
Eigen::VectorXd sol = solver.solve( rhs );
```

17 Working with Solution

17.1 Graphing & Interpolation

Interpolating solution

```
typedef InterpolationGridFunction< gridFactory_t, typename dofHandler_t::fespace_t, double> interpGF_t;
// instantiation of uN
interpGF_t uN(gridFactory, dh.fespace(), mu);
// Define vtu-exporter
typedef vtu::Exporter< gridFactory_t > exporter_t;
// Instantiate vtu-exporter with grid information and basename as the vtu-file name.
exporter_t exporter( gridFactory, basename );
// output the function uN under the name uh. We link uN to the grid points by chosing
// vtu::Entity::Point
exporter("uh", uN, vtu::Entity::Point);
```

Graphing TODO

17.2 Norms

$H1$-norm

```
template<typename FESPACE_T, typename VECTOR_T>
static double H1norm(FESPACE &fespace, VECTOR_T &mu) {
  GalerkinMatrixAssembler <NPDE::AnalyticStiffnessLocalAssembler> Assembler;
  Eigen::SparseMatrix<double> A = Assembler.assemble Matrix(fespace, fespace, 1.0);
  return sqrt(mu.transpose() * A * mu);
}
```

$L2$-norm

```
template<typename FESPACE_T, typename VECTOR_T>
static double L2norm(FESPACE &fespace, VECTOR_T &mu) {
  GalerkinMatrixAssembler<NPDE::AnalyticMassLocalAssembler> Assembler;
  Eigen::SparseMatrix<double> A = Assembler.assemble Matrix(fespace, fespace, 1.0);
  return sqrt(mu.transpose() * A * mu);
}
```

18 Example Codes

Initialize

```
// TYPES
using grid_t = volume2dGrid::hybrid::Grid;
using eth::grid::GridViewTypes view = eth::grid::GridViewTypes::LeafView;
```

```
using gridView_t = typename eth::grid::GridView<grid_t::gridTraits_t::template viewTraits_t<view>>;
// the basic grid information is encoded in the grid traits
using gridTraits_t = gridView_t::gridTraits_t;
using gridCreator_t = GridCreator<grid_t,view>;
using gridFactory_t = gridCreator_t::gridFactory_t;
using itsct_t = eth::grid::Intersection<betl2:: volume2dGrid::hybrid::GridTraits>;

// INITIALIZE GRID
const std::string basename = "./hex" + std::to_string(5);
betl2::input::gmsh::Input input(basename);

// CREATE SURFACE GRID
const gridFactory_t gridFactory = gridCreator_t()(input);
const gridView_t gridView = gridFactory.getView();

// DOF HANDLER
// define a constant linear finite element basis
using febasis_t = fe::FEBasis<fe::Linear, fe::FEBasisType::Lagrange>;
// define dofhandler type for linear basis functions
using dofHandler_t = fe::DofHandler<febasis_t, fe::FESContinuity::Continuous, gridFactory_t>;
// instantiate dofhandler for the grid
dofHandler_t  dh;
// distribute the degrees of freedom
dh.distributeDofs(gridFactory);
// solve Finite Element system
auto mu = solveImpedanceBVP(dh.fespace(), gridView, boundary_inters);
```

Solve

```
template <typename LINFESPACE, typename INTERS>
Eigen::VectorXd solveImpedanceBVP(const LINFESPACE& fes, const gridView_t& gv, const INTERS& boundary_inters)
{
  // ASSEMBLE RHS VECTOR
  // Define the source function g = 1
  const auto f = [](const coords_t& x){  return std::cos(x.norm()); };
  // type computing local intersection vectors
  typedef NPDE::LocalVectorAssembler trapLocFunAssembler_t;
  // type in charge of computing the right hand side vector using load vector assembler
  typedef NPDE::LoadVectorAssembler< trapLocFunAssembler_t > linearForm_t;
  // instantiate corresponding object
  linearForm_t F;
  // compute the global functional vector
  const Eigen::VectorXd& rhs = F.assembleRhs( fes, f );

  // ASSEMBLE GALERKIN MATRIX
  // type of objects computing element matrix
  typedef NPDE::AnalyticStiffnessLocalAssembler LocalMatAssembler1_t;
  // type taking care of assembly of Galerkin matrix
  typedef NPDE::GalerkinMatrixAssembler< LocalMatAssembler1_t > GalMatA_t;
  // instantiate corresponding object
  GalMatA_t A1;
  // compute the (big) Galerkin (stiffness) matrix in Eigen sparse matrix CRS format
  auto A_trips = A1.assembleTripletMatrix( fes, fes, gv );

  // Define the function gamma = 1 + x^2
  const auto gamma = [](const coords_t& x){ return 1.0; };
  // type of objects computing intersection matrix
  typedef NPDE::LaplRobinLocalMatrixAssembler LocalMatAssembler2_t;
  // type taking care of assembly of additions to Galerkin matrix
  typedef NPDE::IntersectionGalMatAsse< LocalMatAssembler2_t > GalMatA2_t;
  // instantiate corresponding object
  GalMatA2_t A2;
  // compute the (big) Galerkin (stiffness) matrix in Eigen sparse matrix CRS format
  const auto A2_trips = A2.assembleTripletMatrix( fes, fes, gamma, boundary_inters );

  A_trips.insert( A_trips.end(), A2_trips.begin(), A2_trips.end() );
  Eigen::SparseMatrix<numeric_t> Ah(fes.numDofs(),fes.numDofs());
  Ah.setFromTriplets(A_trips.begin(),A_trips.end());

  // SOLVE
  // define the (direct) solver
  typedef Eigen::SparseLU< Eigen::SparseMatrix<numeric_t> > solver_t;
  // instantiate the solver
```

```
  solver_t solver;
  // initialize it and solve
  solver.compute( Ah );
  Eigen::VectorXd sol = solver.solve( rhs );
  return sol;
}
```

18.1 Quadrature in BETL

Quadrature over Segment

```
static const refEl_t RET = refEl_t::SEGMENT;
result_t result; result.setZero();
// Get element geometry
const auto& geom = ic.geometry();

// Define quadrature for edge (3-points)
using quadrature_t = betl2::quad::Quadrature<RET, 3>;
// get points and weights over reference edge
const auto & w_i = quadrature_t::getWeights() * quadrature_t::getScale();
const auto & x_i = quadrature_t::getPoints();
// get determinant of Jacobian of 'reference->actual' edge transformation
const auto detJ_i = geom.template integrationElement<3>(x_i);
// map quadrature points and weights to current edge
const auto globw_i = detJ_i.cwiseProduct(w_i);
const auto& globx_i = geom.global(x_i);
// get quadratic basis functions for reference edge;
typedef fe::FEBasis< fe::Linear, fe::FEBasisType::Lagrange > febasis_t;
typedef typename febasis_t::template basisFunction_t<RET> basisFuncts;
// evaluate them on quadrature points
const auto functEval = basisFuncts::Eval(x_i);
// Loop over quadrature points
for (int i = 0; i < x_i.cols(); ++i) {
  // evaluate coefficient function at current quadrature point
  const double c_eval = c(globx_i.col(i));
  // Fetch basis functions evaluated at current quadrature point
  const auto b_i = functEval.template block<2,1>(0,i);
  // evaluate integrand of mass matrix for each ij test/trial basis functions
  result += globw_i(i) * b_i * c_eval * b_i.transpose();
}

return result;
```