Anders Logg • Kent-Andre Mardal
Editors

Lectures on the Finite Element Method

Contents

Preface

Acknowledgement

Miro, Solving, Ingeborg, Mikkel have all done a great deal.

Insert text her.

1 The finite element method

By Anders Logg, Kent–Andre Mardal

1.1 A simple model problem

Consider, in a domain $\Omega \subset \mathbb{R}^d$, the Poisson equation

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \quad && \text{in } \Omega, \\ u &= u_0 \quad && \text{on } \Gamma_{\mathrm{D}} \subset \partial\Omega, \\ -\kappa \nabla u \cdot n &= g \quad && \text{on } \Gamma_{\mathrm{N}} \subset \partial\Omega, \end{aligned} \tag{1.1}$$

where $u = u(x)$ is some unknown field, $\kappa : \Omega \to \mathbb{R}^{(d \times d)}$ is some given coefficient matrix and $f = f(x)$ is a given source function. The boundary $\partial\Omega$ of Ω is a union of two subboundaries, $\partial\Omega = \Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}}$. where Γ_{D} is the Dirichlet boundary and Γ_{N} is the Neumann boundary. The Dirichlet boundary condition, $u = u_0$, specifies a prescribed value for the unknown u on Γ_{D}. The Neumann boundary condition, $-\kappa \nabla u \cdot n = g$, specifies a prescribed value for the (negative) normal derivative of u on Γ_{N}. We often call the Dirichlet boundary condition an essential boundary condition, while we call Neumann boundary condition a natural boundary condition.

Let us look at one of the many examples where the equations (4.55) arises. Let $u = u(x)$ be the temperature in a body $\Omega \subset \mathbb{R}^d$ at a point x in the body, let $q = q(x)$ be the heat flux at x, let f be the heat source and let $\omega \subset \Omega$ be a small test volume. Conservation of energy gives

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \int_{\partial\omega} q \cdot n \,\mathrm{d}s - \int_{\omega} f \,\mathrm{d}x = 0, \tag{1.2}$$

that is, the outflow of the energy over the boundary $\partial\omega$ is equal to the energy emitted by the heat source function f. Fourier's law relates the heat flux to the temperature in the following way:

$$q = -\kappa \nabla u. \tag{1.3}$$

This gives us

$$\int_{\partial\omega} -\kappa \nabla u \cdot n \,\mathrm{d}s = \int_{\omega} f \,\mathrm{d}x. \tag{1.4}$$

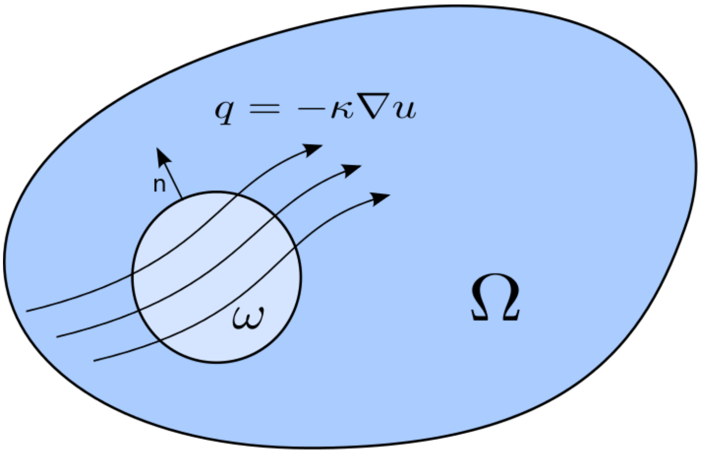


Figure 1.1: Sketch of the domain Ω and the two subboundaries Γ_D and Γ_N.

By the Gauss theorem,

$$\int_{\partial\omega} -\kappa\nabla u \cdot n \,\mathrm{d}s = \int_\omega \nabla\cdot(-\kappa\nabla u)\,\mathrm{d}x \tag{1.5}$$

$$\Rightarrow \quad -\int_\omega \nabla\cdot(\kappa\nabla u)\,\mathrm{d}x = \int_\omega f\,\mathrm{d}x. \tag{1.6}$$

Equation (1.6) holds for all test volumes $\omega \subset \Omega$. Thus, if u, κ and f are regular enough, we obtain

$$\int_\omega (-\nabla\cdot(\kappa\nabla u) - f)\,\mathrm{d}x = 0 \quad \forall\,\omega\subset\Omega \tag{1.7}$$

$$\Rightarrow \quad -\nabla\cdot(\kappa\nabla u) = f \quad \text{in } \Omega. \tag{1.8}$$

The Boundary conditions of this problem becomes

$$\begin{aligned} u &= u_0 \quad \text{on } \Gamma_D \\ -\kappa\nabla u\cdot n &= g \quad \text{on } \Gamma_N \end{aligned} \tag{1.9}$$

(recall that $q = -\kappa\nabla u$). This is illustrated in Figure 1.2. If we choose the special case where $\kappa = 1$, we obtain the more standard Poisson equation

$$-\Delta u = f \quad \text{in } \Omega. \tag{1.10}$$

Then, the boundary conditions becomes

$$u = u_0 \quad \text{on } \Gamma_D \tag{1.11}$$

$$-\frac{\partial u}{\partial n} = g \quad \text{on } \Gamma_N. \tag{1.12}$$

1.2 Solving Poisson's equation using the finite element method

Solving a PDE using the finite element method is done in four steps:

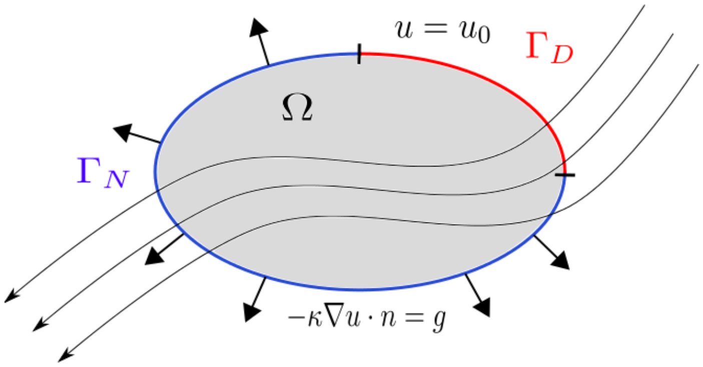


Figure 1.2: Sketch of the domain Ω and the two subboundaries Γ_{D} and Γ_{N}.

1. Strong form,
2. Weak (variational) form,
3. Finite element method,
4. Solution algorithm.

Let us go through these four steps for the Poisson problem.

1.2.1 *Strong form of Poisson's equation*

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \quad && \text{in } \Omega, \\ u &= u_0 && \text{on } \Gamma_{\mathrm{D}} \subset \partial\Omega, \\ -\kappa \nabla u \cdot n &= g && \text{on } \Gamma_{\mathrm{N}} \subset \partial\Omega. \end{aligned} \tag{1.13}$$

Recall that $\nabla u \cdot n = \frac{\partial u}{\partial n}$.

1.2.2 *Weak form of Poisson's equation*

To obtain the weak form we integrate (sometimes integration by parts is needed) the product of the strong form of the equation multiplied by a test function $v \in \hat{V}$, where $\hat{V}$ is called a test space:

$$\int_\Omega -\nabla \cdot (\kappa \nabla u) v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x \quad \forall v \in \hat{V} \tag{1.14}$$

$$\int_\Omega \kappa \nabla u \cdot \nabla v \, \mathrm{d}x - \int_{\partial\Omega} \kappa \frac{\partial u}{\partial n} v \, \mathrm{d}s = \int_\Omega f v \, \mathrm{d}x \quad \forall v \in \hat{V}. \tag{1.15}$$

Here we have done integration by parts using that

$$\int_\Omega (\nabla q) w \, \mathrm{d}x = \int_{\partial\Omega} (q \cdot n) w \, \mathrm{d}s - \int_\Omega q (\nabla w) \, \mathrm{d}x, \tag{1.16}$$

which in our case becomes

$$\int_\Omega -\nabla \cdot (\kappa \nabla u) v \, \mathrm{d}x = \int_{\partial\Omega} -\kappa \frac{\partial u}{\partial n} v \, \mathrm{d}s + \int_\Omega \kappa \nabla u \cdot \nabla v \, \mathrm{d}x. \tag{1.17}$$

Letting $v = 0$ on the Dirichlet boundary, Γ_{D}, the integral over the boundary becomes

$$\int_{\partial\Omega} -\kappa \frac{\partial u}{\partial n} v \,\mathrm{d}s = \int_{\Gamma_{\mathrm{N}}} -\kappa \frac{\partial u}{\partial n} v \,\mathrm{d}s = \int_{\Gamma_{\mathrm{N}}} g v \,\mathrm{d}s. \tag{1.18}$$

We have arrived at the follwing problem: find $u \in V$ such that

$$\int_{\Omega} \kappa \nabla u \cdot \nabla v \,\mathrm{d}x = \int_{\Omega} f v \,\mathrm{d}x - \int_{\Gamma_{\mathrm{N}}} g v \,\mathrm{d}s \quad \forall\, v \in \hat{V}. \tag{1.19}$$

The test space $\hat{V}$ is defined by

$$\hat{V} = H^1_{0,\Gamma_{\mathrm{D}}}(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_{\mathrm{D}}\} \tag{1.20}$$

and the trial space V, containing the unknown function u, is defined similar to $\hat{V}$ but with a shifted Dirichlet condition:

$$V = H^1_{u_0,\Gamma_{\mathrm{D}}}(\Omega) = \{v \in H^1(\Omega) : v = u_0 \text{ on } \Gamma_{\mathrm{D}}\}. \tag{1.21}$$

1.2.3 The finite element method for Poisson's equation

We discretize the variational problem (1.19) by looking for a solution in a discrete trial space and using a discrete test function. The finite element problem is: find $u_h \in V_h \subset V$ such that

$$\int_{\Omega} \kappa \nabla u_h \cdot \nabla v \,\mathrm{d}x = \int_{\Omega} f v \,\mathrm{d}x - \int_{\Gamma_{\mathrm{N}}} g v \,\mathrm{d}s \quad \forall\, v \in \hat{V}_h \subset \hat{V}, \tag{1.22}$$

where V_h and $\hat{V}_h$ are discrete subspaces of V and $\hat{V}$, respectively.

1.2.4 Solution algorithm

Our question is now: How do we solve the discrete variational problem (1.22)? We introduce a basis for V and V_h, and make an Anzats:

$$u_h(x) = \sum_{j=1}^{N} U_j \phi_j(x), \tag{1.23}$$

where

$$\phi_j : \Omega \to \mathbb{R}, \quad j = 1, \ldots, N, \tag{1.24}$$

is basis for V_h. Inserting this into equation (1.22) and letting $v = \hat{\phi}_i$, $i = 1, \ldots, N$, we obtain

$$\begin{aligned} \int_{\Omega} \kappa \nabla \left(\sum_{j=1}^{N} U_j \phi_j \right) \cdot \nabla \hat{\phi}_i \,\mathrm{d}x &= \int_{\Omega} f \hat{\phi}_i \,\mathrm{d}x - \int_{\Gamma_{\mathrm{N}}} g \hat{\phi}_i \,\mathrm{d}s, \quad i = 1, 2, \ldots, N, \\ \sum_{j=1}^{N} U_j \int_{\Omega} \kappa \nabla \phi_j \cdot \nabla \hat{\phi}_i \,\mathrm{d}x &= \int_{\Omega} f \hat{\phi}_i \,\mathrm{d}x - \int_{\Gamma_{\mathrm{N}}} g \hat{\phi}_i \,\mathrm{d}s, \quad i = 1, 2, \ldots, N. \end{aligned} \tag{1.25}$$

We recognize this as a system of linear equations:

$$\begin{aligned} \sum_{j=1}^{N} A_{ij} U_j &= b_i, \quad i = 1, 2, \ldots, N, \\ AU &= b, \end{aligned} \tag{1.26}$$

where

$$\begin{aligned} A_{ij} &= \int_\Omega \kappa \nabla \phi_j \cdot \nabla \hat{\phi}_i \, \mathrm{d}x, \\ b_i &= \int_\Omega f \hat{\phi}_i \, \mathrm{d}x - \int_{\Gamma_\mathrm{N}} g \hat{\phi}_i \, \mathrm{d}s. \end{aligned} \tag{1.27}$$

1.3 *Solving the Poisson equation with FEM using abstract formalism*

1.3.1 *The problem written in strong form*

The strong form of the Poisson equation written as a linear system reads

$$\begin{aligned} Au &= f, \\ (+ \text{ BCs }&), \end{aligned} \tag{1.28}$$

where A is a discrete differential operator.

1.3.2 *The problem written in weak (variational) form*

Let V be a Hilbert space with inner product $\langle \cdot, \cdot \rangle$, then

$$\langle Au, v \rangle = \langle f, v \rangle \tag{1.29}$$

Define

$$\begin{aligned} a(u, v) &= \langle Au, v \rangle, \\ L(v) &= \langle f, v \rangle, \end{aligned} \tag{1.30}$$

where a is a bilinear form (not necessarily an inner product) and L is a linear form (a functional):

$$\begin{aligned} a &: V \times \hat{V} \to \mathbb{R}, \\ L &: \hat{V} \to \mathbb{R}. \end{aligned} \tag{1.31}$$

The variational problem becomes: find $u \in V$ such that

$$a(u, v) = L(v) \quad \forall \, v \in \hat{V}. \tag{1.32}$$

1.3.3 *Finite element method*

In the finite element problem, we look for a discrete solution: find $u_h \in V_h$ such that

$$a(u_h, v) = L(v) \quad \forall \, v \in \hat{V}_h. \tag{1.33}$$

1.3.4 Solution algorithm

Let $\{\phi_i\}_{i=1}^N$ be a basis for V_h. Make an Anzats

$$u_h(x) = \sum_{j=1}^{N} U_j \phi_j(x). \tag{1.34}$$

Inserting this to the variational form, it follows

$$\begin{aligned} a\left(\sum_{j=1}^{N} U_j \phi_j, \hat{\phi}_i\right) &= L\left(\hat{\phi}_i\right), \quad i = 1, 2, \cdots, N, \\ \sum_{j=1}^{N} U_j a(\phi_j, \hat{\phi}_i) &= L\left(\hat{\phi}_i\right), \quad i = 1, 2, \cdots, N. \end{aligned} \tag{1.35}$$

As before, u_h may be computed by solving a linear system

$$\begin{aligned} \sum_{j=1}^{N} A_{ij} U_j &= b_i, \quad i = 1, 2, \ldots, N, \\ AU &= b, \end{aligned} \tag{1.36}$$

where

$$\begin{aligned} A_{ij} &= a(\phi_j, \hat{\phi}_i), \\ b_i &= L\left(\hat{\phi}_i\right). \end{aligned} \tag{1.37}$$

1.4 Galerkin orthogonality

We will now show Galerkin orthogonality. First, we know that

$$\begin{aligned} a(u, v) &= L(v) \quad \forall\, v \in V, \\ a(u_h, v) &= L(v) \quad \forall\, v \in V_h \subset V. \end{aligned} \tag{1.38}$$

Using these results and the linearity of the bilinear form, we get

$$a(u - u_h, v) = a(u, v) - a(u_h, v) = L(v) - L(v) = 0 \quad \forall\, v \in V_h, \tag{1.39}$$

or written symbolically

$$u - u_h \perp_a V_h. \tag{1.40}$$

This property is called Galerkin orthogonality. The error, $e = u - u_h$, is orthogonal (in the sense of the bilinear form a) to the test space V_h. Thus, u_h is the best possible approximation of u in V_h. We will continue this concept in the next chapter.

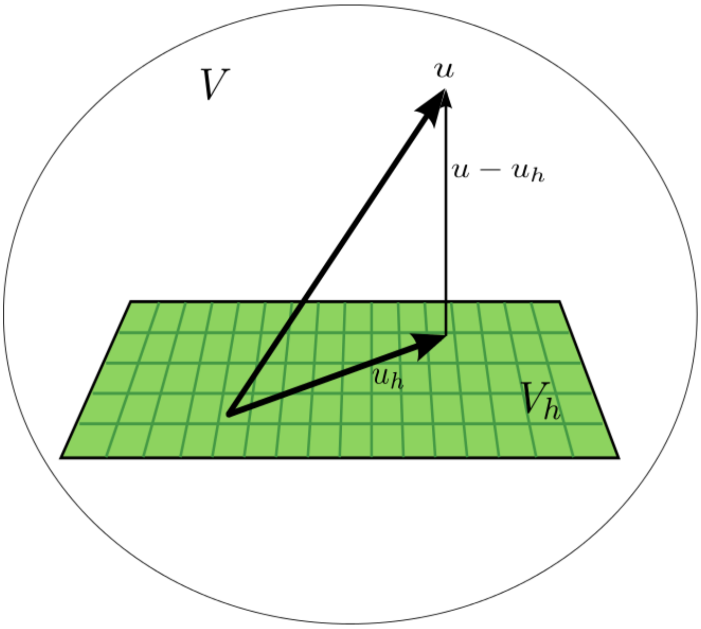


Figure 1.3: The finite element solution $u_h \in V_h \subset V$ is the projection of $u \in V$ in the sense of the bilinear form a onto the subspace V_h and is consequently the best possible approximation of u in V_h.

2 A short look at functional analysis and Sobolev spaces

By Anders Logg, Kent–Andre Mardal

The finite element method (FEM) is a general framework for numerical solution of PDEs. FEM is written in the language of functional analysis, therefore we need to introduce basic concepts and notations from functional analysis and Sobolev spaces.

The fundamental idea is that functions are vectors in a function space which is a vector space. The properties of a vector space is briefly reviewed below. Then we may equip the spaces with norms and inner-products which allow us to quantify, e.g., magnitudes and differences between functions. A fundament mathematical difficulty is, however, that the function spaces typically will be infinite dimensional in the continuous setting, but this difficulty will not be addressed here.

2.1 Functional analysis

Definition 2.1. *Vector space (over a field $F \in \mathbb{R}$)*
A vector space is a set V equipped with,

- *addition $+ : V \times V \to V$*
- *multiplication $\cdot : F \times V \to V$*

Where $+$ and $\cdot$ satisfy the following conditions

1. *$+$ is commutative:* $v + u = u + v$
2. *$+$ is associative:* $u + (v + w) = (u + v) + w$
3. *additive identity:* $\exists\, 0 \in V$ *such that* $v + 0 = 0 + v = v$
4. *additive inverse:* $\exists - v \in V$ *such that* $v + (-v) = (-v) + v = 0$
5. *$\cdot$ is distributive:* $c \cdot (u + v) = c \cdot u + c \cdot v$
6. *$\cdot$ is distributive:* $(c + d) \cdot v = c \cdot v + d \cdot v$
7. *$\cdot$ is associative:* $c \cdot (d \cdot v) = (c \cdot d) \cdot v$
8. *multiplicative identity:* $1 \cdot v = v$

for all $u, v, w \in V$ and $c, d \in \mathbb{R}$.

Examples:

1. $V = \mathbb{R}$
2. $V = \mathbb{R}^3$
3. $V = \mathbb{R}^N$, $[x_1, \dots, x_N] + [y_1, \dots, y_N] = [x_1 + y_1, \dots, x_N + y_N]$ and $\alpha[x_1, \dots, x_N] = [\alpha x_1, \dots, \alpha x_N]$
4. $V = \{v : [0,1] \to \mathbb{R} \mid v \text{ is continuous}\}$
5. $V = \{v : [0,1] \to \mathbb{R} \mid v(x) \leqslant 1, \quad \forall\, x \in [0,1]\}$, **NOT** a vector space!

Definition 2.2. *Inner product space (over a field $F = \mathbb{R}$)*
An inner product space is a vector space with an inner product, a map,

$$\langle \cdot, \cdot \rangle : V \times V \to F,$$

satisfying the following conditions:

$$\begin{array}{lll} 1. & \langle v, w\rangle = \overline{\langle w, v\rangle} \quad \forall\, v, w \in V & \textit{(conjugate symmetry)} \\ 2. & \left.\begin{array}{l} \langle \alpha v, w\rangle = \alpha \langle v, w\rangle \quad \forall\, v \in V \textit{ and } \forall\, \alpha \in F \\ \langle u + v, w\rangle = \langle u, w\rangle + \langle v, w\rangle \quad \forall\, u, v, w \in V \end{array}\right\} & \textit{(linearity)} \\ 3. & \langle v, v\rangle \geqslant 0 \quad \textit{with} \quad \langle v, v\rangle = 0 \textit{ iff } v = 0 & \textit{(positive definite)} \end{array}$$

Examples:

1. $V = \mathbb{R}^N$, $\qquad \langle v, w\rangle = \sum_{i=1}^{N} v_i w_i$
2. $V = \ell^2$, $\qquad \langle v, w\rangle = \sum_{i=1}^{\infty} v_i w_i$
3. $V = C^\infty(\Omega)$, $\qquad \langle v, w\rangle = \int_\Omega v w \,\mathrm{d}x$

ℓ^2 is the space of all sequences (or infinite vectors) that satisfy $\sum_i v_i^2 < \infty$.

Definition 2.3. *Orthogonality*
Let V be an inner product space. Two vectors $u, v \in V$ are said to be orthogonal if

$$\langle v, w\rangle = 0.$$

Examples:

1. $V = \mathbb{R}^3$, $\qquad v = (1, 2, 3), \quad w = (3, 0, -1)$
2. $V = \mathcal{P}^2([-1, 1])$, $\qquad u = 1, \quad v = x, \quad w = \frac{1}{2}(3x^2 - 1)$ (Legendre polynomials)

Definition 2.4. *Normed vector space (over a field F)*
A normed vector space is a vector space with a norm, a map,

$$\| \cdot \| : V \to \mathbb{R},$$

satisfying the following conditions:

1. $\|\alpha v\| = |\alpha|\|v\|, \quad \forall\, v \in V$ *and* $\forall\, \alpha \in F$ *(Positive homogeneity)*
2. $\|u+v\| \leqslant \|u\| + \|v\|, \quad \forall\, u, v \in V$ *(triangle inequality)*
3. $\|v\| = 0 \quad \Rightarrow \quad v = 0$ *(point separation)*

Examples:

1. $V = \mathbb{R}^N, \quad \|v\|_p = \left(\sum_{i=1}^N v_i^p\right)^{1/p}, \quad 1 \leqslant p < \infty$
2. $V = \mathbb{R}^N, \quad \|v\|_\infty = \max_{1 \leqslant i \leqslant N} |v_i|$
3. $V = C^\infty(\Omega), \quad \|v\|_p = \left(\int_\Omega v^p \,\mathrm{d}x\right)^{1/p}, \quad 1 \leqslant p < \infty$
4. $V = C^\infty(\Omega), \quad \|v\|_\infty = \sup_{x\in\Omega} |v(x)|$
5. V inner product space, $\|v\| = \sqrt{\langle v, v\rangle}$. Thus, an inner product space is a normed space. (Exercise: show this!)

Definition 2.5. *Cauchy sequence (on normed space)*
Let V be a normed space[1]*. A sequence $\{v_i\}_{i=1}^\infty \subset V$ is a Cauchy sequence if for all $\epsilon > 0$ there exists a number $N > 0$, such that*

$$\|v_m - v_n\| < \epsilon \quad \forall\, m, n > N.$$

Examples:

1. $V = \mathbb{R}, \quad \|v\| = |v|, \quad v_n = \frac{1}{n}$
2. $V = \mathbb{R}, \quad \|v\| = |v|, \quad v_n = \frac{\sin n}{n}$
3. $V = C([0,1]), \quad \|v\| = \|v\|_\infty, \quad v_n(x) = \sum_{i=0}^n \frac{x^i}{i!}$
4. $$V = C([-1,1]), \quad v_n(x) = \begin{cases} -1, & x \in [-1, -\frac{1}{n}] \\ nx, & x \in (-\frac{1}{n}, \frac{1}{n}) \\ 1, & x \in [\frac{1}{n}, 1] \end{cases}$$

   This sequence is Cauchy in the L^1-norm, $\|v\|_1 = \int_{-1}^1 |v(x)|\,\mathrm{d}x$, but not Cauchy in the max norm, $\|v\|_\infty = \max_{x\in[-1,1]} |v(x)|$, because $C([-1,1])$ with $\|\cdot\|_\infty$ is not complete.

Figure 2.1 and 2.2 show the Cauchy sequence for example 1 and 2.

Definition 2.6. *Completeness*
A (metric) space, V, is complete if all Cauchy sequences converge to a point in V.

Definition 2.7. *Banach space*
A Banach space is a complete normed vector space.

Definition 2.8. *Hilbert space*
A Hilbert space is a complete normed inner product space.

[1] Can be generalized to metric spaces, $d(v_m, v_n) < \epsilon$

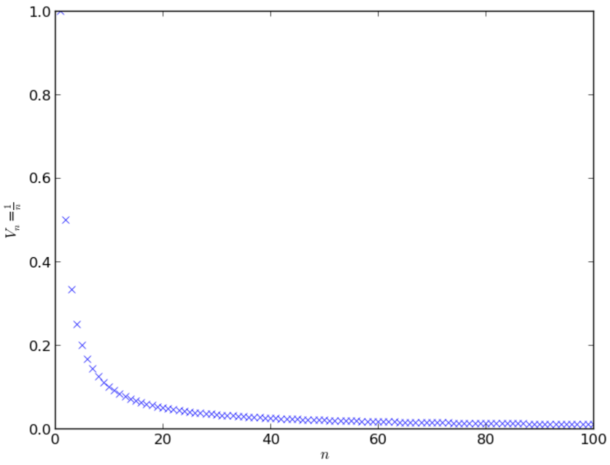


Figure 2.1: Cauchy sequence: $\frac{1}{n}$ for $n = 1, \ldots, 100$.

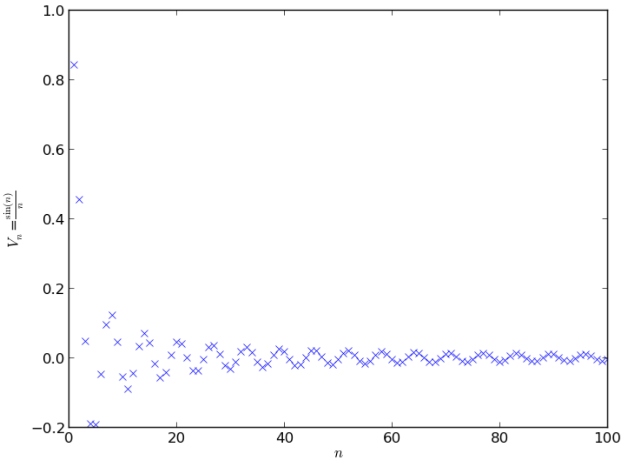


Figure 2.2: Cauchy sequence: $\frac{\sin n}{n}$ for $n = 1, \ldots, 100$.

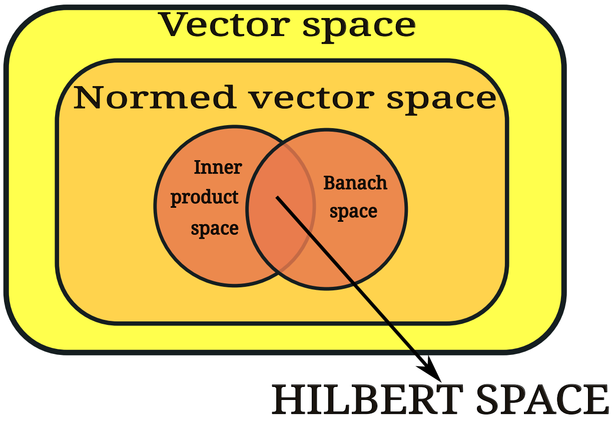


Figure 2.3: Venn diagram of the different spaces.

Definition 2.9. *(Continuous) Dual space*
Let V be a normed vector space. The dual space V' (sometimes denoted $V^\star$) is the space of all continuous, linear functionals on V:

$$V' = \{l : V \to \mathbb{R} \mid \|l\| < \infty\} \quad \textit{where,} \quad \|l\| = \sup_{\|v\| \leqslant 1} |l(v)|$$

So far we have looked at a lot of definitions, let us now consider some important results.

Theorem 2.1. *Cauchy–Schwartz inequality*
Let V be an inner product space. Then

$$|\langle v, w\rangle| \leqslant \|v\| \cdot \|w\| \quad \forall\, v, w \in V.$$

Theorem 2.2. *Banach fixed-point theorem*
Let V be a Banach space and let

$$T : V \to V$$

be a continuous mapping on V, that is,

$$\exists\, M < 1 \,:\, \|T(v) - T(w)\| \leqslant M\|v - w\| \quad \forall\, v, w \in V.$$

Then $\exists!\ \bar{v} \in V$, such that $T\bar{v} = \bar{v}$.

Examples:

1. $V = \mathbb{R}$, $\quad Tv = \frac{v}{2}, \quad \bar{v} = 0$
2. $V = \mathbb{R}^+$, $\quad Tv = \frac{v+2/v}{2}, \quad \bar{v} = \sqrt{2}$

Theorem 2.3. *Riesz representation theorem*
Let H be a Hilbert space and let H' denote its dual space. Then for all $l \in H'$ there exists a unique element $\hat{l} \in H$, such that

$$l(v) = \langle \hat{l}, v\rangle \quad \forall\, v \in H$$

Theorem 2.4. *Integration by parts in n–dimensions*

Let $\Omega \in \mathbb{R}^n$ and let v and w be functions in $H^1(\Omega)$. Then,

$$\int_\Omega \frac{\partial v}{\partial x_i} w \, \mathrm{d}x = -\int_\Omega \frac{\partial w}{\partial x_i} v \, \mathrm{d}x + \int_{\partial\Omega} v \, w \, n_i \, dS,$$

where n_i it the i'th normal component.

2.2 Sobolev spaces

We will now turn our attention to the ralated topic of Sobolev spaces.

Definition 2.10. *The $L^2(\Omega)$ space*
Let Ω be an open subset of $\mathbb{R}^n$, with piecewise smooth boundary, then $L^2(\Omega)$ is defined by

$$L^2(\Omega) = \{v : \Omega \to \mathbb{R} \mid \int_\Omega v^2 \, \mathrm{d}x < \infty\}$$

Examples:

1. $v(x) = \frac{1}{\sqrt{x}}, \quad \Omega = (0,1), \quad v \notin L^2(\Omega)$
2. $v(x) = \frac{1}{x^{\frac{1}{4}}}, \quad \Omega = (0,1), \quad v \in L^2(\Omega)$

Theorem 2.5. *L^2 with $\langle v, w \rangle = \int_\Omega vw \, \mathrm{d}x$ is a Hilbert space.*

Definition 2.11. *Weak derivative (first order)*
Let $v \in L^2(\Omega)$. The weak derivative of v (if it exists), is a function $\frac{\partial v}{\partial x_i} \in L^2(\Omega)$ satisfying,

$$\int_\Omega \frac{\partial v}{\partial x_i} \phi \, \mathrm{d}x = -\int_\Omega v \frac{\partial \phi}{\partial x_i} \, \mathrm{d}x, \quad \forall \phi \in C_0^\infty(\Omega).$$

Definition 2.12. *Weak derivative (general order)*
Let $v \in L^2(\Omega)$. The weak derivative of v (if it exists), is a function $\partial^\alpha v \in L^2(\Omega)$ satisfying

$$\int_\Omega \partial^\alpha v \, \phi \, \mathrm{d}x = (-1)^{|\alpha|} \int_\Omega v \, \partial^\alpha \phi \, \mathrm{d}x, \quad \forall \phi \in C_0^\infty(\Omega)$$

where

$$\partial^\alpha \phi = \frac{\partial^{|\alpha|}}{\partial^{\alpha_1} x_1 \partial^{\alpha_2} x_2 \ldots \partial^{\alpha_n} x_n}.$$

Lemma 2.1. *A weak derivative (if it exist), is unique.*

Lemma 2.2. *A (strong) derivative (if it exist), is a weak derivative.*

Definition 2.13. *The Sobolev space H^m*
The sobolev space H^m is the subspace of functions v in $L^2(\Omega)$, which possess weak derivatives ∂^α for $|\alpha| \leqslant m$. The corresponding norm is

$$\|v\|_{H^k} = \sqrt{\sum_{|\alpha| \leqslant k} \int_\Omega |\partial^\alpha v|^2 \, \mathrm{d}x} \equiv \sqrt{\sum_{|\alpha| \leqslant k} \|\partial^\alpha v\|^2_{L^2(\Omega)}}$$

and seminorm

$$|v|_{H^k} = \sqrt{\sum_{|\alpha| = k} \int_\Omega |\partial^\alpha v|^2 \, \mathrm{d}x} \equiv \sqrt{\sum_{|\alpha| = k} \|\partial^\alpha v\|^2_{L^2(\Omega)}}.$$

Theorem 2.6. *H^1 is a Hilbert space*

$$\langle v, w \rangle = \int_\Omega vw \,\mathrm{d}x + \int_\Omega \nabla v \cdot \nabla w \,\mathrm{d}x$$

Theorem 2.7. *Poincaré inequality*
Let $v \in H_0^1(\Omega)$. Then,

$$\|v\|_{L^2(\Omega)} \leqslant C|v|_{H^1(\Omega)},$$

where C depends only on Ω.

3 Crash course in Sobolev Spaces

By Anders Logg, Kent–Andre Mardal

3.1 Introduction

Sobolev spaces are fundamental tools in the analysis of partial differential equations and also for finite element methods. Many books provide a detailed and comprehensive analysis of these spaces that in themselves deserve significant attention if one wishes to understand the foundation that the analysis of partial differential equations relies on. In this chapter we will however not provide a comprehensive mathematical description of these spaces, but rather try to provide insight into their use.

We will here provide the definition of these spaces. Further we will show typical functions, useful for finite element methods, that are in some but not all spaces. We also show how different norms capture different characteristics.

3.2 Sobolev spaces, norms and inner products

Sobolev spaces are generalizations of L^p spaces. L^p spaces are function spaces defined as follows. Let u be a scalar valued function on the domain Ω, which for the moment will be assumed to be the unit interval $(0,1)$. Then

$$\|u\|_p = (\int_0^1 |u|^p dx)^{1/p}.$$

$L^p(\Omega)$ consists of all functions for which $\|u\|_p < \infty$. Sobolev spaces generalize L^p spaces by also including the derivatives. On the unit interval, let

$$\|u\|_{p,k} = (\int_\Omega \sum_{i\leq k} |(\frac{\partial u}{\partial x})^i|^p dx)^{1/p}. \tag{3.1}$$

Then the Sobolev space $W_k^p(\Omega)$ consists of all functions with $\|u\|_{p,k} < \infty$. W_k^p is a so-called Banach space - that is a complete normed vector space. The corresponding semi-norm, that only include the highest order derivative is

$$|u|_{p,k} = (\int_\Omega \sum_{i=k} |(\frac{\partial}{\partial x})^i u|^p dx)^{1/p}. \tag{3.2}$$

The case $p = 2$ is special in the sense that (3.1) defines an inner product. The Banach space then forms a Hilbert space and these named with H in Hilbert's honor. That is $H^k(\Omega) = W^{2,k}(\Omega)$.

For the most part, we will employ the two spaces $L^2(\Omega)$ and $H^1(\Omega)$, but also H^2 and H^{-1} will be used. The difference between the norm in $L^2(\Omega)$ and $H^1(\Omega)$ is illustrated in the following example.

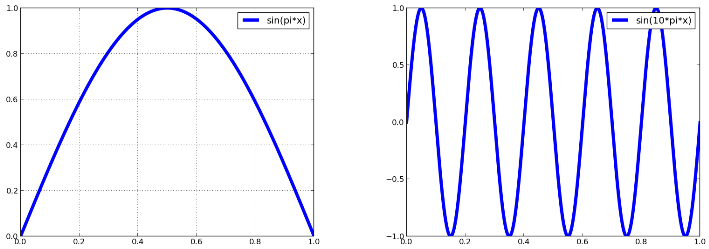


Figure 3.1: Left picture shows $\sin(\pi x)$ on the unit interval, while the right picture shows $\sin(10\pi x)$.

Example 3.1. *Norms of $\sin(k\pi x)$ Consider the functions $u_k = \sin(k\pi x)$ on the unit interval. Figure 3.1 shows the function for $k = 1$ and $k = 10$. Clearly, the L^2 and L^7 behave similarly in the sense that they remain the same as k increases. On the other hand, the H^1 norm of u_k increases dramatically as k increases. The following code shows how the norms are computed using FEniCS.*

Python code

```python
from dolfin import *

N = 10000
mesh = UnitInterval(N)
V = FunctionSpace(mesh, "Lagrange", 1)

for k in [1, 100]:
  u_ex = Expression("sin(k*pi*x[0])", k=k)
  u = project(u_ex, V)

  L2_norm = sqrt(assemble(u**2*dx))
  print "L2 norm of sin(%d pi x) %e " % (k, L2_norm)

  L7_norm = pow(assemble(abs(u)**7*dx), 1.0/7)
  print "L7 norm of sin(%d pi x) %e " % (k, L7_norm)

  H1_norm = sqrt(assemble(u*u*dx + inner(grad(u), grad(u))*dx ))
  print "H1 norm of sin(%d pi x) %e" % (k, H1_norm)
```

$k \backslash norm$	L^2	L^7	H^1
1	0.71	0.84	2.3
10	0.71	0.84	22
100	0.71	0.84	222

Table 3.1: The L^2, L^7, and H^1 norms of $sin(k\pi x)$ for k=1, 10, and 100.

3.3 Spaces and sub-spaces

The Sobolev space with k derivatives in $L_2(\Omega)$ was denoted by $H^k(\Omega)$. The subspace of H^k with $k-1$ derivatives equal to zero at the boundary is denoted $H^k_0(\Omega)$. For example, $H^1_0(\Omega)$ consists of all functions in H^1 that are zero at the boundary. Similarly, we may also defined a subspace $H^1_g(\Omega)$ which consists of all functions in $H^1(\Omega)$ that are equal to the function g on the boundary.

Mathematically, it is somewhat tricky to defined that a function in H^1 is equal to another function as it can not be done in a pointwise sense. This difficulty is resolved by the concept of a trace usually denoted by T. The concept of a trace is tricky, for example if T takes a function u in $H^1(\Omega)$ and restrict it to $\partial\Omega$ then $Tu \notin H^1(\partial\Omega)$. In fact, in general we only have $Tu \in H^{1/2}(\partial\Omega)$.

3.4 Norms and Semi-norms

The norm $\|\cdot\|_{p,k}$ defined in 3.1 is a norm which means that $\|u\|_{p,k} > 0$ for all $u \neq 0$. On the other hand $|\cdot|_{p,k}$ is a semi-norm, meaning that $|u|_{p,k} \geq 0$ for all u. The space $H^1(\Omega)$ is defined by the norm

$$\|u\|_1 = (\int_\Omega u^2 + (\nabla u)^2 dx)^{1/2}$$

and contains all functions for which $\|u\|_1 \leq \infty$. Often we consider subspaces of H^1 satisfying the Dirichlet boundary conditions. The most common space is denoted H^1_0. This space contains all functions in H^1 that are zero on the boundary. The semi-norm $|\cdot|_1$ defined as

$$|u|_1 = (\int_\Omega (\nabla u)^2 dx)^{1/2}$$

is a norm on the subspace H^1_0. In fact, as we will see later, Poincare's lemma ensures that $\|\cdot\|_1$ and $|\cdot|_1$ are equivalent norms on H^1_0 (see Exercise 3.5.

3.5 Examples of Functions in Different Spaces

The above functions $\sin(k\pi x)$ are smooth functions that for any k are infinitely many times differentiable. They are therefore members of any Soblev space.

On the other had, the step function in upper picture in Figure 3.2 is discontinuous in $x = 0.2$ and $x = 0.4$. Obviously, the function is in $L^2(0,1)$, but the function is not in $H^1(0,1)$ since the derivative of the function consists of Dirac's delta functions[1] that are ∞ at $x = 0.2$ and $-\infty$ in $x = 0.4$.

The hat function in the lower picture in Figure 3.2 is a typical first order finite element function. The function is in both $L^2(0,1)$ and $H^1(0,1)$ (see Exercise 3.3). In general, functions in H^q are required to be in C^{q-1}, where C^k is the class where the k'th derivatives exist and are continuous.

3.6 Sobolev Spaces and Polynomial Approximation

From Taylor series we know that a $f(x+h)$ may be approximated by $f(x)$ and a polynomial in h that depends on the derivatives of f. To be precise,

$$|f(x+h) - (P_k f)(x)| \leq \mathcal{O}(h^{k+1}).$$

[1] The Dirac's delta function δ_x is 0 everywhere except at x where it is ∞ and $\int_\Omega \delta_x dx = 1$. Hence, Dirac's delta function is in $L^1(\Omega)$ but not in $L^2(\Omega)$.

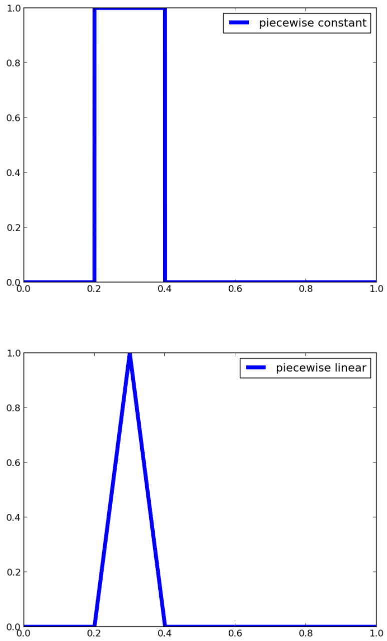


Figure 3.2: The upper picture shows a piecewise function, discontinuous at $x = 0.2$ and $x = 0.2$, while the lower picture shows a linear function that is continuous.

Here, $(P_k f)(x)$ is a polynomial of degree k in h, $f^{(n)}$ denotes the n'th derivative of f, and the error will be of order $k+1$ in h. To be precise,

$$(P_k f)(x) = f(x) + \sum_{n=1}^{k} \frac{f^{(n)}(x)}{n!} h^n.$$

In general, approximation by Taylor series bears strong requirement on the smoothness of the solution which needs to be differentiable in a point-wise sense. However, in Sobolev spaces we have the very usefull approximation property

$$|u - P_k u|_{m,p} \leq C h^{k-m} |u|_{k,p} \quad \text{for } m = 0, 1, \ldots, k \text{ and } k \geq 1.$$

This property is used extensively in analysis of finite element methods. The above approximation property is often called the Bramble-Hilbert lemma for $k \geq 2$ and the case $k = 1$ was included by a special interpolation operator by Clement, the so-called Clement interpolant. For proof, see e.g. **??**.

3.7 Eigenvalues and Finite Element Methods

We remember that for $-\Delta$ on the unit interval $(0, 1)$, the eigenvalues and eigenvectors are $(\pi k)^2$ and $sin(\pi k x)$, $k = 1, \ldots, \infty$, respectively. It is natural to expect that the eigenvalues in the discrete setting approximate the continuous eigenvalues such that the minimal eigenvalue is $\approx \pi^2$, while the maximal eigenvalue is $\approx \pi^2/h^2$, where $k = 1/h$ is the largest k that may be represented on a mesh with element size h. Computing the eigenvalues of the finite element stiffness matrix in FEniCS as[2],

Python code

```
A = assemble_system(inner(grad(u), grad(v))*dx, Constant(0)*v*dx, bc)
```

reveals that the eigenvalues are differently scaled. In fact, the minimal eigenvalue is $\approx \pi^2 h$ and that the maximal eigenvalue is $\approx \pi^2/h$. The reason is that the finite element method introduces a mesh-dependent scaling. To estimate the continuous eigenvalues we instead compute the eigenvalues of the generalized eigenvalue problem,

$$Ax = \lambda M x, \tag{3.3}$$

where A is the above mentioned stiffness matrix and M is the mass matrix (or the finite element identity matrix)

Python code

```
M = assemble_system(inner(u*v*dx, Constant(0)*v*dx, bc)
```

Figure 3.3 shows the eigenvalues of $-\Delta$, A, and (3.3) based on the following code:

Python code

```
from dolfin import *
import numpy
from scipy import linalg, matrix

def boundary(x, on_boundary): return on_boundary

for N in [100, 1000]:
  mesh = UnitIntervalMesh(N)
  V = FunctionSpace(mesh, "Lagrange", 1)
  u = TrialFunction(V)
```

[2] We use the `assemble_system` function to enforce the Dirichlet condition in symmetric fashion.

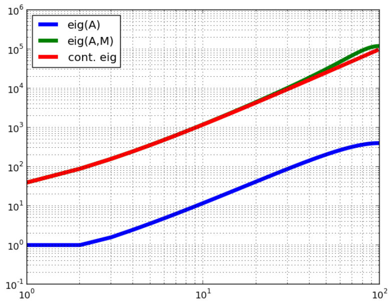


Figure 3.3: A log-log plot of the eigenvalues of A, $M^{-1}A$, and $-\Delta$.

```
v = TestFunction(V)

bc = DirichletBC(V, Constant(0), boundary)
A, _ = assemble_system(inner(grad(u), grad(v))*dx, Constant(0)*v*dx, bc)
M, _ = assemble_system(u*v*dx, Constant(0)*v*dx, bc)

AA = matrix(A.array())
MM = matrix(M.array())

k = numpy.arange(1, N, 1)
eig = pi**2*k**2

l1, v  = linalg.eigh(AA)
l2, v  = linalg.eigh(AA, MM)

print "l1 min, max ", min(l1), max(l1)
print "l2 min, max ", min(l2), max(l2)
print "eig min, max ", min(eig), max(eig)

import pylab
pylab.loglog(l1[2:], linewidth=5)  # exclude the two smallest (they correspond to Dirichlet cond))
pylab.loglog(l2[2:], linewidth=5)  # exclude the two smallest again
pylab.loglog(eig, linewidth=5)
pylab.legend(["eig(A)", "eig(A,M)", "cont. eig"], loc="upper left")
pylab.show()
```

From Figure 3.3 we see that that the eigenvalues of (3.3) and $-\Delta$ are close, while the eigenvalues of A is differently scaled. We remark that we excluded the two smallest eigenvalues in the discretized problems as they correspond to the Dirichlet conditions.

3.8 Negative and Fractional Norms

As will be discussed more thoroughly later, $-\Delta$ is a symmetric positive operator and can be thought of as a infinite dimensional matrix that is symmetric and positive. It is also know from Riesz representation theorem that if u solves the problem

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \partial\Omega \end{aligned}$$

then

$$|u|_1 = \|f\|_{-1}. \tag{3.4}$$

This implicitly define the H^{-1} norm, although the definition then requires the solution of a Poisson problem. For example, in the previous example where $u_k = sin(k\pi x)$, we have already estimated that $|u_k|_1 = \frac{\pi k}{\sqrt{2}}$ and therefore $\|u_k\|_{-1} = |(-\Delta)^{-1} u_k|_1 = \frac{1}{\sqrt{2}k\pi}$.

Let us now generalize these considerations and consider a matrix (or differential operator) A which is symmetric and positive. A has positive and real eigenvalues and defines an inner product which may be represented in terms of eigenvalues and eigenfunctions. Let λ_i and u_i be the eigenvalues and eigenfunctions such that

$$Au_i = \lambda_i u_i$$

Then, x may be expanded in terms of the eigenfunctions u_i as $x = \sum_i c_i u_i$, where $c_i = (x, u_i)$, and we obtain

$$(x,x)_A = (Ax, x) = (A\sum_i c_i u_i, \sum_j c_j u_j) = (\sum_i \lambda_i c_i u_i, \sum_j c_j u_j)$$

Because A is symmetric, the egenfunctions u_i are orthogonal to each other and we may choose a normalized basis such that $(u_i, u_j) = \delta_{ij}$. With this normalization, we simply obtain

$$\|x\|_A^2 = (x,x)_A = (Ax, x) = (A\sum_i c_i u_i, \sum_j c_j u_j) = \sum_i \lambda_i c_i^2$$

A generalization of the $A-$inner product (with corresponding norm) to a A^q-inner product that allow for both negative and franctional q is then as follows

$$\|x\|_{A,q}^2 = (x,x)_{A,q} = \sum_i \lambda_i^q c_i^2. \tag{3.5}$$

Clearly, this definition yields that $|u_k|_1 = \frac{\pi k}{\sqrt{2}}$ and $\|u_k\|_{-1} = \frac{1}{\sqrt{2}k\pi}$, as above.

As mentioned in Section 3.7, care has to be taken in finite element methods if the discrete eigenvalues are to correspond with the continuous eigenvalues. We will therefore detail the computation of negative and fractional norms in the following. Let λ_i and u_i be the eigenvalues and eigenvectors of the following generalized eigenvalue problem

$$Au_i = \lambda_i M u_i \tag{3.6}$$

and let U be the matrix with the eigenvectors as columns. The eigenvalues are normalized in the sense that

$$U^T M U = I$$

where I is the identity matrix. We obtain

$$U^T A U = \Lambda \quad \text{or} \quad A = MU\Lambda(MU)^T,$$

where Λ is a matrix with the eigenvalues λ_i on the diagonal. Hence also in terms of the generalized eigenvalue problem (3.6) we obtain the $A-$norm as

$$\|x\|_A^2 = x^T MU\Lambda(MU)^T x$$

and we may define fractional and negative norms in the same manner as (3.5), namely that

$$\|x\|_{A,M,q}^2 = x^T MU\Lambda^q(MU)^T x.$$

Defining the negative and fractional norms in terms of eigenvalues and eigenvectors is convenient for small scale problems, but it is an expensive procedure because eigenvalue problems are computationally demanding. It may, however, be tractable on subdomains, surfaces, or interfaces of larger problems. We also remark that there are other ways of defining fractional and negative norms. For example, one often used technique is via the Fourier series, c.f. e.g. **?**. These different definitions do in general *not* coincide, in particular because they typically have different requirement on the domain or boundary conditions. One should also be careful when employing the above definition with integer $q > 1$, in particular because boundary conditions requirements will deviate from standard conditions in the Sobolev spaces for $q > 1$.

Example 3.2. *Computing the H^1, L^2, and H^{-1} norms*

Let as before $\Omega = (0,1)$ and $u_k = sin(\pi k x)$. Table 8.1 shows the H^1, L^2, and H^{-1} norms as computed with (3.5) *with $q = 1, 0$, and -1, respectively. Comparing the computed norms with the norms L^2 and H^1 norms computed in Example 3.1, we see that the above definition* (3.5) *reproduces the H^1 and L^2 norms with $q = 1$ and $q = 0$, respectively. We also remark that while the H^1 norm increases as k increases, the H^{-1} norm demonstrates a corresponding decrease. Below we show the code for computing these norms.*

$k\backslash norm$	H^1, $q = 1$	L^2, $q = 0$	H^{-1}, $q = -1$
1	2.2	0.71	0.22
10	22	0.71	0.022
100	222	0.71	0.0022

Table 3.2: The L^2, L^7, and H^1 norms of $sin(k\pi x)$ for k=1, 10, and 100.

Python code

```
from dolfin import *
from numpy import matrix, diagflat, sqrt
from scipy import linalg, random

def boundary(x, on_boundary): return on_boundary

mesh = UnitIntervalMesh(200)
V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
bc = DirichletBC(V, Constant(0), boundary)

A, _ = assemble_system(inner(grad(u), grad(v))*dx, Constant(0)*v*dx, bc)
M, _ = assemble_system(u*v*dx, Constant(0)*v*dx, bc)
AA = matrix(A.array())
MM = matrix(M.array())

l, v = linalg.eigh(AA, MM)
v = matrix(v)
l = matrix(diagflat(l))

for k in [1, 10, 100]:
  u_ex = Expression("sin(k*pi*x[0])", k=k)
  u = interpolate(u_ex, V)
  x = matrix(u.vector().array())

  H1_norm = pi*k*sqrt(2)/2
  print "H1 norm of sin(%d pi x) %e (exact)          " % (k, H1_norm)
  H1_norm = sqrt(assemble(inner(grad(u), grad(u))*dx))
```

```
print "H1 norm of sin(%d pi x) %e (|grad(u)|^2)    " % (k, H1_norm)
H1_norm = sqrt(x*AA*x.T)
print "H1 norm of sin(%d pi x) %e (x A x’ )        " % (k, H1_norm)
W = MM.dot(v)
H1_norm = sqrt(x*W*l*W.T*x.T)
print "H1 norm of sin(%d pi x) %e (eig)            " % (k, H1_norm)

print ""

L2_norm = sqrt(2)/2
print "L2 norm of sin(%d pi x) %e (exact)          " % (k, L2_norm)
L2_norm = sqrt(assemble(u**2*dx))
print "L2 norm of sin(%d pi x) %e   |u|^2          " % (k, L2_norm)
L2_norm = sqrt(x*MM*x.T)
print "L1 norm of sin(%d pi x) %e (x M x’ )        " % (k, L2_norm)
W = MM.dot(v)
L2_norm = sqrt(x*W*l**0*W.T*x.T)
print "L2 norm of sin(%d pi x) %e (eig)            " % (k, L2_norm)

print ""

Hm1_norm = sqrt(2)/2/k/pi
print "H^-1 norm of sin(%d pi x) %e (exact)        " % (k, Hm1_norm)
Hm1_norm = sqrt(x*W*l**-1*W.T*x.T)
print "H^-1 norm of sin(%d pi x) %e (eig)          " % (k, Hm1_norm)
Hm1_norm = sqrt(x*MM*linalg.inv(AA)*MM*x.T)
print "H^-1 norm of sin(%d pi x) %e (x inv(A) x’) " % (k, Hm1_norm)
```

Remark 3.8.1. ***Norms for*** $|q| > 1$**.**
The norm (3.5) *is well defined for any* $|q| > 1$*, but will not correspond to the corresponding Sobolev spaces.*

Remark 3.8.2. ***The standard definition of a dual norm***
Let $(\cdot,\cdot)_A$ *be an inner product over the Hilbert space* V*. The norm of the dual space is then defined by*

$$\|f\|_{A^*} = \sup_{v\in V} \frac{(f,v)}{(v,v)_A}.$$

For example, the H^{-1} *norm is defined as*

$$\|f\|_{-1} = \sup_{v\in H^1} \frac{(f,v)}{(v,v)_1}.$$

3.9 Exercises

Exercise 3.1. *Compute the* H^1 *and* L^2 *norms of a random function with values in* $(0,1)$ *on meshes representing the unit interval of with* 10*,* 100*, and* 1000 *cells.*

Exercise 3.2. *Compute the* H^1 *and* L^2 *norms of* $\sin(k\pi x)$ *on the unit interval analytically and compare with the values presented in Table 3.2.*

Exercise 3.3. *Compute the* H^1 *and* L^2 *norms of the hat function in Picture 3.2.*

Exercise 3.4. *Consider the following finite element function* u *defined as*

$$u = \begin{cases} \frac{1}{h}x - \frac{1}{h}(0.5-h), & x = (0.5-h, 0.5) \\ -\frac{1}{h}x + \frac{1}{h}(0.5-h), & x = (0.5, 0.5+h) \\ 0, & \textit{elsewhere} \end{cases}$$

That is, it corresponds to the hat function in Figure 3.2, where $u(0.5) = 1$ *and the hat function is zero every where in* $(0, 0.5 - h)$ *and* $(0.5 + h, 1)$*. Compute the* H^1 *and* L^2 *norms of this function analytically, and the* L^2*,* H^1 *and* H^{-1} *norms numerically for* $h = 10$, 100 *and* 1000.

Exercise 3.5. *Let* $\Omega = (0, 1)$ *then for all functions in* $H^1(\Omega)$ *Poincaré's inequality states that*

$$|u|_{L^2} \leq C|\frac{\partial u}{\partial x}|_{L^2}$$

Use this inequality to show that the H^1 *semi-norm defines a norm equivalent with the standard* H^1 *norm on* $H^1_0(\Omega)$.

4 Finite element error estimate

By Anders Logg, Kent–Andre Mardal

4.1 Ingredients

We have used the FEM to compute an approximate solution, u_h, of a PDE. Fundamental question: How large is the error $e = u - u_h$? To be able to estimate the error, we need some ingredients:

1. Galerkin orthogonality
2. Interpolation estimates
3. Coercivity (more generally: inf–sup)

We will also state the Fundamental theorem of numerical analysis

Theorem 4.1. *Consistency and stability ⇔ convergence.*

4.1.1 Galerkin orthogonality

Let us look at the "abstract" weak formulation of a PDE,

$$a(u, v) = L(v) \quad \forall\, v \in V. \tag{4.1}$$

Now we let $u_h \in V_h$, where V_h is a finite dimensional function space,

$$a(u_h, v) = L(v) \quad \forall\, v \in V_h \subset V. \tag{4.2}$$

By subtracting (4.2) from (4.1), we get the Galerkin orthogonality:

$$\boxed{a(u - u_h, v) = 0} \quad \forall\, v \in V_h \subset V. \tag{4.3}$$

4.1.2 Interpolation estimates

First, let us note that

$$\|u - u_h\| \geqslant \inf_{v \in V_h} \|u - v\|, \tag{4.4}$$

for some norm. We need to be able to estimate $\inf_{v \in V_h} \|u - v\|$ or at least get a sharp upper bound. We will do this by estimating $\|u - v\|$ for a particular (a good) choice of v!

Let $\pi_h u$ be a piecewise constant approximation of $u(x)$ (1D). Then for $x \in (x_{i-1}, x_i]$, from the theory of Taylor expansion, we have

$$u(x) = \underbrace{u\left(\overbrace{\frac{x_{i-1}+x_i}{2}}^{\bar{x}_i}\right)}_{\equiv \pi_h u} + \int_{\bar{x}_i}^{x} u'(y)\,\mathrm{d}y.$$

which leads to

$$|u - \pi_h u| = |\int_{\bar{x}_i}^{x} u'(y)\,\mathrm{d}y\,| \,.$$

Let us consider the L^2–norm. Then,

$$\begin{aligned}\|u-\pi_h u\|_{L^2}^2 = \int_a^b (u-\pi_h u)^2\,\mathrm{d}x &= \sum_i \int_{x_{i-1}}^{x_i} (u-\pi_h u)^2\,\mathrm{d}x \\ &= \sum_i \int_{x_{i-1}}^{x_i} \left(\int_{\bar{x}_i}^{x} u'(y)\,\mathrm{d}y\right)^2 \mathrm{d}x\end{aligned}$$

We multiply the integrand by one and use Cauchy–Schwartz inequality.

$$\begin{aligned}\|u-\pi_h u\|^2 &= \sum_i \int_{x_{i-1}}^{x_i} \left(\int_{\bar{x}_i}^{x} 1\cdot u'(y)\,\mathrm{d}y\right)^2 \mathrm{d}x \\ &\leqslant \sum_i \int_{x_{i-1}}^{x_i} \left(\left(\int_{\bar{x}_i}^{x} 1^2\,\mathrm{d}y\right)^{1/2} \cdot \left(\int_{\bar{x}_i}^{x} (u'(y))^2\,\mathrm{d}y\right)^{1/2}\right)^2 \mathrm{d}x \\ &= \sum_i \int_{x_{i-1}}^{x_i} |\int_{\bar{x}_i}^{x} 1^2\,\mathrm{d}y| \cdot |\int_{\bar{x}_i}^{x} (u'(y))^2\,\mathrm{d}y|\,\mathrm{d}x \\ &= \sum_i \int_{x_{i-1}}^{x_i} |x - \frac{x_{i-1}+x_i}{2}| \cdot \int_{\bar{x}_i}^{x} (u'(y))^2\,\mathrm{d}y\,\mathrm{d}x \\ &\leqslant \sum_i \frac{h_i}{2} \int_{x_{i-1}}^{x_i} \int_{x_{i-1}}^{x_i} (u'(y))^2\,\mathrm{d}y\,\mathrm{d}x \\ &= \sum_i \frac{h_i^2}{2} \int_{x_{i-1}}^{x_i} (u'(y))^2\,\mathrm{d}y \\ &\leqslant \frac{1}{2} \int_a^b (h\,u'(y))^2\,\mathrm{d}y = \frac{1}{2}\|hu'\|_{L^2}^2,\end{aligned}$$

where $h_i = x_i - x_{i-1}$ and $h = \max_i h_i$. Thus, we have found an interpolation estimate

$$\boxed{\|u-\pi_h u\|_{L^2} \leqslant \frac{1}{\sqrt{2}}\|hu'\|_{L^2}.} \tag{4.5}$$

In general, one can prove that

$$\boxed{\|(\frac{\mathrm{d}}{\mathrm{d}x})^p(u-\pi_h u)\|_{L^2} \leqslant C(p,q)\|h^{q+1-p}(\frac{\mathrm{d}}{\mathrm{d}x})^{q+1}u\|_{L^2},} \tag{4.6}$$

where $\pi_h u$ is an approximation (interpolant) of degree q. $C(p,q)$ is a constant depending only on p and q.

4.1.3 Coercivity

Definition 4.1. *Coercive*
A bilinear form $a : H \times H \to \mathbb{R}$ is called coercive if there exists a constant $\alpha > 0$ such that

$$a(v,v) \geqslant \alpha\|v\|_V^2 \quad \forall\, v \in V.$$

$\|\cdot\|_V$ is the norm we will use to estimate the error.
We now have all the ingredients we need to estimate the error!

4.2 Error estimates

There are two kinds of error estimate and they are both essential!

1. *a priori*: $e = e(u)$

2. *a posteriori*: $e = e(u_h)$

4.2.1 A priori error estimate in energy norm

Assume that $a(\cdot,\cdot)$ is a symmetric and coercive bilinear form. Then $a(\cdot,\cdot)$ is an inner product and $\|v\|_E = \sqrt{a(v,v)}$ is a norm which we will call the energy norm. Let us look at the error in the energy norm. Let $v \in V_h$, then

$$\begin{aligned}
\|e\|_E^2 &= a(e,\, e) = a(e,\, u-u_h) && (4.7)\\
&= a(e,\, u-v+v-u_h) && (4.8)\\
&= a(e,\, u-v) + a(e,\, \underbrace{v-u_h}_{\in V_h}) && (4.9)\\
&= a(e,\, u-v) + 0 \quad \text{(from Galerkin Orthogonality)} && (4.10)\\
&\leqslant \sqrt{a(e,\, e)}\sqrt{a(u-v,\, u-v)} && (4.11)\\
&= \|e\|_E\, \|u-v\|_E. && (4.12)
\end{aligned}$$

We have used Cauchy–Schwartz inequality ones. Now we divide both sides by $\|e\|_E$ and obtain

$$\|u-u_h\|_E \leqslant \|u-v\|_E \quad \forall\, v \in V_h. \tag{4.13}$$

Thus, the FEM solution is the optimal solution in the energy Norm! We combine this with the interpolation estimate (4.5), by setting $v = \pi_h u$:

$$\begin{aligned} \|u - u_h\|_E &\leqslant \|u - \pi_h u\|_E && (4.14)\\ &\leqslant C(p,q)\|h^{q+1-p}(\frac{\mathrm{d}}{\mathrm{d}x})^{q+1}u\|. && (4.15) \end{aligned}$$

For example in the Poisson problem with piecewise linear functions ($q = 1$), we have

$$\|v\|_E = \sqrt{\int_\Omega |\nabla v|^2 \,\mathrm{d}x}.$$

The *a priori*estimate (4.15) becomes

$$\|e\|_E \leqslant C\|hD^2u\|. \tag{4.16}$$

A priori error estimate in the V–norm that does not assume symmetry. From coersivity we get

$$\begin{aligned} \|e\|_V^2 &\leqslant \frac{1}{\alpha}a(e,e) && (4.17)\\ &= \frac{1}{\alpha}a(e, u - v + v - u_h) && (4.18)\\ &= \frac{1}{\alpha}a(e, u - v) \quad \text{(from Galerkin Orthogonality)} && (4.19)\\ &\leqslant \frac{C}{\alpha}\|e\|_V\|u - v\|_V. && (4.20) \end{aligned}$$

Here we assumed boundedness of a. By dividing both sides by $\|e\|_V$, we get an inequality known as *Cea's lemma.*

$$\|e\|_V \leqslant \frac{C}{\alpha}\|u - v\|_V \quad \forall v \in V_h \tag{4.21}$$

As before, we can use an interpolation estimate to obtain

$$\boxed{\|e\|_V \leqslant \frac{C \cdot C(q,p)}{\alpha}\|h^{q+1-p}(\frac{\mathrm{d}}{\mathrm{d}x})^{q+1}u\|}. \tag{4.22}$$

4.2.2 A posteriori error estimate for the Poisson problem in the energy norm

We will now derive an *a posteriori*error estimate for the Poisson problem. To do this we need the following interpolation estimates:

$$\begin{aligned} \|e - \pi_h e\|_T &\leqslant C\, h_T\|\nabla e\|_T, && (4.23)\\ \|e - \pi_h e\|_{\partial T} &\leqslant C\sqrt{h_T}\|\nabla e\|_{\omega_T}, && (4.24) \end{aligned}$$

where ω_T is the patch of of elements surrounding T. Note that the constant C will change throughout the derivation. We will also need Cauchy's inequality,

$$ab \leqslant \delta\, a^2 + \frac{b^2}{4\delta}, \quad a, b, \delta > 0. \tag{4.25}$$

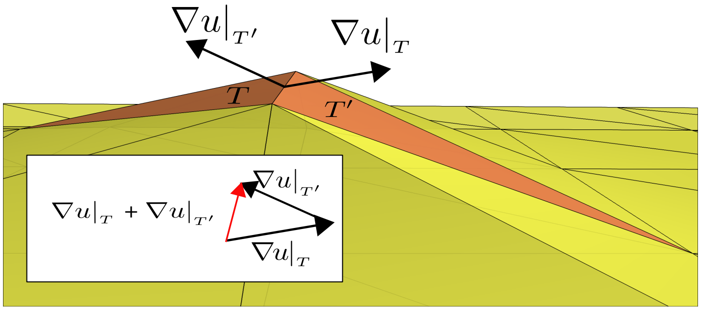


Figure 4.1: Illustration of a "jump" at two neighboring facets.

Recall that the energy–norm for the Poisson problem is

$$\|v\|_E = \sqrt{\int_\Omega |\nabla v|^2 \, \mathrm{d}x}.$$

Let us begin the derivation,

$$\begin{aligned}
\|e\|_E^2 &= a(e,e) && (4.26)\\
&= a(e,e) - \underbrace{a\left(e, \pi_h e\right)}_{=0} && (4.27)\\
&= a(e, e - \pi_h e) && (4.28)\\
&= \int_\Omega \nabla e \cdot \nabla (e - \pi_h e) \, \mathrm{d}x && (4.29)\\
&= \sum_{T \in \mathcal{T}_h} \int_T \nabla e \cdot \nabla (e - \pi_h e) \, \mathrm{d}x && (4.30)\\
&= \sum_{T \in \mathcal{T}_h} \int_T -\Delta e (e - \pi_h e) \, \mathrm{d}x + \int_{\partial T} \partial_n e (e - \pi_h e) dS && (4.31)\\
&= \sum_{T \in \mathcal{T}_h} \int_T (\underbrace{-\Delta u}_{=f} + \Delta u_h)(e - \pi_h e) \, \mathrm{d}x + \int_{\partial T} \partial_n e (e - \pi_h e) dS && (4.32)\\
&= \sum_{T \in \mathcal{T}_h} \int_T \underbrace{(f + \Delta u_h)}_{\equiv R} (e - \pi_h e) \, \mathrm{d}x + \sum_S \int_{\partial S} \underbrace{(\partial_{n^+} e + \partial_{n^-} e)}_{-[\partial_n u_h]} (e - \pi_h e) dS && (4.33)\\
&= \sum_{T \in \mathcal{T}_h} \int_T R(e - \pi_h e) \, \mathrm{d}x - \frac{1}{2} \int_{\partial T} \left[\partial_n u_h\right] (e - \pi_h e) dS. && (4.34)
\end{aligned}$$

Let us explain a bit before we continue. In equation (4.27) we added the term $a(e, \pi_h e)$, which from Galerkin orthogonality is zero (since $\pi_h e \in V_h$). We used integration by part to get equation (4.31). In the first term one the right-hand side of equation (4.33), we insert the residual, $R \equiv f + \Delta u_h$. For the second term, we look at surface integral over two neighboring facets (S), for all S, see figure 4.1. There normal components, n, will be pointing in opposite direction of each other and we get,

$$\partial_{n^+} e + \partial_{n^-} e = n^+ \cdot \nabla^+ e + n^- \cdot \nabla^- e = n^+ \cdot (\nabla^+ e - \nabla^- e) = [\partial_n e] = - [\partial_n u_h] \, . \qquad (4.35)$$

$[\partial_n u_h]$ is called a jump. Note that until now, we have only used equalities. Let us look at equation (4.34) in two terms.

$$\begin{aligned} A &\equiv \int_T R(e - \pi_h e)\,\mathrm{d}x && (4.36)\\ &\leqslant \|R\|_T\,\|e-\pi_h e\|_T && (4.37)\\ &\leqslant \|R\|_T\,C\,h_T\|\nabla e\|_T && (4.38)\\ &\leqslant \frac{C\,h_T^2}{2}\|R\|_T^2 + \frac{1}{2}\|\nabla e\|_T^2 && (4.39)\end{aligned}$$

and

$$\begin{aligned} B &\equiv \frac{1}{2}\int_{\partial T}[\partial_n u_h]\,(e-\pi_h e)dS && (4.40)\\ &\leqslant \frac{1}{2}\|\,[\partial_n u_h]\,\|_{\partial T}\,\|(e-\pi_h e)\|_{\partial T} && (4.41)\\ &\leqslant \|\,[\partial_n u_h]\,\|_{\partial T}\frac{C\sqrt{h_T}}{2}\|\nabla e\|_{\omega_T} && (4.42)\\ &\leqslant \frac{C\,h_T}{\epsilon}\|\,[\partial_n u_h]\,\|_{\partial T}^2 + \epsilon\|\nabla e\|_{\omega_T}^2 && (4.43)\end{aligned}$$

In equation (4.37) and (4.41), we used Cauchy–Schwarz inequality. For equation (4.38) and (4.42), we used the interpolation estimates (4.23) and (4.24) respectively. Finally we used Cauchy's inequality with $\delta = \frac{1}{2}$ for equation (4.39) and $\delta = \frac{\epsilon}{4}$ for equation (4.43). Let us sum up what we have so far:

$$\begin{aligned} \|e\|_E^2 &= \sum_{T\in\mathcal{T}_h} A - B && (4.44)\\ &\leqslant \sum_{T\in\mathcal{T}_h} A + B && (4.45)\\ &\leqslant \sum_{T\in\mathcal{T}_h}\frac{1}{2}\|\nabla e\|_T^2 + \epsilon\|\nabla e\|_{\omega_T}^2 + \frac{C\,h_T^2}{2}\|R\|_T^2 + \frac{C\,h_T}{\epsilon}\|\,[\partial_n u_h]\,\|_{\partial T}^2. && (4.46)\\ & && (4.47)\end{aligned}$$

Now we note that

$$\sum_{T\in\mathcal{T}_h}\|\nabla e\|_T^2 = \|e\|_E^2 \quad \text{and} \quad \sum_{T\in\mathcal{T}_h}\|\nabla e\|_{\omega_T}^2 \leqslant N\|e\|_E^2, \tag{4.48}$$

where N is the maximum number of surrounding elements. We use this and get

$$\|e\|_E^2 \leqslant \left(\frac{1}{2}+\epsilon N\right)\|e\|_E^2 + \sum_{T\in\mathcal{T}_h}\frac{C\,h_T^2}{2}\|R\|_T^2 + \frac{C\,h_T}{\epsilon}\|\,[\partial_n u_h]\,\|_{\partial T}^2 \tag{4.49}$$

$$\left(\frac{1}{2}-\epsilon N\right)\|e\|_E^2 \leqslant \sum_{T\in\mathcal{T}_h}\frac{C\,h_T^2}{2}\|R\|_T^2 + \frac{C\,h_T}{\epsilon}\|\,[\partial_n u_h]\,\|_{\partial T}^2. \tag{4.50}$$

Finally we chose ϵ such that $(\frac{1}{2} - \epsilon N) > 0$ and we get the *a posteriori*error estimate:

$$\boxed{\|e\|_E \leqslant C \left(\sum_T h_T^2 \|R\|_T^2 + h_T \| \left[\partial_n u_h\right] \|_{\partial T}^2 \right)^{\frac{1}{2}} \equiv E} \tag{4.51}$$

4.3 Adaptivity

In many applications we need the error to be less then a given tolerance (TOL). The error will typically be large at some parts of the domain and small at other parts of the domain. We do not want to refine[1] all the elements in $\mathcal{T}$, since this will require a lot more computational power and memory. Instead we want to only refine the elements where the error is big. Let us first rewrite the *a posteriori*error estimate (4.51) in a more general form,

$$\|e\|_E \leqslant C \left(\sum_T \gamma_T^2 \right)^{\frac{1}{2}} \equiv E. \tag{4.52}$$

We consider two alternatives,

1. Given $TOL > 0$, choose $\mathcal{T}$ such that the computational norm is minimized and $\|e\|_V \leqslant TOL$.
2. Given $TOL > 0$, choose $\mathcal{T}$ such that $|\mathcal{T}|$ is minimized and $E \leqslant TOL$.

Both methods are difficult to solve. Here is an algorithm for adaptivity.

- Choose $\mathcal{T}$
- Compute u_h on $\mathcal{T}$
- Compute E for u_h
- While $E > TOL$:

  i Refine all cells where γ_T is large
  ii Compute u_h on $\mathcal{T}$
  iii Compute E for u_h

Exercise 4.1.
Let $\{\phi_i\}_{i=0}^m$ be the standard nodal basis functions for continuous piecewise linear approximation on $\Omega = (0,1)$ with constant mesh size $h = 1/m$.

(a) Take $m = 10$. Draw a picture of the basis functions ϕ_0, ϕ_5 and ϕ_{10}.

(b) Draw a similar picture of the derivatives ϕ_0', ϕ_5' and ϕ_{10}'.

Exercise 4.2. *Consider the equation*

$$\begin{cases} -u'' + u = f \text{ in } (0,1), \\ \quad u(0) = 0, \\ \quad u(1) = 0. \end{cases} \tag{4.53}$$

[1] By refining we mean that the elements T are made smaller

(a) *Write down a finite element method for this equation using standard continuous piecewise linear polynomials. Show that the degrees of freedom U for the solution $u = \sum_{i=1}^{m-1} U_i\phi_i$ may be obtained by solving the linear system $(A + M)U = b$. The matrix A is often called the* stiffness matrix *and M is called the* mass matrix*.*

(b) *Compute the 9×9 matrices A and M for $m = 10$.*

Demonstrate that if $f \in V_h$, then the mass matrix M may be used to compute the right-hand side vector b (the load vector*) for the finite element discretization of (4.53).*

Exercise 4.3.

Consider the following partial differential equation:

$$\begin{cases} -u'' = f \text{ in } (0,1), \\ u'(0) = 0, \\ u'(1) = 0. \end{cases} \tag{4.54}$$

(a) *Explain why there is something wrong with this equation (why it is not* well-posed*). Consider both* uniqueness *and* existence of solutions*.*

(b) *If you would implement a (standard) finite element method for this equation, what would happen? How would you notice that something was wrong?*

Exercise 4.4. *Consider the following partial differential equation:*

$$\begin{cases} -\nabla \cdot (a\nabla u) = f \text{ in } \Omega, \\ \qquad\qquad u = 0 \text{ on } \partial\Omega, \end{cases} \tag{4.55}$$

where $a = a(x)$ is a positive definite $n \times n$ matrix at each $x \in \Omega$. Prove that the stiffness matrix A (for a suitable finite element space on Ω) is also positive definite, and explain why A is only positive semidefinite for homogeneous Neumann conditions.

Implement a simple Python program that computes the stiffness and mass matrices on $\Omega = (0,1)$ for any given $m \geq 2$, where m is the number of intervals partitioning $(0,1)$. Use A and M to solve equation (4.53) for $f(x) = \sin(5\pi x)$. Plot the solution and compare with the analytical solution. Demonstrate that the approximate solution converges to the exact solution when the mesh is refined. What is the convergence rate in L^2? What is the convergence rate in H^1?

Hint*: Use* `numpy.array` *for storing matrices and vectors,* `numpy.linalg.solve` *to solve the linear system and* `pylab.plot` *to plot the solution. Also note that you may approximate $b_i = \int_\Omega \phi_i f \,\mathrm{d}x$ by $f(x_i)\int_\Omega \phi_i \,\mathrm{d}x$.*

Implement a simple Python program that computes the stiffness matrix A on a uniform triangular mesh of the unit square $\Omega = (0,1) \times (0,1)$. Use A to solve Poisson's equation $-\Delta u = f$ for $f = 2\pi^2 \sin(\pi x)\, \sin(\pi y)$ and homogeneous Dirichlet conditions. Plot the solution and compare with the analytical solution. Demonstrate that the approximate solution converges to the exact solution when mesh is refined. What is the convergence rate in L^2? What is the convergence rate in H^1?

Exercise 4.5. *Estimate the H^k Sobolev norm of $u = \sin(k\pi x)$ as a function of k .*

Exercise 4.6. *Solve the problem $-\Delta u = f$ with homogenous boundary conditions on the unit interval for the case where the analytical solution is $u = \sin(k\pi x)$ and f is given as $-\Delta u$. As we learned in this chapter,*

$$\|u - u_h\|_1 \leq Ch^p\|u\|_{p+1}.$$

Estimate C in numerical estimates for $k = 1, 10, 100$ *on meshes with* 100, 1000, *and* 10000 *elements and validate the error estimate.*

Remark: Use `errornorms` *in FEniCS and represent the analytical solution in a higher order space in order to avoid super convergence.*

Exercise 4.7. *Consider the error of the problem in Exercise 4.6 in* L_2, L_∞, *and* L_1 *norms. Is the order of the approximation the same? Hint: use the least square method to estimate* C_x *and* α_x *in*

$$\|u - u_h\|_x \leq C_x h^{\alpha_x},$$

where x denotes the norm and C_x *depends on u in contrast to Exercise 4.6. Hence, it is advisable to determine* C_x *and* α_x *for a given k and then change k.*

Exercise 4.8. *Consider the error of the problem in Exercise 4.6 and 4.7 in* L_2 *and* H^1 *norms, but determine the rate of convergence numerically with respect to the polynomial order of the finite element method. That is, use the least square method to estimate* C_p *and* α_p *in*

$$\|u - u_h\|_1 \leq C_p h^{\alpha_p}.$$

Here, C_p *depends on u in contrast to Exercise 4.6. Hence, it is advisable to determine* C_p *and* α_p *for a given k and then change k.*

Exercise 4.9. *Consider the same problem as in 4.6 in 3D (or 2D if your computer does not allow a proper investigation in 3D). Assume that you tolerate a* H^1 *error of* $1.0e-2$. *What polynomial order of the Lagrange finite element gives you the answer in minimal amount of computational time? Re-do the experiments with tolerance* $1.0e-4$ *and* $1.0e-6$

5 Finite element function spaces

By Anders Logg, Kent–Andre Mardal

Finite element function spaces (v_h) are constructed by patching together local function spaces, $\mathcal{V} = \mathcal{V}(T)$, defines on finite subsets of the domain Ω.

Example: Piecewise linear in 1–D
Figure 5.1 shows a function $u_h \in V_h$. This is a linear combination of basis function for first order Lagrange elements in 1–D. Figure 5.2 show the (global) basis functions of this function space and figure 5.3 show the local basis function on an element T and $T^{'}$.

Example: Piecewise linear in 2–D
Figure 5.4 shows a linear combination of piecewise linear basis functions forming a function u_h, on a triangle. The different color indicate where the different baisis functions contribute. Figure 5.5 shows a (global) basis funcitons and figure 5.6 show the local basis function on an element T and T'.

5.1 The finite element definition

General idea: Define a function space on each local subdomain and patch together the local function space, to create a global function space with the desired continuity. An definition of the finite element was given by Ciarlet in 1975. This serves as our formal definition.

Definition 5.1. *Finite element (Ciarlet 1975)*
A finite element is a triple $(T, \mathcal{V}, L)$, where

i The domain T is a bounded, closed subset of $\mathbb{R}^d$ (for $d = 1, 2, 3, \ldots$) with nonempty interior and piecewise smooth boundary;

ii The space $\mathcal{V} = \mathcal{V}(T)$ is a finite dimensional function space on T of dimension n;

iii The set of degrees of freedom (nodes) $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ is a basis for the dual space $\mathcal{V}'$; that is, the space of bounded linear functionals on $\mathcal{V}$.

Example: (T)
Figure 5.7 shows different kinds of domains, T, for different dimensions, $d = 1, 2, 3$. **Example:** $(\mathcal{V})$

- $\mathcal{V} = \mathcal{P}_q(T) = \{\text{polynomials on } T \text{ of degree } \leqslant q\}$
- $\mathcal{V} = [\mathcal{P}_q(T)]^d$

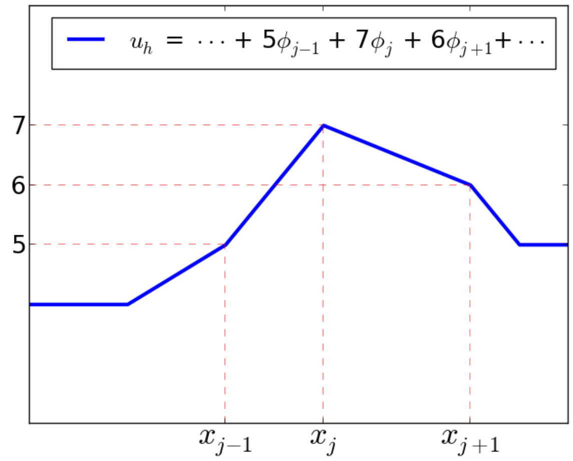


Figure 5.1: Function that is composted of a linear combinatin of basis functions

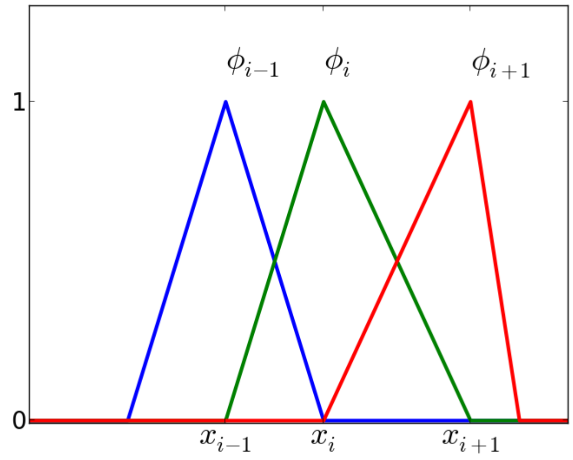


Figure 5.2: Basis functions (global)

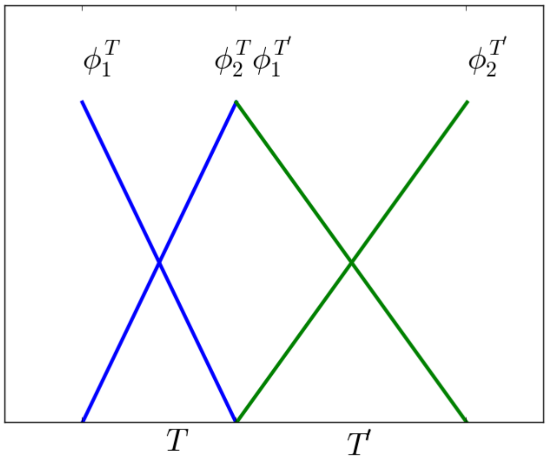


Figure 5.3: Local basis functions

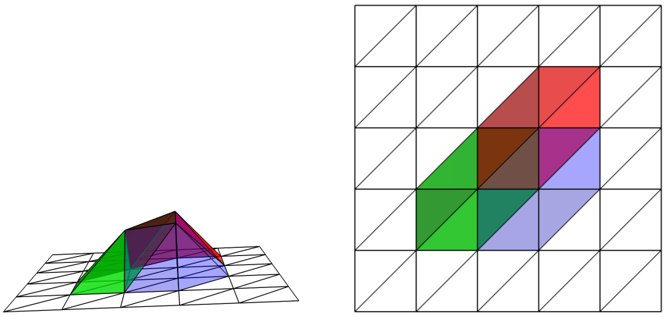


Figure 5.4: Function on a triangle that is composted of a linear combinatin of basis functions. The left figure shows a side view while the right figure shows a view from above.

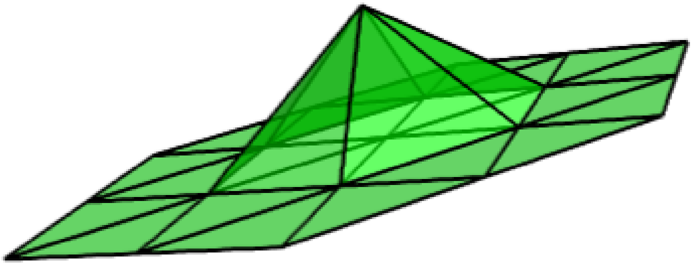


Figure 5.5: Basis functions (global) 2–D

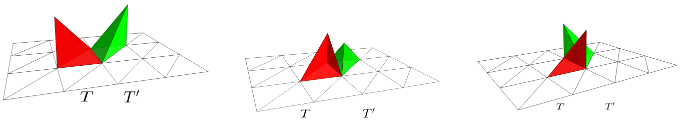


Figure 5.6: Local basis functions 2–D

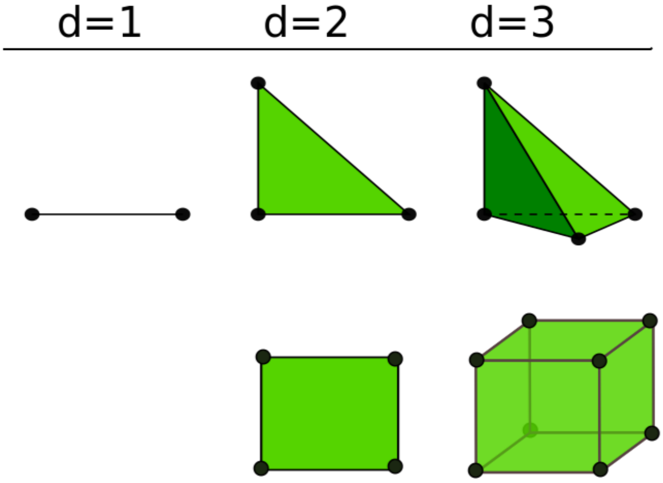


Figure 5.7: Illustration of different domains, T.

- $\mathcal{V} =$ subspace of $\mathcal{P}_q(T)$

Example: $(\mathcal{L})$

- $\mathcal{L}(v) = v(\bar{x})$
- $\mathcal{L}(v) = \int_T v(x)\,\mathrm{d}x$
- $\mathcal{L}(v) = \int_S v \cdot n\, dS$

Is the standard piecewise linear element, P_1, a finite element?

i T is a interval, triangle or tetrahedron: ok

ii $\mathcal{V} = \{v \,:\, v(x) = a + bx\} \equiv \mathcal{P}_1(T)$
dim $\mathcal{V} = n = d + 1$: ok

iii $\mathcal{L} = \{l_1, \ldots, l_n\}$, where $l_i(v) = v(x^i)$ for $i = 1, \ldots, n$.
Is this a basis?

To be able to show that $\mathcal{L}$ is a basis, we need the following lemma.

Lemma 5.1. *Unisolvence*
$\mathcal{L}$ is a basis for the dual space $\mathcal{V}'$, if and only if $\mathcal{L}v = 0$ implies $v = 0$. This can be expressed as:

$$\mathcal{L} \textit{ is basis for } \mathcal{V}' \Leftrightarrow (\mathcal{L}v = 0 \Rightarrow v = 0)$$

Proof. Let $\{\phi_i\}_{i=1}^n$ be a basis for $\mathcal{V}$, take $\ell \in \mathcal{V}'$ and take $v = \sum_{j=1}^n \beta_j \phi_j \in V$. First we look at the left-hand side;

$$\begin{aligned} \mathcal{L} \text{ is basis for } \mathcal{V}' &\Leftrightarrow \exists!\, \alpha \in \mathbb{R}^n \,:\, \ell = \sum_{j=1}^n \alpha_j \ell_j \\ &\Leftrightarrow \exists!\, \alpha \in \mathbb{R}^n \,:\, \underbrace{\ell\phi_i}_{=b_i} = \sum_{j=1}^n \alpha_j \underbrace{\ell_j\phi_i}_{=A_{ij}} \text{ for } i = 1, \ldots, n \\ &\Leftrightarrow \exists!\, \alpha \in \mathbb{R}^n \,:\, A\alpha = b \\ &\Leftrightarrow A \text{ is invertible} \end{aligned}$$

Now we look at the right-hand side;

$$\begin{aligned} (\mathcal{L}v = 0 \Rightarrow v = 0) &\Leftrightarrow \ell_i \sum_j \beta_j \phi_j = 0 \text{ for } i = 1, \ldots, n \ \Rightarrow \beta = 0 \\ &\Leftrightarrow \sum_j \beta_j \underbrace{\ell_i\phi_j}_{=A_{ji}} \text{ for } i = 1, \ldots, n \ \Rightarrow \beta = 0 \\ &\Leftrightarrow A^T\beta = 0 \ \Rightarrow \beta = 0 \\ &\Leftrightarrow A^T \text{ is invertible} \\ &\Leftrightarrow A \text{ is invertible} \end{aligned}$$

To sum up:

$$\mathcal{L} \text{ is basis for } \mathcal{V}' \Leftrightarrow A \text{ is invertible} \Leftrightarrow (\mathcal{L}v = 0 \Rightarrow v = 0).$$

□

We can now check if P_1 is a finite element. Take v on a triangle, set $v = 0$ at each corner. This leads to $v = 0$ for linear functions. P_1 is a finite element.

Definition 5.2. *Nodal basis*
The nodal basis $\{\phi_i\}_{i=1}^n$ for a finite element $(T, \mathcal{V}, \mathcal{L})$ is the unique basis satisfying

$$\ell_i(\phi_J) = \delta_{ij}.$$

A nodal basis has the desired property that if, $u_h = \sum_{j=1}^n u_j \phi_j$, then $\ell_i(u_h) = u_i$.
Example:
We look at P_1 elements on triangle with corners at x_1, x_2 and x_3,

$$\begin{aligned} x_1 &= (0,0), & \ell_1 v &= v(x_1) & \phi_1(x) &= 1 - x_1 - x_2 \\ x_2 &= (1,0), & \ell_2 v &= v(x_2) & \phi_2(x) &= x_1 \\ x_3 &= (0,1), & \ell_3 v &= v(x_3) & \phi_3(x) &= x_2. \end{aligned}$$

from this we see that ϕ_1, ϕ_1 and ϕ_1 are a nodal basis.

Computing the nodal basis: Let $\{\psi_i\}_{i=1}^n$ be any basis for $\mathcal{P}$ and let $\{\psi_i\}_{i=1}^n$ be its nodal basis. Then,

$$\begin{aligned} \sum_{i=1}^n \alpha_{jk} \psi_k &= \phi_i \\ \ell_i(\sum_{i=1}^n \alpha_{jk} \psi_k) &= \delta_{ij} \\ \underbrace{\ell_i(\psi_k)\alpha_{jk}}_{A_{ij}} &= \delta_{ij} \\ A\alpha^T &= I \end{aligned}$$

A is the *generalized Vandermonde matrix*. Solving for α gives the nodal basis!

5.1.1 Conforming

We will introduce some important function spaces:

$$H^1(\Omega) = \{v \in L^2(\Omega) \ : \ \nabla v \in L^2(\Omega)\} \tag{5.1}$$

$$H(\text{div}; \Omega) = \{v \in L^2(\Omega) \ : \ \nabla \cdot v \in L^2(\Omega)\} \tag{5.2}$$

$$H(\text{curl}; \Omega) = \{v \in L^2(\Omega) \ : \ \nabla \times v \in L^2(\Omega)\} \tag{5.3}$$

Note:

$$H^1(\Omega) \subset H(\mathrm{div};\Omega) \approx \{v \,:\, \text{normal component } \in C^0\}$$
$$H^1(\Omega) \subset H(\mathrm{curl};\Omega) \approx \{v \,:\, \text{tangential component } \in C^0\}$$

If a finite element function space is a subspace of function space V, we call it V-conforming. Example, the Lagrange elements are H^1-conforming, $\mathrm{CG}_q(\mathcal{T}) \subset H^1(\Omega)$.

5.2 Common elements

Let us have a look at some common elements. First we will look at the most common group of elements, *the continues Lagrange elements.* These are also know as, continues Galerkin elements or P_q elements.

Definition 5.3 (Lagrange element)**.** *The Lagrange element* (CG_q) *is defined for* $q = 1, 2, \ldots$ *by*

$$T \in \{\text{interval}, \text{triangle}, \text{tetrahedron}\}, \tag{5.4}$$
$$\mathcal{V} = \mathcal{P}_q(T), \tag{5.5}$$
$$\ell_i(v) = v(x^i), \quad i = 1, \ldots, n(q), \tag{5.6}$$

where $\{x^i\}_{i=1}^{n(q)}$ *is an enumeration of points in* T *defined by*

$$x = \begin{cases} i/q, & 0 \leqslant i \leqslant q, & T \text{ interval}, \\ (i/q, j/q), & 0 \leqslant i+j \leqslant q, & T \text{ triangle}, \\ (i/q, j/q, k/q), & 0 \leqslant i+j+k \leqslant q, & T \text{ tetrahedron}. \end{cases} \tag{5.7}$$

The dimension of the Lagrange finite element thus corresponds to the dimension of the complete polynomials of degree q on T and is

$$n(q) = \begin{cases} q+1, & T \text{ interval}, \\ \frac{1}{2}(q+1)(q+2), & T \text{ triangle}, \\ \frac{1}{6}(q+1)(q+2)(q+3), & T \text{ tetrahedron}. \end{cases} \tag{5.8}$$

Figure 5.8 show the Lagrange elements for different dimensions and how the nodal points are placed.

Now we will look at some $H(\mathrm{div})$-conforming elements. First up is the Raviart–Thomas RT_q elements.

Definition 5.4 (Raviart–Thomas element)**.** *The Raviart–Thomas element* (RT_q) *is defined for* $q = 1, 2, \ldots$ *by*

$$T \in \{\text{triangle}, \text{tetrahedron}\}, \tag{5.9}$$
$$\mathcal{V} = [\mathcal{P}_{q-1}(T)]^d + x\mathcal{P}_{q-1}(T), \tag{5.10}$$
$$\mathcal{L} = \begin{cases} \int_f v \cdot n\, p \, \mathrm{d}s, & \textit{for a set of basis functions } p \in \mathcal{P}_{q-1}(f) \textit{ for each facet } f, \\ \int_T v \cdot p \, \mathrm{d}x, & \textit{for a set of basis functions } p \in [\mathcal{P}_{q-2}(T)]^d \textit{ for } q \geqslant 2. \end{cases} \tag{5.11}$$

The dimension of RT_q is

$$n(q) = \begin{cases} q(q+2), & T \text{ triangle}, \\ \frac{1}{2}q(q+1)(q+3), & T \text{ tetrahedron}. \end{cases} \tag{5.12}$$

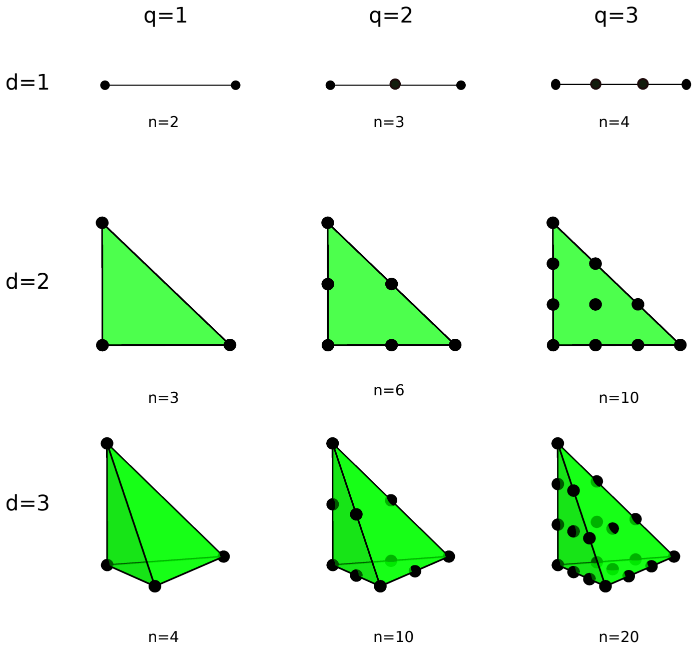


Figure 5.8: The Lagrange (CG_q) elementes. q is the order of the elements, d is the dimention and n is the number of degrees of freedom.

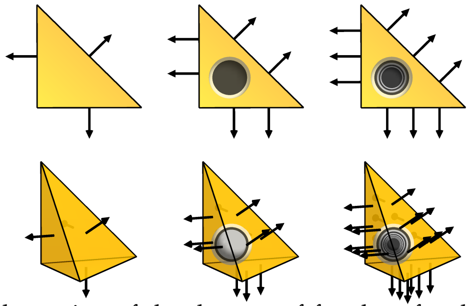


Figure 5.9: Illustration of the degrees of freedom for the first, second and third degree Raviart–Thomas elements on triangles and tetrahedra. The degrees of freedom are moments of the normal component against $\mathcal{P}_{q-1}$ on facets (edges and faces, respectively) and, for the higher degree elements, interior moments against $[\mathcal{P}_{q-2}]^d$.

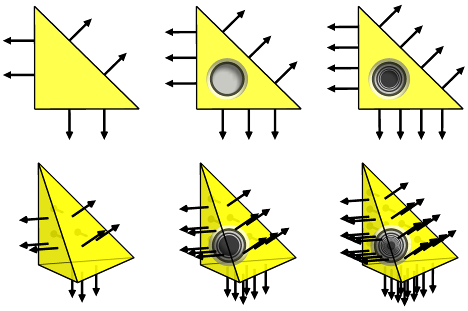


Figure 5.10: Illustration of the first, second and third degree Brezzi–Douglas–Marini elements on triangles and tetrahedra. The degrees of freedom are moments of the normal component against $\mathcal{P}_q$ on facets (edges and faces, respectively) and, for the higher degree elements, interior moments against NED^1_{q-1}.

Figure 5.9 shows some Raviart–Thomas elements.

Next element is the Brezzi–Douglas–Marini BDM_q elements. These are also $H(\mathrm{div})$-conforming elements.

Definition 5.5 (Brezzi–Douglas–Marini element)**.** *The Brezzi–Douglas–Marini element* (BDM_q) *is defined for* $q = 1, 2, \ldots$ *by*

$$T \in \{\text{triangle}, \text{tetrahedron}\}, \tag{5.13}$$

$$\mathcal{V} = [\mathcal{P}_q(T)]^d, \tag{5.14}$$

$$\mathcal{L} = \begin{cases} \int_f v \cdot n \, p \, \mathrm{d}s, & \textit{for a set of basis functions } p \in \mathcal{P}_q(f) \textit{ for each facet } f, \\ \int_T v \cdot p \, \mathrm{d}x, & \textit{for a set of basis functions } p \in \mathrm{NED}^1_{q-1}(T) \textit{ for } q \geqslant 2. \end{cases} \tag{5.15}$$

where NED^1 *refers to the Nédélec* $H(\mathrm{curl})$ *elements of the first kind.*

The dimension of BDM_q is

$$n(q) = \begin{cases} (q+1)(q+2), & T \text{ triangle}, \\ \frac{1}{2}(q+1)(q+2)(q+3), & T \text{ tetrahedron}. \end{cases} \tag{5.16}$$

Figure 5.10 shows the Brezzi–Douglas–Marini elements.

The last elements we will look at, are the Nédélec NED^2_q elements of second kind. These are $H(\mathrm{curl})$-conforming elements.

Definition 5.6 (Nédélec element of the second kind)**.** *The Nédélec element of the second kind* (NED^2_q) *is defined for* $q = 1, 2, \ldots$ *in two dimensions by*

$$T = \text{triangle}, \tag{5.17}$$

$$\mathcal{V} = [\mathcal{P}_q(T)]^2, \tag{5.18}$$

$$\mathcal{L} = \begin{cases} \int_e v \cdot t \, p \, \mathrm{d}s, & \textit{for a set of basis functions } p \in \mathcal{P}_q(e) \textit{ for each edge } e, \\ \int_T v \cdot p \, \mathrm{d}x, & \textit{for a set of basis functions } p \in \mathrm{RT}_{q-1}(T), \textit{ for } q \geqslant 2. \end{cases} \tag{5.19}$$

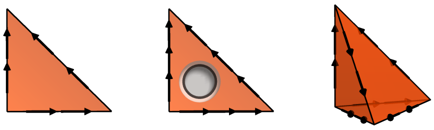


Figure 5.11: Illustration of first and second degree Nédélec $H(\mathrm{curl})$ elements of the second kind on triangles and first degree on tetrahedron. Note that these elements may be viewed as *rotated* Brezzi–Douglas–Marini elements.

where t is the edge tangent, and in three dimensions by

$$T = \text{tetrahedron}, \tag{5.20}$$

$$\mathcal{V} = [\mathcal{P}_q(T)]^3, \tag{5.21}$$

$$\mathcal{L} = \begin{cases} \int_e v \cdot t\, p \,\mathrm{d}l, & \textit{for a set of basis functions } p \in \mathcal{P}_q(e) \textit{ for each edge } e, \\ \int_f v \cdot p \,\mathrm{d}s, & \textit{for a set of basis functions } p \in \mathrm{RT}_{q-1}(f) \textit{ for each face } f, \textit{ for } q \geqslant 2 \\ \int_T v \cdot p \,\mathrm{d}x, & \textit{for a set of basis functions } p \in \mathrm{RT}_{q-2}(T), \textit{ for } q \geqslant 3. \end{cases} \tag{5.22}$$

The dimension of $\mathrm{NED}^2{}_q$ is

$$n(q) = \begin{cases} (q+1)(q+2), & T \text{ triangle}, \\ \frac{1}{2}(q+1)(q+2)(q+3), & T \text{ tetrahedron}. \end{cases} \tag{5.23}$$

Figure 5.11 shows the Nédélec element for second kind.

6 *Discretization of a convection-diffusion problem*

By Anders Logg, Kent–Andre Mardal

6.1 *Introduction*

This chapter concerns convection-diffusion equations of the form:

$$\begin{aligned} -\mu\Delta u + v \cdot \nabla u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega \end{aligned}$$

Here v is typically a velocity, μ is the diffusivity, and u is the unknown variable of interest. We assume the Dirichlet condition $u = g$ on the boundary, while f is a source term.

The problem is a singular perturbation problem. That is, the problem is well-posed for $\mu > 0$ but becomes over–determined as μ tends to zero. For $\mu = 0$ the Dirichlet conditions should only be set on the inflow domain Γ; that is, where $n \cdot v < 0$ for the outward unit normal n.

For many practical situations $\mu > 0$, but small in the sense that $\mu \ll |v|$. For such problems, the solution will often be similar to the solution of the reduced problem with $\mu = 0$ except close to the non-inflow boundary $\partial\Omega\backslash\Gamma$. Here, there will typically be a boundary layer $\exp\left(\|v\|_\infty x/\mu\right)$. Furthermore, discretizations often shows unphysical oscillations starting at this boundary layer.

The next example shows a 1D convection diffusion problem resulting in non-physical oscillations due to the use of a standard Galerkin approximation.

Example 6.1. ***Standard Galerkin approximation***
Consider the following 1D problem convection diffusion problem, where $v = -1$ for simplicity:

$$-u_x - \mu u_{xx} = 0, \tag{6.1}$$

$$u(0) = 0, u(1) = 1. \tag{6.2}$$

The analytical solution is:

$$u(x) = \frac{e^{-x/\mu} - 1}{e^{-1/\mu} - 1}.$$

Hence, for $\mu \to 0$, both $e^{-x/\mu}$ and $e^{-1/\mu}$ will be small and $u(x) \approx 1$ unless $x \approx 0$. However, close to the outflow boundary at $x = 0$, there will be a boundary layer where u has exponential growth.

We solve the problem with a standard Galerkin method using linear first order Lagrange elements. To be specific, the variational problem is:
Find $u \in H^1_{(0,1)}$ such that

$$\int_0^1 -u_x v + \mu u_x v_x \, \mathrm{d}x = 0, \quad \forall\, v \in H^1_{(0,0)}.$$

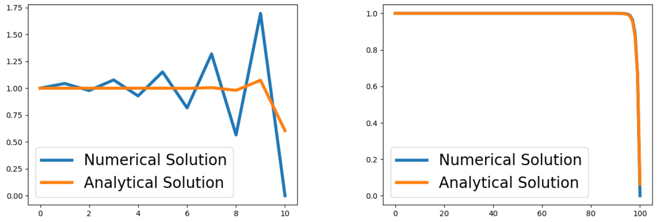


Figure 6.1: Solution of the convection diffusion problem obtained with 10 and 100 elements. The left figure obtained on a mesh with 10 elements shows wild oscillations, while the mesh with 100 elements demonstrate a nicely converged solution.

Here, $H^1_{(0,1)}$ *contains functions* $u \in H^1$ *with* $u = 0$ *at* $x = 0$ *and* $u = 1$ *and* $x = 1$*, while* $H^1_{(0,0)}$ *contains functions that are zero both at* $x = 0$ *and* $x = 1$*. We consider a* $\mu = 0.01$*, a relatively large* μ*, to enable us to see the differences on a relatively coarse mesh.*

Both the numerical and analytical solutions are shown in Figure 6.1. Clearly, the numerical solution is polluted by non-physical oscillations on the coarse mesh with 10 elements, while a good approximation is obtained for 100 elements.

Finally, we show the complete code for this example:

Python code

```python
from dolfin import *
for N in [10, 100]:

  mesh = UnitInterval(N)
  V = FunctionSpace(mesh, "CG", 1)

  u = TrialFunction(V)
  v = TestFunction(V)

  mu_value = 1.0e-2
  mu = Constant(mu_value)
  f = Constant(0)
  h = mesh.hmin()

  a = (-u.dx(0)*v + mu*u.dx(0)*v.dx(0))*dx
  L = f*v*dx

  u_analytical = Expression("(exp(-x[0]/e) - 1)/ (exp(-1/%e) - 1)" % (mu_value, mu_value))
  def boundary(x):
      return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

  bc = DirichletBC(V, u_analytical, boundary)

  U = Function(V)
  solve(a == L, U, bc)

  U_analytical = project(u_analytical, V)

  import pylab
  pylab.plot(U.vector().array())
  pylab.plot(U_analytical.vector().array())
  pylab.legend(["Numerical Solution", "Analytical Solution"])
pylab.show()
```

□

To understand Example 6.1 we first remark that the discretization corresponds to the following central finite difference scheme:

$$\begin{aligned} -\frac{\mu}{h^2}\left[u_{i+1} - 2u_i + u_{i-1}\right] - \frac{v}{2h}\left[u_{i+1} - u_{i-1}\right] &= 0, \quad i = 1, \ldots, N-1 \\ u_0 = 0, \quad u_N = 1 & \end{aligned}$$

Above, we kept v as a variable such that we may discuss the directionality of upwinding in terms of the convection. Clearly, if $\mu = 0$ then the scheme reduces to

$$\begin{aligned} -\frac{v}{2h}\left[u_{i+1} - u_{i-1}\right] &= 0, \quad i = 1, \ldots, N-1 \\ u_0 = 0, \quad u_N = 1 & \end{aligned}$$

Here, it is clear that u_{i+1} is coupled to u_{i-1}, but not to u_i. Hence, this scheme allow for an alternating sequence of $u_{i+1} = u_{i-1} = \ldots$, while $u_i = u_{i-2} = \ldots$ resulting in oscillations.

One cure for these oscillations is upwinding. That is, instead of using a central difference scheme, we employ the following difference scheme:

$$\begin{aligned} \frac{du}{dx}(x_i) &= \frac{1}{h}[u_{i+1} - u_i] \quad \text{if } \ v < 0, \\ \frac{du}{dx}(x_i) &= \frac{1}{h}[u_i - u_{i-1}] \quad \text{if } \ v > 0. \end{aligned}$$

Using this scheme, oscillations will disappear. The approximation will however only be first order.

There is a relationship between upwinding and artificial diffusion. If we discretize u_x with a central difference and add diffusion as $\epsilon = h/2\Delta$ we get

$$\begin{aligned} & \frac{u_{i+1} - u_{i-1}}{2h} && \text{central scheme, first order derivative} \\ +\frac{h}{2} & \frac{-u_{i+1} + 2u_i - u_{i-1}}{h^2} && \text{central scheme, second order derivate} \\ = & \frac{u_i - u_{i-1}}{h} && \text{upwind scheme} \end{aligned}$$

Hence, upwinding is equivalent to adding artificial diffusion with $\epsilon = h/2$; that is, in both cases we actually solve the problem

$$-(\mu + \epsilon)u_{xx} + vu_x = f.$$

using a central difference scheme.

Finite difference upwinding is difficult to express using finite elements methods, but it is closely to adding some kind of diffusion to the scheme. The next example shows the solution of the problem in Example 6.1 with artificial diffusion added.

Example 6.2. ***Stabilization using artificial diffusion***
Consider again the following 1D problem convection diffusion problem:

$$-u_x - \mu u_{xx} = 0, \tag{6.3}$$

$$u(0) = 0, u(1) = 1. \tag{6.4}$$

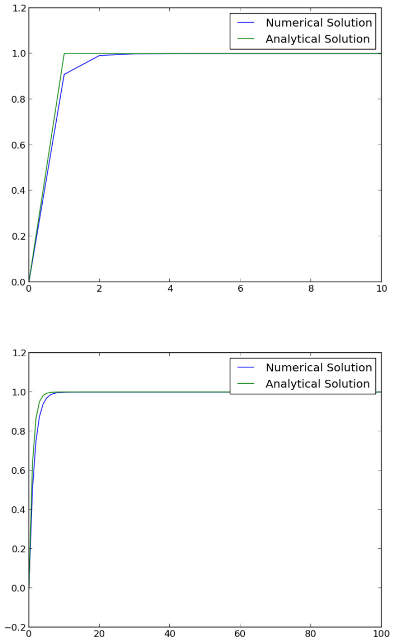


Figure 6.2: Solution of the convection diffusion problem obtained with 10 and 100 elements using artificial diffusion to stabilize.

We solve the problem with a standard Galerkin method using linear first order Lagrange elements as before, but we add artificial diffusion. To be specific, the variational problem is:
Find $u \in H^1_{(0,1)}$ such that

$$\int_0^1 -u_x v + (\mu + \beta h) u_x v_x = 0, \quad \forall\, v \in H^1_{(0,0)},$$

where $\beta = 0.5$ corresponds to the finite difference scheme with artificial diffusion mentioned above. Below is the code for the changed variational form:

Python code

```
beta_value = 0.5
beta = Constant(beta_value)
f = Constant(0)
h = mesh.hmin()
a = (-u.dx(0)*v + mu*u.dx(0)*v.dx(0) + beta*h*u.dx(0)*v.dx(0))*dx
```

Figure 6.2 shows the solution for 10 and 100 elements when using artificial diffusion stabilization. Clearly, the solution for the coarse grid has improved dramatically since the oscillations have vanished and the solution appear smooth. It is, however, interesting to note that the solution for the fine mesh is actually less accurate than the solution in Fig 6.2 for the corresponding fine mesh. The reason is that the scheme is now first order, while the scheme in Example 6.1 is second order.

6.2 Streamline diffusion/Petrov-Galerkin methods

In the previous section we saw that artificial diffusion may be added to convection diffusion dominated problems to avoid oscillations. The diffusion was, however, added in a rather ad-hoc manner. Here, we will see how diffusion may be added in a consistent way; that is, without changing the solution as $h \to 0$. This leads us to streamline diffusion using the Petrov-Galerkin method. Our problem reads: Find u such that

$$\begin{aligned} -\mu\Delta u + v\cdot\nabla u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}$$

The **weak formulation** reads:
Find $u \in H^1_g$ such that

$$a(u,w) = b(w) \quad \forall\ w \in H^1_0,$$

where

$$\begin{aligned} a(u,w) &= \int_\Omega \mu\nabla u\cdot\nabla w\,\mathrm{d}x + \int_\Omega v\cdot\nabla u w\,\mathrm{d}x, \\ b(w) &= \int_\Omega f w\,\mathrm{d}x. \end{aligned}$$

Here, H^1_g is the subspace of H^1 where the trace equals g on the boundary $\partial\Omega$.

The *standard Galerkin* discretization is:
Find $u_h \in V_{h,g}$ such that

$$a(u_h, v_h) = (f, v_h)\ \forall\, v_h \in V_{h,0}. \tag{6.5}$$

Here, $V_{h,g}$ and $V_{h,0}$ are the subspaces with traces that equals g and 0 on the boundary, respectively.

Adding artificial diffusion to the standard Galerkin discretization, as was done in Example 6.2, can be done as:
Find $u_h \in V_{h,g}$ such that

$$a(u_h, v_h) + \frac{h}{2}(\nabla u_h, \nabla v_h) = (f, v_h)\ \forall\, v_h \in V_{h,0}.$$

Let

$$\tau(u, v_h) = a(u_h, v_h) - (f, v_h).$$

Then the *truncation error* is first order in h; that is,

$$\tau(u) = \sup_{v\in V_h, v\neq 0} \frac{\tau(u,v_h)}{\|v\|_V} \sim \mathcal{O}(h).$$

Hence, the scheme is *consistent* in the sense that

$$\lim_{h\to 0} \tau(u) \to 0.$$

However, it is not *strongly consistent* in the sense that $\tau(u) = 0$ for every discretization, which is what is obtained with the Galerkin method due to Galerkin-orthogonality:

$$\tau(u, v_h) = a(u_h, v_h) - (f, v_h) = a(u_h - h, v_h) = 0 \quad \forall\, v_h \in V_h.$$

The *Streamline diffusion/Petrov-Galerkin* method introduces a strongly consistent diffusion by employing alternative test functions. Let us therefore assume that we have a space of test functions W_h. Abstractly, the Petrov-Galerkin method appears very similar to the Galerkin method, that is:
Find $u_h \in V_{h,g}$ such that

$$a(u_h, v_h) = (f, v_h) \quad \forall\, v_h \in W_{h,0}.$$

Again, $V_{h,g}$ and $W_{h,0}$ are the subspaces with traces that equals g and 0 on the boundary, respectively. Notice that the only difference from the standard Galerkin formulation is that test and trial functions differ.

On matrix form, the standard Galerkin formulation reads:

$$A_{ij} = a(N_i, N_j) = \int_\Omega \mu \nabla N_i \cdot \nabla N_j \,\mathrm{d}x + \int_\Omega v \cdot \nabla N_i N_j \,\mathrm{d}x, \tag{6.6}$$

while for the Petrov Galerkin method, we use the test functions L_j:

$$A_{ij} = a(N_i, L_j) = \int_\Omega \mu \nabla N_i \cdot \nabla L_j \,\mathrm{d}x + \int_\Omega v \cdot \nabla N_i L_j \,\mathrm{d}x$$

A clever choice of L_j will enable us to add diffusion in a consistent way. To make sure that the matrix is still quadratic, we should however make sure that the dimension of V_h and W_h are equal.

Let L_j be defined as $L_j = N_j + \beta h\, v \cdot \nabla N_j$. Writing out the matrix A_{ij} in (6.6) now gives

$$\begin{aligned} A_{ij} &= a(N_i, N_j + \beta h\, v \cdot \nabla N_j) \\ &= \int_\Omega \mu \nabla N_i \cdot \nabla(N_j + \beta h\, v \cdot \nabla N_j) \,\mathrm{d}x + \int_\Omega v \cdot \nabla N_i \cdot (N_j + \beta h\, v \cdot \nabla N_j) \,\mathrm{d}x \\ &= \underbrace{\int_\Omega \mu \nabla N_i \cdot \nabla N_j \,\mathrm{d}x + \int_\Omega v \cdot \nabla N_i\, N_j \,\mathrm{d}x}_{\text{standard Galerkin}} \\ &+ \underbrace{\beta h \int_\Omega \mu \nabla N_i \cdot \nabla(v \cdot \nabla N_j) \,\mathrm{d}x}_{=0 \text{ third order term, for linear elements}} \underbrace{+ \beta h \int_\Omega (v \cdot \nabla N_i)(v \cdot \nabla N_j) \,\mathrm{d}x}_{\text{Artificial diffusion in } v \text{ direction}} \end{aligned}$$

Notice that also the righthand side changes

$$b(L_j) = \int_\Omega f L_j \,\mathrm{d}x = \int_\Omega f(N_j + \beta h\, v \cdot \nabla N_j) \,\mathrm{d}x$$

Thus, both the matrix and the righthand side are changed such that artificial diffusion is added in a consistent way.

We summarize this derivation by stating the SUPG problem. Find $u_{h,sd} \in H^1_g$ such that

$$a_{sd}(u, w) = b_{sd}(w) \quad \forall\, w \in H^1_0, \tag{6.7}$$

where

$$\begin{aligned} a_{sd}(u, w) &= \int_\Omega \mu \nabla u \cdot \nabla w \,\mathrm{d}x + \int_\Omega v \cdot \nabla u w \,\mathrm{d}x \\ &+ \beta h \int_\Omega (v \cdot \nabla u)(v \cdot \nabla w) \,\mathrm{d}x + \beta h\, \mu \sum_e \int_{\Omega_e} -\Delta u (v \cdot \nabla w) \,\mathrm{d}x, \\ b_{sd}(w) &= \int_\Omega f w \,\mathrm{d}x + \beta h \int_\Omega f v \cdot \nabla w \,\mathrm{d}x. \end{aligned}$$

6.3 Well posedness of the continuous problem

Before we discuss error estimates of the discrete problem, we briefly describe the properties of the continuous problem.

Theorem 6.1. ***Lax-Milgram theorem***
Let V be a Hilbert space, $a(\cdot,\cdot)$ be a bilinear form, $L(\cdot)$ a linear form, and let the following three conditions be satisfied:

1. $a(u,u) \geq \alpha \|u\|_V^2, \quad \forall\, u \in V,$

2. $a(u,v) \leq C\|u\|_V\|v\|_V, \quad \forall\, u,v \in V,$

3. $L(v) \leq D\|v\|_V, \quad \forall\, v \in V\ .$

Then the problem: Find $u \in V$ such that

$$a(u,v) = L(v) \quad \forall\, v \in V.$$

is well-posed in the sense that there exists a unique solution with the following stability condition

$$\|u\|_V \leq \frac{C}{\alpha}\|L\|_{V^*}.$$

Condition (1) is often refereed to as coersivity or positivity, while (2) is called continuity or boundedness. Condition 3 simply states that the right-hand side should be in the dual space of V.

In the following we will use Lax-Milgram's theorem to show that the convection-diffusion problem is well-posed. The Lax-Milgram's theorem is well-suited since it does not require symmetry of the bilinear form.

We will only consider the homogeneous Dirichlet conditions in the current argument[1]. From Poincare's lemma we know that

$$\|u\|_0 \leq C_\Omega |u|_1.$$

Using Poincare, it is straightforward to show that the semi-norm

$$|u|_1 = (\int (\nabla u)^2 \,\mathrm{d}x)^{1/2}$$

and the standard H^1 norm

$$\|u\|_1 = (\int (\nabla u)^2 + u^2 \,\mathrm{d}x)^{1/2}$$

are equivalent. Hence, on H_0^1 the $|\cdot|_1$ is a norm equivalent the H^1-norm. Furthermore, this norm will be easier to use for our purposes.

For the convection-diffusion problem, we will consider two cases 1) incompressible flow, where $\nabla \cdot v = 0$ and 2) compressible flow, where $\nabla \cdot v \neq 0$. Let us for the begin with the incompressible case.

[1] Has the argument for reducing non-homogeneous Dirichlet conditions to homogeneous Dirichlet conditions been demonstrated elsewhere?

Further, let

$$\begin{aligned} b(u,w) &= \int_\Omega \mu \nabla u \nabla w \, \mathrm{d}x \\ c_v(u,w) &= \int_\Omega v \cdot \nabla u \, w \, \mathrm{d}x \\ a(u,w) &= a(u,w) + b(u,w) \end{aligned}$$

Furthermore, assuming for the moment that $u \in H^1_g, w \in H^1_0$, we have

$$\begin{aligned} c_v(u,w) &= \int_\Omega v \cdot \nabla u \, w \, \mathrm{d}x \\ &= -\int_\Omega v \cdot \nabla w \, u \, \mathrm{d}x - \underbrace{\int_\Omega \nabla \cdot v \, u \, w \, \mathrm{d}x}_{=0 \text{ (incompressibility)}} + \underbrace{\int_\Gamma u \, w \, v \cdot n}_{=0 \text{ (Dirichlet conditions)}} \\ &= -c_v(w,u). \end{aligned}$$

and therefore $c_v(\cdot,\cdot)$ is skew-symmetric. Letting $w = u$ we obtain that $c_v(u,u) = -c_v(u,u)$, which means that $c_v(u,u) = 0$. Therefore, the first condition in Lax-Milgram's theorem (1) is satisfied:

$$a(u,u) = b(u,u) \geq \mu |u|_1^2.$$

The second condition, the boundedness of a (2), follows by applying Cauchy-Schwartz inequality if we assume bounded flow velocities $\|v\|_\infty$.

$$\begin{aligned} a(u,v) &= \int_\Omega \mu \nabla u \nabla w \, \mathrm{d}x + \int_\Omega v \nabla u w \, \mathrm{d}x \\ &\leq \mu |u|_1 |w|_1 + \|v\|_\infty |u|_1 \|w\|_0 \\ &\leq (\mu + \|v\|_\infty C_\Omega) |u|_1 |v|_1. \end{aligned}$$

The third condition simply means that the right-hand side needs to be in the dual space of H^1_g. Hence, we obtain the following bounds by Lax-Milgram's theorem:

$$|u|_1 \leq \frac{\mu + C_\Omega \|v\|_\infty}{\mu} \|f\|_{-1}.$$

Notice that for convection-dominated problems $C_\Omega \|v\|_\infty \gg \mu$ and the stability constant will therefore be large.

In the case where $\nabla \cdot v \neq 0$, we generally obtain that $c_v(u,u) \neq 0$. To ensure that $a(u,u)$ is still positive, we must then put some restrictions on the flow velocities. That is, we need

$$|c_v(u,u)| \leq a(u,u).$$

If $C_\Omega \|v\|_\infty \leq D\mu$ with $D < 1$ we obtain

$$\begin{aligned} a(u,u) &= \int_\Omega \mu \nabla u \nabla u \, \mathrm{d}x + \int_\Omega v \nabla u u \, \mathrm{d}x \\ &\geq \mu |u|_1 |v|_1 - \|v\|_\infty |u|_1 \|u\|_0 \\ &\geq (\mu - \|v\|_\infty C_\Omega) |u|_1 |u|_1 \\ &\geq (\mu(1-D)) |u|_1^2. \end{aligned}$$

Further, the second condition of Lax-Milgram's theorem still applies. However, that $C_\Omega \|v\|_\infty \leq D\mu$ is clearly very restrictive compared to the incompressible case.

We remark that the Lax-Milgram conditions in the presence of the SUPG clearly will not be satisified in the continuous case because of the third order term $-\Delta u (v \cdot \nabla w)$. With this term, the second condition of Lax-Milgram is not satisified with $C \leq \infty$.

Finally, in order to make the term $c_v(u,u)$ skew-symmetric, it was required that the boundary integral $\int_\Gamma u^2 w \cdot n$ was zero. This was a consequence of the Dirichlet conditions. In general, this is neither needed nor possible at Neumann boundaries. As long as $\int_\Gamma u^2 w \cdot n \geq 0$, the above argumentation is valid. From a physical point of view this means that there is outflow at the Neumann boundary, i.e., that $w \cdot n \geq 0$.

6.4 Error estimates

Finally, we provide some error estimates for the Galerkin method and the SUPG method applied to the convection-diffusion equation. Central in the derivation of both results are the following interpolation result.

Theorem 6.2. ***Approximation by interpolation***
There exists an interpolation operator $I_h : H^{t+1} \to V_h$ where V_h is a piecewise polynomial field of order t with the property that for any $u \in H^t(\Omega)$

$$\|u - I_h u\|_m \leq B h^{t+1-m} \|u\|_{t+1}.$$

Proof. The bounds on the interpolation error is provided by the Bramble-Hilbert lemma for $t \geq 1$ and Clement's result (the case $t = 1$), cf. e.g. **??**. □

For the Galerkin method the general and elegant result of Cea's lemma provide us with error estimates. Cea's lemma applies to general conforming approximations, i.e. when $V_h \subset V$. In our case $V = H_0^1(\Omega)$ and V_h is a finite element subspace such as for example a discretization in terms of the Lagrange elements (of any order). Hence, in our case $\| \cdot \|_V = | \cdot |_1$ and the H^1 semi-norm is equivalent with the full H^1 norm due to Poincare's inequality.

Theorem 6.3. ***Cea's lemma***
Suppose the conditions for Lax-Milgram's theorem is satisfied and that we solve the linear problem (6.5) *on a finite element space of order t. Then,*

$$\|u - u_h\|_V \leqslant C_1 \frac{CB}{\alpha} h^t \|u\|_{t+1}.$$

Here $C_1 = \frac{CB}{\alpha}$, where B comes from the approximation property and α and C are the constants of Lax-Milgram's theorem.

Proof. The proof is straightforward and follows from the Galerkin orthogonality:

$$a(u-u_h,v)=0, \quad \forall\, v\in V_h$$

Since $V_h \subset V$:

$$\begin{aligned} \alpha\|u-u_h\|_V^2 &\le a(u-u_h,u-u_h) \\ &= a(u-u_h,u-v)-a(u-u_h,v-u_h) \\ &\le C\|u-u_h\|_V\|u-v\|_V. \end{aligned}$$

Since $v-u_h\in V_h$. Furthermore, v is arbitrary and we may therefore choose $v=I_hu$ and obtain:

$$|u-u_h|_1 \leqslant \frac{C}{\alpha}|u-I_hu|_1 \le \frac{CB}{\alpha}h^t\|u\|_t,$$

where $t-1$ is the order of the polynomials of the finite elements. □

We remark, as mentioned above, that $\frac{C}{\alpha}$ is large for convection dominated problems and that this is what causes the poor approximation on the coarse grid, shown in Example 6.1.

To obtain improved error estimates for the SUPG method, we introduce an alternative norm:

$$\|u\|_{sd}=\left(h\|v\cdot\nabla u\|^2+\mu|\nabla u|^2\right)^{1/2} \tag{6.8}$$

Theorem 6.4. *Suppose the conditions for Lax-Milgram's theorem is satisfied in the Hilbert space defined by the SUPG norm* (6.8) *and that we solve the SUPG problem* (6.7) *on a finite element space of order* 1*. Then,*

$$\|u-u_h\|_{sd}\leqslant Ch^{3/2}\|u\|_2$$

Proof. The proof can be found in e.g. **??**. □

6.5 Exercises

Exercise 6.1. *Show that the matrix obtained from a central difference scheme applied to the operator $Lu=u_x$ is skew-symmetric. Furthermore, show that the matrix obtained by linear continuous Lagrange elements are also skew-symmetric. Remark: The matrix is only skew-symmetric in the interior of the domain, not at the boundary.*

Exercise 6.2. *Estimate numerically the constant in Cea's lemma for various α and h for the Example 6.1.*

Exercise 6.3. *Implement the problem $u=sin(\pi x)$, and $f=-\alpha u_{xx}-u_x$ and estimate numerically the constant in Cea's lemma for various α. Compare with the corresponding constant estimated from Example 6.1.*

Exercise 6.4. *Implement the problem $u=sin(\pi x)$, and $f=-\alpha u_{xx}-u_x$ using SUPG and estimate the constants in the error estimate obtained by both the $|\cdot|_1$ and the $\|\cdot\|_v$ norms. Compare with the corresponding constant estimated from Example 6.1.*

Exercise 6.5. *Investigate whether the coersivity condition holds when a homogeneous Neumann condition is assumed on the outflow. You may assume that $v\cdot n>0$.*

Exercise 6.6. *Consider the eigenvalues of the operators, L_1, L_2, and L_3, where $L_1u=u_x$, $L_2u=-\alpha u_{xx}$, $\alpha=1.0e^{-5}$, and $L_3=L_1+L_2$, with homogeneous Dirchlet conditions. For which of the operators are the eigenvalues positive and real? Repeat the exercise with $L_1=xu_x$.*

Exercise 6.7. *Compute the Soblev norms $\| \cdot \|_m$ of the function $sin(k\pi x)$ on the unit interval. Assume that the Soblev norm is $\|u\|_m = (-\Delta^m u, u)^{1/2}$. What happens with negative m? You may use either Fourier transformation or compute (eigenvalues of) powers of the stiffness matrix.*

Exercise 6.8. *Perform numerical experiments to determine the order of approximation with respect to various Soblev norms and polynomial orders for the function $sin(k\pi x)$ on the unit interval.*

7 Stokes problem

By Anders Logg, Kent–Andre Mardal

7.1 Introduction

The Stokes problem describes the flow of a slowly moving viscous incompressible Newtonian fluid. Let the fluid domain be denoted Ω. We assume that Ω is a bounded domain in $\mathbb{R}^n$ with a smooth boundary. Furthermore, let $u : \Omega \to \mathbb{R}^n$ be the fluid velocity and $p : \Omega \to \mathbb{R}$ be the fluid pressure. The strong form of the Stokes problem can then be written as

$$\begin{aligned} -\Delta u + \nabla p &= f, \text{ in } \Omega, && (7.1)\\ \nabla \cdot u &= 0, \text{ in } \Omega, && (7.2)\\ u &= g, \text{ on } \partial\Omega_D, && (7.3)\\ \frac{\partial u}{\partial n} - pn &= h, \text{ on } \partial\Omega_N. && (7.4) \end{aligned}$$

Here, f is the body force, $\partial\Omega_D$ is the Dirichlet boundary, while $\partial\Omega_N$ is the Neumann boundary. Furthermore, g is the prescribed fluid velocity on the Dirichlet boundary, and h is the surface force or stress on the Neumann boundary. These boundary condition leads to a well-posed problem provided that neither the Dirichlet nor Neumann boundaries are empty. In case of only Dirichlet conditions the pressure is only determined up to a constant, while only Neumann conditions leads to the velocity only being determined up to a constant.

These equations are simplifications of the Navier–Stokes equations for very slowly moving flow. In contrast to elliptic equations, many discretizations of this problem will lead to instabilities. These instabilities are particularly visible as non-physical oscillations in the pressure. The following example illustrate such oscillations.

Example 7.1. *Poiseuille flow*
One of the most common examples of flow problems that can be solved analytically is Poiseuille flow. It describes flow in a straight channel (or cylinder in 3D). The analytical solution is $u = (y\,(1-y), 0)$ and $p = 1 - x$. Since the solution is know, this flow problem is particularly useful for verifying that the code or numerical method. We therefore begin by discretizing the problem in the simplest way possible; that is, linear continuous/Lagrange elements for both velocity and pressure. The results is shown Figure 7.1. Clearly, the velocity is approximated satisfactory, but the pressure oscillate widely and is nowhere near the actual solution.

Python code

```
from dolfin import *

def u_boundary(x):
```

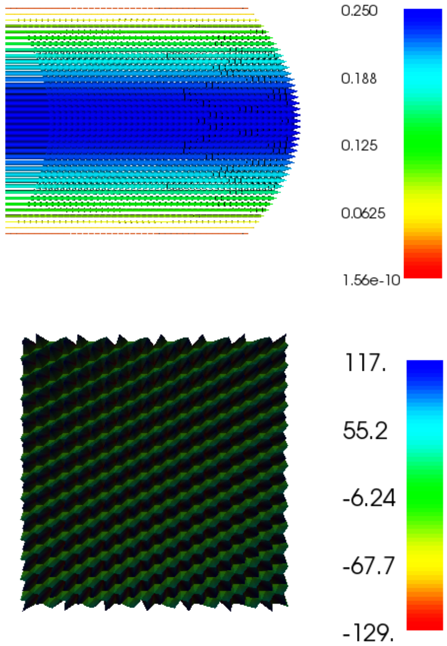


Figure 7.1: Poiseuille flow solution obtained with linear continuous elements for both velocity and pressure. The left figure shows the (well-represented) velocity while the right shows the pressure (with the wild oscillations).

```
  return x[0] < DOLFIN_EPS or x[1] > 1.0 - DOLFIN_EPS or x[1] < DOLFIN_EPS

def p_boundary(x):
  return  x[0] > 1.0 - DOLFIN_EPS

mesh = UnitSquare(40,40)
V = VectorFunctionSpace(mesh, "Lagrange", 1)
Q = FunctionSpace(mesh, "Lagrange", 1)
#Q = FunctionSpace(mesh, "DG", 0)
W = MixedFunctionSpace([V, Q])

u, p = TrialFunctions(W)
v, q = TestFunctions(W)

f = Constant([0,0])

u_analytical = Expression(["x[1]*(1-x[1])", "0.0"])
p_analytical = Expression("-2+2*x[0]")

bc_u = DirichletBC(W.sub(0), u_analytical, u_boundary)
bc = [bc_u]

a = inner(grad(u), grad(v))*dx + div(u)*q*dx + div(v)*p*dx
L = inner(f, v)*dx

UP = Function(W)
A, b = assemble_system(a, L, bc)
solve(A, UP.vector(), b, "lu")

U, P = UP.split()

plot(U, title="Numerical velocity")
plot(P, title="Numerical pressure")

U_analytical = project(u_analytical, V)
P_analytical = project(p_analytical, Q)

plot(U_analytical, title="Analytical velocity")
plot(P_analytical, title="Analytical pressure")

interactive()
```

However, when using the second order continuous elements for the velocity and first order continuous elements for the pressure, we obtain the perfect solution shown in Figure 7.2.

The previous example demonstrates that discretizations of the Stokes problem may lead to, in particular, strange instabilities in the pressure. In this chapter we will describe why this happens and several strategies to circumvent this behaviour.

7.2 Finite Element formulation

Let us first start with a weak formulation of Stokes problem: Find $u \in H^1_{D,g}$ and $p \in L^2$.

$$\begin{aligned} a(u,v) + b(p,v) &= f(v), \quad v \in H^1_{D,0} \\ b(q,u) &= 0, \quad q \in L^2, \end{aligned}$$

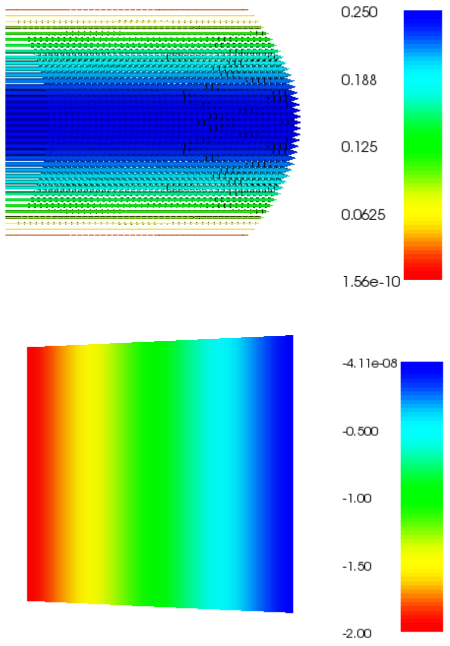


Figure 7.2: Poiseuille flow solution obtained with quadratic continuous elements for the velocity and linear continuous elements for the pressure. The left figure shows the velocity while the right shows the pressure. Both the velocity and the pressure are correct.

where

$$\begin{aligned} a(u,v) &= \int \nabla u : \nabla v \, \mathrm{d}x, \\ b(p,v) &= \int p \, \nabla \cdot v \, \mathrm{d}x, \\ f(v) &= \int f \, v \, \mathrm{d}x + \int_{\Omega_N} h \, v \; \mathrm{d}s. \end{aligned}$$

Here $H^1_{D,g}$ contains functions in H^1 with trace g on $\partial\Omega_D$. To obtain symmetry we have substituted $\hat{p} = -p$ for the pressure and is referint to $\hat{p}$ as p.

As before the standard finite element formulation follows directly from the weak formulation: Find $u_h \in V_{g,h}$ and $p_h \in Q_h$ such that

$$\begin{aligned} a(u_h, v_h) + b(p_h, v_h) &= f(v_h), \quad \forall \, v_h \in V_{0,h}, && (7.5) \\ b(q_h, u_h) &= 0, \quad \forall \, q_h \in Q_h. && (7.6) \end{aligned}$$

Letting $u_h = \sum_{i=1}^{n} u_i N_i$, $p_h = \sum_{i=1}^{m} p_i L_i$, $v_h = N_j$, and $q_h = L_j$ we obtain a linear system on the form

$$\left[\begin{array}{cc} A & B^T \\ B & 0 \end{array} \right] \left[\begin{array}{c} u \\ p \end{array} \right] = \left[\begin{array}{c} f \\ 0 \end{array} \right] \qquad (7.7)$$

Here

$$\begin{aligned} A_{ij} = a(N_i, N_j) &= \int \nabla N_i \nabla N_j \, \mathrm{d}x, && (7.8) \\ B_{ij} = b(L_i, N_j) &= \int \nabla L_i \, N_j \, \mathrm{d}x. && (7.9) \end{aligned}$$

Hence, A is $n \times n$, while B is $m \times n$, where n is the number of degrees of freedom for the velocity field, while m is the number of degrees of freedom for the pressure.

Is the system (7.7) invertible? For the moment, we assume that the submatrix A is invertible. This is typically the case for Stokes problem. We may then perform blockwise Gauss elimination: That is, we multiply the first equation with A^{-1} to obtain

$$u = A^{-1} f - A^{-1} B^T p$$

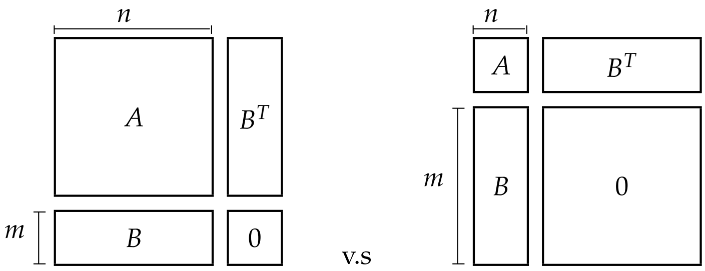
Then, we then insert u in the second equation to get

$$0 = Bu = BA^{-1} f - BA^{-1} B^T p$$

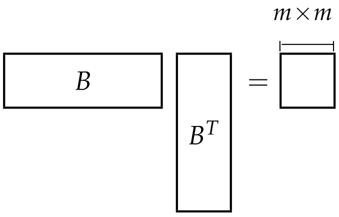
i.e we have removed v and obtained an equation only involving p:

$$BA^{-1} B^T p = BA^{-1} f$$

This equation is often called the pressure Schur complement. The question is then reduced to whether $BA^{-1}B^T$ is invertible. Consider the follwing two situations:



Clearly, the right most figure is not invertible since $n \ll m$ and the 0 in the lower right corner dominates. For the left figure on might expect that the matrix is non-singular since $n \gg m$, but it will depend on A and B. We have already assumed that A is invertible, and we therefore ignore A^{-1} in $BA^{-1}B^T$. The question is then whether BB^T is invertible.



As illustrated above, BB^T will be a relatively small matrix compared to B^T and A as long as $n \gg m$. Therefore, BB^T may therefore be non-singular. To ensure that BB^T is invertible, it is necessary that

$$\text{kernel}(B^T) = 0, \text{ where } B \text{ is } m \times n$$

An equvialent statement is that

$$\max_v \, (v, B^T p) > 0 \quad \forall \, p. \tag{7.10}$$

Alternatively,

$$\max_v \, \frac{(v, B^T p)}{\|v\|} \geq \beta \|p\| \quad \forall \, p. \tag{7.11}$$

which obviously may be written

$$\max_v \, \frac{(Bv, p)}{\|v\|} \geq \beta \|p\| \quad \forall \, p. \tag{7.12}$$

Here, $\beta > 0$. We remark that (7.10) and (7.11) are equivalent for a finite dimensional matrix. However, in the infinite dimentional setting of PDEs (7.10) and (7.11) are different. Inequality (7.10) allow $(v, B^T p)$ to approach zero, while (7.11) requires a lower bound. For the Stokes problem, the corresponding condition is crucial:

$$\sup_{v \in H^1_{D,g}} \frac{(p, \nabla \cdot u)}{\|u\|_1} \geqslant \beta \|p\|_0 > 0, \quad \forall \, p \in L^2 \tag{7.13}$$

Similarly, to obtain order optimal convergence rates, that is

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant Ch^k \|u\|_{k+1} + Dh^{\ell+1} \|p\|_{\ell+1}$$

where k and ℓ are the ploynomial degree of the velocity and the pressure, respectively, the celebrated *Babuska-Brezzi condition* has to be satisfied:

$$\sup_{v \in V_{h,g}} \frac{(p, \nabla \cdot v)}{\|v\|_1} \geqslant \beta \|p\|_0 > 0, \quad \forall\, p \in Q_h \tag{7.14}$$

We remark that the discrete condition (7.14) does not follow from (7.13). In fact, it is has been a major challenge in numerical analysis to determine which finite element pairs V_h and Q_h that meet this condition.

Remark 7.2.1. *For saddle point problems on the form* (7.5)-(7.6) *four conditions have to be satisfied in order to have a well-posed problem:*
Boundedness of a:

$$a(u_h, v_h) \leq C_1 \|u_h\|_{V_h} \|v_h\|_{V_h}, \quad \forall\, u_h, v_h \in V_h, \tag{7.15}$$

and boundedness of b:

$$b(u_h, q_h) \leq C_2 \|u_h\|_{V_h} \|q_h\|_{Q_h}, \quad \forall\, u_h \in V_h, q_h \in Q_h, \tag{7.16}$$

Coersivity of a:

$$a(u_h, u_h) \geq C_3 \|u_h\|^2_{V_h}, \quad \forall\, u_h \in Z_h, \tag{7.17}$$

where $Z_h = \{u_h \in V_h \mid b(u_h, q_h) = 0,\ \forall q_h \in Q_h\}$ *and "coersivity" of b:*

$$\sup_{u_h \in V_h} \frac{b(u_h, q_h)}{\|u_h\|_{V_h}} \geq C_4 \|q_h\|_{Q_h}, \quad \forall\, q_h \in Q_h. \tag{7.18}$$

For the Stokes problem, (7.15)-(7.17) *are easily verified, while* (7.18) *often is remarkably difficult unless the elements are designed to meet this condition. We remark also that condition* (7.17) *strictly speaking only needs to be valid on a subspace of V_h but this is not important for the Stokes problem.*

7.3 Examples of elements

7.3.1 The Taylor-Hoood element

The Taylor-Hood elements are quadratic for the velocity and linear for pressure, i.e., the i'th basis function of the velocity and pressure are of the form

$$\begin{aligned} u:\ N_i &= a_i + b_i x + c_i y + d_i xy + e_i x^2 + f_i y^2, \\ p:\ L_i &= k_i + l_i x + m_i y. \end{aligned}$$

And the basis functions are continuous across elements. For the Taylor-Hood element we have the following error estimate:

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant Ch^2 (\|u\|_3 + \|p\|_2).$$

The generalization of the Taylor–Hood element to higher order, that is $P_k - P_{k-1}$, satisfies the Brezzi conditions (on reasonable meshes). For the Taylor-Hood element of higher order we have the following error estimate:

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant Ch^k (\|u\|_{k+1} + \|p\|_k).$$

7.3.2 The Crouzeix–Raviart element

This element is linear in velocity and constant in pressure. Hence, the i'th basis functions are of the form:

$$\begin{aligned} v: \ N_i &= a_i + b_i x + c_i y \\ p: \ L_i &= a_i \end{aligned}$$

The v element is continuous *only* in the mid-point of each side, and the p element is discontinuous. The Crouzeix-Raviart element satisifies the inf-sup condition, but is non-conforming because it is only continuous at the mid-point of each element. The non-conformity does not affect the approximation properties for the Stokes problem, but can not be used if the $-\Delta u - \nabla p = f$ is replaced with the more "physically correct" $-\nabla \cdot \epsilon(u) - \nabla p = f$, where $\epsilon = \frac{1}{2}(\nabla + \nabla^T)$ is the symmetric gradient. For the Crouzeix–Raviart element we have the following error estimate:

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant Ch(\|u\|_2 + \|p\|_1)$$

The element may be generalized to odd, but not even orders.

7.3.3 The P1-P0 element

The P1-P0 element is perhaps the most natural element to consider as it consists of combining continuous piecewise linear functions for the velocity with piecewise constants for the pressure. This combination often work quite well, and this puzzled the community for quite some time. However, this combination is not inf-sup stable and oscillations in the pressure may occur.

7.3.4 The P2-P0 element

$P_2 - P_0$ element is a popular element that satisfies the Brezzi conditions. An advantage with this approach is that mass conservation is maintained elementwise. However, the approximation properties of the pressure is one order lower than that for the Taylor-Hood element and consequently the velocity approximation is also formally, in general, reduced by one order, i.e.,

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant C_0 h^2 \|u\|_2 + C_1 h \|p\|_2$$

The $P_2 - P_0$ element can be generalized to higher order. In fact, $P_k - P_{k-2}$, satisfies the Brezzi conditions for $k \geq 2$. Here, the pressure element P_{k-2} may in fact consist of both continuous and discontinuous polynomials. The discontinuous polynomials then has the advantage of providing mass conservation, albeit at the expence of many degrees of freedom compared with the continuous variant.

7.3.5 The Mini element

The mini element is linear in both velocity and pressure, but with one degree of freedom added per element since it is well-known that elements that are linear in both v and p will not satisfy the inf-sup condition. The extra degree of freedom is in 2D constructed such it is a cubic polynomial which is zero at all element faces. For example, on the reference element, the barycentric coordinates x, y, and $1 - x - y$ are all zero at their respective faces. Hence, the composition $xy(1 - x - y)$ is zero at all element faces. The barycentric coordinates can be used for this purpose on any element and also in higher dimensions. The function is often called the bubble function as its support is local to one element and is zero at the element faces. For the Mini element we have the following error estimate:

$$\|u - u_h\|_1 + \|p - p_h\|_0 \leqslant C_0 h \|u\|_2 + C_1 h^2 \|p\|_2$$

We notice that the convergence rate for the velocity is linear, hence the extra bubbles bring stability but does not increase approximation order.

7.4 Stabilization techniques to circumvent the Babuska-Brezzi condition

Stabilization techniques typically replace the system:

$$\begin{aligned} Au + B^T p &= f \\ Bu &= 0 \end{aligned}$$

with an alternative system

$$\begin{aligned} Au + B^T p &= f \\ Bu - \epsilon D p &= \epsilon d, \end{aligned}$$

where ϵ is properly chosen and D is a positive, but not necessarily positive definite, matrix.

To see that we obtain a nonsingular system we again multiply the first equation with A^{-1} and then factorize:

$$\begin{aligned} u &= A^{-1}f - A^{-1}B^T p \\ Bu &= BA^{-1}f - BA^{-1}B^T p = \epsilon d + \epsilon D p \\ (BA^{-1}B^T + \epsilon D)p &= BA^{-1}f - \epsilon d \end{aligned}$$

If D is nonsingular then $(BA^{-1}B^T + \epsilon D)$ will be is nonsingular since both D and $BA^{-1}B^T$ are positive (only D is positive definite however).

Factorizing for p we end up with a *Velocity-Schur complement*. Solving for p in the second equation and inserting the expression for p into the first equation we have

$$\begin{aligned} p &= (-\epsilon D)^{-1}(\epsilon d - Bu) \\ &\Downarrow \\ Au + B^T(-\epsilon D)^{-1}(\epsilon d - Bu) &= f \\ (A + \frac{1}{\epsilon} B^T D^{-1} B)u &= f + D^{-1} d \end{aligned}$$

$(A + \frac{1}{\epsilon}B^T D^{-1} B)$ is nonsingular since A is nonsingular and $B^T D^{-1} B$ is positive.

At least, three techniques have been proposed for stabilization. These are:

1. $\nabla \cdot v + \epsilon \Delta p = 0$. Pressure stabilization. Motivated through mathematical intuition (from the convection-diffusion equation).

2. $\nabla \cdot v - \epsilon p = 0$. Penalty method. Typically, one uses the Velocity-Schur complement

3. $\nabla \cdot -\epsilon \frac{\partial p}{\partial t} = 0$. Artificial compressibility. A practical method as one adds the possibility for time stepping.

In other words, these techniques sets D to be

1. $D = A$
2. $D = M$
3. $D = \frac{1}{\Delta t} M$

where A is the stiffness matrix (discrete laplace operator) and M is the mass matrix.

7.5 Exercises

Exercise 7.1. *Show that the conditions* (7.15)-(7.17) *are satisfied for* $V_h = H^1_0$ *and* $Q_h = L^2$.

Exercise 7.2. *Show that the conditions* (7.15)-(7.17) *are satisfied for Taylor–Hood and Mini discretizations. (Note that Crouzeix–Raviart is non-conforming so it is more difficult to prove these conditions for this case.)*

Exercise 7.3. *Condition* (7.18) *is difficult to prove. However, if we assume that* $V_h = L^2$ *and* $Q_h = H^1_0$*, you should be able to prove it. (Hint: This is closely related to Poincare's inequality.)*

Exercise 7.4. *Test other finite elements for the Poiseuille flow problem. Consider* $P_1 - P_0$*,* $P_2 - P_2$*,* $P_2 - P_0$*, as well as the Mini and Crouzeix–Raviart element.*

Exercise 7.5. *Implement stabilization for the Poiseuille flow problem and use first order linear elements for both velocity and pressure.*

Exercise 7.6. *In the previous problem the solution was a second order polynomial in the velocity and first order in the pressure. We may therefore obtain the exact solution and it is therefore difficult to check order of convergence for higher order methods with this solution. In this exercise you should therefore implement the problem* $u = (sin(\pi y), cos(\pi x)$*,* $p = sin(2\pi x)$*, and* $f = -\Delta u - \nabla p$*. Test whether the approximation is of the expected order for* $P_4 - P_3$*,* $P_4 - P_2$*,* $P_3 - P_2$*, and* $P_3 - P_1$*.*

Exercise 7.7. *Implement the Stokes problem with analytical solution* $u = (sin(\pi y), cos(\pi x)$*,* $p = sin(2\pi x)$*, and* $f = -\Delta u - \nabla p$ *on the unit square. Consider the case where you have Dirichlet conditions on the sides 'x=0', 'x=1' and 'y=1' while Neumann is used on the last side (this way we avoid the singular system associated with either pure Dirichlet or pure Neumann problems). Then determine the order of the approximation of wall shear stress on the side 'x=0'. The wall shear stress on the side 'x=0' is* $\nabla u \cdot t$ *where* $t = (0, 1)$ *is the tangent along 'x=0'.*

8 Efficient Solution Algorithms: Iterative methods and Preconditioning

By Anders Logg, Kent–Andre Mardal

To compute the solution of a partial differential equation, we often need to solve a system of linear of equations with a large number of uknowns. The accuracy of the solution increase with the number of unknowns used. Nowadays, unknowns in the order of millions to billions are routinely solved for without the use of (state-of-the-art) high-performance computing. Such computations are fasilitated by the enormous improvements in numerical algorithms and scientific software the last decades.

It should be quite clear that naive Gaussian elimination can not be employed. For a naive Gaussian eliminations implementaton, the number of required floating point operations (FLOPS) scales as the cube of the number of uknowns. Hence, solving a problem with 10^6 unknowns would then require 10^{18} FLOPS which on a modern computer with e.g. 3 GHz still would take about 10 years. As we will see later, such problems may in common cases be solved in just a few seconds. There are two ingrediences in such efficient algorithms: *iterative methods* and *preconditioning*.

Lets therefore consider the numerical solution of large linear systems,

$$Au = b,$$

where the linear system comes from discretization of PDEs. That is, A is a $N \times N$ matrix, and N is between 10^6 and 10^9 in typical simulations. Furthermore, the matrix is normally extremely sparse and contains only $\mathcal{O}(N)$ nonzeros (see Exercise 8.1). It is important to notice that even though A is sparse A^{-1} will in general be full. This is a main reason to consider iterative methods.

8.1 The simplest iterative method: the Richardson iteration

The Richardson iteration[1] is

$$u^n = u^{n-1} - \tau(Au^{n-1} - b), \tag{8.1}$$

where τ is a relaxation parameter that must be determined. Clearly, the method is consistent in the sense that if $u^{n-1} = u$, then $u^n = u$ and the iterative method has converged to the exact solution. It is also clear that each iteration requires the evaluation of A on a vector, in addition to vector addition and scalar multiplication. Hence, one iteration requires the amount of $\mathcal{O}(N)$ FLOPS and only $\mathcal{O}(N)$ of memory. This is a dramatic improvement when compared Gaussian elimination at least if if the

[1] Richardson developed his method prior to computers. In his 1910 paper, where the focus is to predict stresses in a masonry dam, he describes how he uses humans as computational resources. He writes "So far I have paid piece rates for the operation [Laplacian] of about $n/18$ pence per coordinate point, n being the number of digits. As for the rate of working, one of the quickest boys average 2000 operations per week, for numbers of three digits, those done wrong being discounted."

number of iterations are few. The key to obtain few iterations is preconditioning, but lets first consider the Richardson's method without.

The standard approach to analyze iterative methods is to look at what happens with the *error*. Let the error at the n'th iteration be $e^n = u^n - u$. As this is a linear system of equations, we may subtract u from both sides of (8.1) and obtain an equation for the iterative error:

$$e^n = e^{n-1} - \tau A e^{n-1}.$$

We may therefore quantify the error in terms of the L^2-norm as

$$\|e^n\| = \|e^{n-1} - \tau A e^{n-1}\| \leqslant \|I - \tau A\| \|e^{n-1}\|.$$

Clearly, if $\|I - \tau A\| < 1$ then the iteration will be convergent.

Assuming for the moment that A is symmetric and positive definite, then the norm of A in general defined as

$$\|A\| = \max_x \frac{\|Ax\|}{\|x\|}$$

equals the largest eigenvalue of A, λ_{max}. Furthermore, if we assume that the eigenvalues are ordered with respect to increasing value, such that λ_0 and λ_N are the smallest and largest eigenvalue, then the norm of $I - \tau A$,

$$\|I - \tau A\| = \max_x \frac{\|(I - \tau A)x\|}{\|x\|}$$

is attained either for the smallest or largest eigenvalue as either $(1 - \tau\lambda_0)$ or $-(1 - \tau\lambda_N)$. The optimal relaxation parameter τ_{opt} can be stated in terms of the eigenvalues, λ_i, of A. Minimum is attained when $(1 - \tau_{opt}\lambda_0) = -(1 - \tau_{opt}\lambda_N)$ which makes $\tau_{opt} = \frac{2}{\lambda_0 + \lambda_N}$.

Let the convergence factor ρ be defined as

$$\rho = \|I - \tau A\|$$

The convergence factor with an optimal relation is then

$$\rho = \|I - \tau A\| = \max_{\lambda_i} |1 - \tau\lambda_i| = 1 - \tau\lambda_0 = 1 - \frac{2\lambda_0}{\lambda_0 + \lambda_N} = \frac{\lambda_N - \lambda_0}{\lambda_N + \lambda_0} = \frac{\kappa - 1}{\kappa + 1}.$$

Here, $\kappa = \frac{\lambda_N}{\lambda_0}$ is the condition number.

We estimate the error reduction per iteation in terms of the convergence factor as,

$$\|e^n\| = \|(I - \tau A)e^{n-1}\| \leq \rho \|e^{n-1}\|.$$

which leads to

$$\|e^n\| \leqslant (\frac{\kappa - 1}{\kappa + 1})^n \|e^0\|.$$

For iterative methods, we never iterate until the true solution exactly. Instead a convergence criteria needs to be choosen such that the error obtained by the iterative method is less than or at least comparable to the approximation error of the original system. Determining an appropriate convergence criteria is problem dependent and quite often challenging.

Nevertheless, let us assume that we need to reduce the error by a factor of ϵ, that is, we need

$\frac{\|e^n\|}{\|e^0\|} < \epsilon$. From the iteration, we have

$$\|e^n\| \leqslant \rho \|e^{n-1}\| \leqslant \rho^n \|e^0\|. \tag{8.2}$$

An estimate for the number of iterations is then obtained by assuming equality in the equation (8.2) and $\frac{\|e^n\|}{\|e^0\|} = \epsilon$. Then the number of iterations needed to achieve the desired error is:

$$n = \frac{\log \epsilon}{\log \rho} = \frac{\log \epsilon}{\log(\frac{K-1}{K+1})}. \tag{8.3}$$

If n is independent of the resolution of the discretization, the computational cost of the algorithm is $\mathcal{O}(N)$ in FLOPS and memory and the algorithm is *order-optimal*.

The current analysis of the simplest iterative method there is, the Richardson iteration, shows that the efficiency of the method is determined by the condition number of the matrix. In the literature you will find a jungle of methods of which the following are the most famous: the Conjugate Gradient method, the Minimal Residual method, the BiCGStab method, and the GMRES method. It is remarkable that in general the convergence of these methods is determined by the condition number with one exception; the Conjugate Gradient method which often can be estimated in terms of the square root of the condition number. One main advantage is however that these methods do not require the determination of a τ to obtain convergence.

Example 8.1. ***Eigenvalues of an elliptic problem in 1D and 2D.***
Let us consider an elliptic problem:

$$\begin{aligned} u - \Delta u &= f, \quad \text{in } \Omega, \qquad (8.4)\\ \frac{\partial u}{\partial n} &= 0, \quad \text{on } \partial\Omega. \qquad (8.5) \end{aligned}$$

Notice that the lower order term u in front of $-\Delta u$ makes removes the singularity associated with Neumann conditions and that in the continuous case the smallest eigenvalue is 1 (associated with the eigenfunction that is a constant throughout Ω). The following code computes the eigenvalues using linear Lagrangian elements and

Python code

```python
from dolfin import *
from numpy import linalg

for D in [1, 2]:
  for N in [4, 8, 16, 32]:
    if   D == 1:  mesh = UnitIntervalMesh(N)
    elif D == 2:  mesh = UnitSquareMesh(N, N)

    V = FunctionSpace(mesh, "Lagrange", 1)
    u = TrialFunction(V)
    v = TestFunction(V)

    a = u*v*dx  + inner(grad(u), grad(v))*dx
    A = assemble(a)
    e = linalg.eigvals(A.array())
    e.sort()
    c = e[-1] / e[0]

    print "D=\%d, N=\%3d, min eigenvalue=\%5.3f, max eigenvalue=\%5.3f, cond. number=\%5.3f " \% (D, N,
        e[0], e[-1], c)
```

yields the following output:

Output

```
D=1, N=  4, min eigenvalue=0.199, max eigenvalue=14.562,  cond. number=73.041
D=1, N=  8, min eigenvalue=0.111, max eigenvalue=31.078,  cond. number=279.992
D=1, N= 16, min eigenvalue=0.059, max eigenvalue=63.476,  cond. number=1079.408
D=1, N= 32, min eigenvalue=0.030, max eigenvalue=127.721, cond. number=4215.105
D=2, N=  4, min eigenvalue=0.040, max eigenvalue=7.090,   cond. number=178.444
D=2, N=  8, min eigenvalue=0.012, max eigenvalue=7.735,   cond. number=627.873
D=2, N= 16, min eigenvalue=0.003, max eigenvalue=7.929,   cond. number=2292.822
D=2, N= 32, min eigenvalue=0.001, max eigenvalue=7.982,   cond. number=8693.355
```

The output shows that the condition number grows as h^{-2} in both 1D and 2D although the behaviour of the eigenvalues clearly are dimension dependent (see Exercise 8.2). The smallest eigenvalue decrease in both 1D and 2D as $h \to 0$ but at different rates. To obtain eigenvalues corresponding the true eigenvalue we would need to solve a generalized eigenvalue problem as discussed in Chapter 3.

Example 8.2. ***The Richardson iteration applied to a 1D Poisson equation.***
The Richardson iteration on the Poisson equation in 1D, discretized with finite difference method (FDM).

$$Lu = \begin{cases} -u'' = f & \text{for} \quad x \in (0,1) \\ u(0) = u(1) = 0 \end{cases} \tag{8.6}$$

Eigenvalues and eigenfunctions of Lu are $\lambda_k = (k\pi)^2$ and $v_k = \sin(k\pi x)$ for $k \in \mathbb{N}$. When discretizing with FDM we get a $Au = b$ system, where A is a tridiagonal matrix ($A = tridiagonal(-1, 2, -1)$) when the Dirichlet conditions have been eliminated. The discrete and continuous eigenvectors are the same, but the eigenvalues are a little bit different: $\lambda_k = \frac{4}{h^2} sin^2(\frac{k\pi h}{2})$, where h is the step lenght Δx. We find the smallest and largest discrete eigenvalues

$$\lambda_{min}(A) = \pi^2, \quad \lambda_{max}(A) = \frac{4}{h^2}.$$

Let $\tau = \frac{2}{\lambda_{max} + \lambda_{min}}$ then from the analysis above,

$$\|e^n\| \leqslant (\frac{1-K}{1+K})^n \|e^0\|.$$

The below code perform the Richardson iteration for various resolution on the 1D Poisson problem and stops when the convergence criteria $\frac{\|r_k\|}{\|r_0\|} \leq 10^{-6}$ is obtained.

Python code

```
from numpy import *

def create_stiffness_matrix(N):
  h = 1.0/(N-1)
  A = zeros([N,N])
  for i in range(N):
    A[i,i] = 2.0/(h**2)
    if i > 0:
      A[i,i-1] = -1.0/(h**2)
    if i < N-1:
      A[i,i+1] = -1.0/(h**2)
  A = matrix(A)
  return A

Ns = [10, 20, 40, 80, 160, 320]
for N in Ns:
```

```
A = create_stiffness_matrix(N)              # creating matrix
x = arange(0, 1, 1.0/(N))
f = matrix(sin(3.14*x)).transpose()         # right hand side
u0 = matrix(random.random(N)).transpose()   # initial guess
u_prev = u0

eigenvalues = sort(linalg.eigvals(A))       # compute eigenvalues and tau
lambda_max, lambda_min = eigenvalues[-1],  eigenvalues[0]
print "lambda_max ", lambda_max, " lambda_min ", lambda_min
tau = 2/(lambda_max + lambda_min)

norm_of_residual = 1.0                      # make sure the iteration starts
no_iterations= 0
while norm_of_residual > 1.0e-6:
  r = A*u_prev - f                          # compute the residual
  u = u_prev - tau*r                        # the Richardson iteration
  u_prev = u
  norm_of_residual = r.transpose()*r        # check for norm of residual
  no_iterations+=1                          # count no iterations

print "N ", N, " number of iterations ", no_iterations
```

N	λ_{min}	λ_{max}	no. iterations	Estimated FLOPS
10	6.6	317	277	11 10^3
20	8.1	1435	1088	87 10^3
40	8.9	6075	4580	732 10^3
80	9.4	25*10^3	20 10^3	6.4 10^6
160	9.6	101*10^3	84 10^3	53 10^6
320	9.7	407*10^3	354 10^3	453 10^6

Table 8.1: The number of iterations of the Richardson iteration for solving a 1D Poisson problem. The FLOPS is estimated as the number of iterations times four times the number of unknowns, N, as the matrix is tridiagonal and there is both a matrix vector product ($3N$) and a vector addtion involved in (8.1).

We remark that in this example we have initialized the iteration with a random vector because such a vector contains errors at all frequencies. This is recommended practice when trying to estabilish a worst case scenario. Testing the iterative method against a known analytical solution with a zero start vector will often only require smooth error to be removed during the iterations and will therefore underestimate the complications of a real-world problem.

8.1.1 The stopping criteria

In the Example 8.2 we considered the Richardson iteration applied to a Poisson problem in 1D. We saw that in order to stop the iteration we had to choose a stopping criteria. Ideally we would like to stop when the error was small enough. The problem is that the error is uknown. In fact, since $e^n = u^n - u$ we would be able to compute the exact solution if the error was known at the n'th iteration. What is computable is the *residual* at the n'th iteration, defined by

$$r^n = Au^n - f.$$

It is straightforward to show that

$$Ae^n = r^n.$$

But computing e^n from this relation would require the inversion of A (which we try to avoid at all cost since it in general is a $\mathcal{O}(N^3)$ procedure). For this reason, the convergence criteria is typically expressed in terms of some norm of the residual. We may bound the n'th error as

$$\|e^n\| \leq \|A^{-1}\| \|r^n\|.$$

However, estimating $\|A^{-1}\|$ is in general challenging or computationally demanding and therefore usually avoided. To summarize, choosing an appropriate stopping criteria is in general challenging and in practice the choice has to be tailored to concrete application at hand by trial and error.

8.2 The idea of preconditioning

The basic idea of preconditioning is to replace

$$Au = b$$

with

$$BAu = Bb.$$

Both systems have the same solution (if B is nonsingular). However, B should be chosen as a cheap approximation of A^{-1} or at least in such a way that BA has a smaller condition number than A. Furthermore Bu should cost $\mathcal{O}(N)$ operations to evaluate. Obviously, the preconditioner $B = A^{-1}$ would make the condition number of BA be one and the Richardson iteration would converge in one iteration. However, $B = A^{-1}$ is a very computationally demanding preconditioner. We would rather seek preconditioners that are $\mathcal{O}(N)$ in both memory consumption and evaluation.

The generalized Richardson iteration becomes

$$u^n = u^{n-1} - \tau B(Au^{n-1} - b). \tag{8.7}$$

The error in the n-th iteration is

$$e^n = e^{n-1} - \tau BAe^{n-1}$$

and the iteration is convergent if $\|I - \tau BA\| < 1$.

8.2.1 Spectral equivalence and order optimal algorithms

Previously we stated that a good preconditioner is supposed to be similar to A^{-1}. The precise (and most practical) property that is required of a preconditioner is:

- B should be spectrally equivalent with A^{-1}.
- The evaluation of B on a vector, Bv, should be $\mathcal{O}(N)$.
- The storage of B should be $\mathcal{O}(N)$.

Definition 8.1. *Two linear operators or matrices A^{-1} and B, that are symmetric and positive definite are spectral equivalent if:*

$$c_1(A^{-1}v, v) \leqslant (Bv, v) \leqslant c_2(A^{-1}v, v) \quad \forall v \tag{8.8}$$

If A^{-1} and B are spectral equivalent, then the condition number of the matrix BA is $\kappa(BA) \leqslant \frac{c_2}{c_1}$.

If the preconditioner B is spectrally equivalent with A^{-1} then the preconditioned Richardson iteration yields and order optimal algorithm. To see this, we note that $e^n = (I - \tau BA)e^{n-1}$. We can estimate the behavior of e^n by using the A-norm, $\rho_A = \|I - \tau BA\|_A$. Then we get

$$\|e^n\|_A \leqslant \rho_A \|e^{n-1}\|_A.$$

Hence, if the condition number is independent of the discretization then the number of iterations as estimated earlier in (8.3) will be bounded independently of the discretization.

In general, if A is a discretization of $-\Delta$ on a quasi-uniform mesh then both multigrid methods and domain decomposition methods will yield preconditioners that are spectrally equivalent with the inverse and close to $\mathcal{O}(N)$ in evaluation and storage. The gain in using a proper preconditioner may provide speed-up of several orders of magnitude, see Example 8.3.

8.3 *Krylov methods and preconditioning*

For iterative methods, any method involving linear iterations may be written as a Richardson iteration with a preconditioner. However, iterative methods like Conjugate Gradient method, GMRES, Minimal Residual method, and BiCGStab, are different. These are nonlinear iterations where for instance the relaxation parameter τ changes during the iterations and are in fact often choosen optimally with respect to the current approximation. Avoiding the need to determine a fixed relaxation parameter prior to the iterations is of course a huge practical benefit. Still, the convergence in practice can usually be roughly estimated by the convergence analysis above for the Richardson iteration.

We will not go in detail on these methods. We only remark that also with these methods it is essential with a good preconditioning technique in order for efficient computations. Furthermore, some of them have special requirements and in some cases it is well-known what to use.

General Advice for usage of different methods: We classify the methods according to the matrices they are used to solve.

- If a matrix is Symmetric Positive Definite(SPD), i.e., $A = A^T$ and $x^T A x \geq 0 \ \forall x$ the the *Conjugate Gradient method* (CG) is the method of choice. CG needs an SPD preconditioner, see also Exercise 8.6.

- If a matrix is Symmetric but indefinite, i.e. $A = A^T$ but both positive and negative eigenvalues then the *Minimal Residual method* (MR) is the best choice. MR requires an SPD preconditioner, see also Exercise 8.9.

- If the matrix is positive, i.e., $x^T A x \geq 0 \ \forall x$ which is often the case for convection-diffusion problems or flow problems then *GMRES* with either ILU or AMG are often good, but you might need to experiment, see also Exercise 8.7.

- For matrices that are both nonsymmetric and indefinite there is a jungle of general purpose methods but they may be categories in two different families. In our experience the *BiCGStab* and GMRES methods are the two most prominent algorithms in these families. GMRES is relatively roboust but may stagnate. BiCGStab may break down. GMRES has a parameter 'the number of search vectors' that may be tuned.

Most linear algebra libraries for high performance computing like for instance PETSc, Trilinos, Hypre have these algorithms implemented. They are also implemented in various libraries in Python and Matlab. There is usually no need to implement these algorithms yourself.

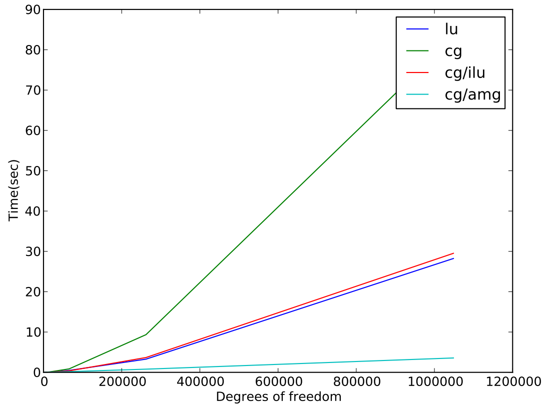


Figure 8.1: CPU time (in seconds) for solving a linear system of equation with N degrees of freedom (x-axis) for different solvers

Example 8.3 (CPU times of different algorithms)**.** *In this example we will solve the problem*

$$\begin{aligned} u - \Delta u &= f, \quad in \quad \Omega \\ \frac{\partial u}{\partial n} &= 0, \quad on \quad \partial\Omega \end{aligned}$$

where Ω is the unit square with first order Lagrange elements. The problem is solved with four different methods:

- *a LU solver,*
- *Conjugate Gradient method,*
- *Conjugate Gradient method with an ILU preconditioner, and*
- *Conjugate Gradient method with an AMG preconditioner,*

for $N = 32^2, 64^2, 128^2, 256^2, 512^2, 1024^2$, where N is the number of degrees of freedom.
Figure 8.1 shows that there is a dramatic difference between the algorithms. In fact the Conjugate gradient (CG) with an AMG preconditioner is over 20 times faster then the slowest method, which is the CG solver without preconditioner. One might wonder why the LU solver is doing so well in this example when it costs $\mathcal{O}(N^2) - \mathcal{O}(N^3)$. However, if we increase the number of degrees of freedom, then the method would slow down compared to the other methods. The problem is then that it would require too much memory and the program would probably crash.

Python code

```
from dolfin import *
import time
lu_time = []; cgamg_time = []
cg_time = []; cgilu_time = []
Ns = []

parameters["krylov_solver"]["relative_tolerance"] = 1.0e-8
parameters["krylov_solver"]["absolute_tolerance"] = 1.0e-8
parameters["krylov_solver"]["monitor_convergence"] = False
parameters["krylov_solver"]["report"] = False
parameters["krylov_solver"]["maximum_iterations"] = 50000
```

```

def solving_time(A,b, solver):
  U = Function(V)
  t0 = time.time()
  if len(solver) == 2:
    solve(A, U.vector(), b, solver[0], solver[1]);
  else:
    solve(A, U.vector(), b, solver[0]);
  t1 = time.time()
  return t1-t0

for N in [32, 64, 128, 256, 512, 1024]:

  Ns.append(N)

  mesh = UnitSquare(N, N)
  print " N ", N, " dofs ", mesh.num_vertices()
  V = FunctionSpace(mesh, "Lagrange", 1)
  u = TrialFunction(V)
  v = TestFunction(V)

  f = Expression("sin(x[0]*12) - x[1]")
  a = u*v*dx  + inner(grad(u), grad(v))*dx
  L = f*v*dx

  A = assemble(a)
  b = assemble(L)

  t2 = solving_time(A,b, ["lu"])
  print "Time for lu ", t2
  lu_time.append(t2)

  t2 = solving_time(A, b, ["cg"])
  print "Time for cg ", t2
  cg_time.append(t2)

  t2 = solving_time(A, b, ["cg", "ilu"])
  print "Time for cg/ilu ", t2
  cgilu_time.append(t2)

  t2 = solving_time(A, b, ["cg", "amg"])
  print "Time for cg/amg ", t2
  cgamg_time.append(t2)


import pylab

pylab.plot(Ns, lu_time)
pylab.plot(Ns, cg_time)
pylab.plot(Ns, cgilu_time)
pylab.plot(Ns, cgamg_time)
pylab.xlabel(’Unknowns’)
pylab.ylabel(’Time(sec)’)
pylab.legend(["lu", "cg", "cg/ilu", "cg/amg"])
pylab.show()

pylab.loglog(Ns, lu_time)
pylab.loglog(Ns, cg_time)
pylab.loglog(Ns, cgilu_time)
pylab.loglog(Ns, cgamg_time)
pylab.legend(["lu", "cg", "cg/ilu", "cg/amg"])
```

$\epsilon\backslash$ N	64	128	256	512	1024
1.0e-1	1.3e-02 (1.1e-02)	1.4e-02 (3.5e-02)	8.8e-03 (1.4e-01)	3.4e-03 (5.9e-01)	1.1e-02 (2.5e+00)
1.0e-2	1.2e-03 (1.0e-02)	2.0e-03 (3.7e-02)	1.3e-03 (1.5e-01)	3.5e-03 (5.8e-01)	3.7e-04 (2.7e+00)
1.0e-3	3.6e-04 (1.1e-02)	3.1e-04 (3.9e-02)	2.6e-04 (1.6e-01)	2.7e-04 (6.3e-01)	3.7e-04 (2.7e+00)
1.0e-4	3.4e-04 (1.2e-02)	8.5e-05 (4.5e-02)	2.4e-05 (1.8e-01)	3.4e-05 (6.7e-01)	1.4e-05 (2.9e+00)
1.0e-5	3.4e-04 (1.2e-02)	8.4e-05 (4.7e-02)	2.1e-05 (1.9e-01)	5.4e-06 (7.6e-01)	2.8e-06 (3.1e+00)
1.0e-6	3.4e-04 (1.3e-02)	8.4e-05 (5.0e-02)	2.1e-05 (2.1e-01)	5.3e-06 (8.1e-01)	1.3e-06 (3.3e+00)

Table 8.2: The error $\|u - u_h^n\|$ and corresponding CPU time in parentesis when solving a Poisson problem with homogenuous Dirichlet conditions.

```
pylab.savefig('tmp_cpu.pdf')
pylab.show()
```

When we employ iterative methods, we need to specify the convergence criterion. This is often not an easy task. We have the continuous solution u, the discrete solution u_h, and the appropriate discrete solution, u_h^n found by an iterative method at iteration n. Obviously, we may estimate the error as

$$\|u - u_h^n\| \leq \|u - u_h\| + \|u_h - u_h^n\|,$$

and it does make sense that the values of $\|u - u_h\|$ and $\|u_h - u_h^n\|$ are balanced. Still both terms may be hard to estimate in challenging applications. In practice, an appropriate convergence criterion is usually found by trial and error by choosing a stopping criterion based on the residual. Let us therefore consider a concrete example and consider $\|u - u_h^n\|$ as a function of the mesh resolution and a varying convergence criterion.

Table 8.2 shows the error and the corresponding CPU timings when solving a Poisson problem at various resolutions and convergence criteria. Here, the convergence criteria is chosen as reducing the relative residual, i.e., $\frac{\|r_k\|}{\|r_0\|}$ by the factor ϵ. This convergence criteria is very common, in particular for stationary problems. There are several things to note here. For coarse resolution, `N=64`, the error stagnates somewhere between $1.0e-3$ and $1.0e-4$ and this stagnation marks where an appropriate stopping criteria is. It is however worth noticing that solving it to a criteria that is $1.0e-6$ is actually only about 30% more computationally demanding than $1.0e-3$. This is due to the fact that we have a very efficient method that reduces the error by about a factor 10 per iteration. If we consider the fine resolution, `N=1024`, we see that the stagnation happens later and that we may not even have reached the stagnating point even at $\epsilon = 1.0e-6$. We also notice that the decreasing ϵ in this case only lead to a moderate growth in CPU time. If we look closer at the table, we find that the stagnation point follows a staircase pattern. The code used to generate the table is as follows:

Python code

```python
from dolfin import *

def boundary(x, on_boundary):
    return on_boundary

parameters["krylov_solver"]["relative_tolerance"] = 1.0e-18
parameters["krylov_solver"]["absolute_tolerance"] = 1.0e-18
parameters["krylov_solver"]["monitor_convergence"] = True
parameters["krylov_solver"]["report"] = True
#parameters["krylov_solver"]["maximum_iterations"] = 50000
epss = [1.0e-1, 1.0e-2, 1.0e-3, 1.0e-4, 1.0e-5, 1.0e-6]
data = {}
Ns= [64, 128, 256, 512, 1024]
```

```
#Ns= [8, 16, 32, 64]
for N in Ns:
  for eps in epss:
    parameters["krylov_solver"]["relative_tolerance"] = eps

    mesh = UnitSquareMesh(N, N)
    V = FunctionSpace(mesh, "P", 1)
    u = TrialFunction(V)
    v = TestFunction(V)

    u_ex = Expression("sin(3.14*x[0])*sin(3.14*x[1])", degree=3)
    f = Expression("2*3.14*3.14*sin(3.14*x[0])*sin(3.14*x[1])", degree=3)
    a = inner(grad(u), grad(v))*dx
    L = f*v*dx

    U = Function(V)

    A = assemble(a)
    b = assemble(L)

    bc = DirichletBC(V, u_ex, boundary)
    bc.apply(A)
    bc.apply(b)

    t0 = time()
    solve(A, U.vector(), b, "gmres", "amg")
    t1 = time()

    cpu_time = t1-t0
    error_L2 = errornorm(u_ex, U, ’L2’, degree_rise=3)
    data[(N, eps)] = (error_L2, cpu_time)

for eps in epss:
  for N in Ns:
    D1, D2 = data[(N, eps)]
    print " %3.1e (%3.1e) " % (D1, D2),
    print ""

```

Example 8.4. ***Eigenvalues of the preconditioned system.*** *It is often interesting to assess the condition number of the preconditioned system, BA. If the preconditioner is a matrix and the size of the system is moderate we may be able to estimate the condition number of BA using NumPy, Matlab or Octave. However, when our preconditioner is an algorithm representing a linear operator, such as in the case of multigrid, then this is not possible. However, as described in* **?***, egenvalues may be estimated as a bi-product of the Conjugate Gradient method. Without going into the algorithmic details of the implmementation, we mention that this is implemented in the FEniCS module* `cbc.block`*, see* **?***. The following code shows the usage.*

Python code

```
from dolfin import *
from block.iterative import ConjGrad
from block.algebraic.petsc import ML
from numpy import random

def boundary(x, on_boundary):
    return on_boundary

class Source(Expression):
    def eval(self, values, x):
        dx = x[0] - 0.5; dy = x[1] - 0.5
        values[0] = 500.0*exp(-(dx*dx + dy*dy)/0.02)

Ns = [8, 16, 32, 64, 128, 256, 512, 1024]
for N in Ns:
    mesh = UnitSquareMesh(N,N)
    V = FunctionSpace(mesh, "CG", 1)

    # Define variational problem
    v = TestFunction(V)
    u = TrialFunction(V)
    f = Source(degree=3)
    a = dot(grad(v), grad(u))*dx
    L = v*f*dx
    bc = DirichletBC(V, Constant(0), boundary)

    # Assemble matrix and vector, create precondition and start vector
    A, b = assemble_system(a,L, bc)
    B = ML(A)
    x = b.copy()
    x[:] = random.random(x.size(0))

    # solve problem and print out eigenvalue estimates.
    Ainv = ConjGrad(A, precond=B, initial_guess=x, tolerance=1e-8, show=2)
    x = Ainv*b
    e = Ainv.eigenvalue_estimates()
    print "N=%d iter=%d K=%.3g" % (N, Ainv.iterations, e[-1]/e[0])
```

In this example we see that the condition number increases logaritmic from 1.1 to 2.1 as the `N` *increases from 8 to 1024. The* `AMG` *preconditioner has better performance and does not show logaritmic growth. For indefinite symmetric systems, the* `CGN` *method provides the means for estimating the condition number, c.f., the* `cbc.block` *documentation.*

8.3.1 Insight from Functional Analysis

In the previous Chapters 6 and 7 we have discussed the well-posedness of the convection-diffusion equations and the Stokes problem. In both cases, the problems were well-posed - meaning that the differential operators as well as their inverse were continuous. However, when we discretize the problems we get matrices where the condition number grows to infinity as the element size goes to zero. This seem to contradict the well-posedness of our discrete problems and may potentially destroy both the accuracy and efficiency of our numerical algorithms. Functional analysis explains this apparent contradiction and explains how the problem is circumvented by preconditioning.

Let us now consider the seeming contradiction in more precise mathematical detail for the Poisson problem with homogeneous Dirichlet conditions: Find u such that

$$-\Delta u = f, \quad \text{in } \Omega, \tag{8.9}$$

$$u = 0, \quad \text{on } \partial\Omega. \tag{8.10}$$

We know from Lax-Milgram's theorem that the weak formulation of this problem: Find $u \in H^1_0$ such that

$$a(u, v) = b(v), \quad \forall v \in H^1_0.$$

where

$$\begin{aligned} a(u,v) &= \int_\Omega \nabla u \cdot \nabla v \, dx, && (8.11) \\ b(v) &= \int_\Omega f v \, dx, && (8.12) \end{aligned}$$

is well-posed because

$$\begin{aligned} a(u,u) &\geq \alpha |u|_1^2 \,, \quad \forall u \in H_0^1 && (8.13) \\ a(u,v) &\leq C |u|_1 |v|_{H_0^1} \quad \forall u,v \in H_0^1 \,. && (8.14) \end{aligned}$$

Here $|\cdot|_1$ denotes the H^1 semi-norm which is known to be a norm on H_0^1 due to Poincare. The well-posedness is in this case stated as

$$|u|_{H_0^1} \leq \frac{1}{\alpha} \|f\|_{H^{-1}}. \tag{8.15}$$

In other words, $-\Delta$ takes a function u in H_0^1 and returns a function $f = -\Delta u$ which is in H^{-1}. We have that $\|f\|_{-1} = \| -\Delta u\|_{-1} \leq C\|u\|_1$. Also, $-\Delta^{-1}$ takes a function f in H^{-1} and returns a function $u = (-\Delta)^{-1} f$ which is in H_0^1. We have that $\|u\|_1 = \|(-\Delta)^{-1} f\|_1 \leq \frac{1}{\alpha}\|f\|_{-1}$. In fact, in this case $\alpha = C = 1$.

This play with words and symbols may be formalized by using operator norms that are equivalent with matrix norms. Let $B \in \mathbb{R}^{n,m}$ then

$$\|B\|_{\mathcal{L}(\mathbb{R}^m,\mathbb{R}^n)} = \max_{x \in \mathbb{R}^m} \frac{\|Bx\|_{\mathbb{R}^n}}{\|x\|_{\mathbb{R}^m}}$$

Here $\mathcal{L}(\mathbb{R}^m,\mathbb{R}^n)$ denotes the space of all $m \times n$ matrices.

Analogously, we may summarize the mapping properties of $-\Delta$ and $(-\Delta)^{-1}$ in terms of the conditions of Lax-Milgram's theorem as

$$\| -\Delta\|_{\mathcal{L}(H_0^1, H^{-1})} \leq C \quad \text{and} \quad \|(-\Delta)^{-1}\|_{\mathcal{L}(H^{-1}, H_0^1)} \leq \frac{1}{\alpha} \,. \tag{8.16}$$

where $\mathcal{L}(X,Y)$ denotes the space of bounded linear operators mapping X to Y. In other words, $-\Delta$ is a bounded linear map from H_0^1 to H^{-1} and $(-\Delta)^{-1}$ is a bounded linear map from H^{-1} to H_0^1. This is a crucial observation in functional analysis that, in contrast to the case of a matrix which is a bounded linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, an operator may be map from one space to another.

From Chapter 3 we know that the eigenvalues and eigenvectors of $-\Delta$ with homogeneous Dirichlet conditions on the unit interval in 1D are $\lambda_k = (\pi k)^2$ and $e_k = sin(\pi k x)$, respectively. Hence the eigenvalues of $-\Delta$ obviously tend to ∞ as k grows to ∞ and similarly the eigenvalues of $(-\Delta)^{-1}$ accumulate at zero as $k \to \infty$. Hence the spectrum of $-\Delta$ is unbounded and the spectrum of $(-\Delta)^{-1}$ has an accumulation point at zero. Still, the operator $-\Delta$ and its inverse are bounded from a functional analysis point of view, in the sense of (8.18).

Let us for the moment assume that we have access to an operator B with mapping properties that are inverse to that of $A = -\Delta$, i.e.,

$$\|B\|_{\mathcal{L}(H^{-1}, H_0^1)} \quad \text{and} \quad \|B^{-1}\|_{\mathcal{L}(H_0^1, H^{-1})}. \tag{8.17}$$

Then it follows directly that

$$\|BA\|_{\mathcal{L}(H_0^1,H_0^1)} \quad \text{and} \quad \|(BA)^{-1}\|_{\mathcal{L}(H_0^1,H_0^1)}. \tag{8.18}$$

and the condition number

$$\kappa(BA) = \frac{\max_i \lambda_i(BA)}{\min_i \lambda_i(BA)} = \|BA\|_{\mathcal{L}(H_0^1,H_0^1)} \|(BA)^{-1}\|_{\mathcal{L}(H_0^1,H_0^1)}$$

would be bounded. In the discrete case, the mapping property (8.17) translates to the fact that B should be spectrally equivalent with the inverse of A when B and A are both positive.

While the above discussion is mostly just a re-iteration of the concept of spectral equivalence in the discrete case when the PDEs are elliptic, the insight from functional analysis can be powerful for systems of PDEs. Let us consider the Stokes problem from Chapter 7. The problem reads:

$$\mathcal{A}\begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} -\Delta & -\nabla \\ \nabla\cdot & 0 \end{bmatrix}\begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} u \\ p \end{bmatrix}$$

As discussed in Chapter 7

$$\mathcal{A} : H_0^1 \times L^2 \to H^{-1} \times L^2$$

was a bounded linear mapping with a bounded inverse. Therefore, a preconditioner can be constructed as

$$\mathcal{B} = \begin{bmatrix} (-\Delta)^{-1} & 0 \\ 0 & I \end{bmatrix}$$

Clearly

$$\mathcal{B} : H^{-1} \times L^2 \to H_0^1 \times L^2$$

and is therefore a suitable preconditioner. However, we also notice that $\mathcal{A}$ and $\mathcal{B}^{-1}$ are quite different. $\mathcal{A}$ is indefinite and has positive and negative egenvalues, while $\mathcal{B}$ is clearly positive. Hence, the operators are not spectrally equivalent. Exercise 8.9 looks deeper into this construction of preconditioners for Stokes problem. A more comprehensive description of this technique can be found in **?**.

8.4 Exercises

Exercise 8.1. *Estimate ratio of non-zeros per unknown of the stiffness matrix on the unit square with Lagrangian elements of order 1, 2, 3 and 4. Hint: the number of non-zeros can be obtained from the function 'nnz' of a matrix object.*

Exercise 8.2. *Compute the smallest and largest eigenvalues of the mass matrix and the stiffness matrix in 1D, 2D and 3D. Assume that the condition number is on the form $\kappa \approx Ch^{\alpha}$, where C and α may depend on the number of dimentions in space. Finally, compute the corresponding condition numbers. Does the condition number have the same dependence on the number of dimentions in space?*

Exercise 8.3. *Repeat Exercise 8.2 but with Lagrange elements of order 1, 2 and 3. How does the order of the polynomial affect the eigenvalues and condition numbers.*

Exercise 8.4. *Compute the eigenvalues the discretized Stokes problem using Taylor-Hood elements. Note that the problem is indefinite and that there are both positive and negative eigenvalues. An appropriate condition number is:*

$$\kappa = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

where λ_i are the eigenvalues of A. Compute corresponding condition numbers for the Mini and Crouzeix-Raviart elements. Are the condition numbers similar?

Exercise 8.5. *Implement the Jacobi iteration for a 1D Poisson problem with homogeneous Dirichlet conditions. Start the iteration with an initial random vector and estimate the number of iterations required to reduce the L_2 norm of the residual with a factor 10^4. For relevant code see Example 8.3.*

Exercise 8.6. *Test CG method without preconditioer, with ILU preconditioner and with AMG preconditioner for the Poisson problem in 1D and 2D with homogeneous Dirichlet conditions, with respect to different mesh resolutions. Do some of the iterations suggest spectral equivalence?*

Exercise 8.7. *Test CG, BiCGStab, GMRES with ILU, AMG, and Jacobi preconditioning for*

$$\begin{aligned} -\mu\Delta u + v\nabla u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

Where Ω is the unit square, $v = c\sin(7x)$, and c varies as $1, 10, 100, 1000, 10000$ and the mesh resolution h varies as $1/8, 1/16, 1/32, 1/64$. You may assume homogeneous Dirichlet conditions.

Exercise 8.8. *The following code snippet shows the assembly of the matrix and preconditioner for a Stokes problem:*

Python code

```python
a = inner(grad(u), grad(v))*dx + div(v)*p*dx + q*div(u)*dx
L = inner(f, v)*dx

# Form for use in constructing preconditioner matrix
b = inner(grad(u), grad(v))*dx + p*q*dx

# Assemble system
A, bb = assemble_system(a, L, bcs)

# Assemble preconditioner system
P, btmp = assemble_system(b, L, bcs)

# Create Krylov solver and AMG preconditioner
solver = KrylovSolver("tfqmr", "amg")

# Associate operator (A) and preconditioner matrix (P)
solver.set_operators(A, P)

# Solve
U = Function(W)
solver.solve(U.vector(), bb)
```

Here, "tfqmr" is a variant of the Minimal residual method and "amg" is an algebraic multigrid implementation in HYPRE. Test, by varying the mesh resolution, whether the code produces an order–optimal preconditioner. HINT: You might want to change the "parameters" as done in Example 8.3:

Python code

```python
# Create Krylov solver and AMG preconditioner
solver = KrylovSolver("tfqmr", "amg")
solver.parameters["relative_tolerance"] = 1.0e-8
solver.parameters["absolute_tolerance"] = 1.0e-8
solver.parameters["monitor_convergence"] = True
solver.parameters["report"] = True
solver.parameters["maximum_iterations"] = 50000
```

Exercise 8.9. *Consider the mixed formulation of linear elasticity that is appropriate when λ is large compared to μ. That is,*

Python code

```
a = inner(grad(u), grad(v))*dx + div(v)*p*dx + q*div(u)*dx - 1/lam*p*q*dx
L = inner(f, v)*dx
```

Create two preconditioners:

Python code

```
b1 = inner(grad(u), grad(v))*dx + p*q*dx
b2 = inner(grad(u), grad(v))*dx + 1/lam*p*q*dx
```

Compare the efficiency of the different preconditioners when increasing the resolution and when $\lambda \to \infty$. Can you explain why the first preconditioner is the best?

9 Linear elasticity and singular problems

By Anders Logg, Kent–Andre Mardal

9.1 Introduction

Let us consider an elastic body Ω_0 that is being deformed under a load to become Ω. the deformation χ of a body in the undeformed state Ω_0 to deformed state Ω. A point in the body has then moved

$$u = x - X, \tag{9.1}$$

by definition this is *displacement field*. An illustration is shown in Figure 9.1.

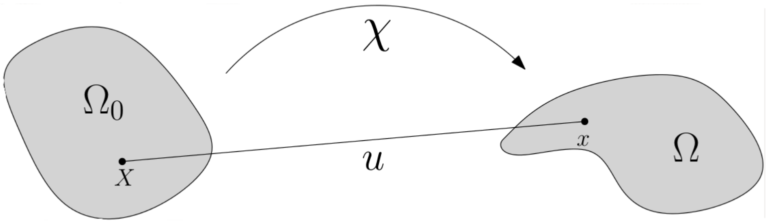


Figure 9.1: Deforming body and displacement vector u.

Here, the domain $\Omega_0 \subset \mathbb{R}^3$. From continuum mechanics, the elastic deformation is modelled by the stress tensor σ which is a symmetric 3×3 tensor. In equilibrium (i.e. no accelration terms) the Newton's second law states the balance of forces as:

$$\begin{aligned} \operatorname{div} \sigma &= f, \quad \text{in } \Omega, \\ \sigma \cdot n &= g, \quad \text{on } \partial\Omega, \end{aligned}$$

where f and g are body and surface forces, respectively and n is the outward normal vector.

For small deformations of an isotropic media, Hooke's law is a good approximation. Hooke's law states that

$$\sigma = 2\mu\epsilon(u) + \lambda \operatorname{tr}(\epsilon(u))\delta.$$

Here, $\epsilon(u)$ is the strain tensor or the symmetric gradient:

$$\epsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^T),$$

μ and λ are the Lame constants, tr is the trace operator (the sum of the diagonal matrix entries), u is the displacement, and

$$\delta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

From Newton's second law and Hooke's law we arrive directly at the equation of linear elasticity:

$$-2\mu(\nabla \cdot \epsilon(u)) - \lambda \nabla(\nabla \cdot u) = f. \tag{9.2}$$

The equation of linear elasticity (9.2) is an elliptic equation, but there are crucial differences between this equation and a standard elliptic equation like $-\Delta u = f$. These differences often cause problems in a numerical setting. To explain the numerical issues we will here focus on the differences between the three operator:

1. $-\Delta = \nabla \cdot \nabla = \operatorname{div} \operatorname{grad}$,

2. $\nabla \cdot \epsilon = \nabla \cdot (\frac{1}{2}(\nabla + (\nabla^T))$,

3. $\nabla \cdot \operatorname{tr} \epsilon = \nabla \nabla \cdot = \operatorname{grad} \operatorname{div}$.

In particular, the differences between the operators in 1. and 2. is that $\nabla \cdot \epsilon$ has a larger kernel than $-\Delta$. The kernel consists of rigid motions and this leads to the usage of of one of Korn's lemmas. This is the subject of Section 9.2. The kernel of the operators grad div and div grad are also different but here in fact the kernel of grad div is infinite dimentional and this has different consequences for the numerical algorithms which not necessarily pick up this kernel at all. This is discussed in Section 9.3.

9.2 The operator $\nabla \cdot \epsilon$ and rigid motions

The challenge with the handling of the $\nabla \cdot \epsilon$ operator is the handling of the singularity in the case of pure Neumann conditions. Let us therefore start with the simpler problem of the Poisson problem with Neumann conditions, i.e.,

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega, \qquad (9.3) \\ \frac{\partial u}{\partial n} &= g, \quad \text{on } \partial\Omega. \qquad (9.4) \end{aligned}$$

It is easy to see that this problem is singular: Let u be a solution of the above equation, then $u + C$ with $C \in \mathbb{R}$ is also a solution because $-\Delta u = \Delta(u + C) = f$ and $\frac{\partial u}{\partial n} = \frac{\partial(u+C)}{\partial n} = g$. Hence, the solution is only determined up to a constant. This means that the kernel is 1-dimentional.

A proper formulation of the above problem can be obtained by using the method of Lagrange multipliers to fixate the element of the 1-dimentional kernel. The following weak formulation is well-posed: Find $u \in H^1$ and $\lambda \in \mathbb{R}$ such that

$$\begin{aligned} a(u, v) + b(\lambda, v) &= f(v) \quad \forall v \in H^1 \qquad (9.5) \\ b(u, \gamma) &= 0, \quad \forall \gamma \in \mathbb{R}. \qquad (9.6) \end{aligned}$$

Here,

$$a(u,v) = (\nabla u, \nabla v), \tag{9.7}$$

$$b(\lambda, v) = (\lambda, v), \tag{9.8}$$

$$f(v) = (f,v) + \int_{\partial\Omega} gv ds. \tag{9.9}$$

Hence, the method of Lagrange multipliers turns the original problem into a saddle problem similar that in Chapter 7. However, in this case the Brezzi conditions are easily verified. We remark however that this formulation makes the problem indefinite rather than positive definite and for this reason alternative techniques such as pin-pointing is often used instead. We will not avocate this approach as it often causes numerical problems. Instead, we include a code example that demonstrate how this problem can be implemented with the method of Lagrange multipliers in FEniCS.

Python code

```python
from dolfin import *

mesh = UnitSquareMesh(64, 64)

# Build function space with Lagrange multiplier
P1 = FiniteElement("Lagrange", mesh.ufl_cell(), 1)
R = FiniteElement("Real", mesh.ufl_cell(), 0)
W = FunctionSpace(mesh, P1 * R)

# Define variational problem
(u, c) = TrialFunction(W)
(v, d) = TestFunctions(W)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)", degree=2)
g = Expression("-sin(5*x[0])", degree=2)
a = (inner(grad(u), grad(v)) + c*v + u*d)*dx
L = f*v*dx + g*v*ds

# Compute solution
w = Function(W)
solve(a == L, w)
(u, c) = w.split()

# Plot solution
plot(u, interactive=True)
```

The kernel of the ϵ operator is the space of rigid motions, RM. The space consists of translations and rotations. Rigid motions are on the following form in 2D and 3D:

$$\mathrm{RM}_{2D} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} + a_2 \begin{bmatrix} -y \\ x \end{bmatrix}, \tag{9.10}$$

$$\mathrm{RM}_{3D} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} 0 & a_3 & a_4 \\ -a_3 & 0 & a_5 \\ -a_4 & -a_5 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{9.11}$$

Hence, the kernel in 2D is three-dimentional and may be expressed as above in terms of the degrees of freedom (a_0, a_1, a_2) whereas the kernel in 3D is six-dimentional $(a_0, \dots, a_5)$.

The Korn's lemmas states suitable conditions for solvability. Here, we include two of the three inequalities typically listed.

- The first lemma: For all $u \in H^1 \backslash$ RM we have that $\|\epsilon(u)\| \geq C\|u\|_1$.

- The second lemma: For all $u \in H^1_0$ we have that $\|\epsilon(u)\| \geq C\|u\|_1$.

These lemmas should be compared with the Poincare's lemma and the equivalence of the $|\cdot|_1$ and $\|\cdot\|_1$ norms. The second lemma states that when we have homogenous Dirichlet conditions we obtain a well-posed problem in a similar manner as for a standard elliptic problem. This case is often called fully-clamped conditions. For the Neumann problem, however, coersivity is not obtained unless we remove the complete set of rigid motions for the function space used for trial and test functions. Removing the rigid motions is most easily done by using the method of Lagrange multipliers.

Let us now consider a weak formulation of the linear elasticity problem and describe how to implement it in FEniCS. For now we consider the case where λ and μ are of comparable magnitude. In the next section we consider the case where $\lambda \gg \mu$. The weak formulation of the linear elasticity problem is: Find $u \in H^1$ and $r \in \mathrm{RM}$ such that

$$\begin{aligned} a(u,v) + b(r,v) &= f(v), \quad \forall v \in H^1, && (9.12)\\ b(s,u) &= 0, \quad \forall s \in \mathrm{RM}\,. && (9.13) \end{aligned}$$

Here,

$$\begin{aligned} a(u,v) &= \mu(\epsilon(u), \epsilon(v)) + \lambda(\operatorname{div} u, \operatorname{div} v) && (9.14)\\ b(r,v) &= (r,v), && (9.15)\\ f(v) &= (f,v) + \int_{\partial\Omega} gv ds. && (9.16) \end{aligned}$$

As we know from Chapter 7, this is a saddle point problem and we need to comply with the Brezzi conditions. Verifying these conditions are left as Exercise 9.4.

Example 9.1. *Our brain and spinal cord is floating in a water like fluid called the cerebrospinal fluid. While the purpose of this fluid is not fully known, it is known that the pressure in the fluid oscillates with about 5-10 mmHg during a cardic cycle which is approximately one second, c.f., e.g.,* **?***. The Youngs' modulus has been estimated 16 kPa and 1 mmHg ≈ 133 Pa, c.f., e.g.,* **?***. To compute the deformation of the brain during a cardiac cycle we consider solve the linear elasticity problem with Neumann condtions set as pressure of 1 mm Hg and ... The following code shows the implmentation in FEniCS. The mesh of the brain was in this case obtained from a T1 magnetic ressonance image and segmentation was performed by using FreeSurfer.*

Python code

```python
from fenics import *

mesh = Mesh('mesh/res32.xdmf')  # mm

plot(mesh,interactive=True)

# Since the mesh is in mm pressure units in pascal must be scaled by alpha = (1e6)**(-1)
alpha = (1e6)**(-1)

# Mark boundaries
class Neumann_boundary(SubDomain):
  def inside(self, x, on_boundry):
    return on_boundry

mf = FacetFunction("size_t", mesh)
mf.set_all(0)

Neumann_boundary().mark(mf, 1)
ds = ds[mf]
```

```

# Continuum mechanics
E = 16*1e3 *alpha
nu = 0.25
mu, lambda_ = Constant(E/(2*(1 + nu))), Constant(E*nu/((1 + nu)*(1 - 2*nu)))
epsilon = lambda u: sym(grad(u))

p_outside = 133 *alpha
n = FacetNormal(mesh)
f = Constant((0, 0, 0))

V = VectorFunctionSpace(mesh, "Lagrange", 1)

# --------------- Handle Neumann-problem --------------- #
R = FunctionSpace(mesh, ’R’, 0)            # space for one Lagrange multiplier
M = MixedFunctionSpace([R]*6)              # space for all multipliers
W = MixedFunctionSpace([V, M])
u, rs = TrialFunctions(W)
v, ss = TestFunctions(W)

# Establish a basis for the nullspace of RM
e0 = Constant((1, 0, 0))        # translations
e1 = Constant((0, 1, 0))
e2 = Constant((0, 0, 1))

e3 = Expression((’-x[1]’, ’x[0]’, ’0’)) # rotations
e4 = Expression((’-x[2]’, ’0’, ’x[0]’))
e5 = Expression((’0’, ’-x[2]’, ’x[1]’))
basis_vectors = [e0, e1, e2, e3, e4, e5]

a = 2*mu*inner(epsilon(u),epsilon(v))*dx + lambda_*inner(div(u),div(v))*dx
L = inner(f, v)*dx + p_outside*inner(n,v)*ds(1)

# Lagrange multipliers contrib to a
for i, e in enumerate(basis_vectors):
  r = rs[i]
  s = ss[i]
  a += r*inner(v, e)*dx + s*inner(u, e)*dx

# -------------------------------------------------------- #

# Assemble the system
A = PETScMatrix()
b = PETScVector()
assemble_system(a, L, A_tensor=A, b_tensor=b)

# Solve
uh = Function(W)
solver = PETScLUSolver(’mumps’) # NOTE: we use direct solver for simplicity
solver.set_operator(A)
solver.solve(uh.vector(), b)

# Split displacement and multipliers. Plot
u, ls = uh.split(deepcopy=True)
plot(u, mode=’displacement’, title=’Neumann_displacement’,interactive=True)

file = File(’deformed_brain.pvd’)
file << u
```

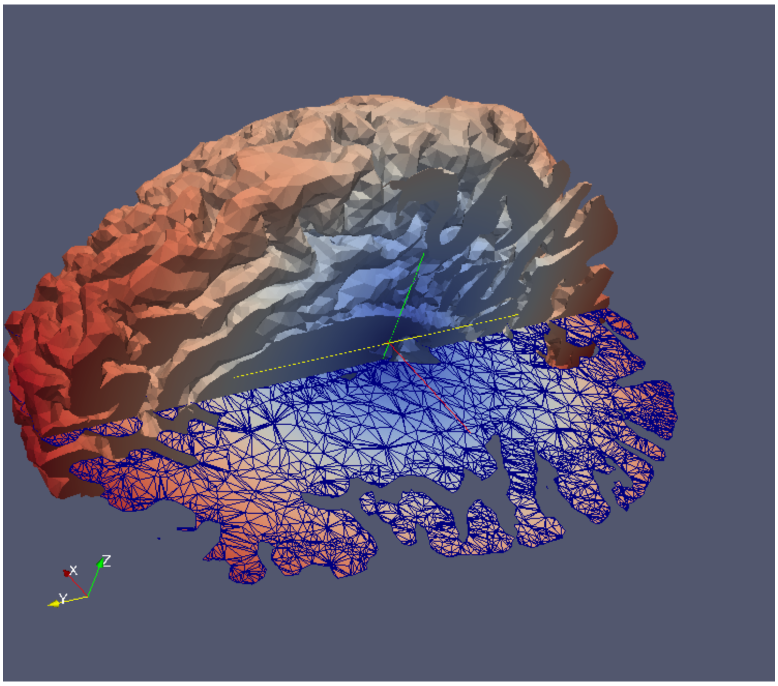


Figure 9.2: Deformation of the human brain during a cardiac cycle.

9.3 Locking

The locking phenomena has nothing to do with the problem related to the rigid motions studied in the previous section. Therefore, we consider locking in the simplest case possible where we have homogenous Dirichlet conditions. In this case the elasticity equation can be reduced to

$$\begin{aligned} -\mu\Delta u - (\mu+\lambda)\nabla\nabla\cdot u &= f, \quad \text{in}\Omega, \\ u &= 0, \quad \text{on}\partial\Omega. \end{aligned}$$

The weak formulation of the problem then becomes: Find $u \in H_0^1$ such that

$$a(u,v) = f(v), \quad \forall v \in H_0^1,$$

where

$$a(u,v) = \mu(\nabla u, \nabla v) + (\mu+\lambda)(\nabla\cdot u, \nabla\cdot v), \tag{9.17}$$

$$f(v) = (f,v). \tag{9.18}$$

The phenomen locking is a purely numerical artifact that arise when $\lambda \gg \mu$. Roughly speaking, approximating ∇ and $\nabla\cdot$ require different methods. While vertices based approximations work fine for ∇, edge based methods are more natural for $\nabla\cdot$ since this operator relates directly to the flux through the element edges.

For smooth functions, it can be verified directly that

$$\Delta = \nabla\cdot\nabla = \nabla\nabla\cdot + \nabla\times\nabla\times$$

where $\nabla\times$ is the curl operator. Hence in H^1_0 we have

$$(\nabla u, \nabla v) = (\nabla \cdot u, \nabla \cdot v) + (\nabla \times u, \nabla \times v).$$

Furthermore, it is well known (the Helmholz decomposition theorem) that any field in L^2 or H^1 can be decomposed into a the gradient of a scalar potential (irrotational, curl-free vector field) and the curl of scalar (a solenoidal, divergence-free vector field). That is,

$$u = \nabla\phi + \nabla \times \psi,$$

where ϕ and ψ are scalar fields that can be determined. Furthermore,

$$\nabla \cdot \nabla \times u = 0, \tag{9.19}$$
$$\nabla \times \nabla \cdot u = 0. \tag{9.20}$$

This means that

$$\nabla\nabla \cdot u = \begin{cases} \Delta u & \text{if } u \text{ is a gradient} \\ 0 & \text{if } u \text{ is a curl} \end{cases}$$

As the material becomes incompressible, when $\lambda \to \infty$ the gradient part is being locked and ϕ tends to zero. However, the curl represented by ψ remains unaffected. Vertex based finite elements such as Lagrange are poor at distinguising between gradients and curls and tend to lock the complete solution. Exercise 9.5 investigates this phenomena numerically.

To avoid locking it is common to introduce a the quantity solid pressure, $p = (\mu + \lambda)\nabla \cdot u$. Introducing this as a separate unknown into the system we obtain the equations:

$$-\mu\Delta u - \nabla p = f,$$
$$\nabla \cdot u - \frac{1}{\mu+\lambda} p = 0.$$

This system of equations is similar to the Stokes problem. Hence, we may formulation a weak problems as follows. Find $u \in H^1_0$ and $p \in L^2$ such that

$$a(u,v) + b(p,v) = f(v) \forall v \in H^1 \tag{9.21}$$
$$b(u,q) - c(p,q) = 0, \forall q \in \mathbb{R}. \tag{9.22}$$

Here,

$$a(u,v) = (\nabla u, \nabla v), \tag{9.23}$$
$$b(p,v) = (\nabla p, v), \tag{9.24}$$
$$c(p,q) = \frac{1}{\mu+\lambda}(p,q) \tag{9.25}$$
$$f(v) = (f,v). \tag{9.26}$$

The case when $\lambda \to \infty$ then represents the Stokes problem as $\frac{1}{\mu+\lambda} \to 0$. Hence, for this problem we know that stable discretizations can be obtained as long as we have Stokes-stable elements like for instance Taylor–Hood. We also remark that Stokes-stable elements handle any μ, λ because the $-c(p,q)$ is a negative term that only stabilize. In fact, this problem is identical to the proposed penalty method that was discussed for the Stokes problem.

Exercise 9.1. *Show that the inner product of a symmetric matrix A and matrix B equals the inner product of*

A and the symmetric part of B, i.e., that $A : B = A : B_S$, where $B_S = \frac{1}{2}(B + B^T)$.

Exercise 9.2. *Show that the term* $\operatorname{div} \epsilon(u)$ *in a weak setting may be written as* $(\epsilon(u), \epsilon(v))$*. Use the result of Exercise 9.1.*

Exercise 9.3. *Show that the Brezzi conditions (7.15-7.18) for the singular problem of homogenous Neumann conditions for the Poisson problem* (9.5)–(9.9)*. Hint: use the following version of Poincare's lemma:*

$$\|u - \bar{u}\|_0 \leq C\|\nabla u\|_0, \quad \forall u \in H^1.$$

Here, $\bar{u} = \frac{1}{|\Omega|}\int_\Omega u dx$*. As always, the inf-sup condition is challenging, but notice that*

$$sup_{u \in V_h} \frac{b(u,q)}{\|u\|_{V_h}} \geq \frac{b(\bar{u},q)}{\|\bar{u}\|_{V_h}}.$$

Exercise 9.4. *Show that three of Brezzi conditions (7.15-7.17) for problem linear elasticity problem with pure Neumann conditions* (9.12)-(9.13) *are valid. Hint: use Korn's lemma for the coersivity. As always, the inf-sup condition is challenging and we refer to* **?**.

Exercise 9.5. *We will consider the topic 'locking'. Consider the following equation on the domain* $\Omega = (0,1)^2$*:*

$$\begin{aligned} -\mu\Delta u - \lambda\nabla\nabla \cdot u &= f \text{ in } \Omega, & (9.27)\\ u &= u_{analytical} \text{ on } \partial\Omega & (9.28) \end{aligned}$$

where $u_{analytical} = (\frac{\partial\phi}{\partial y}, -\frac{\partial\phi}{\partial x})$ *and* $\phi = sin(\pi xy)$*. Here, by construction,* $\nabla \cdot u_{analytical} = 0$.
a) *Derive an expression for* f*. Check that the expression is independent of* λ*.*
b) *Compute the numerical error for* $\lambda = 1, 100, 10000$ *at* $h = 8, 16, 32, 64$ *for polynomial order both 1 and 2.*
c) *Compute the order of convergence for different* λ*. Is locking occuring?*

10 Finite element assembly

By Anders Logg, Kent–Andre Mardal

When using the FEM we get a linear system on the form

$$AU = b, \tag{10.1}$$

where

$$A_{ij} = a(\phi_j, \phi_i) \quad \text{and} \quad b_i = L(\phi_i).$$

Fundamental question: How to compute A? An obvious algorithm is:

```
for i = 1,...,N do
    for j = 1,...,N do
        $A_{ij} = a(\phi_i, \phi_j)$
    end for
end for
```

This algorithm is very inefficient! The reasons are:

1. A is sparse
2. Each element is visited multiple times
3. Basis functions have local support

10.1 Local to global mapping ι_T

We look at the local degrees of freedom and the global degrees of freedom. Figure (10.1) shows local and global degrees of freedoms. From the figure we can see that the local to global mapping is

$$\iota_T = (0, 1, 3, 11, 10, 5)$$
$$\iota_{T'} = (1, 2, 3, 7, 11, 6).$$

Note that the numbering is arbitary as long as neighboring T and T' agree. However some numbering schemes are more efficient then others, especially for parallel computing.

Note that

$$\phi_{\iota_T(i)}|_T = \phi_i^T \quad \Leftrightarrow \quad \phi_I|_T = \underbrace{\phi^T_{\iota_T^{-1}(I)}}_{\text{if it exists}} .$$

I and J are the counters for the global numbering.

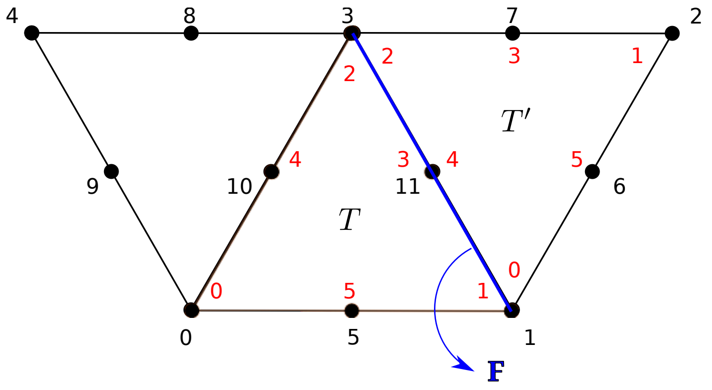


Figure 10.1: Red numbers indicate the local numbering, black number are the gobal numbering. Here P_2 elements where used, *dim* $\mathcal{P}_K = 6$.

10.2 *The element matrix A^T*

Assume that $a(u,v) = \sum_{T\in\mathcal{T}} a_T(u,v)$. Example,

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \sum_{T\in\mathcal{T}} \underbrace{\int_T \nabla u \cdot \nabla v \, \mathrm{d}x}_{a_T(u,v)}. \tag{10.2}$$

We then define

$$\boxed{A^T_{ij} = a_T(\phi^T_i, \phi^T_j).} \tag{10.3}$$

This is a small, typically dense matrix. We now note that

$$A_{IJ} = a(\phi_J, \phi_I) = \sum_{T\in\mathcal{T}} a_T(\phi_J, \phi_I) \tag{10.4}$$

$$= \sum_{T\in\mathcal{T}_{IJ}} a_T(\phi_J, \phi_I), \text{ all triangles where both } \phi_i \text{ and } \phi_j \text{ are nonzero,} \tag{10.5}$$

$$= \sum_{T\in\mathcal{T}_{IJ}} a_T(\phi^T_{\iota_T^{-1}(J)}, \phi^T_{\iota_T^{-1}(I)}) \tag{10.6}$$

$$= \boxed{\sum_{T\in\mathcal{T}_{IJ}} A^T_{\iota_T^{-1}(I)\iota_T^{-1}(J)}} \tag{10.7}$$

The algorithm becomes,

```
for $T \in \mathcal{T}$ do
    for i = 1,...,n do
        for j = 1,...,n do
            $A_{\iota_T(i)\iota_T(j)} += A^T_{ij}$
        end for
    end for
end for
```

or equivalent

```
for $T \in \mathcal{T}$ do
    Compute $A^T$
```

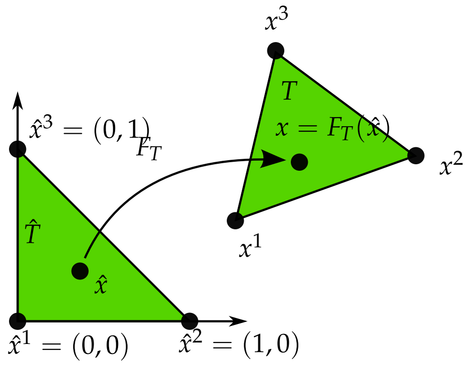


Figure 10.2: The (affine) map F_T from a reference cell $\hat{T}$ to a cell $T \in \mathcal{T}_h$.

```
    Compute $\iota_T$
    Insert $A^T$ to A according to $\iota_T$
end for
```

10.3 Affine mapping

To be able to compute A^T we will use affine mapping. This is a mapping between the reference element $\hat{T}$ to T, see figure 10.2.

$$x = F_T(\hat{x}) = B_T\hat{x} + c_T, \tag{10.8}$$

where B_T is a matrix and c_T is a vector. Let us look at a reference baisis function for P_1 elements,

$$\Phi_0 = 1 - \hat{x}_1 - \hat{x}_2 \tag{10.9}$$

$$\Phi_1 = \hat{x}_1 \tag{10.10}$$

$$\Phi_2 = \hat{x}_2. \tag{10.11}$$

Also recall that $\ell_i(\phi_j) = \delta_{ij}$. The mapping becomes,

$$F_T(\hat{x}) = \Phi_0(\hat{x})x_0 + \Phi_1(\hat{x})x_1 + \Phi_2(\hat{x})x_2 \tag{10.12}$$

10.4 How do we compute A^T?

We consider first the mass matrix

$$M_{ij}^T = \int_T \phi_j^T \phi_i^T \,\mathrm{d}x \tag{10.13}$$

$$= \int_{\hat{T}} \phi_j^T\,(F_T(\hat{x}))\,\phi_i^T\,(F_T(\hat{x}))\;det(F_T')\,d\hat{x} \tag{10.14}$$

$$= \int_{\hat{T}} \Phi_j \Phi_i \,det(F_T')\,d\hat{x} \tag{10.15}$$

$$= det(F_T') \int_{\hat{T}} \Phi_j \Phi_i \,d\hat{x}. \tag{10.16}$$

Now we consider the poisson equation (stiffness matrix)

$$
\begin{aligned}
A_{ij}^T &= \int_T \nabla\phi_j^T \nabla\phi_i^T \,\mathrm{d}x && (10.17)\\
&= \int_{\hat{T}} \frac{\partial}{\partial x_k}\phi_j^T \, \frac{\partial}{\partial x_k}\phi_j^T \,\mathrm{d}x && (10.18)\\
&= \int_{\hat{T}} \left(\frac{\partial \hat{x}_m}{\partial x_k}\frac{\partial}{\partial \hat{x}_m}\right)\Phi_J \, \left(\frac{\partial \hat{x}_n}{\partial x_k}\frac{\partial}{\partial \hat{x}_n}\right)\Phi_i \; det(F_T')\, d\hat{x} && (10.19)\\
&= \int_{\hat{T}} J_{mk}^{-1}\frac{\partial \Phi_j}{\partial \hat{x}_m} J_{nk}^{-1}\frac{\partial \Phi_i}{\partial \hat{x}_n} \; det(J)\, d\hat{x} && (10.20)\\
&= \int_{\hat{T}} \left(J^{-T}\nabla\Phi_j\right)\left(J^{-T}\nabla\Phi_i\right) \; det(J)\, d\hat{x}. && (10.21)
\end{aligned}
$$

11 The finite element method for time-dependent problems

By Anders Logg, Kent–Andre Mardal

Recall that there are two classes of problems:

$$\begin{aligned} \text{ODE:} \quad & \dot{u} = f(u,t) \\ \text{PDE:} \quad & \dot{u} + A(u) = f(x,t) \end{aligned} \tag{11.1}$$

11.1 The FEM for $\dot{u} = f$

Strong form

$$\begin{aligned} &\dot{u}(t) = f(u(t),t), \quad t \in (0,T] \\ &u(0) = 0 \end{aligned} \tag{11.2}$$

$$\begin{aligned} u &: [0,T] \to \mathbb{R}^N \\ f &: \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N \end{aligned} \tag{11.3}$$

Weak form

Find $u \in V$ such that

$$\int_0^T v \cdot \dot{u} \, \mathrm{d}t = \int_0^T v \cdot f \, \mathrm{d}t \quad \forall v \in \hat{V}. \tag{11.4}$$

Here, V is called the trial space and $\hat{V}$ is the test space.

Finite element method

Find $U \in V_k$ such that

$$\int_0^T v \cdot \dot{U} \, \mathrm{d}t = \int_0^T v \cdot f \, \mathrm{d}t \quad \forall v \in \hat{V}_k, \tag{11.5}$$

where V_k and $\hat{V}_k$ are the discrete trial space and discrete test space, respectively.

Solution algorithm

There are two different methods: continuous Galerkin, using CG_q elements, or discontinuous Galerkin, using DG_q elements. In this chapter we will go through continuous Galerkin.

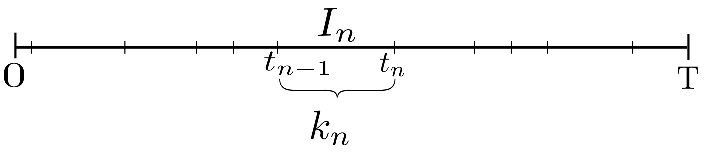


Figure 11.1

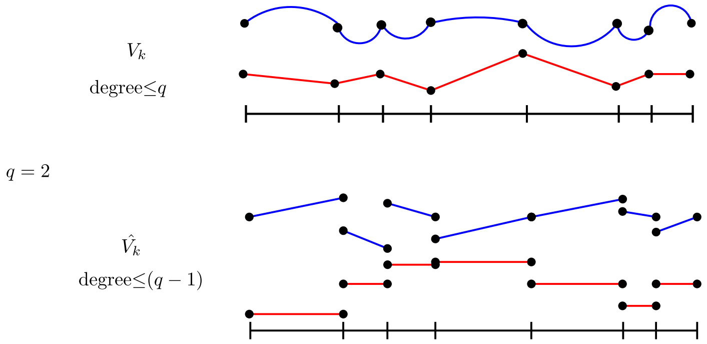


Figure 11.2

$$\begin{aligned} I_n &= (t_{n-1}, t_n) \\ k_n &= t_n - t_{n-1} = \text{time step} \end{aligned} \tag{11.6}$$

$$\begin{aligned} V_k &= \{\text{continuous piecewise polynomials of degree } \leqslant q\} \\ &= \{v \in [C(0,T)]^N \ : \ v|_{I_n} \in [\mathcal{P}_q(I_n)]^N \ \forall \, I_n\} \end{aligned} \tag{11.7}$$

$$\begin{aligned} \hat{V}_k &= \{\text{piecewise polynomials of degree } \leqslant q-1\} \\ &= \{v : [0,T] \to \mathbb{R}^N \ : \ v|_{I_n} \in [\mathcal{P}_{q-1}(I_n)]^N \ \forall \, I_n\} \end{aligned} \tag{11.8}$$

The continuous Galerkin method with $q = 1$

Find $U \in V_k$ such that

$$\int_0^T v \cdot \dot{U} \, \mathrm{d}t = \int_0^T v \cdot f \, \mathrm{d}t \quad \forall \, v \in \hat{V}_k, \tag{11.9}$$

where

$$V_k = \{v \in [C(0,T)]^N \ : \ v|_{I_n} \in [\mathcal{P}_1(I_n)]^N \ \forall \, I_n\} \tag{11.10}$$

and

$$V_k = \{v : [0,T] \to \mathbb{R}^N \ : \ v|_{I_n} \in [\mathcal{P}_0(I_n)]^N \ \forall \, I_n\}. \tag{11.11}$$

Take $v = 0$ on $[0,T] \backslash I_n$, then

$$\int_{I_n} v \cdot \dot{U} \, \mathrm{d}t = \int_{I_n} v \cdot f \, \mathrm{d}t \quad \forall \, v \in [\mathcal{P}_0(I_n)]^N. \tag{11.12}$$

Take $v = (0, \cdots, 0, 1, 0, \cdots, 0)$ (the value 1 is at position i), then

$$\int_{I_n} \dot{U}_i \,\mathrm{d}t = \int_{I_n} f_i \,\mathrm{d}t \quad i = 1, \cdots, N, \ \forall I_n \tag{11.13}$$

$$\Rightarrow \quad U_i(t_n) - U_i(t_{n-1}) = \int_{I_n} f_i \,\mathrm{d}t \quad i = 1, \cdots, N, \ \forall I_n \tag{11.14}$$

$$\Rightarrow \quad U_(t_n) - U_(t_{n-1}) = \int_{I_n} f_i \,\mathrm{d}t \quad \forall I_n \tag{11.15}$$

$$\Rightarrow \quad U(t_n) = U(t_{n-1}) + \int_{I_n} f_i \,\mathrm{d}t \quad \forall I_n \tag{11.16}$$

Let $U^n = U(t_n)$ and $U^{n-1} = U(t_{n-1})$, then

$$\boxed{U^n = U^{n-1} + \int_{I_n} f_i \,\mathrm{d}t} \quad \forall I_n, \tag{11.17}$$

here U^n is unknown and U^{n-1} is known. Note that this derivation holds for all q, but it is sufficient to determine U^n for $q = 1$ only! We approximate (11.17) by quadrature

$$\int_{t_{n-1}}^{t_n} f \,\mathrm{d}t \approx k_n f\left(\frac{U^{n-1} + U^n}{2}, \frac{t_{n-1} + t_n}{2}\right) \tag{11.18}$$

and obtain

$$\boxed{U^n = U^{n-1} + k_n f\left(\frac{U^{n-1} + U^n}{2}, \frac{t_{n-1} + t_n}{2}\right).} \tag{11.19}$$

Solving the discrete equations

In general (11.19) is a nonlinear system. We use one of the following two approaches to solve it:

i) Fixed-point iteration

ii) Newton's method

We will consider fixed-point iteration in this chapter. Take $U^{n,0} = U^{n-1}$, then the fixed-point iteration for (11.19) will look as follows

$$\boxed{U^{n,j} = U^{n-1} + k_n f\left(\frac{U^{n-1} + U^{n,j}}{2}, \frac{t_{n-1} + t_n}{2}\right).} \tag{11.20}$$

An important question is: When does (11.20) converge? Remember the contraction mapping theorem:

$$x^k = T(x^{k-1} \tag{11.21}$$

converges if

$$\|T'\| \leqslant M < 1. \tag{11.22}$$

Here:

$$T(x) = U^{n-1} + k_n f\left(\frac{U^{n-1}+x}{2}, \frac{t_{n-1}+t_n}{2}\right) \tag{11.23}$$

$$\Rightarrow \quad T'(x) = k_n J\left(\frac{U^{n-1}+x}{2}, \frac{t_{n-1}+t_n}{2}\right), \tag{11.24}$$

where J is defined

$$J_{ij} = \frac{\partial f_i}{\partial U_j}. \tag{11.25}$$

From equation (11.24) and the result from the contraction mapping theorem we see that equation (11.20) converges when k_n is small enough.

Stiff problems

If k_n is small enough to give an accurate solution, but not small enough for (11.20) to converge, we say that the problem is stiff.

Example 11.1 (Basic example)**.**

$$\dot{u} = \lambda u, \quad \lambda = 100 \tag{11.26}$$

Continuous Galerkin method with $q > 1$

Make an Anzats on each interval

$$U(t) = \sum_{j=0}^{q} U^{n,j} \lambda_j^q(t) \tag{11.27}$$

$$\Rightarrow \quad \int_{t_{n-1}}^{t_n} \sum_{j=0}^{q} U^{n,j} \dot{\lambda}_j^q(t) \cdot \lambda_i^{q-1}(t)\,\mathrm{d}t = \int_{t_{n-1}}^{t_n} \lambda_i^{q-1}(t) f_i\,\mathrm{d}t \tag{11.28}$$

This leads to a $q \times q$ linear system to be solved. It gives an *implicit Runge–Kutta method* for computing $U^{n,j}, j = 1, 2, \ldots, q$.

11.2 The FEM for $\dot{u} + A(u) = f$

Strong form

$$\begin{aligned} \dot{u} + A(u) &= f \quad \text{in } \Omega \times (0, T], \\ u(\cdot, 0) &= u_0 \quad \text{in } \Omega, \\ &+ \text{BC}. \end{aligned} \tag{11.29}$$

Weak form

Find $u \in V$ such that

$$\int_0^T \int_\Omega v\dot{u}\,\mathrm{d}x\,\mathrm{d}t + \int_0^T \int_\Omega vA(u)\,\mathrm{d}x\,\mathrm{d}t = \int_0^T \int_\Omega vf\,\mathrm{d}x\,\mathrm{d}t \quad \forall v \in \hat{V}. \tag{11.30}$$

Finite element method

Find $u_{hk} \in V_{hk}$ such that

$$\int_0^T \int_\Omega v\dot{u}_{hk}\,\mathrm{d}x\,\mathrm{d}t + \int_0^T \int_\Omega vA(u_{hk})\,\mathrm{d}x\,\mathrm{d}t = \int_0^T \int_\Omega vf\,\mathrm{d}x\,\mathrm{d}t \quad \forall\, v \in \hat{V}_{hk}. \tag{11.31}$$

Solution algorithm

$$\begin{aligned} V_{hk} &= \operatorname{span}\{v = v_h v_k \ :\ v_h \in V_h,\ v_k \in V_k\} \\ \hat{V}_{hk} &= \operatorname{span}\{v = v_h v_k \ :\ v_h \in \hat{V}_h,\ v_k \in \hat{V}_k\} \end{aligned} \tag{11.32}$$

$$\int_0^T \int_\Omega v_h v_k \dot{u}_{hk}\,\mathrm{d}x\,\mathrm{d}t + \int_0^T \int_\Omega v_h v_k A(u_{hk})\,\mathrm{d}x\,\mathrm{d}t = \int_0^T \int_\Omega vf\,\mathrm{d}x\,\mathrm{d}t \tag{11.33}$$

$$\int_0^T v_k \int_\Omega v_h \dot{u}_{hk}\,\mathrm{d}x\,\mathrm{d}t + \int_0^T v_k \int_\Omega v_h A(u_{hk})\,\mathrm{d}x\,\mathrm{d}t = \int_0^T \int_\Omega vf\,\mathrm{d}x\,\mathrm{d}t \tag{11.34}$$

Take

$$\begin{aligned} u_{hk}(x,t) &= \sum_{j=1}^{N} U_j(t)\phi_j(x) \\ v_h &= \phi_i, \quad i = 1, 2, \ldots, N \end{aligned} \tag{11.35}$$

and A linear. Then

$$\int_0^T v_k \sum_{j=1}^{N} \dot{U}_j \int_\Omega \phi_i\phi_j\,\mathrm{d}x\,\mathrm{d}t + \int_0^T v_k \sum_{j=1}^{N} U_j \int_\Omega \phi_i A(\phi_j)\,\mathrm{d}x\,\mathrm{d}t = \int_0^T \int_\Omega vf\,\mathrm{d}x\,\mathrm{d}t \tag{11.36}$$

We define the mass matrix M and the "stiffness matrix" A_k by

$$M_{ij} = \int_\Omega \phi_i\phi_j\,\mathrm{d}x, \tag{11.37}$$

$$A_{k,ij} = \int_\Omega \phi_i A(\phi_j)\,\mathrm{d}x. \tag{11.38}$$

Thus, we obtain

$$\boxed{\int_0^T v_k \cdot M\dot{U}\,\mathrm{d}t + \int_0^T v_k \cdot A_k(U)\,\mathrm{d}t = \int_0^T v_k b\,\mathrm{d}t,} \tag{11.39}$$

where

$$U = (U_1, U_2, \ldots, U_N)^T, \tag{11.40}$$

$$b = \int_\Omega v_h f\,\mathrm{d}x. \tag{11.41}$$

The overall solution algorithm is sketched in Figure 11.3.

Example 11.2 (Heat equation)**.**

$$\dot{u} - \Delta u = f \tag{11.42}$$

FEM in space gives

$$M\dot{U} - AU = b \tag{11.43}$$

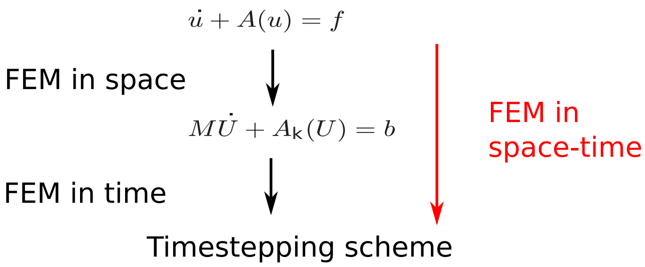


Figure 11.3

Continuous Galerkin with $q = 1$ leads to

$$\left(M + \frac{k_n A}{2}\right) U^n = \left(M + \frac{k_n A}{2}\right) U^{n-1} + k_n b_n. \tag{11.44}$$