Lectures on the Finite Element Method
## Contents

1 A short look at functional analysis and Sobolev spaces .......................... 1
   1.1 Functional analysis ................................................. 1
   1.2 Sobolev spaces .................................................... 6

2 Crash course in Sobolev Spaces ................................................. 9
   2.1 Introduction .......................................................... 9
   2.2 Sobolev spaces and norms ........................................... 9
   2.3 Examples of Functions in Different Spaces ............................... 10
   2.4 Eigenvalues and Finite Element Methods ............................... 11
   2.5 Negative Norms ..................................................... 12

3 The finite element method .................................................... 17
   3.1 A simple model problem .............................................. 17
   3.2 Solving Poisson’s equation using the finite element method .......... 18
   3.3 Solving the Poisson equation with FEM using abstract formalism ..... 21
   3.4 Galerkin orthogonality .............................................. 22

4 Finite element function spaces ............................................... 25
   4.1 The finite element definition ....................................... 25
   4.2 Common elements ................................................... 31

5 Discretization of a convection-diffusion problem ............................... 35
   5.1 Introduction .......................................................... 35
   5.2 Streamline diffusion/Petrov-Galerkin methods ......................... 38
   5.3 Well posedness of the continuous problem ............................ 41
   5.4 Error estimates ..................................................... 43
   5.5 Exercise .............................................................. 44

6 Stokes problem ........................................................................ 45
   6.1 Introduction ............................................................ 45
   6.2 Examples of elements .................................................. 51
   6.3 Stabilization techniques to circumvent the Babuska-Brezzi condition 53
   6.4 Exercises .............................................................. 54

7 Finite element error estimate .................................................. 55
   7.1 Ingredients ............................................................. 55
   7.2 Error estimates ......................................................... 57
   7.3 Adaptivity .............................................................. 61
## Contents

8 **Iterative methods** 65
8.1 A simple iterative method: Richardson iteration .......................... 65
8.2 Generalizing by a preconditioning ........................................... 67
8.3 Multigrid ................................................................. 69
8.4 Spectral equivalence and order optimal algorithms ....................... 71
8.5 Krylov methods and preconditioning ....................................... 72
8.6 Examples ............................................................... 72
8.7 Exercise ............................................................... 76

9 **Finite element assembly** 79
9.1 Local to global mapping \( i_T \) ........................................... 79
9.2 The element matrix \( A^T \) ................................................. 80
9.3 Affine mapping ........................................................... 81
9.4 How do we compute \( A^T \)? ............................................. 81

10 **The incompressible Navier–Stokes equations** 83
10.1 The equations ............................................................ 83
10.2 Operator splitting ......................................................... 84
10.3 Algebraic splitting ......................................................... 86

11 **The finite element method for time-dependent problems** 91
11.1 The FEM for \( \dot{u} = f \) ................................................. 91
11.2 The FEM for \( \dot{u} + A(u) = f \) ........................................ 94
Preface

Insert text here.
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

\[
-\nabla \cdot (\kappa \nabla u) = f \quad \text{in} \quad \Omega,
\]

\[
u = u_0 \quad \text{on} \quad \Gamma_D \subset \partial \Omega,
\]

\[
-\kappa \nabla u \cdot n = g \quad \text{on} \quad \Gamma_N \subset \partial \Omega,
\]

where \( u = u(x) \) is some unknown field, \( \kappa : \Omega \rightarrow \mathbb{R}^{(d \times d)} \) is some given coefficient matrix and \( f = f(x) \) is a given source function. The boundary \( \partial \Omega \) of \( \Omega \) is a union of two subboundaries, \( \partial \Omega = \Gamma_D \cup \Gamma_N \), where \( \Gamma_D \) is the Dirichlet boundary and \( \Gamma_N \) is the Neumann boundary. The Dirichlet boundary condition, \( u = u_0 \), specifies a prescribed value for the unknown \( u \) on \( \Gamma_D \). The Neumann boundary condition, \( -\kappa \nabla u \cdot n = g \), specifies a prescribed value for the (negative) normal derivative of \( u \) on \( \Gamma_N \). 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 (7.55) 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

\[
\frac{dE}{dt} = \int_{\partial \omega} q \cdot n \, ds - \int_{\omega} f \, dx = 0,
\]

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:

\[
q = -\kappa \nabla u.
\]

This gives us

\[
\int_{\partial \omega} -\kappa \nabla u \cdot n \, ds = \int_{\omega} f \, dx.
\]
Chapter 1. The finite element method

By the Gauss theorem,
\[ \int_{\partial \omega} - \kappa \nabla u \cdot n \, ds = \int_{\omega} \nabla \cdot (-\kappa \nabla u) \, dx \quad (1.5) \]
\[ \Rightarrow - \int_{\omega} \nabla \cdot (\kappa \nabla u) \, dx = \int_{\omega} f \, dx. \quad (1.6) \]

Equation (3.6) holds for all test volumes \( \omega \subset \Omega \). Thus, if \( u, \kappa \) and \( f \) are regular enough, we obtain
\[ \int_{\omega} (-\nabla \cdot (\kappa \nabla u) - f) \, dx = 0 \quad \forall \, \omega \subset \Omega \quad (1.7) \]
\[ \Rightarrow -\nabla \cdot (\kappa \nabla u) = f \quad \text{in} \, \Omega. \quad (1.8) \]

The Boundary conditions of this problem becomes
\[ u = u_0 \quad \text{on} \, \Gamma_D \]
\[ -\kappa \nabla u \cdot n = g \quad \text{on} \, \Gamma_N \quad (1.9) \]
(recall that \( q = -\kappa \nabla u \)). This is illustrated in Figure 3.2. If we choose the special case where \( \kappa = 1 \), we obtain the more standard Poisson equation
\[ -\Delta u = f \quad \text{in} \, \Omega. \quad (1.10) \]

Then, the boundary conditions becomes
\[ u = u_0 \quad \text{on} \, \Gamma_D \quad (1.11) \]
\[ -\frac{\partial u}{\partial n} = g \quad \text{on} \, \Gamma_N. \quad (1.12) \]

1.2 Solving Poisson’s equation using the finite element method

Solving a PDE using the finite element method is done in four steps:
1. Strong form,
Chapter 1. The finite element method

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

\[-\nabla \cdot (\kappa \nabla u) = f \quad \text{in } \Omega, \]
\[u = u_0 \quad \text{on } \Gamma_D \subset \partial \Omega, \]
\[-\kappa \nabla u \cdot n = g \quad \text{on } \Gamma_N \subset \partial \Omega. \quad (1.13)\]

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:

\[\int_\Omega -\nabla \cdot (\kappa \nabla u) v \, dx = \int_\Omega f v \, dx \quad \forall v \in \hat{V} \quad (1.14)\]
\[\int_\Omega \kappa \nabla u \cdot \nabla v \, dx - \int_{\partial \Omega} \kappa \frac{\partial u}{\partial n} v \, ds = \int_\Omega f v \, dx \quad \forall v \in \hat{V}. \quad (1.15)\]

Here we have done integration by parts using that

\[\int_\Omega (\nabla q) w \, dx = \int_{\partial \Omega} (q \cdot n) w \, ds - \int_\Omega q (\nabla w) \, dx, \quad (1.16)\]

which in our case becomes

\[\int_\Omega -\nabla \cdot (\kappa \nabla u) v \, dx = \int_{\partial \Omega} -\kappa \frac{\partial u}{\partial n} v \, ds + \int_\Omega \kappa \nabla u \cdot \nabla v \, dx. \quad (1.17)\]
Letting \( v = 0 \) on the Dirichlet boundary, \( \Gamma_D \), the integral over the boundary becomes
\[
\int_{\partial \Omega} -\kappa \frac{\partial u}{\partial n} v \, ds = \int_{\Gamma_N} -\kappa \frac{\partial u}{\partial n} v \, ds = \int_{\Gamma_N} g v \, ds. \tag{1.18}
\]
We have arrived at the following problem: find \( u \in V \) such that
\[
\int_{\Omega} \kappa \nabla u \cdot \nabla v \, dx = \int_{\Omega} f v \, dx - \int_{\Gamma_N} g v \, ds \quad \forall \, v \in \hat{V}. \tag{1.19}
\]
The test space \( \hat{V} \) is defined by
\[
\hat{V} = H^1_{0,\Gamma_D}(\Omega) = \{ v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D \} \tag{1.20}
\]
and the trial space \( V \), containing the unknown function \( u \), is defined similar to \( \hat{V} \) but with a shifted Dirichlet condition:
\[
V = H^1_{u_0,\Gamma_D}(\Omega) = \{ v \in H^1(\Omega) : v = u_0 \text{ on } \Gamma_D \}. \tag{1.21}
\]

### 1.2.3 The finite element method for Poisson’s equation

We discretize the variational problem (3.19) 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
\[
\int_{\Omega} \kappa \nabla u_h \cdot \nabla v \, dx = \int_{\Omega} f v \, dx - \int_{\Gamma_N} g v \, ds \quad \forall \, v \in \hat{V}_h \subset \hat{V}, \tag{1.22}
\]
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 (3.22)? We introduce a basis for \( V \) and \( V_h \), and make an Ansatz:
\[
u_h(x) = \sum_{j=1}^{N} U_j \phi_j(x), \tag{1.23}\]
where
\[
\phi_j : \Omega \to \mathbb{R}, \quad j = 1, \ldots, N, \tag{1.24}
\]
is basis for \( V_h \). Inserting this into equation (3.22) and letting \( v = \hat{\phi}_i \), \( i = 1, \ldots, N \), we obtain
\[
\int_{\Omega} \kappa \nabla \left( \sum_{j=1}^{N} U_j \phi_j \right) \cdot \nabla \hat{\phi}_i \, dx = \int_{\Omega} f \hat{\phi}_i \, dx - \int_{\Gamma_N} g \hat{\phi}_i \, ds, \quad i = 1, 2, \ldots, N, \tag{1.25}
\]
\[
\sum_{j=1}^{N} U_j \int_{\Omega} \kappa \nabla \phi_j \cdot \nabla \hat{\phi}_i \, dx = \int_{\Omega} f \hat{\phi}_i \, dx - \int_{\Gamma_N} g \hat{\phi}_i \, ds, \quad i = 1, 2, \ldots, N.
\]
Chapter 1. The finite element method

We recognize this as a system of linear equations:

\[
\sum_{j=1}^{N} A_{ij} U_j = b_i, \quad i = 1, 2, \ldots, N, \\
AU = b,
\]

where

\[
A_{ij} = \int_{\Omega} \kappa \nabla \phi_j \cdot \nabla \phi_i \, dx, \\
b_i = \int_{\Omega} f \phi_i \, dx - \int_{\Gamma_N} g \phi_i \, ds.
\]

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

\[
Au = f, \\
(\text{+ BCs }),
\]

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

\[
\langle Au, v \rangle = \langle f, v \rangle
\]

Define

\[
a(u, v) = \langle Au, v \rangle, \\
L(v) = \langle f, v \rangle,
\]

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

\[
a : V \times \hat{V} \rightarrow \mathbb{R}, \\
L : \hat{V} \rightarrow \mathbb{R}.
\]

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

\[
a(u, v) = L(v) \quad \forall v \in \hat{V}.
\]

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

\[
a(u_h, v) = L(v) \quad \forall v \in \hat{V}_h.
\]
1.3.4 Solution algorithm

Let \( \{\phi_i\}_{i=1}^N \) be a basis for \( V_h \). Make an Anzats
\[
u_h(x) = \sum_{j=1}^N U_j \phi_j(x).
\]
(1.34)

Inserting this to the variational form, it follows
\[
a \left( \sum_{j=1}^N U_j \phi_j, \hat{\phi}_i \right) = L(\hat{\phi}_i), \quad i = 1, 2, \ldots, N,
\]
\[
\sum_{j=1}^N U_j a(\phi_j, \hat{\phi}_i) = L(\hat{\phi}_i), \quad i = 1, 2, \ldots, N.
\]
(1.35)

As before, \( u_h \) may be computed by solving a linear system
\[
\sum_{j=1}^N A_{ij} U_j = b_i, \quad i = 1, 2, \ldots, N,
\]
\[
AU = b,
\]
(1.36)

where
\[
A_{ij} = a(\phi_j, \hat{\phi}_i),
\]
\[
b_i = L(\hat{\phi}_i).
\]
(1.37)

1.4 Galerkin orthogonality

We will now show Galerkin orthogonality. First, we know that
\[
a(u, v) = L(v) \quad \forall v \in V,
\]
\[
a(u_h, v) = L(v) \quad \forall v \in V_h \subset V.
\]
(1.38)

Using these results and the linearity of the bilinear form, we get
\[
a(u - u_h, v) = a(u, v) - a(u_h, v) = L(v) - L(v) = 0 \quad \forall v \in V_h,
\]
or written symbolically
\[
u - u_h \perp_a V_h.
\]
(1.39)

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.

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. $V = \mathbb{R}^3$
Chapter 2. A short look at functional analysis and Sobolev spaces

3. \( V = \mathbb{R}^N, [x_1, \ldots, x_N] + [y_1, \ldots, y_N] = [x_1 + y_1, \ldots, x_N + y_N] \) and \( a[x_1, \ldots, x_N] = [ax_1, \ldots, ax_N] \)

4. \( V = \{ v : [0, 1] \to \mathbb{R} \mid v \text{ is continuous} \} \)

5. \( V = \{ v : [0, 1] \to \mathbb{R} \mid v(x) \leq 1, \ \forall x \in [0, 1] \}, \text{ NOT a vector space!} \)

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:

1. \( \langle v, w \rangle = \langle w, v \rangle \ \forall v, w \in V \) (conjugate symmetry)
2. \( \langle \alpha v, w \rangle = \alpha \langle v, w \rangle \ \forall v \in V \text{ and } \forall \alpha \in F \)
3. \( \langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle \ \forall u, v, w \in V \) (linearity)
4. \( \langle v, v \rangle \geq 0 \ \text{with} \ \langle v, v \rangle = 0 \iff v = 0 \) (positive definite)

Examples:

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

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

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. \( V = \mathbb{R}^3, \quad v = (1, 2, 3), \quad w = (3, 0, -1) \)
2. \( V = \mathcal{P}^2([-1, 1]), \quad u = 1, \quad v = x, \quad w = \frac{1}{2} (3x^2 - 1) \) (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. \( \| a v \| = |a| \| v \|, \ \forall v \in V \text{ and } \forall a \in F \) (Positive homogeneity)
2. \( \| u + v \| \leq \| u \| + \| v \|, \ \forall u, v \in V \) (triangle inequality)
3. \( \| v \| = 0 \ \Rightarrow \ v = 0 \) (point separation)
Examples:

1. $V = \mathbb{R}^N$, $\|v\|_p = \left(\sum_{i=1}^{N} v_i^p\right)^{1/p}$, $1 \leq p < \infty$
2. $V = \mathbb{R}^N$, $\|v\|_\infty = \max_{1 \leq i \leq N} |v_i|
3. $V = C^\infty(\Omega)$, $\|v\|_p = \left(\int_{\Omega} |v|^p \, dx\right)^{1/p}$, $1 \leq p < \infty$
4. $V = C^\infty(\Omega)$, $\|v\|_\infty = \sup_{x \in \Omega} |v(x)|$
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. 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 $\|v_m - v_n\| < \epsilon$ for all $m, n > N$.

Examples:

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

This sequence is Cauchy in the $L^1$-norm, $\|v\|_1 = \int_{-1}^{1} |v(x)| \, dx$, 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 1.1 and 1.2 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.

Definition 2.9. (Continuous) Dual space

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

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

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

---

1Can be generalized to metric spaces, $d(v_m, v_n) < \epsilon$
Chapter 2. A short look at functional analysis and Sobolev spaces

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

Figure 2.2: Cauchy sequence: \( \frac{\sin n}{n} \) for \( n = 1, \ldots, 100 \).
Chapter 2. A short look at functional analysis and Sobolev spaces

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

$$|⟨v, w⟩| \leq ∥v∥ · ∥w∥ \quad ∀ v, w ∈ V.$$

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,

$$∃ M < 1 : ∥T(v) − T(w)∥ ≤ M∥v − w∥ \quad ∀ v, w ∈ V.$$

Then $∃! \bar{v} ∈ V$, such that $T\bar{v} = \bar{v}$.

Examples:

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

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

$$l(v) = ⟨\hat{l}, v⟩ \quad ∀ v ∈ H$$

Theorem 2.4. Integration by parts in $n$-dimensions
Let $\Omega ∈ \mathbb{R}^n$ and let $v$ and $w$ be functions in $H^1(\Omega)$. Then,

$$\int_Ω \frac{∂v}{∂x_i} w \, dx = − \int_Ω \frac{∂w}{∂x_i} v \, dx + \int_{∂Ω} v \, w \, n_i \, dS,$$

where $n_i$ it the $i$'th normal component.
2.2 Sobolev spaces

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

**Definition 2.10. The $L^2(\Omega)$ space**

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

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

**Examples:**

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

**Theorem 2.5.** $L^2$ with $\langle v, w \rangle = \int_\Omega vw \, dx$ 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

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

**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

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

where

$$\partial^\alpha \phi = \frac{\partial^{|\alpha|}}{\partial x_1 \partial x_2 \cdots \partial x_n}.$$

**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 $\partial^\alpha$ for $|\alpha| \leq m$. The corresponding norm is

$$\|v\|_{H^k} = \sqrt{\sum_{|\alpha| \leq k} \int_\Omega |\partial^\alpha v|^2 \, dx} \equiv \sqrt{\sum_{|\alpha| \leq k} \|\partial^\alpha v\|^2_{L^2(\Omega)}}$$

and seminorm

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

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

$$\langle v, w \rangle = \int_\Omega vw \, dx + \int_\Omega \nabla v \cdot \nabla w \, dx$$
Theorem 2.7. Poincaré inequality

Let \( v \in H^1_0(\Omega) \). Then,

\[
\|v\|_{L^2(\Omega)} \leq C|v|_{H^1(\Omega)},
\]

where \( C \) depends only on \( \Omega \).
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 and norms

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 $\Omega$, which for the moment will be assumed to be the unit interval $(0, 1)$. Then

$$\|u\|_p = \left( \int_0^1 |u|^p dx \right)^{1/p}.\tag{3.1}$$

$L^p(\Omega)$ 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

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

Then the Sobolev space $W^p_k(\Omega)$ consists of all functions with $\|u\|_{p,k} < \infty$. $W^p_k$ 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

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

The case $p = 2$ is special in the sense that (2.1) defines an inner product. The Banach space then forms a Hilbert space and these named with $H$ in Hilbert’s honor. That is $H^k(\Omega) = W^{2k}(\Omega)$.

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.
Chapter 3. Crash course in Sobolev Spaces

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 $k = 1$, $k = 10$, and $k = 100$. 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)
```

<table>
<thead>
<tr>
<th>$k$ \ norm</th>
<th>$L^2$</th>
<th>$L^7$</th>
<th>$H^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.71</td>
<td>0.84</td>
<td>2.3</td>
</tr>
<tr>
<td>10</td>
<td>0.71</td>
<td>0.84</td>
<td>22</td>
</tr>
<tr>
<td>100</td>
<td>0.71</td>
<td>0.84</td>
<td>222</td>
</tr>
</tbody>
</table>

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 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 Sobolev space.
Chapter 3. Crash course in Sobolev Spaces

19

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.

On the other hand, the step function in upper picture in Figure 2.2 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 2.2 is a typical first order finite element function. The function is in both $L^2(0,1)$ and $H^1(0,1)$ (see Exercise 2.3). 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.

3.4 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
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

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

\(^2\)We use the `assemble_system` function to enforce the Dirichlet condition in symmetric fashion.
mesh-dependent scaling. To estimate the continuous eigenvalues we instead compute the eigenvalues of the generalized eigenvalue problem,

$$Ax = \lambda Mx,$$

(3.2)

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

Python code

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

Figure 2.2 shows the eigenvalues of $-\Delta$, $A$, and (2.2) based on the following code:

Python code

```python
from dolfin import *
import pytave
import numpy
def boundary(x, on_boundary): return on_boundary
for N in [10, 100, 1000]:
    mesh = UnitIntervalMesh(N)
    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)
    k = numpy.arange(1, N, 1)
    eig = pi**2*k**2
    l1, = pytave.feval(1, "eig", A.array())
    l2, = pytave.feval(1, "eig", A.array(), M.array())
    l1 = l1.flatten(); l1.sort()
    l2 = l2.flatten(); l2.sort()
import pylab
pylab.loglog(l1)
pylab.loglog(l2)
pylab.loglog(eig)
pylab.legend(["eig(A)", "eig(A,M)", "cont. eig"])
pylab.show()
```

From Figure 2.3 we see that the eigenvalues of (2.2) and $-\Delta$ are close, while the eigenvalues of $A$ is differently scaled. We remark that we use `pytave`, the Python interface of Octave because `numpy` is not trustworthy for eigenvalue computations on "large" linear systems. A description of `pytave` is found in ?.

### 3.5 Negative Norms

As will be discussed more thoroughly later, $-\Delta$ is a symmetric positive operator and can be thought of as an infinite dimensional matrix that is symmetric and positive. Let us therefore consider a matrix $A$ that is symmetric and positive. $A$ has positive and real eigenvalues and defines an inner product
which may be represented in terms of eigenvalues and eigenvectors:

\[(x, x)_A = (Ax, x) = (\sum_i \lambda_i e_i \cdot x, e_i \cdot x),\]

where \(\lambda_i\) and \(e_i\) are the eigenvalues and eigenvectors of \(A\), respectively. The \(A^q\) inner product is defined as

\[(x, x)_{A,q} = (\sum_i \lambda_i^q e_i x, e_i x).\] (3.3)

Example 2.2 uses the above definition to compute various norms based on the discrete \(-\Delta\), using \(q = 1, 0, -1\).

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

Let as before \(\Omega = (0, 1)\) and \(u_k = \sin(\pi kx)\). Table 2.1 shows the \(H^1\), \(L^2\), and \(H^{-1}\) norms as computed with (2.3) with \(q = 1, 0, -1\), respectively. Comparing the computed norms with the norms \(L^2\) and \(H^1\) norms computed in Example 2.1, we see that the above definition (2.3) 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.

<table>
<thead>
<tr>
<th>(k)</th>
<th>(H^1, q = 1)</th>
<th>(L^2, q = 0)</th>
<th>(H^{-1}, q = -1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.2</td>
<td>0.71</td>
<td>0.22</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>0.71</td>
<td>0.022</td>
</tr>
<tr>
<td>100</td>
<td>222</td>
<td>0.71</td>
<td>0.0022</td>
</tr>
</tbody>
</table>

Table 3.2: The \(L^2\), \(L^2\), and \(H^1\) norms of \(\sin(\pi kx)\) for \(k = 1, 10, 100\).
Chapter 3. Crash course in Sobolev Spaces

import pytave
from numpy import matrix, diagflat, sqrt, linalg
# e, l = eigenvectors, eigenvalues
e, l = pytave.feval(2, "eig", A.array(), M.array())
e = matrix(v)
l = matrix(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())
ex = e.T*x.T
H1_norm = pi*k*sqrt(2)/2
print "H1 norm of sin(\%d pi x) \% (k, H1_norm)"
H1_norm = sqrt(assemble(inner(grad(u), grad(u))*dx))
print "H1 norm of sin(\%d pi x) \% (k, H1_norm)"
H1_norm = sqrt(ex.T*l*ex) / len(ex)
print "H1 norm of sin(\%d pi x) \% (k, H1_norm)"
L2_norm = sqrt(assemble(u**2*dx))
print "L2 norm of sin(\%d pi x) \% (k, L2_norm)"
L2_norm = sqrt(ex.T*l**0*ex) / len(ex)
print "L2 norm of sin(\%d pi x) \% (k, L2_norm)"
Hm1_norm = sqrt(ex.T*l**-1*ex) / len(ex)
print "H^-1 norm of sin(\%d pi x) \% (k, Hm1_norm)"

Remark 3.5.1. Norms for |q| > 1.
The norm (2.3) is well defined for any |q| > 1, but will not correspond to the corresponding Sobolev spaces.

Remark 3.5.2. The standard definition of a dual norm
Let (·, ·)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
\[
\|f\|_{-1} = \sup_{v \in H^1} \frac{(f, v)}{(v, v)_1}.
\]

Exercise 3.1. Compute the H1, L2, and H−1 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 H1 and L2 norms of sin(kπx) on the unit interval analytically and compare with the values presented in Table 2.2.

Exercise 3.3. Compute the H1 and L2 norms of the hat function in Picture 2.2.

Exercise 3.4. Compute the H1 and L2 norms of the hat function in Picture 2.2.
Exercise 3.5. Consider the following finite element function $u$ defined as

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

That is, it corresponds to the hat function in Figure 2.2, where $u(0.5) = 1$ and the hat function is zero everywhere 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 3.6. 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 \( \iff \) 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 (7.2) from (7.1), we get the Galerkin orthogonality:

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

4.1.2 Interpolation estimates

First, let us note that

\[
 \| u - u_h \| \geq \inf_{v \in V_h} \| u - v \|, \tag{4.4}
\]

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

$$ u(x) = u \left( \frac{x_i - x_{i-1}}{2} + \int_{x_{i-1}}^{x_i} u'(y) \, dy \right) $$

which leads to

$$ |u - \pi_h u| = \left| \int_{x_{i-1}}^{x_i} u'(y) \, dy \right|. $$

Let us consider the $L^2$-norm. Then,

$$ \|u - \pi_h u\|_{L^2}^2 = \int_a^b (u - \pi_h u)^2 \, dx = \sum_i \int_{x_{i-1}}^{x_i} (u - \pi_h u)^2 \, dx $$

$$ = \sum_i \int_{x_{i-1}}^{x_i} \left( \int_{x_{i-1}}^{x_i} u'(y) \, dy \right)^2 \, dx $$

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

$$ \|u - \pi_h u\|_{L^2}^2 = \sum_i \int_{x_{i-1}}^{x_i} \left( \int_{x_{i-1}}^{x_i} \frac{1}{2} \cdot u'(y) \, dy \right)^2 \, dx $$

$$ \leq \sum_i \int_{x_{i-1}}^{x_i} \left( \left( \int_{x_{i-1}}^{x_i} 1^2 \, dy \right)^{1/2} \cdot \left( \int_{x_{i-1}}^{x_i} (u'(y))^2 \, dy \right)^{1/2} \right)^2 \, dx $$

$$ = \sum_i \int_{x_{i-1}}^{x_i} \left| x - \frac{x_{i-1} + x_i}{2} \right| \cdot \int_{x_{i-1}}^{x_i} (u'(y))^2 \, dy \, dx $$

$$ \leq \sum_i h_i^2 \int_{x_{i-1}}^{x_i} \int_{x_{i-1}}^{x_i} (u'(y))^2 \, dy \, dx $$

$$ = \sum_i h_i^2 \int_{x_{i-1}}^{x_i} (u'(y))^2 \, dy $$

$$ \leq \frac{1}{2} \int_a^b (h u'(y))^2 \, dy = \frac{1}{2} \|hu'\|_{L^2}^2, $$

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

$$ \|u - \pi_h u\|_{L^2} \leq \frac{1}{\sqrt{2}} \|hu'\|_{L^2}. \quad (4.5) $$

In general, one can prove that

$$ \|D^p(u - \pi_h u)\|_{L^2} \leq C(p, q) \|h^{q+1-p} D^{q+1} u\|_{L^2}, \quad (4.6) $$
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

\[
a(v, v) \geq \alpha \|v\|^2_V \quad \forall v \in V.
\]

\( \| \cdot \|_V \) 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

\[
\|e\|_E^2 = a(e, e) = a(e, u - u_h)
\]

\[
= a(e, u - v + v - u_h)
\]

\[
= a(e, u - v) + a(e, v - u_h)
\]

\[
\in V_h
\]

\[
= a(e, u - v) + 0 \quad \text{(from Galerkin Orthogonality)}
\]

\[
\leq \sqrt{a(e, e)} \sqrt{a(u - v, u - v)}
\]

\[
= \|e\|_E \|u - v\|_E.
\]

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

\[
\|u - u_h\|_E \leq \|u - v\|_E \quad \forall v \in V_h.
\]

Thus, the FEM solution is the optimal solution in the energy Norm! We combine this with the interpolation estimate (7.5), by setting \( v = \pi_h u \):

\[
\|u - u_h\|_E \leq \|u - \pi_h u\|_E \leq C(p, q)\|h^{q+1-p}D^{q+1}u\|.
\]

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

\[
\|v\|_E = \sqrt{\int \Omega |\nabla v|^2 \, dx}.
\]
Chapter 4. Finite element error estimate

The \textit{a priori} estimate (7.15) becomes
\[
\| e \|_E \leq C \| h D^2 u \|. \quad (4.16)
\]

A priori error estimate in the $V$–norm that does not assume symmetry. From coercivity we get
\[
\| e \|_V^2 \leq \frac{1}{\alpha} a(e,e) \quad (4.17)
\]
\[
= \frac{1}{\alpha} a(e, u - v + v - u_h) \quad (4.18)
\]
\[
= \frac{1}{\alpha} a(e, u - v) \quad \text{(from Galerkin Orthogonality)} \quad (4.19)
\]
\[
\leq \frac{C}{\alpha} \| e \|_V \| u - v \|_V. \quad (4.20)
\]

Here we assumed boundedness of $a$. By dividing both sides by $\| e \|_V$, we get an inequality known as \textit{Cea’s lemma}.
\[
\| e \|_V \leq \frac{C}{\alpha} \| u - v \|_V \quad \forall \ v \in V_h \quad (4.21)
\]

As before, we can use an interpolation estimate to obtain
\[
\| e \|_V \leq \frac{C \cdot C(q,p)}{\alpha} \| h^{q+1-p} D^{q+1} u \|. \quad (4.22)
\]

\[
4.2.2 \quad \textit{A posteriori} error estimate for the Poisson problem in the energy norm
\]

We will now derive an \textit{a posteriori} error estimate for the Poisson problem. To do this we need the following interpolation estimates:
\[
\| e - \pi_h e \|_T \leq C h_T \| \nabla e \|_T, \quad (4.23)
\]
\[
\| e - \pi_h e \|_{\partial T} \leq C \sqrt{h_T} \| \nabla e \|_{\omega_T}, \quad (4.24)
\]

where $\omega_T$ 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,
\[
ab \leq \delta a^2 + \frac{b^2}{4\delta}, \quad a, b, \delta > 0. \quad (4.25)
\]

Recall that the energy–norm for the Poisson problem is
\[
\| v \|_E = \sqrt{\int_{\Omega} |\nabla v|^2 \, dx}.
\]
Let us begin the derivation,

\[ \|e\|_E^2 = a(e,e) \]

\[ = a(e,e) - a(e, \pi_h e) \]

\[ = a(e, e - \pi_h e) \]

\[ = \int_\Omega \nabla e \cdot \nabla (e - \pi_h e) \, dx \]

\[ = \sum_{T \in T_h} \int_T \nabla e \cdot \nabla (e - \pi_h e) \, dx \]

\[ = \sum_{T \in T_h} \int_T -\Delta e (e - \pi_h e) \, dx + \int_{\partial T} \partial_n e (e - \pi_h e) \, dS \]

\[ = \sum_{T \in T_h} \int_T (-\Delta u + \Delta u_h) (e - \pi_h e) \, dx + \int_{\partial T} \partial_n e (e - \pi_h e) \, dS \]

\[ = \sum_{T \in T_h} \int_T (f + \Delta u_h) (e - \pi_h e) \, dx + \sum_S \int_{\partial S} \left[ \partial_n e + \partial_n u_h \right] (e - \pi_h e) \, dS \]

\[ = \sum_{T \in T_h} \int_T R (e - \pi_h e) \, dx - \frac{1}{2} \sum_{\partial T} \left[ \partial_n u_h \right] (e - \pi_h e) \, dS. \]

Let us explain a bit before we continue. In equation (7.27) 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 (7.31). In the first term one the right-hand side of equation (7.33), 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 7.1. There normal components, \( n \), will be pointing in opposite direction of each other and we get,

\[ \partial_n e + \partial_n u_h = n^+ \cdot \nabla^+ e + n^- \cdot \nabla^- e = n^+ \cdot (\nabla^+ e - \nabla^- e) = [\partial_n e] = - [\partial_n u_h]. \]
\[ \partial_n u_h \] is called a jump. Note that until now, we have only used equalities. Let us look at equation (7.34) in two terms.

\[
A \equiv \int_T R(e - \pi_h e) \, dx 
\leq ||R||_T ||e - \pi_h e||_T 
\leq ||R||_T C h_T ||\nabla e||_T 
\leq \frac{C h_T^2}{2} ||R||_T^2 + \frac{1}{2} ||\nabla e||_T^2
\] (4.36)

and

\[
B \equiv \frac{1}{2} \int_{\partial T} \partial_n u_h (e - \pi_h e) dS 
\leq \frac{1}{2} ||\partial_n u_h||_{\partial T} ||(e - \pi_h e)||_{\partial T} 
\leq ||\partial_n u_h||_{\partial T} C \sqrt{h_T} ||\nabla e||_{\omega_T} 
\leq \frac{C h_T}{e} ||\partial_n u_h||_{\partial T}^2 + e ||\nabla e||_{\omega_T}^2
\] (4.40)

In equation (7.37) and (7.41), we used Cauchy–Schwarz inequality. For equation (7.38) and (7.42), we used the interpolation estimates (7.23) and (7.24) respectively. Finally we used Cauchy’s inequality with \( \delta = \frac{1}{2} \) for equation (7.39) and \( \delta = \frac{\epsilon}{4} \) for equation (7.43). Let us sum up what we have so far:

\[
||e||_E^2 = \sum_{T \in T_h} A - B 
\leq \sum_{T \in T_h} A + B 
\leq \sum_{T \in T_h} \frac{1}{2} ||\nabla e||_T^2 + e ||\nabla e||_{\omega_T}^2 + \frac{C h_T^2}{2} ||R||_T^2 + \frac{C h_T}{e} ||\partial_n u_h||_{\partial T}^2.
\] (4.46)

Now we note that

\[
\sum_{T \in T_h} ||\nabla e||_T^2 = ||e||_E^2 \quad \text{and} \quad \sum_{T \in T_h} ||\nabla e||_{\omega_T}^2 \leq N ||e||_E^2,
\] (4.48)

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

\[
||e||_E^2 \leq \left( \frac{1}{2} + eN \right) ||e||_E^2 + \sum_{T \in T_h} \frac{C h_T^2}{2} ||R||_T^2 + \frac{C h_T}{e} ||\partial_n u_h||_{\partial T}^2
\] (4.49)

\[
\left( \frac{1}{2} - eN \right) ||e||_E^2 \leq \sum_{T \in T_h} \frac{C h_T^2}{2} ||R||_T^2 + \frac{C h_T}{e} ||\partial_n u_h||_{\partial T}^2.
\] (4.50)

Chapter 4. Finite element error estimate
Chapter 4. Finite element error estimate

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

$$
\|e\|_E \leq C \left( \sum_T h_T^2 \|R\|_T^2 + h_T \|\partial_n u_h\|_\partial_T^2 \right)^{\frac{1}{2}} \equiv E
$$

(4.51)

4.3 Adaptivity

In many applications we need the error to be less than 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 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 (7.51) in a more general form,

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

(4.52)

We consider two alternatives,

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

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 $\gamma_T$ 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 $\phi_0$, $\phi_5$ and $\phi_{10}$.

(b) Draw a similar picture of the derivatives $\phi_0'$, $\phi_5'$ and $\phi_{10}'$.

Exercise 4.2. Consider the equation

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

(4.53)

1By 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 $9 \times 9$ 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 (7.53).

Exercise 4.3.
Consider the following partial differential equation:
$$
\begin{align*}
-\frac{d^2 u}{dx^2} &= f \text{ in } (0, 1), \\
u'(0) &= 0, \\
u'(1) &= 0.
\end{align*}
$$

(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:
$$
\begin{align*}
-\nabla \cdot (a \nabla u) &= f \text{ in } \Omega, \\
u &= 0 \text{ on } \partial \Omega,
\end{align*}
$$
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 $\Omega$) 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 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 $H^1$?

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 $b_i = \int_\Omega \phi_i f \, dx$ by $f(x_i) \int_\Omega \phi_i \, dx$.

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 5.2 in $L_2$, $L_\infty$, and $L_1$ norms. Is the order of the approximation the same?

**Exercise 4.8.** Consider the same problem as in 5.2 in 3D. Assume that you tolerate a $H^1$ error of $1.0 \times 10^{-2}$. What polynomial order of the Lagrange finite element gives you the answer in minimal amount of computational time? Re-do the experiments with tolerance $1.0 \times 10^{-4}$ and $1.0 \times 10^{-6}$.
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, \(V = V(T)\), defined on finite subsets of the domain \(\Omega\).

**Example:** Piecewise linear in 1–D
Figure 4.1 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 4.2 shows the (global) basis functions of this function space and figure 4.3 show the local basis function on an element \(T\) and \(T'\).

**Example:** Piecewise linear in 2–D
Figure 4.4 shows a linear combination of piecewise linear basis functions forming a function \(u_h\), on a triangle. The different color indicate where the different basis functions contribute. Figure 4.5 shows a (global) basis functions and figure 4.6 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 \((T, V, L)\), 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 \(V = V(T)\) is a finite dimensional function space on \(T\) of dimension \(n\);

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

**Example:** \((T)\)
Figure 4.7 shows different kinds of domains, \(T\), for different dimensions, \(d = 1, 2, 3\). **Example:** \((V)\)

- \(V = \mathcal{P}_q(T) = \{\text{polynomials on } T \text{ of degree } \leq q\}\)
- \(V = [\mathcal{P}_q(T)]^d\)
- \(V = \text{subspace of } \mathcal{P}_q(T)\)
Figure 5.1: Function that is composed of a linear combination of basis functions

\[ u_h = \cdots + 5\phi_{j-1} + 7\phi_j + 6\phi_{j+1} + \cdots \]

Figure 5.2: Basis functions (global)
Figure 5.3: Local basis functions

Figure 5.4: Function on a triangle that is composed of a linear combination 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
Chapter 5. Finite element function spaces

Figure 5.6: Local basis functions 2-D

Figure 5.7: Illustration of different domains, $T$. 
Example: \((L)\)

- \(L(v) = v(x)\)
- \(L(v) = \int_T v(x) \, dx\)
- \(L(v) = \int_S v \cdot n \, dS\)

Is the standard piecewise linear element, \(P_1\), a finite element?

i. \(T\) is a interval, triangle or tetrahedron: ok

ii. \(V = \{v : v(x) = a + bx\} \equiv P_1(T)\)
   \(\dim V = n = d + 1\) : ok

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

To be able to show that \(L\) is a basis, we need the following lemma.

**Lemma 5.1. Unisolvence**

\(L\) is a basis for the dual space \(V'\), if and only if \(Lv = 0\) implies \(v = 0\). This can be expressed as:

\[L\text{ is basis for } V' \iff (Lv = 0 \Rightarrow v = 0)\]

**Proof.** Take \(\ell \in V'\) and let \(\{\phi_i\}_{i=1}^n\) be a basis for \(V\). First we look at the left-hand side;

- \(L\) is basis for \(V' \iff \exists! \alpha \in \mathbb{R}^n : \ell = \sum_{j=1}^n \alpha_j \phi_j\)
- \(\Rightarrow \exists! \alpha \in \mathbb{R}^n : \ell \phi_i = \sum_{j=1}^n \alpha_j \phi_j \Rightarrow \beta = 0 \text{ for } i = 1, \ldots, n\)
- \(\Rightarrow \exists! \alpha \in \mathbb{R}^n : A\alpha = b\)
- \(\Rightarrow A\) is invertible

Now we look at the right-hand side;

- \((Lv = 0 \Rightarrow v = 0) \iff \ell \sum_j \beta_j \phi_j \Rightarrow \beta = 0 \text{ for } i = 1, \ldots, n\)
- \(\Rightarrow \sum_j \beta_j \phi_j \Rightarrow \beta = 0 \text{ for } i = 1, \ldots, n\)
- \(\Rightarrow A^T \beta = 0\)
- \(\Rightarrow A^T\) is invertible
- \(\Rightarrow A\) is invertible

To sum up:

\(L\) is basis for \(V' \iff A\) is invertible \(\iff (Lv = 0 \Rightarrow v = 0)\).
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 $\{\varphi_i\}_{i=1}^n$ for a finite element $(T, V, L)$ is the unique basis satisfying

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

A nodal basis has the desired property that if, $u_h = \sum_{j=1}^n u_j \varphi_j$, then $\ell_i(u_h) = u_i$.

**Example:**

We look at $P_1$ elements on triangle with corners at $x_1, x_2$ and $x_3$,

- $x_1 = (0, 0), \quad \ell_1 v = v(x_1) \quad \varphi_1(x) = 1 - x_1 - x_2$
- $x_2 = (1, 0), \quad \ell_2 v = v(x_2) \quad \varphi_2(x) = x_1$
- $x_3 = (0, 1), \quad \ell_3 v = v(x_3) \quad \varphi_3(x) = x_2$.

From this we see that $\varphi_1, \varphi_2$ and $\varphi_3$ are a nodal basis.

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

$$\sum_{i=1}^n a_{jk} \psi_k = \varphi_i$$

$$\ell_i(\sum_{i=1}^n a_{jk} \psi_k) = \delta_{ij}$$

$$\ell_i(\psi_k a_{jk} = \delta_{ij}$$

$$A a^T = I$$

$A$ is the generalized Vandermonde matrix. Solving for $a$ gives the nodal basis!

**5.1.1 Conforming**

We will introduce some important function spaces:

- $H^1(\Omega) = \{v \in L^2(\Omega) : \nabla v \in L^2(\Omega)\}$ (5.1)
- $H(\text{div}; \Omega) = \{v \in L^2(\Omega) : \nabla \cdot v \in L^2(\Omega)\}$ (5.2)
- $H(\text{curl}; \Omega) = \{v \in L^2(\Omega) : \nabla \times v \in L^2(\Omega)\}$ (5.3)
Chapter 5. Finite element function spaces

Note:

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

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, \( \text{CG}_q(T) \subset H^1(\Omega) \).

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 \( (\text{CG}_q) \) is defined for \( q = 1, 2, \ldots \) by

\[ T \in \{ \text{interval, triangle, tetrahedron} \}, \]
\[ V = P_q(T), \]
\[ \ell_i(v) = v(x^i), \quad i = 1, \ldots, n(q), \]

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

\[
 x = \begin{cases} 
 i/q, & 0 \leq i \leq q, \quad T \text{ interval}, \\
 (i/q, j/q), & 0 \leq i + j \leq q, \quad T \text{ triangle}, \\
 (i/q, j/q, k/q), & 0 \leq i + j + k \leq q, \quad T \text{ tetrahedron}. 
\end{cases}
\]

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

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

Figure 4.8 show the Lagrange elements for different dimensions and how the nodal points are placed.

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

**Definition 5.4** (Raviart–Thomas element). The Raviart–Thomas element \( (\text{RT}_q) \) is defined for \( q = 1, 2, \ldots \) by

\[ T \in \{ \text{triangle, tetrahedron} \}, \]
\[ V = [P_{q-1}(T)]^d + xP_{q-1}(T), \]
\[ L = \begin{cases} 
 \int_T v \cdot n \ p \ ds, & \text{for a set of basis functions } p \in P_{q-1}(f) \text{ for each facet } f, \\
 \int_T v \cdot p \ dx, & \text{for a set of basis functions } p \in [P_{q-2}(T)]^d \text{ for } q \geq 2. 
\end{cases} \]

The dimension of RT\(_q\) is

\[ n(q) = \begin{cases} 
 q(q+2), & T \text{ triangle}, \\
 \frac{1}{2}q(q+1)(q+3), & T \text{ tetrahedron}. 
\end{cases} \]
Figure 5.8: The Lagrange (CG\(_q\)) elements. \(q\) is the order of the elements, \(d\) is the dimension 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 \(P_{q-1}\) on facets (edges and faces, respectively) and, for the higher degree elements, interior moments against \([P_{q-2}]^d\).
Chapter 5. Finite element function spaces

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 $P^q$ on facets (edges and faces, respectively) and, for the higher degree elements, interior moments against $\text{NED}_1^{q-1}$.

Figure 4.9 shows some Raviart–Thomas elements.

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

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

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

$$ V = [P_q(T)]^d, $$

$$ L = \left\{ \begin{array}{ll}
\int_f \mathbf{v} \cdot n p \, ds, & \text{for a set of basis functions } p \in P_q(f) \text{ for each facet } f, \\
\int_T \mathbf{v} \cdot p \, dx, & \text{for a set of basis functions } p \in \text{NED}_1^{q-1}(T) \text{ for } q \geq 2.
\end{array} \right. $$

where $\text{NED}_1$ refers to the Nédélec $H(\text{curl})$ elements of the first kind.

The dimension of $\text{BDM}_q$ is

$$ n(q) = \left\{ \begin{array}{ll}
(q+1)(q+2), & T \text{ triangle}, \\
\frac{1}{2}(q+1)(q+2)(q+3), & T \text{ tetrahedron}.
\end{array} \right. $$

Figure 4.10 shows the Brezzi–Douglas–Marini elements.

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

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

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

$$ V = [P_q(T)]^2, $$

$$ L = \left\{ \begin{array}{ll}
\int_e \mathbf{v} \cdot t p \, ds, & \text{for a set of basis functions } p \in P_q(e) \text{ for each edge } e, \\
\int_T \mathbf{v} \cdot p \, dx, & \text{for a set of basis functions } p \in \text{RT}_{q-1}(T), \text{ for } q \geq 2.
\end{array} \right. $$
where $t$ is the edge tangent, and in three dimensions by

$$T = \text{tetrahedron}, \quad \mathcal{V} = [\mathcal{P}_q(T)]^3, \quad \mathcal{L} = \begin{cases} 
\int_e v \cdot t \, p \, dl, & \text{for a set of basis functions } p \in \mathcal{P}_q(e) \text{ for each edge } e, \\
\int_f v \cdot p \, ds, & \text{for a set of basis functions } p \in \mathcal{RT}_{q-1}(f) \text{ for each face } f, \text{ for } q \geq 2 \\
\int_T v \cdot p \, dx, & \text{for a set of basis functions } p \in \mathcal{RT}_{q-2}(T), \text{ for } q \geq 3.
\end{cases}$$

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

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

Figure 4.11 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:

\[-\mu \Delta u + v \nabla u + cu = f \text{ in } \Omega\]

\[u = g \text{ on } \partial \Omega\]

Here \(v\) is typically a velocity, \(\mu\) is the diffusivity, \(c\) a reaction constant, 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 for \(\mu = 0\). For \(\mu = 0\) the Dirichlet conditions should only be set on the inflow domain \(\Gamma\); 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 close to the solution of the reduced problem; that is, for \(\mu = 0\) except close to the non-inflow boundary \(\partial \Omega \setminus \Gamma\). Here, there will typically be a boundary layer \(\exp(\|v\|_\infty x / \mu)\). 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:

\[-u_x - \alpha u_{xx} = 0, \quad u(0) = 0, u(1) = 1.\]  

The analytical solution is:

\[u(x) = \frac{e^{-x/\alpha} - 1}{e^{-1/\alpha} - 1}.\]

Hence, for \(\alpha \to 0\), both \(e^{-x/\alpha}\) and \(e^{-1/\alpha}\) 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

\[\int_0^1 -u_x v + \alpha u_x v_x \, dx = 0, \quad \forall v \in H^1_{(0,1)},\]
Chapter 6. Discretization of a convection-diffusion problem

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 $x = 1$. Further, we will consider a $\mu = 0.01$, a relatively large $\mu$, to enable us to see the differences on a relatively coarse mesh.

Both the numerical and analytical solutions are shown in Figure 5.1. 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
from dolfin import *

for N in [10, 100]:
    mesh = UnitInterval(N)
    V = FunctionSpace(mesh, "CG", 1)
    u = TrialFunction(V)
    v = TestFunction(V)
    alpha_value = 1.0e-2
    alpha = Constant(alpha_value)
    f = Constant(0)
    h = mesh.hmin()
    a = (-u.dx(0)*v + alpha*u.dx(0)*v.dx(0))*dx
    L = f*v*dx
```

Python code
To understand Example 5.1 we first remark that the discretization corresponds to the following central finite difference scheme:

\[- \frac{\mu}{h^2} \left( u_{i+1} - 2u_i + u_{i-1} \right) - \frac{\nu}{2h} \left( u_{i+1} - u_{i-1} \right) = 0, \quad i = 1, \ldots, N - 1 \]

Clearly, if \( \mu = 0 \) then the scheme reduces to

\[- \frac{\nu}{2h} \left( u_{i+1} - u_{i-1} \right) = 0, \quad i = 1, \ldots, N - 1 \]

Here, it is clear that \( u_{i+1} \) is coupled to \( u_{i-1} \), but not \( u_i \). 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:

\[ \frac{d^2u}{dx^2} (x_i) = \frac{1}{h^2} [u_{i+1} - 2u_i + u_{i-1}] \quad \text{if} \quad \nu < 0, \]

\[ \frac{d^2u}{dx^2} (x_i) = \frac{1}{h^2} [u_{i+1} - u_{i-1}] \quad \text{if} \quad \nu > 0. \]

Using this scheme oscillations will disappear, but the approximation is only 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

\[ \frac{u_{i+1} - u_{i-1}}{2h} \quad \text{central scheme, first order derivative} \]

\[ + \frac{h}{2} \frac{u_{i+1} + 2u_i - u_{i-1}}{h^2} \quad \text{central scheme, second order derivate} \]

\[ = \frac{u_i - u_{i-1}}{h} \quad \text{upwind scheme} \]

Hence, upwinding is equivalent to adding artificial diffusion with \( \epsilon = h/2 \); that is, in both cases we
actually solve the problem
\[-(\mu + \epsilon)u_{xx} + v u_x = f.\]

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 5.1 with artificial diffusion added.

**Example 6.2. Stabilization using artificial diffusion**

Consider again the following 1D problem convection diffusion problem:

\[- u_x - \alpha u_{xx} = 0, \quad \text{ (6.3)}\]
\[ u(0) = 0, u(1) = 1. \quad \text{ (6.4)}\]

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 \( u \in H^1_{L(0,1)} \) such that

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

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
beta_value = 0.5
beta = Constant(beta_value)
eta = Constant(0)

h = mesh.hmin()

a = (-u.dx(0)*v + alpha*u.dx(0)*v.dx(0) + beta*h*u.dx(0)*v.dx(0))*dx
```

Figure 5.2 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 5.2 for the corresponding fine mesh. The reason is that the scheme is now first order, while
the scheme in Example 5.1 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

\[-\mu \Delta u + v \nabla u = f \quad \text{in} \; \Omega,\]
\[ u = g \quad \text{on} \; \partial \Omega.\]

The weak formulation reads:
Find \( u \in H^1_0 \) such that

\[ a(u, w) = b(w) \quad \forall w \in H^1_0,\]
Chapter 6. Discretization of a convection-diffusion problem

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

where

\[ a(u, w) = \int_{\Omega} \mu \nabla u \nabla w \, dx + \int_{\Omega} \nu \nabla u w \, dx, \]

\[ b(w) = \int_{\Omega} f w \, dx. \]

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

\[ a(u_h, v_h) = (f, v_h) \quad \forall \, v_h \in V_{h,0}. \quad (6.5) \]

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 5.2, can be done as:

Find \( u_h \in V_{h,g} \) such that

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

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

\[
a(u_h, v_h) = (f, v_h) \quad \forall v_h \in W_h.
\]

Again, \( V_h \) and \( W_h \) 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:

\[
A_{ij} = a(N_i, N_j) = \int_\Omega \mu \nabla N_i \cdot \nabla N_j \, dx + \int_\Omega v \nabla N_i \cdot N_j \, dx,
\]

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

\[
A_{ij} = a(N_i, L_j) = \int_\Omega \mu \nabla N_i \cdot \nabla L_j \, dx + \int_\Omega v \nabla N_i \cdot L_j \, dx
\]

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 \( L_j = N_j + \beta v \cdot \nabla N_j \). Writing out the matrix \( A_{ij} \) in (5.6) now gives

\[
A_{ij} = a(N_i, N_j + \beta v \cdot \nabla N_j)
\]

\[
= \int_\Omega \mu \nabla N_i \cdot \nabla (N_j + \beta v \cdot \nabla N_j) \, dx + \int_\Omega v \nabla N_i \cdot (N_j + \beta v \cdot \nabla N_j) \, dx
\]

\[
= \underbrace{\int_\Omega \mu \nabla N_i \cdot N_j \, dx + \int_\Omega v \nabla N_i \cdot N_j \, dx}_{\text{standard Galerkin}}
\]

\[
+ \beta \underbrace{\int_\Omega \mu \nabla N_i \cdot \nabla (v \nabla N_j) \, dx}_{=0 \text{ for linear elements}} + \beta \underbrace{\int_\Omega v \cdot \nabla N_i v \cdot N_j \, dx}_{\text{Artificial diffusion in } v \text{ direction}}
\]

Notice that also the righthand side changes

\[
b(L_j) = \int_\Omega f L_j \, dx = \int_\Omega f (N_j + \beta v \cdot \nabla N_j) \, dx
\]
Chapter 6. Discretization of a convection-diffusion problem

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 $u_{h, sd} \in H^1_0$ such that

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

(6.7)

where

$$a_{sd}(u, w) = \int_\Omega \mu \nabla u \nabla w \, dx + \int_\Omega v \nabla u w \, dx + \beta \int_\Omega \sum_{e} -\Delta u (v \cdot \nabla w) \, dx,$$

$$b_{sd}(w) = \int_\Omega f w \, dx + \beta \int_\Omega f v \cdot \nabla w \, dx.$$

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

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

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

$$\|u\| = (\int (\nabla u)^2 \, dx)^{1/2}$$

\(1\)Has the argument for reducing non-homogeneous Dirichlet conditions to homogeneous Dirichlet conditions been demonstrated elsewhere?
Chapter 6. Discretization of a convection-diffusion problem

and the standard $H^1$ norm
\[ \|u\| = (\int (\nabla u)^2 + u^2 \, dx)^{1/2} \]
are equivalent. Hence, on $H^1_0$ the $\| \cdot \|$ 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. Further, let
\[
\begin{align*}
    b(u, w) &= \int_\Omega \mu \nabla u \nabla w \, dx \\
    c_v(u, w) &= \int_\Omega v \nabla u w \, dx \\
    a(u, w) &= a(u, w) + b(u, w)
\end{align*}
\]
Furthermore, assuming for the moment that $u \in H^1_0, w \in H^1_0$, we have
\[
\begin{align*}
    c_v(u, w) &= \int_\Omega v \nabla w u \, dx \\
    &= -\int_\Omega v \nabla w u \, dx - \int_\Omega \nabla \cdot v u w \, dx - \int_\Gamma u w v \cdot n \\
    &= -c_v(w, u).
\end{align*}
\]
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:
\[
a(u, u) = b(u, u) \geq \mu \|u\|^2.
\]

The second condition, the boundedness of $a$ (2), follows by applying Cauchy-Schwartz inequality if we assume bounded flow velocities $\|v\|_\infty$.
\[
\begin{align*}
    a(u, v) &= \int_\Omega \mu \nabla u \nabla v \, dx + \int_\Omega v \nabla u v \, dx \\
    &\leq \mu \|u\| \|v\| + \|v\|_\infty \|u\| \|v\|_0 \\
    &\leq (\mu + \|v\|_\infty C_\Omega) \|u\| \|v\|.
\end{align*}
\]

The third condition simply means that the right-hand side needs to be in the dual space of $H^1_0$. Hence, we obtain the following bounds by Lax-Milgram’s theorem:
\[
\|u\| \leq \frac{\mu + C_\Omega \|v\|_\infty}{\mu} \|f\|_{-1}.
\]
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

$$
a(u,u) = \int_{\Omega} \mu \nabla u \nabla u \, dx + \int_{\Omega} v \nabla u u \, dx
\geq \mu \|u\|_0 \|v\|_\infty - \|v\|_\infty \|u\|_0
\geq (\mu - \|v\|_\infty C_\Omega) \|u\|_0,
$$

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.

### 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 $I_h : H^t \to V_h$ where $V_h$ is a piecewise polynomial field of order $t-1$ with the property that for any $u \in H^t(\Omega)$

$$
\|u - I_h u\|_m \leq Bh^{t-m} \|u\|.
$$

**Proof.** The bounds on the interpolation error is provided by the Bramble-Hilbert lemma for $t \geq 2$ 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.

**Theorem 6.3. Cea’s lemma**

Suppose the conditions for Lax-Milgram’s theorem is satisfied and that we solve the linear problem (5.5) on a finite element space of order $t-1$. Then,

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

Here $C_1 = \frac{CB}{\alpha}$, where $B$ comes from the approximation property and $\alpha$ 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 $V_h \subset V$:

$$
\alpha \|u - u_h\|_V \leq a(u - u_h, u - u_h) = a(u - u_h, u - v) - a(u - u_h, v - u_h) \leq C \|u - u_h\|_V \|u - v\|_V.
$$

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

$$
\|u - u_h\|_V \leq \frac{C}{\alpha} \|u - I_h u\| \leq C h^{t-1} \|u\|.
$$
where \( t - 1 \) is the order of the polynomials of the finite elements.

We remark, as mentioned above, that \( \frac{C}{n} \) is large for convection dominated problems and that this is what causes the poor approximation on the coarse grid, shown in Example 5.1.

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

\[
\| u \|_v = \left( h \| v \cdot \nabla u \|^2 + \frac{1+h}{2} | \nabla u | \right)^{1/2}
\]

(6.8)

**Theorem 6.4.** Suppose the conditions for Lax-Milgram’s theorem is satisfied in the Hilbert space defined by the SUPG norm (5.8) and that we solve the SUPG problem (5.7) on a finite element space of order \( t - 1 \). Then,

\[
\| u - u_h \|_v \leq C h^{t+1/2} \| u \|_{t+1}
\]

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

### 6.5 Exercise

**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.

**Exercise 6.2.** 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 \( \alpha \). Compare with the corresponding constant estimated from Example 5.1.

**Exercise 6.3.** 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 \( \| \cdot \| \) and the \( \| \cdot \|_v \) norms. Compare with the corresponding constant estimated from Example 5.1.

**Exercise 6.4.** Investigate whether the coercivity condition holds when a homogeneous Neumann condition is assumed on the outflow. You may assume that \( \nu \cdot n > 0 \).

**Exercise 6.5.** Consider the eigenvalues of the operators, \( L_1, L_2, \) and \( L_3 \), where \( L_1 u = u_x, \) \( L_2 u = -\alpha u_{xx}, \) \( \alpha = 1.0 \times 10^{-5} \), and \( L_3 = L_1 + L_2 \), with homogeneous Dirichlet conditions. For which of the operators are the eigenvalues positive and real? Repeat the exercise with \( L_1 = x u_x \).

**Exercise 6.6.** 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 6.7.** 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 $\Omega$. We assume that $\Omega$ 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

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

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 6.1. Clearly, the velocity is approximated satisfactory, but the pressure oscillate widely and is nowhere near the actual solution.

```python
from dolfin import *

def u_boundary(x):
    pass
```
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).
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 6.2.

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.

Let us first start with a weak formulation of Stokes problem: Find $u \in H^1_{D_{0,8}}$ and $p \in L^2$.

\[
\begin{align*}
  a(u,v) + b(p,v) &= f(v), & v &\in H^1_{D_{0,8}}, \\
  b(q,u) &= 0, & q &\in L^2,
\end{align*}
\]
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

\[ a(u, v) = \int \nabla u : \nabla v \, dx, \]
\[ b(p, v) = \int p \nabla \cdot v \, dx, \]
\[ f(v) = \int f v \, dx + \int_{\partial \Omega} N h v \, ds. \]

Here \( H^1_{DG} \) 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 referring 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

\[ a(u_h, v_h) + b(p_h, v_h) = f(v_h), \quad \forall v_h \in V_{0,h}, \quad (7.5) \]
\[ b(q_h, u_h) = 0, \quad \forall q_h \in Q_h. \quad (7.6) \]

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

\[
\begin{bmatrix}
A & B^T \\
B & 0
\end{bmatrix}
\begin{bmatrix}
u \\
p
\end{bmatrix} =
\begin{bmatrix}
f \\
0
\end{bmatrix}
\quad (7.7)
\]

Here

\[ A_{ij} = a(N_i, N_j) = \int \nabla N_i \nabla N_j \, dx, \quad (7.8) \]
\[ B_{ij} = b(L_i, N_j) = \int \nabla L_i N_j \, dx. \quad (7.9) \]

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 (10.47) 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

\[ 0 = B u = B A^{-1} f - B A^{-1} B^T p \]

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

\[ B A^{-1} B^T p = B A^{-1} f \]

This equation is often called the pressure Schur complement. The question is then reduced to whether \( B A^{-1} B^T \) is invertible. Consider the following two situations:
Chapter 7. Stokes problem

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

\[
\text{kernel}(B) = 0, \quad \text{where } B \in \mathbb{R}^{m \times n}
\]

An equivalent statement is that

\[
\max_v (v, B^T p) > 0 \quad \forall p.
\] (7.10)

Alternatively,

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

Here, \( \beta > 0 \). We remark that (6.10) and (6.11) are equivalent for a finite dimensional matrix. However, in the infinite dimensional setting of PDEs (6.10) and (6.11) are different. Inequality (6.10) allow \((v, B^T p)\) to approach zero, while (6.11) requires a lower bound. For the Stokes problem, the corresponding condition is crucial:

\[
\sup_{v \in H^1_D, g} \frac{(p, \nabla \cdot u)}{\|u\|_1} \geq \beta \|p\|_0 > 0, \quad \forall p \in \mathbb{L}^2
\] (7.12)

Similarly, to obtain order optimal convergence rates, that is

\[
\|u - u_h\|_1 + \|p - p_h\|_0 \leq Ch^k \|u\|_{k+1} + Dh^{\ell+1} \|p\|_{\ell+1}
\]

where \( k \) and \( \ell \) are the polynomial degree of the velocity and the pressure, respectively, the celebrated Babuska-Brezzi condition has to be satisfied:

\[
\sup_{v \in V_{h,d}} \frac{(p, \nabla \cdot v)}{\|v\|_1} \geq \beta \|p\|_0 > 0, \quad \forall p \in Q_h
\] (7.13)
We remark that the discrete condition (6.13) does not follow from (6.12). 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 7.1.1.** For saddle point problems on the form (6.5)-(6.6) four conditions have to be satisfied in order to have a well-posed problem:

- **Boundedness of $a$:**
  \[ a(u_h, v_h) \leq C_1 \| u_h \|_{V_h} \| v_h \|_{V_h}, \quad \forall u_h, v_h \in V_h, \]  
  \[ (7.14) \]

- **Boundedness of $b$:**
  \[ b(u_h, q_h) \leq C_2 \| u_h \|_{V_h} \| q_h \|_{Q_h}, \quad \forall u_h \in V_h, q_h \in Q_h, \]  
  \[ (7.15) \]

- **Coersivity of $a$:**
  \[ a(u_h, u_h) \geq C_3 \| u_h \|^2_{V_h}, \quad \forall u_h \in V_h, \]  
  \[ (7.16) \]

- **“Coerivility” of $b$:**
  \[ \sup_{u_h \in V_h} \frac{b(u_h, q_h)}{\| u_h \|_{V_h}} \geq C_4 \| q_h \|_{Q_h}, \quad \forall q_h \in Q_h. \]  
  \[ (7.17) \]

For the Stokes problem, (6.14)-(6.16) are easily verified, while (6.17) often is remarkably difficult unless the elements are designed to meet this condition.

### 7.2 Examples of elements

#### 7.2.1 The Taylor-Hood element

The Taylor-Hood elements are quadratic for the velocity and linear for pressure

\[
\begin{align*}
\text{v : } N_i & = a_i^p + b_i^p x + c_i^p y + d_i^p xy + e_i^p x^2 + f_i^p y^2 \\
\text{p : } L_i & = a_i^p + b_i^p x + c_i^p y
\end{align*}
\]

and are continuous across elements

![Taylor-Hood: quadratic element for v and linear element for p](image)

For the Taylor-Hood element we have the following error estimate:

\[
\| u - u_h \|_1 + \| p - p_h \|_0 \leq C h^2 (\| u \|_3 + \| p \|_2)
\]
7.2.2 The Crouzeix–Raviart element

This element is linear in velocity and constant in pressure

\[ v : N_i = a_i^v + b_i^v x + c_i^v y \]
\[ p : L_i = a_i^p \]

The \( v \) element is continuous only in the mid-point of each side (see ??), and the \( p \) element is discontinuous. This ensures that it satisfies the Babuska-Brezzi condition.

For the Crouzeix–Raviart element we have the following error estimate:

\[ \|u - u_h\|_1 + \|p - p_h\|_0 \leq Ch(\|u\|_2 + \|p\|_1) \]

7.2.3 The \( P1-P0 \) element

If we on the other hand choose to locate the nodal points on the corners in the \( v \) element as shown in ?? (called a \( P1-P0 \) element) the inf-sup condition is not satisfied and we get oscillations in the pressure term.

7.2.4 The Mini element

The mini element is linear in both velocity and pressure, but the velocity element contains a cubic bubble. Notice that elements that are linear in both \( v \) and \( p \) will not satisfy the inf-sup condition.
Thus we add the extra bubble in $v$ to give an extra degree of freedom as depicted in ??.

Figure 7.6: Mini: linear element with bubble for $v$ and linear element for $p$

For the Mini element we have the following error estimate:

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

7.3 Stabilization techniques to circumvent the Babuska-Brezzi condition

Stabilization techniques typically replace the system:

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

with an alternative system

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

where $\epsilon$ is properly chosen.

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

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

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

$$
\begin{align*}
p &= (-\epsilon D)^{-1} (\epsilon d - Bu) \\
\downarrow \\
Au + B^T (-\epsilon D)^{-1} (\epsilon d - Bu) &= f \\
(A + \frac{1}{\epsilon} B^T D^{-1} B) u &= f + D^{-1} d
\end{align*}
$$
Chapter 7. Stokes problem

\((A + \frac{1}{\epsilon} B^T D^{-1} B)\) 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. \(\nabla \cdot \mathbf{v} = \epsilon \Delta p\). Pressure stabilization. Motivated through mathematical intuition (from the convection-diffusion equation).

2. \(\nabla \cdot \mathbf{v} = -\epsilon \mathbf{p}\). Penalty method. Typically, one uses the Velocity-Schur complement

3. \(\nabla \cdot = -\epsilon \frac{\partial p}{\partial t}\). 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.

7.4 Exercises

Exercise 7.1. Show that the conditions (6.14)-(6.16) are satisfied for \(V_h = H^1_0\) and \(Q_h = L^2\).

Exercise 7.2. Show that the conditions (6.14)-(6.16) 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 (6.17) 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 \(u = (\sin(\pi y), \cos(\pi x)), p = \sin(2\pi x), \text{ and } f = -\Delta u - \nabla p\). Test whether the approximation is of the expected order for \(P_4 - P_3, P_4 - P_2, P_3 - P_2, \text{ and } P_3 - P_1\).

Exercise 7.7. Implement the problem \(u = (\sin(\pi y), \cos(\pi x)), p = \sin(2\pi x), \text{ and } f = -\Delta u - \nabla p\) and determine the order of the approximation of wall shear stress.
8 Iterative methods

By Anders Logg, Kent–Andre Mardal

This chapter concerns the numerical solution of large linear systems, 

$$ Au = b, $$

where the linear system comes from discretization of PDEs. In such linear systems $A$ is a $N \times N$ matrix, and $N$ might be between $10^6$ and $10^9$. Furthermore, the matrix is normally extremely sparse and contains only $O(N)$ nonzeros. However, $A^{-1}$ will typically be full. A Naive Gauss-elimination requires $O(N^2) - O(N^3)$ floating point operations and $O(N^{2/3}) - O(N^2)$ floating point numbers of storage. Iterative methods need $O(N)$ floating point operations and $O(N)$ floating point numbers of storage.

8.1 A simple iterative method: Richardson iteration

Let us consider the problem: Find $u$ such that 

$$ Au = b. $$

The Richardson iteration is 

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

where $\tau$ is a parameter that must be determined. We see that if $u^{n-1} = u$, then $u^n = u$. Hence the iteration will not change the solution.

The standard approach to analyze iterative methods is to look at what happens with the error. Let $e^n = u^n - u$. We subtract $u$ from both sides of (8.1) and obtain 

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

We may therefore quantify the error in the $L^2$ norm,

$$ \| e^n \| = \| e^{n-1} - \tau Ae^{n-1} \| \leq \| I - \tau A \| \| e^{n-1} \| $$

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

Let's say that we need to reduce the error by a factor of $\epsilon$, that is, we need $\| e^n \| \leq \epsilon$. Let $\rho = \| I - \tau A \|$, then

$$ \| e^n \| \leq \rho \| e^{n-1} \| \leq \rho^n \| e^0 \|. \quad (8.2) $$
Chapter 8. Iterative methods

Assuming equality in the equation (8.2) and \( \|e^n\|/\|e_0\| = \epsilon \). Then the number of iterations needed to achieve the desired error is:

\[ n = \frac{\log \epsilon}{\log \rho} \]

If \( n \) is independent of the resolution of the discretization, the algorithm will be order-optimal, meaning, the algorithm will cost \( O(N) \) floating point operations and storage.

Let us look at an example to see what happens with the error when we use the Richardson iteration (also the same happens for Jacobi, Gauss Seidel, etc.).

**Example 8.1 (1D Poisson equation).** The Richardson iteration on the Poisson equation in 1D, discretized with finite difference method (FDM).

\[
Lu = \begin{cases} 
-u'' = f & \text{for } x \in (0,1) \\
u(0) = u(1) = 0
\end{cases} \tag{8.3}
\]

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 (\( A = \text{tridiagonal}(-1,2,-1) \)). We have the same eigenvectors, but the eigenvalues are a little bit different: \( \mu_k = \frac{4}{h^2} \sin^2\left(\frac{k\pi h}{2}\right) \). Here \( h \) is the step length \( \Delta x \). We find the smallest and largest discrete eigenvalues \( \mu_{\min}(A) = \pi^2, \quad \mu_{\max}(A) = \frac{4}{h^2} \).

Let \( \tau = \frac{c}{\mu_{\max}} \) and assume that \( e^0 = \sum_{k=1}^{N} c_k v_k \), then, by the Richardson iteration:

\[
e^n = (I - \tau A)^en_{e^0} \\
= (I - \tau A)^n c_k v_k \\
= \sum_{k=1}^{N} (1 - \frac{c \mu_k}{\mu_{\max}})^n c_k v_k.
\]

When using the equality above and the fact that \( \{v_k\}_{k=1}^{N} \) are orthonormal, we can show that (show this!)

\[ \|e^n\| \leq (1 - c)^2n \|e^0\|. \]

Thus, we have convergence for \( 0 < c < 2 \).

However, different parts of the error will decrease at different speed! To illustrate this, first assume that \( e^0 = v_N \), where \( v_N \) is the eigenfunction corresponding to the largest eigenvalue, then

\[ e^n = (1 - c \frac{\mu_{\max}}{\mu_{\min}})^n v_N. \]

Setting \( c = 0.9 \) will reduce the error by a factor of 0.1 per iteration. Now assume that \( e^0 = v_1 \), where
$v_1$ is the eigenvector corresponding to the smallest eigenvalue, then

$$e^n = (1 - c \frac{H_{\min}}{H_{\max}})^n v_1 = (1 - c \frac{1}{\kappa(A)})^n v_1 \approx (1 - c h^2)^n v_1 \approx e^0$$

for small $h$. The convergence is very slow in this case.

The idea of writing $e^n = \sum_{k=1}^{N} c_k v_k$, where $v_k = \sin(k\pi x)$ are the eigenvectors of $A$, is motivated from Fourier series. If $e^n$ is mostly composed of $v_k$ for small $k$, we will refer to this as the low frequent error. Similarly if $e^n$ is mostly composed of $v_k$ for large $k$, we refer to this as the high frequent error.

In summary: From the example 8.1 we saw that high frequent error is removed quickly, while low frequent error stays essentially unchanged throughout the iterations. This is common for many methods like Jacobi, Gauss–Seidel, SOR and SSOR.

8.2 Generalizing by a preconditioning

The basic idea of preconditioning is to replace $Au = b$ with $BAu = Bb$. Both systems have the same solution (if $B$ is nonsingular). $B$ should be chosen in a way that $BA$ will have a “nicer” spectrum. Furthermore $Bu$ should cost $O(N)$ operations to evaluate. The generalized Richardson iteration becomes

$$u^n = u^{n-1} - \tau(Bu^{n-1} - b). \quad (8.4)$$

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 Preconditioner: Jacobi

Look at the $i$-th equation of the system $Au = b$, $\sum_{j=1}^{N} a_{ij} u_j = b_i$, this can be rewritten in an iterative way

$$u_i^n = \frac{1}{a_{ii}} (b_i - \sum_{j \neq i} a_{ij} u_j^{n-1}) \quad \text{which holds for} \quad i = 1, 2, ..., N. \quad (8.5)$$

On the Richardson iteration form this becomes,

$$u^n = u^{n-1} - D^{-1}(Au^{n-1} - b). \quad (8.6)$$

Here $D^{-1} = (\text{diag}(A))^{-1}$ is the preconditioner. The eigenvalues $\mu_k = \cos(\pi k h)$ of the Jacobi matrix $J = (I - D^{-1}A)$, are displayed in figure 8.1. Here $|\mu_k|$ close to 0 corresponds to high frequent errors and $|\mu_k|$ close to 1 corresponds to low frequent error. We recall from the previous section that high frequent error is easily handled, while low frequent errors is problematic. Therefor we would like $|\mu_k|$ close to zero.

\footnote{$\kappa(A)$ is the condition number of $A$, which is $\kappa(A) = |\lambda_{\max} / \lambda_{\min}|$.}
\footnote{By “nicer” spectrum, we mean smaller condition number, $\kappa BA < \kappa A$.}
8.2.2 Preconditioner: Relaxed Jacobi

By looking at the Jacobi iteration (8.5), we see that $u^n_i$ is computed based on $u^{n-1}_j$ for $j \neq i$. We can extend (8.5) by including information from the previous iteration $u^{n-1}_i$

$$u^n_i = (1 - \omega)u^{n-1}_i + \frac{\omega}{a_{ii}} (b_i - \sum_{j \neq i} a_{ij}u^{n-1}_j) \quad \text{for} \quad i = 1, 2, \ldots, N. \quad (8.7)$$

On the Richardson iteration form this becomes,

$$u^n = u^{n-1} - \omega D^{-1} (Au^{n-1} - b). \quad (8.8)$$

The relaxation parameter $\omega$ needs to be chosen. By choosing $\omega = 2/3$, we can see from figure 8.2 that more of the eigenvalues correspond to high frequent errors ($|\mu_k|$ close to 0). While the low frequent error ($\mu_k$ close to 1) is still a problem.

8.2.3 Preconditioner: Gauss–Seidel

A natural extension to the Jacobi iteration (8.5) is to use $u^n_i$ instead of $u^{n-1}_j$ for $j < i$ in the sum, since these values is already computed. This method is known as the Gauss–Seidel iteration,

$$u^n_i = \frac{1}{a_{ii}} (b_i - \sum_{j < i} a_{ij}u^n_j - \sum_{j > i} a_{ij}u^{n-1}_j) \quad \text{for} \quad i = 1, 2, \ldots, N. \quad (8.9)$$

Let $A = D + U + L$, where D is the diagonal, U is the upper diagonal part and L is the lower diagonal part of A. On the Richardson iteration form this becomes,

$$u^n = u^{n-1} - (D + L)^{-1} (Au^{n-1} - b). \quad (8.10)$$

We can apply the same ideas for the relaxed Jacobi and get a relaxed Gauss–Seidel,

$$u^n_i = (1 - \omega)u^{n-1}_i + \frac{\omega}{a_{ii}} (b_i - \sum_{j < i} a_{ij}u^n_j - \sum_{j > i} a_{ij}u^{n-1}_j) \quad \text{for} \quad i = 1, 2, \ldots, N. \quad (8.11)$$
Chapter 8. Iterative methods

Figure 8.2: Plot of the eigenvalues of the relaxed Jacobi method on FDM discretized 1D Poisson with \( N = 150 \) and \( \mu_k = 1 + \omega(\cos(\pi kh) - 1) \).

On the Richardson iteration form,

\[
\begin{align*}
    u^n & = u^{n-1} - \omega(D + L)^{-1}(Au^{n-1} - b).
\end{align*}
\]  

(8.12)

Note that Gauss–Seidel is not symmetric (unless \( A \) is a diagonal matrix). As mention above, these methods handles high frequent error well, while the low frequent errors remains almost unchanged. The low frequent errors can be handled by multigrid methods.

8.3 Multigrid

The idea is to use smoothers, \( S \), like Jacobi or Gauss–Seidel, to remove the high frequent errors, while the low frequent error is handled with a coarser mesh. These coarser meshes are more efficient to compute on, and are well suited to represent smooth components. The smoother, \( S \), is then used on the coarse meshes.

So, let’s concretize, we have \( Au = b \). Assume we have performed \( m \) relaxed Jacobi iterations. The error is \( e^m = u - u^m \) and assume that the error is smooth (containing only low frequent errors). Then,

\[
Ae^m = r^m \quad \text{where} \quad r^m = b - Au^m,
\]

(8.16)

\( r^m \) is the residual. Let us further introduce two different grids,

\[
\begin{align*}
    \Omega_h & = \{ x_j = jh, j = 1, 2, ..., N + 1 \} \quad \text{(8.13)} \\
    \Omega_H & = \{ x_j = jH, j = 1, 2, ..., M + 1 \}. \quad \text{(8.14)}
\end{align*}
\]

An obvious choice is \( H = 2h \). Instead of solving

\[
A_h e_h = r_h, \quad \text{(8.16)}
\]

we solve

\[
A_H e_H = r_H, \quad \text{(8.17)}
\]
Note that we update the solution by adding the error $u = u^m + e^m$. To be able to solve (8.17) instead of (8.16), we need an operator that transfer $e_h \rightarrow e_H$ and $r_h \rightarrow r_H$. A suitable choice is a standard restriction operator $I^H_h$ defined by

$$u^H_j = (I^H_h u^h_j) = \frac{1}{4} (u^h_{2j-1} + 2u^h_{2j} + u^h_{2j+1}) \quad \text{for } j = 1, \ldots, M.$$  (8.18)

Now we may compute $e_H$ as

$$e_H = A^{-1}_H r_H = A^{-1}_H I^H_h r_h.$$

To go back from the coarser mesh we use an interpolation operator defined as $I^H_h = 2(I^H_h)^T$, such that

$$u^H_{2j} = u^h_j \quad \text{for } j = 1, \ldots, M,$$

$$u^H_{2j+1} = \frac{1}{2} (u^h_j + u^h_{j+1}) \quad \text{for } j = 0, \ldots, M.$$  (8.19)  (8.20)

The 2–level algorithm is summarized in Algorithm 1. Here we have included a smoothing sweep before and after we solve the coarser problem. The post–smoothing gives symmetry, and it also removes any high frequent components introduced by the interpolation operator $I^H_h$. One may define a multigrid operator $P : \mathbb{R}^{N+1} \rightarrow \mathbb{R}^{N+1}$, it can then be shown that if $S$ is symmetric and linear, then the multigrid $P$ is symmetric and linear. Prove it!

In the 2–level algorithm we coarsened the grid only once, then solved the system with a direct solver.

Algorithm 1 2–level multigrid algorithm

1: Remove the high frequent errors by a Pre–smoothing S (Richardson, Jacobi, GS, e.i.),

$$u^{n+1/3} = u^n - S(Au^n - b).$$  (8.21)

2: Solve on a coarser grid,

$$u^{n+2/3} = u^{n+1/3} - I^H_h A^{-1}_H (I^H_h (Au^{n+1/3} - b)).$$  (8.22)

3: Post–smoothing,

$$u^{n+1} = u^{n+2/3} - S(Au^{n+2/3} - b).$$  (8.23)

In reality we may need to make the grid coarser several times before solving directly. We do the same as for the 2–level algorithm, but use the restriction operator $I^H_h$ several times to send the residual down to a coarse grid, smoothing each time to remove the relative high-frequent error. When the grid is small enough, we find the error with a direct solver. The interpolation operator $I^H_h$ is used to send the error back to the original grid-size. Again we are smoothing after each interpolation. This procedure is illustrated in figure 8.3 and is know as the V–cycle algorithm. Note that the 2–level algorithm is a special case of the V–cycle algorithm, where we only do one restriction and one interpolation.
8.4 Spectral equivalence and order optimal algorithms

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

\[
c_1 (A^{-1} v, v) \leq (Bv, v) \leq c_2 (A^{-1} v, v) \quad \forall \ v
\]  

(8.24)

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

Proper conditions for an order optimal preconditioner \( B \):

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

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

\[
\|e^n\|_A \leq \rho_A \|e^{n-1}\|_A.
\]

Because \( BA \) is symmetric with respect to the \( A \)-inner product, \( \rho_A \) can be stated in terms of the eigenvalues \( \mu_i \) of \( BA \), such that

\[
\rho_A = \|I - \tau BA\|_A = \sup_{\mu_i} |1 - \tau \mu_i| = \max(|1 - \tau \mu_0|, |1 - \mu_N|),
\]
where \( \mu_0 \) and \( \mu_N \) is the smallest and largest eigenvalue. We may choose \( \tau = \frac{2}{\mu_0 + \mu_N} \) which makes

\[
\rho_A = 1 - \tau \mu_0 = 1 - \frac{2 \mu_0}{\mu_0 + \mu_N} = \frac{\mu_N - \mu_0}{\mu_N + \mu_0} = \frac{\kappa - 1}{\kappa + 1}.
\]

This leads to

\[
\|e^n\|_A \leq (\frac{\kappa - 1}{\kappa + 1})^n \|e^0\|_A.
\]

Hence, if the condition number is independent of the discretization parameter, then the convergent rate is independent as well! This gives an order optimal algorithm (Convergence in \( O(N) \) iterations independent of the discretization).

Multigrid produces a linear operator that is spectrally equivalent with \( A^{-1} \) for elliptic operators.

### 8.5 Krylov methods and preconditioning

Any linear iteration method may be written as a Richardson iteration with a preconditioner. However, iterations methods like Conjugate Gradient method, GMRES, Minimal Residual method, and BiCGStab, are different. These are nonlinear iteration methods. We will not go in detail on these methods, but they should be used together with a preconditioner, such as the Richardson methods. Furthermore, some of them have special requirements:

**Classification of methods:** We classify the methods according to the matrices they solve. The matrix may be:

- Symmetric Positive Definite (SPD): Use Conjugate Gradient with an SPD preconditioner, see also Exercise 8.3.
- Symmetric and indefinite: Use Minimal Residual method with and SPD preconditioner, see also Exercise 8.5.
- Positive: GMRES and ILU (or AMG) are often good, but you might need to experiment, see also Exercise 8.4.
- Nonsymmetric and indefinite: All bets are off.

### 8.6 Examples

**Example 8.2** (The Richardson iteration on a Poisson problem). Let us consider the Richardson iteration on a two point boundary problem discretized with a finite difference method,

\[
(Au)_i = -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f_i, \quad i = 1, \ldots, N,
\]

\[
u_0 = 0, \quad \nu_{N+1} = 0.
\]

In this example we will choose \( f_i = \sin(\pi x_i), \quad i = 1, \ldots, N \) and \( N = 150 \). For the initial vector \( v^0 \) we use random numbers between 0 and 1.

Figure 8.4 shows the numerical solution at different number of iterations for two different \( \tau \) values. In both cases it is clear that the solution improves quite a lot in the first 10 iterations, but the improvement slows down since
Chapter 8. Iterative methods

Figure 8.4: The numerical solution $u^n$ from example 8.2 at different steps of the Richardson method. The right figure uses the optimal $\tau = \frac{2}{\mu_{\text{max}} + \mu_{\text{min}}}$, while in the left figure $\tau = \frac{0.9}{\mu_{\text{max}}}$. A large part of the error remains essentially unchanged during the iterations. The left figure clearly shows that the solution after 1000 iterations is dominated by both low and high frequency components. The figure on the left, where $\tau = \frac{0.9}{\mu_{\text{max}}}$, shows that the high frequency components of the error can be removed by choosing a larger $\tau$ than the optimal $\tau$. However, in both cases the low frequency components of the error remains essentially unchanged throughout the iterations. The code used in this example is as follows:

```python
def create_stiffness_matrix(N):
    h = 1.0/(N+1)
    A = numpy.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 = numpy.matrix(A)
    return A

N = 150
x = numpy.arange(0, 1, 1.0/(N))
f = numpy.matrix(numpy.sin(pi*x)).transpose()
u_true = (1.0/(pi*pi))*numpy.matrix(numpy.sin(pi*x)).transpose()
u0 = numpy.matrix(numpy.random.random(N)).transpose()
A = create_stiffness_matrix(N)
eigenvalues = numpy.sort(linalg.eigvals(A))
mu_max = eigenvalues[-1]
mu_min = eigenvalues[0]

print "Highest eigenvalue", mu_max
print "Lowest eigenvalue", mu_min

def iterate(tau):
    u_prev = u0; u = u0
    pylab.plot(x,u, 'o')
    for i in range(1001):
        if i == 10 or i == 100 or i == 1000 or i == 1000:
            A = create_stiffness_matrix(N)
            eigenvalues = numpy.sort(linalg.eigvals(A))
            mu_max = eigenvalues[-1]
            mu_min = eigenvalues[0]

        u_prev = u
        u = A*mu*u
        pylab.plot(x,u, 'o')
```

---

**Python code**

```python
def create_stiffness_matrix(N):
    h = 1.0/(N+1)
    A = numpy.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 = numpy.matrix(A)
    return A

N = 150
x = numpy.arange(0, 1, 1.0/(N))
f = numpy.matrix(numpy.sin(pi*x)).transpose()
u_true = (1.0/(pi*pi))*numpy.matrix(numpy.sin(pi*x)).transpose()
u0 = numpy.matrix(numpy.random.random(N)).transpose()
A = create_stiffness_matrix(N)
eigenvalues = numpy.sort(linalg.eigvals(A))
mu_max = eigenvalues[-1]
mu_min = eigenvalues[0]

print "Highest eigenvalue", mu_max
print "Lowest eigenvalue", mu_min

def iterate(tau):
    u_prev = u0; u = u0
    pylab.plot(x,u, 'o')
    for i in range(1001):
        if i == 10 or i == 100 or i == 1000 or i == 1000:
            A = create_stiffness_matrix(N)
            eigenvalues = numpy.sort(linalg.eigvals(A))
            mu_max = eigenvalues[-1]
            mu_min = eigenvalues[0]

        u_prev = u
        u = A*mu*u
        pylab.plot(x,u, 'o')```
Chapter 8. Iterative methods

```python
pylab.plot(x, u)
u = u_prev - tau*(A*u_prev - f)
u_prev = u
pylab.plot(x, u_true)
pylab.xlabel('x')
pylab.ylabel('u')
pylab.legend(['$u^0$', '$u^{10}$', '$u^{100}$', '$u^{1000}$', '$u_{true}$'])
pylab.show()

tau = 0.9/mu_max
iterate(tau)
tau = 2/(mu_max + mu_min)
iterate(tau)
```

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

\[
\begin{align*}
u - \Delta u &= f, \quad \text{in } \Omega \\
\frac{\partial u}{\partial n} &= 0, \quad \text{on } \partial \Omega
\end{align*}
\]

where \( \Omega \) 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 8.5 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 \( O(N^2) - O(N^3) \). 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
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:
```

Python code
Figure 8.5: CPU time (in seconds) for solving a linear system of equation with $N$ degrees of freedom (x-axis) for different solvers

```python
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
```
8.7 Exercise

**Exercise 8.1.** Implement the Richardson iteration of the 1D Poisson problem with homogeneous Dirichlet conditions, using finite differences and a multigrid preconditioner.

**Exercise 8.2.** Estimate the convergence factor for the Jacobi and Gauss-Seidel iteration for the 1D Poisson problem from Example 8.3.

**Exercise 8.3.** Test CG method without preconditioner, 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 8.4.** Test CG, BiCGStab, GMRES with ILU, AMG, and Jacobi preconditioning for

\[-\mu \Delta u + v \nabla u = f \quad \text{in } \Omega\]

\[u = 0 \quad \text{on } \partial \Omega\]

Where $\Omega$ 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 8.5.** The following code snippet shows the assembly of the matrix and preconditioner for a Stokes problem:

```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:

---

# 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
9 Finite element assembly

By Anders Logg, Kent–Andre Mardal

When using the FEM we get a linear system on the form

\[ AU = b, \tag{9.1} \]

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:

\[
\text{for } i = 1, \ldots, N \text{ do} \\
\quad \text{for } j = 1, \ldots, N \text{ do} \\
\quad \quad A_{ij} = a(\phi_j, \phi_i) \\
\quad \text{end for} \\
\text{end for}
\]

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

9.1 Local to global mapping \( i_T \)

We look at the local degrees of freedom and the global degrees of freedom. Figure (9.1) shows local and global degrees of freedoms. From the figure we can see that the local to global mapping is

\[ i_T = (0, 1, 3, 11, 10, 5) \]
\[ i_{T'} = (1, 2, 3, 7, 11, 6). \]

Note that the numbering is arbitrary as long as neighboring \( T \) and \( T' \) agree. However some numbering schemes are more efficient then others, especially for parallel computing.

Note that

\[ \phi_{i_T(i)}|_T = \phi_i^T \iff \phi_{\bar{i}}|_T = \phi_{i_T(i)}^{\bar{i}} \]

if it exists.

\( I \) and \( J \) are the counters for the global numbering.
9.2 The element matrix $A^T$

Assume that $a(u,v) = \sum_{T \in T} a_T(u,v)$. Example,

$$a(u,v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx = \sum_{T \in T} \int_{T} \nabla u \cdot \nabla v \, dx.$$  \hfill (9.2)

We then define

$$A^T_{ij} = a_T(\phi^T_i, \phi^T_j).$$ \hfill (9.3)

This is a small, typically dense matrix. We now note that

$$A_{ij} = a(\phi_i, \phi_j) = \sum_{T \in T} a_T(\phi_i, \phi_j) = \sum_{i \in I_{ij}} a_T(\phi_i, \phi_j), \text{ all triangles where both } \phi_i \text{ and } \phi_j \text{ are nonzero},$$ \hfill (9.4)

$$= \sum_{T \in T} a_T(\phi^T_{i^{-1}(j)}, \phi^T_{j^{-1}(i)})$$ \hfill (9.5)

$$= \sum_{T \in T} A^T_{i^{-1}(j)j^{-1}(i)}$$ \hfill (9.6)

The algorithm becomes,

\begin{verbatim}
for $T \in \mathcal{T}$ do
  for $i = 1, \ldots, n$ do
    for $j = 1, \ldots, n$ do
      $A^-_{(i)T(j)} + = A^-_{ij}$
    end for
  end for
end for
\end{verbatim}

or equivalent

\begin{verbatim}
for $T \in \mathcal{T}$ do
  Compute $A^T$
end for
\end{verbatim}

Figure 9.1: Red numbers indicate the local numbering, black number are the gobal numbering. Here $P_2$ elements where used, $\text{dim } P_2 = 6$. 
Chapter 9. Finite element assembly

9.3 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 9.2.

$$x = F_T(\hat{x}) = B_T \hat{x} + c_T, \quad (9.8)$$

where $B_T$ is a matrix and $c_T$ is a vector. Let us look at a reference basis function for $P_1$ elements,

$$\Phi_0 = 1 - \hat{x}_1 - \hat{x}_2 \quad (9.9)$$
$$\Phi_1 = \hat{x}_1 \quad (9.10)$$
$$\Phi_2 = \hat{x}_2. \quad (9.11)$$

Also recall that $\ell_i(\phi_j) = \delta_{ij}$. The mapping becomes,

$$F_T(\hat{x}) = \Phi_0(\hat{x}) x_0 + \Phi_1(\hat{x}) x_1 + \Phi_2(\hat{x}) x_2 \quad (9.12)$$

9.4 How do we compute $A^T$?

We consider first the mass matrix

$$M_{ij}^T = \int_T \phi_j^T \phi_i^T \, d\hat{x} \quad (9.13)$$

$$= \int_{\hat{T}} \phi_j^T (F_T(\hat{x})) \phi_i^T (F_T(\hat{x})) \, det(F_T') \, d\hat{x} \quad (9.14)$$

$$= \int_{\hat{T}} \Phi_j \Phi_i \, det(F_T') \, d\hat{x} \quad (9.15)$$

$$= det(F_T') \int_{\hat{T}} \Phi_j \Phi_i \, d\hat{x}. \quad (9.16)$$
Now we consider the poisson equation (stiffness matrix)

\[ A_{ij}^T = \int_T \nabla \phi_i^T \nabla \phi_j^T \, dx \]  \hspace{2cm} (9.17)

\[ = \int_T \frac{\partial}{\partial x_k} \phi_i^T \frac{\partial}{\partial x_k} \phi_j^T \, dx \]  \hspace{2cm} (9.18)

\[ = \int_T \left( \frac{\partial \hat{x}_m}{\partial x_k} \frac{\partial}{\partial \hat{x}_m} \right) \Phi_j \left( \frac{\partial \hat{x}_n}{\partial x_k} \frac{\partial}{\partial \hat{x}_n} \right) \Phi_i \, det(F_T^T) \, d\hat{x} \]  \hspace{2cm} (9.19)

\[ = \int_T \left( J^{-1} \frac{\partial \Phi_j}{\partial \hat{x}_m} J^{-1} \frac{\partial \Phi_i}{\partial \hat{x}_n} \right) \, det(J) \, d\hat{x} \]  \hspace{2cm} (9.20)

\[ = \int_T \left( J^{-T} \nabla \Phi_j \right) \left( J^{-T} \nabla \Phi_i \right) \, det(J) \, d\hat{x}. \]  \hspace{2cm} (9.21)
10  The incompressible Navier–Stokes equations

By Anders Logg, Kent–Andre Mardal

10.1  The equations

The Navier–Stokes equations are important equations used for describing the motion of fluid. For several decades now, they have been an active research area (and still is), both in analytical and in numerical mathematics. The incompressible Navier–Stokes equations can be stated as,

\[
\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho} \nabla p + \nu \Delta u + g \tag{10.1}
\]

\[
\nabla \cdot u = 0. \tag{10.2}
\]

Here \( u \) and \( p \) are the unknown velocity and pressure, \( \rho \) is the fluid density, \( g \) is the body force and \( \nu \) is the kinematic viscosity. Possible boundary and initial conditions are,

\[
u \frac{\partial u}{\partial n} - p \cdot n = h \quad \text{on} \quad \partial \Omega_N, \quad \text{(Neumann)} \tag{10.4}
\]

\[
u \Delta u + p |_{n} = h \quad \text{on} \quad \partial \Omega, \quad \text{(Dirichlet)} \tag{10.3}
\]

\[
u \frac{\partial u}{\partial n} = 0 \quad \text{on} \quad \partial \Omega. \tag{10.5}
\]

The Navier–Stokes equations are a composition of operators with different characters:

1. \( \nu \Delta u \) — elliptic
2. \( \frac{\partial u}{\partial t} - \nu \Delta u \) — parabolic
3. \( \frac{\partial u}{\partial t} + u \cdot \nabla u \) — nonlinear hyperbolic
4. \( \nu \Delta u \) — constrained problem (Stokes)

\( \nabla \cdot u = 0 \)

Good and well-understood methods exist for operator 1 to 4, but the combination of them is difficult. In physics these can model turbulence, which suggests that the equations are hard and “relatively unstable” problems. The combinations 2 – 4 and 3 – 4 are particularly hard, although 2 – 4 is now relatively well understood. There are two main ways of discretizing the Navier–Stokes equations, first discretize in time and then in space, or vice versa.
10.2 Operator splitting

10.2.1 A naive scheme

We will first consider discretizing in time prior to space. Let us look at a naive derivation of an explicit scheme,

\[
\frac{v^{n+1} - v^n}{\Delta t} + v^n \cdot \nabla v^n = -\frac{1}{\rho} \nabla p^n + \nu \Delta v^n + g^n. \tag{10.6}
\]

The problems with this scheme is that there is no update for pressure and \( \nabla \cdot v^{n+1} \) is not equal to zero. Let us try an implicit handling of \( p \)

\[
v^{n+1} + \frac{\Delta t}{\rho} \nabla p^{n+1} = v^n - \Delta t v^n \cdot \nabla v^n + \Delta t \Delta v^n + \Delta t g^n \tag{10.7}
\]

\[\nabla \cdot v^{n+1} = 0. \tag{10.8}\]

We may eliminate \( v^{n+1} \) by taking the divergence of 10.7. By doing this, we obtain a Poisson equation for the pressure \( p \),

\[
\Delta p^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot (v^n - \Delta t v^n \cdot \nabla v^n + \Delta t \Delta v^n + \Delta t g^n).
\]

The problem with this scheme is that there are no natural boundary conditions for \( p \), we will explain this in more detail later.

Projections schemes are variants of this approach, but they are more sophisticated. Still they have problems with boundary condition.

10.2.2 An explicit scheme

We will now improve the previous scheme. Instead of finding \( v^{n+1} \) from equation 10.6 we find a velocity guess, a tentative velocity \( v^* \),

\[
v^* = v^n - \Delta t v^n \cdot \nabla v^n - \frac{\Delta t}{\rho} \beta \nabla p^n + \Delta t \Delta v^n + \Delta t g^n. \tag{10.9}
\]

\( \beta \) is a constant we chose to be between 0 and 1. The tentative velocity is not divergence free; that is, \( \nabla \cdot v^* \neq 0 \). To ensure that \( v^{n+1} \) is divergence free, we add a correction velocity, \( v^c \), such that

\[
v^{n+1} = v^* + v^c. \tag{10.10}
\]

\( v^{n+1} \) involves a pressure update,

\[
v^{n+1} = v^n - \Delta t v^n \cdot \nabla v^n - \frac{\Delta t}{\rho} \nabla p^{n+1} + \Delta t \Delta v^n + \Delta t g^n. \tag{10.11}
\]

By subtracting 10.9 from 10.11, we obtain

\[
v^{n+1} - v^* = -\frac{\Delta t}{\rho} (\nabla p^{n+1} - \beta \nabla p^n). \tag{10.12}
\]
Furthermore, since $\nabla \cdot v^{n+1} = 0$, we can take the divergence of (10.12) to eliminate $v^{n+1}$. Also we let $\phi = p^{n+1} - \beta p^n$. This gives a Poisson equation

$$\Delta \phi = \frac{\rho}{\Delta t} \nabla \cdot v^\ast. \quad (10.13)$$

The scheme becomes:

1. Compute $v^\ast$ by solving equation (10.9).

2. Compute $\phi$ by solving the Poisson equation (10.13).

3. Update the pressure by $p^{n+1} = \beta p^n + \phi$.

4. Update the velocity by $v^{n+1} = v^\ast - \frac{\Delta t}{\rho} \nabla \phi$.

By choosing $\beta = 0$ we get Chorin’s projection scheme and by choosing $\beta = 1$ we get the incremental pressure correction scheme (IPCS). The problem with these schemes are the boundary conditions. The Poisson equation (10.13) requires boundary conditions for the pressure on the whole boundary, to be well posted. While the Navier–Stokes equation only requires pressure boundary conditions at one point. Therefore we need to construct boundary conditions for $\phi$. There are two ways of doing this.

Alternative 1: Multiply the Navier–stokes equation (10.2) with a normal vector $n$ to obtain,

$$\frac{\partial p}{\partial n} = (\nu \Delta u - \frac{\partial u}{\partial t} - u \cdot \nabla u + g) \cdot n.$$

We must then assume that the Navier–Stokes equation is valid on the boundary.

Alternative 2: Restrict $\nabla \phi$ to only look at the boundary, we get

$$\nabla \phi|_{\partial \Omega} = \frac{\Delta t}{\rho} (v^\ast - v^{n+1})|_{\partial \Omega} = 0,$$

since $v^{n+1}$ and $v^\ast$ have the same boundary conditions.

These two alternatives produce different boundary conditions. The reason we have problem with the boundary conditions is that we impose one to many condition, the problem becomes overdetermined! The consequence is an error in the boundary layer for the pressure. This explicit scheme is fairly unstable ($\Delta t$ must be small compared to $\Delta x$, for high velocities). Let us derive the corresponding implicit scheme.

10.2.3 An (semi) implicit scheme

First we compute a tentative velocity,

$$v^\ast + \Delta t v^\ast \cdot \nabla v^\ast + \frac{\Delta t}{\rho} \beta \nabla p^n - \Delta t v^\ast = v^n + \Delta t g^n. \quad (10.14)$$

The equation above contains a nonlinear term $v^\ast \cdot \nabla v^\ast$. We can linearize this term by using $v^n \cdot \nabla v^\ast$ instead. This gives the semi implicit scheme

$$v^\ast + \Delta t v^n \cdot \nabla v^\ast + \frac{\Delta t}{\rho} \beta \nabla p^n - \Delta t v^n = v^n + \Delta t g^n. \quad (10.15)$$
What we really want to solve is,
\[ v^{n+1} + \Delta t v^n \cdot \nabla v^{n+1} + \frac{\Delta t}{\rho} \nabla p^{n+1} - \Delta t \Delta \varphi^n = v^n + \Delta t g^n. \]  
(10.16)

As before, we introduce a correction velocity \( v^c \), such that \( v^{n+1} = v^* + v^c \). We subtract (10.15) from (10.16) and obtain
\[ v^{n+1} - v^* = v^c = s(v^c) - \frac{\Delta t}{\rho} \nabla \varphi, \quad \text{where} \quad s(v^c) = \Delta t (-v^n \cdot \nabla v^c + \nu \Delta v^c). \]  
(10.17)

So far we have done nothing “illegal” and since
\[ \nabla \cdot v^{n+1} = 0 \Rightarrow \nabla \cdot v^c = -\nabla \cdot v^*, \]  
we may write the system as
\[ \begin{align*}
\nabla \cdot v^c - s(v^c) + \frac{\Delta t}{\rho} \nabla \phi &= 0, \\
\nabla \cdot v^c &= -\nabla \cdot v^*.
\end{align*} \]  
(10.18) (10.19)

The problem now, is the \( s(v^c) \) term. This term is of order \( \Delta t \), therefore we can neglect it by doing a \( \Delta t \) perturbation! By setting \( s(v^c) = 0 \) and taking the divergence of equation (10.18), we get,
\[ \Delta \varphi = \frac{\rho}{\Delta t} \nabla \cdot v^*. \]  
(10.20)

Here we have used that \( -\nabla \cdot v^c = \nabla \cdot v^* \). The scheme then becomes,

1. Compute \( v^* \) by solving the convection–diffusion equation (10.15).
2. Compute \( \phi \) by solving the Poisson equation (10.20).
3. Update the pressure by \( p^{n+1} = \beta p^n + \phi \).
4. Update the velocity by \( v^{n+1} = v^* - \frac{\Delta t}{\rho} \nabla \phi \).

Note that the boundary conditions requires changes, when neglecting \( s(v^c) \)!

In this sections we discretized the Navier–Stokes equations in time first. By doing this, we split the equations into simpler problems. We got a convection-diffusion equation for the tentative velocity and a Poisson problem for the pressure. At the end we updated the pressure and velocity. The main problem with these operator splitting methods are the boundary conditions. We do not have boundary conditions for the pressure on the whole boundary, which the Poisson problem requires. To fix this, we had two different ways of constructing boundary conditions, however these gave different boundary conditions. The consequence was high error in pressure and tangential velocity at the boundary layer. Note that we did not encounter any mixed problems (as we did for the Stokes problem). In the next section we will discretize in space prior to time.

10.3 Algebraic splitting

In this section we discretize first in space, with FEM, then discretize in time. The weak formulation reads:
Chapter 10. The incompressible Navier–Stokes equations

Find $u_h \in V_h, u_D$ and $p_h \in Q_h$ such that,

$$
\int_\Omega \frac{\partial u_h}{\partial t} \cdot v_h + (u_h \cdot \nabla u_h) \cdot v_h + \frac{1}{\rho} \nabla p_h \cdot v_h + \nu \nabla u_h : \nabla v_h \, dx = \int_\Omega f \cdot v_h \, dx \quad \forall v_h \in V_h, \theta_D
$$

(10.21)

$$
\int_\Omega (\nabla \cdot u_h) q_h \, dx = 0 \quad \forall q_h \in Q_h
$$

(10.22)

We have used integration by parts on the elliptic term and that,

$$
\int_\Omega f \cdot v_h \, dx = \int_\Omega g \cdot v_h \, dx + \int_{\partial \Omega} (h + p n) \cdot v_h \, dS.
$$

Let,

$$
u_h = \sum_i u_i N_i \quad \text{and} \quad v_h = N_j
$$

and

$$
p_h = \sum_i p_i L_i \quad \text{and} \quad q_h = L_j.
$$

When inserting this into the weak form, we obtain the following DAE(differential algebraic equation):

$$
M \dot{u} + K(u) u = -Q p + A u + f
$$

(10.23)

$$
Q^T u = 0.
$$

(10.24)

- $u$ – time derivative of $u$
- $M$ – mass matrix
- $K(u)$ – convection matrix
- $A$ – stiffness matrix
- $Q$ – discrete gradient matrix
- $Q^T$ – discrete divergence matrix

This system is challenging, so we begin with an algebraic projection scheme, which has similar approach as section 10.2.

10.3.1 An explicit scheme

Let us start with an explicit scheme for $\dot{u}$, First find a tentative velocity $u^*$,

$$
\frac{M(u^* - u^n)}{\Delta t} = -K(u^n) u^n - \beta Q p^n + A u^n + f^n.
$$

(10.25)

What we really want is

$$
\frac{M(u^{n+1} - u^n)}{\Delta t} = -K(u^n) u^n - Q p^{n+1} + A u^n + f^n,
$$

(10.26)

where $Q^T u^{n+1} = 0$. We seek a correction velocity, $u^c$, such that $u^{n+1} = u^* + u^c$. By subtracting (10.25) from (10.26), we get:

$$
\frac{1}{\Delta t} M u^c = -Q(p^{n+1} - \beta p^n)
$$
As before, we introduce $\phi = p^{n+1} - \beta p^n$, and get $u^c$ alone on the left-hand side of the equation.

$$u^c = -\Delta t M^{-1} Q \phi.$$  \hspace{1cm} (10.27)

Now we multiply equation (10.27) with $Q^T$ and use that

$$Q^T u^{n+1} = 0 \Rightarrow Q^T u^c = -Q^T u^*.$$  

This gives us a discrete Poisson equation for the pressure.

$$Q^T M^{-1} Q \phi = \frac{1}{\Delta t} Q^T u^*.$$  \hspace{1cm} (10.28)

The scheme then becomes:

1. Compute $u^*$ by solving equation (10.25).
2. Compute $\phi$ by solving the discrete Poisson equation (10.28).
3. Update the pressure by, $p^{n+1} = \beta p^n + \phi$.
4. Update the velocity by, $u^{n+1} = u^* - \Delta t M^{-1} Q \phi$.

Note that we don’t need to worry about boundary condition. They where “baked” into the weak formulation.

### 10.3.2 More implicit schemes

Let’s use the $\theta$-rule. The equations (10.23) and (10.24), can then be written as:

$$Nu^{n+1} + \Delta t Q p^{n+1} = q$$  \hspace{1cm} (10.29)

$$Q^T u^{n+1} = 0$$  \hspace{1cm} (10.30)

where

$$N = M + \theta \Delta t R(u^n),$$  \hspace{1cm} (10.31)

$$R(u^n) = K(u^n) - A,$$  \hspace{1cm} (10.32)

$$q = (M - (1 - \theta) \Delta t R(u^n)) u^n + \Delta t f^{n+1}.$$  \hspace{1cm} (10.33)

This system is nonsymmetric and indefinite, which is the worst kind to solve. We will try an algebraic projection scheme! First we begin with a tentative velocity:

$$Nu^* = q - \beta \Delta t Q p^n$$  \hspace{1cm} (10.34)

The governing equations for correction $u^c = u^{n+1} - u^*$ are obtained by subtracting (10.34) from (10.29) and using the fact that $Q^T u^{n+1} = 0 \Rightarrow Q^T u^c = -Q^T u^*$.

$$Nu^c + \Delta t Q \phi = 0$$  \hspace{1cm} (10.35)

$$Q^T u^c = -Q^T u^*$$  \hspace{1cm} (10.36)
Further, elimination of $u^c$ gives,

$$Q^T N^{-1} Q \phi = \frac{1}{\Delta t} Q^T u^*.$$  (10.37)

Equation (10.37) is called the Schur complement pressure equation. Solving this equation requires inverting $N$, which is not an option since $N^{-1}$ is dense ($N$ is sparse). We need an approximation $\tilde{N}^{-1}$ to $N^{-1}$. In other words, we solve

$$Q^T \tilde{N}^{-1} Q \phi = \frac{1}{\Delta t} Q^T u^*.$$  (10.38)

One simple approximation is to let $\tilde{N}^{-1}$ be equal to the identity matrix, $\tilde{N}^{-1} = I$.

Let us summarize the scheme:

1. Compute $u^*$ by solving equation (10.34).
2. Compute $\phi$ by solving the discrete Poisson equation (10.38).
3. Update the pressure by, $p^{n+1} = \beta p^n + \phi$.
4. Update the velocity by, $u^{n+1} = u^* - \Delta t \tilde{N}^{-1} Q \phi$.

### 10.3.3 Fully implicit schemes with a preconditioner

Eliminating $u^{n+1}$ in equation (10.29) by using equation (10.30), gives the following system

$$Q^T N^{-1} Q p^{n+1} = \frac{1}{\Delta t} Q^T q.$$  (10.39)

This equation is also called the Schur complement pressure equation. For simplicity, let us write this as $B p = b$. We which to form a Richardson iteration to solve this equation; that is,

$$p^{n+1,k+1} = p^{n+1,k} - C^{-1} (B p^{n+1,k} - b).$$  (10.40)

Written out, this becomes

$$p^{n+1,k+1} = p^{n+1,k} - C^{-1} (Q^T N^{-1} Q p^{n+1,k} - \frac{1}{\Delta t} Q^T q).$$  (10.41)

We can now show that the components involved in the splitting scheme in section 10.3.2, can be reused here. That is, we write up the splitting scheme in terms of the pressure,

$$p^{n+1} = p^n + \phi$$  (10.42)

$$p^{n+1} = p^n + (Q^T N^{-1} Q)^{-1} \frac{1}{\Delta t} Q^T u^*$$  (10.43)

$$p^{n+1} = p^n + (Q^T N^{-1} Q)^{-1} \frac{1}{\Delta t} Q^T (N^{-1} q - \Delta t N^{-1} Q p^n)$$  (10.44)

$$p^{n+1} = p^n - (Q^T N^{-1} Q)^{-1} (Q^T N^{-1} Q p^n - \frac{1}{\Delta t} Q^T N^{-1} q)$$  (10.45)

In equation (10.42), we found $\phi$ from solving equation (10.34) and in equation (10.43), we found $u^*$ from solving equation (10.34). We have used $\beta = 1$. Equation (10.45) can be generalized to an iteration on $p^{n+1},$

$$p^{n+1,k+1} = p^{n+1,k} - (Q^T N^{-1} Q)^{-1} (Q^T N^{-1} Q p^{n+1,k} - \frac{1}{\Delta t} Q^T N^{-1} q).$$  (10.46)
Chapter 10. The incompressible Navier–Stokes equations

A natural and good starting point, would be the solution from the previous time step, \( p^{n+1,0} = p^n \). The performance of this iteration will depend on the quality of \((Q^T \tilde{N}^{-1} Q)^{-1}\). We saw earlier that for \( \tilde{N}^{-1} \) set to be the inverse of a lumped mass matrix, then this operator, \( Q^T (\text{lumped}(M))^{-1} Q \) is quite close to a Laplacian. We can then use standard multigrid. One proposed form of \( C^{-1} \) is

\[
C^{-1} = \alpha_R B^{-1}_R + \alpha_D B^{-1}_D + \alpha_K B^{-1}_K.
\]

Where

- \( B^{-1}_R \) is a reactive preconditioner for \( Q^T M^{-1} Q \)
- \( B^{-1}_D \) is a diffusive preconditioner for \( Q^T A^{-1} Q \)
- \( B^{-1}_K \) is a convective preconditioner for \( Q^T K^{-1} Q \).

Here, both \( B^{-1}_R \) and \( B^{-1}_D \) have been well studied and it is known how to construct them.

Finally, for fully coupled solvers on

\[
\begin{bmatrix}
N & Q \\
Q^T & 0
\end{bmatrix}
\begin{bmatrix}
u \\
p
\end{bmatrix} =
\begin{bmatrix}
q \\
0
\end{bmatrix}.
\tag{10.47}
\]

It is natural to consider block preconditioner on the form

\[
\begin{bmatrix}
C_1 & 0 \\
0 & C_2
\end{bmatrix},
\]

here \( C_1 \) should be an approximation to \( N^{-1} \) and \( C_2 \) should be an approximation to \( (Q^T N^{-1} Q)^{-1} \).
11 The finite element method for time-dependent problems

By Anders Logg, Kent–Andre Mardal

Recall that there are two classes of problems:

\[ \begin{align*}
\text{ODE:} & \quad \dot{u} = f(u, t) \\
\text{PDE:} & \quad \dot{u} + A(u) = f(x, t)
\end{align*} \] (11.1)

11.1 The FEM for $\dot{u} = f$

**Strong form**

\[ \begin{align*}
\dot{u}(t) &= f(u(t), t), \quad t \in (0, T] \\
u(0) &= 0
\end{align*} \] (11.2)

\[ u : [0, T] \to \mathbb{R}^N \]

\[ f : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N \] (11.3)

**Weak form**

Find $u \in V$ such that

\[ \int_0^T v \cdot \dot{u} \, dt = \int_0^T v \cdot f \, dt \quad \forall v \in \hat{V}. \] (11.4)

Here, $V$ is called the trial space and $\hat{V}$ is the test space.

**Finite element method**

Find $\mathbf{U} \in V_k$ such that

\[ \int_0^T v \cdot \dot{\mathbf{U}} \, dt = \int_0^T v \cdot f \, dt \quad \forall v \in \hat{V}_k, \] (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.
Chapter 11. The finite element method for time-dependent problems

Figure 11.1

\[ I_n = (t_{n-1}, t_n) \]
\[ k_n = t_n - t_{n-1} = \text{time step} \] (11.6)

\[ V_k = \{ \text{continuous piecewise polynomials of degree } \leq q \} \]
\[ = \{ v \in [C(0, T)]^N : v|_{I_n} \in [P_q(I_n)]^N \forall I_n \} \] (11.7)

\[ \hat{V}_k = \{ \text{piecewise polynomials of degree } \leq q - 1 \} \]
\[ = \{ v : [0, T] \to \mathbb{R}^N : v|_{I_n} \in [P_{q-1}(I_n)]^N \forall I_n \} \] (11.8)

The continuous Galerkin method with \( q = 1 \)

Find \( U \in V_k \) such that
\[ \int_0^T v \cdot U \, dt = \int_0^T v \cdot f \, dt \quad \forall v \in \hat{V}_k, \] (11.9)

where
\[ V_k = \{ v \in [C(0, T)]^N : v|_{I_n} \in [P_1(I_n)]^N \forall I_n \} \] (11.10)
and
\[ V_k = \{ v : [0, T] \to \mathbb{R}^N : v|_{I_n} \in [P_0(I_n)]^N \forall I_n \}. \] (11.11)

Take \( v = 0 \) on \([0, T]\)\( \setminus I_n\), then
\[ \int_{I_n} v \cdot U \, dt = \int_{I_n} v \cdot f \, dt \quad \forall v \in [P_0(I_n)]^N. \] (11.12)
Take \( v = (0, \cdots, 0, 1, 0, \cdots, 0) \) (the value 1 is at position \( i \)), then

\[
\int_{I_i} \dot{U}_i \, dt = \int_{I_i} f_i \, dt \quad i = 1, \cdots, N, \forall I_n
\]

(11.13)

\[
\Rightarrow U_i(t_n) - U_i(t_{n-1}) = \int_{I_i} f_i \, dt \quad i = 1, \cdots, N, \forall I_n
\]

(11.14)

\[
\Rightarrow U_i(t_n) = U_i(t_{n-1}) + \int_{I_i} f_i \, dt \quad \forall I_n
\]

(11.15)

Let \( U^n = U(t_n) \) and \( U^{n-1} = U(t_{n-1}) \), then

\[
U^n = U^{n-1} + \int_{I_n} f_i \, dt \quad \forall I_n, \quad (11.17)
\]

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

\[
\int_{t_{n-1}}^{t_n} f \, dt \approx k_n f \left( \frac{U^{n-1} + U^n}{2}, \frac{t_{n-1} + t_n}{2} \right)
\]

(11.18)

and obtain

\[
U^n = U^{n-1} + k_n f \left( \frac{U^{n-1} + U^n}{2}, \frac{t_{n-1} + t_n}{2} \right).
\]

(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

\[
U^{n,j} = U^{n-1} + k_n f \left( \frac{U^{n-1} + U^{n,j}}{2}, \frac{t_{n-1} + t_n}{2} \right).
\]

(11.20)

An important question is: When does (11.20) converge? Remember the contraction mapping theorem:

\[
x^k = T(x^{k-1})
\]

(11.21)

converges if

\[
\| T' \| \leq M < 1.
\]

(11.22)
Here:

\[ T(x) = U^{n-1} + k_n f \left( \frac{U^{n-1} + x}{2}, \frac{t_{n-1} + t_n}{2} \right) \]  \hspace{1cm} (11.23)

\[ \Rightarrow T'(x) = k_n J \left( \frac{U^{n-1} + x}{2}, \frac{t_{n-1} + t_n}{2} \right) , \]  \hspace{1cm} (11.24)

where \( J \) is defined

\[ J_{ij} = \frac{\partial f_i}{\partial U_j} . \]  \hspace{1cm} (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).

\[ \dot{u} = \lambda u, \hspace{1cm} \lambda = 100 \]  \hspace{1cm} (11.26)

**Continuous Galerkin method with \( q > 1 \)**

Make an Anzats on each interval

\[ U(t) = \sum_{j=0}^{q} U^{n,j} \lambda_j^q (t) \]  \hspace{1cm} (11.27)

\[ \Rightarrow \int_{t_{n-1}}^{t_n} \sum_{j=0}^{q} U^{n,j} \lambda_j^q (t) \cdot \lambda_j^{q-1} (t) \, dt = \int_{t_{n-1}}^{t_n} \lambda_j^{q-1} (t) f_i \, dt \]  \hspace{1cm} (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**

\[ \dot{u} + A(u) = f \quad \text{in} \quad \Omega \times (0,T], \]
\[ u(\cdot,0) = u_0 \quad \text{in} \quad \Omega, \]
\[ + \text{BC}. \]  \hspace{1cm} (11.29)

**Weak form**

Find \( u \in V \) such that

\[ \int_0^T \int_\Omega \dot{u} \, v \, dx \, dt + \int_0^T \int_\Omega v A(u) \, dx \, dt = \int_0^T \int_\Omega v f \, dx \, dt \quad \forall v \in \mathcal{V}. \]  \hspace{1cm} (11.30)
Chapter 11. The finite element method for time-dependent problems

Finite element method

Find $u_{hk} \in V_{hk}$ such that

$$\int_{0}^{T} \int_{\Omega} v u_{hk} \, dx \, dt + \int_{0}^{T} \int_{\Omega} v A(u_{hk}) \, dx \, dt = \int_{0}^{T} \int_{\Omega} v f \, dx \, dt \quad \forall v \in \hat{V}_{hk}. \quad (11.31)$$

Solution algorithm

$$V_{hk} = \text{span}\{v = v_h v_k : v_h \in V_h, v_k \in V_k\} \quad (11.32)$$

$$\hat{V}_{hk} = \text{span}\{v = v_h v_k : v_h \in \hat{V}_h, v_k \in \hat{V}_k\} \quad (11.32)$$

$$\int_{0}^{T} \int_{\Omega} v_h v_k u_{hk} \, dx \, dt + \int_{0}^{T} \int_{\Omega} v_h v_k A(u_{hk}) \, dx \, dt = \int_{0}^{T} \int_{\Omega} v f \, dx \, dt \quad (11.33)$$

$$\int_{0}^{T} v_k \int_{\Omega} v_h u_{hk} \, dx \, dt + \int_{0}^{T} v_k \int_{\Omega} v_h A(u_{hk}) \, dx \, dt = \int_{0}^{T} \int_{\Omega} v f \, dx \, dt \quad (11.34)$$

Take

$$u_{hk}(x, t) = \sum_{j=1}^{N} U_j(t) \phi_j(x) \quad (11.35)$$

and $A$ linear. Then

$$\int_{0}^{T} v_k \sum_{j=1}^{N} U_j \int_{\Omega} \phi_j \phi_j \, dx \, dt + \int_{0}^{T} v_k \sum_{j=1}^{N} U_j \int_{\Omega} \phi_j A(\phi_j) \, dx \, dt = \int_{0}^{T} \int_{\Omega} v f \, dx \, dt \quad (11.36)$$

We define the mass matrix $M$ and the "stiffness matrix" $A_k$ by

$$M_{ij} = \int_{\Omega} \phi_i \phi_j \, dx, \quad (11.37)$$

$$A_{k,ij} = \int_{\Omega} \phi_i A(\phi_j) \, dx \quad (11.38)$$

Thus, we obtain

$$\int_{0}^{T} v_k \cdot M U \, dt + \int_{0}^{T} v_k \cdot A_k(U) \, dt = \int_{0}^{T} v_k b \, dt \quad (11.39)$$

where

$$U = (U_1, U_2, \ldots, U_N)^T, \quad (11.40)$$

$$b = \int_{\Omega} v_h f \, dx. \quad (11.41)$$

The overall solution algorithm is sketched in Figure 11.3.

**Example 11.2** (Heat equation).

$$\dot{u} - \Delta u = f \quad (11.42)$$

**FEM in space gives**

$$M \dot{U} - A U = b \quad (11.43)$$
\[ \dot{u} + A(u) = f \]

FEM in space

\[ M\dot{U} + A_k(U) = b \]

FEM in time

Timestepping scheme

Figure 11.3

Continuous Galerkin with \( q = 1 \) leads to

\[
\left( M + \frac{k_n A}{2} \right) U^n = \left( M + \frac{k_n A}{2} \right) U^{n-1} + k_n b_n.
\] (11.44)