PDE Project Course

2. Implementation of the Finite Element Method

Karin Kraft

kakr@math.chalmers.se

Department of Mathematical Sciences

Lecture Plan

- From Last Friday
- Matrix Notation
- Assembling the Matrices
- Mapping from a Reference Element
- Solving Nonlinear Problems
- Time-stepping

Warning: ξ is used with two different meanings!

From Last Friday

Model problem/B.C:

Equation:

$$
\begin{aligned}
-\nabla \cdot (\epsilon(x)\nabla u) &= f \quad \text{on } \Omega \\
u &= g \quad \text{on } \Gamma_1 \\
(\epsilon(x)\nabla u) \cdot n &= p \quad \text{on } \Gamma_2
\end{aligned}
$$

Variational Formulation

Variational formulation from Greens formula:
Find

$$v \in V = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_1 \right\}$$

such that

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

Variational Formulation, continued

where

$$a(u,v) = \int_\Omega \epsilon(x) \nabla u \cdot \nabla v \, dx$$

and

$$L(v) = \int_\Omega f v \, dx + \int_{\Gamma_2} p v \, ds$$

FEM

- Put $p = 0$ for simplicity.
- Introduce the finite-dimensional spaces $V_h = \hat{V}_h = \mathrm{span}\{\varphi_j\}_{j=1}^N$.
- The approximate solution is given by $U(x) = \sum_{j=1}^N \xi_j \varphi_j(x)$.
- Linear system:

$$A_h = \int_\Omega \nabla \varphi_j \cdot \nabla \hat{\varphi}_i \, dx, \quad b = \int_\Omega f \hat{\varphi}_i \, dx$$

Matrix Notation

The stiffness matrix S

The *stiffness matrix* S is given by

$$S_{ij} = \int_\Omega \epsilon(x) \nabla \varphi_j(x) \cdot \nabla \hat{\varphi}_i(x) \, dx.$$

In one dimension, with $\Omega = (a, b)$, we have

$$S_{ij} = \int_a^b \epsilon(x) \varphi_j'(x) \hat{\varphi}_i'(x) \, dx.$$

The load vector b

The *load vector* b is given by

$$b_i = \int_\Omega f(x)\hat{\varphi}_i(x)\, dx.$$

Example: Poisson's equation

For Poisson's equation, $-\nabla \cdot (\epsilon(x)\nabla u(x)) = f(x)$ in Ω, with homogenous Neumann B.C we obtain the system

$$S\xi = b,$$

where S is the stiffness matrix, b is the load vector and ξ is the vector containing the degrees of freedom for the finite element solution U given by

$$U(x) = \sum_{j=1}^{N} \xi_j \varphi_j(x).$$

The mass matrix M

The *mass matrix* M is given by

$$M_{ij} = \int_\Omega \varphi_j(x)\hat{\varphi}_i(x)\,dx.$$

The convection matrix B

The *convection matrix* B is given by

$$B_{ij} = \int_\Omega \beta(x) \cdot \nabla \varphi_j(x) \hat{\varphi}_i(x) \, dx.$$

In one dimension, with $\Omega = (a, b)$, we have

$$B_{ij} = \int_a^b \beta(x) \varphi_j'(x) \hat{\varphi}_i(x) \, dx.$$

Example: convection–diffusion

Using matrix notation, the convection-diffusion equation

$$\dot{u}(x,t) + \beta(x) \cdot \nabla u(x,t) - \nabla \cdot (\epsilon(x) \nabla u(x)) = f(x),$$

can be written in the form

$$M\dot{\xi}(t) + B\xi(t) + S\xi(t) = b.$$

This is an ODE for the degrees of freedom $\xi(t)$.

General bilinear form $a(\cdot,\cdot)$

In general the matrix A_h, representing a bilinear form

$$a(u,v) = (A(u),v),$$

is given by

$$(A_h)_{ij} = a(\varphi_j, \hat{\varphi}_i).$$

Assembling the matrices

Computing $(A_h)_{ij}$

Note that

$$\begin{aligned}(A_h)_{ij} &= a(\varphi_j, \hat{\varphi}_i) = \int_\Omega A(\varphi_j)\hat{\varphi}_i \, dx \\ &= \sum_{K \in \mathcal{T}} \int_K A(\varphi_j)\hat{\varphi}_i \, dx = \sum_{K \in \mathcal{T}} a(\varphi_j, \hat{\varphi}_i)_K.\end{aligned}$$

Iterate over all elements K and for each element K compute the contributions to all $(A_h)_{ij}$, for which φ_j and $\hat{\varphi}_i$ are supported within K.

Assembling A_h

for all elements $K \in \mathcal{T}$

for all test functions $\hat{\varphi}_i$ on K

for all trial functions φ_j on K

1. Compute $I = a(\varphi_j, \hat{\varphi}_i)_K$
2. Add I to $(A_h)_{ij}$

end

end

end

Assembling A_h in Puffin

...

Assembling b

for all elements $K \in \mathcal{T}$

for all test functions $\hat{\varphi}_i$ on K

1. Compute $I = (f, \hat{\varphi}_i)_K$
2. Add I to b_i

end

end

Assembling b in Puffin

...

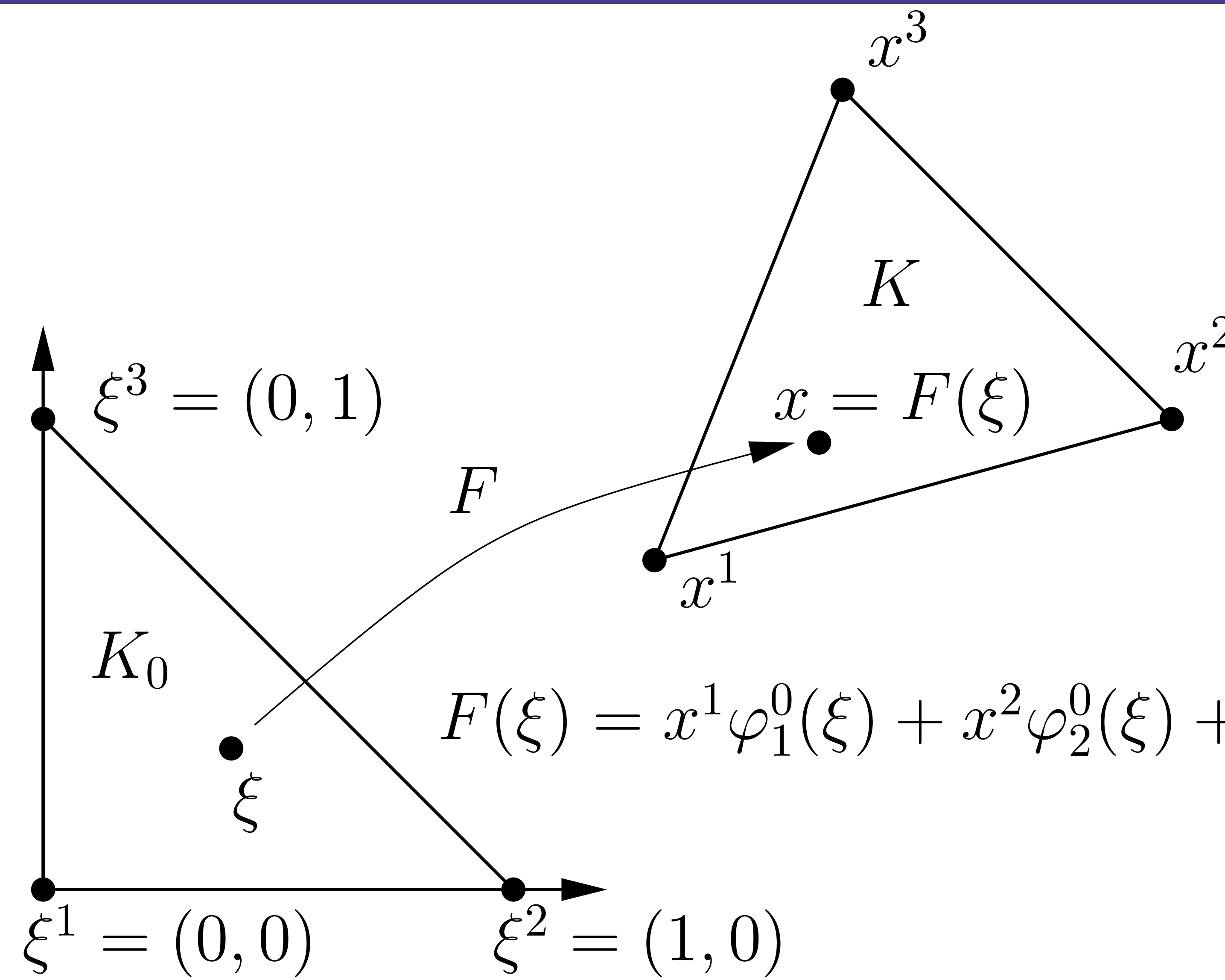
Mapping from a reference element

Mapping

- One set of basis functions on a reference element
- Map the integral from the simple to the real element
- The most common mapping is the iso-parametric mapping

$$x(\xi) = \sum_{i=1}^{n} \varphi_i(\xi) x_i,$$

The mapping $F : K_0 \to K$



$$F(\xi) = x^1\varphi_1^0(\xi) + x^2\varphi_2^0(\xi) + x^3\varphi_3^0(\xi)$$

Some basic calculus

Let $v = v(x)$ be a function defined on a domain Ω and let

$$F : \Omega_0 \to \Omega$$

be a (differentiable) mapping from a domain Ω_0 to Ω. We then have $x = F(\xi)$ and

$$\begin{aligned}\int_\Omega v(x)\, dx &= \int_{\Omega_0} v(F(\xi))\, |\det \partial F_i/\partial \xi_j|\, d\xi \\ &= \int_{\Omega_0} v(F(\xi))\, |\det \partial x/\partial \xi|\, d\xi.\end{aligned}$$

Affine mapping

When the mapping is affine, the determinant is constant:

$$
\begin{aligned}
&\int_K \varphi_j(x)\hat{\varphi}_i(x)\; dx \\
= \;&\int_{K_0} \varphi_j(F(\xi))\hat{\varphi}_i(F(\xi))\;|\det \partial x/\partial \xi|\; d\xi \\
= \;&|\det \partial x/\partial \xi| \int_{K_0} \varphi_j^0(\xi)\hat{\varphi}_i^0(\xi)\; d\xi
\end{aligned}
$$

Transformation of derivatives

To compute derivatives of the basis functions, we use the transformation (chain rule)

$$\nabla_\xi = \left(\frac{\partial x}{\partial \xi} \right)^\top \nabla_x,$$

or

$$\nabla_x = \left(\frac{\partial x}{\partial \xi} \right)^{-\top} \nabla_\xi.$$

The stiffness matrix

For the computation of the stiffness matrix, this means that we have

$$\begin{aligned}
&\int_K \epsilon(x)\nabla\varphi_j(x)\cdot\nabla\hat{\varphi}_i(x)\,dx \\
= &\int_{K_0}\epsilon_0(\xi)\left[(\partial x/\partial\xi)^{-\top}\nabla_\xi\varphi_j^0(\xi)\right]\cdot\left[(\partial x/\partial\xi)^{-\top}\nabla_\xi\hat{\varphi}_i^0(\xi)\right]\cdots \\
&\quad\cdots|\det(\partial x/\partial\xi)|\,d\xi.
\end{aligned}$$

Note that we have used the short notation

$\nabla = \nabla_x$.

Computing integrals on K_0

- The integrals on K_0 can be computed exactly or by quadrature.
- In some cases quadrature is the only option.

Standard form:

$$\int_{K_0} v(\xi)\, d\xi \approx |K_0| \sum_{i=1}^{n} w_i v(\xi^i)$$

where $\{w_i\}_{i=1}^n$ are quadrature weights and $\{\xi^i\}_{i=1}^n$ are quadrature points in K_0.

Assembling in Puffin

...

Solving nonlinear problems

Nonlinear problems

If the problem is nonlinear, for example,

$$-\nabla \cdot (a(u)\ \nabla u) = f,$$

we rewrite the problem as

$$-\nabla \cdot (a(u_{old})\ \nabla u_{new}) = f.$$

As before, we obtain a linear system $A_h \xi = b$, but now

$$A_h = A_h(u_{old}) = A_h(u) = A_h(\xi),$$

i.e. $A_h(\xi)\xi = f$.

Fixed-point iteration

To solve a nonlinear problem $F(\xi) = 0$, we rewrite the problem in fixed-point form

$$\xi = g(\xi),$$

and apply fixed-point iteration as follows:

$$\begin{aligned} \xi^0 &= \text{a clever guess} \\ \xi^1 &= g(\xi^0) \\ \xi^2 &= g(\xi^1) \\ \ldots \end{aligned}$$

Fixed-point iteration

According to the contraction-mapping theorem, fixed-point iteration converges if

$$L_g < 1,$$

where L_g is a Lipschitz-constant of g:

$$\|g(\xi) - g(\eta)\| \leq L_g \|\xi - \eta\|.$$

Basic algorithm

$\xi = \xi^0$

$d = 2 \cdot \mathrm{tol}$

while $d > \mathrm{tol}$

$\quad \xi_{\mathrm{new}} = g(\xi)$

$\quad d = \|\xi_{\mathrm{new}} - \xi\|$

$\quad \xi = \xi_{\mathrm{new}}$

end

Newton's method

Newton's method is a special type of fixed-point iteration for $F(\xi) = 0$, where we take

$$g(\xi) = \xi - (\partial F / \partial \xi)^{-1} F(\xi).$$

Usually converges faster than basic fixed-point iteration, but requires more work to implement.

Time-stepping

A shortcut

Replace $\dot{\xi}$ by $(\xi(t_n) - \xi(t_{n-1}))/k_n$, and replace ξ by

- $\xi(t_{n-1})$: forward / explicit Euler
- $\xi(t_n)$: backward / implicit Euler
- $(\xi(t_{n-1}) + \xi(t_n))/2$: Crank-Nicolson / $\mathrm{cG}(1)$

Example: backward Euler

Discretizing the heat equation $\dot{u} - \Delta u = f$ in space, we have

$$M\dot{\xi} + S\xi = b.$$

Using the implicit Euler method for time-stepping, we obtain

$$M(\xi(t_n) - \xi(t_{n-1}))/k_n + S\xi(t_n) = b(t_n),$$

or

$$(M + k_n S)\xi(t_n) = M\xi(t_{n-1}) + k_n b(t_n).$$

Basic algorithm

$t_0 = 0$

$n = 1$

while $t < T$

$\quad t_n = t_{n-1} + k$

$\quad \xi^n = \ldots$

$\quad n = n + 1$

end