PDE Project Course

2. Implementation of the finite element method

Johan Jansson

johanjan@math.chalmers.se

Division of Computational Technology

Lecture plan

- Matrix notation
- Assembling the matrices
- Mapping from a reference element
- Solving nonlinear problems
- Time-stepping
- General solution strategy

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, we obtain

$$S\xi = b,$$

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

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

and the vector b_h representing a linear form

$$L(v) = (f,v),$$

is given by

$$(b_h)_i = L(\hat{\varphi}_i).$$

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 b

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

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

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

end

end

Mapping from a reference element

Isoparametric mapping

- We want to compute basis functions and integrals on a *r*eference element K_0
- Most common mapping is isoparametric mapping (use the basis functions also to define the geometry):

$$x(X) = F(X) = \sum_{i=1}^{n} \phi_i(X) x_i$$

- Linear basis functions $\Rightarrow$ Affine mapping: $x(X) = F(X) = BX + b$

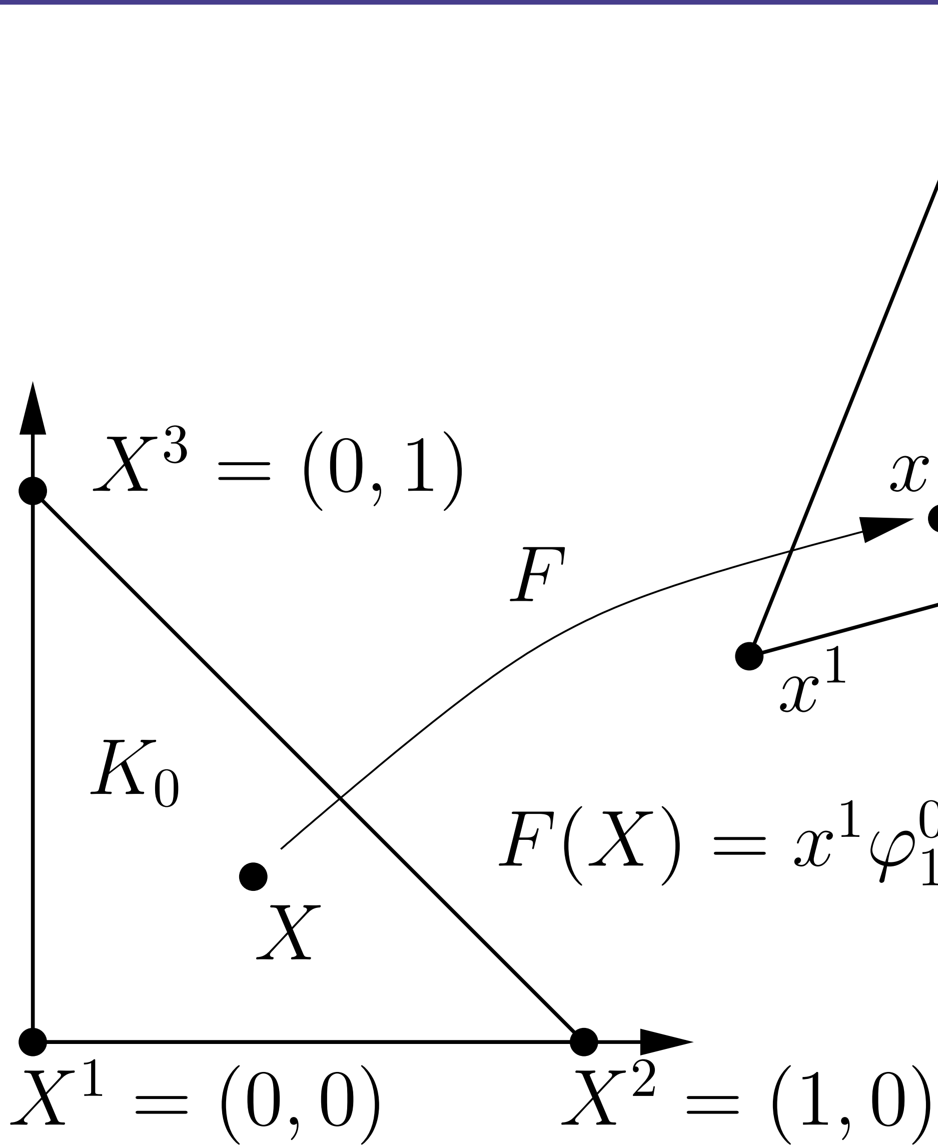
Piola mapping

- Piola mapping:

$$x(X) = P(X) = \frac{1}{detF'} F'(\psi \circ F^{-1})$$

- Affine mapping: $F(X) \Rightarrow F'$ constant (B)

The mapping $F : K_0 \to K$



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

Some basic calculus

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

$$F : \Omega_0 \rightarrow \Omega$$

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

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

Affine mapping

When the mapping is affine, the determinant is constant:

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

Transformation of derivatives

To compute derivatives, we use the transformation

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

or

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

The stiffness matrix

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

$$\int_K \epsilon(x)\nabla\varphi_j(x)\cdot\nabla\hat{\varphi}_i(x)\,dx$$

$$= \int_{K_0}\epsilon_0(X)\left[(\partial x/\partial X)^{-\top}\nabla_X\varphi_j^0(X)\right]\cdot\left[(\partial x/\partial X)^{-\top}\nabla_X\hat{\varphi}_i^0(X)\right]\cdot\cdots$$

$$\cdots|\det(\partial x/\partial X)|\,dX.$$

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:

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

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

Solving nonlinear problems

Nonlinear problems

If the problem is nonlinear, for example,

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

we rewrite the problem as

$$-\nabla \cdot (|\nabla \tilde{u}| \, \nabla u) = f.$$

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

$$A_h = A_h(\tilde{u}) = A_h(u) = A_h(\xi),$$

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

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

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

General solution strategy

We only allow PDEs in the form:

$$\dot{u} = f(u).$$

$$M\dot{\xi} + S\xi = b \Rightarrow \dot{\xi} = f(\xi) = M^{-1}(b - S\xi)$$

Then we can give this f to a general ODE solver which can do time adaptivity, fixed-point iteration and Newton's method where necessary.