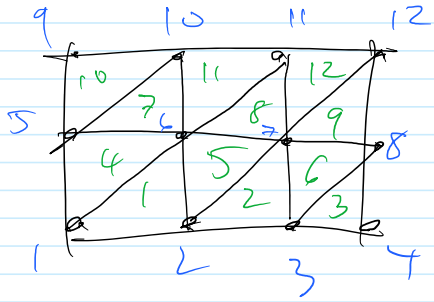
2D poisson 3

torsdag 14. september 2017 10.47

1. Recap: to solve $\begin{cases} -\nabla^2 u = f & \text{on } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$

we

1. Triangulate domain



p

Decide on a degree of Lagrange basis function: may require internal nodes if $p > 1$.

2. Solve $Au = f$,

$$A_{ij} = \int \nabla\varphi_i \cdot \nabla\varphi_j, \quad f_i = \int f\varphi_i$$

where $\{\varphi_i\}$ is basis.

→ Construction (assembly) runs over elements:

$$A_{ij} = \sum_k \int_{K_k} \nabla\varphi_i \cdot \nabla\varphi_j,$$

where K_k are the various elements etc.

→ Use the formulation

$$\int \nabla\varphi_i \cdot \nabla\varphi_j = |T| \int c(\lambda)^T c(\lambda)$$

$$\int_{K_k} \nabla\varphi_i \cdot \nabla\varphi_j = |J| \int_{\hat{K}} G(\lambda)^T G(\lambda)$$

where $G(\lambda)$ is the $k \times (d+1)$ matrix that solves

no. of nodes in an element

$$J^T \cdot G = [I, -\mathbb{1}] \cdot \underbrace{\nabla_\lambda \hat{\varphi}(\lambda)}$$

$$= [\underline{x}_1 - \underline{x}_{d+1}, \dots \underline{x}_d - \underline{x}_{d+1}]$$

matrix w/ i^{th} col. the gradient wrt λ of basis functions on reference element written in barycentric coords (e.g. identity for linear elements etc.)

eg $\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$ in 2d

$\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$ in 3d

$\Rightarrow$ this block $\int_{K_k} \nabla\varphi_i \cdot \nabla\varphi_j$ must be added to the correct places within A.

Outlook: i) Practicalities of assembly

ii) Boundary conditions

iii) On to error estimates.

2. Our code will look something like

```
Δ = GetTriangulation
Phi = [eye(d), -ones(d,1)];
GradLambda = @(λ) Phi · (∇̂_λ φ̂_i(λ))
A = zeros(M)
f = zeros(M,1)
for i in elements(Δ)
    x_1, ..., x_{d+1} = vertices(Δ_i)
    v_1, ..., v_{d+1} = vertex_numbers(Δ_i)
    J = [x_1 - x_{d+1}, ..., x_d - x_{d+1}]
    Jacdet = abs(det(J))
    linear elements:
    G = J \ Phi;
    A_k = (Jacdet * G' * G) / d!;
    nonlinear elements:
    G = @(λ) J \ GradLambda(λ);
    A_k = integrate(G'·G) * Jacdet / d!;
    % Now for load:
    x = @(λ) x_1 λ_1 + ... + x_{d+1} λ_{d+1}
    f_k = integrate(f(x(λ)) · φ(λ)) * Jacdet / d!
    % write into matrix:
```

if using quadratic or higher elements

appropriate matrix.

(or implement w/ LU factoring of J.)

your own quadrature routine on $\hat{K}$.

column vector of $\varphi_i(\lambda)$

$A(v_1, \dots, v_{d+1} ; v_1, \dots, v_{d+1}) = A(\dots, \dots) + A_k$

$f_k(v_1 ; \dots ; v_{d+1}) = f(\dots) + f_k$

end

$[A, f] = \text{Remove\_boundary}(A, f)$

$u = A \backslash f$

a) Triangulation class: want to store Δ so that

i) we can look up an element Δ_i

ii) for each element Δ_i, we want

→ vertex coords: $\underline{x}_1, \dots, \underline{x}_{d+1}$ (Δ_i)

→ vertex numbers: $v_1, \dots, v_{d+1}$ (Δ_i)

(so we know where to insert the blocks of A and f from Δ_i).

→ Matlab implements a triangulation class that allows us to do this (see forthcoming project)

→ Will also need info on which vertices etc. are on the edge

→ Possibly location of interior nodes

---

3. Boundary conditions:

→ consider mixed, non-homogeneous problem:

$$\begin{cases} -\nabla^2 u = f & \text{in } \Omega \\ u = g & \text{on } \Gamma_D \\ \dfrac{\partial u}{\partial n} = \phi & \text{on } \Gamma_N \end{cases} \quad \text{here } \partial\Omega = \Gamma_D \cup \Gamma_N$$

(NB: for Neumann problems, require a compatibility condition:

$$-\int_{\partial\Omega} \phi \, dy = \int_\Omega f \, d\Omega \,)$$

→ First we construct a lifting of boundary data as per 1d case

i.e. $R_g \in H^1(\Omega)$: $R_g\big|_{\Gamma_D} = g$

we will then solve for $\mathring{u} = u - R_g$ (with homogeneous condition).

→ let us see how to define the weak form: (homogeneous Dirichlet case)

$$\underbrace{-\int_\Omega v \nabla^2 u}_{\text{apply Green's identity:}} = \int_\Omega f$$

$$= \int_\Omega \nabla v \cdot \nabla u - \oint_{\partial\Omega} v \frac{\partial u}{\partial n}$$

$$\int_\Omega \nabla v \cdot \nabla u = \oint_{\Gamma_N} v\phi + \int_\Omega f v$$

$$\int_\Omega \nabla v \cdot \nabla u = \oint_{\Gamma_N} v\phi + \int_\Omega fv$$

(have used that $\oint_{\partial\Omega} v \frac{\partial u}{\partial n}$ vanishes on Γ_D, and that $\frac{\partial u}{\partial n} = \phi$ on Γ_N).

→ Now return to the inhomogeneous case:

$u = \mathring{u} + R_g$, hence $\nabla u = \nabla \mathring{u} + \nabla R_g$

i.e. solve for $\mathring{u} \in H^1_{\Gamma_D}(\Omega)$

$$\int_\Omega \nabla v \cdot \nabla \mathring{u} = \oint_{\Gamma_N} v\phi + \int_\Omega fv - \int_\Omega \nabla R_g \cdot \nabla v$$

for all $v \in H^1_{\Gamma_D}(\Omega)$

Galerkin case: consider $V_h \subset H^1(\Omega)$

→ first must construct R_g: as we use a nodal basis w/ $\varphi_i(N_j) = \delta_{ij}$, this is straightforward: take (where N_j is a node)

$$R_g = \sum_{i: N_i \in \Gamma_D} g(N_i) \cdot \varphi_i$$

→ then setting $\mathring{u} = \sum \mathring{u}_i \varphi_i(\underline{x})$, $v = \sum v_j \varphi_j(\underline{x})$ results as before in a linear system

$$\boxed{A\mathring{u} = f - Bg}$$

$$\boxed{A\mathring{u} = f - Bg}$$

A: $M \times M$ matrix

B: $M \times N$ matrix

g: vector of values of g on Dirichlet boundary nodes (N no. of such nodes)

where $A = \int \nabla\varphi_i \cdot \nabla\varphi_j$

$$f = \underbrace{\oint_{\Gamma_N} \phi\,\varphi_i}_{\text{compute by Gaussian quadrature}} + \int_\Omega f\varphi_i$$

and $B_{ij} = \int \nabla\varphi_i \cdot \nabla\varphi_j$, where j runs over only nodes on the boundary

then set $\boxed{u = \mathring{u} + \bar{g}}$.

Recall: vector of values of g on boundary.

(have given a hat to make same dimension, introduce zeros at non-boundary nodes).

---

4. Implementation of boundary conditions:

→ in 1d case, this was easy, eg

$$\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times \\ \times & \times \end{pmatrix}$$

$$\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times \end{pmatrix}$$

→ in higher dimensions, more awkward, as the boundary rows/cols are scattered throughout the matrix

- moreover, resizing the matrix can be memory intensive.

→ First, set $f_i = 0$ at (Dirichlet) boundary nodes

Then, two options

i) set $A_{ii} = 1$ and all other $A_{ij} = 0$ (i.e., along row), this forces $u_i = 0$

ii) <u>'Big number'</u>: set $A_{ii} = \frac{1}{\varepsilon}$, $\varepsilon \ll 1$ (i.e. an enormous number)

then $u_i \approx 0$.
