Notes

- Make-up lecture tomorrow 1-2, room 204

Linear basis in 1D

- From last-time, the equation for test function i was:

$$\frac{1}{x_{i+1}-x_i}u_{i+1} - \left(\frac{1}{x_{i+1}-x_i} + \frac{1}{x_i - x_{i-1}}\right)u_i + \frac{1}{x_i - x_{i-1}}u_{i-1} = \int f(x)\phi_i(x)dx$$

- Can match up left-hand-side (matrix) to finite difference approximation
- Right-hand-side is a bit different:

$$f(x_i) \quad vs. \quad \int f(x)\phi_i(x)dx$$

The Mass Matrix

- Assuming f is from the same space, get:

$$\int f\phi_i = \int \sum_{j=1}^{n} f_j\phi_j\ \phi_i = \sum_{j=1}^{n}\left(\int \phi_i\phi_j\right)f_j = \left(Mf\right)_i$$

- M is called the **mass matrix**
  - Obviously symmetric, positive definite
- In piecewise linear element case, tridiagonal

Lumped Mass Matrix

- The fact that M is not diagonal can be inconvenient
  - E.g. if solving a time-dependent PDE, M multiplies the time derivative - so even an "explicit" method requires solving linear systems
  - Can be viewed as a low-pass / smoothing filter of the data, which may not be desired
- Thus often people will "lump" the offdiagonal entries onto the diagonal: **lumped mass matrix** (versus "consistent" mass matrix)
- This makes the connection with finite differences (for piecewise linear elements) perfect

Stiffness matrix

- The matrix A (where derivatives show up) is called the **stiffness matrix**
  - "Stiffness" and "mass" come from original FEM application, simulating solid mechanics

Assembling matrices

- Entry of the stiffness matrix:

$$A_{ij} = \int_{\Omega} \nabla\phi_i \bullet \nabla\phi_j = \sum_{e}\int_{e} \nabla\phi_i \bullet \nabla\phi_j$$

- Here we sum over "elements" e where basis functions i and j are nonzero
  - Usually an "element" is a chunk of the mesh, e.g. a triangle
- Can loop over elements, adding contribution to A for each
  - Each contribution is a small submatrix: the **local** (or **element**) stiffness matrix
  - A is the **global** stiffness matrix
  - Process is called **assembly**

Quadrature

- Integrals may be done analytically for simple elements
  - E.g. piecewise linear
- But in general it's fairly daunting – or impossible (e.g. curved elements)
- Can tolerate some small error: numerically estimate integrals = **quadrature**
- Basic idea: sample integrand at **quadrature points**, use a weighted sum
  - Accuracy: make sure it's exact for polynomials up to a certain degree

FEM convergence

- Let the exact solution be u^*

  and for a given finite element space V let the numerical solution be $u \in V$
- Galerkin FEM for Poisson is equivalent to:

$$u = \arg\min_{u \in V} \int_\Omega \left\| \nabla \left(u - u^* \right) \right\|^2$$

(closest in a least-squares, semi-norm way)

FEM convergence cont'd

- Don't usually care about this semi-norm: want to know error in a regular norm. With some work, can show equivalence…
- The theory eventually concludes: for a well-posed problem, accuracy of FEM determined by how close function space V can approximate solution
- If e.g. solution is smooth, can approximate well with piecewise polynomials…

Some more element types

- Polynomials on triangles etc.
- Polynomials on squares etc.
- More exotic:
  - Add gradients to data
  - "Non-conforming" elements
  - Singularity-matching elements
  - Mesh-free elements

Mesh generation

- Still left with problem: how to generate the underlying mesh (for usual elements)
- More or less solved in 2D, still heavily researched in 3D
- Triangles/tetrahedra much easier than quads/hexahedra
- We'll look at one particular class of methods for producing triangle meshes: **Delaunay triangulation**

Meshing goals

- Robust: doesn't fail on reasonable geometry
- Efficient: as few triangles as possible
  - Easy to refine later if needed
- High quality: triangles should be "well-shaped"
  - Extreme triangles make for poor performance of FEM - particularly large obtuse angles