A practical view of the classical FEM

(and some FEM software)

Ed Bueler

Dept of Mathematics and Statistics
University of Alaska Fairbanks

Outline

nonlinear Poisson problems with general boundary conditions

assembly of FEM system

solving the equations

modern software: Gmsh, Firedrake, PETSc

fluid flow example

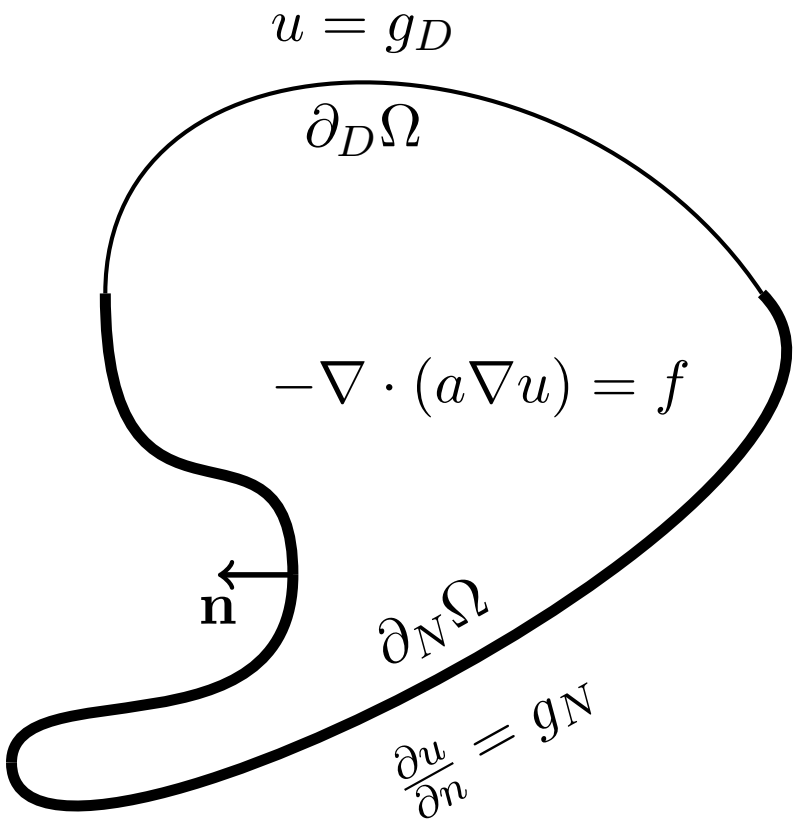
general, nonlinear Poisson problem

- $\Omega \subset \mathbb{R}^2$
- recall our usual problem: $\triangle u = f$, $u = 0$ on $\partial\Omega$
- these slides cover a nonlinear version with general boundary conditions:

$$\begin{aligned} -\nabla \cdot (a(u)\nabla u) &= f(u) && \text{on } \Omega, \\ u &= g_D && \text{on } \partial_D\Omega, \\ \frac{\partial u}{\partial n} &= g_N && \text{on } \partial_N\Omega. \end{aligned}$$

- $a(u,x,y)$, $f(u,x,y)$, $g_N(x,y)$, $g_D(x,y)$ are given (data)
- nonlinear because $a = a(u)$ and/or $f = f(u)$ is allowed

general, nonlinear Poisson problem



weak form

- $u \in H^1_g(\Omega) = \{u \in H^1(\Omega) \,:\, u = g_D \text{ on } \partial_D\Omega\}$
  - Dirichlet data g_D is used in defining $H^1_g(\Omega)$
- test functions: $H^1_0(\Omega) = \{v \in H^1(\Omega) \,:\, v = 0 \text{ on } \partial_D\Omega\}$
- for $v \in H^1_0(\Omega)$, multiply PDE by v and integrate by parts:

$$\int_\Omega a(u)\nabla u \cdot \nabla v - \int_{\partial\Omega} \frac{\partial u}{\partial n} v = \int_\Omega f(u)v$$

- apply boundary conditions:

$$\int_\Omega a(u)\nabla u \cdot \nabla v = \int_\Omega f(u)v + \int_{\partial_N\Omega} g_N v \qquad \text{(WF)}$$

  - Neumann data g_N appears here

examples

some examples of $-\nabla \cdot (a(u)\nabla u) = f(u)$:

- Poisson equation:

$$-\triangle u = f(x, y), \quad u = 0 \text{ on } \partial\Omega$$

- Liouville-Bratu equation:

$$-\triangle u = \lambda e^u$$

  - critical λ for well-posedness

- porous-medium equation:

$$-\nabla \cdot \Big((u^{m-1} + \epsilon)\nabla u \Big) = f$$

  - regularized with $\epsilon > 0$ for uniform ellipticity
  - $\epsilon = 0$: degeneracies within domain

Outline

nonlinear Poisson problems with general boundary conditions

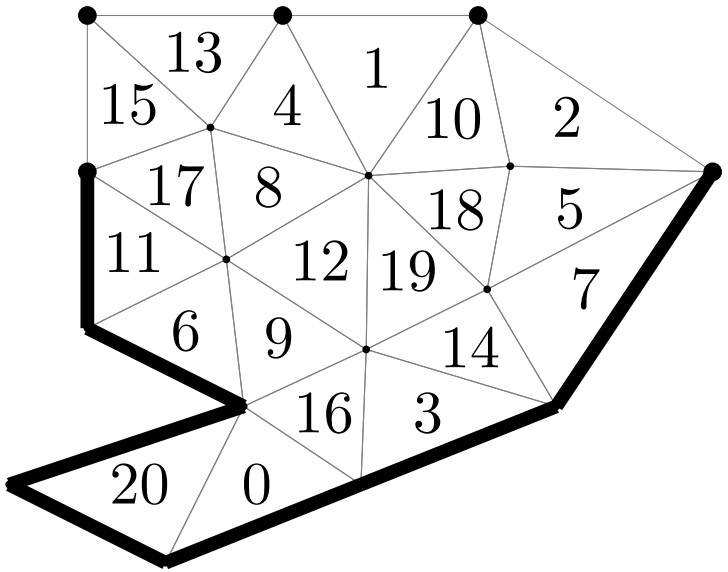
assembly of FEM system

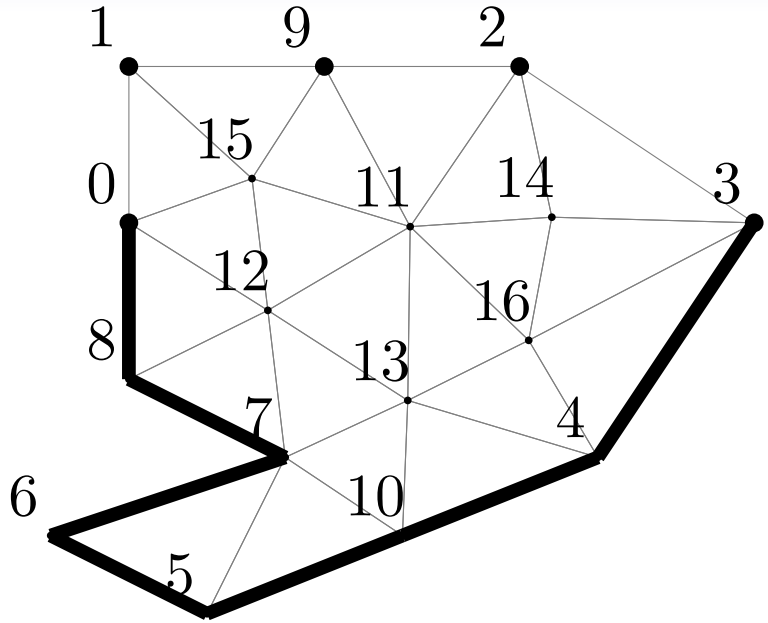
solving the equations

modern software: Gmsh, Firedrake, PETSc

fluid flow example

meshes and numbering

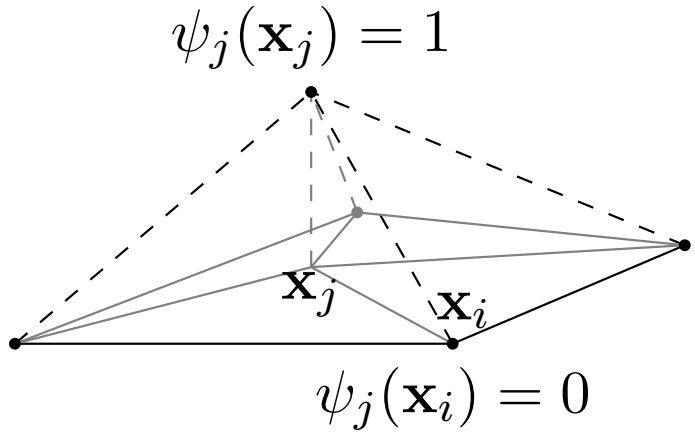




- triangulation $\mathcal{T}_h$ of a polygonal domain Ω
- numbering:
  - K elements ($= 21$ above; left)
  - N nodes ($= 17$ above; right)
  - P segments in Neumann boundary ($= 7$ above; bold)
  - n_D nodes in Dirichlet boundary ($= 5$ above)
- to fully describe mesh, also need node locations $\mathbf{x}_i = (x_i, y_i)$

FEM spaces

- hat functions satisfy:
  - $\psi_j(\mathbf{x})$ continuous
  - $\psi_j \in P_1$ on each element
  - $\psi_j(\mathbf{x}_i) = \delta_{ij}$



- use hat functions to approximate $g_D(\mathbf{x})$:

$$\hat{g}_D(\mathbf{x}) = \sum_{l=0}^{n_D-1} g_D(\mathbf{x}_{j_l})\psi_{j_l}(\mathbf{x}) \quad \in C(\overline{\Omega})$$

- use hat functions to define test function space:

$$S_0^h = \operatorname{span}\langle \psi_j \,:\, \mathbf{x}_j \notin \partial_D\Omega \rangle \subset H_0^1(\Omega)$$

- . . . and trial function space:

$$S_g^h = \left\{ \hat{g}_D + w \,:\, w \in S_0^h \right\} \overset{?}{\subset} H_g^1(\Omega)$$

FEM spaces

- expand the FEM solution u^h:

$$u^h(\mathbf{x}) = \hat{g}_D(\mathbf{x}) + \sum_{\mathbf{x}_j \notin \partial_D \Omega} u_j \, \psi_j(\mathbf{x})$$

- discrete weak form: for all i such that $\mathbf{x}_i \notin \partial_D \Omega$,

$$\int_\Omega a(u^h) \nabla u^h \cdot \nabla \psi_i = \int_\Omega f(u^h) \psi_i + \int_{\partial_N \Omega} g_N \psi_i \qquad \text{(DWF)}$$

- $\dim(S_0^h) = \dim(S_g^h) = N - n_D$
- however, it is easier to also make Dirichlet boundary nodes into degrees of freedom
  - we assemble a nonlinear system of N equations in N unknowns

element residuals

- for each element $\triangle_k$ and ψ_i, define an *element residual*:

$$F_i^k(\mathbf{u}) = \int_{\triangle_k} a(u^h)\nabla u^h \cdot \nabla\psi_i - f(u^h)\psi_i$$

- for each edge s_ν in the Neumann boundary we define

$$\varphi_i^\nu = \int_{s_\nu} g_N\psi_i$$

- (DWF) becomes:

$$F_i(\mathbf{u}) = \sum_{k=0}^{K-1} F_i^k(\mathbf{u}) - \sum_{\nu=0}^{P-1} \varphi_i^\nu = 0 \qquad \text{for } \mathbf{x}_i \notin \partial_D\Omega$$

- for Dirichlet boundary nodes have easy equation:

$$F_i(\mathbf{u}) = u_i - g_D(\mathbf{x}_i) \qquad \text{for } \mathbf{x}_i \in \partial_D\Omega$$

assembled FEM system of equations

- recall: N is number of nodes in mesh
- P^1 FEM generates an algebraic system of N equations in N unknowns:

$$\mathbf{F}(\mathbf{u}) = 0 \qquad (*)$$

  - one equation $F_i(\mathbf{u}) = 0$ per node
  - one unknown u_i per node
  - nonlinear system if $\partial a/\partial u \neq 0$ and/or $\partial f/\partial u \neq 0$
  - sparse because support of a hat function ψ_i only overlaps with a few triangles $\triangle_k$, and thus only a few nodal values $\mathbf{u} = \{u_j\}$ enter into a given $F_i^k(\mathbf{u})$
  - $J_{\mathbf{F}} = (\partial \mathbf{F}_i/\partial u_j)$ is the *Jacobian*, a sparse matrix

Outline

nonlinear Poisson problems with general boundary conditions

assembly of FEM system

solving the equations

modern software: Gmsh, Firedrake, PETSc

fluid flow example

nonlinear residual

- the result of FEM assembly is a nonlinear system:

$$\mathbf{F}(\mathbf{u}) = 0 \qquad (*)$$

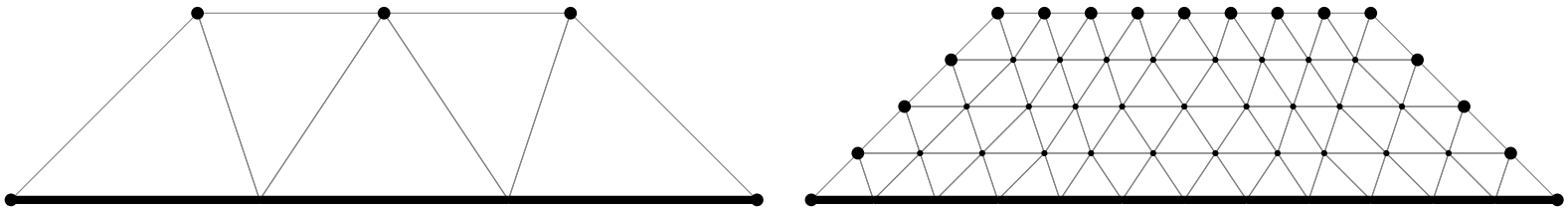
- the function $\mathbf{F} : \mathbb{R}^N \to \mathbb{R}^N$ is called the *residual*
  - continuous and differentiable if a, f are
- propose to solve $(*)$ by Newton method:

$$J_{\mathbf{F}}(\mathbf{u}_k)\mathbf{p} = -\mathbf{F}(\mathbf{u}_k)$$
$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha\mathbf{p}$$

  - find α using line search based on *merit function* $\Phi(\mathbf{u}) = \frac{1}{2}\|\mathbf{F}(\mathbf{u})\|^2$
  - converges in one step if $a = a(x, y)$ and $f = f(x, y)$ [linear]
  - converges quadratically if a, f smooth

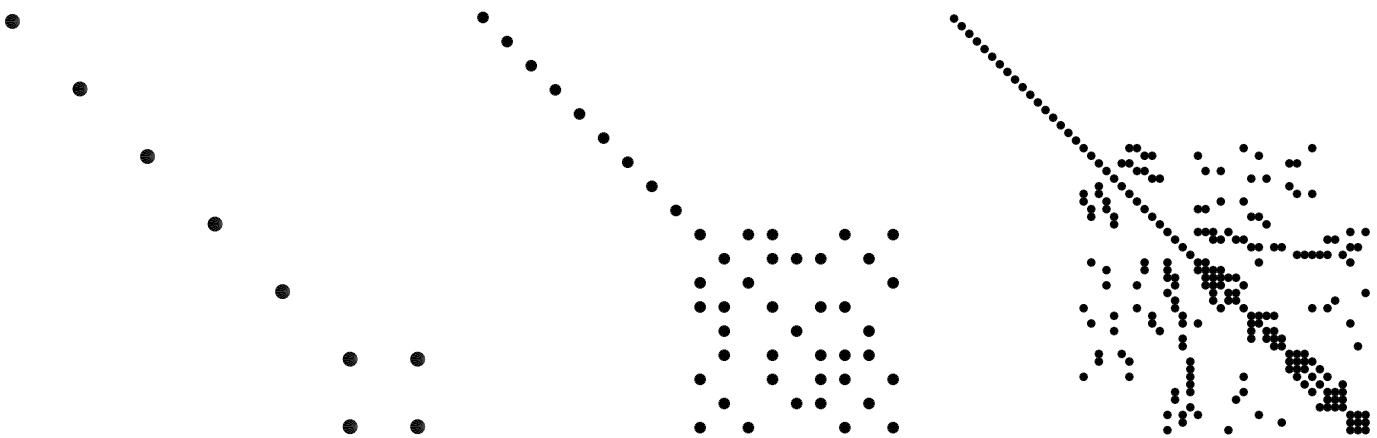
the Jacobian is the FEM stiffness matrix

- $J_{\mathbf{F}}(\mathbf{u})$ is the FEM "stiffness matrix"
- example: meshes with $N = 7, 18, 55$ nodes



  - generated by uniform refinement of $N = 7$ mesh
  - $N = 18$ mesh not shown

- sparsity patterns for corresponding $J_{\mathbf{F}}(\mathbf{u})$:



the Newton step is a sparse linear system

- $J_{\mathbf{F}}(\mathbf{u}_k)\mathbf{p} = -\mathbf{F}(\mathbf{u}_k)$ is a sparse linear system
  - solve it for the Newton step $\mathbf{p}$
  - actually take the step after line search: $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha\mathbf{p}$
- so, how to solve?:

$$A\mathbf{p} = \mathbf{b}$$

  - always: $A = J_{\mathbf{F}}(\mathbf{u}_k)$ is sparse $N \times N$ matrix
  - if PDE is well-posed: $A = J_{\mathbf{F}}(\mathbf{u}_k)$ is invertible
  - if uniformly elliptic and assembled as described: $A = J_{\mathbf{F}}(\mathbf{u}_k)$ is symmetric positive-definite

solving sparse linear systems

- the Newton step is $A\mathbf{p} = \mathbf{b}$ where $A = J_{\mathbf{F}}(\mathbf{u}_k)$
- $O(N^3)$ dense algorithms are possible:
  - Gaussian elimination (LU decomposition)
  - Cholesky decomposition
- . . . but they struggle for big N
  - variable reordering (minimum degree, nested dissection) helps a lot . . . but is still slow
- modern expectation: use Krylov methods
  - $\mathcal{K}_m(A, \mathbf{b}) = \left\langle \mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^{m-1}\mathbf{b} \right\rangle$ is *Krylov space*
  - CG and GMRES are best-known Krylov methods

solving sparse linear systems

- actually one uses *preconditioned* Krylov methods
- a *preconditioner* is an M which is spectrally-equivalent to A and such that $M\mathbf{p} = \mathbf{c}$ can be rapidly solved
- for $A\mathbf{p} = \mathbf{b}$ the preconditioned system is $M^{-1}A\mathbf{p} = M^{-1}\mathbf{b}$
  - $\mathcal{K}_m(M^{-1}A, M^{-1}\mathbf{b})$ contains good approximation of $\mathbf{p}$
- preconditioners include:
  - diagonal of A — *Jacobi*
  - incomplete factorization — *ILU,IC*
  - approximate block inverse — *ASM*
  - application of multigrid cycles — *GMG,AMG*

Outline

nonlinear Poisson problems with general boundary conditions

assembly of FEM system

solving the equations

modern software: Gmsh, Firedrake, PETSc

fluid flow example

modern FEM software

- slogan: modern FEM software should make FEM easy
- application packages:
  - COMSOL Multiphysics ← grew out of a Matlab toolbox
  - ABACUS
  - Autodesk Simulation
  - *I don't actually know much about these packages* . . .
- open source libraries:
  - deal.II [C++]
  - libmesh [C++]
  - Elmer [Fortran90]
  - FEniCS [Python]
  - Firedrake [Python]

tool stacks

- one needs 4 levels of tools:

mesh generation
↓
FEM assembly[a]
↓
solver library[b]
↓
visualization

- commercial packages may partially hide this structure, but libraries always allow separate control of these levels
- notes:
  - a includes weak form declaration/implementation, element definitions, and actual assembly
  - b includes both nonlinear solvers (e.g. Newton method) and linear solvers (e.g. CG and preconditioners too)

tool stacks I have actually used

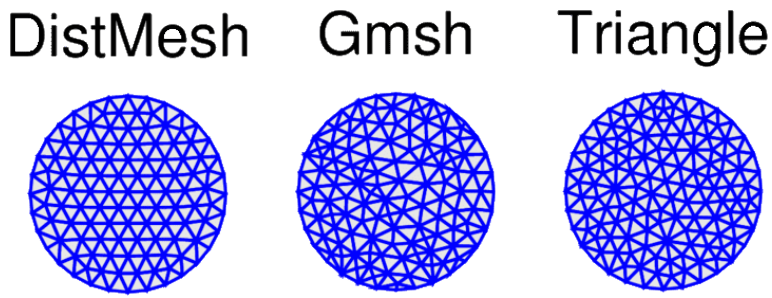
- stacks I have actually used:

	1.	2.	3.
mesh generation	Distmesh	Gmsh	Gmsh
↓	↓	↓	↓
FEM assembly	Matlab code*	C code*	Firedrake
↓	↓	↓	↓
solver library	Matlab	PETSc	PETSc
↓	↓	↓	↓
visualization	Matlab	my own tools	Paraview

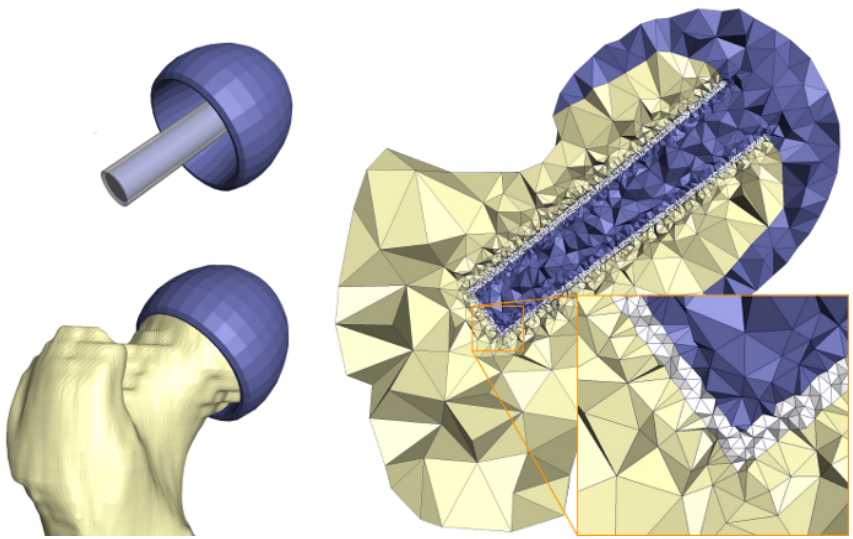
- $*$ = (hand-coded FEM assembly)
- I will show examples using 2. and 3.

mesh generation

- mesh generation varies:



  - 2004 FEM seminar used DistMesh in Matlab
- I'll demonstrate Gmsh (`gmsh.info`), a standard choice for "real" applications . . .



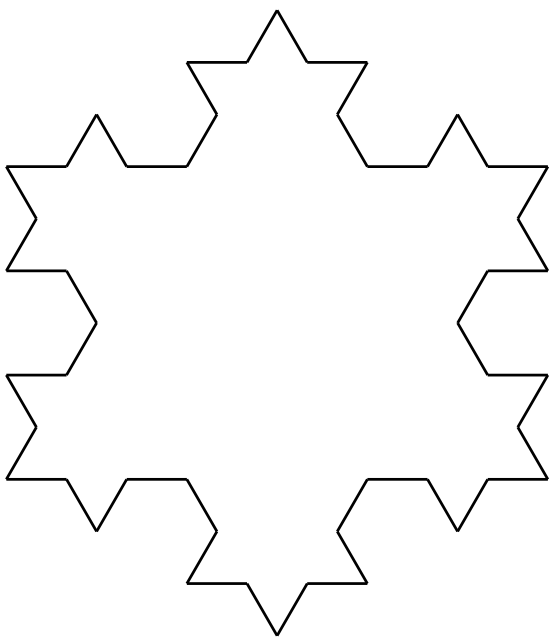
fractal Poisson: mesh generation

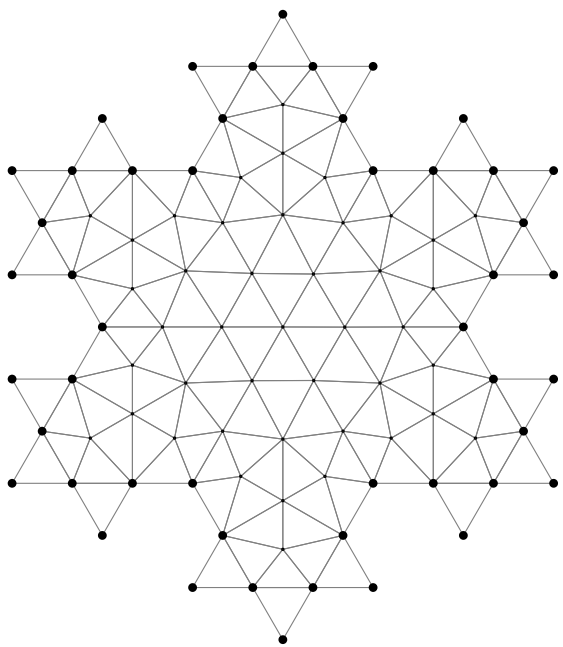
- problem:

$$-\triangle u = 1, \qquad u = 0 \text{ on } \partial\Omega$$

- generate polygon Ω (left figure) and ask Gmsh to triangulate it (right figure):

```
$ ./domain.py -l 2 -o koch2.geo
$ gmsh -2 koch2.geo
```





fractal Poisson: solving the PDE

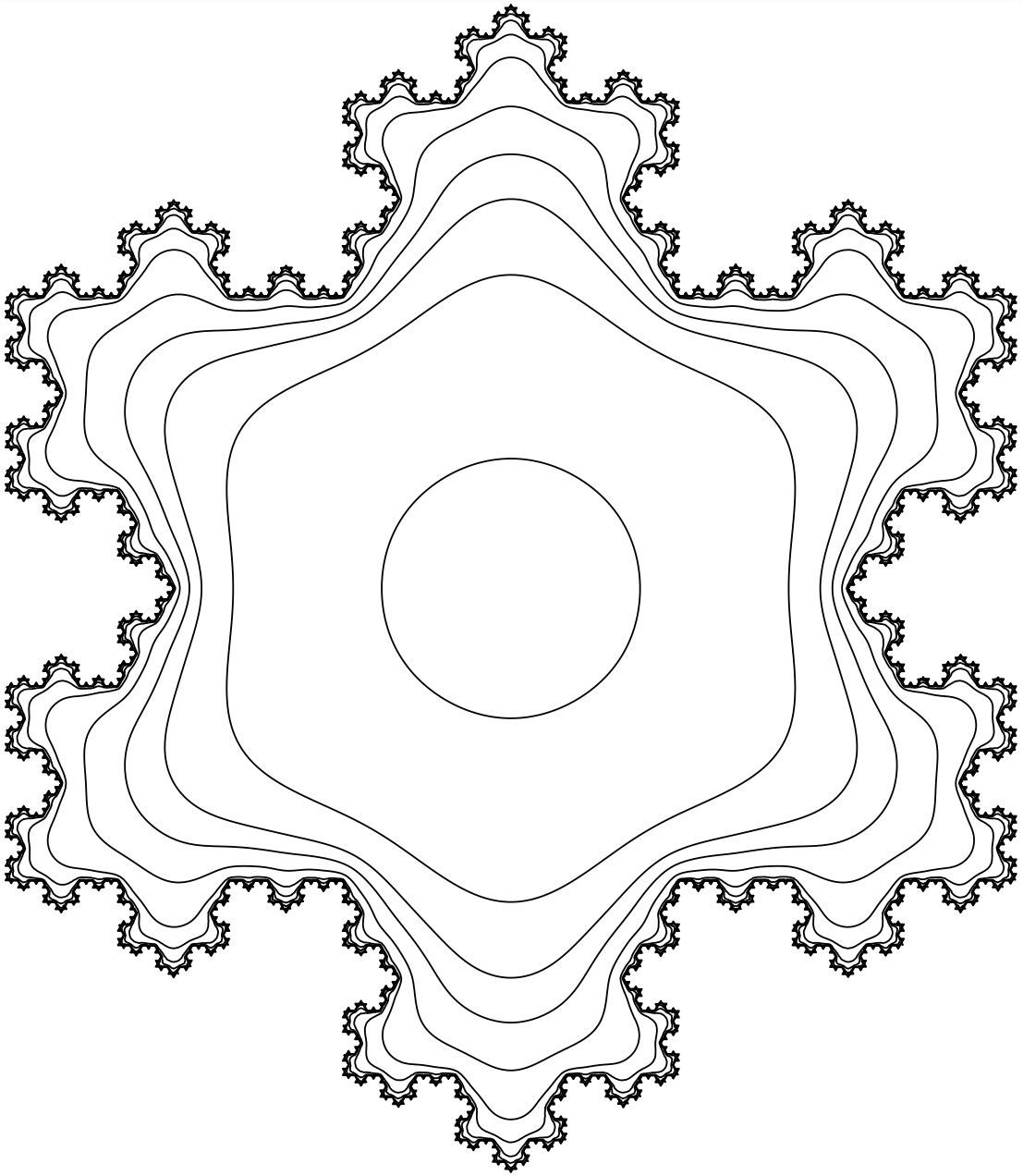
- in fact I wrote three codes:

`domain.py`	generates the boundary polygon
`msh2petsc.py`	converts Gmsh output to binary format
`unfem.c`	assemble the FEM residual using PETSc (`www.mcs.anl.gov/petsc`) data structures, including parallelization; then it calls a PETSc solver

- PDE solution by algebraic multigrid-preconditioned CG on the level-6 fractal domain with $N \approx 480{,}000$ nodes:

```
$ domain.py -l 6 -o koch6.geo
$ gmsh -2 koch6.geo
$ msh2petsc.py koch6
$ unfem -un_mesh koch6 \
    -snes_type ksponly -ksp_type cg -pc_type gamg
```

contours of solution



ugly code

- the result was pretty but the code was ugly
- even using PETSc, `unfem.c` was 900 lines of C code

```
for (k = 0; k < user->mesh->K; k++) {
    ...
    for (l = 0; l < 3; l++) {
        if (abf[en[l]] == 2) {
            ...
            aF[en[l]] = au[en[l]] - user->gD_fcn(xx,yy);
        } else {
            sum = 0.0;
            for (r = 0; r < q.n; r++) {
                psi = chi(l,q.xi[r],q.eta[r]);
                ip  = InnerProd(gradu,gradpsi[l]);
                sum += q.w[r] * ( ... );
            }
            aF[en[l]] += fabs(detJ) * sum;
        }
    }
}
```

better tools

- previous tool stack requires too much hand-coding
- want better tools for FEM assembly stage
- recall the second stack:

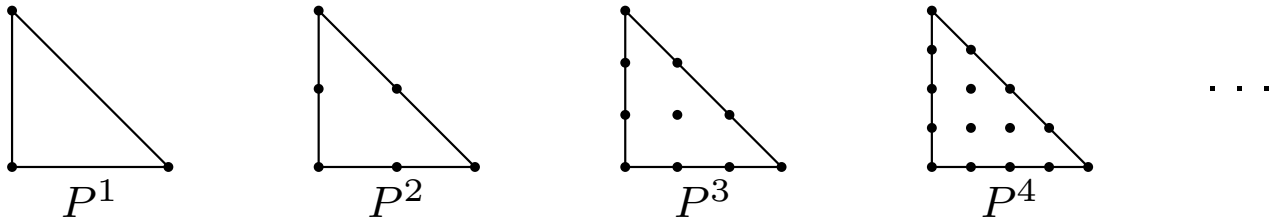
  Gmsh $\rightarrow$ Firedrake $\rightarrow$ PETSc $\rightarrow$ Paraview

- Firedrake: `www.firedrakeproject.org`

  . . . is an automated system for the solution of partial differential equations using the finite element method (FEM) . . . uses sophisticated code generation . . . Expressive specification of any PDE using the Unified Form Language from the FEniCS Project. Sophisticated, programmable solvers through seamless coupling with PETSc.

Unified Form Language (UFL) example

- UFL is an *expressive* language for weak forms
- choose function space $S^h = P^k$ for arbitrary degree k:



- Poisson problem ($-\triangle u = f$ on Ω) has b.c.s $u = g$ on $\partial\Omega$
- weak form:

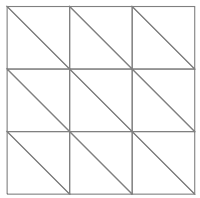
$$\int_\Omega \nabla u \cdot \nabla v - fv = 0$$

- Python UFL Firedrake code for above:

```
Sh = FunctionSpace(mesh,'CG',degree=k)       # Sh = P^k
bc = DirichletBC(Sh, g, bdry_ids)
u, v = Function(Sh), TestFunction(Sh)
F = (dot(grad(u), grad(v)) - f * v) * dx     # weak form
solve(F == 0, u, bcs=bc, ...)
```

solving the whole PDE problem

- for convergence analysis, use unit square $\Omega = (0,1)^2$ and consider regular triangulations only



- this time I wrote one code of only 70 lines:

`fish.py` uses UFL to define function space, boundary conditions, and weak form, and calls the Firedrake (PETSc) nonlinear solver . . . only a few important lines as before:

```
Sh = FunctionSpace(mesh,'CG',degree=k)
bc = DirichletBC(Sh, g, bdry_ids)
u, v = Function(Sh), TestFunction(Sh)
F = (dot(grad(u), grad(v)) - f * v) * dx
solve(F == 0, u, bcs=bc, ...)
```

- test problem: $u(x,y) = -xe^y$ is exact solution
  - very smooth solution
  - f is computed from u $\longleftarrow$ *manufactured exact solution*

refine and measure convergence

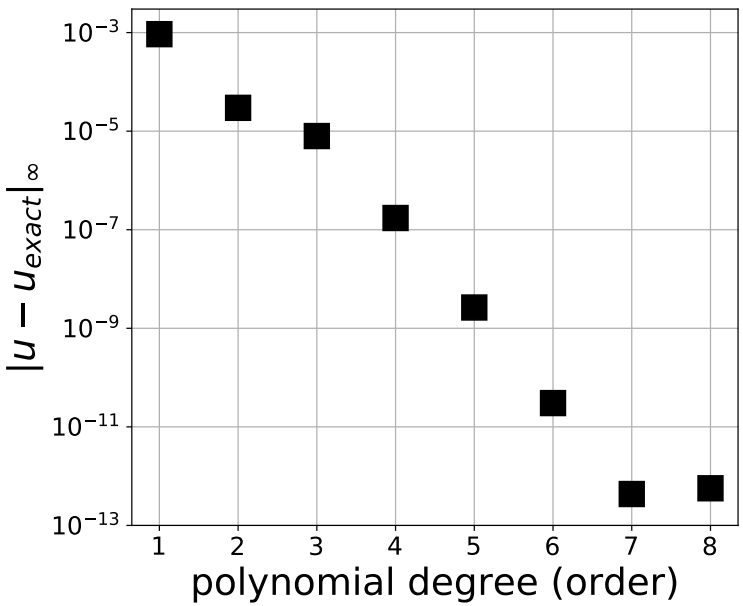
- one can approximate the continuum solution by either
  - h-refinement: make the triangles in `mesh` smaller
  - p-refinement: increase the polynomial degree `k`
- h-refinement chooses $m \times m$ grid where $m = 2^{\mathtt{X}+1} + 1$:

```
$ ./fish.py -refine X
```

- p-refinement (*spectral method*) chooses $P^{\mathtt{K}}$ on each element:

```
$ ./fish.py -k K
```

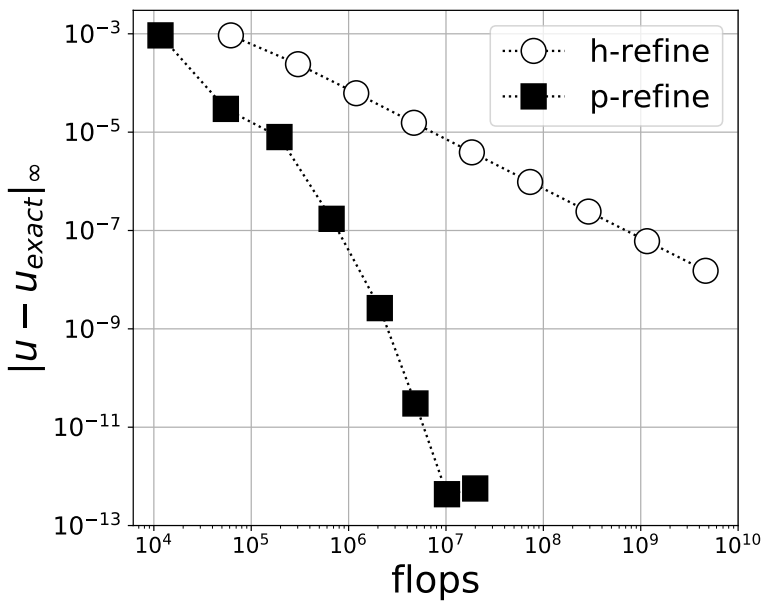
  - for example ...
  - fix $\mathtt{X} = 1$ for $m = 5$
  - vary k

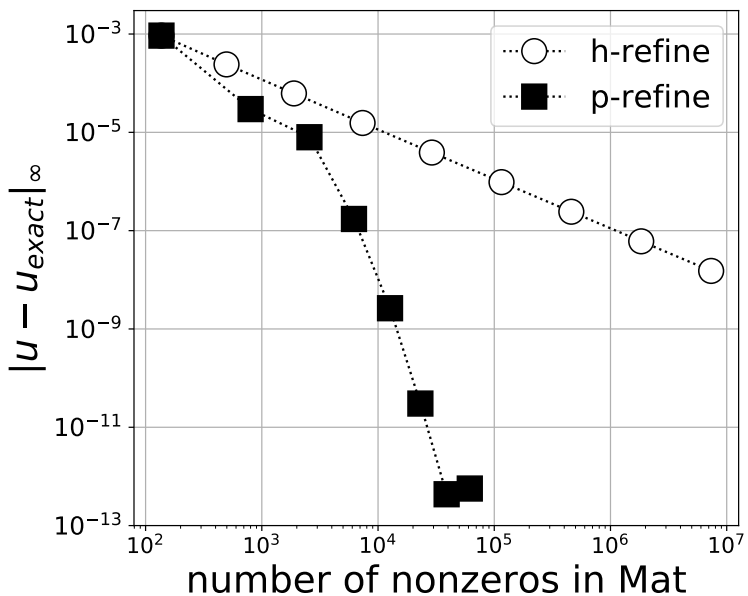


- combine options for h/p-refinement: `-refine X -k K`

measure performance

- performance runs use different solvers (educated choices!)
  - h-refinement runs use CG + (geometric multigrid)
  - p-refinement runs use Cholesky
- how to do a fair comparison of refinement modes?
- by error-vs-flops (left figure)
- by number of nonzeros in Jacobian matrix (right figure)





Outline

nonlinear Poisson problems with general boundary conditions

assembly of FEM system

solving the equations

modern software: Gmsh, Firedrake, PETSc

fluid flow example

Stokes equations for viscous, incompressible fluids

- the model for very-viscous fluids with viscosity $\mu > 0$
  - "very viscous" = (Reynold's number $\to 0$)
  - e.g. honey, motor oil, glacier ice
- solve for velocity $\mathbf{u}$ and pressure p on domain Ω:

$$-\mu \triangle \mathbf{u} + \nabla p = 0$$
$$\nabla \cdot \mathbf{u} = 0$$

- here only consider Dirichlet boundary conditions:

$$\mathbf{u} = \mathbf{g} \quad \text{on } \partial\Omega$$

Stokes weak form

- weak form from multiplying by test functions $\mathbf{v}$, q:

$$\int_\Omega \mu \nabla \mathbf{u} : \nabla \mathbf{v} - p \nabla \cdot \mathbf{v} = 0, \qquad \int_\Omega q \nabla \cdot \mathbf{u} = 0$$

- can write this as one bilinear form,

$$k(\mathbf{u}, p; \mathbf{v}, q) = \mu\, a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) + b(\mathbf{u}, q)$$

  where

$$a(\mathbf{w}, \mathbf{v}) = \int_\Omega \nabla \mathbf{w} : \nabla \mathbf{v}, \qquad b(\mathbf{v}, q) = -\int_\Omega q \nabla \cdot \mathbf{v}$$

- so $\mathbf{u}$, p satisfy: $\quad k(\mathbf{u}, p; \mathbf{v}, q) = \ell(\mathbf{v})$ for all $\mathbf{v}, q$

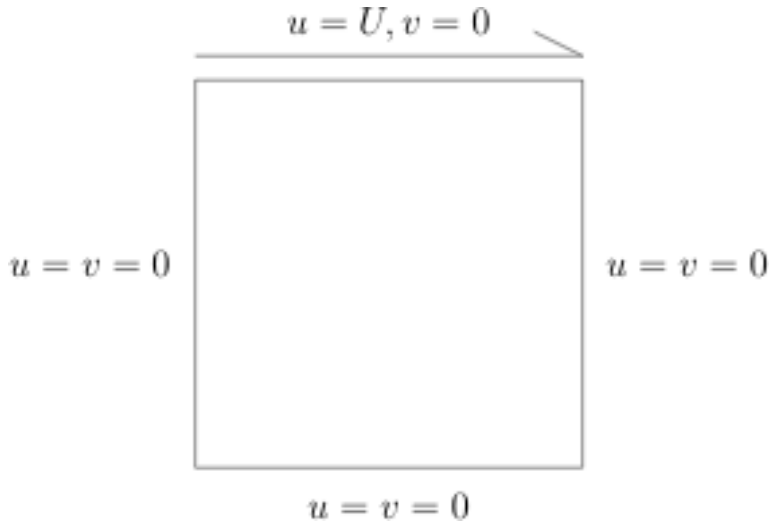
second UFL example

- suppose we choose $\mathbf{u} \in P^{k+2}$ and $p \in P^k$
  - "mixed" FEM for an incompressible fluid
  - nontrivial fact: these are a stable element choice
  - "Taylor-Hood"
- but Firedrake/UFL makes it easy to put into code ...

```
V = VectorFunctionSpace(mesh, 'CG', degree=k+2)  # velocity space
W = FunctionSpace(mesh, 'CG', degree=k)          # pressure space
Z = V * W
up = Function(Z)
u,p = split(up)
v,q = TestFunctions(Z)
F = (mu * dot(grad(u), grad(v)) - p * div(v) - div(u) * q) * dx
bc = ...
solve(F == 0, up, bcs=bc, ...)
```

lid-driven cavity problem

- choose a particular problem: lid-driven cavity
  - standard test problem
  - Dirichlet boundary conditions
  - $\Omega = (0,1)^2$

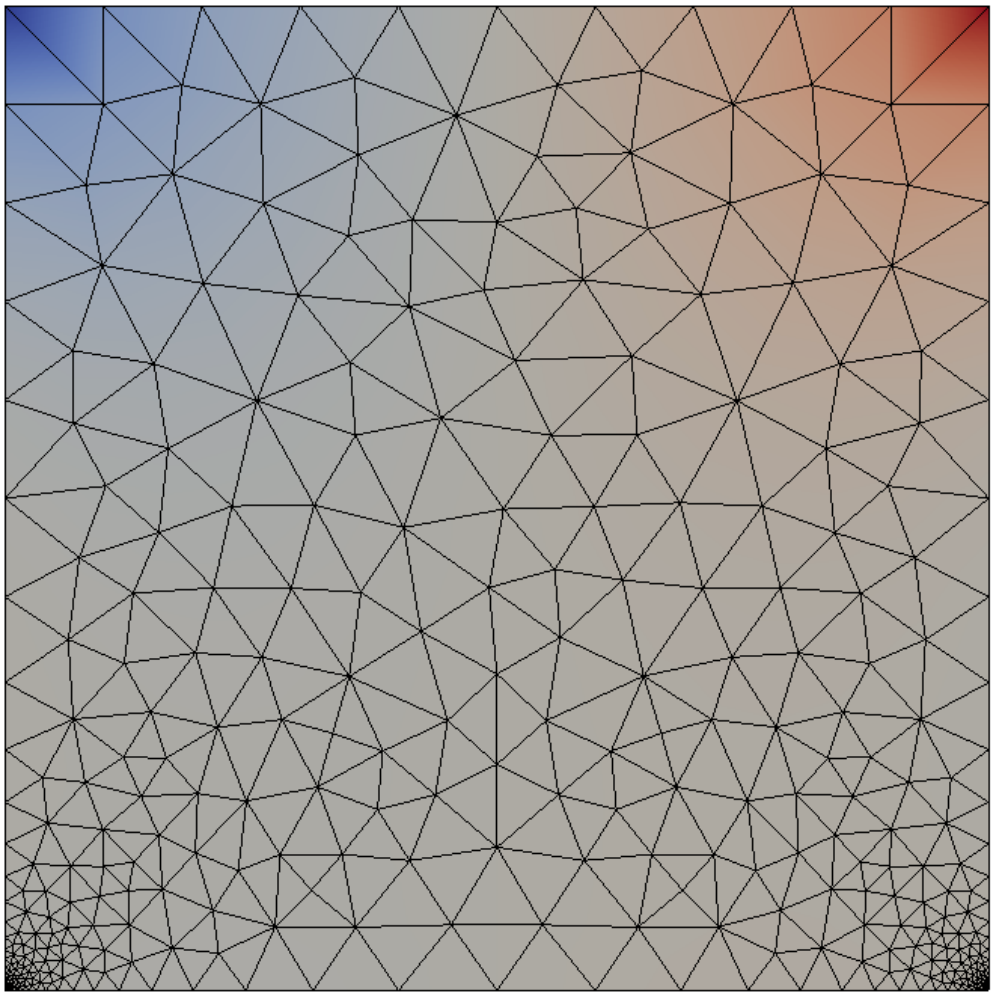


- boundary conditions code:

```
bc = [ DirichletBC(Z.sub(0), u_noslip, other),
       DirichletBC(Z.sub(0), u_lid,    lid)   ]
```

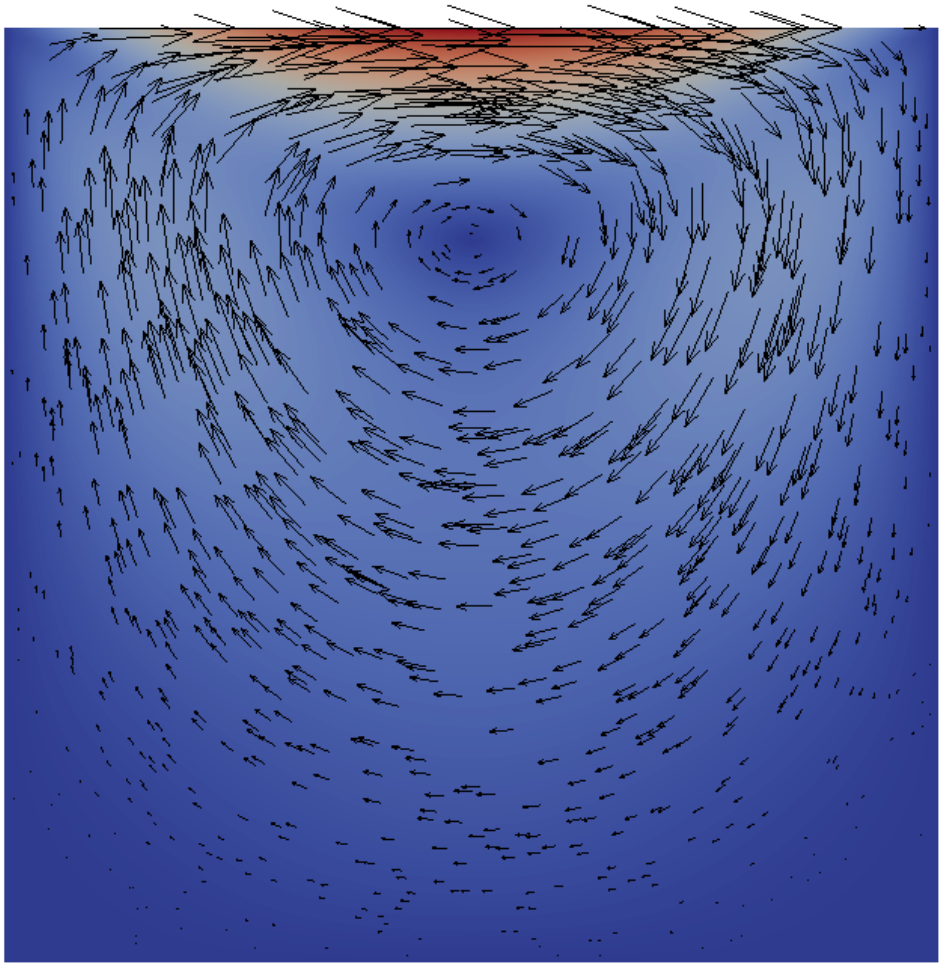
mesh and pressure

- non-uniform mesh: $N = 394$ nodes, $K = 686$ elements
- coloring by pressure



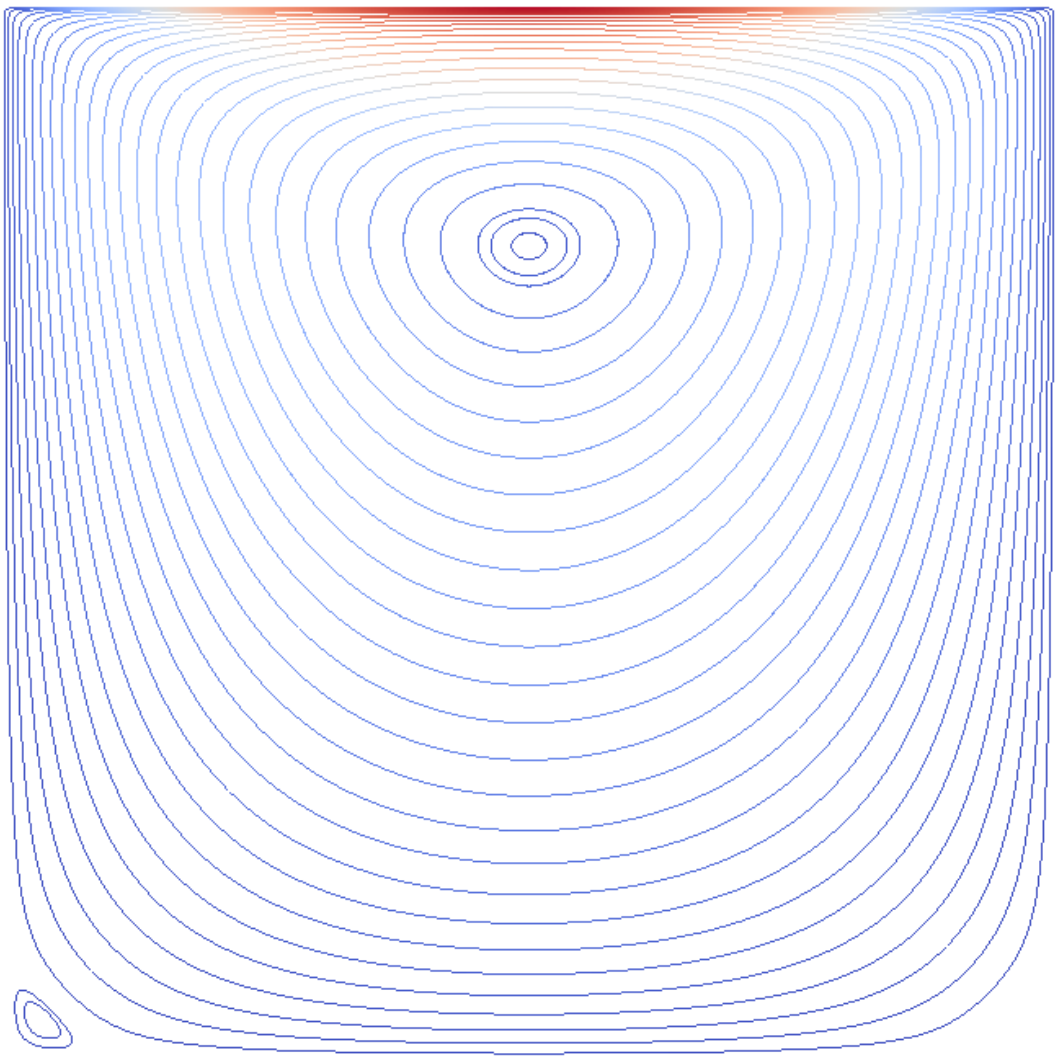
velocity vector field

- refined mesh: $N = 350{,}000$ nodes, $K = 700{,}000$ elements
- coloring by velocity magnitude



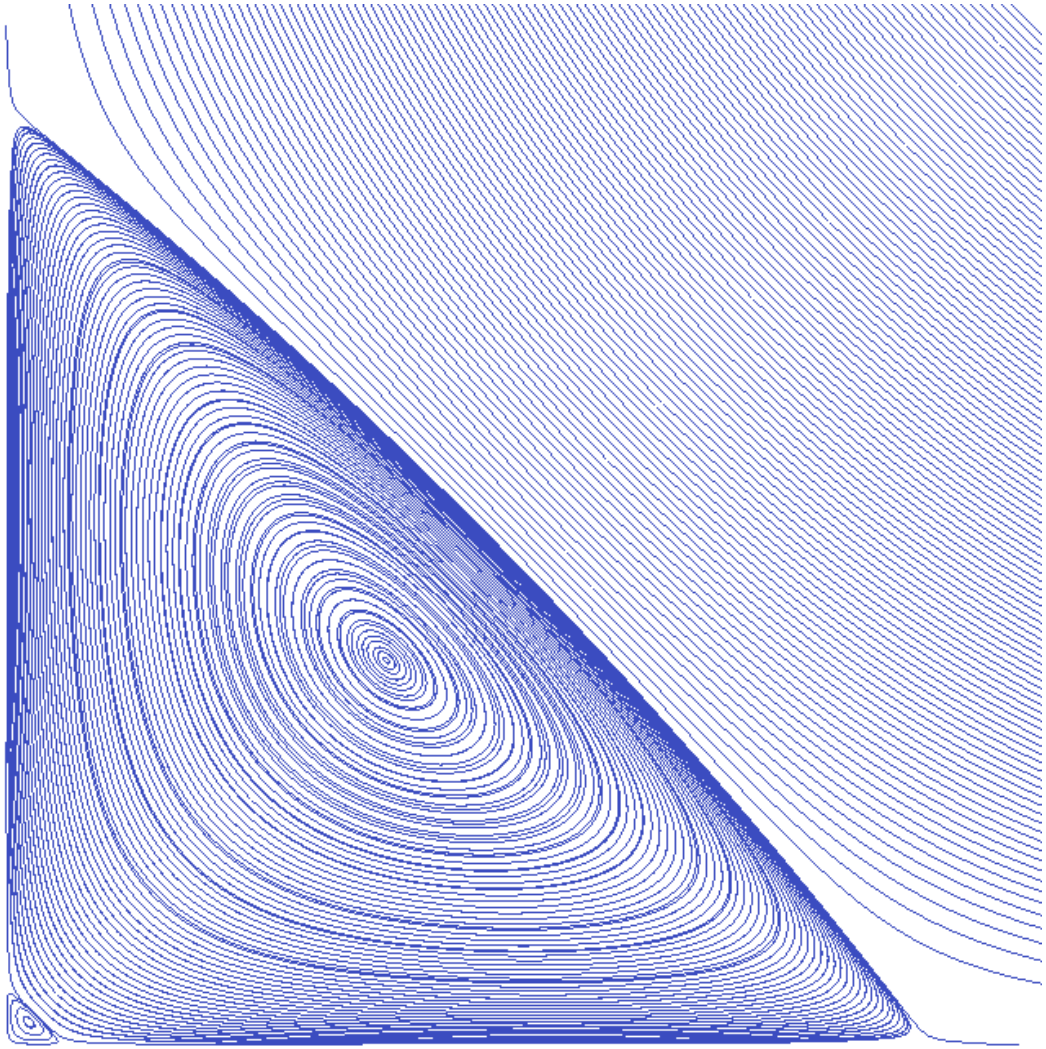
streamlines

- another view: streamlines



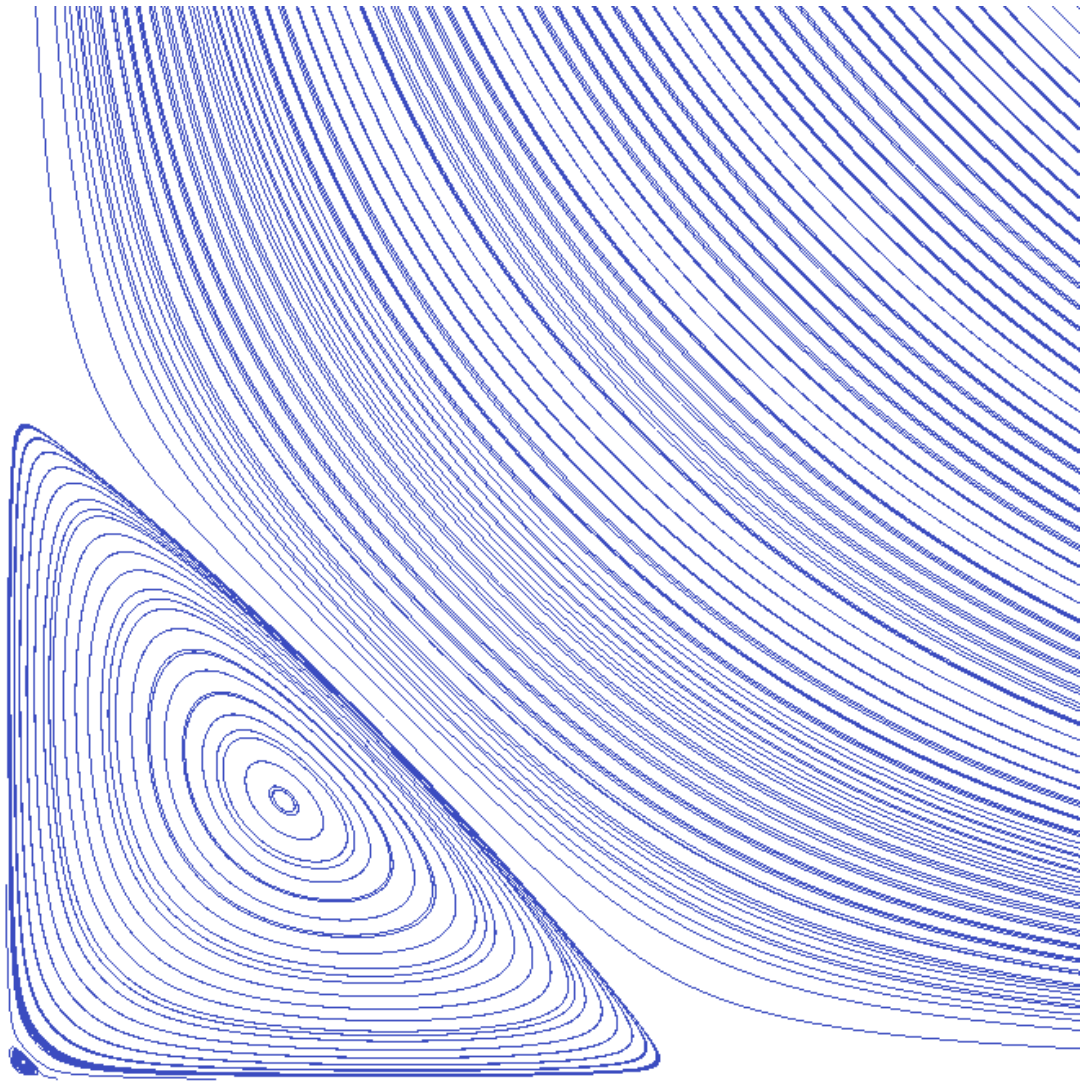
zoom to Moffat eddy

- zoom to show eddy in corner; add streamlines . . .



3 levels of Moffat eddies

- zoom again; third appears!



regarding Moffatt eddies

- Moffatt (1964): beautiful similarity solution analysis
- Acheson (1990), *Elementary Fluid Mechanics*:

  > *[if the corner angle is] below 146.3° . . . then a simple flow . . . is not possible and corner eddies occur instead. Indeed, as we probe deeper and deeper into each corner we find, in [Moffatt's] theory, not just one eddy but an infinite sequence of nested, alternately rotating eddies . . . each eddy is 1000 times weaker than the next; even with a 90 minute exposure time [a laboratory experiment] failed to detect the third eddy.*