Oct 25, 2018

Incompressible Navier-Stokes with Finite Element

FENICS PROJECT

FEniCS is an automated programming environment for differential equations

- C++/Python library
- Initiated 2003 in Chicago
- 1000–2000 monthly downloads
- Part of Debian and Ubuntu
- Licensed under the GNU LGPL

`http://fenicsproject.org/`

Collaborators

Simula Research Laboratory, University of Cambridge, University of Chicago, Texas Tech University, KTH Royal Institute of Technology, Chalmers University of Technology, Imperial College London, University of Oxford, Charles University in Prague, ...

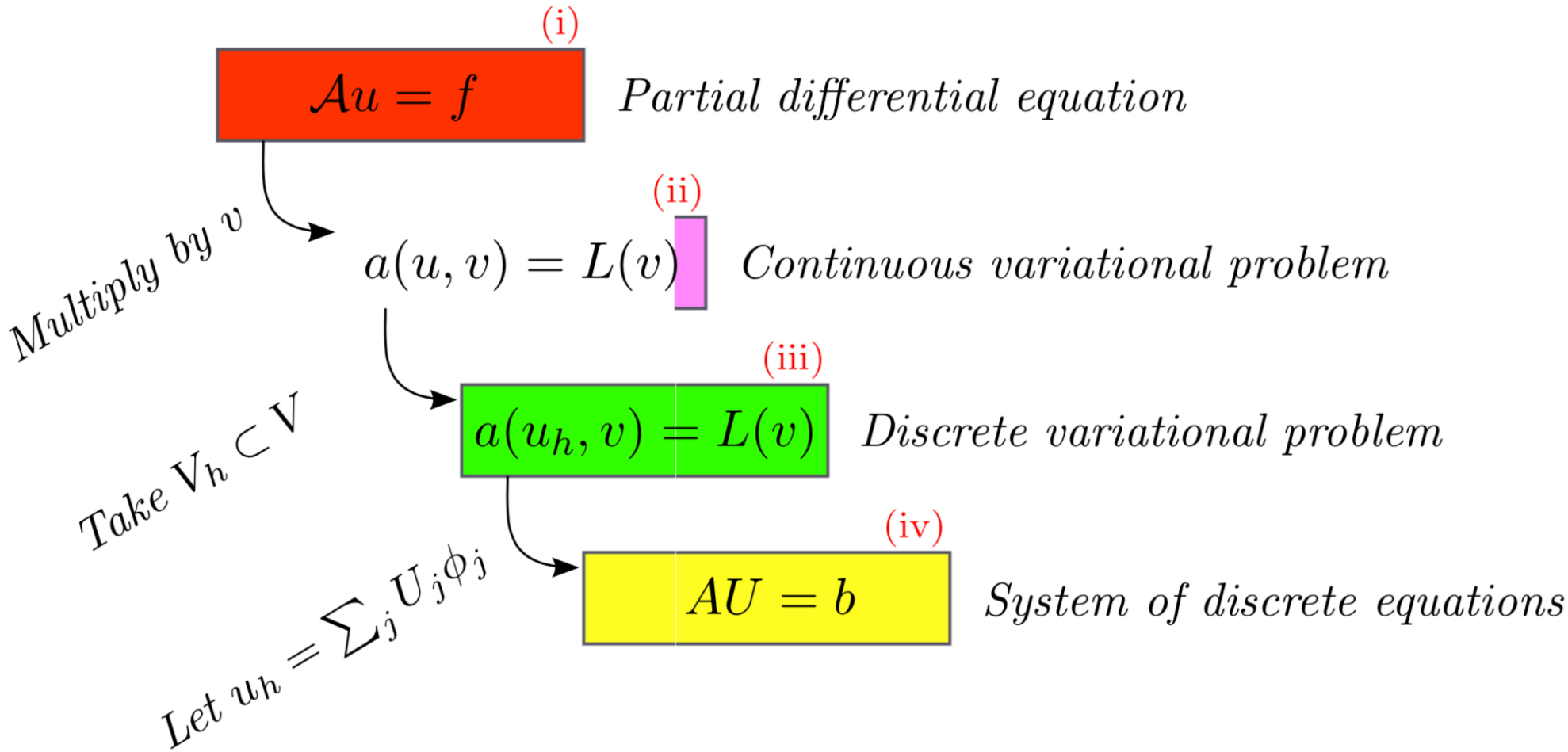
What is FEM?

The finite element method is a framework and a recipe for discretization of differential equations

- Ordinary differential equations
- Partial differential equations
- Integral equations

- A recipe for discretization of PDE
- PDE $\rightarrow Ax = b$
- Different bases, stabilization, error control, adaptivity

The FEM cookbook



Hello World in FEniCS: problem formulation

Poisson's equation

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

Finite element formulation

Find $u \in V$ such that

$$\underbrace{\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x}_{a(u,v)} = \underbrace{\int_\Omega f \, v \, \mathrm{d}x}_{L(v)} \quad \forall \, v \in V$$

From PDE (i) to variational problem (ii)

The simple recipe is: multiply the PDE by a test function v and integrate over Ω:

$$-\int_\Omega (\Delta u) v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

Then integrate by parts and set $v = 0$ on the Dirichlet boundary:

$$-\int_\Omega (\Delta u) v \,\mathrm{d}x = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x - \underbrace{\int_{\partial\Omega} \frac{\partial u}{\partial n} v \,\mathrm{d}s}_{=0}$$

We find that:

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

The variational problem (ii)

Find $u \in V$ such that

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$$

for all $v \in \hat{V}$

The trial space V and the test space $\hat{V}$ are (here) given by

$$V = \{v \in H^1(\Omega) : v = u_0 \text{ on } \partial\Omega\}$$
$$\hat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$$

From continuous (ii) to discrete (iii) problem

We approximate the continuous variational problem with a discrete variational problem posed on finite dimensional subspaces of V and $\hat{V}$:

$$V_h \subset V$$
$$\hat{V}_h \subset \hat{V}$$

Find $u_h \in V_h \subset V$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$$

for all $v \in \hat{V}_h \subset \hat{V}$

From discrete variational problem (iii) to discrete system of equations (iv)

Choose a basis for the discrete function space:

$$V_h = \text{span}\,\{\phi_j\}_{j=1}^{N}$$

Make an ansatz for the discrete solution:

$$u_h = \sum_{j=1}^{N} U_j \phi_j$$

Test against the basis functions:

$$\int_\Omega \nabla(\underbrace{\sum_{j=1}^{N} U_j \phi_j}_{u_h}) \cdot \nabla \phi_i \,\mathrm{d}x = \int_\Omega f \phi_i \,\mathrm{d}x$$

From discrete variational problem (iii) to discrete system of equations (iv), contd.

Rearrange to get:

$$\sum_{j=1}^{N} U_j \underbrace{\int_\Omega \nabla\phi_j \cdot \nabla\phi_i \,\mathrm{d}x}_{A_{ij}} = \underbrace{\int_\Omega f\phi_i \,\mathrm{d}x}_{b_i}$$

A linear system of equations:

$$AU = b$$

where

$$A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \,\mathrm{d}x \tag{1}$$

$$b_i = \int_\Omega f\phi_i \,\mathrm{d}x \tag{2}$$

The assembly algorithm

$A = 0$

for $T \in \mathcal{T}$

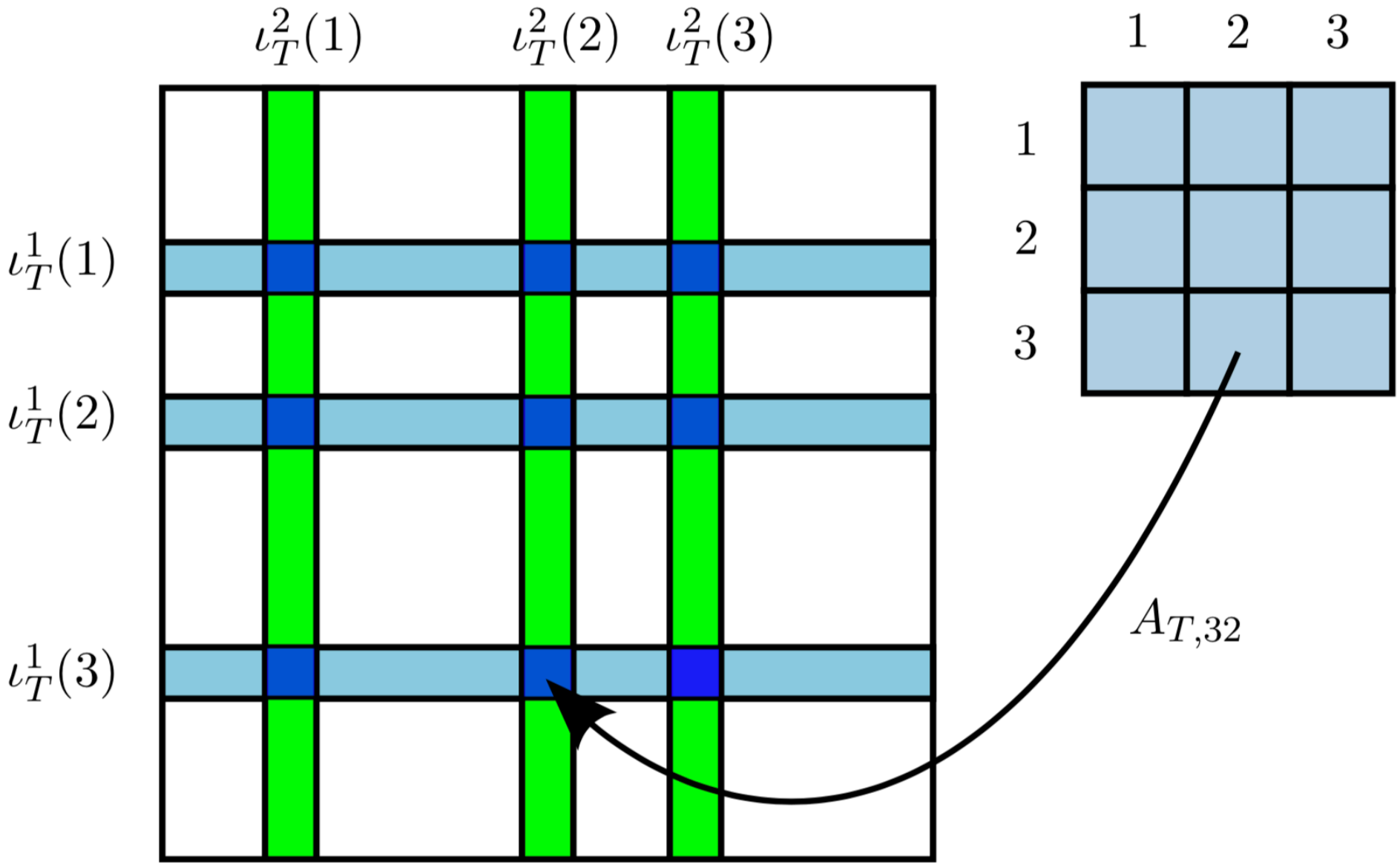
Compute the element matrix A_T

Compute the local-to-global mapping ι_T

Add A_T to A according to ι_T

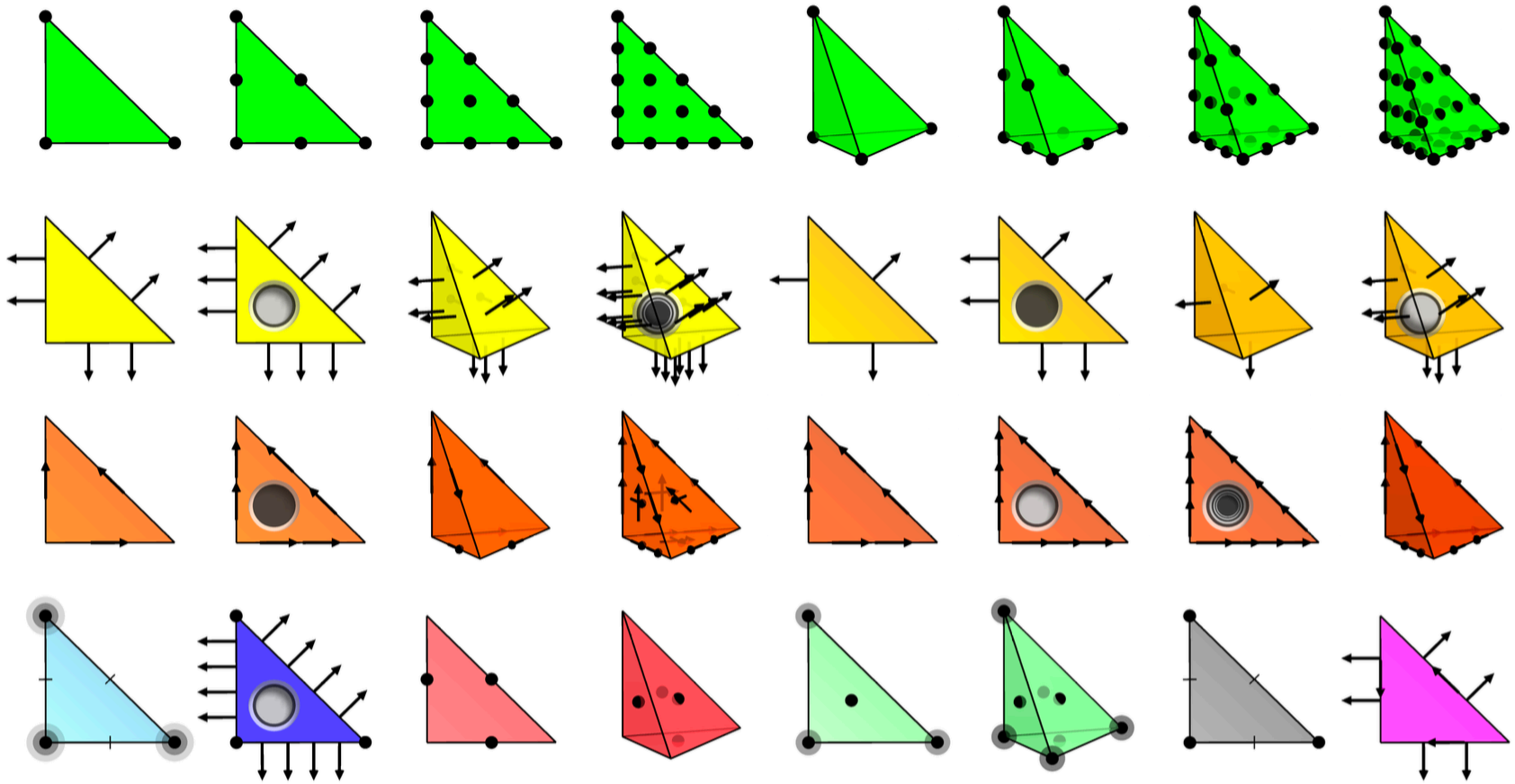
end for

Adding the element matrix A_T



FEniCS is automated FEM

- Automated generation of basis functions
- Automated evaluation of variational forms
- Automated finite element assembly
- Automated adaptive error control



Installation using Docker

Follow instructions to install Docker on linux, mac, or windows:

https://docs.docker.com/linux/ or `mac/`, `windows/`

Download and open a terminal in a clean FEniCS environment:

Bash code

```
$ docker run -ti quay.io/fenicsproject/dev
```

More instructions on using FEniCS Docker images here:

http://fenics-containers.readthedocs.org

TASK

Step 1. Sharing files from the host (your computer) into the docker container

```
docker run -ti -v $(pwd):/home/fenics/shared quay.io/fenicsproject/stable:1.6.0
```

Step 2. Use an editor and run Poisson.py file

visualize results in ParaView

Artificial Compressibility Method

Incompressible Navier-Stokes Equations (INSE) in Eulerian form:

$$\partial_t \rho + \rho \nabla . \mathbf{u} = 0 \tag{1}$$

$$\partial_t \mathbf{u} + \mathbf{u} . \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \tag{2}$$

Artificial Compressibility Method

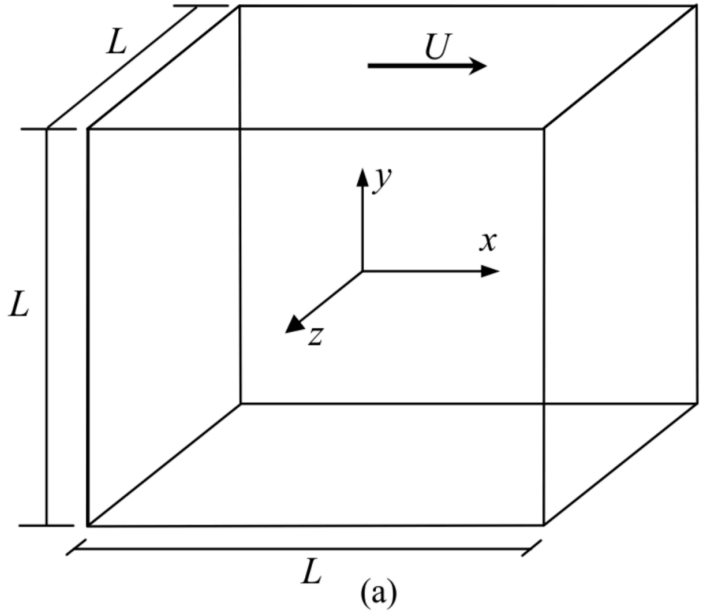
$$\partial_t P + c^2 \nabla . \mathbf{u} = 0$$

$$\partial_t \mathbf{u} + \mathbf{u} . \nabla \mathbf{u} = -\nabla P + \nu \nabla^2 \mathbf{u} + \mathbf{g}$$

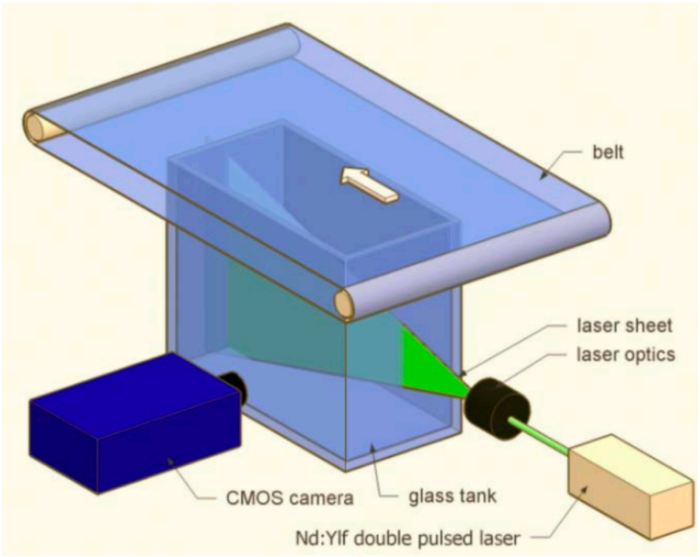
where $P = \frac{p}{\rho}$.

Lid-Driven Cavity

- Characteristics: steady-state, incompressible
- Boundary Conditions: walls everywhere (top wall is moving)
  - Velocity: Dirichlet ($\mathbf{U} = 0$ except for the top wall where $\mathbf{U}_x = 1$ and $\mathbf{U}_y = 0$)
  - Pressure: Neumann ($\nabla p = 0$)



(a)



(b)

Parameters

$$L = 1$$
$$U_x = 1$$
$$dx = 10$$
$$Re = 100 \rightarrow \nu = 0.01$$
$$CFL < 1 \rightarrow dt = 0.015$$
$$c = 2.4$$

Pseudo Code

- Generate mesh
- Define function spaces $\rightarrow$ test and trial functions
- Define Γ^D (and Γ^N) BCs
- Solve
    1. Solve momentum equation to get $\mathbf{u}^{(n+1)}$
    2. Solve continuity equation to get $p^{(n+1)}$
    3. Check convergence criteria
- Output the fields data

Convergence Criteria

1. $E_u = \sqrt{(\Delta t \Delta x \Delta y) \sum_{i,j} (u_{i,j}^{n+1} - u_{i,j}^{n})^2}$
2. $E_v = \sqrt{(\Delta t \Delta x \Delta y) \sum_{i,j} (v_{i,j}^{n+1} - v_{i,j}^{n})^2}$
3. $E_p = \sqrt{\frac{\Delta t \Delta x \Delta y}{c^2} \sum_{i,j} (P_{i,j}^{n+1} - P_{i,j}^{n})^2}$
4. $E_\nabla = (\Delta t \Delta x \Delta y) \nabla . \mathbf{u}$
5. $E_{total} = \max\{E_u, E_v, E_p, E_\nabla\} < \varepsilon$

where ε is the desired tolerance.

Derive weak forms for

$$\partial_t P + c^2 \nabla . \mathbf{u} = 0$$

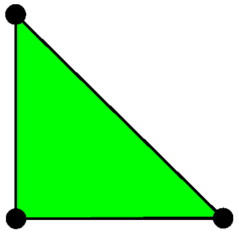
$$\partial_t \mathbf{u} + \mathbf{u} . \nabla \mathbf{u} = -\nabla P + \nu \nabla^2 \mathbf{u} + \mathbf{g}$$

?

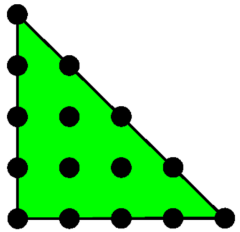
Use CG1 element (linear shape functions)

Lagrange (CG) elements

First order on triangles



Fourth order on triangles



Direct methods

- Gaussian elimination
  - Requires $\sim \frac{2}{3}N^3$ operations
- LU factorization: $A = LU$
  - Solve requires $\sim \frac{2}{3}N^3$ operations
  - Reuse L and U for repeated solves
- Cholesky factorization: $A = LL^\top$
  - Works if A is symmetric and positive definite
  - Solve requires $\sim \frac{1}{3}N^3$ operations
  - Reuse L for repeated solves

Iterative methods

Krylov subspace methods

- GMRES (Generalized Minimal RESidual method)
- CG (Conjugate Gradient method)
  - Works if A is symmetric and positive definite
- BiCGSTAB, MINRES, TFQMR, ...

Multigrid methods

- GMG (Geometric MultiGrid)
- AMG (Algebraic MultiGrid)

Preconditioners

- ILU, ICC, SOR, AMG, Jacobi, block-Jacobi, additive Schwarz, ...

Which method should I use?

Rules of thumb

- Direct methods for small systems
- Iterative methods for large systems
- Break-even at ca 100–1000 degrees of freedom
- Use a symmetric method for a symmetric system
  - Cholesky factorization (direct)
  - CG (iterative)
- Use a multigrid preconditioner for Poisson-like systems
- GMRES with ILU preconditioning is a good default choice

```
solve(a == L, u, bc)
      solver_parameters={'linear_solver': 'cg',
                         'preconditioner': 'ilu'})
```

```
list_linear_solver_methods()
list_krylov_solver_preconditioners()
```

Name	Method
'icc'	Incomplete Cholesky factorization
'ilu'	Incomplete LU factorization
'petsc_amg'	PETSc algebraic multigrid
'sor'	Successive over-relaxation

Name	Method
'bicgstab'	Biconjugate gradient stabilized method
'cg'	Conjugate gradient method
'gmres'	Generalized minimal residual method
'minres'	Minimal residual method
'petsc'	PETSc built in LU solver
'richardson'	Richardson method
'superlu_dist'	Parallel SuperLU
'tfqmr'	Transpose-free quasi-minimal residual method
'umfpack'	UMFPACK

TASK

A numerical solution to Taylor–Green Vortex benchmark problem:

- Initial Condition:

$$u(x, y) = \sin(x)\cos(y)$$
$$v(x, y) = -\cos(x)\sin(y)$$

- Boundary condition: walls all around
- Parameters: $L = 2\pi$, $\nu = 0.2$, $\Delta t = 10^{-4}$ and $Re = 1000$.
- Analytical Solution:

$$u(x, y) = \sin(x)\cos(y)e^{-2\nu t}$$
$$v(x, y) = \cos(x)\sin(y)e^{-2\nu t}$$
$$p(x, y) = \frac{\rho}{4}(\cos(2x) + \cos(2y))e^{-4\nu t}$$

Mixed Finite Element Method

- Different interpolation for velocity and pressure are required

The incompressible Navier–Stokes equations

Constitutive equations

$$\rho(\dot{u} + u \cdot \nabla u) - \nabla \cdot \sigma(u, p) = f \quad \text{in } \Omega \times (0, T]$$
$$\nabla \cdot u = 0 \quad \text{in } \Omega \times (0, T]$$

Boundary conditions

$$u = g_{\mathrm{D}} \quad \text{on } \Gamma_{\mathrm{D}} \times (0, T]$$
$$\sigma \cdot n = t_{\mathrm{N}} \quad \text{on } \Gamma_{\mathrm{N}} \times (0, T]$$

Initial condition

$$u(\cdot, 0) = u_0 \quad \text{in } \Omega$$

Mixed variational formulation of Navier–Stokes

Multiply the momentum equation by a test function v and integrate by parts:

$$\int_\Omega \rho(\dot{u}+u\cdot\nabla u)\cdot v\,\mathrm{d}x+\int_\Omega \sigma(u,p):\varepsilon(v)\,\mathrm{d}x=\int_\Omega f\cdot v\,\mathrm{d}x+\int_{\Gamma_\mathrm{N}} t_\mathrm{N}\cdot v\,\mathrm{d}s$$

- $\varepsilon(u)=\frac{1}{2}(\nabla u+\nabla u^\top)$ is the strain rate tensor
- $\sigma(u,p)=2\mu\varepsilon(u)-pI$ is the Cauchy stress tensor

$$\langle\rho\dot{u},v\rangle+\langle\rho u\cdot\nabla u,v\rangle+\langle\sigma(u,p),\varepsilon(v)\rangle=\langle f,v\rangle+\langle t_\mathrm{N},v\rangle_{\Gamma_\mathrm{N}}$$

Multiply the continuity equation by a test function q and sum up: find $(u,p)\in V$ such that

$$\langle\rho\dot{u},v\rangle+\langle\rho u\cdot\nabla u,v\rangle+\langle\sigma(u,p),\varepsilon(v)\rangle+\langle\nabla\cdot u,q\rangle=\langle f,v\rangle+\langle t_\mathrm{N},v\rangle_{\Gamma_\mathrm{N}}$$

for all $(v,q)\in\hat{V}$

Newton's method

$-\nabla \cdot (q(u)\nabla u) = f.$ $\qquad u^{k+1} = u^k + \delta u$

$$q(u^{k+1}) = q(u^k) + q'(u^k)\delta u + \mathcal{O}((\delta u)^2) \approx q(u^k) + q'(u^k)\delta u,$$

and dropping other nonlinear terms in δu, we get

$$\nabla \cdot \left(q(u^k)\nabla u^k\right) + \nabla \cdot \left(q(u^k)\nabla \delta u\right) + \nabla \cdot \left(q'(u^k)\delta u \nabla u^k\right) = 0.$$

We may collect the terms with the unknown δu on the left-hand side,

$$\nabla \cdot \left(q(u^k)\nabla \delta u\right) + \nabla \cdot \left(q'(u^k)\delta u \nabla u^k\right) = -\nabla \cdot \left(q(u^k)\nabla u^k\right),$$

$$a(\delta u, v) = \int_\Omega \left(q(u^k)\nabla \delta u \cdot \nabla v + q'(u^k)\delta u \nabla u^k \cdot \nabla v\right) \mathrm{d}x,$$

$$L(v) = -\int_\Omega q(u^k)\nabla u^k \cdot \nabla v \,\mathrm{d}x.$$

Weak form:

$$F(u;v) = \int_\Omega q(u)\nabla u \cdot \nabla v \, \mathrm{d}x,$$

Gateaux derivative of F:

$$J = a(\delta u, u) = DF(u^k; \delta u, v) \text{ of } F(u;v)$$

$$= \lim_{\epsilon \to 0} \frac{d}{d\epsilon} F_i(u^k + \epsilon \delta u; v)$$

$$= \frac{d}{d\epsilon} \int_\Omega \nabla v \cdot \left(q(u^k + \epsilon \delta u) \nabla (u^k + \epsilon \delta u) \right) \, \mathrm{d}x$$

Which leads to linearized Bilinear form

$$= \int_\Omega \nabla v \cdot [q'(u_- + \epsilon \delta u)\delta u \nabla (u_- + \epsilon \delta u) + q(u_- + \epsilon \delta u)\nabla \delta u] \, \mathrm{d}x,$$

```
du = TrialFunction(V)
v  = TestFunction(V)
u_ = Function(V)      # the most recently computed solution
F  = inner(q(u_)*nabla_grad(u_), nabla_grad(v))*dx

J  = derivative(F, u_, du)  # Gateaux derivative in dir. of du
```

```
problem = NonlinearVariationalProblem(F, u, bcs, J)
solver  = NonlinearVariationalSolver(problem)
solver.solve()
```

Splitting Scheme for NS

- The schemes do not require special mixed elements and have for this reason been prefered
- There are issues with boundary layers, errors etc that you need to be aware of

Splitting scheme for Navier-Stokes: Core idea

- Solving the full coupled system for the velocity and the pressure simultaneously is computationally expensive.
- To reduce computational cost, iterative *splitting schemes* are an attractive alternative.
- Splitting schemes are typically based on solving for the velocity and the pressure separately.
- We will consider a splitting scheme solving three different (smaller!) systems at each time step n:
  1. Compute the *tentative velocity*
  2. Compute the *pressure*
  3. Compute the *corrected velocity*

1. Compute the tentative velocity

$$u^* = u^n + \Delta t(-u^n \cdot \nabla u^n) + \nu \Delta u^n + f^n$$

2. Solve a Poisson problem for the pressure update

$$-\nabla^2 \phi = -\frac{\rho}{\Delta t} \nabla \cdot u^* \qquad \phi = (p^{n+1} - p^n)$$

3. Update velocity

$$u^{n+1} = u^* - \frac{\Delta t}{\rho} \nabla \phi$$

4. Update pressure

$$p^{n+1} = p^n + \phi$$