CFD Code Development Frameworks
Python, C, OpenFOAM, Fenics

Dr. Rodolfo Monico, Taher Chegini, M. Sarraf Joshaghani

CTFM 2018
University of Houston

rostilla@uh.edu
tchegini@uh.edu
m.sarraf.j@uh.edu

June 2, 2018

Table of Content

Governing Equations

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

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

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

Eqn. 1: Density can be omitted, but let's keep it for now.
Eqn. 2:

- LHS is acceleration which is intrinsically Lagrangian and when it is translated to Eulerian coordinates it becomes non-linear
- RHS describes influence of pressure gradient, viscosity and body forces.

Governing Equations
Complexities

Difficulties for numerical solution:

- Nonlocality of pressure gradient (Solution: None)
- Nonlinearity of acceleration term (Solution: e.g. Picard's Method)
- Pressure and velocity are coupled and there is no separate equation for pressure (Solution: e.g. SIMPLE, PISO and ACM)
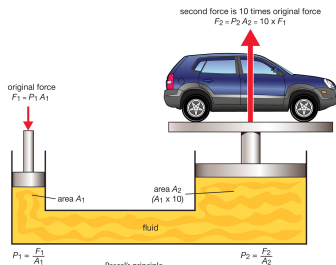
Governing Equations
Artificial Compressibility Method

ACM first developed by A.J. Chorin in 1967 assumes a small compressibility for the fluid and isothermal condition for the flow:

$$\rho = \rho(p) \therefore \partial_t \rho = \frac{\partial \rho}{\partial p} \frac{\partial p}{\partial t} = \frac{1}{c^2} \frac{\partial p}{\partial t}$$

where c is artificial sound speed. So INSE becomes:

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

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

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

Governing Equations
Remarks on ACM

- Limited scope of applicability: incompressible steady state problems.
- Small time step: t is pseudo-time step and depends on c. Therefore, if an explicit discretization method is used time step should have a small value.
- Efficient parallelization: no elliptic PDE

Although applicability of the method is very limited, ACM provides a simple yet informative framework for pedagogical purposes
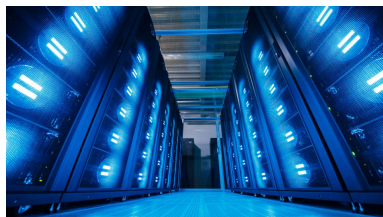
Table of Content

Governing Equations
○○○○○

Solution algorithm
○●○○○○○○

Grid
○○○

Problem Statement
○○○○○○

Solution algorithm
Pseudo Code

1. Initialization of the domain and the variables (u, v, p)
2. While convergence is reached do:
   1. Solve momentum equation in x-direction to get u
   2. Solve momentum equation in y-direction to get v
   3. Update the boundary conditions for u and v
   4. Solve continuity equation to get P
   5. Check convergence criteria
3. Output the fields data

Governing Equations
○○○○○

Solution algorithm
○○●○○○○○

Grid
○○○

Problem Statement
○○○○○○

Solution algorithm
Discretization

Spatial: central difference, second order

$$\partial_x \phi = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x}$$

Temporal: backward difference, first order

$$\partial_t \phi = \frac{\phi^{n+1} - \phi^n}{\Delta t}$$

Note: for brevity we use the superscript *n* to denote the variable in the current time step and no superscript for the previous time step.

Solution algorithm
Momentum Equation in x-direction

$$\partial_t u + \partial_x(uu) + \partial_y(uv) = -\partial_x P + \nu\left(\partial_{xx} u + \partial_{yy} u\right)$$

$$\frac{u_{i,j}^n - u_{i,j}}{\Delta t} + \frac{\left(\frac{u_{i+1,j}+u_{i,j}}{2}\right)^2 - \left(\frac{u_{i,j}+u_{i-1,j}}{2}\right)^2}{\Delta x}$$

$$+ \frac{\left(\frac{u_{i,j+1}+u_{i,j}}{2}\right)\left(\frac{v_{i+1,j}+v_{i,j}}{2}\right) - \left(\frac{u_{i,j}+u_{i,j-1}}{2}\right)\left(\frac{v_{i+1,j-1}+v_{i,j-1}}{2}\right)}{\Delta y}$$

$$=$$

$$-\frac{P_{i+1,j} - P_{i,j}}{\Delta x}$$

$$+ \nu\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right)$$

Solution algorithm
Momentum Equation in y-direction

$$\partial_t v + \partial_x(uv) + \partial_y(vv) = -\partial_y P + \nu\left(\partial_{xx}v + \partial_{yy}v\right)$$

$$\frac{v_{i,j}^n - v_{i,j}}{\Delta t} + \frac{\left(\frac{u_{i,j+1}+u_{i,j}}{2}\right)\left(\frac{v_{i+1,j}+v_{i,j}}{2}\right) - \left(\frac{u_{i-1,j+1}+u_{i-1,j}}{2}\right)\left(\frac{v_{i,j}+v_{i-1,j}}{2}\right)}{\Delta x}$$

$$+ \frac{\left(\frac{v_{i,j+1}+v_{i,j}}{2}\right)^2 - \left(\frac{v_{i,j}+v_{i,j-1}}{2}\right)^2}{\Delta y}$$

$$=$$

$$- \frac{P_{i,j+1} - P_{i,j}}{\Delta y}$$

$$+ \nu\left(\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{\Delta x^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta y^2}\right)$$

Governing Equations
○○○○○

Solution algorithm
○○○○○●○○

Grid
○○○

Problem Statement
○○○○○○

Solution algorithm
Continuity Equation

$$\partial_t P + c^2 \left(\partial_x u + \partial_y v \right) = 0$$

$$\frac{P_{i,j}^n - P_{i,j}}{\Delta t} + c^2 \left(\frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x} + \frac{v_{i,j}^n - v_{i,j-1}^n}{\Delta y} \right) = 0$$

Governing Equations
ooooo

Solution algorithm
ooooooeo

Grid
ooo

Problem Statement
oooooo

Solution algorithm
Boundary Conditions

The two common boundary conditions can be formulated as follows:

- Dirichlet: $\phi_g = 2\phi_b - \phi_i$
- Neumann: $\phi_g = \phi_i - \left(\Delta n \frac{\partial \phi}{\partial n}\right)_b$

Indicies: g, b and i denote a ghost, boundary and inner node, respectively

Variables: ϕ is a quantity of interest (u, v or P) and n represents a direction normal to the boundary cell face

Solution algorithm
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. $E_{tot} = \max\{E_u, E_v, E_p, E_\nabla) < \varepsilon$

where ε is the desired tolerance.

Table of Content

Governing Equations
○○○○○

Solution algorithm
○○○○○○○○

Grid
○●○

Problem Statement
○○○○○○

Grid

There are two types of grids from variable placement point of view:

- Collocated: all the variables at the node centers; easier to implement and extend while issues such as checkerboard problem arise
- Staggered: scalar variables at the node centers and vector variables at the face centers; more accurate though more difficult to implement
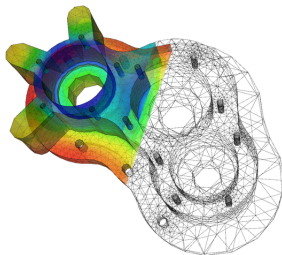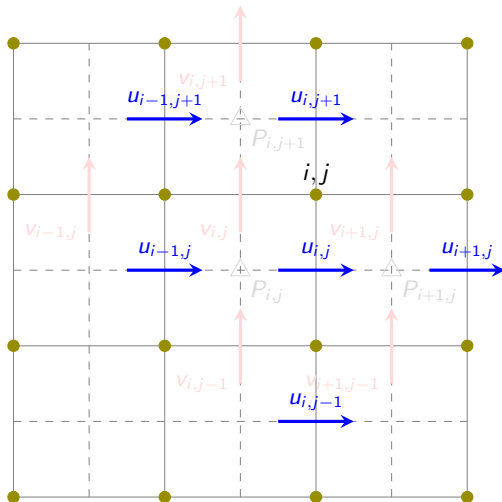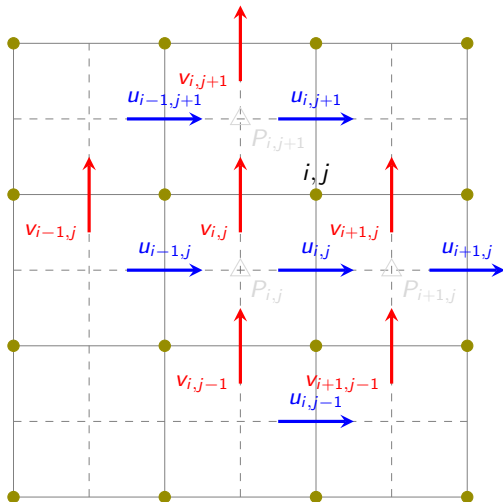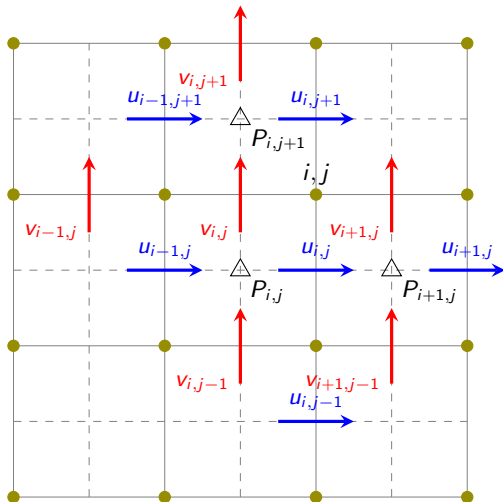
Grid
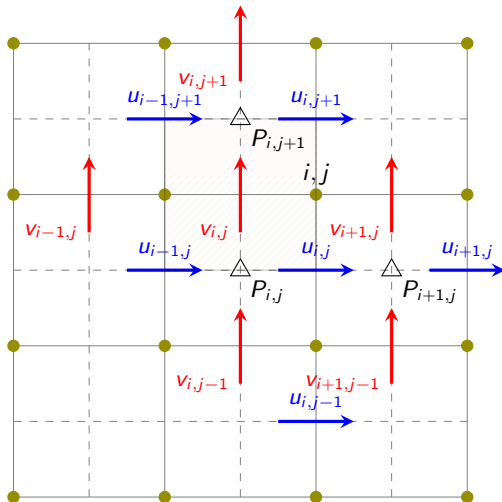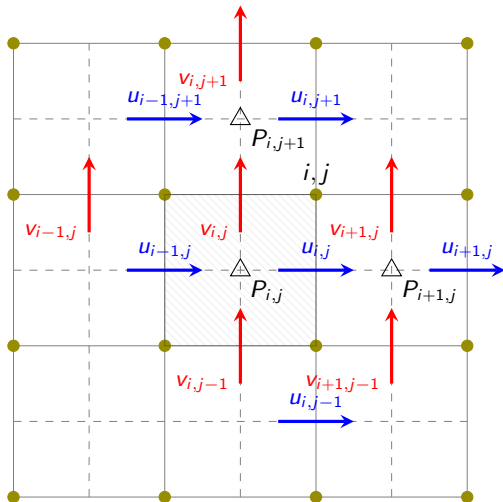## Staggered Grid
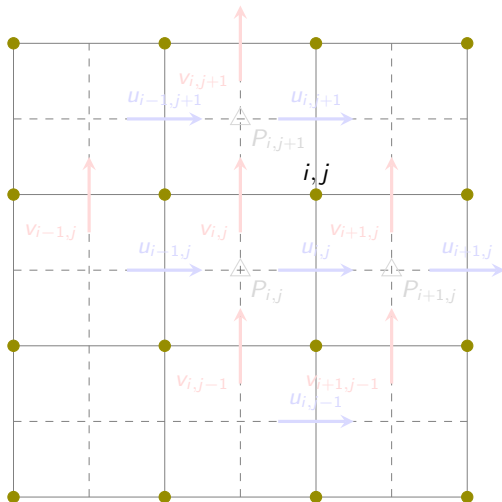
Grid
Staggered Grid

Grid
Staggered Grid

Governing Equations
○○○○○

Solution algorithm
○○○○○○○○

Grid
○○●

Problem Statement
○○○○○○

Grid
Staggered Grid

Grid
Staggered Grid

Governing Equations
○○○○○

Solution algorithm
○○○○○○○○

Grid
○○●
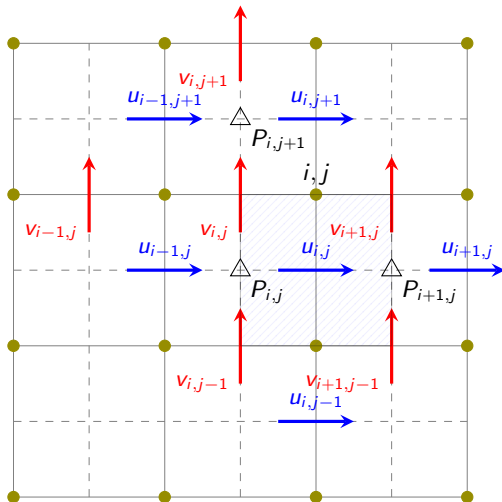
Problem Statement
○○○○○○

Grid
Staggered Grid

Grid
Staggered Grid

Table of Content

Governing Equations
○○○○○

Solution algorithm
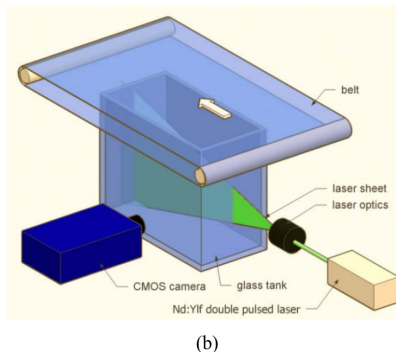○○○○○○○○

Grid
○○○

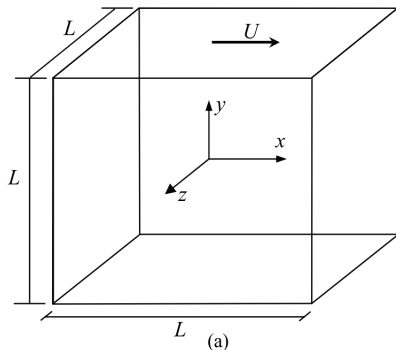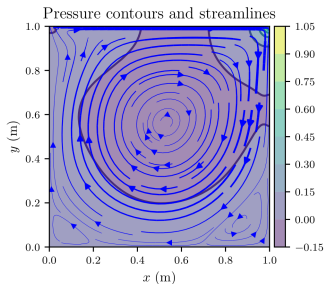Problem Statement
○●○○○○

Problem Statement
Lid-Driven Cavity

Characteristics: steady-state, incompressible
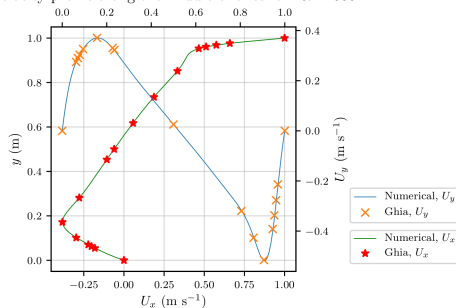BC: walls everywhere (top wall is moving)

(a)

(b)

Problem Statement

Outputs

Pressure contours and streamlines

Velocity profile along the middle of axes for Re=1000

Governing Equations
○○○○○

Solution algorithm
○○○○○○○○

Grid
○○○

Problem Statement
○○○●○○

Problem Statement
Workshop

A numerical solution to this benchmark problem will be presented based on:

- (Finite Difference) Coding from scratch: Python, C
- (Finite Volume) Modifying one of the solvers in OpenFOAM
- (Finite Element) Using libraries provided by FEniCS

Final Remarks

Some other interesting open source CFD projects to look into:

- **Basilisk**: solution of partial differential equations on adaptive Cartesian meshes. (C)

                    http://basilisk.fr

- **FluidDyn**: some Python packages specialized for different tasks, in particular for 2D and 3D Fast Fourier Transforms, numerical simulations, laboratory experiments and processing of images of fluid.

        https://fluiddyn.readthedocs.io/en/latest

- **Firedrake**: an automated system for the solution of partial differential equations using the finite element method. (Python)

            https://firedrakeproject.org

Let's Get Started!

```
$ mkdir UHWS

$ cd UHWS

$ git clone https://github.com/taataam/UHOFWorkshop.git

$ cd UHOFWorkshop/workshop3
```