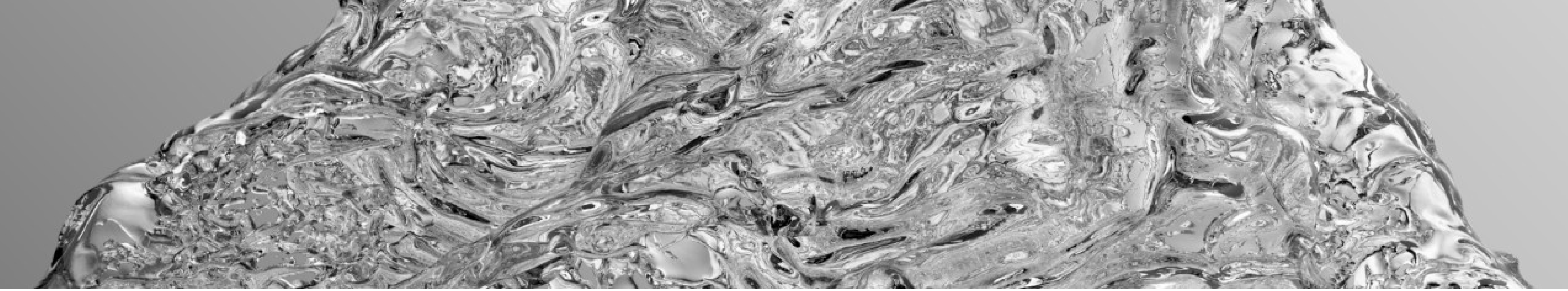


EEN-E2001 Computational Fluid Dynamics

Lecture 3*: OpenFOAM code and structure

Taught by Prof. Ville Vuorinen
Presenter: MEng. Ilya Morev

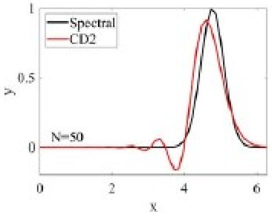
January 30th 2023

Aalto University, School of Engineering

Lecture 1: Linear PDEs and finite difference method

$$\boxed{\frac{\partial T}{\partial t}+\nabla\cdot T\boldsymbol{u}=\alpha\nabla^2 T}$$

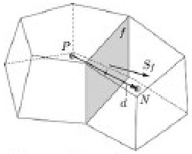
$$\boxed{\frac{\partial T}{\partial x}\approx\frac{T_{i+1}-T_{i-1}}{2\Delta x}}$$



Lecture 2: Gauss' theorem and finite volume method

$$\boxed{\int_\Omega\nabla\cdot(T\boldsymbol{u})\,d\Omega=\int_{\partial\Omega}(T\boldsymbol{u})\cdot\boldsymbol{n}\,dS}$$

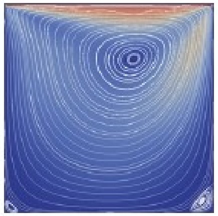
$$\boxed{\int_{\partial\Omega}\boldsymbol{u}\cdot\boldsymbol{n}\,dS\approx\Sigma_f\boldsymbol{u}_f\cdot\boldsymbol{n}_f\,dS_f}$$



Lecture 3: Navier-Stokes equation and pressure

$$\boxed{\frac{\partial\boldsymbol{u}}{\partial t}+\nabla\cdot\boldsymbol{uu}=-\nabla p+\nu\nabla^2\boldsymbol{u}}$$

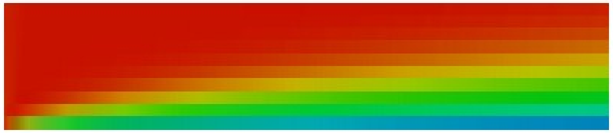
$$\boxed{-\nabla^2 p=\nabla\cdot\nabla\cdot\boldsymbol{uu}}$$

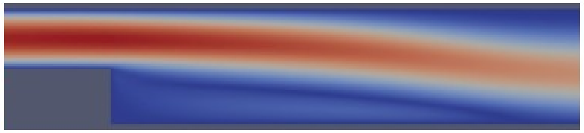


Lecture 4: OpenFOAM code and structure

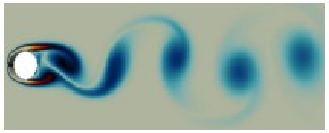
```
fvVectorMatrix UEqn
(
      fvm::ddt(U)
    + fvm::div(phi, U)
    - fvm::laplacian(nu, U)
);
```

Lecture 5: Simulating fluid physical phenomena: part A





Lecture 6: Simulating fluid physical phenomena: part B



*Lectures 3 and 4 had to be swapped this year

OpenFOAM

Part 1

How to do simulations?

Part 2

How do solvers work?

Part 1

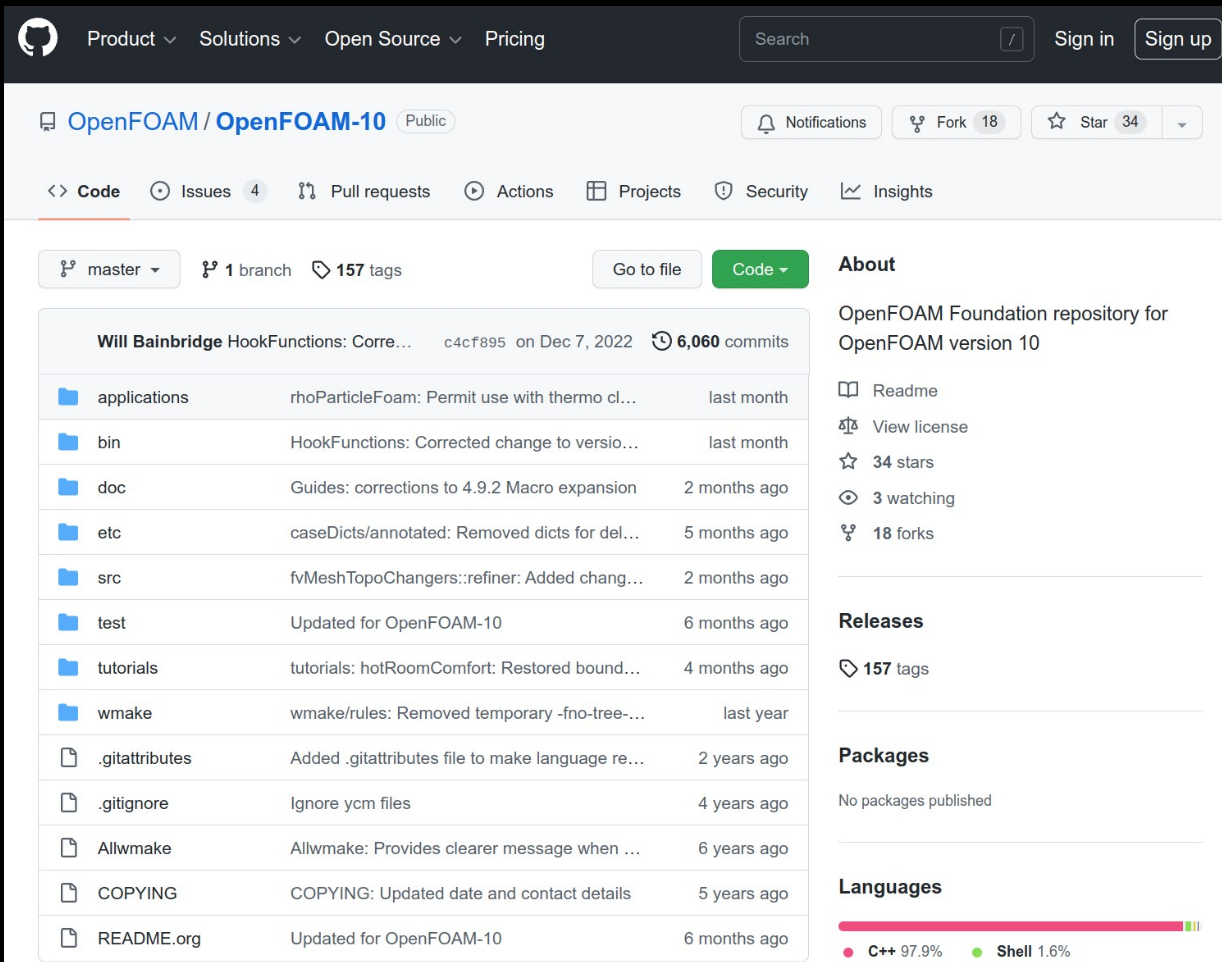
How to do simulations?

Terminology

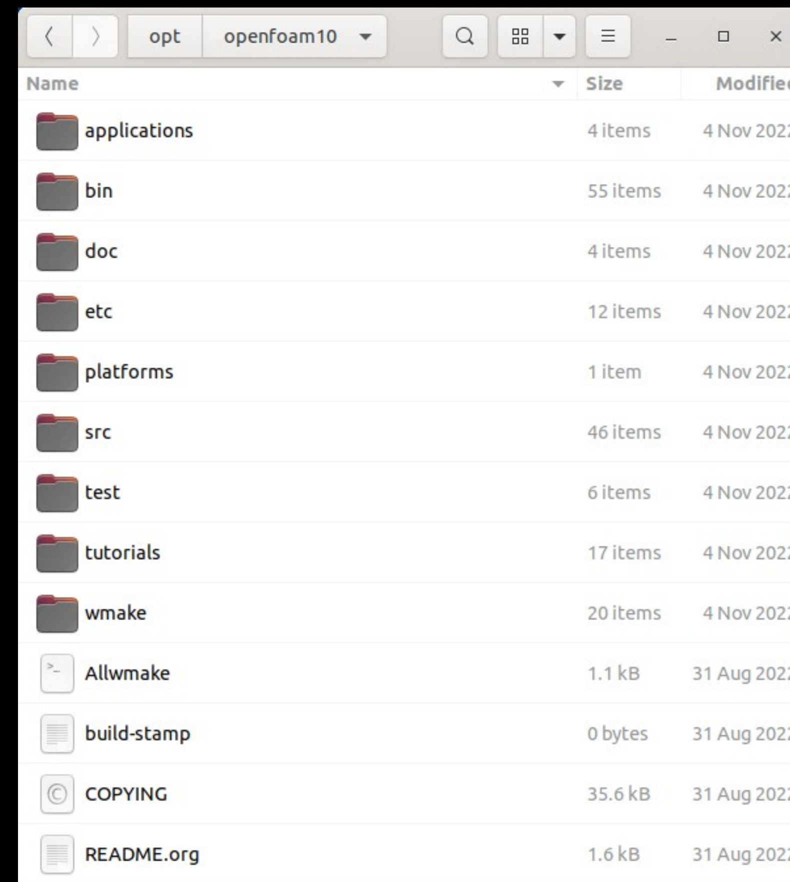
Solver	Executable application designed to solve a specific problem in fluid or continuum mechanics (**scalarTransportFoam**, **icoFoam**,...)
Utility	Executable application, designed to perform tasks that involve data manipulation (**blockMesh**, **postProcess**, **foamListTimes**,...)
Library	Precompiled C++ libraries that are dynamically linked to the solvers and utilities. There are separate libraries containing e.g. turbulence models, post-processing functions, etc.
Case folder	Folder containing OpenFOAM dictionaries, describing particular problem (case)
Dictionary	A file/entity that contains data entries in the format understandable by OpenFOAM (**0/U**, **system/controlDict**, ...)
Function objects	Tools to ease workflow configurations and enhance workflows by producing additional user-requested data

OpenFOAM 10 – open-source CFD library

github.com/OpenFOAM/OpenFOAM-10



/opt/openfoam10/



OpenFOAM

Pre-processing

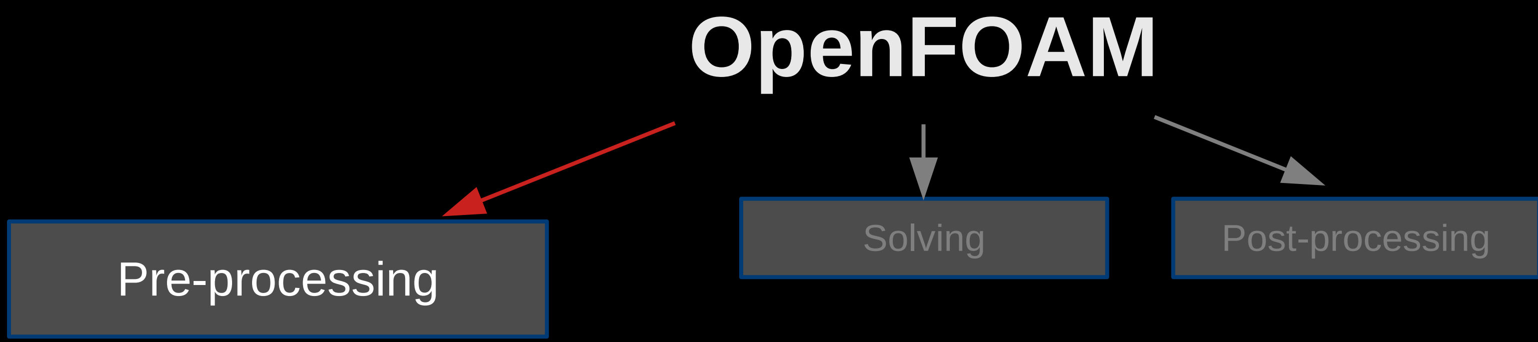
1. Choose solver
2. Set up case dictionaries
3. Generate mesh
4. (optional) Adding extra terms and constraints
5. (optional) Set up function objects

Solving

1. Running solvers
2. Run-time controls

Post-processing

1. Post-processing and sampling
2. Visualization



1. Choosing solver

Example list of solvers	
scalarTransportFoam	**Convection-diffusion** of a **passive scalar**
icoFoam	Simplified **PISO** solver for **incompressible**, **laminar** flow of **Newtonian** fluids
simpleFoam	**Incompressible** fluid flow solver, using **SIMPLE** algorithm. Used for **steady-state** flows
pisoFoam	**Incompressible** fluid flow solver, using **PISO** algorithm. Used for **transient** flows
pimpleFoam	**Incompressible** fluid flow solver, using **PIMPLE** algorithm. Used for **transient** flows
rhoPimpleFoam	**Compressible** fluid flow solver, using **PIMPLE** algorithm
buoyantPimpleFoam	**Compressible** fluid flow solver with buoyancy modeling, using **PIMPLE** algorithm
reactingFoam	**Compressible** reacting fluid flow solver with **chemistry reactions**, using **PIMPLE** algorithm
interFoam	**Two incompressible fluids** flow solver using "**volume of fluid**" model, using **PIMPLE** algorithm
interPhaseChangeFoam	**Two incompressible** fluids flow solvers using "**volume of fluid**" model **with phase-change**, using **PIMPLE** algorithm
multiphaseEulerFoam	System of any **number of compressible fluid phases** flow solver, using **PIMPLE** algorithm

https://doc.cfd.direct/openfoam/user-guide-v10/standard-solvers

OpenFOAM

Pre-processing | Solving | Post-processing

2. Setting up case dictionaries

Usually starts with copying tutorial case

```
<Case folder >
├─ 0
│  ├─ p
│  ├─ T
│  └─ U
├─ constant
│  ├─ polyMesh
│  │  └─ ...
│  ├─ momentumTransport
│  └─ physicalProperties
└─ system
   ├─ blockMeshDict
   ├─ controlDict
   ├─ fvSchemes
   └─ fvSolution
```

- p, T, U → Initial and boundary conditions for fields
- polyMesh → Mesh files
- momentumTransport → Turbulence models
- physicalProperties → Physical constants, equations of state, etc.
- blockMeshDict → Instructions for blockMesh utility
- controlDict → Time stepping, data writing controls, additional function objects setup
- fvSchemes → Discretization schemes
- fvSolution → Matrix solvers and pressure velocity coupling algorithms setup

OpenFOAM

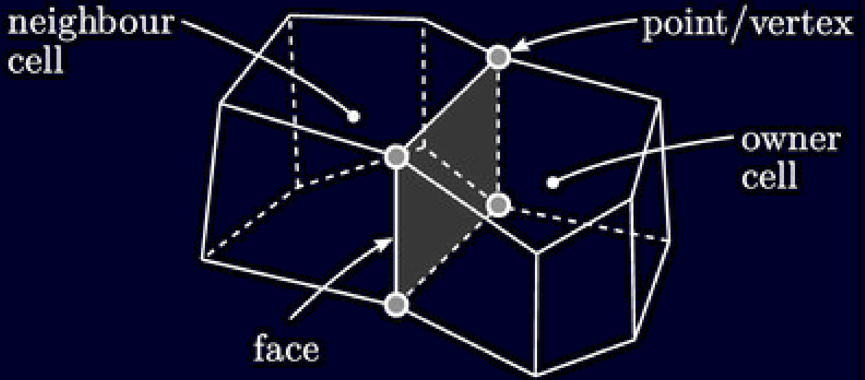
- **Pre-processing**
- Solving
- Post-processing

3. Mesh generation

< polyMesh >

- boundary → Specifications of boundary patches
- cellZones
- faces → Ordered list of points numbers, defining faces
- faceZones
- neighbour, owner → Specifications, which face belongs to which cell
- points → Ordered list of point coordinate triplets
- pointZones

Main mesh generation utilities	
blockMesh	for simple structured meshes
snappyHexMesh	for meshing complex geometries (using stl, obj, vtk, ...)
starToFoam fluentMeshToFoam gmshToFoam ...	import 3rd party mesh formats



https://github.com/OpenFOAM/OpenFOAM-10/tree/master/applications/utilities/mesh

https://doc.cfd.direct/notes/cfd-general-principles/computational-mesh

OpenFOAM

Pre-processing | Solving | Post-processing

4. Adding extra terms and constraints

constant/fvModels

```
energySource
{
    type            heatSource;

    selectionMode   all;

    q               1e7;
}
```

system/fvConstraints

```
limitp
{
    type       limitPressure;

    min        0.8e5;
    max        1.2e5;
}
```

Commands to list available options:

```
scalarTransportFoam -listFvModels
scalarTransportFoam -listFvConstraints
```

Command to show info and find usage examples:

```
foamInfo -a heatSource
foamInfo -a limitPressure
```

https://github.com/OpenFOAM/OpenFOAM-10/tree/master/src/fvModels
https://github.com/OpenFOAM/OpenFOAM-10/tree/master/src/fvConstraints

5. Set up function objects

system/controlDict

```
functions
{
    probes
    {
        type            probes;
        libs            ("libsampling.so");
        writeControl    timeStep;
        writeInterval   1;

        fields
        (
            p
        );

        probeLocations
        (
            (0.0254 0.0253 0)
            (0.0508 0.0253 0)
            (0.0762 0.0253 0)
            (0.1016 0.0253 0)
            (0.127 0.0253 0)
            (0.1524 0.0253 0)
            (0.1778 0.0253 0)
        );

    }

    #includeFunc fieldAverage(U, p, prime2Mean = yes)

    #includeFunc scalarTransport
}
```

```
scalarTransportFoam -listFunctionObjects
FoamInfo -a probes
```

https://doc.cfd.direct/openfoam/user-guide-v10/post-processing-cli

OpenFOAM

Pre-processing | **Solving** | Post-processing

1. Running solver

a) On a single core

```
scalarTransportFoam
```

b) On multiple cores (requires *system/decomposeParDict*)

```
decomposePar
mpirun -np 4 scalarTransportFoam -parallel
reconstructPar
```

c) On supercomputing cluster: using schedulers

2. Run-time controls

If you set in *system/controlDict*:

```
runTimeModifiable true;
```

You can modify entries in dictionaries during run-time.
E.g. adjust tolerances, change write times, change time step etc.

You can also track residuals of your simulation using foamMonitor utility.

You can stop the simulation and write the fields immediately by setting in *system/controlDict*:

```
stopAt          writeNow;
```

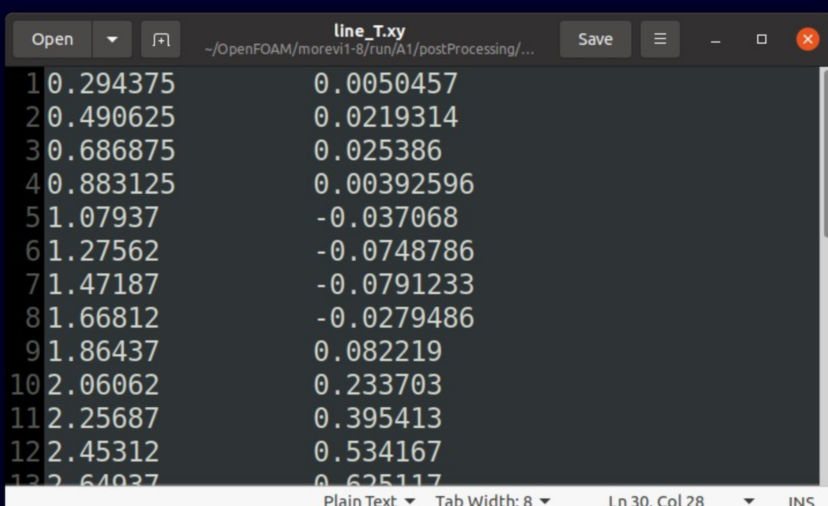
OpenFOAM

Pre-processing | Solving | **Post-processing**

1. Post-processing and sampling

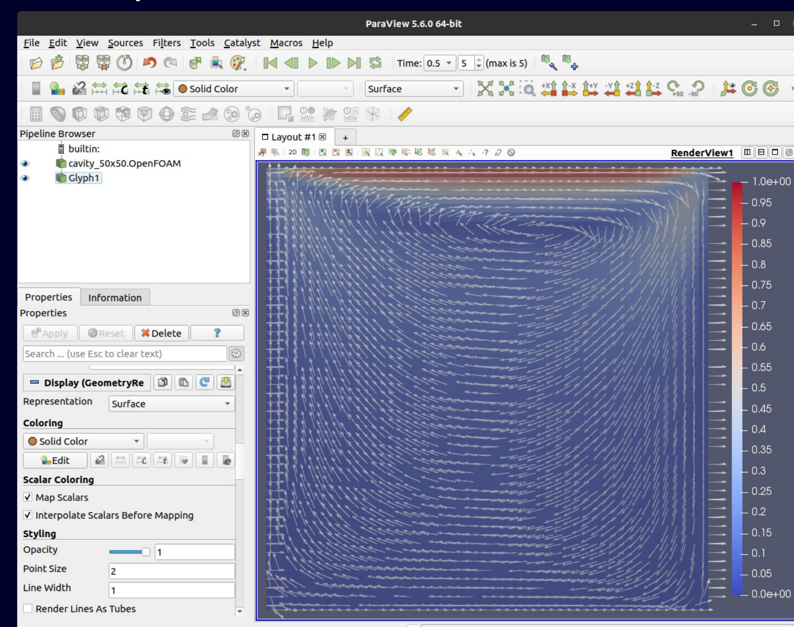
Utility postProcess can execute functionObjects after simulation is finished.
E.g. sample data over a line (requires file *system/singleGraph*):

```
postProcess -func singleGraph
```



2. Visualization

Use paraFoam to visualize fields, make videos, renders, plots, etc.



Or export the data to 3rd part format, e.g. VTK:

```
foamToVTK
```

Part 2

How do solvers work?

Main question:

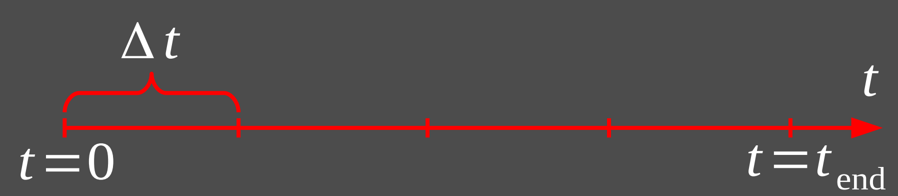
How does OpenFOAM solver proceed from one time step to the next one?

```
Time = 0.96

smoothSolver:  Solving for T, Initial residual = 0.0190827, Final residual = 2.62384e-06, No Iterations 2
Time = 0.965

smoothSolver:  Solving for T, Initial residual = 0.0190088, Final residual = 2.47331e-06, No Iterations 2
Time = 0.97
```

Matrix equation solver logs. But where does the matrix come from?

Time-marching

1. Equations are integrated over some time step.
2. New values of fields (u, p, T…) are obtained and then used as initial values for next iteration.



Scalar transport equation

$$\underbrace{\frac{\partial c}{\partial t}}_{\text{Temporal derivative}} + \underbrace{\vec{u} \cdot \nabla c}_{\text{Convection}} - \underbrace{\alpha \nabla^2 c}_{\text{Diffusion}} = \underbrace{\dot{\omega}_c}_{\text{Source}}$$

Initial condition

$$c\left(t = t_0\right) = c^{t_0}$$

Time step

$$\Delta t = t_1 - t_0$$

$$c^{t_1} - ?$$

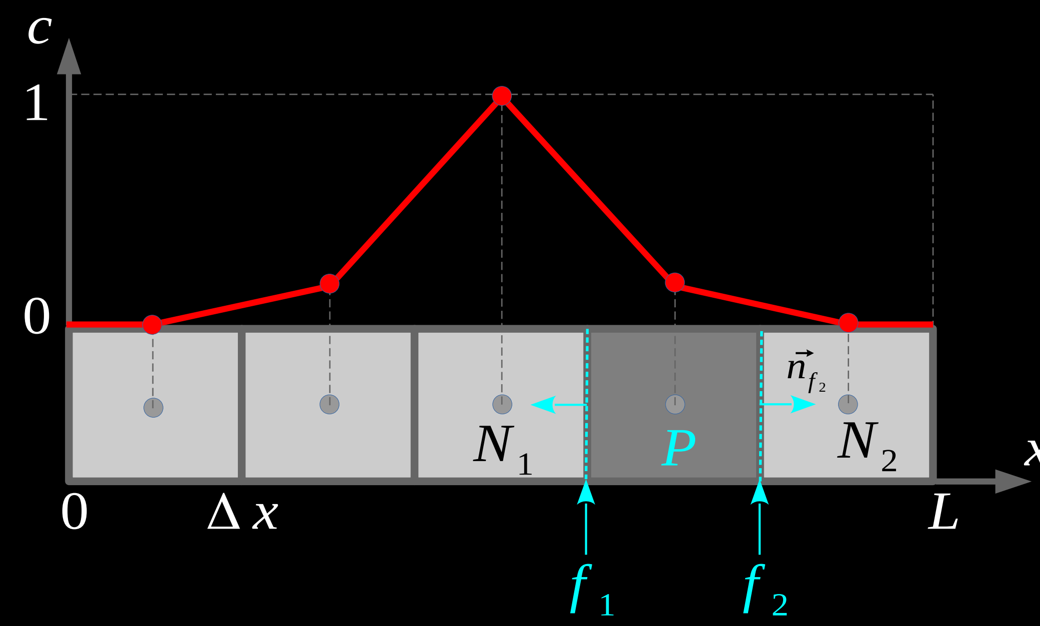
N – number of cells

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} \cdot \begin{pmatrix} c_1^{t_1} \\ c_2^{t_1} \\ \vdots \\ c_N^{t_1} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

1d convection of a Gaussian

Goal: calculate field c after Δt

$u = const > 0\ ,\ t_1 = t_0 + \Delta t\ ,$ **5 cells**



$$\frac{\partial c}{\partial t} + \nabla\cdot(\vec{u}\,c) = 0$$

Integrate over cell **P**

$$\frac{1}{V_P}\int_{V_P}\nabla\cdot(\vec{u}\,c)\,dV = \frac{1}{V_P}\int_{A_P}(\vec{u}\,c)\,\vec{n}\,dA_P \approx \frac{1}{V_P}\sum_f(\vec{u}_f c_f)\cdot\vec{n}_f A_f = \frac{u_{f_2}c_{f_2} - u_{f_1}c_{f_1}}{\Delta x} = u\,\frac{c_{f_2} - c_{f_1}}{\Delta x}$$

Gauss theorem

Finite number of faces

1d uniform mesh

Constant velocity

Implicit Euler temporal discretization:

$$\frac{c_P^{t_1} - c_P^{t_0}}{\Delta t} + u\,\frac{c_{f_2}^{t_1} - c_{f_1}^{t_1}}{\Delta x} = 0$$

Putting constant coefficients to a_i

$$\boxed{a_P c_P^{t_1} + \sum_{N_i} a_{N_i} c_{N_i}^{t_1} = b_P}$$

Interpolating c_{f1}, c_{f2}

linear

$$c_{f_1} = \frac{c_{N_1} + c_P}{2}$$

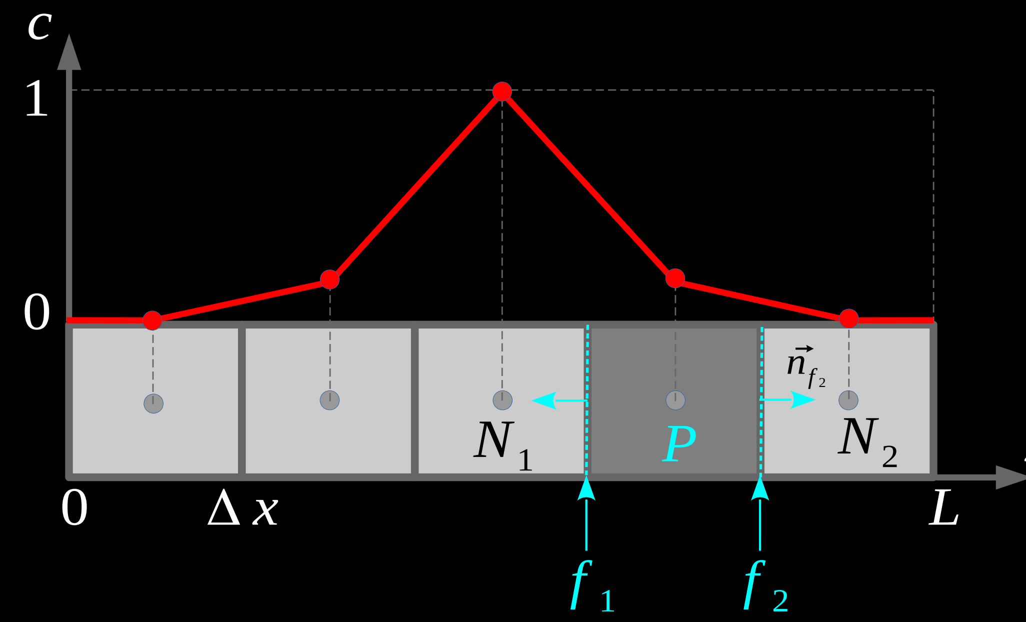
upwind

$$c_{f_1} = c_P$$

We have 1 equation for each of 5 cells. Total:

- 5 equations
- 5 unknowns

$$\boxed{\underbrace{a_P c_P^{t_1}}_{\text{Diagonal Components (“owner”)}} + \underbrace{\sum_{N_i} a_{N_i} c_{N_i}^{t_1}}_{\text{Off-diagonal Components (“neighbor”)}} = b}$$



$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}}_{\text{Known coefficient matrix}} \cdot \underbrace{\begin{pmatrix} c_1^{t_1} \\ c_2^{t_1} \\ c_3^{t_1} \\ c_4^{t_1} \\ c_5^{t_1} \end{pmatrix}}_{\text{Field vector to be solved}} = \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}}_{\text{Known vector of explicit terms}}$$

$$\boxed{\mathbf{A}\vec{c}^{\,t_1} = \vec{b}}$$

Use some matrix solver to obtain field *c* at time t_1

scalarTransportFoam source code

Located in *applications/solvers/basic/scalarTransportFoam/*

```
Info<< "\nCalculating scalar transport\n" << endl;

#include "CourantNo.H"

while (simple.loop(runTime))
{
    Info<< "Time = " << runTime.userTimeName() << nl << endl;

    fvModels.correct();

    while (simple.correctNonOrthogonal())
    {
        fvScalarMatrix TEqn
        (
            fvm::ddt(T)
          + fvm::div(phi, T)
          - fvm::laplacian(DT, T)
         ==
            fvModels.source(T)
        );

        TEqn.relax();
        fvConstraints.constrain(TEqn);
        TEqn.solve();
        fvConstraints.constrain(T);
    }

    runTime.write();
}

Info<< "End\n" << endl;
```

Loop over time steps, as defined in *system/controlDict*

$$\frac{\partial c}{\partial t}+\vec{u}\cdot\nabla c-\alpha\nabla^2 c=\dot{\omega}_c$$

Matrix equation is constructed here

$$A\vec{c}^{\,t_1}=\vec{b}$$

Call matrix solver, defined in *system/fvSolution*

Write data, if current time step fits writeTime defined in *system/controlDict*

```
Time = 0.96

smoothSolver:  Solving for T, Initial residual = 0.0190827, Final residual = 2.62384e-06, No Iterations 2
Time = 0.965

smoothSolver:  Solving for T, Initial residual = 0.0190088, Final residual = 2.47331e-06, No Iterations 2
Time = 0.97
```

Discretization schemes

Located in *system/fvSchemes*

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,T)      Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}
```

$\frac{\partial}{\partial t}$

Temporal discretization schemes: Euler (1st ord.), backward (2nd ord.), ...

∇

In the most cases, linear works perfectly well here. In our applications we discretize pressure gradient here.

$\nabla \cdot$

div(phi,...) are the most important schemes usually! Here we discretize convection term.
upwind (1st ord.), linear (2nd ord.), limitedLinear, Gamma and vanLeer are probably the most common choices

∇^2

The keyword “linear” refers to interpolation scheme, where linear is usually enough. The second keyword in surface normal gradient scheme, which usually is either orthogonal or corrected (for meshes with orthogonality)

c_f

Cell to face interpolations of values. Used in the interpolation of velocity to face centers for the calculation of flux

∇_n

Component of gradient normal to a cell face

Check what schemes are used in tutorials:

```
foamSearch -c $FOAM_TUTORIALS fvSchemes "divSchemes/div(phi,U)"
```

Flux limiting schemes

$$c_{f_1} = c_{f_1}^{low} - \phi(r_{N_1})(c_{f_1}^{low} - c_{f_1}^{high})$$

$$c_{f_2} = c_{f_2}^{low} - \phi(r_P)(c_{f_2}^{low} - c_{f_2}^{high})$$

$c_{f_1}^{low}$ - low resolution flux (upwind)

$c_{f_1}^{high}$ - high resolution flux (linear)

$\phi(r_P)$ - **flux limiter function**

$$r_P = \frac{c_P - c_{N_1}}{c_{N_2} - c_P}$$

- ratio of successive gradients

Recall:

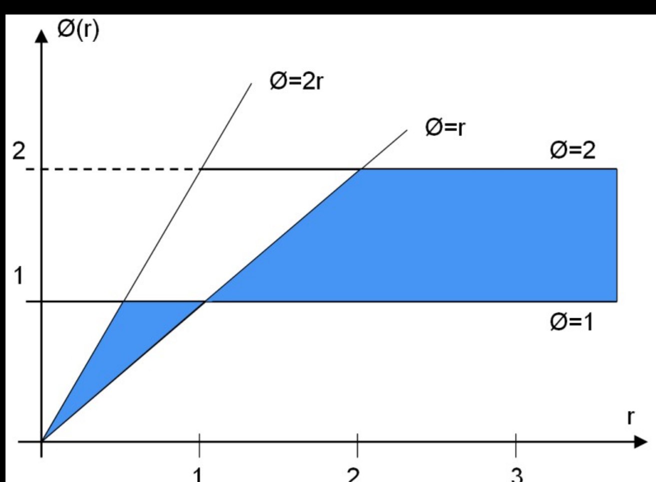
Interpolating c_f

linear

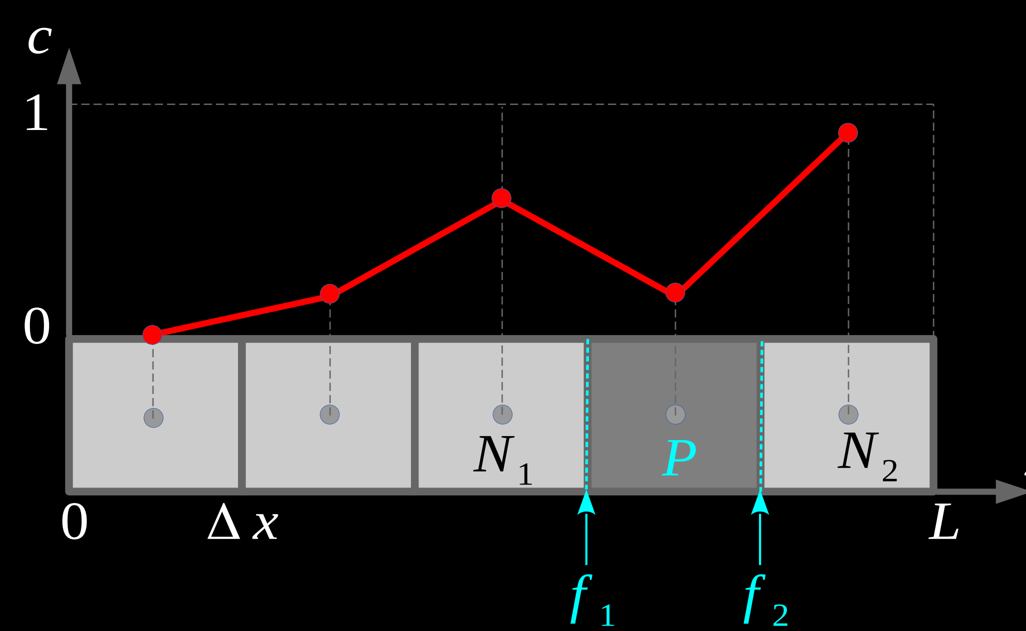
$$c_{f_1} = \frac{c_{N_1} + c_P}{2}$$

upwind

$$c_{f_1} = c_P$$

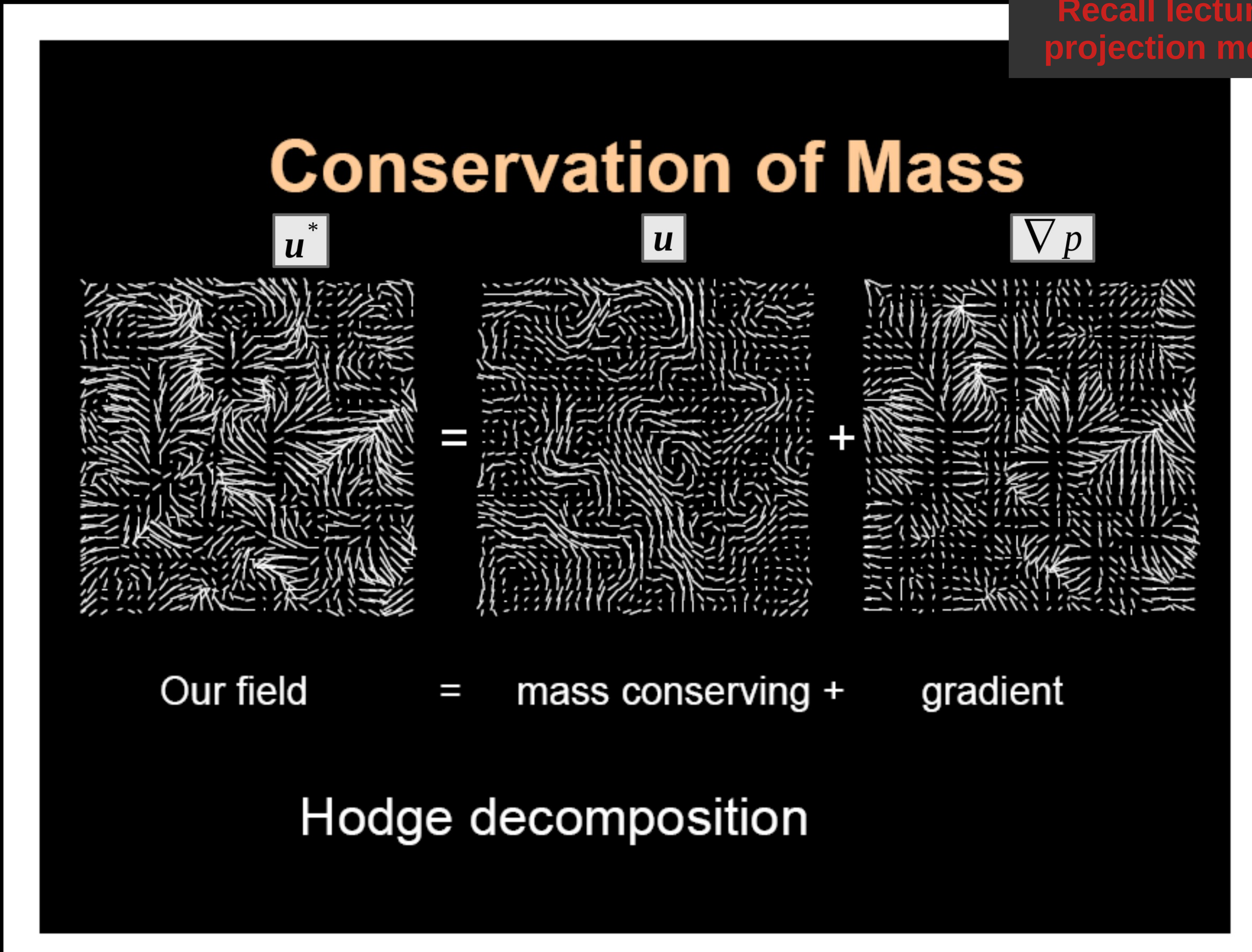


Admissible limiter region for second-order TVD schemes (Sweby, 1984)



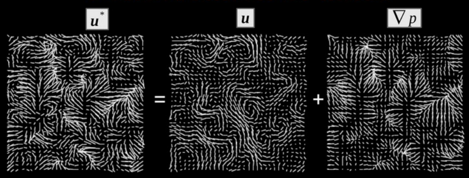
Vector fields can be divided into two parts via "Helmholtz-Hodge" decomposition

Recall lecture 3:
projection method



Pressure-velocity coupling algorithms

Conservation of Mass



Hodge decomposition

Start with Navier-Stokes equations:

$$\begin{cases} \nabla \cdot \vec{u} = 0 \\ \dfrac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \nabla \cdot (\nu \nabla \vec{u}) = -\nabla p \end{cases}$$

Semi-discretize LHS (velocity and pressure gradient are unknown):

$$M \vec{u} = \vec{b} - \nabla p$$

A Diagonal matrix

Off-diagonal part of M, evaluated with u from prev. solution

$$H = A \vec{u} - M \vec{u} + \vec{b}$$

$$A \vec{u} - H = -\nabla p$$

$$\vec{u} = A^{-1} H - A^{-1} \nabla p$$

Substitute to the continuity equation

Poisson equation

$$\nabla \cdot (A^{-1} \nabla p) = \nabla \cdot (A^{-1} H) \rightarrow p$$

Simplified solution scheme

Initial guess

1. Momentum predictor
Solve the momentum equation for the velocity field. This velocity field does not satisfy the continuity equation.

$$M\vec{u} = -\nabla p$$

2. Explicit part evaluation
Use the velocity to calculate explicit part H

$$H = A\vec{u} - M\vec{u} + \vec{b}$$

3. Pressure-corrector
Solve the Poisson equation for the pressure field.

$$\nabla \cdot (A^{-1} \nabla p) = \nabla \cdot (A^{-1} H)$$

4. Explicit velocity calculation
Use the pressure field to calculate new velocity field, satisfying the continuity equation. Pressure field is not corrected anymore

$$\vec{u} = A^{-1} H - A^{-1} \nabla p$$

Outer corrector (**SIMPLE** loop)

non-orthogonal corrector

Inner corrector (**PISO** loop)

OpenFOAM code (icoFoam)

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);
```

```
if (simple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));

    fvOptions.correct(U);
}
```

```
volScalarField rAU(1.0/UEqn.A());
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
surfaceScalarField phiHbyA
(
    "phiHbyA",
    fvc::flux(HbyA)
  + fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
);
```

```
// Non-orthogonal pressure corrector loop
while (simple.correctNonOrthogonal())
{
    fvScalarMatrix pEqn
    (
        fvm::laplacian(rAtU(), p) == fvc::div(phiHbyA)
    );

    pEqn.setReference(pRefCell, pRefValue);

    pEqn.solve();

    if (simple.finalNonOrthogonalIter())
    {
        phi = phiHbyA - pEqn.flux();
    }
}
```

```
U = HbyA - rAU*fvc::grad(p);
```

Pressure-velocity coupling algorithms

Parameters are located in *system/fvSolution*

```
PIMPLE
{
    momentumPredictor no;
    nOuterCorrectors  1;
    nCorrectors      2;
    nNonOrthogonalCorrectors 0;
}
```

momentumPredictor	switch controlling the momentum predictor. Can be set to “off” for some flows, including low Reynolds number and multiphase.
nOuterCorrectors	sets the number of **outer** correctors, number of loops over the entire system of equations within on time step, representing the total number of times the system is solved; must be ≥ 1 and is typically set to 1, replicating the PISO algorithm. If you experience pressure fluctuations, increasing this number can help.
nCorrectors	sets the number of **inner** correctors, i.e. times the algorithm solves the pressure equation and momentum corrector in each step; typically set to 2 or 3.
nNonOrthogonalCorrectors	specifies repeated solutions of the pressure equation, used to update the explicit **non-orthogonal** correction; typically set to 0 for orthogonal meshes and ≥ 1 for meshes with non-orthogonality

Further reading:

SIMPLE: https://openfoamwiki.net/index.php/The_SIMPLE_algorithm_in_OpenFOAM

PISO: https://openfoamwiki.net/index.php/OpenFOAM_guide/The_PISO_algorithm_in_OpenFOAM

PIMPLE: https://openfoamwiki.net/index.php/OpenFOAM_guide/The_PIMPLE_algorithm_in_OpenFOAM

Matrix solver setup

Located in *system/fvSolution*

$$A\vec{c}=\vec{b} \rightarrow \vec{c}$$

$$\text{residual}_i = A\vec{c}_{\text{guess}_i} - \vec{b}$$

$$\text{relTol}_i = \frac{\text{residual}_i}{\text{residual}_0}$$

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}
```

Usually:
- PCG with DIC preconditioner
- GAMG with GaussSeidel smoother

Tolerance for the final **inner corrector** step. Usually tolerance is tightened here and relTol=0

Solver selection here depends on your grid parameters, which determines the filling of your matrix.
PBiCGStab with DILU preconditioner is quite robust

```
Time = 0.295

smoothSolver:  Solving for Ux, Initial residual = 0.00336414, Final residual = 4.87212e-06, No Iterations 2
smoothSolver:  Solving for Uy, Initial residual = 0.00395571, Final residual = 6.13208e-06, No Iterations 2
DICPCG:  Solving for p, Initial residual = 0.00198918, Final residual = 9.51425e-05, No Iterations 27
time step continuity errors : sum local = 1.56381e-08, global = 5.72285e-20, cumulative = 2.48268e-20
DICPCG:  Solving for p, Initial residual = 0.00061602, Final residual = 9.64995e-07, No Iterations 65
time step continuity errors : sum local = 1.49115e-10, global = 2.02111e-20, cumulative = 4.50379e-20
ExecutionTime = 0.31 s  ClockTime = 1 s
```

Further reading

User guide:

Online: https://doc.cfd.direct/openfoam/user-guide-v10/index

Offline: /opt/openfoam10/doc/Guides/OpenFOAMUserGuide-A4.pdf

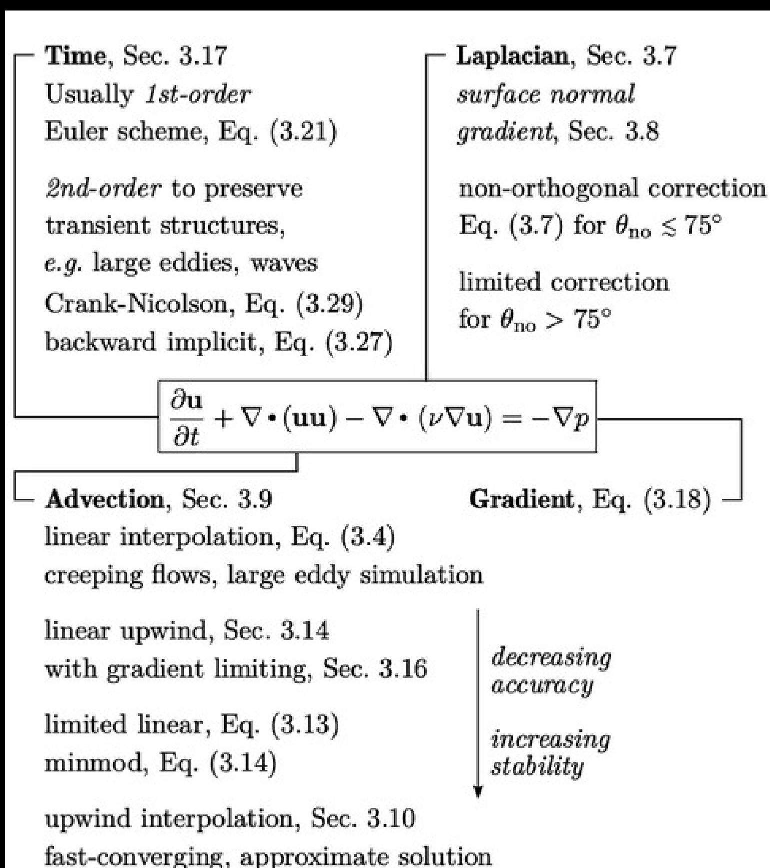
Programmers Guide:

https://sourceforge.net/projects/openfoam/files/v2112/ProgrammersGuide.pdf/download

CFD textbook by authors of OpenFOAM (free web version):

https://doc.cfd.direct/notes/cfd-general-principles/

Textbook



User guide

Tutorial relevant to **HW2**

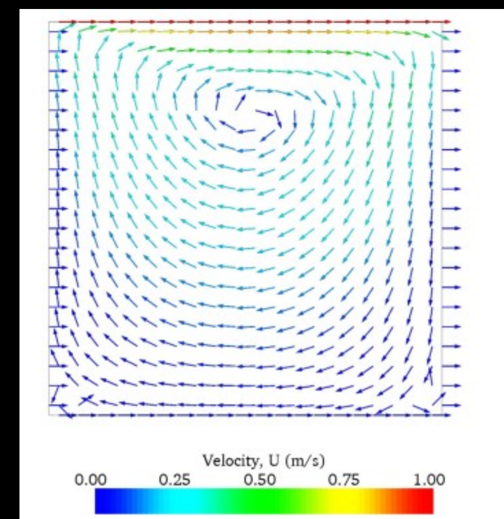


Figure 2.8: Velocities in the cavity case.

Programmer's guide

Operation	Comment	Mathematical Description	Description in OpenFOAM
Addition		$\mathbf{a}+\mathbf{b}$	`a + b`
Subtraction		$\mathbf{a}-\mathbf{b}$	`a - b`
Scalar multiplication		$s\mathbf{a}$	`s * a`
Scalar division		$\mathbf{a}/s$	`a / s`
Outer product	rank $\mathbf{a}, \mathbf{b} >= 1$	$\mathbf{ab}$	`a * b`
Inner product	rank $\mathbf{a}, \mathbf{b} >= 1$	$\mathbf{a}\bullet\mathbf{b}$	`a & b`