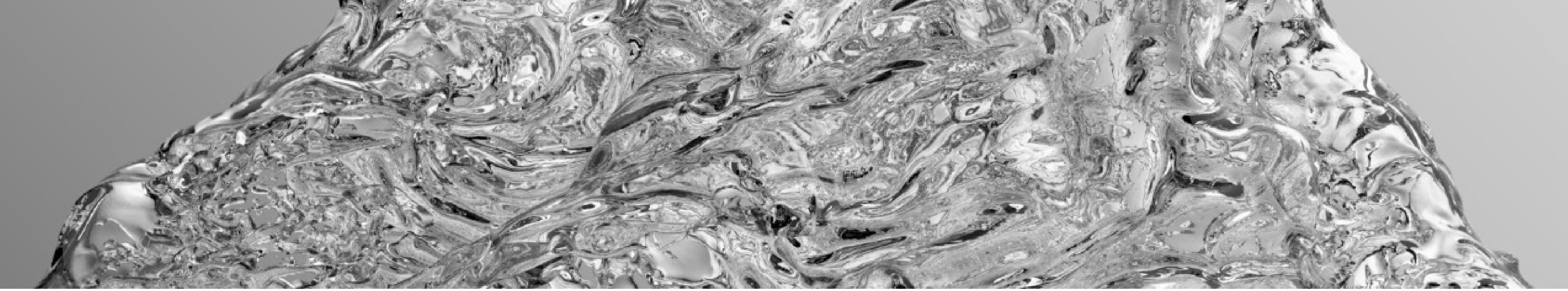


AAE-E2001 Computational Fluid Dynamics

Lecture 4: OpenFOAM code and structure

M.Sc. Ilya Morev

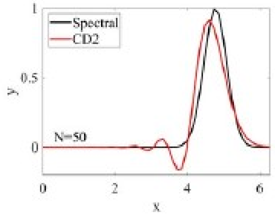
February 5th 2024

Aalto University, School of Engineering

Lecture 1: Linear PDEs and finite difference method

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

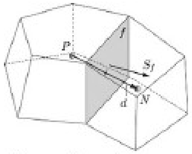
$$\boxed{\frac{\partial T}{\partial x}\approx\frac{T_{i+1}-T_{i-1}}{2\Delta x}}$$



Lecture 2: Gauss' theorem and finite volume method

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

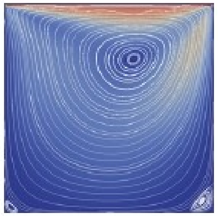
$$\boxed{\int_{\partial\Omega}\boldsymbol{u}\cdot\boldsymbol{n}\,dS\approx\Sigma_f\boldsymbol{u}_f\cdot\boldsymbol{n}_f\,dS_f}$$



Lecture 3: Navier-Stokes equation and pressure

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

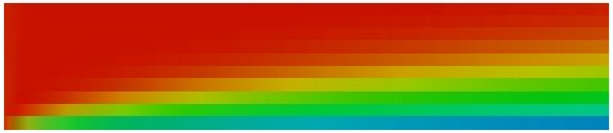
$$\boxed{-\nabla^2 p=\nabla\cdot\nabla\cdot\boldsymbol{uu}}$$

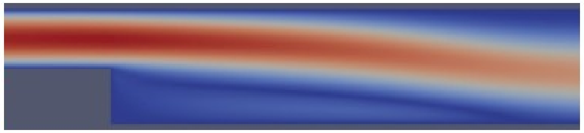


Lecture 4: OpenFOAM code and structure

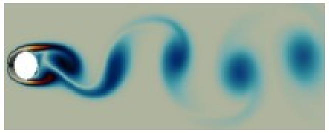
```
fvVectorMatrix UEqn
(
      fvm::ddt(U)
    + fvm::div(phi, U)
    - fvm::laplacian(nu, U)
);
```

Lecture 5: Simulating fluid physical phenomena: part A





Lecture 6: Simulating fluid physical phenomena: part B



OpenFOAM

Part 1

How to do simulations?

Part 2

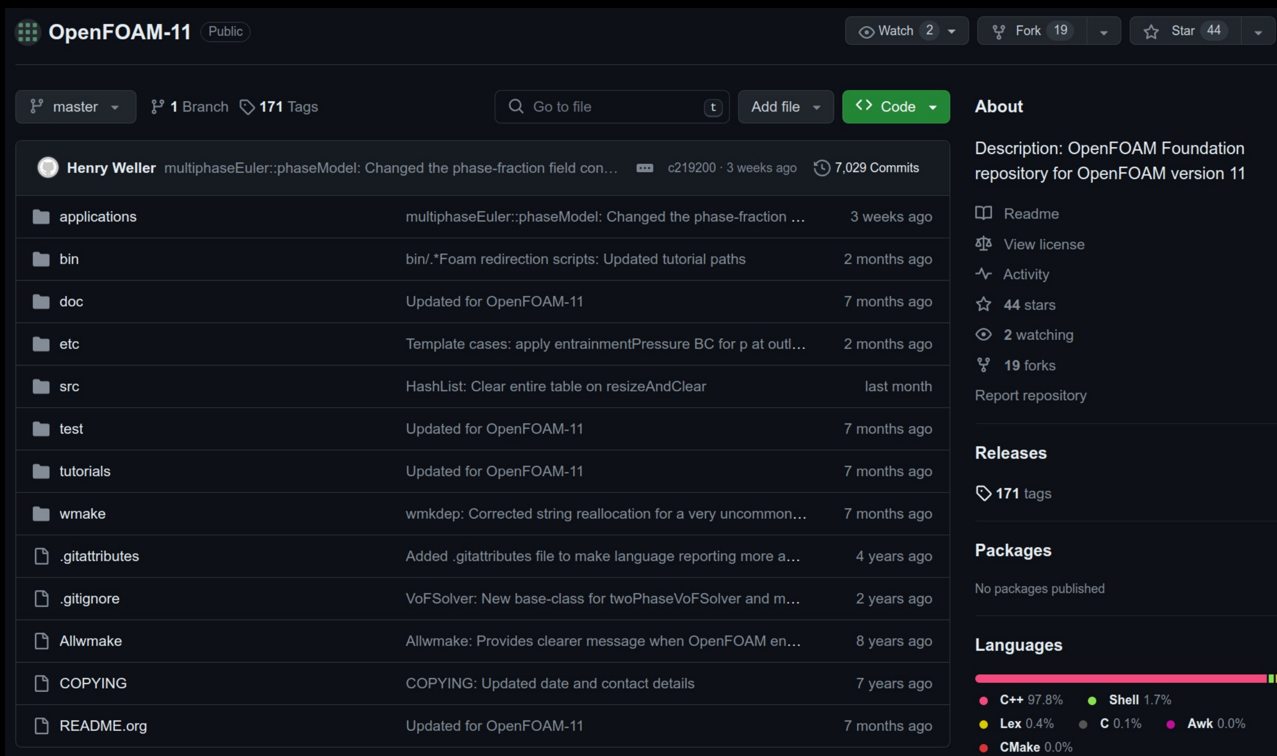
How do solvers work?

Part 1

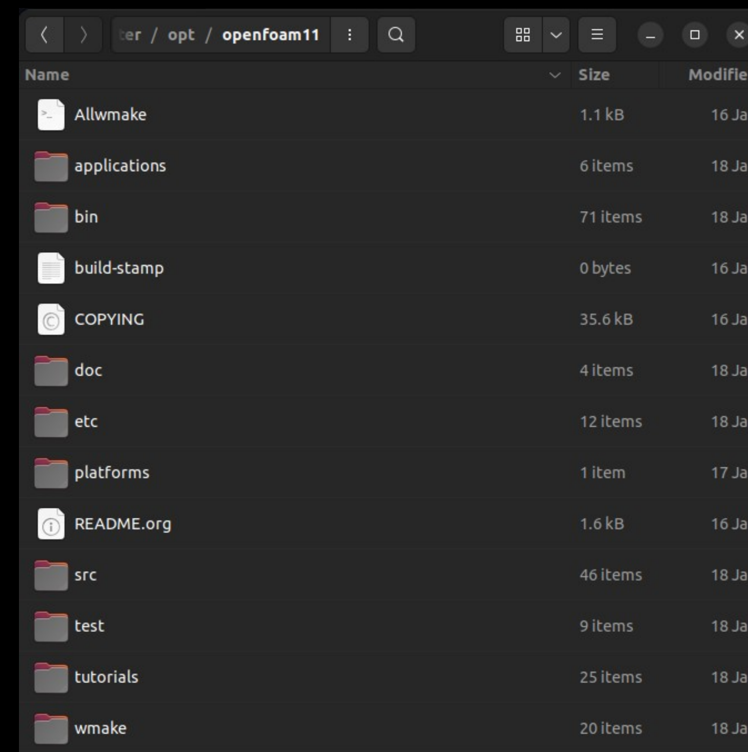
How to do simulations?

OpenFOAM 11 – open-source CFD library

github.com/OpenFOAM/OpenFOAM-11



/opt/openfoam11/



Useful utilities

foamToC	"Table of contents". Lists available models/libraries/objects
	`foamToC -fvModels`
	`foamToC -fvConstraints`
	`foamToC -functionObjects`

foamInfo	Print information about models, objects, utilities, etc.
	`foamInfo incompressibleFluid`
	`foamInfo -a heatSource`

foamSearch	Searches a directory for files with some name and extracts specified entries
	`foamSearch $FOAM_TUTORIALS physicalProperties viscosityModel`
	`foamSearch -c $FOAM_TUTORIALS fvSchemes ddtSchemes/default`

foamGet	Copies example dictionaries from /opt/openfoam11*/etc/caseDicts*
	`foamGet graphCell`
	`foamGet decomposeParDict`

foamDictionary	Prints or changes dictionary entries
	`foamDictionary -entry endTime system/controlDict`
	`foamDictionary -entry "divSchemes/div(phi,T)" -set "Gauss linear" system/fvSchemes`

foamListTimes	Prints or removes time folders
	`foamListTimes -time ":0.4"`
	`foamListTimes -time "0.2,0.4" -rm`

checkMesh	Checks validity of mesh and prints some statistics
patchSummary	Print all boundary condition information for the case

Note that any OpenFOAM executable has a *-help* option:

```
blockMesh -help
foamToC -help
```

OpenFOAM

Pre-processing	Solving	Post-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	1. Running solvers 2. Run-time controls	1. Post-processing and sampling 2. Visualization

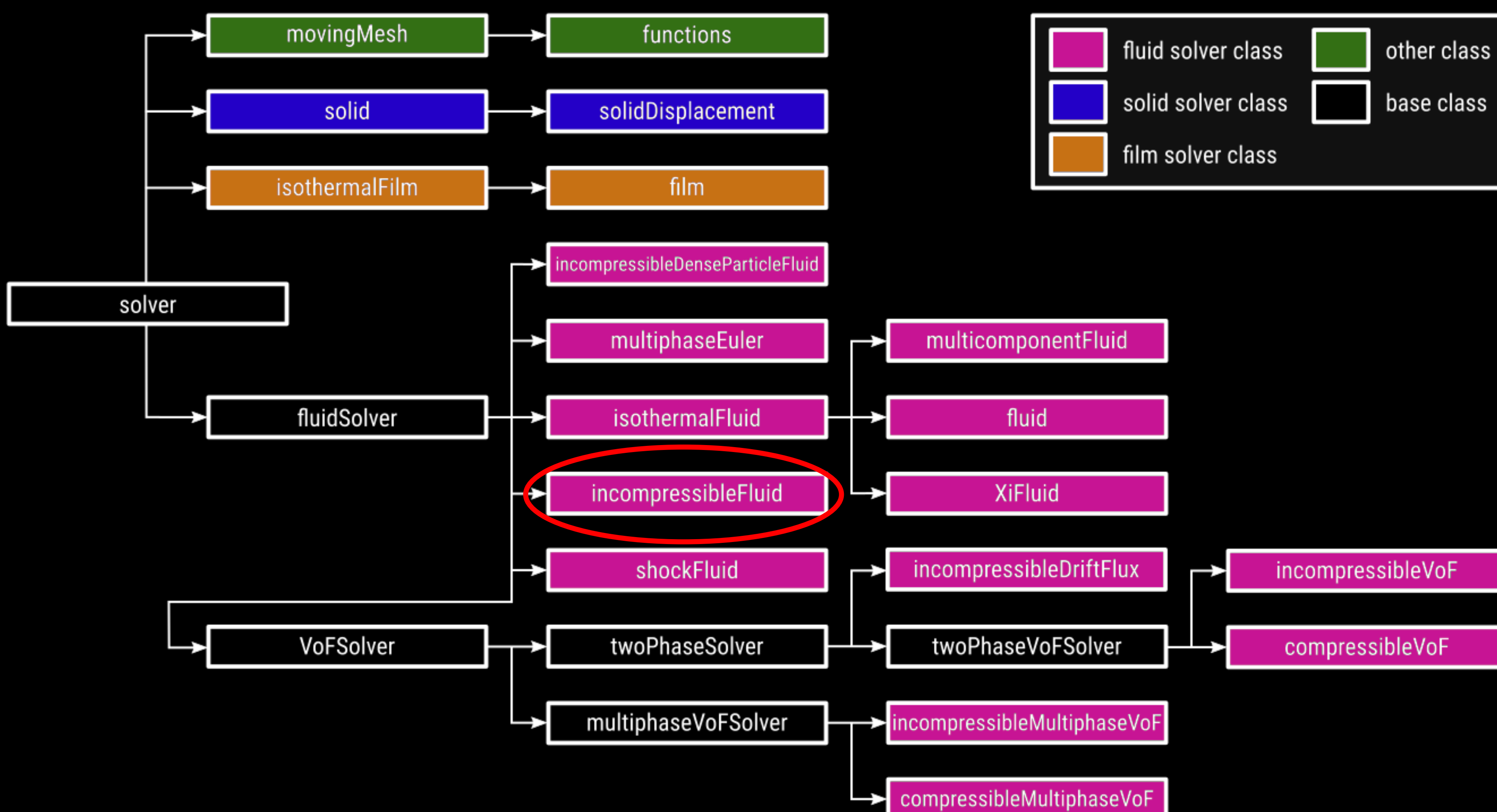
OpenFOAM

Pre-processing

Solving

Post-processing

1. Choosing solver



https://doc.cfd.direct/openfoam/user-guide-v11/solvers-modules

OpenFOAM

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

2. Setting up case dictionaries

Usually starts with copying tutorial case

- *<Case folder >*
  - 0
    - p
    - T
    - U

    → Initial and boundary conditions for fields
  - constant
    - polyMesh → Mesh files (generated by blockMesh)
      - ...
    - momentumTransport → Turbulence models
    - physicalProperties → Physical constants, equations of state, etc.
  - system
    - 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

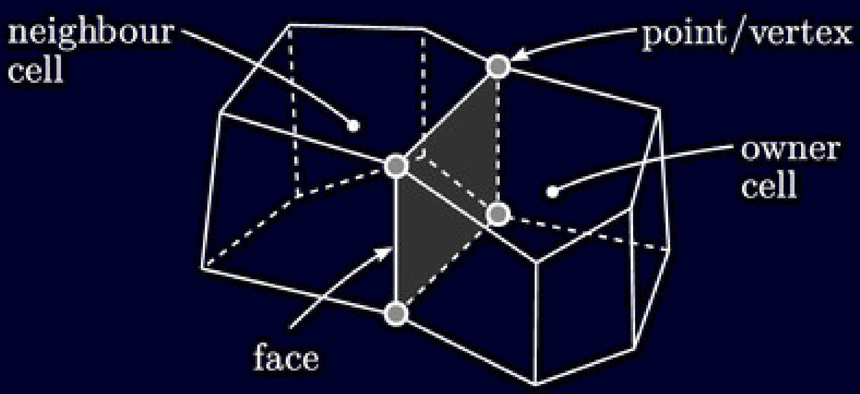
Mesh format

< *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

https://doc.cfd.direct/openfoam/user-guide-v11/mesh-files

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://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:

```
foamToC -fvModels
foamToC -fvConstraints
```

Command to show info and find usage examples:

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

https://github.com/OpenFOAM/OpenFOAM-11/tree/master/src/fvModels/derived
https://github.com/OpenFOAM/OpenFOAM-11/tree/master/src/fvConstraints

5. Set up function objects

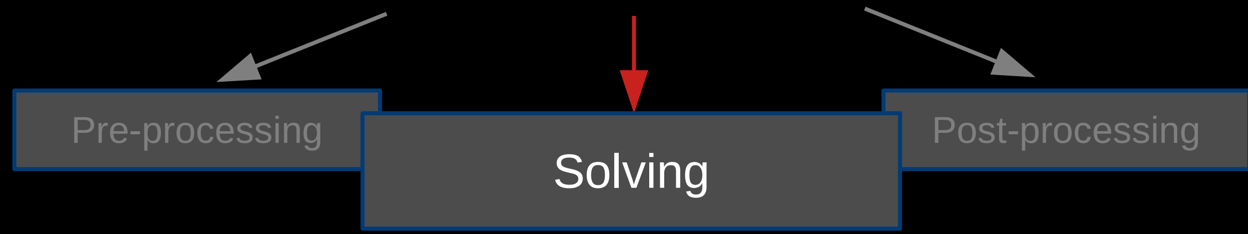
system/controlDict

```
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
}
```

```
foamToC -functionObjects
foamInfo -a probes
```

https://doc.cfd.direct/openfoam/user-guide-v11/post-processing-cli

OpenFOAM



1. Running solver

a) On a single core

```
foamRun
```

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

```
decomposePar
mpirun -np 4 foamRun -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, which is useful for steady-state problems.

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

```
stopAt          writeNow;
```

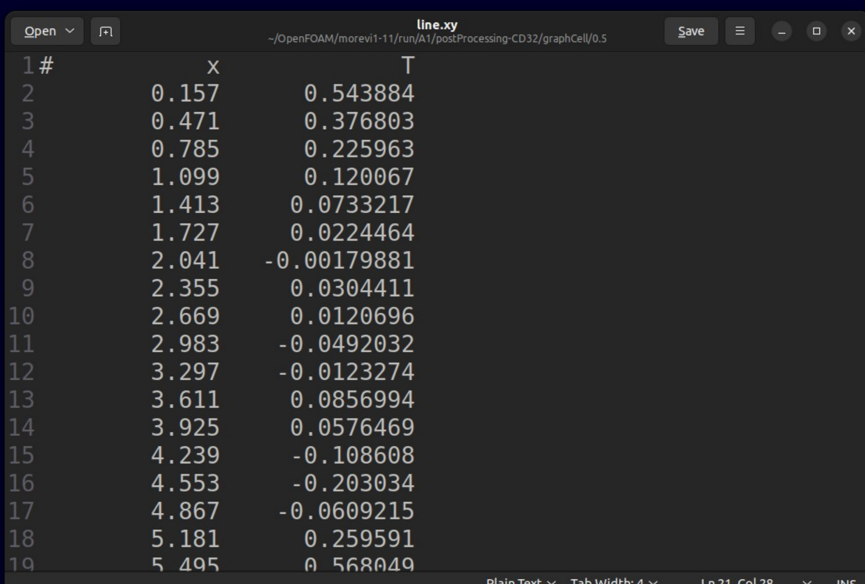
OpenFOAM

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

1. Post-processing and sampling

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

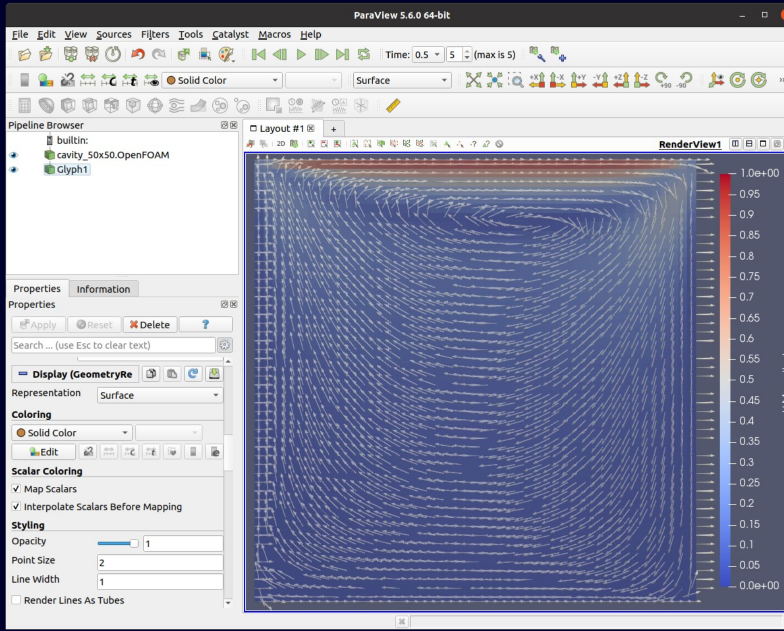
foamPostProcess -func *graphCell*



#	x	T
2	0.157	0.543884
3	0.471	0.376803
4	0.785	0.225963
5	1.099	0.120067
6	1.413	0.0733217
7	1.727	0.0224464
8	2.041	-0.00179881
9	2.355	0.0304411
10	2.669	0.0120696
11	2.983	-0.0492032
12	3.297	-0.0123274
13	3.611	0.0856994
14	3.925	0.0576469
15	4.239	-0.108608
16	4.553	-0.203034
17	4.867	-0.0609215
18	5.181	0.259591
19	5.495	0.568049

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:

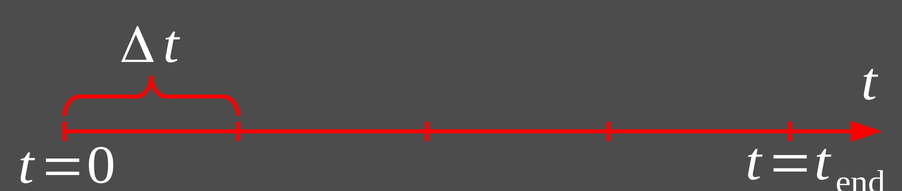
How does OpenFOAM solver proceed from one time step to the next one?

```
Time = 0.49s

ExecutionTime = 0.040446 s  ClockTime = 0 s

scalarTransport write:
smoothSolver:  Solving for T, Initial residual = 0.015539, Final residual = 4.09709e-17, No Iterations 2

Time = 0.495s

ExecutionTime = 0.040506 s  ClockTime = 0 s

scalarTransport write:
smoothSolver:  Solving for T, Initial residual = 0.0155755, Final residual = 5.83005e-17, No Iterations 2

Time = 0.5s
```

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

$$\Delta t = t_1 - t_0$$

$$c^{t_1} - ?$$

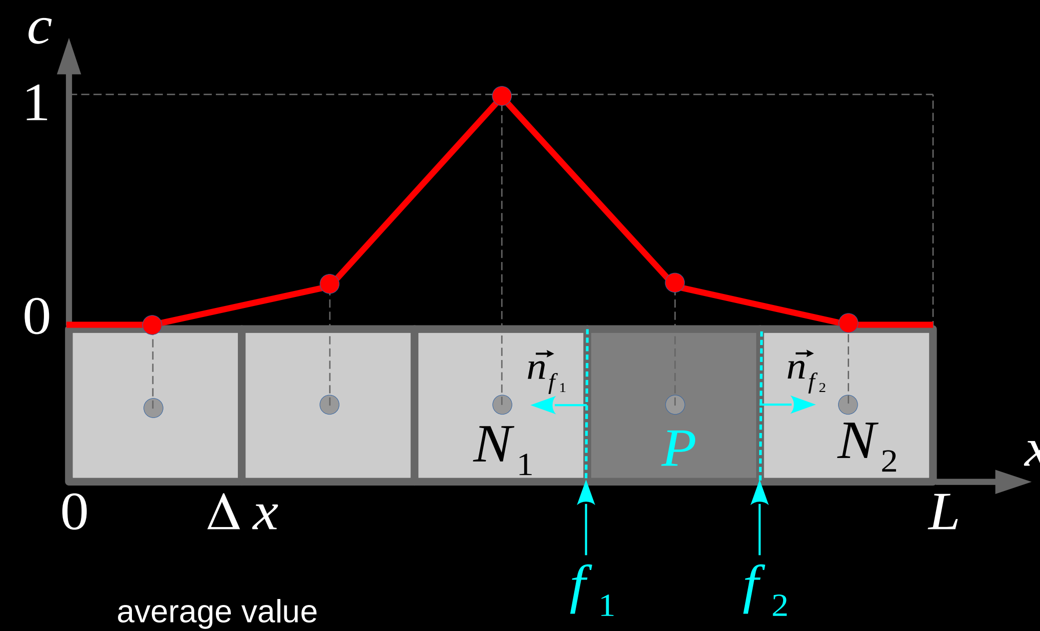
N – number of cells

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

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**

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

Gauss theorem

Finite number of flat faces

1d uniform mesh (only 2 faces)

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

Interpolating c_{f1}, c_{f2}

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

linear

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

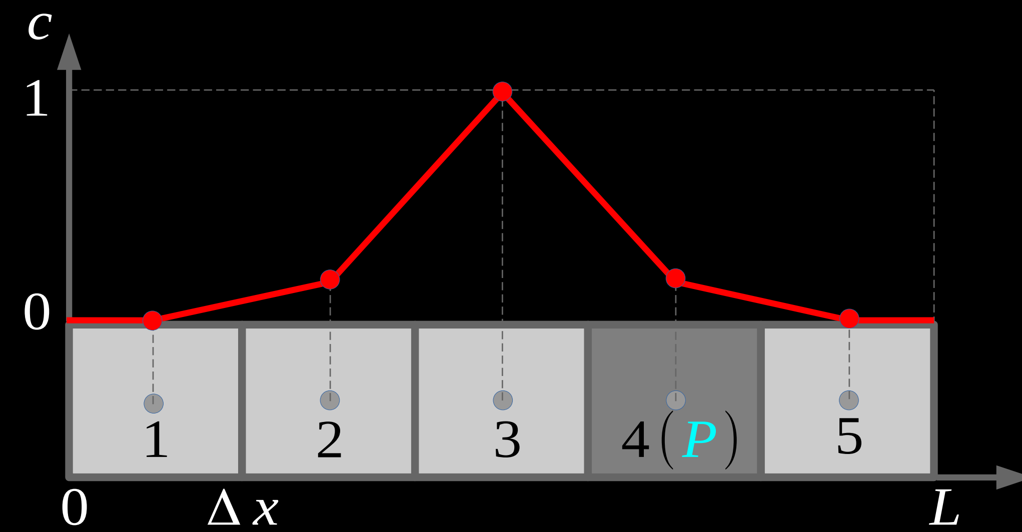
upwind

$$c_{f_1}=c_{N_1}$$

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

- 5 equations
- 5 unknowns

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



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

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

Use some matrix solver to obtain field c at time t_1

scalarTransport source code

Located in *src/functionObjects/solvers/scalarTransport/scalarTransport.C*

```
for (int i=0; i<=nCorr_; i++)
{
    fvScalarMatrix sEqn
    (
        fvm::ddt(s_)
      + fvm::div(phi, s_, divScheme)
     ==
        fvModels.source(s_)
    );

    if (diffusivity_ != diffusivityType::none)
    {
        sEqn -= fvm::laplacian(D(), s_);
    }

    sEqn.relax(relaxCoeff);

    fvConstraints.constrain(sEqn);

    sEqn.solve(schemesField_);

    fvConstraints.constrain(s_);
}
```

Loop over correctors, as defined in *system/fvSolution*

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

$$-\alpha\nabla^2 c$$

Matrix equation is constructed here

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

Call matrix solver, defined in *system/fvSolution*

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 well 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 good 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 non-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. Can be orthogonal, uncorrected or corrected, depending on your mesh

Check what schemes are used in tutorials:

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

Flux limiting schemes

Blended, but each face has its own blending coefficient

$$c_{f_2} = c_{f_2}^{UW1} - \beta(r)\left(c_{f_2}^{UW1} - c_{f_2}^{CD2}\right)$$

Limiter, e.g. depending on the ratio of successive gradients $r = \dfrac{\nabla_n c_P}{\nabla_n c_{f_2}}$

Recall:

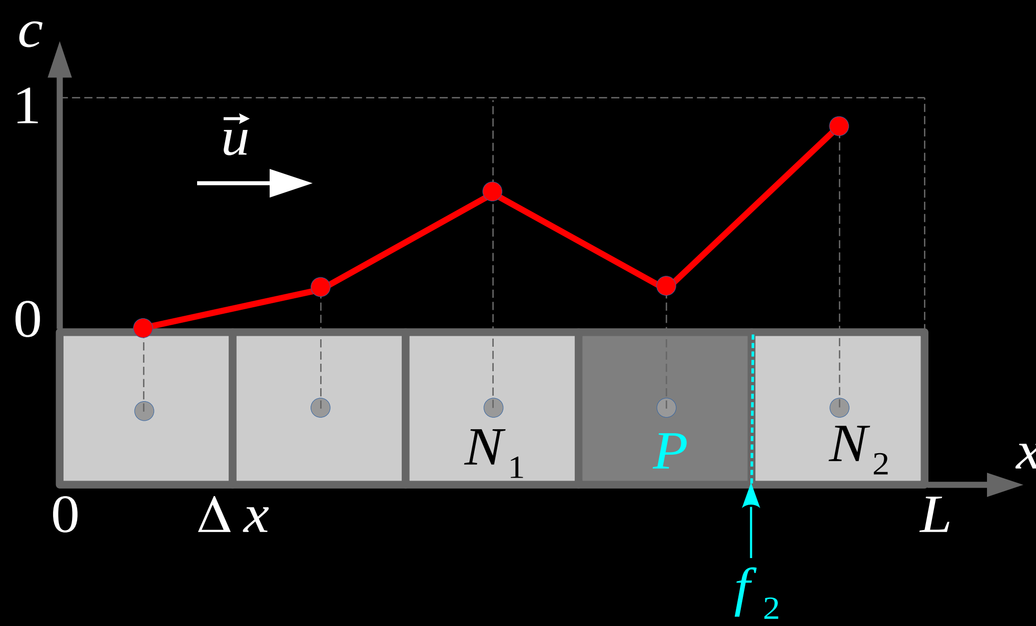
Interpolating c_f

linear

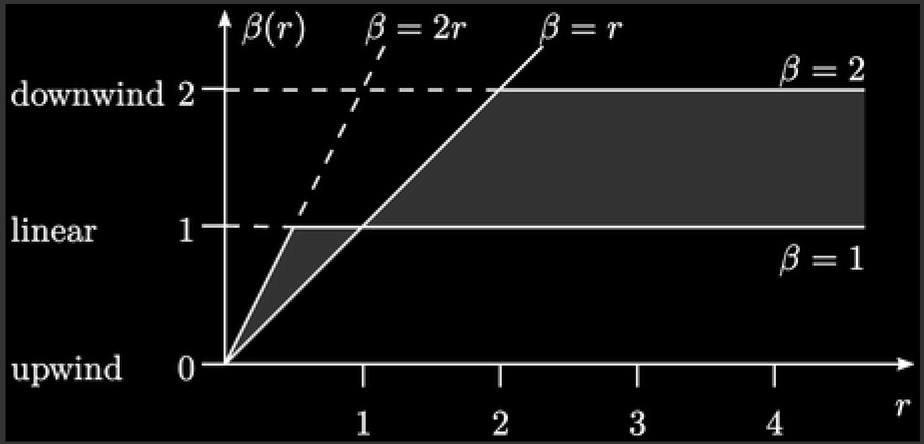
$$c_{f_1} = \frac{c_{N_1} + c_P}{2}$$

upwind

$$c_{f_1} = c_P$$



Total variation diminishing (TVD) schemes have *β(r)* defined such that it lies in the highlighted region



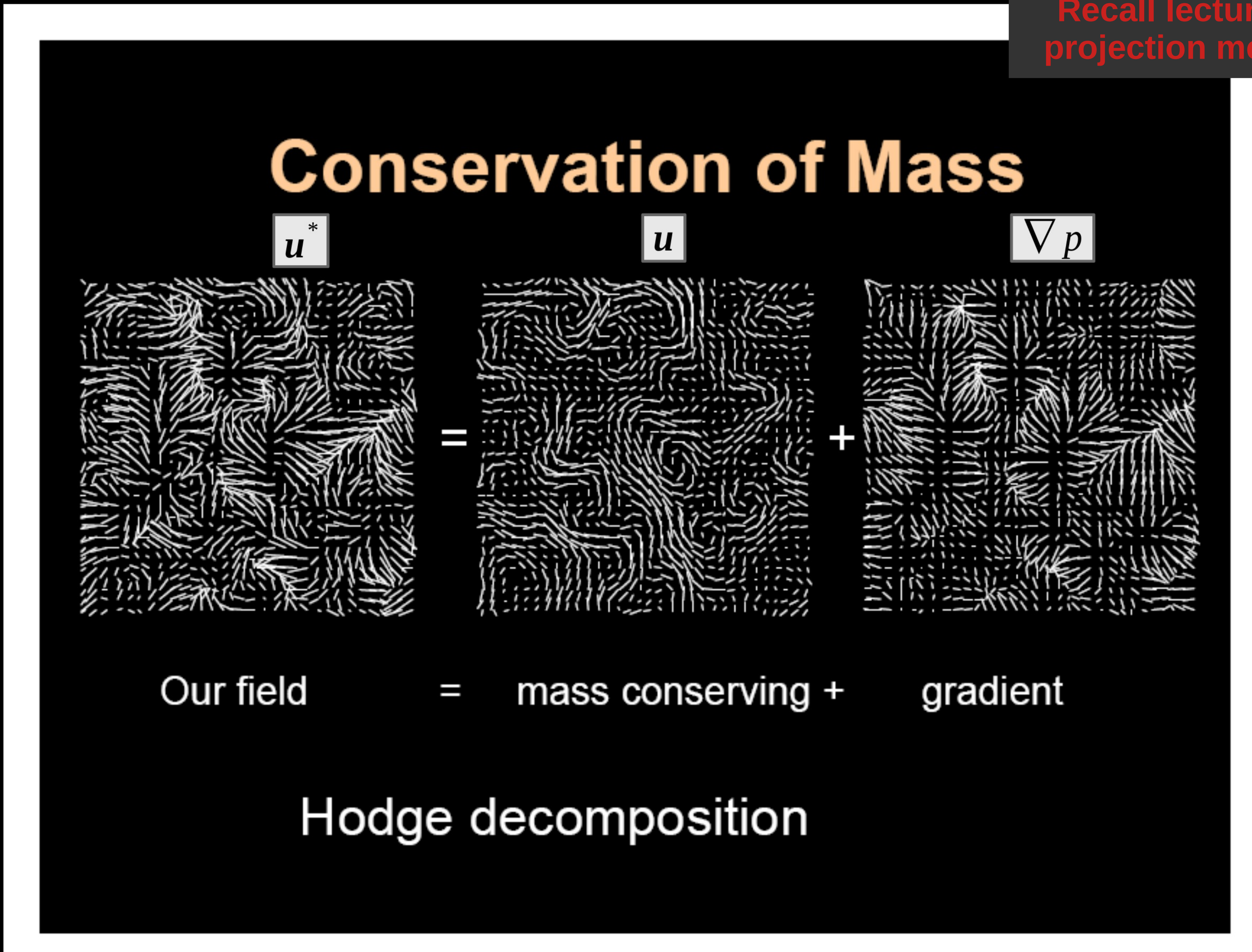
Sweby diagram

https://doc.cfd.direct/notes/cfd-general-principles/limited-advection-scheme

https://doc.cfd.direct/notes/cfd-general-principles/useful-tvd-schemes

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

Recall lecture 3: projection method



Simplified solution scheme

Initial guess

Step 0: Momentum predictor
Solve the momentum equation for the velocity field (using previous time step data). This velocity field does not satisfy the continuity equation.

Step 1: Explicit part evaluation
Use the velocity to calculate explicit part

$$\boldsymbol{u}^* = \boldsymbol{u}_n + \Delta t\,(D - C)$$

Step 2: Pressure-corrector
Solve the Poisson equation for the pressure field.

$$\nabla^2 p = \nabla \cdot \boldsymbol{u}^*$$

Step 3: Explicit velocity calculation
Use the pressure field to calculate new velocity field, satisfying the continuity equation. Pressure field is not corrected anymore

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}^* - \nabla p$$

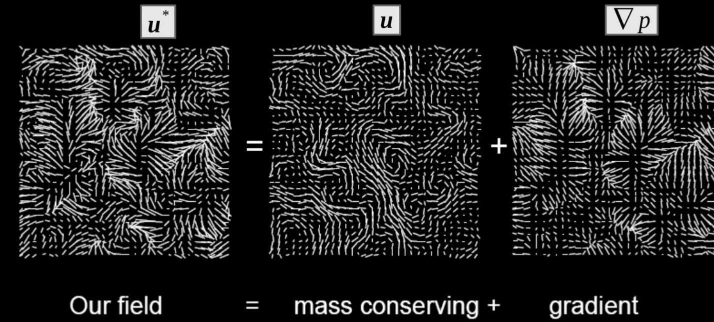
Mass-conserving solution $\nabla \cdot \boldsymbol{u}_{n+1} = 0$

Outer corrector (**SIMPLE** loop)

non-orthogonal corrector

Inner corrector (**PISO** loop)

Conservation of Mass



Hodge decomposition

Note: only the idea is given here, the real loop is more complex

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 unphysical 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. In transient simulations, the relTol for the final iteration is usually set to 0 to enforce convergence to absolute tolerance.

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

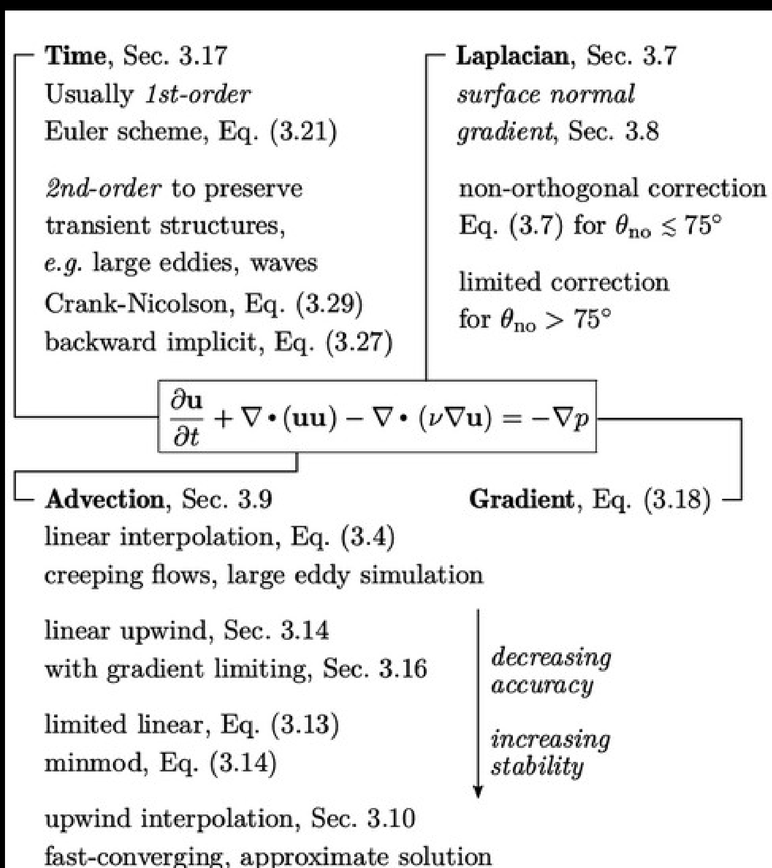
As long as the **solver converges**, the **solution is accurate** to a given tolerance value!

```
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-v11/index
Offline: /opt/openfoam11/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**

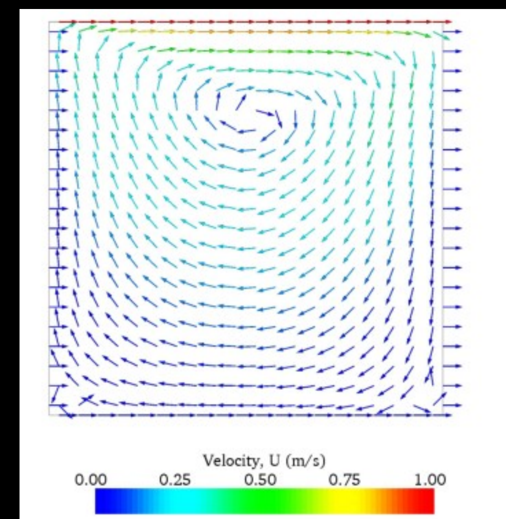


Figure 2.8: Velocities in the cavity case.

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 $\mathbf{a}, \mathbf{b} >= 1$	$\mathbf{a}\mathbf{b}$	`a * b`
Inner product	rank $\mathbf{a}, \mathbf{b} >= 1$	$\mathbf{a} \bullet \mathbf{b}$	`a & b`