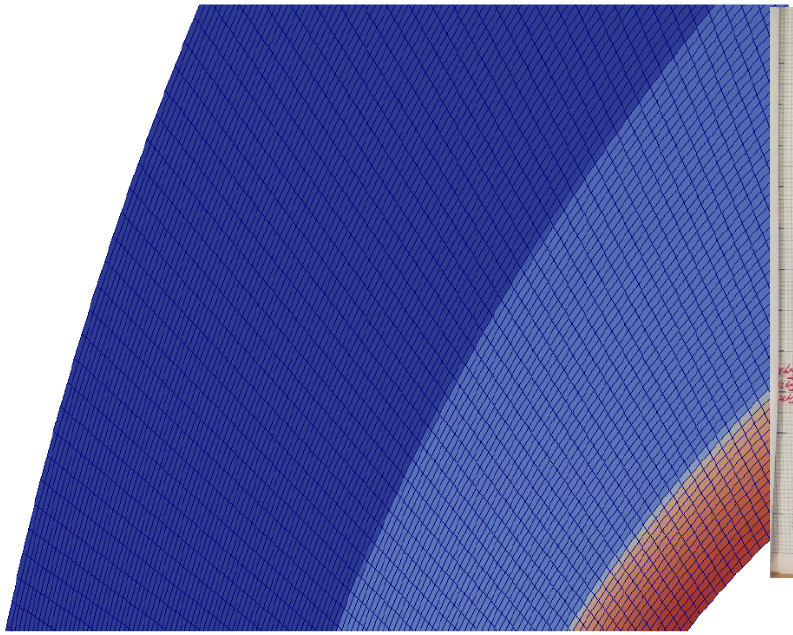
MECH4480
Computational Fluid Dynamics

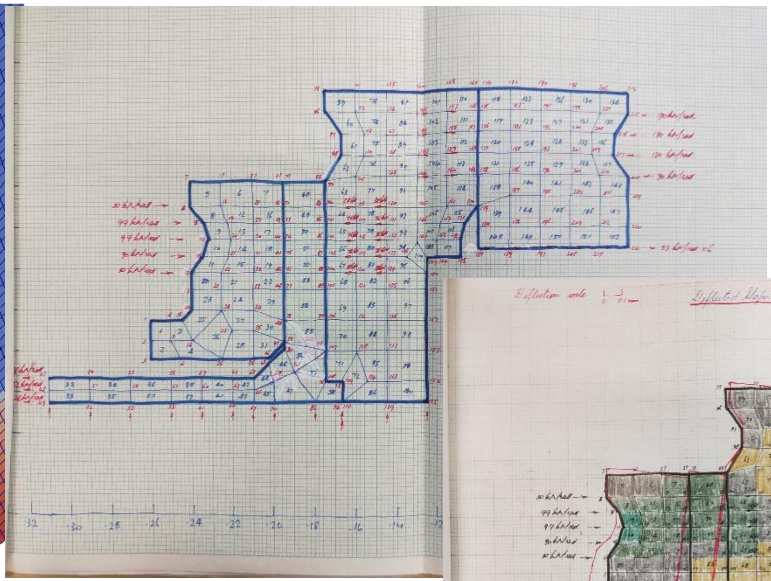
CFD

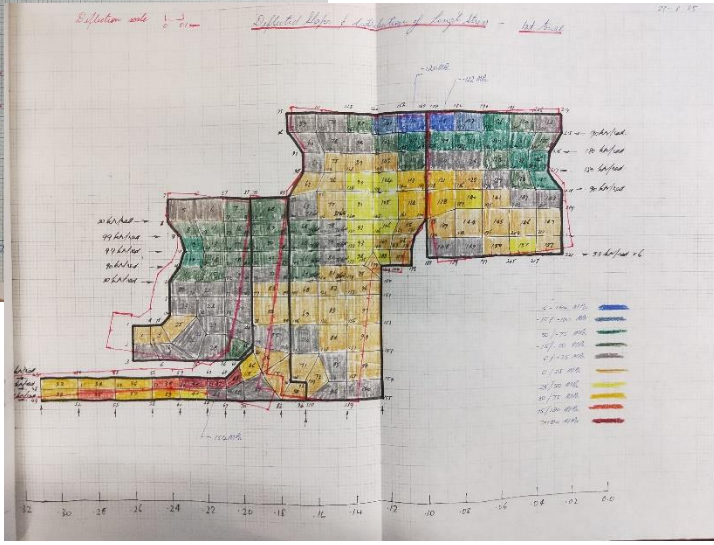
Finite-Volume Formulation

Content

Why we need a solver







FEA problem(single equation) solved by hand (1985)

- Fluid flows require solution of four or six (or more) simultaneous equations sets.
- Finer meshes are required to accurately solve convection and associated non-linearities.
- Solvers are designed to solve simultaneous sets of general transport equation.
- We will focus on OpenFOAM solver, but there are many other solvers.

Resources / Reading

Resources on OpenFOAM:

- OpenFOAM webpage http://www.openfoam.org/
- Offical documentation https://cfd.direct/openfoam/user-guide/
- Code Download https://openfoam.org/download/history/
- OpenFOAM Wiki http://openfoamwiki.net/index.php/Main_Page
- CFD-online ← good forum for general CFD questions
- Extensive OpenFOAM course

  http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/
  Slides from 2013
  http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2013/

Resources Meshing:

- Use built-in OF meshing tool → look at OF manual
- **Use `foamMesh` an extension to the Dlang Geometry Package**
- Use 3rd party meshing tool, then output or convert to *foam* format

What is OpenFOAM

- OpenFOAM is foremost a `C++` library that can be combine into executables (applications)
  - solvers, tools to solve the governing equations of continuum mechanics
  - utilities, tools designed to perform tasks and manipulate data
- There are tools for data pre and post-processing and converters from/to other programs for pre- and post-processing
  - foamMesh an add-on for the Dlang Geometry package to interface with OF
  - paraFoam to view results in paraview
  - many others
- OpenFOAM is distributed with a large number of applications, and this is constantly growing. Many advanced users develop their own applications and/or modify the existing applications.

Note: Some of the commands shown here may be slightly different in different versions of OpenFOAM, but the procedures remain the same.

OpenFOAM directory structure

Simulations are conducted in a standard directpry structure. `Case/` describes the *case* and can have any name. All commands are executed from the `Case/` directory.

To create a mesh:

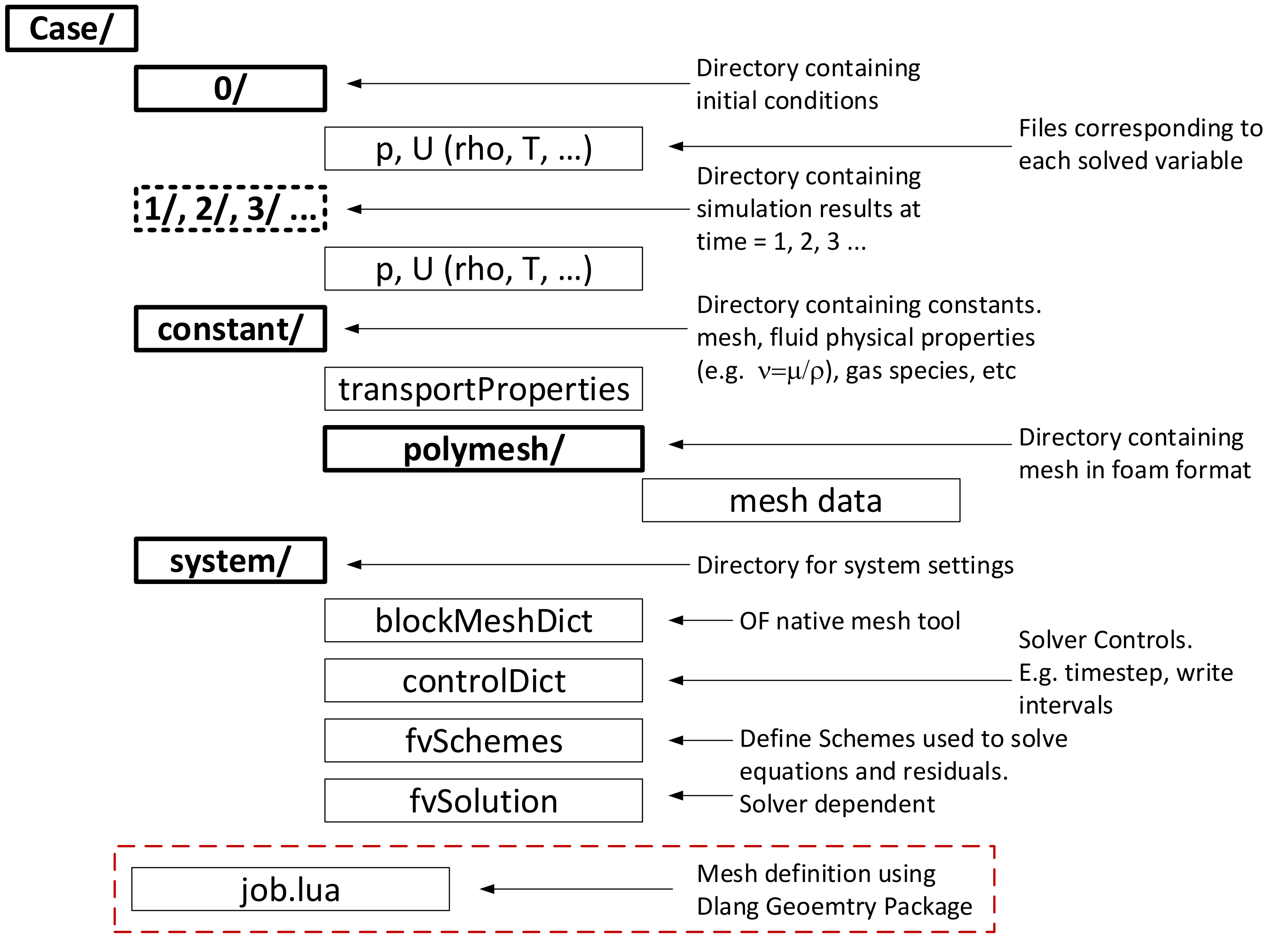
```
$ blockMesh or
$ foamMesh
```

To run a solver:
E.g. incompressible

```
$ icoFoam
```

To view results:

```
$ paraFoam
```



Case/
0/
Directory containing initial conditions
p, U (rho, T, ...)
Files corresponding to each solved variable
1/, 2/, 3/ ...
Directory containing simulation results at time = 1, 2, 3 ...
p, U (rho, T, ...)
constant/
Directory containing constants. mesh, fluid physical properties (e.g. ν=μ/ρ), gas species, etc
transportProperties
polymesh/
Directory containing mesh in foam format
mesh data
system/
Directory for system settings
blockMeshDict
OF native mesh tool
controlDict
Solver Controls. E.g. timestep, write intervals
fvSchemes
fvSolution
Define Schemes used to solve equations and residuals. Solver dependent
job.lua
Mesh definition using Dlang Geoemtry Package

icoFoam : Cavity Example

Step 0 - Load OpenFoam commands: `$ of50` (50, corresponds to version number, e.g. 5.0)

Step 1 - Copy directory structure & create mesh

You may have to create this directory. Use `$ mkdir NAME`

- Use aliases to accelerate movement between folders. E.g.

  `$ cd $FOAM_RUN` is short-hand for `$ cd ~/OpenFOAM/username-5.0/run`

- Standard procedure for running a tutorial (apply the same for your cases)

  ```
  $ cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN
  ```
  `$ run` (alias for changing to run directory)
  ```
  $ cd cavity/cavity
  ```
  You have created your own copy of the cavity tutorial and are now in the <case> directory

- To create the grid, there are two options:

Use OF meshing: 1st Example for cavity

- View blockMeshDict
  ```
  $ gedit /system/blockMeshDict
  ```
- Create mesh
  ```
  $ blockMesh
  ```

Use foamMesh: 2nd Example for cavityClipped

- Copy `cavityClipped.lua` from Blackboard
- View setup file
  ```
  $ gedit cavityClipped.lua
  ```
- Run `$ foamMesh --job=cavityClipped`

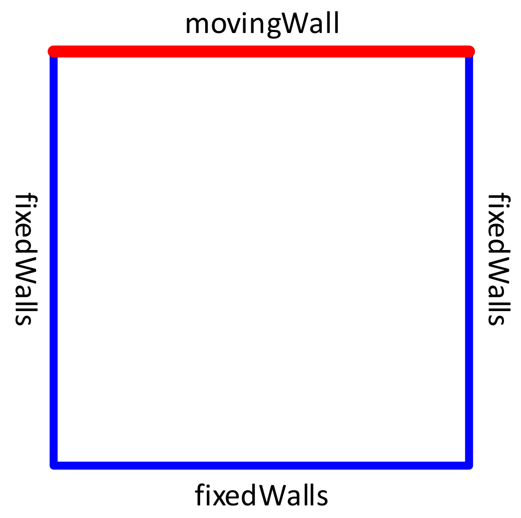
You can get a copy from the repository in ~Desktop/Examples/OpenFoam

- View the mesh: `$ paraFoam`

icoFoam: 1st Example Cavity (blockMesh)

Step 2 - Setting Initial & Boundary Conditions

- Run `$ blockMesh`
- Open files `/0/p` and `/0/U`.
  Here initial conditions and boundary conditions are set.
  - `dimensions` sets dimensions
  - `InternalFiled` sets value at cells within domain
  - `boundaryField` sets type and value for each external face of fluid domain
- Run `$ paraFoam` to view mesh



🟦	**Wall:** Velocity = (0,0,0) Pressure = zeroGradient
🟥	**Moving Wall:** Velocity = (1.0,0,0) Pressure = zeroGradient

frontAndBack:
type empty
(setting for 2-D solution)

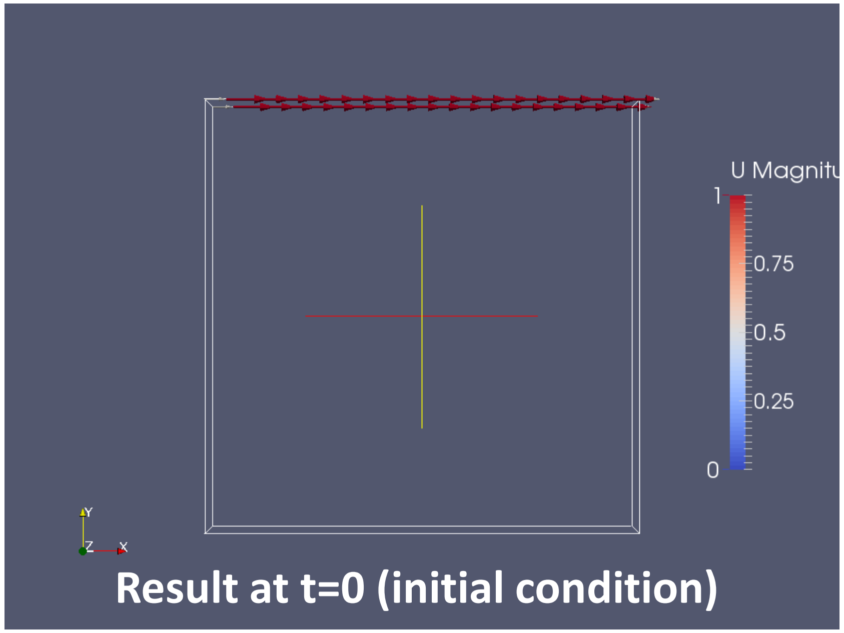
icoFoam : Cavity Example (blockMesh)

Step 3- check Mesh and run solver

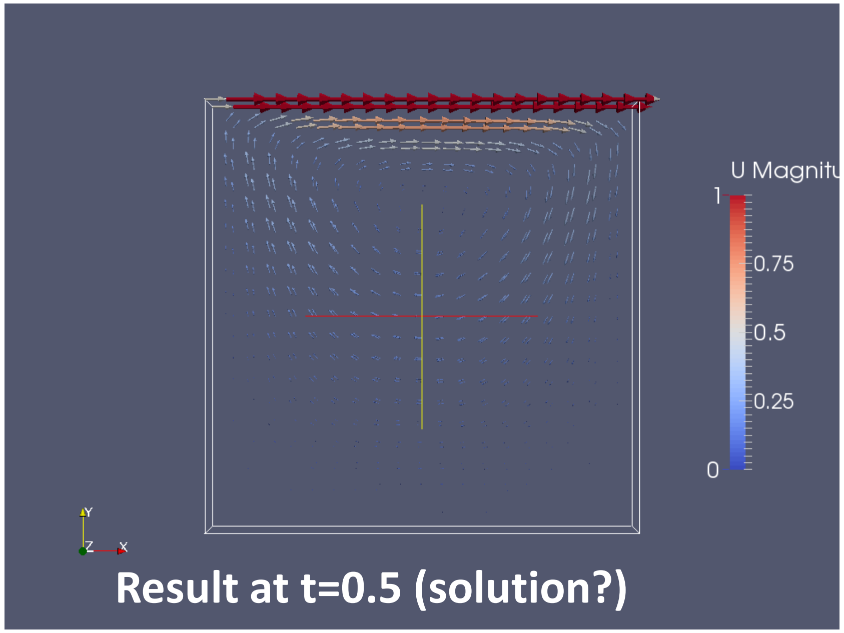
- Check the mesh, to see if it is valid and to check quality.
  ```
  $ checkMesh
  ```
- Run the solver (in this case icoFoam, an incompressible solver)
  ```
  $ icoFoam
  ```
  - Optionally you can run in the background and create a log file. Use
    ```
    $ icoFoam > log &
    ```
    The log file can be viewed using the command `$ tail log`
- View the result in paraview
  ```
  $ paraFoam &
  ```
  This converts OpenFOAM results into an object for viewing in paraview and opens paraview.

Cavity Results

Results from cavity example using blockMesh.



Result at t=0 (initial condition)



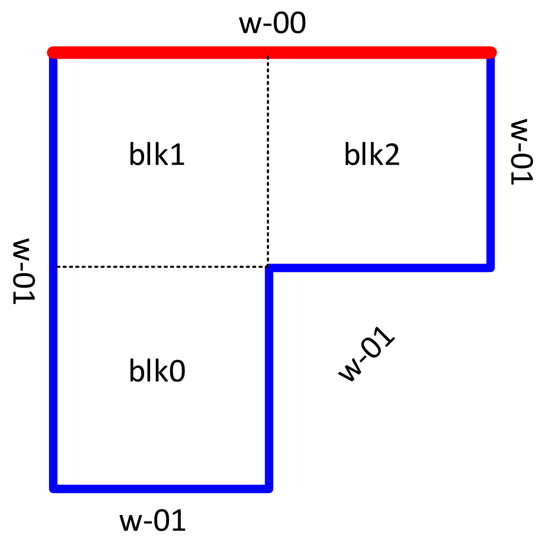
Result at t=0.5 (solution?)

- Use arrows in paraview / paraFoam to explore result. Different solutions are obtained for different values of Time.
- `icoFoam` is a transient solver. Hence solution shows what happens to a stationary fluid, in a box, if the lid starts to move instantaneously.
- Try altering the boundary conditions and re-running `icoFoam`

icoFoam: 2nd Example cavityClipped (foamMesh)

Step 2- Setting Initial & Boundary Conditions

- View `cavityClipped.lua` → Note extra step to assign boundary labels
- Run `$ foamMesh`
- Open files `/0/p` and `/0/U`.
  This time we need to set the boundary conditions.
  - Set `w-00` as moving wall
  - Set `w-01` as stationary wall
  - Remove all un-used boundary condition templates
- Run `$ paraFoam` to view mesh and boundaries



🟦	**Wall:** Velocity = (0,0,0) Pressure = zeroGradient
🟥	**Moving Wall:** Velocity = (1.0,0,0) Pressure = zeroGradient

frontAndBack:
type empty
(setting for 2-D solution)

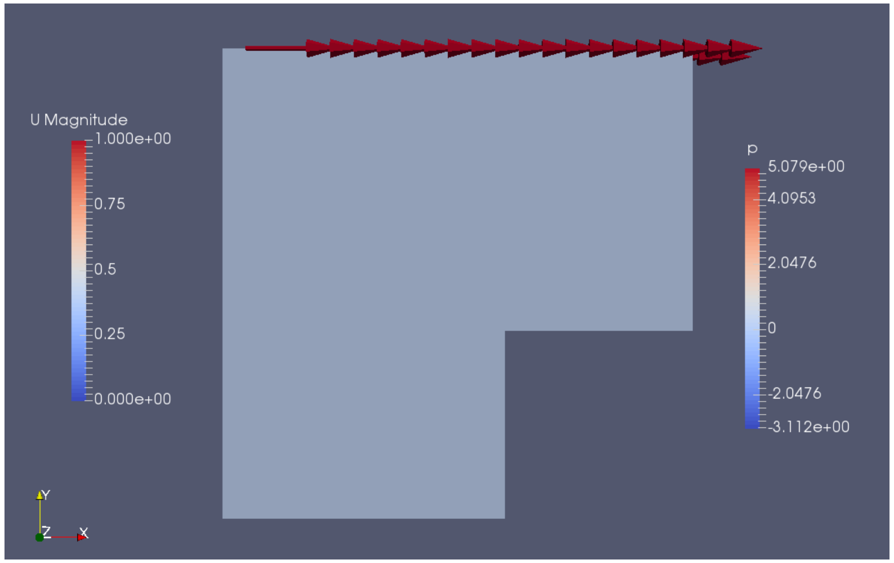
icoFoam : CavityClipped Example (foamMesh)

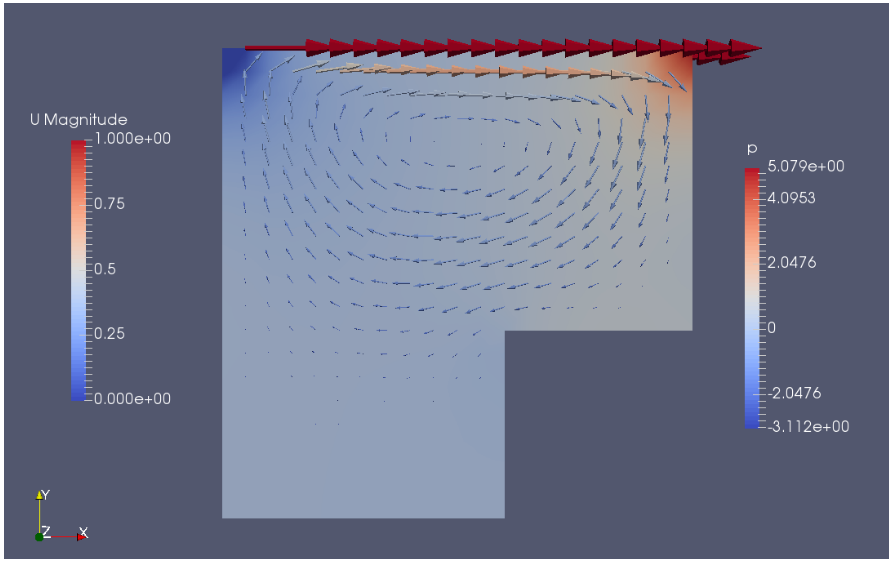
Step 3 - check Mesh and run solver

- Check the mesh, to see if it is valid and to check quality.
  ```
  $ checkMesh
  ```
- Run the solver (in this case icoFoam, an incompressible solver)
  ```
  $ icoFoam
  ```
  – Optionally you can run in the background and create a log file. Use
    ```
    $ icoFoam >& log &
    ```
    The log file can be viewed using the command `$ tail log`
- View the result in paraview
  ```
  $ paraFoam &
  ```
  This converts OpenFOAM results into an object for viewing in paraview and opens paraview.

CavityClipped Results

Results from cavity example using foamMesh.





- Results are created by overlaying different results / images
  - Pressure + Surface to show pressure in background
  - Glyphs for U to create velocity vectors
    Active Attributes: Vectors - **U**; Scale Mode - **on**; Scale factor - **auto**

We have several *solutions*!

- A number of `/0.*` directories have been created. These contain snap-shots of the solution at corresponding time-step or iteration step.
  - They are similar to the `/0` directory, but `internalField` has become a `nonuniform List<vector>` (or `List<scalar>`). These are the data at the cell centers
  - Boundary Conditions can be found at bottom. Useful for changing B/Cs and then restarting run from later point.
  - The `phi` file contains the face fluxes.
  - What do the fluxes correspond to for `icoFoam`?

```
dimensions [ Mass Length Time Temperature
             Quantity(mole) Current LuminousIntensity ]
```

How was the solver able to get from the starting point to the final results?

Tricks and getting help

- As with most command line commands, there are help options. E.g. to display usage of `icoFoam`

  ```
  $ icoFoam -help
  ```

  However better place is usually the openfoam wiki / userguide.
- Within OF files the `dummy` command can be used. E.g. to find out about possible boundary condition types use following syntax and then run `icoFoam` to display a list of options.

  ```
  boundaryField
  {
          movingWall
              {
                      type        dummy;
               }
  ….  (other B/Cs go here)
  }
  ```

  The list of options will be displayed in the terminal
- Read the error message! Last line also identifies file with error.

$$\text{file: /…/}\underbrace{\text{cavityClipped/0/p}}_{\text{directory and filename}}\text{.}\underbrace{\text{boundaryField.lid}}_{\text{dictionary / sub-dictionary}}\text{ from }\underbrace{\text{line 26 to line 26.}}_{\text{line number}}$$

How the problem is solved

Step 0: Define a grid - DONE (see Dlang Geometry User Guide)

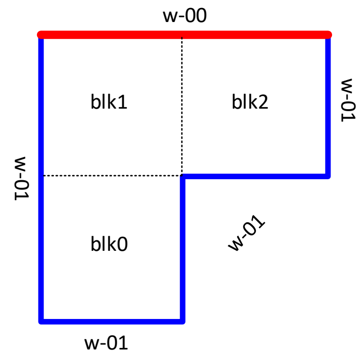
Step 1: Convert the grids to FoamBlocks and assign boundary labels.
To define a *foam* block use the following constructor

```
FoamBlock:new{grid=grid_0, bndry_labels={west="Name1", north="Name2"}}
```

`grid_0` is a grid instance.
`bndry_labels` is a table containing boundary labes for block sides. Internal faces must be omitted.

The following naming convention is supported:
`w-00, w-01, ... w-NN` for walls
`i-00, i-01, ... i-NN` for inlets
`o-00, o-01, ... o-NN` for outlets
`s-00, s-01, ... s-NN` for symmetry

Use labels to group external faces that have same boundary condition



The foamBlock constructor

<u>Structured Meshing in MECH4480/7480</u>

Eilmer 4.0 Geometry Package
lua script `job.lua`:
- Define geometry (points & lines)
- Define parametric surfaces/volumes

```
  surface = CoonsPatch:new{…}
```

- Create grids

```
 grid = structuredGrid:new{…}
```

For Eilmer:
- define B/Cs in `job.lua`
- *run:*

```
$ e4shared --prep --job=job
```

<u>Unstructured Meshing in MECH4480/7480 (optional)</u>

SnappyHexMesh
- define .stl
- generate mesh

For OpenFOAM
- Create foamBlocks and label boundaries

```
  fblock = FoamBlock:new{grid=grid,
      bndry_labels={south=''w-00'',
      north=''w-01'', west=''i-00''}}
```

(Use labels to 'group' external boundaries)
- run:

```
  $ foamMesh --job=job
```

- Set boundary conditions in `/0/p`, `/0/U`, etc

How the problem is solved

Step 2a: set Boundary Conditions

- B/Cs must be defined for each `transported' property at every domain boundary. In current case pressure and velocity
- Velocity in **0/U**

```
dimensions [0 1 -1 0 0 0 0];
internalField  uniform (0 0 0);
boundaryField
{
    movingWall  // w-00
    {
        type   fixedValue;
        value  uniform (1 0 0);
    }
    fixedWalls // w-01
    {
        type   noSlip;
    }
    frontAndBack
    {
        type   empty;
    }
}
```

B/C types:

`fixedValue`, creates a constant value B/C

For stationary walls:

(Ux, Uy, Uz) = (0, 0, 0)

For sliding wall, Ux = 1

(Ux, Uy, Uz) = (1, 0, 0)

Can also be used for fixed velocity inlets.

`noSlip`, same as fixedValue with (0,0,0)

`frontAndBack` is used for 2-D meshes with **`type empty`**

How the problem is solved

- Pressure in **0/p**

```
dimensions [0 2 -2 0 0 0 0];

internalField  uniform 0;

boundaryField
{
    movingWall  // w-00
    {
        type   zeroGradient;
    }
    fixedWalls  // w-00
    {
        type zeroGradient;
     }
    frontAndBack
    {
        type   empty;
    }
}
```

Dimensions are in $m^2 s^{-2}$ rather than Pa = kg $m^{-1} s^{-2}$.
I.e. dimensions are for Pressure/ρ.
`icoFoam` and many other incompressible flow solvers actually solve for volume flux. This is ok, as long as all terms are multiplied by 1/ρ.
➔ use ν rather than μ for viscosity.

`zeroGradient`, sets a d/dn = 0 boundary condition, where n is the boundary face normal.
This allows the value at wall to move up/down without constraint, but the d/dn = 0 enforces no diffusion.

`frontAndBack` is used for 2-D meshes with **`type empty`**

How the problem is solved

Step 3: define the fluid and other constant properties. Settings depend on solver.

Laminar Solver, incompressible flow only needs viscosity.

- Fluid properties are in **`constant/transportProperties`**

```
nu [0 2 -1 0 0 0 0] 0.01;
```

$\nu = 0.01\,\mathrm{m}^2\,\mathrm{s}^{-1}$

Turbulent Solver, incompressible flow needs viscosity and density and turbulence model settings

- Fluid properties and constants in **`constant/transportProperties`**

```
transportModel  Newtonian;
rho             rho [ 1 -3 0 0 0 0 0 ] 1;
nu              nu [ 0 2 -1 0 0 0 0 ] 1e-05;
```

$\nu = 1 \times 10^{-5}\,\mathrm{m}^2\,\mathrm{s}^{-1}$
$\rho = 1\,\mathrm{kg}\,\mathrm{m}^{-3}$

- Turbulent Model Setting **`constant/turbulenceProperties`**

```
RASModel        SpalartAllmaras;
turbulence      on;
printCoeffs     on;
```

Define model used for Reynolds Averaged Simulation. Use `RASModel dummy;` to get list of possible options.

How the problem is solved

Step 4: define settings for solver.
E.g. how long to run the simulation (no. of time steps)? How big should the iterative steps be? When to write data to file? Where to start (i.e. from which file)?

- Defined in **`system/controlDict`**

```
application      icoFoam;
startFrom        latestTime;
startTime        0;
stopAt           endTime;
endTime          0.5;
deltaT           0.005;

writeControl     timeStep;
writeInterval    20;

purgeWrite       0;
writeFormat      ascii;
writePrecision   6;
writeCompression off;
timeFormat       general;
timePrecision    6;
runTimeModifiable true;
```

Sets simulation start and stop time.
`startFrom latestTime`
→ read data from `/N` folder with highest number.
`startFrom  startTime`
→ start from folder with number defined by variable `startTime` in next line.

Define when to stop simulation. For transient solvers, typically 3-5 *flow-length* are sufficient. Always check convergence based on solution!

deltaT, see next slide.

Define how often to write data to file. I.e. how often to create a /N folder. Currently folders are generated every $20 \times 0.005\,\mathrm{s} = 0.1\,\mathrm{s}$

Something about time-steps

- Transient or quasi transient methods
  - Courant number criteria must be full-filled to ensure a stable and time-accurate solution. There are slightly different definitions, but in essence it is the following ratio
  $$Co = \frac{\text{Distance properties are convected in a timestep}}{\text{cell dimension}} = \frac{dt\,|\mathbf{U}|}{dx}$$
  Courant number must be calculated for each dimension/cell in the mesh to find limiting cell.
  - Courant number limits depend on solver. Typically
    - <0.5 for time accurate solutions
    - 0.5 to 1.0 for some implicit transient solvers
    - > 1.0 for steady state solvers
  - icoFoam, deltaT must be set manually
  $$\text{for moving lid } U_{max} = 1\,\mathrm{m\,s^{-1}} \text{ and } dx = \frac{0.1\,\mathrm{m}}{20} = 0.005\,\mathrm{m} \rightarrow t = 0.005\,\mathrm{s}$$
  - interFoam, a limiting Courant number is defined and the solver then automatically adjusts time-step for each iteration.
- Steady state methods – try to go directly to steady state solution (e.g. simpleFoam, pimpleFoam)
  - Here a timestep is equivalent to one iteration step.
  Best to set `deltaT = 1`

Unstable Solutions

Divergence / Instability of your solution can be caused by many reasons.

- Time-step is too large. This results in growing solution oscilaltions
  – Try running the cavity example with (in ControlDict)
    ```
    deltaT          0.005;
    ```

- Poor initialisation of the solution. This requires large correction during the first step.
  The resulting large gradients (which are typically linear approximations) lead to unstable/non-physical results.
  – Try running the cavity example with
    ```
    internalField  uniform (-1 0 0);
    ```

- Gibbs Phenomena with higher order solvers. Can be addressed using limiters (will be covered as part of compressible flow)

How the problem is solved

Step 5: define Finite Volume Solution. – set how linear set of equation to find `p` and `U` are solved.

- Defined in **system/fvSolution**

```
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;
    }
}
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

This defines the PCG Preconditioned Conjugate Gradient solver to solve the Pressure Equations.
This solver runs until solution has converged to an absolute tolerance < tolerance.
Optionally to stop solver once a relative Tolerance has bee attained, set relTol

Defines solver for U fields

Overall scheme used for timestepping

Incompressible solvers are independent of absolute pressure (only dP/dx matters). To stop pressure solution from floating, a single cell (with index pRefCell) is fixed at a given pressure value pRefValue

(Cell 0 is typically in SW corner of 1st block)

How the problem is solved

Step 6: define solution schemes. (Only for very advanced users)
[Don’t alter, simply copy from existing example]

- Defined in **system/fvScheme**

```
ddtSchemes
{
    default         Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear orthogonal;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         orthogonal;
}
```

In the process of solving the general transport equation different algebraic equations need to be evaluated for the scalar and vector field. Here one select the corresponding discretisation schemes

For example currently the `Euler` scheme is used for forward time-stepping. Alternatives would be to use a Crank-Nicholson scheme or bounded Euler. Use dummy to find out options.

Revisit log-file output

```
Time = 0.495

Courant Number mean: 0.222158 max: 0.852134
smoothSolver:  Solving for Ux, Initial residual = 2.46591e-07, Final residual = 2.46591e-07, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 5.36152e-07, Final residual = 5.36152e-07, No Iterations 0
DICPCG:  Solving for p, Initial residual = 6.20776e-07, Final residual = 6.20776e-07, No Iterations 0
time step continuity errors : sum local = 6.85402e-09, global = -2.53944e-19, cumulative = -2.04992e-18
DICPCG:  Solving for p, Initial residual = 8.33045e-07, Final residual = 8.33045e-07, No Iterations 0
time step continuity errors : sum local = 8.59385e-09, global = 5.07889e-19, cumulative = -1.54203e-18
ExecutionTime = 0.08 s  ClockTime = 1 s
```

Ux and Uy are evaluated

P is evaluated twice, as set by PISO algorithm nCorrectors

OpenFOAM Tutorials

- OpenFOAM documentation is "average". However there is a good collection of tutorials. Use these as a starting point for your simulations. E.g. copy → modify → simulate.
- Tutorials are stored in
  `opt/openfoamX/tutorials` (Note X to be replaced by version number, e.g. 5)
  - Shortcuts to get there are
    `$ tut` or
    `$ cd $FOAM_TUTORIALS`
  - For current course we focus on `icoFoam` and `simpleFoam`
- Other sources for tutorials:
  - The User guide or The Programmers Guide, chapter 3
    http://foam.sourceforge.net/docs/Guides-a4/ProgrammersGuide.pdf
  - The OpenFOAM Wiki
  - The OpenFoam Forum
  - On an OpenFOAM training course...

I strongly recommend working through the full set of `icoFoam` Cavity examples (see User Guide) and also the `simpleFoam` 2D aerofoil.

Different solvers

icoFoam: Transient solver for laminar flows.

simpleFoam: Implicit solver for incompressible flow problems. Supports laminar and turbulent flows. The SIMPLE algorithm is efficient when we only want to solve the steady state continuity and momentum equations.

$$\int_A \mathbf{n}\,.\,(\rho \mathbf{u}) dA = \int_{CV} S dV \quad \rightarrow \quad div(\rho \mathbf{u}) = 0$$

$$\int_A \mathbf{n}\,.\,(\rho u \mathbf{u}) dA = \int_A \mathbf{n}\,.\,(\Gamma\, grad\, u) dA + \int_{CV} S_{Mx} dV \quad \rightarrow \quad \begin{array}{l} div(\rho u \mathbf{u}) = -\frac{dp}{dx} + div(\mu\, grad\, u) + S_{Mx} \\ div(\rho v \mathbf{u}) = -\frac{dp}{dy} + div(\mu\, grad\, v) + S_{My} \end{array}$$

This is a challenging set of equations to solve as:

- there are nonlinear terms on the left hand side $\rho \mathbf{u} u$
- all three equations are intrinsically coupled and role of pressure is not clear.

We will look at the details of the simple algorithm and how turbulence is modelled later.

Setting up an aerofoil simulation

Lets browse for a suitable tutorial…

```
$ cd $FOAM_TUTORIALS/incompressible/simpleFoam
```

Copy the tutorial to your run directory

```
$ cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/airFoil2D $FOAM_RUN
```

Example of simpleFoam simulation of a 2-D aerofoil. By inspection of the directories and files, find answers to the following.

- Is a turbulence model used for the simulation?
  If yes, which one?
- What density is used for the simulations?
- What is the velocity and direction of the air entering the simulation domain?
  Why isn't it horizontal?

some useful B/Cs

Setting far-field B/Cs can be difficult due to conflicting requirements

- Boundary to be as close as possible → small meshes an efficient solution
- Boundary must not affect the solution → location of boundary should be far from area of interest
- Option 1: Fix pressure and velocity, but move boundaries far far far away. 100 x chord length is recommend
- Option 2: Use "intelligent", boundaries. Here behaviour of B/C automatically changes based on local flow properties. This tries to mimic the behaviour of a B/C that is far far far away. See below for one example that considers flow direction.

Pressure in p

```
{
  type  freestreamPressure;
}
```

This boundary condition provides a free-stream condition for pressure. It is a zero-gradient condition that constrains the flux across the patch based on the free-stream velocity.

Velocity in U

```
{
  type                freestream;
  freestreamValue     uniform (5 1 0);
}
```

This boundary condition provides a free-stream condition. It is a 'mixed' condition derived from the inletOutlet condition, whereby the mode of operation switches between fixed (free stream) value and zero gradient based on the sign of the flux.

Aerofoil specific tricks and functions

- Changing angle of attack
  - Keep mesh the same, but change velocity direction

$$U_x = U_\infty \cos\alpha; \quad U_y = U_\infty \sin\alpha$$

- To get Force Coefficients, add the function object `forces` to the end of `ControlDict`

```
functions
{
  forces
  {
  type               forceCoeffs;
  libs ( "libforces.so" );
  writeControl      timeStep;
  writeInterval     1;
  patches
  (
    w-00
  );
  p             p;
  U             U;
  rho           rhoInf;
  rhoInf        1.;
  CofR ( 0 0 0 );
  liftDir ( 0.087 0.996 0 );
  dragDir ( -0.996 0.087 0 );
  pitchAxis ( 0 0 1 );
  magUInf 26.00;
  lRef 1;
  Aref 1;
  }
}
```

Instructions on where to find function

Give list of patches over which forces shall be evaluated

Define where to get p, U and rho

- Define Centre of Rotation CoR;
- Direction vector for lift&drag (needs to be adjusted if air flow direction changes)
- Set pitch axis

Set U_inf, Area and L used for coefficient calculations

$$C_L = \frac{\sum F_{L,\,patches}}{\frac{1}{2}\rho_\infty U_\infty^2 A_{ref}}$$

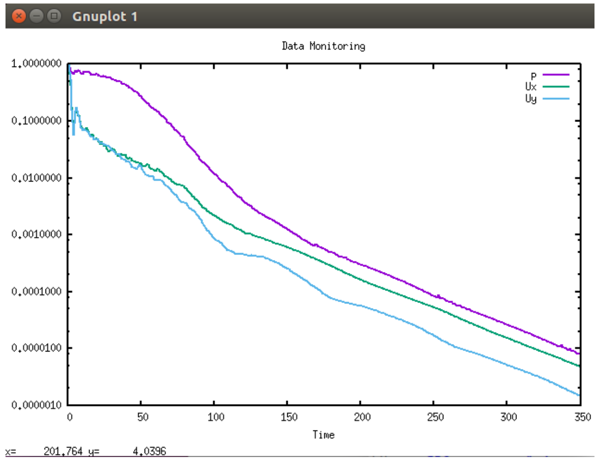
Probing Data

See section 6.3 of OF user Manual for tools to extract data. Live monitoring (section 6.3.4) is useful to keep an eye on your simulation.
Is it diverging? Has is converged?
How to use the residuals function:

- Find residuals file: `$ find $FOAM_ETC -name residuals`
- Copy residuals file to `/system`: `$ cp /path-to/residuals /system`
- Add function call to controlDict

```
    functions
    {
    #includeFunc residuals
        ... other function objects here
 }
```

- Clear postProcessing directory: `$ rm -rf postProcessing`
- Run solver: E.g. `$ simpleFoam > log &`
- Show residuals: `$ foamMonitor -l postProcessing/residuals/0/residuals.dat`

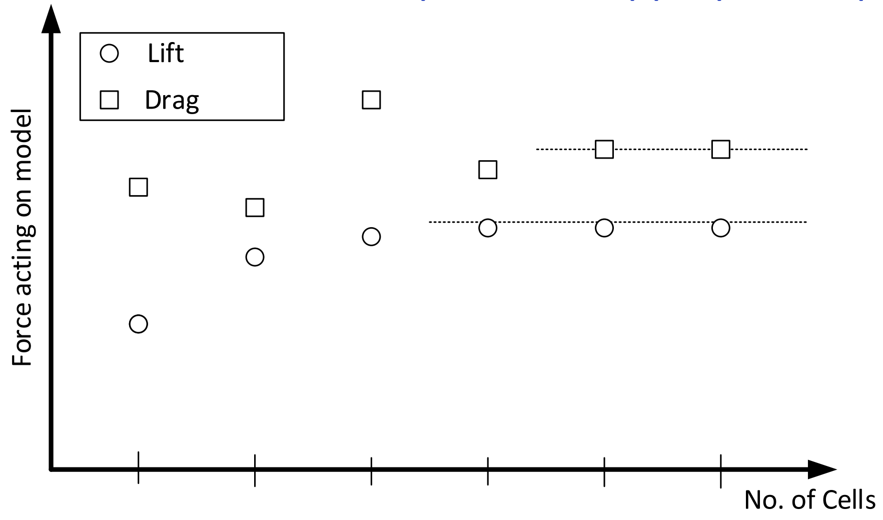


Edit the residuals file if you want to show variables other than `p` and `U`.

Dependency studies

How to demonstrate that a solution is accurate

- Best option is to compare to experimental data. → validation of accuracy
- Alternative is to show that solution is independent of method. → verification (demonstration that approach is correct)
  These are hypothesis tests of the form:
  "If my solution is independent of parameter **A**, changing parameter **A** will not have any effect on the solution."
  - Most common assessment is grid-independence study. (See example below)
  - Others that may be considered are dependency on solver, turbulence model (Note: these will give you different results and typically requires literature reference to pick most appropriate option)



- Monitor different parameters to confirm independence.
- Ideally parameter is monitored at a location that is highly sensitive flow and solution accuracy
- Inertial effects typically become independent first. Viscous effects are much more sensitive. Hence slower convergence of drag force.