Open∇FOAM

The open source CFD toolbox

Version 1.7.1

Foundation Training

OpenCFD Ltd

Notes v1.7.1 rev 7. 3/5/2011

Copyright and Disclaimer

Contents

1 Introduction

1.1 Overview

Plan of the course

Aim: Enable people to use OpenFOAM effectively and independently

- Will utilise the power of GNU/Linux (UNIX), using shell commands, e.g.

```
>> echo "Welcome to OpenFOAM"
Welcome to OpenFOAM
```

- Will view/edit code and case files, displayed with line numbers, e.g.

```
1  GNU GENERAL PUBLIC LICENSE
2     Version 2, June 1991
```

- Everything is demonstrated with cases users can follow on their machines
- Emphasis on how to explore OpenFOAM

What is OpenFOAM?

- Software for computational fluid dynamics (CFD) (and other continuum mechanics)...
- ...designed as a programmable toolbox...
- ...for simulation of real, 3-dimensional problems in science/engineering
- Freely available and open source, licensed under the GNU General Public Licence
- Produced by OpenCFD Ltd

A toolbox, not a black box

- Supplied with source code and compilers
- Customised applications can be created for specific problems...
- ...using functionality built into generic modules (libraries)

- Top level code represents the equations being solved, *e.g.*

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot \rho \mathbf{U}\mathbf{U} - \nabla \cdot \rho \mathbf{R} = -\nabla p$$

```
solve
(
      fvm::ddt(rho, U)
    + fvm::div(phi, U)
    + turbulence->divRhoR(U)
    ==
    - fvc::grad(p)
);
```

What is in a typical CFD software package?

- 1 (or a few) software executable(s)
- Example case files
- Documentation
- Data and configuration files

What is in the OpenFOAM distribution?

- 200+ executable applications, not a single executable
- Example case files
- Documentation
- Data and configuration files
- Source code files
- Shared-object libraries
- Compilation scripts
- Other scripts

OpenFOAM is different

- The source code is a key source of information itself
- It can be modified and recompiled
- Can be frequently extended and upgraded

⇒ Users benefit from being familiar with the OpenFOAM distribution

Installation

- At present there are 2 OpenFOAM distributions:

  Ubuntu/Debian installation of binaries and sources using the `apt` package manager; easy to install.

  General Linux distribution supplied as source code including third party software, requires compilation; more difficult to install.

- See the latest information:
  - `http://www.openfoam.com/download`: Installation overview
  - `http://www.openfoam.com/download/ubuntu.php`: Ubuntu installation information
  - `http://www.openfoam.com/download/source.php`: Source installation information

What is in the OpenFOAM distribution?

<installDir>	Installation directory
└ <packageDir>	OpenFOAM package directory
└ src	Source code files
└ applications	Application sources and executables
└ lib	Shared-object libraries
└ wmake	Compilation scripts
└ bin	Other scripts
└ tutorials	Example case files
└ doc	Documentation
└ etc	Data and configuration files

- <installDir> can be `$HOME/OpenFOAM`, `/opt`, ...
- <packageDir> can be `OpenFOAM-1.7.1` or `openfoam171`

Navigating the OpenFOAM distribution

- Environment variables are pre-defined for important OpenFOAM directories, e.g. `$WM_PROJECT_DIR`, in files in `etc`
- Quick changes of directory pre-defined using aliases, e.g. `alias foam='cd $WM_PROJECT_DIR'`

Directory	Description	Env. variable	Alias
<packageDir>	Installation dir.	$WM_PROJECT_DIR	foam
└ src	Source files	$FOAM_SRC	src
└ OpenFOAM	Main library source		foamsrc
└ finiteVolume	finiteVolume library		foamfv
└ applications	Applications source	$FOAM_APP	app
└ solvers	Solver apps	$FOAM_SOLVERS	sol
└ utilities	Utility apps	$FOAM_UTILITIES	util
└ tutorials	Example cases	$FOAM_TUTORIALS	tut

OpenFOAM user directory

- OpenFOAM expects a user directory to exist in the `~/OpenFOAM` directory (~ = home dir)
- Named `${LOGNAME}-1.7.1` — i.e. `ubuntu-1.7.1` for user 'ubuntu'
- Environment variable `$WM_PROJECT_USER_DIR` set to the user directory
- Version numbered user directories provides convenient version control
- Can mirror the installation directory, e.g. solvers located in:

  `/opt/openfoam171/applications/solvers` in the installation
  `~/OpenFOAM/ubuntu1.7.1/applications/solvers` in the user's files
- Case files are stored in a `run` subdirectory
  - `~/OpenFOAM/ubuntu1.7.1/run`
  - env. variable: `$FOAM_RUN`
  - pre-defined alias to change directory: `run`

1.2 Applications

Applications

- OpenFOAM is distributed with ~200 applications, in `applications` directory
- Split into `solvers` and `utilities` subdirectories, where

solvers simulate specific problems in CFD and other engineering mechanics

utilities perform pre- and post-processing tasks data manipulations, visualisation, mesh processing, ...

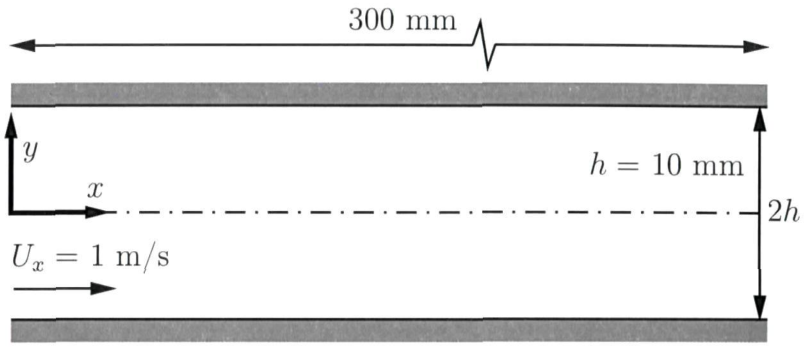
- The `solvers` and `utilities` directories are organised into subdirectories whose names represent types of flow, utility, ...
- For a given application, e.g. `icoFoam`
  - The source code is in a directory named `icoFoam` see `$FOAM_SOLVERS/incompressible/icoFoam`
  - The main `.C` source file is named `icoFoam.C`
  - The executable is named `icoFoam`
- The header of the main `.C` file has a description of the application's use, e.g.

```
Description
    Transient solver for incompressible, laminar flow of Newtonian fluids.
```

2 Flow between parallel plates

2.1 Overview and case files

Flow between parallel plates



- Transient solution
- Laminar, $\nu = 10^{-4} \mathrm{m}^2/\mathrm{s}^2$, $\mathrm{Re} = 400$
- Isothermal
- Incompressible
- Plug flow inlet $U_x = 1$ m/s

$$U_x = -\frac{(\nabla p)_x}{2\nu}(h^2 - y^2)$$

$$(\nabla p)_x = -\frac{3\nu}{h^2} Q$$

where Q =volumetric flow rate / unit area

Setting up a case in OpenFOAM

`icoFoam`	→	• Choose solver
└ `cavity`	→	• Copy example case for that solver
└ `constant`	{	• Create/modify mesh • Select models and properties
└ `0`	→	• Initialise fields & BCs for p, $\mathbf{U}$, *etc.*
└ `system`	→	• Set case controls, *e.g.* Δt, schemes, tolerances

Solver and case

- Solver: `icoFoam` and `pisoFoam` are both suitable
  - from the `$FOAM_SOLVERS/incompressible` directory
  - `Description` in `.C` file fits the requirements
- Let's use `icoFoam` — a stripped down solver we can examine later
- Create a new case called `parallelPlate` from existing `icoFoam` case called `cavity` — in `$FOAM_TUTORIALS/incompressible/icoFoam`
- Make a local copy of the `cavity` case, renaming it `parallelPlate`

```
>> run
>> cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity parallelPlate
```

- Change to the `parallelPlate` case directory

```
>> cd parallelPlate
```

Case files

- In OpenFOAM case data is stored in a set of files within a case directory, not in a single case file
- The `case directory` can be given any name, e.g. `cavity` for cavity flow

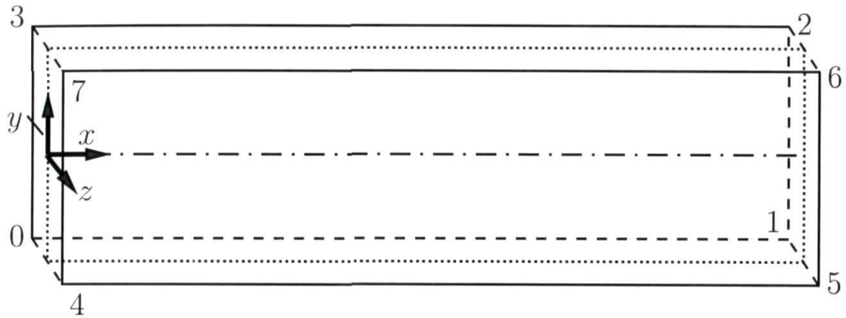
<case>, e.g. `cavity`	Case directory
└ `system`	Dictionaries of solution parameters
└ `controlDict`	Time and data input/output control
└ `fvSchemes`	Numerical schemes
└ `fvSolution`	Linear-solver parameters, e.g. tolerances
└ `constant`	Constant (unchanging) data
└ `polyMesh`	Mesh
└ `...Properties`	Physical properties, e.g. `transportProperties`
└ Time directories	Time-varying data, directories named after simulated time, e.g. `0`, `0.1`, `0.2`, ...

2.2 Meshing

Mesh generation using `blockMesh`

- `blockMesh` is a simple mesh generator using blocks
- Allows multiple blocks and curved edges
- Configured by a `blockMeshDict` file in the `constant/polyMesh` directory of a case
- Produces a 3D mesh of hexahedral cells
- OpenFOAM always uses 3D meshes, handling 1D, 2D and axisymmetric cases using special boundary conditions
- There are lot of examples in the tutorials — copy something suitable for your needs and modify it

`blockMeshDict` configuration: `vertices`



- A list of `vertices` is specified
- All values are scaled by `convertToMeters`
- Edit these in the `blockMeshDict` file
- For convenience: z-depth $= 0.1\times y$-height

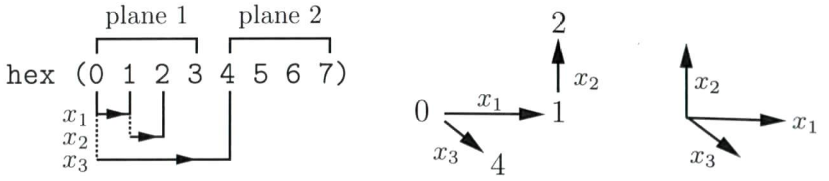
```
convertToMeters 0.01;

vertices
(
    (0  -1 -0.1)
    (30 -1 -0.1)
    (30  1 -0.1)
    (0   1 -0.1)
    (0  -1  0.1)
    (30 -1  0.1)
    (30  1  0.1)
    (0   1  0.1)
);
```

`blockMeshDict` configuration: `blocks`

```
blocks
(
    hex (0 1 2 3 4 5 6 7) // vertex list
    (300 20 1)            // no. cells in each dir.
    simpleGrading (1 1 1) // cell expansion ratios
);
```

- Block description begins with `hex` followed by list of vertices
- Order of vertices is critical
- The 1st 4 vertices describe one plane; the 2nd 4 describe another plane

- The order defines a local coordinate system (x_1, x_2, x_3)

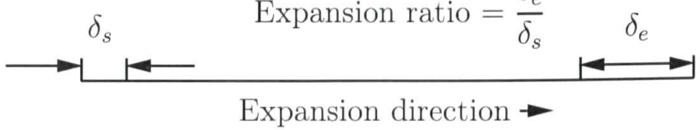


- The resulting (x_1, x_2, x_3) local coordinate system must be right-handed
  - looking down the Ox_3 axis — with O nearest — an arc from the Ox_1 axis to the Ox_2 axis is in a clockwise sense.
- *e.g.*, `(5 4 7 6 1 0 3 2)` would also work; but `(1 0 3 2 5 4 7 6)` would produce a left-handed block

`blockMeshDict` configuration: `blocks` (2)

- `(300 20 1)` gives the number of cells in each direction of the block's local coordinate system
  - Here, only 1 is needed in the z-direction because the case is 2D
- The final entries specify cell grading
- Either `simpleGrading` or `edgeGrading`
  - `simpleGrading`: requires 3 expansion ratios
  - `edgeGrading`: requires 12 expansion ratios — see User Guide for details
- Expansion ratios are ratios of cell lengths from end to start cell; grading is linear in between

$$\text{Expansion ratio} = \frac{\delta_e}{\delta_s}$$



`blockMeshDict` configuration: `patches`

```
patches
(
    patch inlet
    (
        (0 4 7 3)
    )
    patch outlet
    (
        (2 6 5 1)
    )
    wall walls
    (
        (3 7 6 2)
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
```

- Patch defined by "<type> <name> <faceList>"
- <type> = `patch` is the default type
- <type> = `empty` for 2D front and back planes
- <type> = `wall` for walls (or `patch`, if no turbulence modelling)
- <name>: used to identify the patch
- <faceList>: set of faces, each defined by 4 vertex points in order along a path around the face edges

Running `blockMesh`

- In a terminal type the utility name with the `-help` option — to display usage

```
>> blockMesh -help

Usage: blockMesh [-region region name] [-case dir] [-blockTopology] [-help]
[-doc] [-srcDoc]
```

- Requires no arguments if executed from within the case directory
- Otherwise use `-case <caseDir>` to specify the case directory
- ⇒ to run `blockMesh` on the `parallelPlate` case

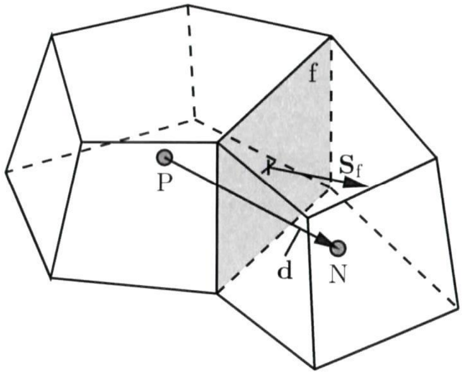
```
>> cd $FOAM_RUN/parallelPlate
>> blockMesh
```

- That generates the mesh, written to a set of files in `constant/polyMesh`

Meshes in OpenFOAM

- OpenFOAM operates in a 3 dimensional Cartesian coordinate system
  - 1- and 2- dimensional and axi-symmetric simulated on 3-D meshes by applying special boundary conditions
- Arbitrary polyhedral cells in 3-D, bounded by arbitrary polygonal faces
  - A cell can have an unlimited number of faces
  - A face has an unlimited number of edges
  - No restriction on face alignment
  - Internal faces intersect two cells only
  - Boundary faces belong to one cell
- Offers great freedom in mesh generation and manipulation
- Known as `polyMesh` in OpenFOAM

The polyMesh description

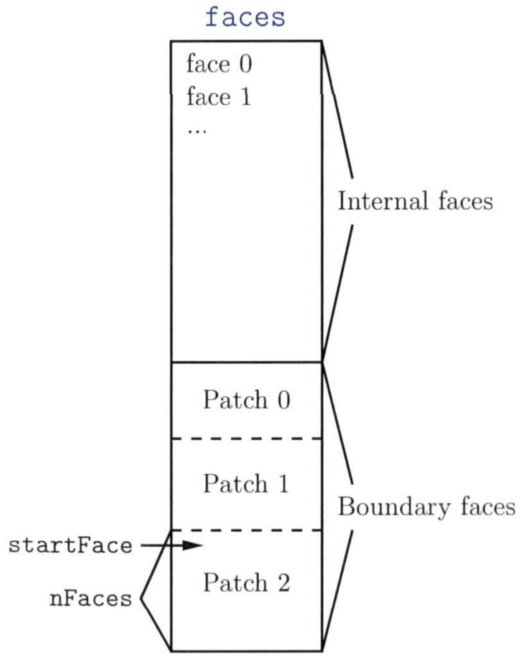


- Face-based description
- Each face is assigned an owner (P) and neighbour (N) cell

  - If a boundary face, neighbour index = -1

`constant`	
└ `polyMesh`	Mesh
└ `points`	List of points (vectors)
└ `faces`	List of faces, each face being a list of indices to `points`
└ `owner`	List of face owner-cell labels, the index = face index
└ `neighbour`	List of face neighbour-cell labels, the index = face index
└ `boundary`	List of patches, each a dictionary, declared by name

The faces list



- The list of faces is ordered in OpenFOAM
- Internal faces appear first
- Boundary faces then appear, ordered according to patches
- Each patch described by `startFace` and `nFaces`

The boundary file

- The boundary file can be viewed and edited
- Users can modify patch names and types

```
4
(
inlet                    // boundary patch name
{
    type patch;          // patch type
    nFaces 20;           // no. of faces in patch
    startFace 11680;     // index of first face
}                        // => inlet patch is faces 11680-11699

outlet
{
    ... more entries ...
}

walls
{
    type wall;           // wall type
    ... more entries ...
}

frontAndBack
{
    type empty;          // empty type
    ... more entries ...
}
)

// ************************************************************************* //
```

2.3 Case setup and running

parallelPlate: setting initial/boundary conditions

- Field data are stored in time directories, e.g. 0, 0.1, 0.2, ...
- Usually initial conditions are stored at $t = 0$, *i.e.* directory 0
- The icoFoam solver reads field data for pressure (p) and velocity (U)
- Let's look at these files...

parallelPlate: pressure field

$FOAM_RUN/parallelPlate/0/p:

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

internalField   uniform 0;

```



```
boundaryField
{
    inlet
    {
        type    zeroGradient;
    }

    outlet
    {
        type    fixedValue;
        value   uniform 0;
    }

    walls
    {
        type    zeroGradient;
    }

    frontAndBack
    {
        type    empty;
    }
}
```

- Dimensions are m^2/s^2, *i.e.* kinematic pressure, in icoFoam
- Internal (and boundary) fields can be:

  uniform a single value

  nonuniform all values in a List
- type describes the numerical boundary condition
- walls are zeroGradient
- outlet is fixedValue which requires a value; can be anything (pressure is relative), use 0 for convenience
- frontAndBack planes of a 2D case must be empty to match their base type

parallelPlate: velocity field

$FOAM_RUN/parallelPlate/0/U:

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type    fixedValue;
        value   uniform (1 0 0);
```

```
    }

    outlet
    {
        type    zeroGradient;
    }

    walls
    {
        type    fixedValue;
        value   uniform (0 0 0);
    }

    frontAndBack
    {
        type    empty;
    }
}
```

- Velocity is a vector field
- No-slip walls:
  - a `fixedValue` type
  - requires a `value`
- `uniform (1 0 0)` on `inlet`
- `uniform (0 0 0)` on `walls`
- `outlet` is `zeroGradient`
- File syntax is easy to interpret!

Case file syntax: general

- Follows general principles of C++ source code
- Files have free form
  - No particular meaning assigned to a column
  - No need to indicate continuation across lines
- Lines have no particular meaning except to a `//` comment delimiter which ignores text that follows it until the end of line
- Enclose text between `/*` and `*/` delimiters to comment over multiple lines

Case file syntax: specific

- OpenFOAM uses a flexible I/O format based mainly on a keyword syntax

```
<keyword> <dataEntry1> ... <dataEntryN>; // usually 1 entry
```

- A data entry can be a string (`"hello"`), word (`hello`), integer (`3`) scalar (`3.14`), ...
- ... a dictionary: curly brackets `{...}`

```
{
    ... keyword entries ...
}
```

- ... a list: round brackets `(...)`

```
List<Type> // optional, Type = elements of list, e.g. scalar
<n>        // optional, number of entries
(
    ... dataEntry1 ...
);
```

- ... a dimensionSet: square brackets `[...]`

```
[ 1 -1 -2 0 0 0 0 ] // [ Mass Len. Time Temp. Qnt. Cur. Lum. ]
```

Dimensions and dimension checking

- Fields and properties have dimensions associated with them
- Specified as powers of: 1) mass; 2) length [L]; 3) time [T]; 4) temperature; 5) quantity; 6) current; 7) luminosity
- Velocity is L^1T^{-1}, *i.e.* `[0 1 -1 0 0 0 0]`
- For $+$, $-$ and $=$ operations, solver stops if dimensions are not the same
- For $\times$ and $/$ operations, dimensions are modified
- Dimensions only relate to a specific system of units, *e.g.* SI, if physical constants (*e.g.* R, p_{std}) are used
- Physical constants read from the global `controlDict` file in `etc` directory `$WM_PROJECT_DIR/etc/controlDict`:

```
DimensionedConstants
{
    // SI units
    //- Universal gas constant [J/(kmol K)]
    R    8314.51;
    ...
    /* USCS units
    //- Universal gas constant [lbm ft2/(s2 kmol R)]
    R    3406.78;
    ...
    */
}
```

parallelPlate: physical properties

- What properties do I need to set in an `icoFoam` simulation?
- Look in the case `constant` directory

```
>> ls -1 constant
polyMesh
transportProperties
```

- Most properties files are named `...Properties`, so here just `transportProperties`
- It contains only an entry for kinematic viscosity ν

```
nu              nu [0 2 -1 0 0 0 0] 1e-04;
```

- The keyword `nu` requires a `dimensionedScalar` entry, which includes

  `word` "nu", used for internal naming of other fields

  `dimensionSet` specifying m^2/s

  `scalar` a value set to `1e-04`

- $\nu = 10^{-4}$ corresponds to $\mathrm{Re} = 400$, for $|\mathbf{U}| = 1$ and $h = 0.01$

$$\mathrm{Re} = \frac{L|\mathbf{U}|}{\nu} \quad \text{where } L = 4h$$

parallelPlate: control parameters

`$FOAM_RUN/parallelPlate/system/controlDict`:

Entry	Value	Meaning
`startFrom`	`startTime;`	Start time $t = 0$
`startTime`	`0;`	
`stopAt`	`endTime;`	End time $t = 0.3$
`endTime`	`0.3;`	
`deltaT`	`0.0002;`	– Time step $\Delta t = 0.0002$
`writeControl`	`runTime;`	Writes out every 0.05s
`writeInterval`	`0.05;`	
`purgeWrite`	`0;`	– Does not rewrite over time directories
`writeFormat`	`ascii;`	Writes ASCII, 6 sig. figs
`writePrecision`	`6;`	
`writeCompression`	`uncompressed;`	– Writes uncompressed
`timeFormat`	`general;`	Time dir. naming format, 6 sig. figs
`timePrecision`	`6;`	
`runTimeModifiable`	`yes;`	– Allows modification of settings during run

Parameter options

- Q: How does a user find out what entries are valid for a particular keyword?
- A: If keyword entry is invalid, OpenFOAM prompts the user with valid entries, e.g.
  - Setting "`stopAt xxx;`" in `controlDict`, a solver would return:

```
xxx is not in enumeration:
4
(
endTime
writeNow
noWriteNow
nextWrite
)

file: ::stopAt at line 24.
```

  - Entries for "`stopAt`" are clearly listed

parallelPlate: other control parameters

- What other control parameters can I set for an `icoFoam` simulation?
- Look in the case `system` directory

```
>> ls -1 system
controlDict
fvSchemes
fvSolution
```

- Let's discuss details of `fvSchemes` and `fvSolution` later

parallelPlate: running the case - initialising

```
>> icoFoam
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 1.7.1
Exec   : icoFoam
Date   : Apr 15 2010
Time   : 13:07:42
Host   : manfred
PID    : 5668
Case   : $FOAM_RUN/parallelPlate
nProcs : 1
SigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Reading transportProperties

Reading field p

Reading field U

Reading/calculating face flux field phi
```

- Creates mesh, reads physical properties and fields

parallelPlate: running the case - start

```
continued...

Starting time loop

Time = 0.0002

Courant Number mean: 0 max: 0.4
DILUPBiCG:  Solving for Ux, Initial residual = 1, Final residual = 1.30079e-07, No Iterations 2
DILUPBiCG:  Solving for Uy, Initial residual = 0, Final residual = 0, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1, Final residual = 7.75518e-07, No Iterations 163
time step continuity errors : sum local = 5.17011e-10, global = 3.2103e-13, cumulative = 3.2103e-13
DICPCG:  Solving for p, Initial residual = 0.000146046, Final residual = 5.91098e-07, No Iterations 140
time step continuity errors : sum local = 2.41625e-07, global = -1.47373e-09, cumulative = -1.47341e-09
ExecutionTime = 0.31 s  ClockTime = 1 s

Time = 0.0004

Courant Number mean: 0.102403 max: 0.401517
```



```
DILUPBiCG:  Solving for Ux, Initial residual = 0.991042, Final residual = 6.13964e-07, No Iterations 2
DILUPBiCG:  Solving for Uy, Initial residual = 0.338071, Final residual = 1.16197e-06, No Iterations 2
DICPCG:  Solving for p, Initial residual = 0.00191905, Final residual = 8.00036e-07, No Iterations 147
time step continuity errors : sum local = 2.77161e-07, global = -3.18057e-09, cumulative = -4.65398e-09
DICPCG:  Solving for p, Initial residual = 0.00481002, Final residual = 6.91451e-07, No Iterations 147
time step continuity errors : sum local = 1.78984e-08, global = 1.30046e-10, cumulative = -4.52393e-09
ExecutionTime = 0.45 s  ClockTime = 1 s

Time = 0.0006

Courant Number mean: 0.102409 max: 0.403097
DILUPBiCG:  Solving for Ux, Initial residual = 0.178209, Final residual = 5.56988e-07, No Iterations 2
DILUPBiCG:  Solving for Uy, Initial residual = 0.197402, Final residual = 7.33383e-07, No Iterations 2
DICPCG:  Solving for p, Initial residual = 0.0168414, Final residual = 6.50678e-07, No Iterations 146
time step continuity errors : sum local = 9.03965e-10, global = 9.43093e-12, cumulative = -4.5145e-09
DICPCG:  Solving for p, Initial residual = 0.0013938, Final residual = 8.0808e-07, No Iterations 141
time step continuity errors : sum local = 1.07365e-09, global = 1.19455e-11, cumulative = -4.50255e-09
ExecutionTime = 0.59 s  ClockTime = 1 s
```

parallelPlate: running the case - end

```
continued...

Time = 0.2998

Courant Number mean: 0.102818 max: 0.596245
DILUPBiCG:  Solving for Ux, Initial residual = 1.44075e-07, Final residual = 1.44075e-07, No Iterations 0
DILUPBiCG:  Solving for Uy, Initial residual = 1.69237e-06, Final residual = 1.69237e-06, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1.25578e-06, Final residual = 9.8682e-07, No Iterations 1
time step continuity errors : sum local = 2.37135e-10, global = -7.3208e-11, cumulative = -1.43971e-08
DICPCG:  Solving for p, Initial residual = 1.23148e-06, Final residual = 9.78693e-07, No Iterations 1
time step continuity errors : sum local = 2.35182e-10, global = -6.76382e-11, cumulative = -1.44647e-08
ExecutionTime = 106.03 s  ClockTime = 119 s

Time = 0.3

Courant Number mean: 0.102818 max: 0.596246
DILUPBiCG:  Solving for Ux, Initial residual = 1.42147e-07, Final residual = 1.42147e-07, No Iterations 0
DILUPBiCG:  Solving for Uy, Initial residual = 1.67561e-06, Final residual = 1.67561e-06, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1.32791e-06, Final residual = 9.20289e-07, No Iterations 47
time step continuity errors : sum local = 2.21148e-10, global = -8.8838e-13, cumulative = -1.44656e-08
DICPCG:  Solving for p, Initial residual = 1.17776e-06, Final residual = 6.49667e-07, No Iterations 1
time step continuity errors : sum local = 1.56118e-10, global = 1.17773e-11, cumulative = -1.44538e-08
ExecutionTime = 106.14 s  ClockTime = 119 s

End
```

Courant Number

```
Time = 0.01
Courant Number mean: 0.102409 max: 0.403097
```

- `Courant Number` $Co = U\Delta t/\Delta x$
- U is the flow speed in a given direction
- Δx is the cell length in a given direction
- Δt is the time step
- $Co > 1$ means the flow can pass through a cell within one time step
- Some numerical methods/algorithms stable only when Co is below a particular limit

parallelPlate: screen output

```
DILUPBiCG: Solving for Ux, Initial residual = 0.178209, Final residual = 5.56988e-07,
  No Iterations 2
DILUPBiCG: Solving for Uy, Initial residual = 0.197402, Final residual = 7.33383e-07,
  No Iterations 2
```

- `DILUPBiCG` is the chosen linear-solver for the `U` equation
- Linear-solver `decouples vector equation` for `U` into components `Ux` and `Uy`
- Iterates until `Final residual` < `tolerance`
- `fvSolution::solvers` (`solvers` subdictionary of `fvSolution`):

```
        U
        {
            solver          PBiCG;
            preconditioner  DILU;
            tolerance       1e-05;
            relTol          0;
        }
```

parallelPlate: screen output (2)

```
DICPCG: Solving for p, Initial residual = 0.0168414, Final residual = 6.50678e-07,
  No Iterations 146
time step continuity errors : sum local = 9.03965e-10, global = 9.43093e-12,
  cumulative = -4.5145e-09
DICPCG: Solving for p, Initial residual = 0.0013938, Final residual = 8.0808e-07,
  No Iterations 141
time step continuity errors : sum local = 1.07365e-09, global = 1.19455e-11,
  cumulative = -4.50255e-09
ExecutionTime = 0.03 s ClockTime = 0 s
```

- `DICPCG` is the chosen linear-solver for the `p` equation
- p equation solved 2 times according to the PISO correctors `nCorrectors`
- Errors in mass continuity presented as:
  - `sum local`: sum of magnitudes of continuity errors in each cell
  - `global`: across the boundary of the domain
  - `cumulative`: `global` error accumulated over time

`fvSolution::solvers`:

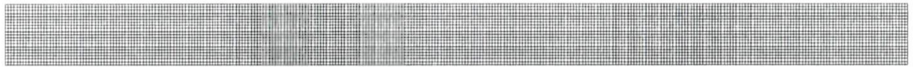

```
        p
        {
            solver          PCG;
            preconditioner  DIC;
            tolerance       1e-06;
            relTol          0;
        }
```

`fvSolution::PISO`:

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

2.4 Post-processing

Post-processing a mesh



- Run `paraFoam`:

  `>> paraFoam`

- Check the Mesh Parts and Volume Fields box
- Click the Apply button
- Select the Display panel
- In Style: select `Wireframe` representation
- Set `Color by Solid Color`
- Edit `Set Solid Color`, *e.g.* black

Useful settings in paraview

In `Edit -> View Settings`:

- General panel:

  `Set Background Color` white for printed material

Use Parallel Projection usual for CFD, especially 2D cases

- Lights panel:

  Default Light Set to on, strength 1, white

- Annotation panel:

  Orientation Axes Set to on, Interactive, Axes Label Color black

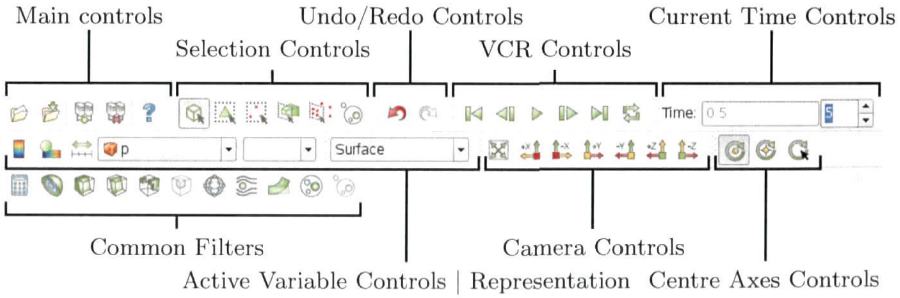
In `Edit -> Settings`:

- Render View - General panel:

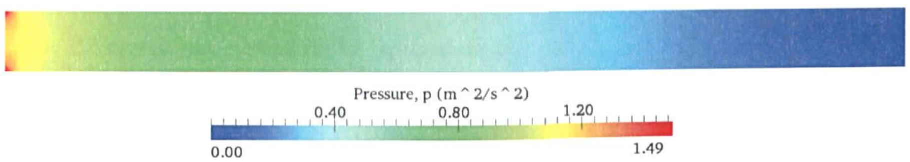
  `Level of Detail (LOD)` controls the rendering while image is manipulated; 'smaller numbers speed things up'

- Colors panel controls global colour settings

Paraview toolbars



Pressure field plot



- Current Time Controls toolbar:

  - Change time to $t = 0.3$ s

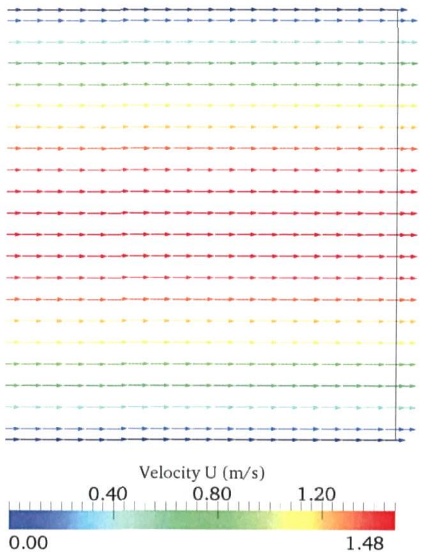
- Active Variable Controls toolbar:
  - `Color by` → ◦p
  - `Surface` representation.
  - Rescale to Data Range if required

Also:

- `Color by` → ◦p attributes single value for pressure to each cell
- For a colour bar, select Toggle Color Legend Visibility and configure with Edit Color Map

Velocity vector plot

- We want vectors at cell centres
  - Select `Cell centers` from the top `Filter` menu
  - Click Apply
- Select `Glyph` from the `Filter` menu
  - Open Parameters panel
  - `Glyph` → `Arrow`
  - `Scale Mode` → `off`
  - Specify Scale Factor 0.001
  - Set Max. no. of points to 20,000
- Colour the glyphs by velocity magnitude
  - Display panel → `Color by U`
- Edit Color Map → Color Legend.
  - Uppercase Times Roman font
  - Deselect Automatic Label Format and enter `%-#6.2f` in Label Format to fix to 2 sig. figs.



Sampling data along a line

- We want to plot graphs of velocity profile and compare with the analytical solution
- The GUI post-processing is not particularly useful; graphing is not very good
- Instead we can use the sample utility
- Easy to automate/configure with a sampleDict configuration file
- Examples can be found in the release

```
>> find $FOAM_TUTORIALS -name sampleDict
```

- Let's copy one of those

```
>> cp $FOAM_TUTORIALS/compressible/rhoCentralFoam/shockTube/system/sampleDict system
```


Configuring the sample utility

- We will monitor the velocity across the channel at $x = 0.29$ (near the outlet)
- Change the sets entry in the sampleDict file

```
sets
(
    acrossFlow
    {
        type    midPoint;     // samples at mid-point between faces
        start   ( 0.29 -0.011 0 );
        end     ( 0.29  0.011 0 );
        axis    y;    // prints the y ordinate at each sample point
    }
);
```

- Change the fields entry in the sampleDict file

```
fields ( U ); // sampling velocity field U
```

- Execute the sample utility

```
>> sample
```

- Results are written into sets directory

Plotting results using gnuplot

- We will use gnuplot to plot graphs
- Can be run with configuration files
- Pre-configured files in $FOAM_RUN/EXAMPLES/gnuplot
- Copy this directory into our system directory

```
>> cp -r $FOAM_RUN/EXAMPLES/gnuplot system
```

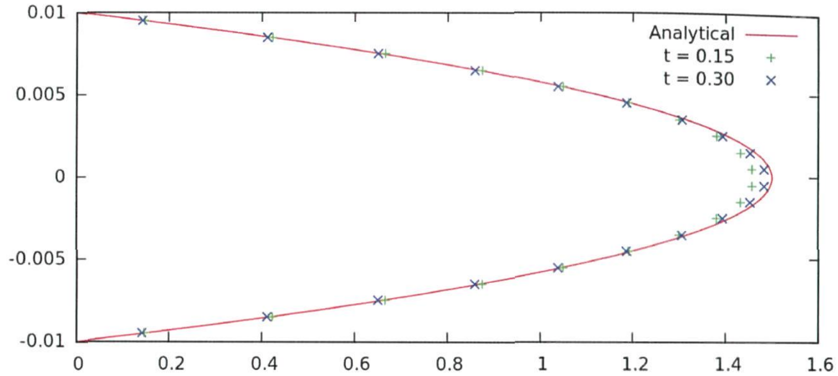
- Use the configuration file called plot_parallelPlate:

```
dPdx = -3
mu   = 1e-04
h    = 1e-02
set parametric
plot [-h:h] -dPdx/mu/2*(h**2 - t**2),t title "Analytical", \
     "sets/0.15/acrossFlow_U.xy" using 2:1 title "t = 0.15", \
     "sets/0.3/acrossFlow_U.xy" using 2:1 title "t = 0.30"
```

Plotting results using `gnuplot` (2)

- Execute `gnuplot`

```
>> gnuplot
gnuplot> load "system/gnuplot/plot_parallelPlate"
```



2.5 Mapping one case to another

Increasing mesh resolution

- Clone the `parallelPlate` case to make `parallelPlateFine`

```
>> run                                          # alias for cd $FOAM_RUN
>> mkdir parallelPlateFine
>> cp -r parallelPlate/system parallelPlateFine
>> cp -r parallelPlate/constant parallelPlateFine
>> cd parallelPlateFine
```

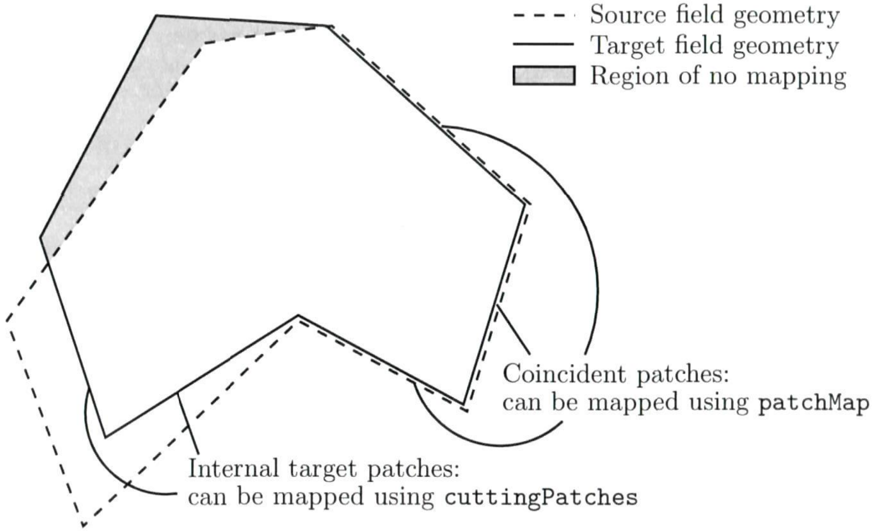
- Refine mesh to 40 cells across the channel
  - In `blockMeshDict` change `blocks` to

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (300 40 1) simpleGrading (1 1 1)
);
```

  - Run `blockMesh`

Field mapping

- Fields can be mapped from one mesh to another with `mapFields` utility
- Mapping between conforming geometries/fields done with `-consistent` option
- Otherwise, mapping on patches specified in `mapFieldsDict` file



parallelPlateFine: mapping, then running case in background

- Map consistent fields from `parallelPlate` case at $t = 0.3$

```
>> mapFields ../parallelPlate -consistent -sourceTime 0.3
```

- Fields are written into `0` directory
- Run the case using `foamJob` script to output to a `log` file

```
> foamJob icoFoam
Application : icoFoam
Executing: $WM_PROJECT_DIR/applications/bin/linuxGccDPOpt/icoFoam
           > log 2>&1 &
```

- View the `log` file in the terminal window or an editor. Useful is:


```
>> tail -f parallelPlateFine/log
```

- Terminate with `Control-C`

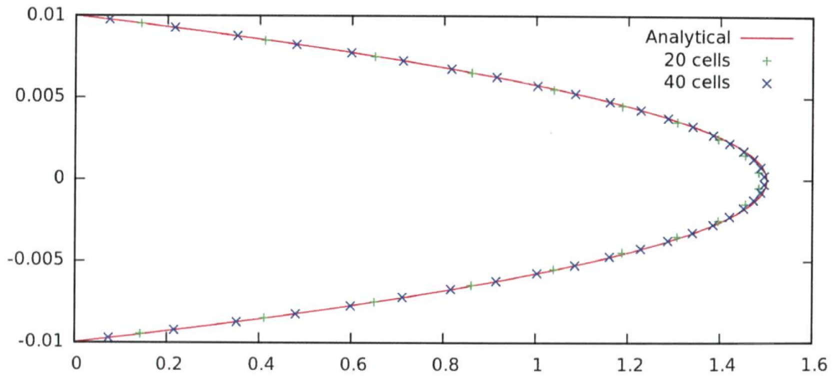
Results from fine mesh

- Run the `sample` on `parallelPlateFine`

```
>> sample
```

- Plot the results; note the better results from the finer mesh

```
>> gnuplot
gnuplot> load "system/gnuplot/plot_parallelPlateFine"
```



2.6 Example boundary conditions

Boundary conditions (BCs) in OpenFOAM

- Geometry boundary is broken into patches on which BCs are applied
- Patch types (BCs) relating to **geometry**, *e.g.* symmetry plane, are ascribed on the mesh in OpenFOAM
  - Specified through `type` keyword in `constant/polyMesh/boundary`
- For **fields**, we specify actual numerical BCs, *e.g.* `fixedValue` for `U`, `zeroGradient` for `p` (rather than "inlet")

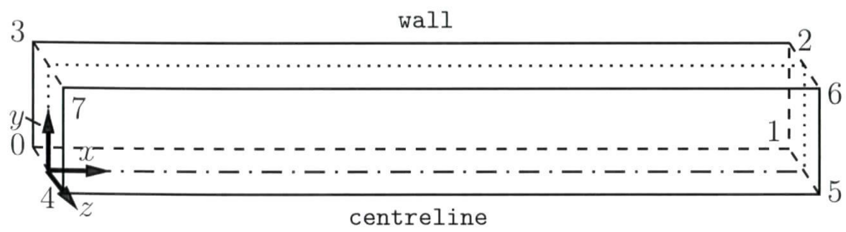
  - Specified through `type` keyword in `boundaryField` of field files, *e.g.* `0/p`
  - Can be simple `fixedValue`, `zeroGradient`, `fixedGradient`, ...
  - ... or a more complex "derived" patch type `flowRateInletVelocity`, `totalPressure`, ...

Geometry boundary types

- Boundary patches are given a geometric type in OpenFOAM
  - See the `boundary` file of a mesh
- The default type is `patch`
- There are other special types relating to geometry or data communcation

Selection Key	Description
`patch`	generic patch
`symmetryPlane`	plane of symmetry
`empty`	front and back planes of a 2D geometry
`wedge`	wedge front and back for an axi-symmetric geometry
`cyclic`	cyclic plane
`wall`	wall — used for wall functions in turbulent flows
`processor`	inter-processor boundary

Parallel plate flow with symmetry plane



- Laminar flow is symmetric; let's put a symmetry plane along the flow centreline

1. Create a `parallelPlateSymm` case by cloning the `parallelPlate` case; for convenience, type

```
>> run
>> cp -r parallelPlate parallelPlateSymm
>> cd parallelPlateSymm
>> rm -rf 0.*
```

Parallel plate flow with symmetry plane (2)

2. Edit the `constant/polyMesh/blockMeshDict` file
   - `vertices`: change the "`-1`" y-ordinate to "`0`"
   - `patches`: create the `centreline` patch and modify the `walls` patch by:

```
wall walls
(
    (3 7 6 2)
)
symmetryPlane centreline
(
    (1 5 4 0)
)
```

3. Run `blockMesh`
4. Edit the `0/p` and `0/U` files; in `boundaryField` sub-dictionary, add a new patch entry

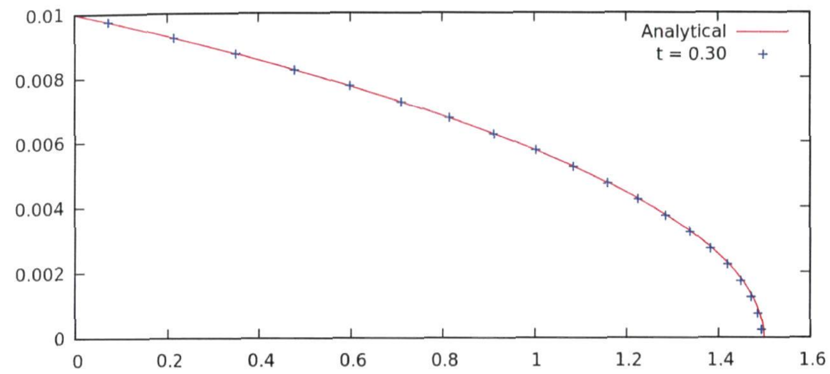
```
centreline { type symmetryPlane; }
```

5. Run `icoFoam`

Parallel plate flow with symmetry plane (3)

6. Run `sample`
7. Plot the results using `gnuplot`

```
>> gnuplot
gnuplot> load "system/gnuplot/plot_parallelPlateSymm"
```

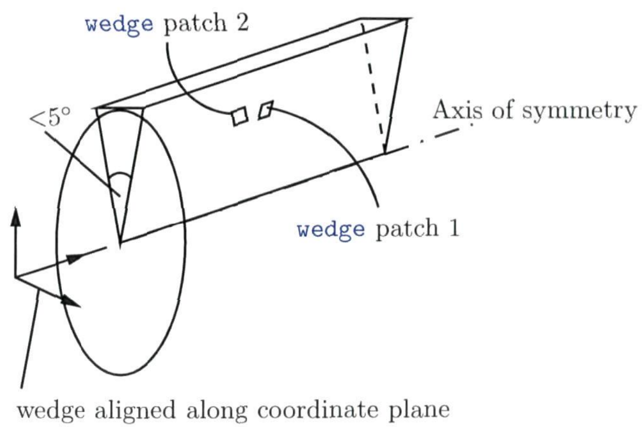



Poiseuille flow (1)

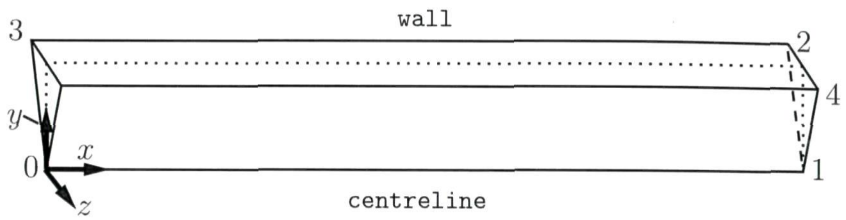
- Laminar flow in a cylinder has the analytical solution (Poiseuille):

$$U_x = -\frac{(\nabla p)_x}{4\nu}(R^2 - y^2) \qquad (\nabla p)_x = -\frac{8\nu}{R^2}Q$$

- Can be simulated as 2D axisymmetric using `wedge` patches
- Define a block with 2 `wedge` patches



Poiseuille flow (2)



1. Create a poiseuille case by cloning the parallelPlateSymm case

```
>> run
>> cp -r parallelPlateSymm poiseuille
```

2. Vertex pairs are collapsed along the centreline — blockMesh supports this; edit the constant/polyMesh/blockMeshDict file

```
vertices
(
    (0   0  0)
    (30  0  0)
    (30  1 -0.01) // approx tan(0.6 deg)
    (0   1 -0.01)
    (30  1  0.01)
    (0   1  0.01)
);
```

Poiseuille flow (3)

3. Modify the blocks and patches accordingly

```
blocks
(
    hex (0 1 2 3 0 1 4 5) (300 20 1) simpleGrading (1 1 1)
);

patches
(
    patch inlet  ( (0 0 5 3) )
    patch outlet ( (2 4 1 1) )
    wall  walls  ( (3 5 4 2) )
    wedge front  ( (0 1 4 5) )
    wedge back   ( (0 3 2 1) )
);
```

4. Run blockMesh

5. Edit the 0/p and 0/U files; in boundaryField sub-dictionary:

- add new patch entries

```
front { type wedge; }
back  { type wedge; }
```

- frontAndBack and centreline can be removed

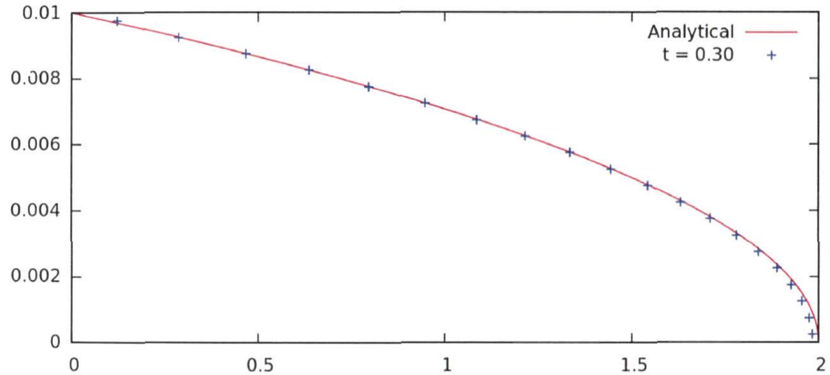
6. Run icoFoam

Poiseuille flow (4)

7. Run sample

8. Plot the results using gnuplot

```
>> gnuplot
gnuplot> load "system/gnuplot/plot_poiseuille"
```



Parallel plate flow: pressure inlet BC (1)

- Let's try specifying pressure at the inlet instead of velocity
- Create a parallelPlatePinlet case by cloning the parallelPlateSymm case

```
>> run
>> cp -r parallelPlateSymm parallelPlatePinlet
```

- Change the inlet BC to fixedValue in the 0/p file
- A value of 0.9 gives $(\nabla p)_x = (0.0 - 0.9)/0.3 = -3$ as before


```
inlet
{
    type        fixedValue;
    value       uniform 0.9;
}
```

- Recall that $(\nabla p)_x \propto Q$: we can specify $(\nabla p)_x$ or Q, not both
- If we now run with `fixedValue` on velocity, the problem is overspecified
  - Sudden jumps in p and $\mathbf{U}$ will appear at the inlet if inlet values are mismatched (*i.e.* $(\nabla p)_x$ and Q don't equate)
  - Can cause the code to blow up

Parallel plate flow: pressure inlet BC (2)

- Change the `inlet` BC to `pressureInletVelocity` in the `0/U` file

```
inlet
{
    type        pressureInletVelocity;
    value       uniform (0 0 0);
}
```

- `pressureInletVelocity` is a `fixedValue` BC that sets $\mathbf{U} = \phi \mathbf{n}_\mathrm{f}/|\mathbf{S}_\mathrm{f}|$ (ϕ = flux; $\mathbf{n}_\mathrm{f}$ = unit face area vector; $|\mathbf{S}_\mathrm{f}|$ = face area magnitude)
- Pressure gradient accelerates the flow to steady-state
- $\Rightarrow$ solution slow to converge, so set `endTime` (in `controlDict`) to 3
- Run `icoFoam`
- Smooth solution produced
- Profile is same along the length, *i.e.* no plug flow (constant velocity profile) at inlet
- ...but convergence was much slower than with velocity inlet

2.7 Introduction to turbulence modelling

Increasing the Reynolds number

- Aim: to run the Poiseuille case with $\mathrm{Re} = 10^4$
- Schlichting *Boundary Layer Theory* gives an approximate solution for the velocity profile for this case

$$U_x \approx \frac{5}{4} Q \left(1 - \frac{y}{R}\right)^{1/7} \quad \text{for} \quad 10^4 < \mathrm{Re} < 10^5$$

- Let's choose $\nu = 2 \times 10^{-6}$, $\Rightarrow \mathrm{Re} = 2R|\mathbf{U}|/\nu = 10^4$
- No longer laminar — `icoFoam` no longer suitable
- Examine the `Description` in the `.C` files in the `$FOAM_SOLVERS/incompressible` directory, *e.g.*

```
>> cd $FOAM_SOLVERS/incompressible
>> find . -name "*.C" -exec grep -H -A3 Description {} \;
./pisoFoam/pisoFoam.C:Description
./pisoFoam/pisoFoam.C-    Transient solver for incompressible flow.
./pisoFoam/pisoFoam.C-
./pisoFoam/pisoFoam.C-    Turbulence modelling is generic...
...
```

- `pisoFoam` suitable
- We will use Reynolds-averaged stress (RAS) turbulence modelling

Copying case files

- We need `poiseuilleHighRe` case for the `pisoFoam` solver
- The `poiseuille` case files are set up for `icoFoam`
- $\Rightarrow$ we should copy a `pisoFoam` case...
- then copy any `poiseuille` case files that can be reused, e.g.

Mesh `constant/polyMesh/*`

Fields `0/*`

Sampling `system/sampleDict`

Controls system/controlDict

- Then modify other files, e.g. turbulent modelling and fields
- To save time/typing, use pre-configured case in EXAMPLES directory

```
>> run
>> cp -r EXAMPLES/poiseuilleHighRe .
```

Turbulence simulation and RAS modelling

- The type of turbulence modelling is specified under simulationType in constant/turbulenceProperties from:

  laminar uses no turbulence models

  RASModel uses RAS modelling;

  LESModel uses large-eddy simulation (LES) modelling

- The choice of Reynolds-averaged stress (RAS) turbulence model is then set in constant/RASProperties, containing:

Keyword	Description
RASModel	Name of RAS turbulence model
turbulence	Switch to turn turbulence modelling on/off
printCoeffs	Switch to print model coeffs to terminal at simulation startup
<RASModel>Coeffs	Optional dictionary of coefficients for the respective RASModel

- We select the $k - \epsilon$ model

```
RASModel        kEpsilon;

turbulence      on;

printCoeffs     on;
```

Initialising turbulence fields

- The $k - \varepsilon$ model contains two new fields
  - Turbulent kinetic energy $k = \overline{\mathbf{U}' \cdot \mathbf{U}'}/2$
  - Turbulent dissipation rate $\varepsilon = C_\mu^{0.75} k^{1.5}/L$

- Initialise isotropic turbulence ${U'_x}^2 = {U'_y}^2 = {U'_z}^2 = 5\%$ of the inlet velocity

$$k = \frac{3}{2}\left(\frac{5}{100}\right)^2 = 3.75 \times 10^{-3}\ \mathrm{m}^2/\mathrm{s}^2$$

- Assume a turbulent length scale $L = 20\%$ of the tube diameter ($C_\mu = 0.09$)

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{L} \approx 9.4 \times 10^{-3}\ \mathrm{m}^2/\mathrm{s}^3$$

Wall functions

- Wall functions are specified through boundary conditions on turbulent viscosity ν_t
- In this case a standard wall function is specified by the **nutWallFunction** type on wall boundaries, see 0/nut
- epsilonWallFunction must be specified on corresponding patches in the 0/epsilon
- kqRWallFunction must be specified on corresponding patches in the turbulent fields k, q and R (k in this case)

Turbulence field example (1): k

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

internalField   uniform 0.00375;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.00375;
    }
    walls
    {
        type            kqRWallFunction;
        value           uniform 0.00375;
    }
    outlet {type zeroGradient;}
    back   {type wedge;}
    front  {type wedge;}
}
```

Turbulence field example (2): nut

```
dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0; // Overridden by the code

boundaryField
{
    inlet
    {
        type        calculated;
        value       uniform 0;
    }
    walls
    {
        type        nutWallFunction; // The only entry of importance
        value       uniform 0;
    }
    outlet {type zeroGradient;}
    back   {type wedge;}
    front  {type wedge;}
}
```

Incompressible transport models

- Need to set $\nu = 2 \times 10^{-6}$ for $\mathrm{Re} = 10^4$
- pisoFoam uses the incompressibleTransportModels library
- More options in constant/transportProperties:

```
transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 2e-06;
```

- User selects the transportModel, then necessary values/coeffs

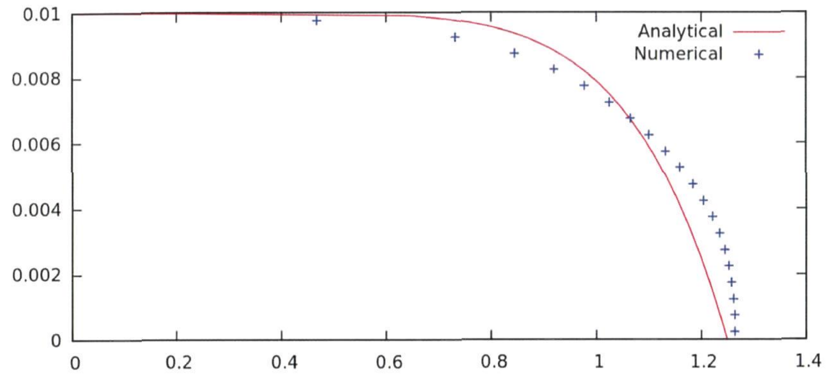
Running the pisoFoam solver

- Converges a bit slower than before: $\Rightarrow$ set endTime to 0.5 in controlDict
- Run pisoFoam

```
>> foamJob pisoFoam
```

- Run sample
- Run gnuplot on a new configuration file called plot:


```
h = 0.01
set parametric
plot [0:h] 5.0/4.0*(1.0 - t/h)**(1.0/7.0),t title "Analytical", \
     "sets/0.5/acrossFlow_U.xy" using 2:1 title "Numerical"
```

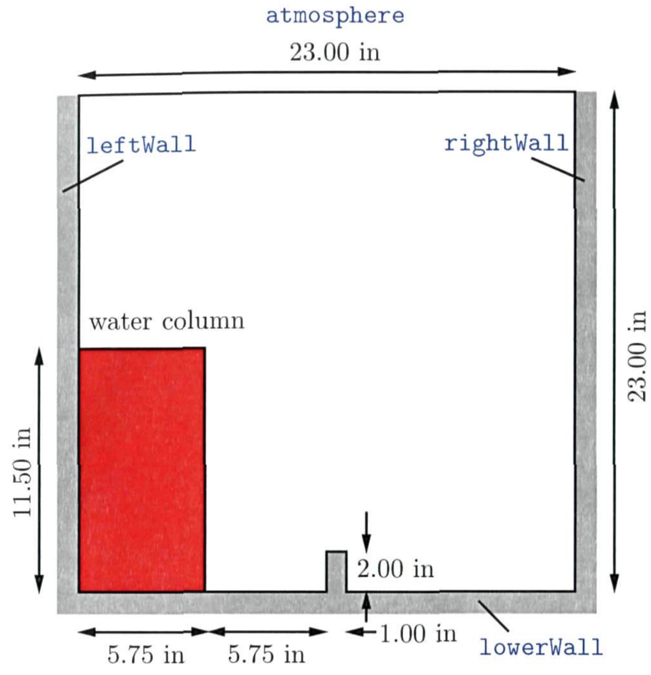


3 Dam break

3.1 Subsetting a mesh

Dam break

- Water column release
- Use the interFoam solver
- Different set up to the User Guide
- Geometry in inches (in)
- Dimensions divisible by 0.25 in
- Let's create a uniform cell size 0.25 × 0.25 in
- Let's create a single block...
- ...then cut out the 2 × 1 in obstacle
- Retain the same patches



Modifying the tutorial damBreak case

- Copy the tutorial interFoam/damBreak case locally

```
>> cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak .
```

- Create a single 2D block with 92 × 92 cells, in blockMeshDict:

```
convertToMeters 0.0254; // inches to metres conversion

vertices
(
    ( 0  0 -1)
    (23  0 -1)
    (23 23 -1)
    ( 0 23 -1)
    ( 0  0  1)
    (23  0  1)
    (23 23  1)
    ( 0 23  1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (92 92 1) simpleGrading (1 1 1)
);
```

Patches blockMesh

- Now create patches using existing patch names

```
patches
(
    wall leftWall
    (
        (0 4 7 3)
    )
    wall rightWall
    (
        (1 2 6 5)
    )
    wall lowerWall
    (
        (0 1 5 4)
    )
    patch atmosphere
    (
        (2 3 7 6)
    )
);
```

- Run blockMesh

Subsetting mesh

- subsetMesh utility: creates a subset of a mesh from a cell set
- cellSet utility: creates a set of cells based on cellSetDict
- Copy an example cellSetDict file to system directory

```
>> cp $FOAM_UTILITIES/mesh/manipulation/cellSet/cellSetDict system
```

- Edit the file accordingly:

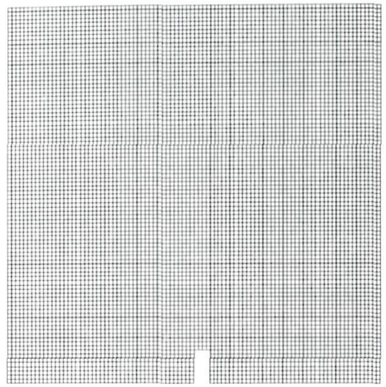
```
name c0;
action new;      // use this on 1st cellSet run
//action invert; // use this on 2nd cellSet run

topoSetSources
(
    boxToCell
    {
       box   (0.2921 0 -1) (0.3175 0.0508 1);
    }
);
```

- Run cellSet: with action new; to create a cell set of the obstacle
- Run cellSet again: with action invert; to create a set of all cells except the obstacle
- Run subsetMesh on the cell set c0, merging new faces into patch lowerWall

```
>> subsetMesh c0 -patch lowerWall -overwrite
```


Final mesh

- -overwrite option puts mesh in constant
- Without this option, time is incremented before writing to prevent the mesh being overwritten



3.2 Nonuniform initial fields

Setting nonuniform initial field

- Volume of Fluid (VoF) method solves for fraction α of fluid phase(s)
- Nonuniform initial condition for the phase fraction of phase 1 α_1 (alpha1)

$$\alpha_1 = \begin{cases} 1 & \text{for pure phase 1 (liquid)} \\ 0 & \text{for pure phase 2 (gas)} \end{cases}$$

- setFields: initialises nonuniform fields according to setFieldsDict

```
defaultFieldValues // specify default values
(
    volScalarFieldValue alpha1 0
);

regions              // specify regions of different values
(
    boxToCell        // uses same mesh set tools as cellSet
    {
        box (0 0 -1) (0.146 0.292 1);
        fieldValues
```

```
            (
                volScalarFieldValue alpha1 1
            );
        }
);
```

- Now look at the 0/alpha1 file; note the boundary conditions

Transport and interface properties

- transportProperties is split into two subditionaries phase1 and phase2
- Each subdictionary includes a transportModel for the phase
  - If Newtonian, kinematic viscosity specified under the keyword nu
  - If another model, e.g. CrossPowerLaw, viscosity parameters are specified in a further subdictionary, e.g. CrossPowerLawCoeffs
- The surface tension, a property of both phases, is specified by sigma
- The damBreak case uses properties of water and air
- Gravitational acceleration is specified as a uniformDimensionedField in the constant/g file

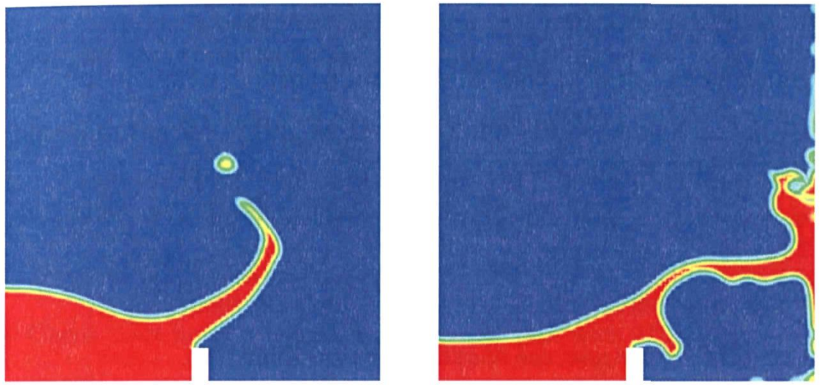
Discretisation schemes

- OpenFOAM's interface tracking solvers use OpenCFD's multidimensional universal limiter for explicit solution (MULES) method
  - maintains boundedness of alpha1...
  - ...independently of the underlying numerical scheme and mesh structure
- ⇒ choice of convection scheme not restricted to those that are strongly stable or bounded, e.g. upwind differencing
- A reliable set of convection schemes is set up in system/fvSchemes

```
divSchemes
{
    div(rho*phi,U)  Gauss limitedLinearV 1;
    div(phi,alpha)  Gauss vanLeer;
    div(phirb,alpha) Gauss interfaceCompression;
}
```

- Run the case!!

Results



- Field plot of phase fraction alpha1 at $t = 0.25$ s and at $t = 0.50$ s

Creating an animation

- The case can be animated in ParaView by clicking Play in the animation toolbars
- An animation can be saved by selecting File -> Save Animation
  - Writes a set of image frames, *e.g.* in PNG format, damBreak*png
- Can be converted into an animation, *e.g.* in MPG format with the convert utility in the ImageMagick package

```
>> convert -quality 100% damBreak*png damBreak.mpg
```

- OR directly using mencoder

```
>> mencoder "mf://*.png" -mf fps=2 -o damBreak.mpg -ovc lavc -lavcopts
vcodec=mpeg4:autoaspect
```

- The animation can be played with mplayer

```
>> mplayer -loop 0 damBreak.mpg
```

3.3 Running in parallel

Parallel running overview

- Parallel computing in OpenFOAM uses domain decomposition
- Geometry and associated fields are broken into pieces and allocated to separate processors
- `decomposePar`: performs domain decomposition using `decomposeParDict` configuration file
- Simple geometries: use `hierarchical` decomposition
- Complex geometries: use `scotch` decomposition

Domain decomposition

- Let's run `damBreak` case on 2 CPUs or 2 cores
- Clone the `damBreak` case to `damBreakPar`

```
>> mkdir damBreakPar
>> cp -r damBreak/0 damBreak/[cs]* damBreakPar
```

- Modify the `system/decomposeParDict` file to 2 domains split in the x-direction

```
numberOfSubdomains 2;    // no. of subdomains for decomposition

method hierarchical;     // method of decomposition, simple geometries
//method scotch;         // method of decomposition, complex geometries

hierarchicalCoeffs
{
    n           (2 1 1); // domain split into 2 in x direction
    order       xyz;     // directions in which decomposition is done
    delta       0.001;   // set it to 0.001
}

distributed     no;      // is the data distributed across several disks?
```

- Now execute `decomposePar` on `damBreakPar`

Parallel running

- The case should be split into `processor<n>` directories, each containing its own part of the mesh and fields

```
>> ls damBreakPar
0  constant  processor0  processor1  system
```

- The case can be run in parallel using `mpirun`; the solver must be executed with the `-parallel` option

```
>> cd damBreakPar
>> mpirun -np 2 interFoam -parallel
```

- Check there are 2 processes running, e.g. (kill with `CTRL-C`)

```
>> top
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
18767 chris     25   0  184m  17m  10m R  101  1.7   0:04.28 interFoam
18768 chris     25   0  184m  17m  10m R   99  1.7   0:04.25 interFoam
...
```

Parallel running options

- Running on a cluster, a user wishes to run from machine `aaa` on the following machines: `aaa`; `bbb`, which has 2 processors; and `ccc`
- The user should create a file, e.g. `machines`, containing:

```
aaa
bbb cpu=2
ccc
```

- The application is run with `mpirun` using the option:
  `-hostfile /path/to/machines`
- For further information

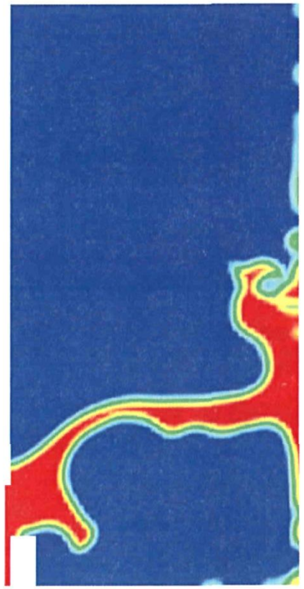
```
>> mpirun --help
```

- `foamJob` script with `-p` option runs parallel cases using `mpirun`:
  - automatically picks up no. of processors from no. of `processor<n>` directories
  - automatically uses a file named `machines` if present in `system` directory

Post-processing and reconstruction

- reconstructPar utility: reassembles decomposed fields and mesh from processor<n> directories into normal time directories
- Segments of domain can be post-processed individually by treating an individual processor<n> directory as a case in its own right, e.g. running

```
>> paraFoam -case processor1
```



4 Programming background

4.1 C++ overview

OpenFOAM Programming: language in general

- The success of any language is due to efficiency in expressing concepts
- Using verbal language, 'velocity field'. . .
  - has abstract meaning without reference to the type of the flow or specific data
  - encapsulates the idea of movement with direction and magnitude
  - relates to other physical properties
- In mathematical language, we represent velocity field by a single symbol '$\mathbf{U}$'
  - Symbols can express further concepts, *e.g.* the field of velocity magnitude by $|\mathbf{U}|$
- This is emulated in OpenFOAM
  - Our velocity field can be represented by `U`
  - 'The field of velocity magnitude' can be `mag(U)`
- The idea is taken much further. . .

Equation representation in OpenFOAM

- Top level code represents the equations being solved, *e.g.*

$$\underbrace{\frac{\partial \rho \mathbf{U}}{\partial t}}_{1} + \underbrace{\nabla \bullet \rho \mathbf{U}\mathbf{U}}_{2} - \underbrace{\nabla \bullet \rho \mathbf{R}}_{3} = - \underbrace{\nabla p}_{4}$$

1. Local rate of change of $\rho\mathbf{U}$
2. Convective rate of change of $\rho\mathbf{U}$
3. Viscous dissipation (laminar + turbulent)

4. Pressure gradient

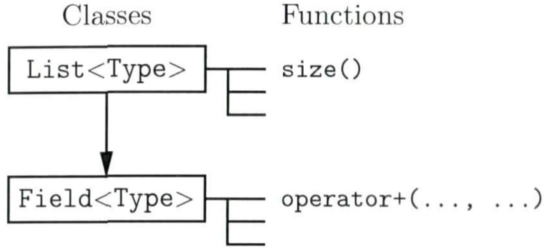
```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  + turbulence->divRhoR(U)
  ==
  - fvc::grad(p)
);
```

- Uses polymorphism: objects of different classes respond differently to functions of the same name
  - `solve` function behaves differently depending on the class (type) it operates on
  - +, -, == operators have been overloaded
  - `divRhoR` function is overridden by each turbulence model, so is interpreted differently depending on the model selected at runtime
  - Improves code readability

Classes and objects

- Velocity is a vector field. In object-oriented programming...
  - there could be a `vectorField` class
  - the velocity field `U` would be an instance or object of that class
- Temperature, pressure, density are scalar fields
  - there could be a `scalarField` class
  - `p`, `T`, `rho` would be objects of that class
- C++ provides template classes, *e.g.* `Field<Type>`
  - The `<Type>` can be `scalar`, `vector`, `tensor`
  - General features of the template class are passed on to any class created from it
  - Reduces code duplication

Class hierarchy



- C++ allows a hierarchy of classes
- Generic concepts can be defined in a base class, *e.g.* `List<Type>`
- A derived class, e.g. `Field<Type>` can be formed from base classes
- The derived class inherits attributes/behaviour of the base classes
- *e.g.* we could access the size of `vectorField U` with `U.size()`

How much C++ do we need to know?

- Users do not need a deep knowledge of C++ programming to work with utilities, solvers and model libraries...
- ...because these top level codes are largely procedural since they represent solution algorithms
- Users need to:
  - understand C++ and OpenFOAM syntax and mechanisms;
  - locate classes and their functionality;
  - make modifications and compile them into executables/libraries.

Class files

For a class `vector`

- Class definition in `vector.C`:

  - a set of instructions such as object construction, data storage and functions

- Compilation of `vector.C`:
  - either with an application file `vectorTest.C` — containing the `main` function — into an application executable `vectorTest`
  - or into a shared object library `OpenFOAM.so` that is linked to `vectorTest`

- Class declaration in `vector.H`:
  - a list of defined functions *etc.*, not the functions themselves
  - every compiled (`.C`) file needs this list for the classes it uses
  - the `.H` file must be included before any code using the class (including the class declaration `.C` code itself)

```
#include "vector.H";
```

4.2 Code compilation

Compilation example: `pisoFoam`

1. Make a local source code directory in the user's account and go into that directory

```
>> mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/incompressible
>> cd !$
```

   - Note: `!$` or `!:$` — word designator, meaning "last argument on previous line", i.e. the directory that has been created

2. Copy the `pisoFoam` source code from the installation and go into the directory

```
>> cp -r $FOAM_SOLVERS/incompressible/pisoFoam .
>> cd pisoFoam
```

3. Change `$FOAM_APPBIN` to `$FOAM_USER_APPBIN` in the `Make/files` file

```
pisoFoam.C

EXE = $(FOAM_USER_APPBIN)/pisoFoam
```

4. Compile with `wmake`

```
>> wmake
```

- Let's explain how the compilation works...

Compilation with `wmake`

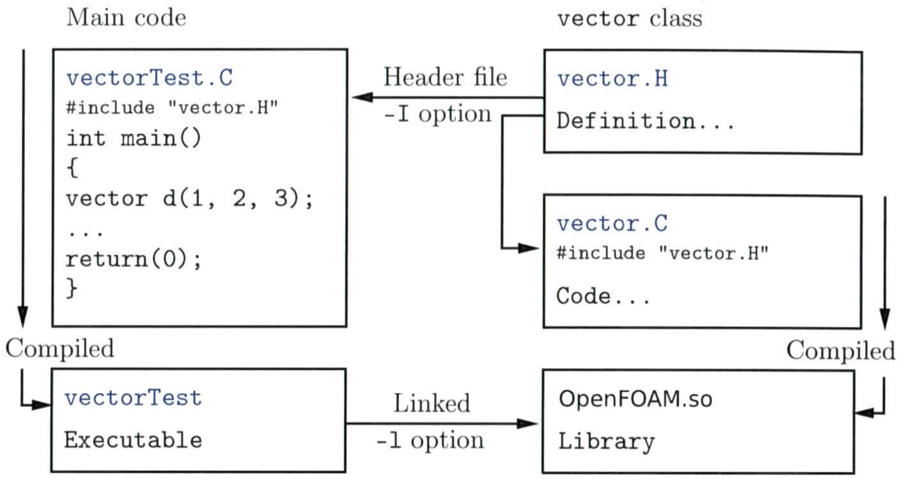
- OpenFOAM uses its own compilation tool `wmake`
- ... designed for a single package containing 100s of applications/libraries
- Application/library code requires a `Make` directory, containing 2 files
  - `files`: List of compiled `.C` file(s) and executable name
  - `options`: Compilation options
- `wmake` builds a dependency list with `.dep` file extension, *e.g.* `pisoFoam.dep`
  - Can be used to locate included files on the system
- For a library, `wmake` creates an `lnInclude` directory with links to all source files
  - A good place to search for files, *e.g.* `$FOAM_SRC/OpenFOAM/lnInclude`

`wmake`: the `files` file

- The `files` file contains a list of `.C` source files that must be compiled
  - It does not need the `.C` files already compiled into linked libraries
  - Often, the 'list' is just the single main `.C` file
- For applications, "`EXE =`" specifies the path/name of the compiled executable
- For libraries, "`LIB =`" specifies the path/name of the compiled library
- Standard release applications are stored in `$FOAM_APPBIN` (libraries in `$FOAM_LIBBIN`)
- User applications should go in `$FOAM_USER_APPBIN`
- `$FOAM_USER_APPBIN` takes precedence on the system `$PATH` so a user application will 'override' a release application of the same name
  - *e.g.* test for `pisoFoam` by typing:

```
>> which pisoFoam
/home/ubuntu/OpenFOAM/.../bin/linuxGccDPOpt/pisoFoam
```

Compiling and linking



wmake: including headers

- The compiler searches for included files in the following directories:
  1. $WM_PROJECT_DIR/src/OpenFOAM/lnInclude
  2. pisoFoam/lnInclude: a local lnInclude directory
  3. pisoFoam: the local directory
  4. platform dependent paths set in $WM_DIR/rules/$WM_ARCH, *e.g.* /usr/X11/include
  5. Other directories specified in the Make/options file with the -I option
- The Make/options file contains the full directory paths, *e.g.*

```
EXE_INC = \
    -I$(LIB_SRC)/turbulenceModels/incompressible/turbulenceModel \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel \
    -I$(LIB_SRC)/finiteVolume/lnInclude
```

- Uses a common UNIX scripting syntax of a backslash (\) to continue across lines

wmake: linking to libraries

- The compiler links to shared object libraries in the following directory paths:
  1. $FOAM_LIBBIN
  2. platform dependent paths set in $WM_DIR/rules/$WM_ARCH/ directory, *e.g.*/usr/X11/lib
  3. Other directories specified in the Make/options file with the -L option, typically

```
EXE_LIBS = -L$(FOAM_USER_LIBBIN)
```

- The actual library **files** to be linked are:
  1. the libOpenFOAM.so library from the $FOAM_LIBBIN directory
  2. platform dependent libraries, *e.g.*libm.so from /usr/X11/lib
  3. other libraries specified in the Make/options file with the -l option, removing the lib prefix and .so extension from the library file name

```
EXE_LIBS = \
    -lincompressibleTurbulenceModel \
    -lincompressibleRASModels \
    -lincompressibleLESModels \
    -lincompressibleTransportModels \
    -lfiniteVolume \
    -lmeshTools
```

Running wmake

- To compile an application, change to the application directory and type

```
>> wmake
```

- Alternatively, include the application directory path as an argument, *e.g.*

```
>> wmake $WM_PROJECT_USER_DIR/applications/solvers/incompressible/pisoFoam
```

- To compile a library, go to the library directory and type

```
>> wmake libso
```

wclean: cleaning up after wmake

- Sometimes after making code changes, or before packing a solver to send elsewhere, .dep files need removing
- To clean an application source directory, go to the directory and type

```
>> wclean
```

- To clean a library source directory, go to the directory and type

```
>> wclean libso
```

- The rmdepall script also removes .dep files recursively down a directory tree

4.3 Utility walk through

Utility walk through

- Let us look at a post-processing utility that creates total pressure from static p and U
- A search in the release finds the utility ptot
- Let's go to the source code directory

```
>> cd $FOAM_UTILITIES/postProcessing/miscellaneous/ptot
```

- It contains a file called ptot.C...
- ...and a Make directory

The fvCFD.H file

- Following the comment block at the top of ptot.C:

```
#include "fvCFD.H"
```

- fvCFD.H is a file containing a selection of included class header files that are generally relevant to finite volume CFD, e.g.:
  - Time.H: the Time database

  - fvMesh.H: the finite volume mesh class
  - fvc.H, fvMatrices.H, fvm.H, *etc.*: finite volume equation discretisation
  - argList.H: handles terminal argument list
  - ...and more
- Some of these may not be needed, but it does not matter much; if the application is CFD-related, just include fvCFD.H
- fvCFD.H is in $FOAM_SRC/finiteVolume/lnInclude and classes compiled in the finiteVolume library
- Make/options:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -lgenericPatchFields
```

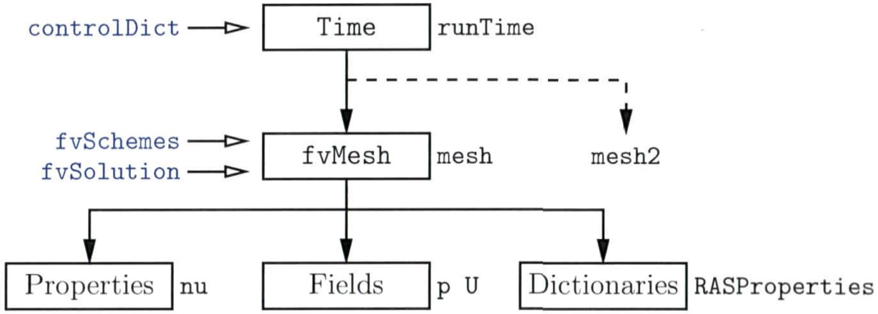
Time and command line options

- The first part of the code is concerned with Time and command line options

```
int main(int argc, char *argv[])
{
    timeSelector::addOptions();

#   include "setRootCase.H"
#   include "createTime.H"

    instantList timeDirs = timeSelector::select0(runTime, args);
```

- timeSelector::addOptions(): reads command line options that apply utility to data from selected time directories only
- setRootCase.H: sets the root path and case directories according to the arguments
- createTime.H: instantiate runTime of the type Time — a class that holds information relating to time which acts as a database for the simulation
- timeSelector::select0() function returns a list of time directories to calculate ptot for

Database for OpenFOAM cases



- OpenFOAM has a hierarchical database for case data
- `Time`: top-level `objectRegistry`; controls time and data reading/writing
  - Object typically named `runTime`
  - Reads `controlDict`
- `fvMesh`: next level of `objectRegistry`
  - Object typically named `mesh`, but can be more than one
  - Reads `fvSchemes` and `fvSolution`
- Fields, properties, dictionaries registered with a particular `fvMesh`

Creating a mesh and looping over times

```
#   include "createMesh.H"

    forAll(timeDirs, timeI)
    {
        runTime.setTime(timeDirs[timeI], timeI);
```

- `createMesh.H`: instantiates `mesh` of type `fvMesh` — finite volume mesh

```
fvMesh mesh
(
    IOobject
    (
        fvMesh::defaultRegion,
        runTime.timeName(),
        runTime,
        IOobject::MUST_READ
    )
);
```

- `runTime` set to particular time from the `timeDirs` list

`Info` statements

- The next line is an `Info<<` statement

```
Info<< "Time = " << runTime.timeName() << endl;
```

- Messages (terminal) written by `Info messageStream`, syntax:

```
Info<< "message1" << "message2" << FoamDataType << endl;
```

- Useful diagnosatic tool:
  - used to monitor a field, cell value, energies, *etc.* in a simulation
  - statements can be inserted to find the line where a code 'blows up'
- Useful post-processing tool: this will be demonstrated later

Reading and writing case data

- Most case data is read/written using the `IOobject` class
- An `IOobject` is typically constructed from

  Name used for the name of the file

  Instance used for the name of the directory

  Object Registry that the object is registered with

  Read Option controls reading from file; defaults to `NO_READ`

  Write Option controls writing to file; defaults to `NO_WRITE`

- For the pressure field, the `IOobject` is

```
IOobject pheader
(
    "p",                    // Name of pressure field file
    runTime.timeName(),     // Time directory it is read from
    mesh,                   // The mesh object registry
    IOobject::MUST_READ     // Read p in from file
    IOobject::NO_WRITE      // Do not "automatically" write it out
);
```

- Further code creates an `IOobject` for `U`

Reading a field

```
if (pheader.headerOk() && Uheader.headerOk())
{
    mesh.readUpdate();

    Info<< "    Reading p" << endl;
    volScalarField p
    (
        pheader, // IOobject
        mesh     // fvMesh
    );
```

- `headerOk()` function checks if `p` and `U` files exist
- `mesh.readUpdate()` re-reads the mesh if modified
- Constructs a `volScalarField` for `p` by reading from file, using

  `IOobject` instructs where to read/write the file

  `fvMesh` the mesh that relates to the field, to ensure consistency
- Further code creates a `volVectorField` for `U`

Access functions

- Objects like `p` contain a lot of stored data
- Data can be accessed by functions using syntax `object.functionName()`
- For example, for `p`, we could call the following

  `p.mesh()` returns the mesh relating to the pressure field

  `p.name()` returns the name of the pressure field

  `p.dimensions()` returns the dimensions

  `p.internalField()` returns the internal field (cell values) only

  `p.oldTime()` returns the pressure field from the previous time step

  `p.size()` returns size of the pressure field (no of cells)

Finding functions that exist

- Users want to know if functions exist that do what they need
- Source code documented using Doxygen (`http://www.doxygen.org`)
- Can be accessed online:

  `http://www.openfoam.com/docs/cpp`
- Can be built from sources using `doxygen` (may require **root** permission; ensure OpenFOAM env variables are set)

```
>> cd $WM_PROJECT_DIR/doc/Doxygen
>> doxygen
```

- From the top level, the `Classes` menu is particularly useful
- `Alphabetical List` and `Class Members` are useful sub-menus
- In `Class` description, `List of all members` is particularly useful
- Inheritance and collaboration diagrams provide description of class hierarchy

Creating a new field

```
if (p.dimensions() == dimensionSet(0, 2, -2, 0, 0))
{
    volScalarField ptot
    (
        IOobject
        (
            "ptot",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ
        ),
        p + 0.5*magSqr(U)
    );
    ptot.write();
}
```

- Code compares dimensions of pressure to those of kinematic pressure: L^2/T^2
- Code evaluates differently depending on dimensions
  - for kinematic pressure, it evaluates $p + |\mathbf{U}|^2/2$

  - for dynamic pressure, it evaluates $p + \rho|\mathbf{U}|^2/2$

- For kinematic pressure, constructs a `volScalarField` named `ptot` from an `IOobject` and `p + 0.5*magSqr(U)`
- `write()` function writes the `ptot` field to the current time dir.

Ending the utility

```
        else
        {
            Info<< "    No p or U" << endl;
        }

        Info<< endl;
    }
    return(0);
}
```

- Print a terminal `Info` message if `p` and `U` field does not exist in the current time directory
- Ends with `return(0);`

Summary of key classes

- `#include createTime.H`: creates `Time` database named `runTime`
- `#include createMesh.H`: creates `fvMesh` named `mesh`
- `IOobject` class controls data read/write and storage on database
- `volScalarField`, `volVectorField` classes create field objects, e.g. `p`, `U`
- `dimensionSet` class defines dimensional units

5 Solver development

5.1 Modifying a solver

Solver source code

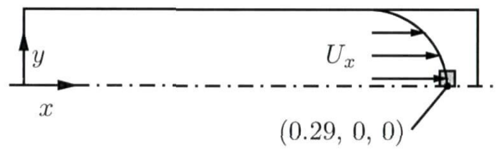
- Let us look at `icoFoam`
- Go to the user's equivalent solver directory

```
>> cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
>> cp -r $FOAM_SOLVERS/incompressible/icoFoam .
>> cd icoFoam
```

- It contains the C++ files `icoFoam.C` and `createFields.H`

Info statements revisited

- Demonstrate `Info<<` statements as a post-processing tool
- Aim: in the `parallelPlateSymm` case, monitor U_x in a cell adjacent to centreline at $x = 0.29$



- Before making changes, edit `Make/files` to write compiled solver to user's account

```
icoFoam.C

EXE = $(FOAM_USER_APPBIN)/icoFoam
```

- Compile the solver with `wmake`

Writing new OpenFOAM code

- In `icoFoam.C`, add code at the end of the time step loop, (line 101)
- First we need to add our location; use the OpenFOAM `vector` class

```
101    vector location(0.29, 0, 0);
```

- Find the nearest cell to that location; use the `label` class
- Search the code for an appropriate function

```
>> find $FOAM_SRC -type f -name "*.C" -exec grep -iE "find.*cell" {} \;
   ...
   cellI = owner_.mesh().findNearestCell(position);
```

- `findNearestCell(...)` can be applied to a `mesh`, taking `location` as an argument; add to `icoFoam.C`:

```
102    label cellI = mesh.findNearestCell(location);
```

- Note: When searching for a definition of a function, precede the function name by `::`, *e.g.*

```
>> find $FOAM_SRC -type f -exec grep -l "::findNearestCell" {} \;
```

Accessing field components

- Cell values are obtained from fields using the syntax for an element of an array `U[...]`
- The `vector` class has `x()`, `y()` and `z()` functions to access scalar components
- Add the `Info` statement

```
103    Info<< "centreline: t = " << runTime.timeName()
104        << " Ux = " << U[cellI].x() << endl;
```

- `endl` = end line
- `nl` = insert new line

Test the Info statement

- Recompile the solver
- Run the modified `icoFoam` on the `parallelPlateSymm` case redirecting output to a `log` file
- Output produces lines like

```
centreline: t = 0.0002 Ux = 1.00833
```


- Strip out values for plotting with `grep` and `cut`

```
>> grep centreline log | cut -d" " -f4,7
0.0002 1.00833
0.0004 1.01321
0.0006 1.01875
...
```

- This command can be redirected to another file and the data plotted

Basic classes in OpenFOAM

- There are a number of basic classes in OpenFOAM
- ... derived from more fundamental C++ classes
- OpenFOAM's classes have a bit more functionality, so use them

C++ class	OpenFOAM class and additional features	
`int/long`	`label`	Automatic switching
`bool`	`Switch`	Accepts `true/false`, `on/off`, `yes/no`
`string`	`word`	Strings with no whitespace, '/', *etc.*
`float/double`	`scalar`	Depends on `$WM_PRECISION_OPTION`
—	`vector`	3D vector with algebra
—	`tensor`	3×3 tensor with algebra

- Note: C++ has a `vector` class, similar to an array

5.2 Dictionary I/O

The `createFields.H` file

- The `icoFoam.C` file begins with `#include` files discussed previousy until...

```
43    #    include "createFields.H"
```

- Creation of solver fields usually contained within a `createFields.H` file
- Creates an `IOdictionary` from an `IOobject` that reads in the case file `constant/transportProperties`:

```
IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties", // name of the file
        runTime.constant(),    // the case "constant" directory
        mesh,                  // the mesh object registry
        IOobject::MUST_READ,   // read in from file
        IOobject::NO_WRITE     // do not write out to file
    )
);
```

- Creates kinematic viscosity `nu` of type `dimensionedScalar` by a `lookup` of keyword `nu` from `transportProperties`

```
dimensionedScalar nu
(
    transportProperties.lookup("nu")
);
```

Dictionary lookup

- The user can therefore read in keyword entries by
  1. Creating an `IOdictionary`
  2. Looking up entries with the `.lookup("keyword")` function
- `lookup("keyword")` returns an `Istream`
- Most objects can be constructed from `Istream`, e.g.

```
class object(dict.lookup("keyword")); // Construct from Istream
```

- ...because type of object can be established from syntax, e.g.
  - `string` `("hello world")`
  - `word` `(hello)`
  - `Switch` `(on/yes/true)`
  - `dimensionedScalar`
- `scalar` (1) and `label` (1) are **exceptions**
- ⇒ special `readLabel`/`readScalar` functions create a `scalar`/`label`

```
scalar a(readScalar(dictionary.lookup("a")));
label  i(readLabel (dictionary.lookup("i")));
```


5.3 Fields and field algebra

Field construction

- Pressure created by

```
volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE  // write out to file automatically
    ),
    mesh
);
```

- `AUTO_WRITE` applied to fields we write out to time directories according to `controlDict` settings
- All fields written out by `write()` function called on the database in `icoFoam.C`

```
runTime.write();
```

- 3 common ways to construct a field, with different file read/write options:

Read	Write	Constructor
✓	✓/✗	`volScalarField(IOobject, fvMesh)`
✗	✓	`volScalarField(IOobject, volScalarField)`
✗	✗	`volScalarField(volScalarField)`

More about fields

- Velocity flux instantiated as `surfaceScalarField`

```
#include "createPhi.H"
```

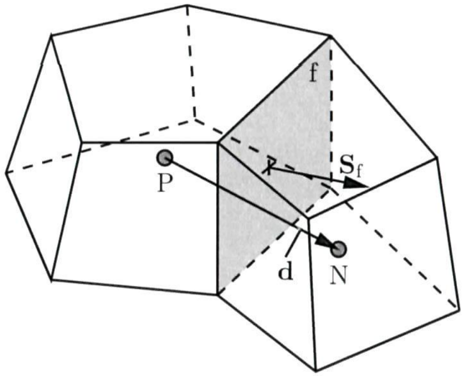
`$FOAM_SRC/finiteVolume/lnInclude/createPhi.H`:

```
surfaceScalarField phi
(
    IOobject
    (
        "phi",
        runTime.timeName(),
        mesh,
```

```
            IOobject::READ_IF_PRESENT,    // Read if file exists
            IOobject::AUTO_WRITE
        ),
        linearInterpolate(U) & mesh.Sf() // ...otherwise evaluate
    );
```

- What does "`linearInterpolate(U) & mesh.Sf()`" do?
  - What is a `volScalarField`, `surfaceScalarField`, *etc.*?
  - What is the "`&`" symbol?
  - What about "`mesh.Sf()`"?

Meshes



Description	Symbol	Function
Cell volumes	V	`V()`
Old time step cell volumes	V^o	`V0()`
Old-old time cell volumes	V^{oo}	`V00()`
Face area vectors	$\mathbf{S}_\mathrm{f}$	`Sf()`
Face area magnitudes	$\lvert\mathbf{S}_\mathrm{f}\rvert$	`magSf()`
Cell centres	$\mathbf{C}$	`C()`
Face centres	$\mathbf{C}_\mathrm{f}$	`Cf()`
Face motion fluxes	ϕ	`phi()`

- There is a hierarchy of mesh classes including `geometricMesh`, `polyMesh`
- `fvMesh`: includes extra functionality for finite volume discretisation
- In particular, it stores data relating to access functions above

`volScalarField`, `volVectorField`, *etc.*

- `volScalarField` is not a class; it is a `typedef` (alias)
- `typedef`: an alias for a class to make the code more easily readable
- Used particularly with template classes, *e.g.* for a "scalar field" `Field<scalar>` reads as "field scalar"
- ⇒ `$FOAM_SRC/OpenFOAM/lnInclude/scalarField.H`

```
typedef Field<scalar> scalarField;
```

- To find a `typedef`: use multiple `grep` commands:

```
>> find $FOAM_SRC -type l | xargs grep -l typedef | xargs grep -l
"volScalarField;"
$WM_PROJECT_DIR/src/finiteVolume/lnInclude/volFieldsFwd.H
```

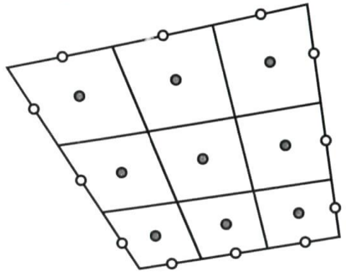
  - Note the terminating semicolon (;)
- The actual class is `GeometricField`
  `$FOAM_SRC/finiteVolume/lnInclude/volFieldsFwd.H`:

```
typedef GeometricField<scalar, fvPatchField, volMesh> volScalarField;
```

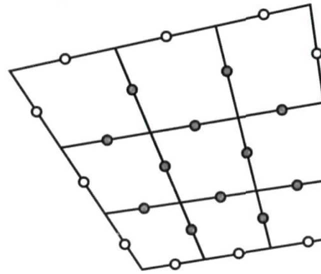
The `GeometricField` class

- `GeometricField<Type, PatchField, GeoMesh>` is templated on 3 arguments
- `GeoMesh` and `PatchField` arguments specify where values are defined
- `Type` specifies what the values are, *e.g.* `scalar`, `vector`, *etc.*

`volMesh`/`fvPatchField`
e.g. `p` and `U`



`surfaceMesh`/`fvsPatchField`
e.g. `phi`



Tensor fields

- Fields can be of the following <Type>

Rank	Name	<Type>	Example	nCmpts
0	Scalar	`scalar`	p	1
1	Vector	`vector`	$\mathbf{U}$	3
2	Tensor (general)	`tensor`	$\nabla\mathbf{U}$	9
2	Symmetric tensor	`symmetricTensor`	$\nabla\mathbf{U} + \nabla\mathbf{U}^T$	6
2	Spherical tensor	`sphericalTensor`	$p\mathbf{I}$	1

- Special tensors are constructed by default, *e.g.* Identity tensor $\mathbf{I}$: `$FOAM_SRC/OpenFOAM/lnInclude/sphericalTensor.H`:

```
static const sphericalTensor I(1);
```

Field algebra

- Algebra can be performed on fields

Operator	Ranks	Expression	OpenFOAM
Inner product	≥ 1	$\mathbf{a} \bullet \mathbf{b}$	`a & b`
Double inner product	2	$\mathbf{a} : \mathbf{b}$	`a && b`
Cross product	1	$\mathbf{a} \times \mathbf{b}$	`a ^ b`
Outer product		$\mathbf{ab}\dagger$	`a * b`
Square		$\mathbf{a}^2 \equiv \mathbf{aa}$	`sqr(a)`
Magnitude squared		$\|\mathbf{a}\|^2 \equiv \mathbf{a} \overset{R}{\bullet} \mathbf{a}$	`magSqr(a)`
Magnitude		$\|\mathbf{a}\| \equiv \sqrt{\mathbf{a} \overset{R}{\bullet} \mathbf{a}}$	`mag(a)`
Transpose	2	$\mathbf{a}^T$	`a.T()`
Trace	2	$\mathrm{tr}\,\mathbf{a} = \mathbf{I} : \mathbf{a}$	`tr(a)`
Symmetric	2	$\mathrm{symm}\,\mathbf{a} = (\mathbf{a} + \mathbf{a}^T)/2$	`symm(a)`
Skew	2	$\mathrm{skew}\,\mathbf{a} = (\mathbf{a} - \mathbf{a}^T)/2$	`skew(a)`
Deviatoric	2	$\mathrm{dev}\,\mathbf{a} = \mathbf{a} - (\mathrm{tr}\,\mathbf{a})\mathbf{I}/3$	`dev(a)`
Deviatoric (II)	2	$\mathrm{dev}_{II}\,\mathbf{a} = \mathbf{a} - 2(\mathrm{tr}\,\mathbf{a})\mathbf{I}/3$	`dev2(a)`

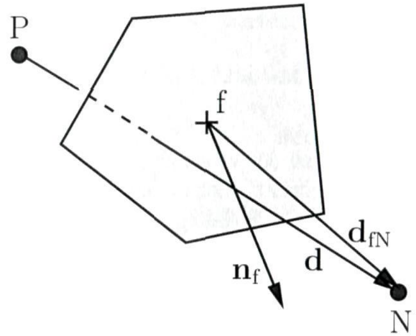
$\dagger\mathbf{a} \otimes \mathbf{b}$

- There are more operators, like transcendental scalar functions, sin, exp *etc.*

Field interpolation

- Interpolation: transforms a `vol<Type>Field` to a `surface<Type>Field`
- `fvc::interpolate(Q)`: generic interpolation function
  - scheme selected in `fvSchemes` case file
- `linearInterpolate`: hard coded linear interpolation, *e.g.* for $\mathbf{Q}$

$$\mathbf{Q}_f = w_f \mathbf{Q}_P + (1 - w_f)\mathbf{Q}_N, \qquad w_f = \frac{|\mathbf{n}_f \bullet \mathbf{d}_{fN}|}{|\mathbf{n}_f \bullet \mathbf{d}|}$$



The flux `phi`

- Let's return to the expression for `phi`

$$\underbrace{\texttt{linearInterpolate(}\overbrace{\texttt{ U }}^{\text{volVectorField}}\texttt{)}}_{\text{surfaceVectorField}} \quad \underbrace{\texttt{\&}}_{\bullet} \quad \underbrace{\overbrace{\texttt{mesh}}^{\text{fvMesh}}\texttt{.Sf()}}_{\text{surfaceVectorField}}$$

- Returns the `surfaceScalarField phi` from inner product of two `surfaceVectorFields`
- `phi`: volumetric flux through the cell faces
- OpenFOAM will not permit algebra between a `vol<Type>Field` and a `surface<Type>Field`

5.4 Implementing equations

Back to the solver...

- Next...

```
while (runTime.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

#   include "readPISOControls.H"
#   include "CourantNo.H"

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

    solve(UEqn == -fvc::grad(p));
```

- Important... creating and solving equations
- We need to understand what `fvVectorMatrix`, `fvm::`, `solve`, *etc.* mean

Discretisation

- Discretisation ≡ approximation of a continuous problem into discrete quantities

Continuous	Discrete	OpenFOAM class
Time	Time steps (intervals)	`Time`
Space (geometry)	Mesh of cells	`fvMesh`
Fields	Cell values	`vol<Type>Field`
Differential eqns.	Algebraic eqns.	`fv<Type>Matrix`

- `fv<Type>Matrix` describes an algabraic equation for a `vol<Type>Field`, *e.g.* $\mathbf{Q}$, storing:
  - $[M]$ = matrix coefficients
  - $\mathbf{B}$ = source — also a `vol<Type>Field`

$$\begin{bmatrix} M_{11} & M_{12} & \dots & M_{1N} \\ M_{21} & M_{22} & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \vdots \\ \mathbf{Q}_N \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_N \end{bmatrix}$$

$$[M] \quad \mathbf{Q} = \mathbf{B}$$

Terms in equations/expressions

- Eqns contain derivatives such as $\nabla\bullet$, ∇, ∇^2, $\nabla\times$
- OpenFOAM has functions for derivatives, *e.g.* `div`, `grad`, `laplacian`, `curl`
- To calculate derivatives with current values, prefix with `fvc::`
  - *e.g.*`fvc::grad(p)` calculates the pressure gradient ∇p
  - `fvc::` returns a field
- To discretise a term into matrix equation you wish to solve, prefix with `fvm::`
  - *e.g.* to solve $\nabla\bullet\Gamma\nabla p = 0$, use `fvm::laplacian(Gamma, p)`
  - `fvm::` returns an `fvMatrix`

Solution method and equations

- OpenFOAM uses the finite volume method for discretisation
- Co-located framework: solution fields defined at cell centres
- Segregated, decoupled: solves scalar matrix equations in an iterative sequence
- ⇒ There are only 4 terms that can form matrix coefficients, *i.e.* can "be" `fvm::`

Description	Expression	Function
Time derivative	$\partial \rho \mathbf{Q} / \partial t$	`fvm::ddt(rho, Q)`
Convection	$\nabla \bullet (\rho \mathbf{U} \mathbf{Q})$	`fvm::div(phi, Q)`
Laplacian	$\nabla \bullet \Gamma \nabla \mathbf{Q}$	`fvm::laplacian(Gamma, Q)`
Source	$\rho \mathbf{Q}$	`fvm::Sp(rho, Q)`

- Equivalent functions exist for $\partial \mathbf{Q} / \partial t$ and $\nabla^2 \mathbf{Q}$ (without ρ, Γ)
- Convective derivative: `fvm::div` function with `surfaceScalarField` flux (`phi`) as the 1st argument

Common source terms/derivatives

- Equation (explicit) source terms can be calculated using the following functions

Description	Expression	Function
Divergence	$\nabla \bullet \mathbf{Q}$	`fvc::div(Q)`
Gradient	$\nabla \mathbf{Q}$	`fvc::grad(Q)`
Curl	$\nabla \times \mathbf{Q}$	`fvc::curl(Q)`
Source	$\mathbf{Q}$	`Q`

Back to the solver...

- Solver implements the momentum equation

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \bullet (\mathbf{U}\mathbf{U}) - \nabla \bullet \nu \nabla \mathbf{U} = -\nabla p$$

- All terms in $\mathbf{U}$ can be treated implicitly (`fvm::`)
- `fvVectorMatrix` created for all terms except ∇p (we will find out why later)

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

- The "right hand side" (`fvc::grad(p)`) introduced with the `==` operator

- `solve` function solves the `fvVectorMatrix` equation

```
solve(UEqn == -fvc::grad(p));
```

- Next... the PISO loop

5.5 The PISO algorithm

Mass conservation in incompressible flows

- Mass conservation equation

$$\frac{\partial \rho}{\partial t} + \nabla \bullet (\rho \mathbf{U}) = 0$$

- Incompressible $\Rightarrow \rho = \text{const}$

$$\nabla \bullet \mathbf{U} = 0$$

- 3 components of velocity U_x, U_y, U_z; only 1 equation
- A constraint, not a solvable equation
- Incompressible flow is often dominated by this constraint
- Q: How can we solve the system ensuring this constraint is satisfied?
- A: Use the pressure-implicit split-operator (PISO) algorithm

The "trick" in PISO

- Manipulation of [$\mathbf{U}$ Eqn]

$$\begin{bmatrix} + & & \circ & & \circ \\ & + & \circ & \circ & \\ \circ & \circ & + & & \\ & \circ & & + & \circ \\ \circ & & & \circ & + \end{bmatrix} \begin{bmatrix} \mathbf{U} \end{bmatrix} = \begin{bmatrix} \mathbf{B} \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} + & & & & \\ & + & & & \\ & & + & & \\ & & & + & \\ & & & & + \end{bmatrix}}_{A} \begin{bmatrix} \mathbf{U} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{B} \end{bmatrix} - \begin{bmatrix} & & \circ & & \circ \\ & & \circ & \circ & \\ \circ & \circ & & & \\ & \circ & & & \circ \\ \circ & & & \circ & \end{bmatrix} \begin{bmatrix} \mathbf{U} \end{bmatrix}}_{\mathbf{H}(\mathbf{U})}$$

- A and $\mathbf{H}$ are evaluated by functions `UEqn.A()` and `UEqn.H()`

- A contains 1 value per cell $\Rightarrow$ a `volScalarField`
- $\mathbf{H}$ is calculated using latest values of $\mathbf{U} \Rightarrow$ a `volVectorField`

- Explicit momentum equation

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla \cdot \nu \nabla \mathbf{U} = -\nabla p$$
$$[\mathbf{U}\,\mathrm{Eqn}] = -\nabla p$$
$$A\mathbf{U} = -\nabla p + \mathbf{H}$$

Equations in PISO

- From the expression for momentum

$$A\mathbf{U} = \mathbf{H} - \nabla p$$

- ...a momentum corrector equation can be written

$$\mathbf{U} = \frac{\mathbf{H}}{A} - \frac{1}{A}\nabla p$$

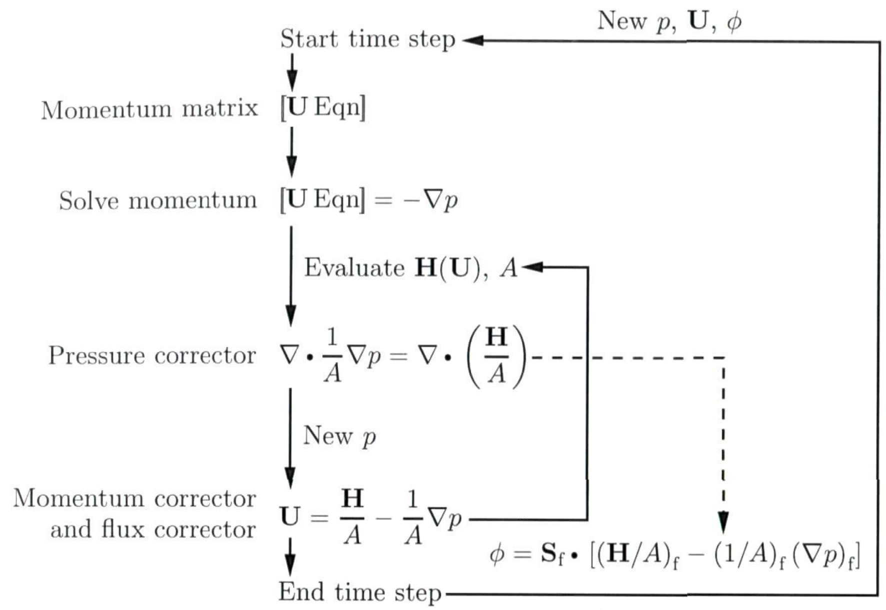
- Applying mass continuity ($\nabla \cdot \mathbf{U} = 0$), a pressure corrector equation is derived

$$\nabla \cdot \frac{1}{A}\nabla p = \nabla \cdot \left(\frac{\mathbf{H}}{A}\right)$$

- A flux corrector equation can be written

$$\phi = \mathbf{S}_\mathrm{f} \cdot \mathbf{U}_\mathrm{f} = \mathbf{S}_\mathrm{f} \cdot \left[(\mathbf{H}/A)_\mathrm{f} - (1/A)_\mathrm{f}(\nabla p)_\mathrm{f}\right]$$

PISO algorithm



PISO algorithm and OpenFOAM code

Start time step

$[\mathbf{U}\,\mathrm{Eqn}]$

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

$[\mathbf{U}\,\mathrm{Eqn}] = -\nabla p$

```
solve(UEqn == -fvc::grad(p));
```

Evaluate $\mathbf{H}(\mathbf{U})$, A

```
volScalarField rUA = 1.0/UEqn.A();
U = rUA*UEqn.H();
```

$\nabla \cdot \frac{1}{A}\nabla p = \nabla \cdot \left(\frac{\mathbf{H}}{A}\right)$

```
phi = (fvc::interpolate(U) & mesh.Sf())

fvScalarMatrix pEqn
(
    fvm::laplacian(rUA, p) == fvc::div(phi)
);

pEqn.solve();
```

New p

$\mathbf{U} = \frac{\mathbf{H}}{A} - \frac{1}{A}\nabla p$

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

$\phi = \mathbf{S}_\mathrm{f} \cdot [(\mathbf{H}/A)_\mathrm{f} - (1/A)_\mathrm{f}(\nabla p)_\mathrm{f}]$

```
phi -= pEqn.flux();
```

End time step

Final comments on the solver

- U field temporarily stores $\mathbf{H}/A$, rather than creating a new field
- Similarly, phi field temporarily stores the flux of $\mathbf{H}/A$
- Recovering U with the momentum corrector is simple (`-= rUA*fvc::grad(p)`)
- A `flux()` function returns the flux field from the matrix
- Loop over the pressure, momentum and flux correctors

```
for (int corr=0; corr<nCorr; corr++)
```

- Correct fluxes to conserve globally in badly-posed cases

```
adjustPhi(phi, U, p);
```

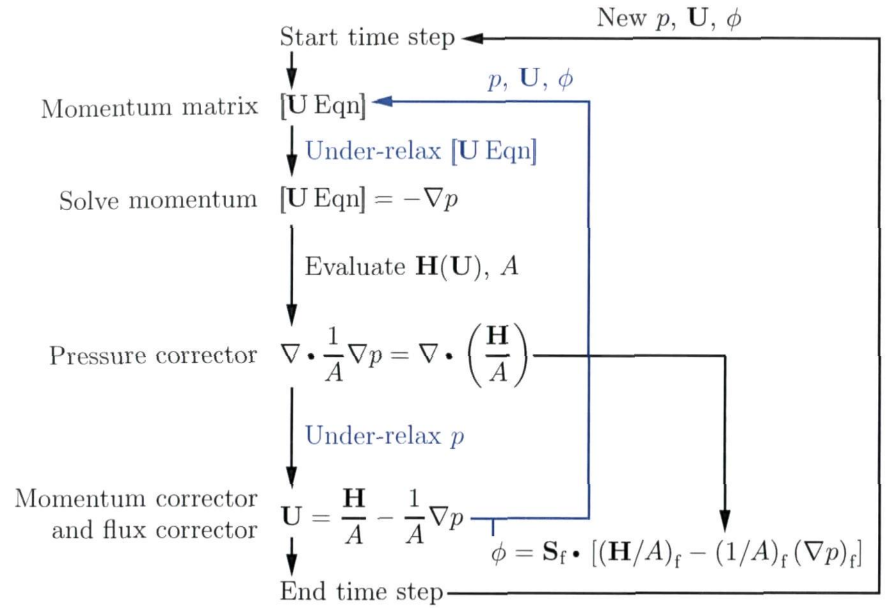
- Loop over the pressure to correct non-orthogonality

```
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
```

- Set value in pRefCell to pRefValue for cases with no fixedValue p boundary

```
pEqn.setReference(pRefCell, pRefValue);
```


SIMPLE: Semi-implicit methods pressure-linked equations



Advantages and disadvantages of PISO and SIMPLE

Algorithm	PISO	SIMPLE
Efficiency	Fast: [$\mathbf{U}$ Eqn] created once	Slower: Under-relaxation
Stability	Can be unstable for Co > 1	Stable for Co > 1
Accuracy	Potential $\partial/\partial t$ error	—

- Advantages/disadvantages relate to algorithm loop structure, particularly construction of the momentum matrix [$\mathbf{U}$ Eqn]

SIMPLE solvers in OpenFOAM

- SIMPLE algorithm generally used in steady-state solvers
- `simpleFoam` is the most basic solver with the SIMPLE algorithm; includes turbulence modelling
- Under-relaxation is performed using the `relax()` function

  - fvMatrix can be under-relaxed by increasing the diagonal and adding an equivalent contribution to source based on existing values

```
UEqn().relax();
```

  - A field can be explicitly under-relaxed using values from the previous iteration; requires the previous iteration to be stored

```
p.storePrevIter();
```

```
p.relax();
```

5.6 Modifying a solver

Exercise: creating a specialised parallel plate flow solver

- Task: to create a new parallelPlateFoam solver by modifying icoFoam
- ...by adding $(\nabla p)_x$ as a body force
- $\Rightarrow$ flow drives itself — no need to apply $(\nabla p)_x$ through boundary conditions
- Create the parallelPlateFoam source directory by copying icoFoam

```
>> cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
>> cp -r icoFoam parallelPlateFoam
>> cd parallelPlateFoam
>> wclean
>> mv icoFoam.C parallelPlateFoam.C
```

- Edit the Make/files file:

```
parallelPlateFoam.C

EXE = $(FOAM_USER_APPBIN)/parallelPlateFoam
```

- Compile the solver

Parallel plate flow solver: modifications

- The $(\nabla p)_x$ terms needs to be added to the momentum equation:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \bullet (\mathbf{U}\mathbf{U}) - \nabla \bullet \nu \nabla \mathbf{U} = -(\nabla p)_x - \nabla p$$

- Add $(\nabla p)_x$ to [U Eqn] so that PISO still constructs a p equation from [U Eqn] $= -\nabla p$
- Let's try adding $(\nabla p)_x = -3$ by the following modification:

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
  - 3.0
);
```

- Try compiling this. It does not compile. Why not? There are two reasons. Think about it...

Parallel plate flow solver: modifications (2)

- Mistake 1: Equations and field algebra does dimension checking — you cannot add/subtract tensors without dimensions
- You can do tensor operations between fields and single dimensioned tensors: the dimensioned<Type> class
- Mistake 2: Momentum is a vector equation; $(\nabla p)_x$ needs to be a vector but "-3" is a scalar
- $\Rightarrow$ $(\nabla p)_x$ needs to be a dimensionedVector
- The dimensioned<Type> class stores 3 items of data

  word used for internal naming of other fields

  dimensionSet the dimensions

  Type the type of value, scalar, vector, tensor, ...

Parallel plate flow solver: modifications (3)

- Let's create a dimensionedVector in createFields.H:

```
dimensionedVector gradPx
(
    "gradPx",
    p.dimensions()/dimLength,
    vector(-3, 0, 0)
);
```

- Now add gradPx to UEqn in parallelPlateFoam.C

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

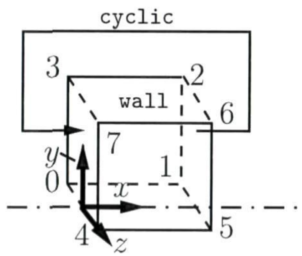
- Compile the solver

Parallel plate flow solver: test case

- Test case: create a parallelPlateCyclic case by cloning the parallel-PlateSymm case

```
>> run
>> cp -r parallelPlateSymm parallelPlateCyclic
>> cd parallelPlateCyclic
```

- Apply cyclic (periodic) boundary on left and right, one cell in x-direction



- Edit the constant/polyMesh/blockMeshDict file
  - blocks: make 1 cell in x-direction

```
hex (0 1 2 3 4 5 6 7) (1 20 1) simpleGrading (1 1 1)
```

  - patches: replace the inlet and outlet patches by:

```
cyclic leftAndRight
(
    (0 4 7 3)
    (2 6 5 1)
)
```


Parallel plate flow solver: test case (2)

- Run blockMesh
- Edit the 0/p and 0/U files; in boundaryField sub-dictionary, add a new patch entry:

```
leftAndRight { type cyclic; }
```

- Change endTime to 5 in the controlDict file
- Run parallelPlateFoam
- Run sample and gnuplot to view results

Parallel plate flow solver: improvement

- Let's make $(\nabla p)_x$ runtime-selectable instead of hard-coded
- In createFields.H:
  - Initialise gradPx to be zero: replace vector(-3.0, 0.0, 0.0) by vector(0.0, 0.0, 0.0)
  - Read a scalar called parallelPlateGradPx from transportProp-erties

```
scalar parallelPlateGradPx
(
    readScalar(transportProperties.lookup("parallelPlateGradPx"))
);
```

  - Set the x component of gradPx; add minus sign, so user supplies a positive value

```
gradPx.value().x() = -parallelPlateGradPx;
```

- Compile
- In the constant/transportProperties file of the parallelPlateCycl-ic case, add the entry

```
parallelPlateGradPx    3.0;
```

- Re-run parallelPlateFoam

Summary of further key classes

- `IOdictionary` class stores data file (dictionaries) on database, from which keywords can be looked up
- `scalar, vector, tensor` include associated algebra
  see `Scalar.H`, `TensorI.H`
- `GeometricField`: actual class for `p`, `U`, `phi`, *etc.*
- `fvMatrix` class for finite volume matrix, *e.g.* `UEqn`

6 Boundary conditions (BCs)

6.1 Introduction to BCs

Overview of boundary condition (BC) modelling

- Information must be supplied at all boundaries to solve an equation for some dependent variable, $\mathbf{Q}$
- Data for $\mathbf{Q}$ can be provided, typically as
  - A `fixedValue` (or Dirichlet) BC specifies values $\mathbf{Q}_b$
  - A `fixedGradient` (or Neumann) BC specified gradients normal to the boundary $(\mathbf{n} \bullet \nabla\mathbf{Q})_b \equiv (\nabla_{\mathbf{n}}\mathbf{Q})_b$
- A geometric constraint can be applied, *e.g.* `symmetryPlane`, `cyclic`
- Complex BCs vary $\mathbf{Q}_b$ or $(\nabla_{\mathbf{n}}\mathbf{Q})_b$ depending on other fields, *e.g.*

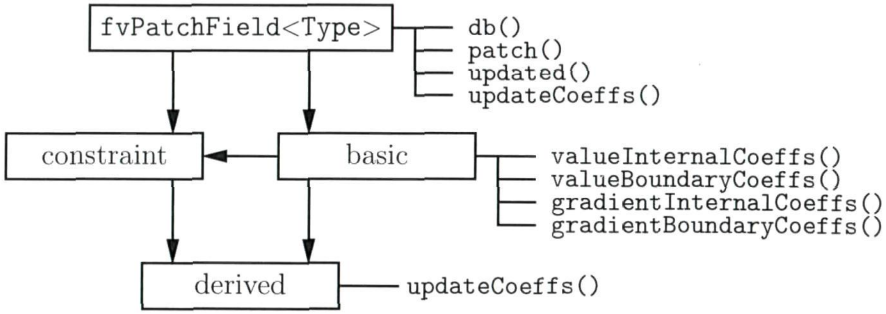
$$p = p_0 - |\mathbf{U}|^2/2$$

- More complex BCs change type according to other local fields, *e.g.* for an outlet

$$\begin{aligned} p &= p_b \quad \text{for Mach} < 1 \\ \nabla_{\mathbf{n}} p &= 0 \quad \text{for Mach} \geq 1 \end{aligned}$$

BC classes

- `fvPatch`: class describing geometry of a boundary patch, *i.e.* the set of faces
- `fvPatchField<Type>`: base class for a field of `Type`, *i.e.* set of values, on an `fvPatch`
- Hierarchy of classes exists for the application of BCs



- basic: data BC base classes, *e.g.* `fixedValue` and `fixedGradient`
- constraint: geometric constraint BC classes, *e.g.* `symmetryPlane`
- derived: higher-level classes of complex BCs

BC classes (2)

- basic and constraint BCs are generally template classes
- For example, `fixedValue` can be for a field of scalars, vectors, tensors, ...
- These low-level BCs hook into matrix discretisation
- No need to modify these classes
- `derived` BCs are more important to users because...
- Users may wish to interpret functionality in existing BCs
- Users may wish to add a new derived BC

Where is the source code for BCs?

1. General BCs in `$FOAM_SRC/finiteVolume/fields/fvPatchFields`

```
>> ls -1 $FOAM_SRC/finiteVolume/fields/fvPatchFields
basic
constraint
derived
fvPatchField
```


2. Model-specific BCs in `derivedFvPatchFields` directories

```
>> find $FOAM_SRC -name derivedFvPatchFields
```

3. Solver-specific BCs in `solvers` directory

```
>> find $FOAM_SOLVERS -name "*FvPatch*"
```

6.2 Understanding existing BCs

Understanding existing BCs

To understand what an existing BC does:

1. Locate the class files
2. In the `.H` file, find the class it is derived from, *e.g.* `fixedValue`
3. In the `.H` file, locate the private data that the BC uses
4. In the `.C` file, examine the `updateCoeffs()` function — it describes the BC

Derived `fixedValue` example: overview

- In an earlier example we do a "back of an envelope" calculation for turbulent kinetic energy k at the inlet
- isentropic turbulence $U_x'^2 = U_y'^2 = U_z'^2 = 0.05$ of the inlet velocity
- Instead, the `turbulentIntensityKineticEnergyInlet` BC calculates this
- Specifies kinetic energy k at an inlet by the `intensity` — a fraction of $|\overline{\mathbf{U}}|$
- Go to the source code

```
>> src
>> cd finiteVolume/fields/fvPatchFields
>> cd derived/turbulentIntensityKineticEnergyInlet
```

Derived `fixedValue` example: base class

- We first ask: which class is this BC derived from?
- Look after `public` the class declaration code in the `.H` file, *e.g.*

```
class turbulentIntensityKineticEnergyInletFvPatchScalarField
:
    public fixedValueFvPatchScalarField
{ ...
```

- $\Rightarrow$ it is derived from `fixedValue`
- $\Rightarrow$ it is specifically for a scalar field, *i.e.* k

Derived `fixedValue` example: input data

- Look in `turbulentIntensityKineticEnergyInlet.H`
- Private data for the BC is `intensity`

```
//- Turbulent intensity as fraction of mean velocity
scalar intensity_;
```

- The `TypeName` used for runtime selection is also there

```
TypeName("turbulentIntensityKineticEnergyInlet");
```

- To use the BC, the user would specify on relevant patches in a case `0/k` file

```
inlet
{
    type         turbulentIntensityKineticEnergyInlet;
    intensity    0.05;
    value        uniform 1; // typically needed by paraview
}
```

- Note: it is safer to initialise BCs with a `value`, even though it is overridden, to stop ParaView complaining

Derived `fixedValue` example: `updateCoeffs()`

```
void Foam::turbulentIntensityKineticEnergyInletFvPatchScalarField::
updateCoeffs()
{
```

- Checks the `updated` switch is off, otherwise returns the function

```
    if (updated())
    {
        return;
    }
```

- Gets a reference to the field data for the corresponding patch in the velocity `volVectorField` (`U`) stored on the database

```
    const fvPatchField<vector>& Up =
        patch().lookupPatchField<volVectorField, vector>("U");
```

- Evaluates the fixed value boundary from the `intensity_` and `U`
- Note! A `fixedValue` boundary value requires a double equal '`==`' to reset it

```
    operator==(1.5*sqr(intensity_)*magSqr(Up));
```

- Sets the `updated` switch to on

```
    fixedValueFvPatchField<scalar>::updateCoeffs();
}
```

Derived `fixedGradient` example

- `buoyantPressure` BC
- Derived from `fixedGradient` but similar structure to the `fixedValue` example
- Difference in the `updateCoeffs()` function: instead of updating the value through `operator==`, the gradient is updated through a non-const access function — `gradient()`
- Looks up density patch field `rho` and gravitational acceleration `g` from the database
- For static pressure, evaluates $\nabla_{\mathbf{n}} p = -\rho(\mathbf{n} \bullet \nabla[\mathbf{g} \bullet \mathbf{x}]) = \rho(\mathbf{n} \bullet \mathbf{g})$

```
const uniformDimensionedVectorField& g =
    db().lookupObject<uniformDimensionedVectorField>("g");

const fvPatchField<scalar>& rho =
    patch().lookupPatchField<volScalarField, scalar>(rhoName_);

gradient() = rho*(g.value() & patch().nf());
```

Derived `mixed` example

- `mixed` is a blend of `fixedValue` and `fixedGradient`
- It stores: a `valueFraction` α; a `refValue` $\mathbf{Q}_b$ and `refGradient` $(\nabla_\mathbf{n}\mathbf{Q})_b$

$$\alpha = \begin{cases} 1 & \text{for } \texttt{fixedValue}\ \mathbf{Q} = \mathbf{Q}_b \\ 0 & \text{for } \texttt{fixedGradient}\ \nabla_\mathbf{n}\mathbf{Q} = (\nabla_\mathbf{n}\mathbf{Q})_b \end{cases}$$

- `inletOutlet` BC — a template `mixed` condition, switches between:
  - `fixedValue` if velocity flux is inward (an inlet)
  - `zeroGradient` if velocity flux is outward (an outlet)
- The `inletOutlet` condition
  - Reads in `inletValue` assigns it to `refValue`
  - Sets `refGradient` to zero
  - Sets `valueFraction` according to the direction of the velocity flux

```
this->valueFraction() = 1.0 - pos(phip);
```

6.3 Creating a customised BC

Creating a customised BC

- We want to create an inlet BC for velocity with the turbulent profile described previously

$$U_x \approx \frac{5}{4} Q \left(1 - \frac{y}{R}\right)^{1/7}$$

- Let's call the new BC `turbulentProfileInletVelocity`
- For speed: we will "hack" something together here
- Similar to the existing `flowRateInletVelocity` BC

$$U_x = Q = \frac{\text{flow rate}}{\text{patch area}}$$

Creating the source code directory and files

- We will compile our new BC into a new turbulentProfileInlet library
- Create a new directory structure in the `$FOAM_RUN` directory

```
>> run
>> mkdir -p turbulentProfileInlet/Make
>> cd turbulentProfileInlet
```

- Copy the `flowRateInletVelocity` class code and rename the files

```
>> DERIVED=$FOAM_SRC/finiteVolume/fields/fvPatchFields/derived
>> cp $DERIVED/flowRateInletVelocity/* .
>> rename 's/flowRateInlet/turbulentProfileInlet/g' *.*
```

- Do a word replacement of `flowRateInlet` by `turbulentProfileInlet`

```
>> sed -i 's/flowRateInlet/turbulentProfileInlet/g' *.*
```

- We have a new `turbulentProfileInletVelocity` class — that does the same as `flowRateInletVelocity`

Compiling the library

- Create the necessary files in the `Make` directory
- Our class is derived from bases classes in the finiteVolume library
- Copy an `options` file from elsewhere; one exists that is exactly what we need

```
>> cp $FOAM_SRC/turbulenceModels/LES/LESfilters/Make/options Make/
```

- `Make/options` — copied (above):

```
EXE_INC = -I$(LIB_SRC)/finiteVolume/lnInclude

LIB_LIBS = -lfiniteVolume
```

- `Make/files` — create ourselves:

```
turbulentProfileInletVelocityFvPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/libturbulentProfileInlet
```

- Compile the library

```
>> wmake libso
```

The existing BC

- Open turbulentProfileInletVelocityFvPatchVectorField.C in an editor
- Examine the `updateCoeffs` function
- Existing code calculates average flow speed `avgU`

```
121    scalar avgU = -flowRate_/gSum(patch().magSf());
```

- ...then it stores the unit normal vector `patch().nf()`

```
123    vectorField n = patch().nf();
```

- ...then applies the average speed in that direction

```
131    operator==(n*avgU);
```

- `patch().nf()` points out of domain so negative `avgU` ensures inflow
- Geometric data accessed via the `patch()` function which returns the `fvPatch`

Modifying the BC

- Our flow speed varies across the patch, not a single average value
- ⇒ it must be a `scalarField`, not a `scalar`
- Replace the name `avgU` with `magU`
- Replace the `magU` (`avgU`) constructor (line 121) with

```
121    scalarField y = patch().Cf().component(1);
122    scalar R = 0.01;
123    scalarField magU = -5.0/4.0*flowRate_*pow((1.0 - y/R), 1.0/7.0);
```

- Note: we are now using `flowRate_` as Q "flow rate per unit area"
- Recompile the library

```
>> wmake libso
```


Testing the new BC

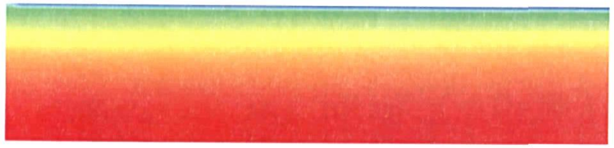
- Go back to the `poiseuilleHighRe` case
- We can use our new library by linking it to the solver dynamically at runtime
- ...by adding the new library to the `libs` list in `system/controlDict` with

```
libs ("libturbulentProfileInlet.so");
```

- Change the BC for the `inlet` patch in `0/U`:

```
inlet
{
    type        turbulentProfileInletVelocity;
    value       uniform (1 0 0);
    flowRate    1.0;
}
```

- Run `pisoFoam`
- Solution: at the inlet, velocity profile is similar to the developed flow downstream



A Finite volume discretisation

The finite volume method

The finite volume (FV) method is one form of equation discretisation. **In general**:

- Discrete quantities $\mathbf{Q}_i$ are primarily stored at the centroids ($\mathbf{C}$) of each cell i
- Terms in a PDE are discretised by integrating over the cell volume V
- Integrals over V of spatial derivatives are converted to integrals over the cell surface S using Gauss's Theorem

$$\int_V \nabla \star \mathbf{Q}\; dV = \int_S d\mathbf{S} \star \mathbf{Q} \quad (\star = \bullet, \times, \otimes)$$

- Other integrals over V are approximated assuming $\mathbf{Q}$ is uniform across the cell

$$\int_V \mathbf{Q}\; dV = \mathbf{Q}_\mathrm{P} V_\mathrm{P}$$

- Integrals over S are approximated by summations over cell faces (f) of products of the normal area vector $\mathbf{S}_\mathrm{f}$ and $\mathbf{Q}_\mathrm{f}$, interpolated from cell centres to faces

$$\int_S d\mathbf{S} \star \mathbf{Q} = \sum_\mathrm{f} \mathbf{S}_\mathrm{f} \star \mathbf{Q}_\mathrm{f}$$

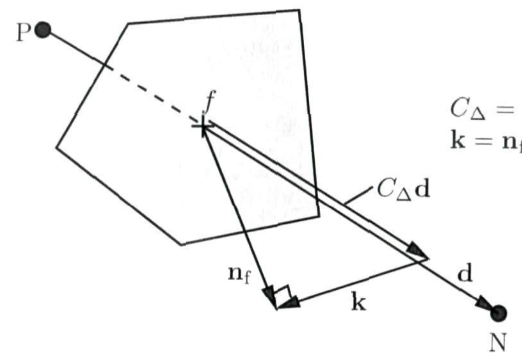
Laplacian discretisation

- `laplacian(Gamma, Q)` $\Rightarrow \nabla \bullet (\Gamma \nabla \mathbf{Q})$

$$\Rightarrow \int_V \nabla \bullet (\Gamma \nabla \mathbf{Q})\; dV = \int_S d\mathbf{S} \bullet (\Gamma \nabla \mathbf{Q}) \approx \sum_\mathrm{f} \Gamma_\mathrm{f} \mathbf{S}_\mathrm{f} \bullet (\nabla \mathbf{Q})_\mathrm{f} = \sum_\mathrm{f} \Gamma_\mathrm{f} |\mathbf{S}_\mathrm{f}| \mathbf{n}_\mathrm{f} \bullet (\nabla \mathbf{Q})_\mathrm{f}$$

- Surface normal gradient `snGrad()`

$$\mathbf{n}_\mathrm{f} \bullet (\nabla \mathbf{Q})_\mathrm{f} = \underbrace{C_\Delta \mathbf{d} \bullet (\nabla \mathbf{Q})_\mathrm{f}}_{orthogonal} + \underbrace{\mathbf{k} \bullet (\nabla \mathbf{Q})_\mathrm{f}}_{non-orthogonal}$$
$$= C_\Delta (\mathbf{Q}_\mathrm{N} - \mathbf{Q}_\mathrm{P}) + \mathbf{k} \bullet (\nabla \mathbf{Q})_\mathrm{f}$$



$C_\Delta = 1/|\mathbf{n}_\mathrm{f} \bullet \mathbf{d}| \quad (= 1/\Delta x)$
$\mathbf{k} = \mathbf{n}_\mathrm{f} - C_\Delta \mathbf{d}$

Laplacian discretisation (2)

- `fvm::laplacian(Gamma, Q)`

$$\nabla \bullet (\Gamma \nabla \mathbf{Q}) \Rightarrow \sum_\mathrm{f} \left(\Gamma_\mathrm{f} |\mathbf{S}_\mathrm{f}| C_\Delta (\mathbf{Q}_\mathrm{N} - \mathbf{Q}_\mathrm{P}) + \Gamma_\mathrm{f} |\mathbf{S}_\mathrm{f}| \mathbf{k} \bullet (\nabla \mathbf{Q})_\mathrm{f} \right)$$

- Contribution to matrix and source from face spanning cells 1 and 2, (positive $\mathbf{k}$ for owner cell P= 1):

$$\begin{bmatrix} -\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|C_\Delta & +\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|C_\Delta & \dots & M_{1N} \\ +\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|C_\Delta & -\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|C_\Delta & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \vdots \\ \mathbf{Q}_N \end{bmatrix} = \begin{bmatrix} -\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|\mathbf{k} \bullet (\nabla \mathbf{Q})_\mathrm{f} \\ +\Gamma_\mathrm{f}|\mathbf{S}_\mathrm{f}|\mathbf{k} \bullet (\nabla \mathbf{Q})_\mathrm{f} \\ \vdots \\ \mathbf{B}_N \end{bmatrix}$$

Gradient and divergence discretisation

- `fvc::div(Q)` $\Rightarrow \nabla \bullet \mathbf{Q}$

$$\Rightarrow \int_V \nabla \bullet \mathbf{Q}\; dV = \int_S d\mathbf{S} \bullet \mathbf{Q} \approx \sum_\mathrm{f} \mathbf{S}_\mathrm{f} \bullet \mathbf{Q}_\mathrm{f}$$

- $\mathbf{Q}_\mathrm{f}$ usually evaluated by `linear` interpolation

- Must be `fvc::` - decreases rank by one (*e.g.* $\mathbf{Q}$ = vector, $\nabla \bullet \mathbf{Q}$ = scalar)
- `fvc::grad(Q)` $\Rightarrow \nabla \mathbf{Q}$

$$\Rightarrow \int_V \nabla \mathbf{Q}\, dV = \int_S d\mathbf{S}\,\mathbf{Q} \approx \sum_{\mathrm{f}} \mathbf{S}_{\mathrm{f}}\,\mathbf{Q}_{\mathrm{f}}$$

- $\mathbf{Q}_{\mathrm{f}}$ usually evaluated by `linear` interpolation
- Must be `fvc::` - increases rank by one (*e.g.* $\mathbf{Q}$ = vector, $\nabla \bullet \mathbf{Q}$ = 2nd rank tensor)
- `fvc::laplacian(Q)` $\neq$ `fvc::div(fvc::grad(Q))`

Convection - a special case of divergence

- `div(phi, Q)` $\Rightarrow \nabla \bullet (\mathbf{U}\mathbf{Q})$

$$\Rightarrow \int_V \nabla \bullet (\mathbf{U}\mathbf{Q})\, dV = \int_S d\mathbf{S} \bullet (\mathbf{U}\mathbf{Q}) \approx \sum_{\mathrm{f}} \mathbf{S}_{\mathrm{f}} \bullet \mathbf{U}_f \mathbf{Q}_{\mathrm{f}} = \sum_{\mathrm{f}} \phi_{\mathrm{f}} \mathbf{Q}_{\mathrm{f}}$$

- Typically implicit in $\mathbf{Q}$: `fvm::div(phi, Q)`, where `phi = (fvc::interpolate(U) & mesh.Sf())`
- Interpolation of $\mathbf{Q} \rightarrow \mathbf{Q}_{\mathrm{f}}$ is critical for stability
  - `upwind`: bounded, 1st-order accurate

$$\mathbf{Q}_{\mathrm{f}} = \begin{cases} \mathbf{Q}_{\mathrm{P}} & \text{for } \phi_{\mathrm{f}} \geq 0 \\ \mathbf{Q}_{\mathrm{N}} & \text{for } \phi_{\mathrm{f}} < 0 \end{cases}$$

  - `linear`: unbounded, 2nd-order accurate
  - Other schemes, *e.g.* SFCD, `limitedLinear`, `vanLeer`, *etc.*, effectively blend between `upwind` and `linear`

Time derivative discretisation

- `ddt(rho, Q)`

$$\frac{\partial(\rho\mathbf{Q})}{\partial t} \Rightarrow \int_V \frac{\partial(\rho\mathbf{Q})}{\partial t}\, dV \approx \frac{\rho_{\mathrm{P}}\mathbf{Q}_{\mathrm{P}}V - \rho_{\mathrm{P}}^o\mathbf{Q}_{\mathrm{P}}^o V^o}{\Delta t}$$

- Contribution to matrix and source from current time (no superscript) and old time (o superscript) at P= 1:

$$\begin{bmatrix} \rho_{\mathrm{P}}V/\Delta t & \dots & \dots & M_{1N} \\ \vdots & \ddots & \dots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N1} & M_{N2} & \dots & M_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \vdots \\ \mathbf{Q}_N \end{bmatrix} = \begin{bmatrix} \rho_{\mathrm{P}}^o\mathbf{Q}_{\mathrm{P}}^o V^o/\Delta t \\ \vdots \\ \vdots \\ \mathbf{B}_N \end{bmatrix}$$

- `fvm::ddt(rho, Q)` The usual choice — time derivatives almost always implicit
- `Euler` scheme shown above, first order in time
- Other second-order schemes available including `backward` differencing and `CrankNicholson`

B The USB memory stick

B.1 Booting the USB OpenFOAM/Linux memory stick

- The stick must be booted from the machine BIOS (i.e. when the machine is switched on).
- On newer computers the USB device can be detected as a hard drive (`USB-HDD0`).
  - If your machine in one of these, you need to press a specific key — typically one of either `F2`, `F10`, `F11` or `ESC` — immediately after powering on the machine.
  - Some machines do not respond to a single (even long) press of the key, but repeated pressing works effectively.
  - Once in the "Boot Menu", select `USB DISK (USB DISK 2.0)` with `up/down` arrows and then hit `enter` to select and resume startup.
- On computers that are a bit older, or uses a simplified BIOS, you may not have a `Boot Menu` option. In this case you will need to make the system detect and boot your USB device by changing the settings in the BIOS.
  - Again, you need to enter the BIOS settings by pressing a specific function key.
- If your BIOS lists the USB memory stick as a hard drive, you should select it as the `1st boot device`.
- The preferred choice of boot option is `USB-HDD`; `USB-ZIP` might work, but `USB-FDD` is not supported.
- It is recommended to remove other USB boot options from the boot priority list, e.g. if booting with `USB-HDD`, remove `USB-ZIP`.
- Be careful to note that on some BIOSes you effectively need to make 2 selections:
  - Move `hard drive` to the top of the boot priority list;
  - Move `USB` to the top of the hard drive priority list.
- Once booting, the display briefly reads `SYSLINUX loading...`

- A menu of languages appears from which the user can select a preference.
- If the user does not select English, they maybe asked later whether they want certain standard directory names to be translated; select `No`.
- The kernel loads; be patient, it can take some time.
- Next, Ubuntu Linux loads; again be patient.
- If all is working a live Linux session is automatically started.

B.2 Shutting down the memory stick

- Shut down by pressing the off bottom at the top right of the terminal screen.
- When asked to `remove the disc... and press ENTER to continue`, simply press `enter`.
- Remove the USB stick once the machine is switched off.

B.3 General use

- DO NOT REMOVE THE MEMORY STICK WHILE IN USE, ESPECIALLY IF IT IS FLASHING: this is a likely way to corrupt the file system on the stick, which is very difficult to restore.
- Problem at login: After Ubuntu is loaded, it can occasionally happen that the live session fails to start, giving a message saying that it does not have write permission for the `.ICEauthority` file. If this occurs, do the following:
  - In the menu at the bottom left of the screen select `Options->Failsafe login`.
  - In the login screen, type user `ubuntu`, leaving password blank.
  - A small terminal window appears in the Failsafe login.
  - In the terminal type: `rm -f .ICEauthority*` (hit `enter`).
  - Now type: `exit` to return to the login screen
  - In the menu at the bottom left of the screen select `Options->GNOME login`.

  - In the login screen, type user `ubuntu`, leaving password blank.
  - You will now be logged into the live session.

- Problem at boot (1): If the system hangs towards the end of the boot process, it is often due to a problem with the X-server starting the graphics. If this occurs, reboot the machine and, at the point where the user selects the language, hit the `F6` function key. This brings up a menu of additional boot options. The user should select `nomodeset`, then hit `Esc` to continue booting.

- Problem at boot (2): If memory stick is not working, the Master Boot Record may need fixing

  - Mount the USB drive on a Linux machine, check the device name, here we will call it `/dev/sdx`.
  - Then either try running `install-mbr /dev/sdx` (requires mbr package in ubuntu).
  - Or try `lilo -M /dev/sdx` (requires lilo package).

- Problem at boot (3): if a message appears like `Boot Error` immediately that the BIOS attempts to boot from the disk then the problem may be due to a setting in the BIOS. To fix this:

  - Go into BIOS Boot Menu and search for `USB Mass Storage Emulation Type`.
  - If this is set to `Default:<Auto>`, change it to `<All Fixed Disc>` or something similar.

- Problem at boot (4): if the boot process appears to hang following a message about `fd0`, it is likely because the system is searching for a floppy drive that does not exist. To fix this, go into BIOS Boot Menu and disable `Floppy drive` from the list of boot devices.