OpenFOAM® tutorial collection

Steady-state laminar flow

Provided by

Initial author:	Andras Horvath
E-Mail:	andras.horvath@rheologic.at
Affiliation:	Rheologic GmbH, www.rheologic.net

OpenFOAM version:	tested using v2012 (OpenFOAM.com)
Paraview version:	tested using 5.7.0
Solver:	simpleFoam
Models:	Steady-state, laminar, incompressible Newtonian fluid, moving (rotating) walls

| Licensed under: | Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0) |

| Updated: | May 4, 2021 by Andras Horvath |

1 Introduction

This documentation is part of an effort to give OpenFOAM® users curated and peer-reviewed tutorials of high quality to various solvers of the OpenFOAM® toolbox. This walk-through is suitable for users new to OpenFOAM® but has some prerequisites:

- Good knowledge of Linux/UNIX environments and commands
- User is familiar with a typical Bourne-Shell like BASH
- Knowledge of an effective text editor
- Some background in numerics and the theory behind CFD
- A working installation of OpenFOAM®

Note: The installation process of OpenFOAM® for your operating system of choice is not covered in this tutorial. Also note that mesh generation is not part of this tutorial. A working mesh is included in the case-files though.

In this tutorial we would like to give a concise introduction to simpleFoam, a solver for steady-state simulation of incompressible laminar or turbulent flow. Here we will stick to the case of laminar flow. No turbulence models will be employed in the simulation. We chose to simulate part of a classic experiment by G. I. Taylor published in 1923 titled "Stability of a Viscous Liquid contained between Two Rotating Cylinders". In a closed geometry interesting secondary flow patterns (and temporal instabilities not covered in this tutorial) evolve in the gap between concentric, roating cylinders. This tutorial by no means gives a complete and perfectly correct numerical description and solution to phenomena discovered by Taylor almost 100 years ago. Also meshing (generation of the finite volume grid) is not covered in this tutorial. The tutorial is designed to show the OpenFOAM®-novice handy ways to run simulations in serial and parallel, monitor residuals, quickly modify (linearly transform) geometries and boundary conditions and do the post-processing and visualisation of results.

2 Case, Geometry and Mesh

If you unpack the .tar.gz archive and have a look around you will find following files and directories:

```
0: p U

0_template: p U

constant: polyMesh/ polyMesh_originalSize/ transportProperties turbulenceProperties

system:   controlDict decomposeParDict fvSchemes fvSolution
```

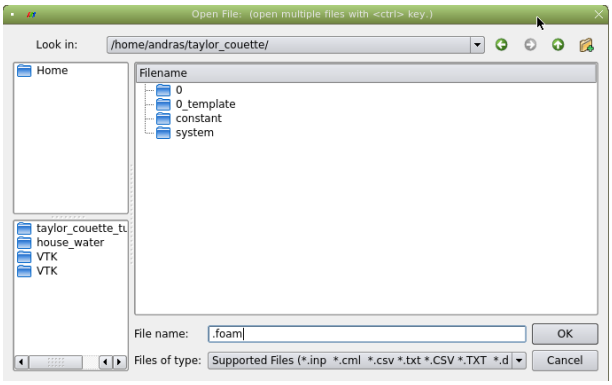
When working with OpenFOAM® like with every versatile system there is no *one correct way* to get the work done. Every user has a different style, every problem requires different strategies. Much of what follows comes from experience and previous failiures of the author. Take it with a grain of salt.

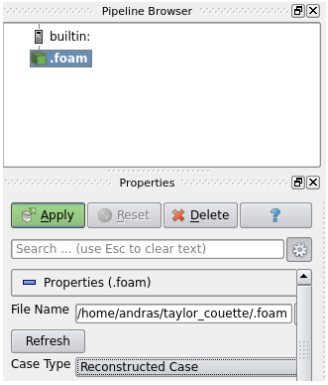
First things first: what is a *case*? A case is the combination of geometry definition (the finite volume mesh), configuration files (called dictionaries in OpenFOAM® language), definition of boundary conditions and initial conditions, custom function and results all structured in many files and directories. The 0-directory contains the initial- and boundary-conditions. The (initial) mesh is in `constant/polyMesh`. Most config files are in `system/`. Material and turbulence properties are in `constant/`. If you are interested you can go deeper into the subdirectories and see how the data is structured further. This tutorial provides extra backup directories for `0` and `constant/polyMesh` in case something happens (and it will) to important original data.

The best way to learn cooking is to cook. In this manner we'll dive right into it. Let's have a look at our geometry using Paraview, a post-processor and data visualiser. Paraview is a complex software of its own and we will just see the tip of the iceberg here. Whether you installed Paraview bundled with OpenFOAM® or downloaded the lastest version from somewhere else does not matter so much, the functionality is basically the same. The screenshots presented here are still from Paraview-4.1.0. Note: your GUI might vary a little. First: get into the taylor_couete directory and create a file:

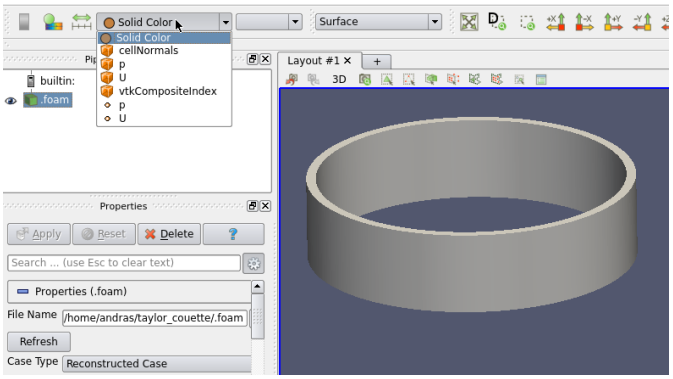
```
$ cd taylor_couette
$ touch .foam
```

Type ".foam" into the "File name:" text-box in Paraview, click "OK", select "Reconstructed Case" from the "Case Type" drop-down menu and click "Apply".





Next you should see a blue circle on a gray backdrop in "Layout #1". Next: select "Solid Color" and you should see a shaded gray ring like this after rotating the view direction:



Rotate the geometry using LMB (Left Mouse Button), zoom with RMB (Right Mouse Button), move with MMB (Middle Mouse Button). If you select "Surface With Edges" from the pull down menu "Surface" the surface mesh will be shown. CTRL-Space will produce a pop-up window. Type S L and select the "Slice" filter (also found in the manu bar at Filters → Common → Slice). Select "Y Normal" and "Apply" and have a look at the mesh in detail. Play around in the pipeline browser and turn on and off visibility with the eye symbol. You now have an overview what the geometry and mesh look like. The field variables p and U are already found by Paraview and read from `0/`, the mesh is read from `constant/polyMesh`. The combination of both is needed to post-process any results. If you lose or destroy your mesh data the field variables themselves are worthless because they contain no geometry information. Bear in mind that field variables (a long list of doubles in ASCII or binary form) and mesh correspond to each other!

3 Boundary conditions

"0" is the first and initial so called time-directory. The semantics of storing field data for steady and unsteady (transient) simulations in OpenFOAM® is the same. They are stored in numbered directories where the number is simply advanced every iteration and every processing step by the setting "deltaT" found in `system/controlDict`. Just by looking at numbered directories in an OpenFOAM®-case there is no way to find out wether the simulation was transient or steady-state as it depends solely on the used solver or tool what those numbers mean.

Have a look at `system/controlDict`. deltaT is defined as 1. This is standard for steady-state solvers. In our tutorial the numbers of the directories containing the saved field variables (here: pressure p and velocity U) will denote the iteration number. In our case the definitions in `system/controlDict` are:

```
application     simpleFoam;
startFrom       latestTime;
stopAt          endTime;
endTime         25000;
deltaT          1;
writeControl    runTime;
writeInterval   400;
purgeWrite      0;
writeFormat     ascii;
writePrecision  9;
writeCompression yes;
timeFormat      general;
timePrecision   9;
runTimeModifiable no;
```

The settings are described in detail in the official OpenFOAM®-documentation. To cut it short: The solver will write out field variables to disk every 400 iterations and also if the convergence criteria are met (in this case the solvers also stops iterating). At most 25000 iterations will be calculated.

Have a look at the boundary conditions for the velocity field located in `0/U`. The interesting part outside the C++-style comments and the header is shown here:

```
boundaryField
{

    "inner_wall"
    {

        type            rotatingWallVelocity;
        origin          ( 0 0 0 );
        axis            ( 0 0 1 );
        omega           4; // rad/s

    }

    "outer_wall"        {type fixedValue; value uniform ( 0 0 0 );}

    "top"               {type slip;}
    "bottom"            {type slip;}

}
```

In this file all the boundaries defined in `constant/polyMesh` need to be adressed. Otherwise OpenFOAM® will throw an error. (POSIX.2 regular expressions are supported. You could leave out the line containing "outer_wall" and e.g. substitute "inner_wall" with "*._wall". This is valid input for OpenFOAM®.) In `0/p` and `0/U` two important types of boundary conditions are present "fixedValue" (Dirichlet type) and "zeroGradient" (von Neumann type). "rotatingWallVelocity" is a calculated type where the actual wall velocity depends on the geometry (radial distance) and angular velocity ($\omega =$ omega) defined in rad s^{-1}. OpenFOAM® adheres strictly to SI-units and does run-time unit checking (see official documentation). By changing the entries in `0/p` and `0/U` boundary conditions and initial conditions are modified.

4 Solution

Let's run our first simulation (assuming OpenFOAM® is installed in the users home directory) and watch the solver's output:

```
$ . ~/OpenFOAM/OpenFOAM-v1812/etc/bashrc
$ cd taylor_couette
$ simpleFoam > log &
$ tail -f log
```

The first command sources the OpenFOAM® environment. You don't have execute this every time if it is already part of your users `$HOME/.bashrc`. Without this command OpenFOAM® binaries

and libraries are invisible to the active shell. Lot's of shell variables essential to OpenFOAM® are set. Other software is not necessarily compatible to the OpenFOAM® -environment. The user can unset the variables with the "wmUnset" command (uppercase U!). While the solver keeps running in the background we will present a very simple example to monitor the residuals using foamLog, a utility that creates X-Y-Files readable by e.g. gnuplot. foamLog informs you which data is extracted from the logfile.

```
$ . ~/OpenFOAM/OpenFOAM-v1812/etc/bashrc
```

After the simulation run finishes you can use the *foamLog* utility to parse the log file an generate appropriate files for input into Gnuplot like this:

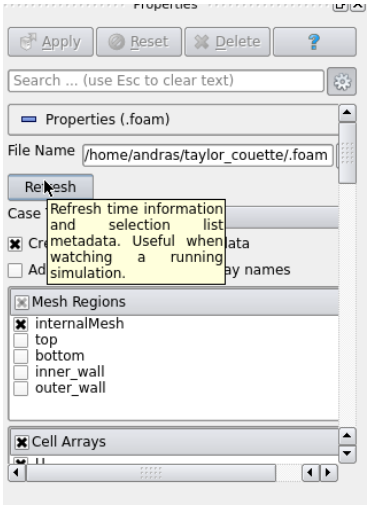
```
$ foamLog log
$ wmUnset
$ cd logs
$ gnuplot
gnuplot> plot 'UzFinalRes_0' with lines
```

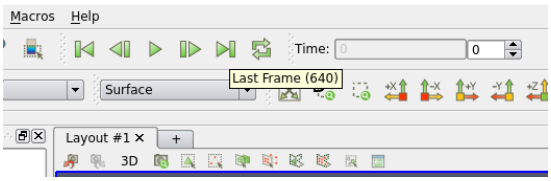
By the end of the simulation run the initial residuals should have fallen (in the best case) by several orders of magnitude. The simulation is now complete. Notice two new directories containing results.

```
$ cd ..
$ ls -F
0/  _0_org/  1200/  1246/  400/  800/  constant/  log  system/
```

5 Post processing and visualisation

The description of the fun part is short: get back to Paraview and hit "Refresh" in the Properties Window. The folder structure will be re-read. If you point on the button for the last time-step you will notice the tool-tip says "Last Frame (1246)". Click on the button and view the results in Paraview. Notice that "Time" gives 1246 which is the iteration number. For a transient simulation it will actually be the simulated time.





The 3D-data of p(scalars) and U(vectors) can readily be selected from the coloring drop-down menu. For swirling flows vorticity can be relevant. The solver itself did not produce this information, but it can be optained by a built-in post-processing tool.

```
$ postProcess -func vorticity
```

Notice how vorticity is calculated for each time-step (here it means iteration count). If you only want it to be calculated for the latest time-directory add the option "-time latestTime" to the command. Notice the output of the postProcess tool.

6 Parallelisation

This is all swell but we want the results quicker. Let's speed things up by running the solver in parallel. OpenFOAM® uses a method called domain-decomposition where sub-parts of the mesh and variables are run on seperate CPUs (processors, cores, threads, ...) communicating via MPI. To be able to run in parallel the domain needs to be split up into connected chunks (one for each processor) first. This is done using decomposePar which looks in `system/decomposeParDict` for its settings:

```
numberOfSubdomains 8;

//method          scotch;

method          hierarchical;

hierarchicalCoeffs
{
    n               ( 2 2 2 );
    delta           0.001;
    order           xyz;
```

```
}

manualCoeffs
{
    dataFile        "";
}

//distributed     no;

//roots           ( );
```

Here we are decomposing into 8 sub-domains with an algorithm called “hierarchical“. The part hierarchicalCoeffs{...} is called a sub-dictionary. The configuration here says: split up the geometry in 8 parts, 2 in x-direction, 2 in y-direction and 2 in z-direction, first split in x, then in y and last in z direction, allow a maximum relative tolerance in cell distribution of 0.001. Other methods and parameters are commented out. Adapt the count of sub-domains to number of available CPU-cores, change n accordingly and run the simulation, e.g. 4x parallel, like this:

```
$ . ~/OpenFOAM/OpenFOAM-v1812/etc/bashrc
$ decomposePar
$ mpirun -np 4 simpleFoam > log.parallel &
$ foamLog log.parallel
$ wmUnset
$ gnuplot
gnuplot> plot 'UzFinalRes_0' with lines
```

Don't forget to run foamLog at the end of the simulation to update the X-Y-files. How much speed-up you can gain by running in parallel depends very much on the size of the mesh. If you fall below 20 000 cells per CPU-core the communication overheads between the sub-domains usually start to eat up performance gains by a higher core count. YMMV, so try to find the limits of your system by doing benchmarks. Note: Parallel runs of OpenFOAM will yield slightly different results than serial runs due to rounding errors.

For parallel runs post-processing in Paraview requires to set the case type to “Decomposed Case“. A different method, sometimes more suitable, is to reconstruct the fields which have been split up into the processor directories and convert them into native VTK-format efficiently read by Paraview. The VTK-files in the VTK-directory can be directly opened in Paraview. Give it a try...

```
$ . ~/OpenFOAM/OpenFOAM-v1812/etc/bashrc
$ reconstructPar
$ foamToVTK
```

7 Comparison of CFD-results with theory and experiments

By using a "Glyph" filter on a slice 2D arrows are overlayed clearly visualising the vortex cores. The "Ruler" filter in Paraview (hit CTRL-space) can measure the distance of one vortex-center to another. The fit with experimental data is not bad at all...

values of d/θ, are shown by means of circles and the observed points by means of dots. The curve is drawn roughly through the dots. Unfortunately, owing to an oversight, no observation was taken for the value $\mu = -1{\cdot}5$, but it will be seen that the calculated point lies almost exactly on the observed curve.

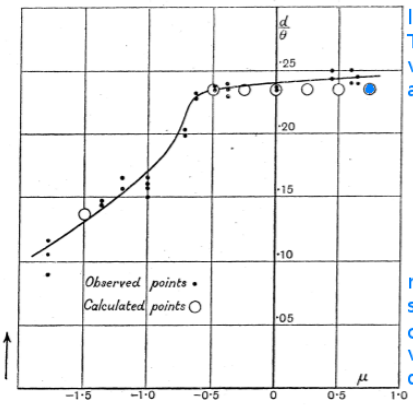
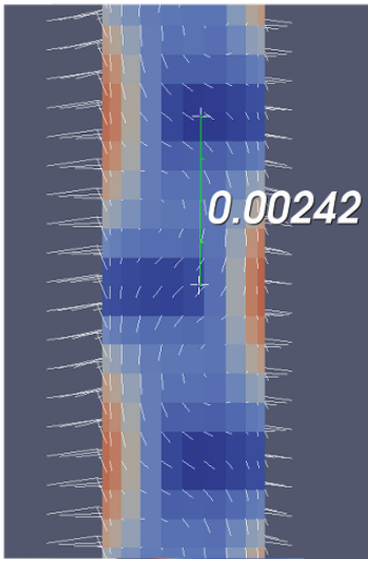


Fig. 12. Comparison between observed and predicted spacing of vortices for various values of μ; case when $R_1 = 3{\cdot}80$ cm., $R_2 = 4{\cdot}035$ cm.

left:
Taylor (1922)
vortex-distance:
approx: 0.24 cm

right:
simulation result
color: vorticity
vortex-
distance: 0.242 cm



8 Excercises

- Try different visualisation filters in Paraview.
- Save a state file in Paraview and come back in a fresh session where you left off by loading the state-file.
- What is the limit for n when decomposing? How many cores before there is no more gain in speed?
- Try different n for decomposition.
- Are single- and multi-core results identical? Have a look at the log files. Have a look at the time-directories and data. Can you open them in an editor?
- Change the boundary conditions for angular velocity. Does the vortex spacing change if the outer cylinder rotates at half the angular speed of the inner cylinder? Compare results.
- Reduce the height of the rotating, concentric cylinders by using the transformPoints command in the taylor_couette directory like this `transformPoints -scale "( 1 1 0.5 )"`. How do the results compare?
- Get back the original mesh by deleting `constant/polyMesh` and recursively copying `constant/polyMesh_originalSize` to `constant/polyMesh`.

Rheologic GmbH
Liniengasse 40/12
1060 Vienna
Austria

andras.horvath@rheologic.net
www.rheologic.net
+43 699 819 032 36

Legal disclaimer:

This offering is not approved or endorsed by OpenCFD® Limited, the producer of the OpenFOAM®, software and owner of the OpenFOAM®and OpenCFD® trade marks. OpenFOAM®is a registered trade mark of OpenCFD® Limited, the producer of the OpenFOAM®software.