Flow around a cylinder – 10 < Re < 2 000 000
Incompressible and compressible flow

Physical and numerical side of the problem:

- In this case we are going to solve the flow around a cylinder. We are going to use incompressible and compressible solvers, in laminar and turbulent regime.
- Therefore, the governing equations of the problem are the incompressible/compressible laminar/turbulent Navier-Stokes equations.
- We are going to work in a 2D domain.
- Depending on the Reynolds number, the flow can be steady or unsteady.
- This problem has a lot of validation data.

All the dimensions are in meters
Flow past a cylinder – From laminar to turbulent flow

Workflow of the case

1. blockMesh
   Or
   fluentMeshToFoam

2. icoFoam
    pisoFoam
    pimpleFoam
    pimpleDyMFoam
    simpleFoam
    rhoPimpleFoam
    interFoam
    sonicFoam
    potentialFoam
    mapFields

3. functionObjects

4. postProcessing
   utilities

5. sampling
6. paraview

NOTE: One single mesh can be used with all solvers and utilities.
Flow past a cylinder – From laminar to turbulent flow

Vortex shedding behind a cylinder

- Creeping flow (no separation)
  - Steady flow
  - $Re < 5$

- A pair of stable vortices in the wake
  - Steady flow
  - $5 < Re < 40 - 46$

- Laminar vortex street (Von Karman street)
  - Unsteady flow
  - $40 - 46 < Re < 150$

- Laminar boundary layer up to the separation point, turbulent wake
  - Unsteady flow
  - $150 < Re < 300$

- Boundary layer transition to turbulent
  - Unsteady flow
  - $300 < Re < 3 \times 10^5$

- Turbulent vortex street, but the wake is narrower than in the laminar case
  - Unsteady flow
  - $3 \times 10^5 < Re < 3 \times 10^6$

- $3 \times 10^6 > Re$

Drag coefficient

Strouhal number
Flow past a cylinder – From laminar to turbulent flow

Some experimental \(^{(E)}\) and numerical \(^{(N)}\) results of the flow past a circular cylinder at various Reynolds numbers

<table>
<thead>
<tr>
<th>Reference</th>
<th>(c_d – \text{Re} = 20)</th>
<th>(L_{rb} – \text{Re} = 20)</th>
<th>(c_d – \text{Re} = 40)</th>
<th>(L_{rb} – \text{Re} = 40)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1] Tritton (^{(E)})</td>
<td>2.22</td>
<td>–</td>
<td>1.48</td>
<td>–</td>
</tr>
<tr>
<td>[2] Cuntanceau and Bouard (^{(E)})</td>
<td>–</td>
<td>0.73</td>
<td>–</td>
<td>1.89</td>
</tr>
<tr>
<td>[3] Russel and Wang (^{(N)})</td>
<td>2.13</td>
<td>0.94</td>
<td>1.60</td>
<td>2.29</td>
</tr>
<tr>
<td>[4] Calhoun and Wang (^{(N)})</td>
<td>2.19</td>
<td>0.91</td>
<td>1.62</td>
<td>2.18</td>
</tr>
<tr>
<td>[5] Ye et al. (^{(N)})</td>
<td>2.03</td>
<td>0.92</td>
<td>1.52</td>
<td>2.27</td>
</tr>
<tr>
<td>[6] Fornbern (^{(N)})</td>
<td>2.00</td>
<td>0.92</td>
<td>1.50</td>
<td>2.24</td>
</tr>
<tr>
<td>[7] Guerrero (^{(N)})</td>
<td>2.20</td>
<td>0.92</td>
<td>1.62</td>
<td>2.21</td>
</tr>
</tbody>
</table>

\(L_{rb}\) = length of recirculation bubble, \(c_d\) = drag coefficient, \(\text{Re}\) = Reynolds number,

Flow past a cylinder – From laminar to turbulent flow

Some experimental (E) and numerical (N) results of the flow past a circular cylinder at various Reynolds numbers

<table>
<thead>
<tr>
<th>Reference</th>
<th>$c_d$ – Re = 100</th>
<th>$c_l$ – Re = 100</th>
<th>$c_d$ – Re = 200</th>
<th>$c_l$ – Re = 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1] Russel and Wang (N)</td>
<td>1.38 ± 0.007</td>
<td>± 0.322</td>
<td>1.29 ± 0.022</td>
<td>± 0.50</td>
</tr>
<tr>
<td>[2] Calhoun and Wang (N)</td>
<td>1.35 ± 0.014</td>
<td>± 0.30</td>
<td>1.17 ± 0.058</td>
<td>± 0.67</td>
</tr>
<tr>
<td>[3] Braza et al. (N)</td>
<td>1.386 ± 0.015</td>
<td>± 0.25</td>
<td>1.40 ± 0.05</td>
<td>± 0.75</td>
</tr>
<tr>
<td>[4] Choi et al. (N)</td>
<td>1.34 ± 0.011</td>
<td>± 0.315</td>
<td>1.36 ± 0.048</td>
<td>± 0.64</td>
</tr>
<tr>
<td>[5] Liu et al. (N)</td>
<td>1.35 ± 0.012</td>
<td>± 0.339</td>
<td>1.31 ± 0.049</td>
<td>± 0.69</td>
</tr>
<tr>
<td>[6] Guerrero (N)</td>
<td>1.38 ± 0.012</td>
<td>± 0.333</td>
<td>1.408 ± 0.048</td>
<td>± 0.725</td>
</tr>
</tbody>
</table>

$c_l$ = lift coefficient, $c_d$ = drag coefficient, $Re$ = Reynolds number

At the end of the day, you should get something like this

Instantaneous velocity magnitude field
www.wolfdynamics.com/wiki/cylinder_vortex_shedding/movvmag.gif

Instantaneous vorticity magnitude field
www.wolfdynamics.com/wiki/cylinder_vortex_shedding/movvort.gif

Incompressible flow – Reynolds 200
Flow past a cylinder – From laminar to turbulent flow

At the end of the day, you should get something like this

---

**Flow past a cylinder – Reynolds 200**

Incompressible flow – Reynolds 200
Flow past a cylinder – From laminar to turbulent flow

• Let us run this case. Go to the directory:

$PTOFC/101OF/vortex_shedding

• In the case directory, you will find the README.FIRST file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.

• You will also find a few additional files (or scripts) with the extension .sh, namely, run_all.sh, run_mesh.sh, run_sampling.sh, run_solver.sh, and so on. These files can be used to run the case automatically by typing in the terminal, for example, sh run_solver.

• We highly recommend to open the README.FIRST file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM® commands.

• If you are already comfortable with OpenFOAM®, use the automatic scripts to run the cases.
Flow past a cylinder – From laminar to turbulent flow

What are we going to do?

• We will use this case to learn how to use different solvers and utilities.
• Remember, different solvers have different input dictionaries.
• We will learn how to convert the mesh from a third party software.
• We will learn how to use setFields to accelerate the convergence.
• We will learn how to map a solution from a coarse mesh to a fine mesh.
• We will learn how to setup a compressible solver.
• We will learn how to setup a turbulence case.
• We will use gnuplot to plot and compute the mean values of the lift and drag coefficients.
• We will visualize unsteady data.
Let us first convert the mesh from a third-party format (Fluent format).

You will find this tutorial in the directory $PTOFC/101OF/vortex_shedding/c2

In the terminal window type:

1. $> foamCleanTutorials
2. $> fluent3DMeshToFoam ../../../meshes_and_geometries/vortex_shedding/ascii.msh
3. $> checkMesh
4. $> paraFoam

In step 2, we convert the mesh from Fluent format to OpenFOAM® format. Have in mind that the Fluent mesh must be in ascii format.

If we try to open the mesh using paraFoam (step 4), it will crash. Can you tell what is the problem (read the screen)?
Running the case

- To avoid this problem, type in the terminal,

1. $> \text{paraFoam -builtin}

- Basically, the problem is related to the names and type of the patches in the file `boundary` and the boundary conditions \((U, p)\). Notice that OpenFOAM® is telling you what and where is the error.

```
Created temporary 'c2.OpenFOAM'

--> FOAM FATAL IO ERROR:
  patch type 'patch' not constraint type 'empty'  What
  for patch front of field p in file "/home/joegi/my_cases_course/5x/101OF/vortex_shedding/c2/0/p"
  file: /home/joegi/my_cases_course/5x/101OF/vortex_shedding/c2/0/p.boundaryField.front from line 60 to line 60.

  From function Foam::emptyFvPatchField<Type>::emptyFvPatchField(const Foam::fvPatch&, const Foam::DimensionedField<Type, Foam::volMesh>&, const Foam::dictionary&) [with Type = double]
    in file fields/fvPatchFields/constraint/empty/emptyFvPatchField.C at line 80.

FOAM exiting
```
Flow past a cylinder – From laminar to turbulent flow

- Remember, when converting meshes the **name** and **type** of the patches are not always set as you would like, so it is always a good idea to take a look at the file `boundary` and modify it according to your needs.

- Let us modify the `boundary` dictionary file.

- In this case, we would like to setup the following **numerical type** boundary conditions.
Flow past a cylinder – From laminar to turbulent flow

The boundary dictionary file

• This dictionary is located in the `constant/polyMesh` directory.

• This file is automatically created when converting or generating the mesh.

• To get a visual reference of the patches, you can view the mesh with `paraFoam/paraview`.

• The type of the **out** patch is OK.

• The type of the **sym1** patch is OK.

• The type of the **sym2** patch is OK.

• The type of the **in** patch is OK.
• The type of the cylinder patch is OK.
• The type of the back patch is NOT OK. Remember, this is a 2D simulation, therefore the type should be empty.
• The type of the front patch is NOT OK. Remember, this is a 2D simulation, therefore the type should be empty.
• Remember, we assign the numerical type boundary conditions (numerical values), in the field files found in the directory 0.
Flow past a cylinder – From laminar to turbulent flow

- At this point, check that the **name** and **type** of the **base type** boundary conditions and **numerical type** boundary conditions are consistent. If everything is ok, we are ready to go.

- Do not forget to explore the rest of the dictionary files, namely:
  - 0/p (p is defined as relative pressure)
  - 0/U
  - constant/transportProperties
  - system/controlDict
  - system/fvSchemes
  - system/fvSolution

- Reminder:
  - The diameter of the cylinder is 2.0 m.
  - And we are targeting for a \( Re = 200 \).

\[
\nu = \frac{\mu}{\rho} \quad Re = \frac{\rho \times U \times D}{\mu} = \frac{U \times D}{\nu}
\]
Running the case

- You will find this tutorial in the directory $PTOFC/101OF/vortex_shedding/c2
- In the folder c1 you will find the same setup, but to generate the mesh we use blockMesh (the mesh is identical).
- To run this case, in the terminal window type:

1. $> renumberMesh -overwrite
2. $> icoFoam | tee log.icofoam
   $> pyFoamPlotWatcher.py log.icofoam
   You will need to launch this script in a different terminal
3. $> gnuplot scripts0/plot_coeffs
   You will need to launch this script in a different terminal
4. $> paraFoam
Running the case

• In step 1 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers. This is inexpensive (even for large meshes), therefore is highly recommended to always do it.
• In step 2 we run the simulation and save the log file. Notice that we are sending the job to background.
• In step 3 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
• In step 4 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.
• The force coefficients are computed using `functionObjects`.
• After the simulation is over, we use `paraFoam` to visualize the results. Remember to use the VCR Controls to animate the solution.
• In the folder `c1` you will find the same setup, but to generate the mesh we use `blockMesh` (the mesh is identical).
Flow past a cylinder – From laminar to turbulent flow

At this point try to use the following utilities. In the terminal type:

- `$> postProcess -func vorticity -noZero`
  This utility will compute and write the vorticity field. The -noZero option means do not compute the vorticity field for the solution in the directory 0. If you do not add the -noZero option, it will compute and write the vorticity field for all the saved solutions, including 0.

- `$> postprocess -func 'grad(U)' -latestTime`
  This utility will compute and write the velocity gradient or grad(U) in the whole domain (including at the walls). The -latestTime option means compute the velocity gradient only for the last saved solution.

- `$> postprocess -func 'grad(p)'
  This utility will compute and write the pressure gradient or grad(p) in the whole domain (including at the walls).

- `$> postProcess -func 'div(U)'
  This utility will compute and write the divergence of the velocity field or grad(U) in the whole domain (including at the walls). You will need to add the keyword div(U) Gauss linear; in the dictionary fvSchemes.

- `$> foamToVTK -time 50:300`
  This utility will convert the saved solution from OpenFOAM® format to VTK format. The -time 50:300 option means convert the solution to VTK format only for the time directories 50 to 300.

- `$> pisoFoam -postProcess -func CourantNo`
  This utility will compute and write the Courant number. This utility needs to access the solver database for the physical properties and additional quantities, therefore we need to tell what solver we are using. As the solver icoFoam does not accept the option -postProcess, we can use the solver pisoFoam instead. Remember, icoFoam is a fully laminar solver and pisoFoam is a laminar/turbulent solver.

- `$> pisoFoam -postProcess -func wallShearStress`
  This utility will compute and write the wall shear stresses at the walls. As no arguments are given, it will save the wall shear stresses for all time steps.
Non-uniform field initialization

• In the previous case, it took about 150 seconds of simulation time to onset the instability.
• If you are not interested in the initial transient or if you want to speed-up the computation, you can add a perturbation in order to trigger the onset of the instability.
• Let us use the utility `setFields` to initialize a non-uniform flow.
• This case is already setup in the directory

```
$PTOFC/101OF/vortex_sheding/c3
```
Let us run the same case but using a non-uniform field

The `setFieldsDict` dictionary

- This dictionary file is located in the directory `system`.
- In lines 17-20 we set the default value of the velocity vector to be \((0 \ 0 \ 0)\) in the whole domain.
- In lines 24-31, we initialize a rectangular region (box) just behind the cylinder with a velocity vector equal to \((0.98480 \ 0.17364 \ 0)\)
- In this case, `setFields` will look for the dictionary file \(U\) and it will overwrite the original values according to the regions defined in `setFieldsDict`.

Cylinder and boxToCell region
Flow past a cylinder – From laminar to turbulent flow

• Let us run the same case but using a non-uniform field.
• You will find this tutorial in the directory \$PTOFC/101OF/vortex_shedding/c3
• Feel free to use the Fluent mesh or the mesh generated with blockMesh. Hereafter, we will use blockMesh.
• To run this case, in the terminal window type:

1. $> foamCleanTutorials
2. $> blockMesh
3. $> rm -rf 0 > /dev/null 2>&1
4. $> cp -r 0_org/ 0
5. $> setFields
6. $> renumberMesh -overwrite
7. $> icoFoam | log.icofoam
   $> pyFoamPlotWatcher.py log.icofoam
   You will need to launch this script in a different terminal
8. $> gnuplot scripts0/plot_coeffs
   You will need to launch this script in a different terminal
9. $> paraFoam
Running the case – Non-uniform field initialization

- In step 2 we generate the mesh using `blockMesh`. The **name** and **type** of the patches are already set in the dictionary `blockMeshDict` so there is no need to modify the **boundary** file.
- In step 4 we copy the original files to the directory 0. We do this to keep a backup of the original files as the file `0/U` will be overwritten when using `setFields`.
- In step 5 we initialize the solution using `setFields`.
- In step 6 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.
- In step 7 we run the simulation and save the log file. Notice that we are sending the job to background.
- In step 8 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
- In step 9 we use the gnuplot script `scripts0/plot_coeffs` to plot the lift and drag coefficients on-the-fly. Besides monitoring the residuals, it is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.
Flow past a cylinder – From laminar to turbulent flow

Does non-uniform field initialization make a difference?

• A picture is worth a thousand words. No need to tell you yes, even if the solutions are slightly different.

• This bring us to the next subject, for how long should we run the simulation?

No field initialization

With field initialization
For how long should run the simulation?

- This is the difficult part when dealing with unsteady flows.
- Usually you run the simulation until the behavior of a quantity of interest does not oscillate or it becomes periodic.
- In this case we can say that after the 50 seconds mark the solution becomes periodic, therefore there is no need to run up to 350 seconds (unless you want to gather a lot of statistics).
- We can stop the simulation at 150 seconds (or maybe less), and do the average of the quantities between 100 and 150 seconds.
What about the residuals?

- Residuals are telling you a lot, but they are difficult to interpret.
- In this case the fact that the initial residuals are increasing after about 10 seconds, does not mean that the solution is diverging. This is an indication that something is happening (in this case the onset of the instability).
- Remember, the residuals should always drop to the tolerance criteria set in the *fvSolution* dictionary (final residuals). If they do not drop to the desired tolerance, we are talking about unconverged time-steps.
- Things that are not clear from the residuals:
  - For how long should we run the simulation?
  - Is the solution converging to the right value?
To compute the force coefficients we use functionObjects.

Remember, functionObjects are defined at the end of the controlDict dictionary file.

In line 195 we give a name to the functionObject.

In line 208 we define the patch where we want to compute the forces.

In lines 212-213 we define the reference density value.

In line 218 we define the center of rotation (for moments).

In line 219 we define the lift force axis.

In line 220 we define the drag force axis.

In line 221 we define the axis of rotation for moment computation.

In line 223 we give the reference length (for computing the moments)

In line 224 we give the reference area (in this case the frontal area).

The output of this functionObject is saved in the file forceCoeffs.dat located in the directory forceCoeffs_object/0/
Can we compute basic statistics of the force coefficients using gnuplot?

- Yes we can. Enter the gnuplot prompt and type:

1. `gnuplot> stats 'postProcessing/forceCoeffs_object/0/forceCoeffs.dat' u 3`
   This will compute the basic statistics of all the rows in the file forceCoeffs.dat (we are sampling column 3 in the input file)

2. `gnuplot> stats 'postProcessing/forceCoeffs_object/0/forceCoeffs.dat' every ::3000::7000 u 3`
   This will compute the basic statistics of rows 3000 to 7000 in the file forceCoeffs.dat (we are sampling column 3 in the input file)

3. `gnuplot> plot 'postProcessing/forceCoeffs_object/0/forceCoeffs.dat' u 3 w l`
   This will plot column 3 against the row number (iteration number)

4. `gnuplot> exit`
   To exit gnuplot

- Remember the force coefficients information is saved in the file `forceCoeffs.dat` located in the directory `postProcessing/forceCoeffs_object/0`
On the solution accuracy

- At the end of the day we want a solution that is second order accurate.
- We define the discretization schemes (and therefore the accuracy) in the dictionary `fvSchemes`.
- In this case, for time discretization (`ddtSchemes`) we are using the `backward` method.
- For gradient discretization (`gradSchemes`) we are using the `leastSquares` method with slope limiters (`cellLimited`).
- For the discretization of the convective terms (`divSchemes`) we are using `linearUpwindV` interpolation method for the term `div(rho,U)`.
- For the discretization of the Laplacian (`laplacianSchemes` and `snGradSchemes`) we are using the `Gauss linear limited 1` method.
- This method is second order accurate.
We define the solution tolerance and linear solvers in the dictionary *fvSolution*.

To solve the pressure (*p*) we are using the **GAMG** method with an absolute **tolerance** of 1e-6 and a relative tolerance **relTol** of 0.01.

The entry **pFinal** refers to the final correction of the **PISO** loop. It is possible to use a tighter convergence criteria only in the last iteration.

To solve **U** we are using the solver **PBiCG** and the **DILU** preconditioner, with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0 (the solver will stop iterating when it meets any of the conditions).

Solving for the velocity is relative inexpensive, whereas solving for the pressure is expensive.

The **PISO** sub-dictionary contains entries related to the pressure-velocity coupling (in this case the **PISO** method). Hereafter we are doing two **PISO** correctors (**nCorrectors**) and two non-orthogonal corrections (**nNonOrthogonalCorrectors**).
• This case starts from the latest saved solution (**startFrom**).
• In this case as there are no saved solutions, it will start from 0 (**startTime**).
• It will run up to 350 seconds (**endTime**).
• The time step of the simulation is 0.05 seconds (**deltaT**). The time step has been chosen in such a way that the Courant number is less than 1.
• It will write the solution every 1 second (**writeInterval**) of simulation time (**runTime**).
• It will keep all the solution directories (**purgeWrite**).
• It will save the solution in ascii format (**writeFormat**).
• The write precision is 8 digits (**writePrecision**).
• And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.
Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the icoFoam solver.

```plaintext
Time = 350
Courant Number mean: 0.11299953 max: 0.87674198
DILUPBiCG: Solving for Ux, Initial residual = 0.0037946307, Final residual = 4.8324843e-09, No Iterations 3
DILUPBiCG: Solving for Uy, Initial residual = 0.011990022, Final residual = 5.8815028e-09, No Iterations 3
GAMG: Solving for p, Initial residual = 0.022175872, Final residual = 6.2680545e-07, No Iterations 14
GAMG: Solving for p, Initial residual = 0.0033723932, Final residual = 5.8494331e-07, No Iterations 8
GAMG: Solving for p, Initial residual = 0.0010074964, Final residual = 4.4726195e-07, No Iterations 7
time step continuity errors : sum local = 1.9569266e-11, global = 3.471923e-14, cumulative = 2.8708402e-10
GAMG: Solving for p, Initial residual = 0.0023505548, Final residual = 9.922424e-07, No Iterations 8
GAMG: Solving for p, Initial residual = 0.00045248026, Final residual = 7.7250386e-07, No Iterations 6
GAMG: Solving for p, Initial residual = 0.00014664077, Final residual = 4.5825218e-07, No Iterations 5
time step continuity errors : sum local = 2.0062733e-11, global = 1.2592813e-13, cumulative = 2.8695809e-10
ExecutionTime = 746.46 s  ClockTime = 807 s
faceSource inMassFlow output:
  sum(in) of phi = -40
faceSource outMassFlow output:
  sum(out) of phi = 40
fieldAverage fieldAverage output:
  Calculating averages
  Writing average fields
forceCoeffs forceCoeffs_object output:
  Cm    = 0.0043956828
  Cd    = 1.4391786
  Cl    = 0.44532594
  Cl(f) = 0.22705865
  Cl(r) = 0.21826729
fieldMinMax minmaxdomain output:
  min(p) = -0.82758125 at location (2.2845502 0.27072681 1.4608125e-17)
  max(p) = 0.55952746 at location (-1.033408 -0.040619346 0)
  min(U) = (-0.32263726 -0.054404584 -1.8727033e-19) at location (2.4478235 -0.69065656 -2.5551406e-17)
  max(U) = (1.4610304 0.10220218 2.199981e-19) at location (0.43121241 1.5285504 -1.4453535e-17)
```
Let us use a potential solver to find a quick solution

• In this case we are going to use the potential solver `potentialFoam` (remember potential solvers are inviscid, irrotational and incompressible).

• This solver is super fast and it can be used to find a solution to be used as initial conditions (non-uniform field) for an incompressible solver.

• A good initial condition will accelerate and improve the convergence rate.

• This case is already setup in the directory

  `$PTOFC/101OF/vortex_shedding/c4`

• Do not forget to explore the dictionary files.

• The following dictionaries are different
  
  • `system/fvSchemes`
  
  • `system/fvSolution`

Try to spot the differences.
You will find this tutorial in the directory $PTOFC/101OF/vortex_shedding/c4

Feel free to use the Fluent mesh or the mesh generated with blockMesh. In this case we will use blockMesh.

To run this case, in the terminal window type:

1. $> foamCleanTutorials
2. $> blockMesh
3. $> rm -rf 0
4. $> cp -r 0_org 0
5. $> potentialFoam -noFunctionObjects -initialiseUBCs -writep -writePhi
6. $> paraFoam
Flow past a cylinder – From laminar to turbulent flow

Running the case – Let us use a potential solver to find a quick solution

• In step 2 we generate the mesh using blockMesh. The name and type of the patches are already set in the dictionary blockMeshDict so there is no need to modify the boundary file.

• In step 4 we copy the original files to the directory 0. We do this to keep a backup of the original files as they will be overwritten by the solver potentialFoam.

• In step 5 we run the solver. We use the option –noFunctionObjects to avoid conflicts with the functionobjects. The options –writep and –writePhi will write the pressure field and fluxes respectively.

• At this point, if you want to use this solution as initial conditions for an incompressible solver, just copy the files $U$ and $p$ into the start directory of the incompressible case you are looking to run. Have in mind that the meshes need to be the same.

• Be careful with the name and type of the boundary conditions, they should be same between the potential case and incompressible case.
Potential solution

- Using a potential solution as initial conditions is much better than using a uniform flow. It will speed up the solution and it will give you more stability.
- Finding a solution using the potential solver is inexpensive.
Flow past a cylinder – From laminar to turbulent flow

The output screen

• This is the output screen of the `potentialFoam` solver.

• The output of this solver is also a good indication of the sensitivity of the mesh quality to gradients computation. If you see that the number of iterations are dropping iteration after iteration, it means that the mesh is fine.

• If the number of iterations remain stalled, it means that the mesh is sensitive to gradients, so should use non-orthogonal correction.

• In this case we have a good mesh.
Let us map a solution from a coarse mesh to a finer mesh

- It is also possible to map the solution from a coarse mesh to a finer mesh (and all the way around).
- For instance, you can compute a full Navier Stokes solution in a coarse mesh (fast solution), and then map it to a finer mesh.
- Let us map the solution from the potential solver to a finer mesh (if you want you can map the solution obtained using icoFoam). To do this we will use the utility mapFields.
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c6
```
Flow past a cylinder – From laminar to turbulent flow

Running the case – Let us map a solution from a coarse mesh to a finer mesh

• You will find this tutorial in the directory $PTOFC/101OF/vortex_shedding/c6
• To generate the mesh, use blockMesh (remember this mesh is finer).
• To run this case, in the terminal window type:

1. $> foamCleanTutorials
2. $> blockMesh
3. $> rm –rf 0
4. $> cp –r 0_org 0
5. $> mapfields ../c4 –consistent –noFunctionObjects –mapMethod cellPointInterpolate –sourceTime 0
6. $> paraFoam
Flow past a cylinder – From laminar to turbulent flow

Running the case – Let us map a solution from a coarse mesh to a finer mesh

- In step 2 we generate a finer mesh using `blockMesh`. The name and type of the patches are already set in the dictionary `blockMeshDict` so there is no need to modify the boundary file.

- In step 4 we copy the original files to the directory 0. We do this to keep a backup of the original files as they will be overwritten by the utility `mapFields`.

- In step 5 we use the utility `mapFields` with the following options:
  - We copy the solution from the directory `../c4`
  - The options `-consistent` is used when the domains and BCs are the same.
  - The option `-noFunctionObjects` is used to avoid conflicts with the functionObjects.
  - The option `-mapMethod cellPointInterpolate` defines the interpolation method.
  - The option `-sourceTime 0` defines the time from which we want to interpolate the solution.
Flow past a cylinder – From laminar to turbulent flow

The meshes and the mapped fields

Coarse mesh

Fine mesh

mapFields
Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the `mapFields` utility.
- The utility `mapFields`, will try to interpolate all fields in the source directory.
- You can control the target time via the `startFrom` and `startTime` keywords in the `controlDict` dictionary file.

Source: "." "c4"  
Target: "/home/joegi/my_cases_course/5x/101OF/vortex_shedding" "c6"

Mapping method: `cellPointInterpolate`

Create databases as time

- Source time: 0
- Target time: 0

Create meshes

- Source mesh size: 9200  Target mesh size: 36800

Consistently creating and mapping fields for time 0

- interpolating Phi
- interpolating p
- interpolating U

End
Setting a turbulent case

• So far we have used laminar incompressible solvers.
• Let us do a turbulent simulation.
• When doing turbulent simulations, we need to choose the turbulence model, define the boundary and initial conditions for the turbulent quantities, and modify the `fvSchemes` and `fvSolution` dictionaries to take account for the new variables we are solving (the transported turbulent quantities).
• This case is already setup in the directory

`$PTOFC/101OF/vortex_sheding/c14`
Flow past a cylinder – From laminar to turbulent flow

- The following dictionaries remain unchanged
  - `system/blockMeshDict`
  - `constant/polyMesh/boundary`
  - `0/p`
  - `0/U`

- The following dictionaries need to be adapted for the turbulence case
  - `constant/transportProperties`
  - `system/controlDict`
  - `system/fvSchemes`
  - `system/fvSolution`

- The following dictionaries need to be adapted for the turbulence case
  - `constant/turbulenceProperties`
Flow past a cylinder – From laminar to turbulent flow

- The `transportProperties` dictionary file
  - This dictionary file is located in the directory `constant`.
  - In this file we set the transport model and the kinematic viscosity (`nu`).

```plaintext
16 transportModel Newtonian;
17
19 nu              nu [ 0 2 -1 0 0 0 0 ] 0.0002;
```

- Reminder:
  - The diameter of the cylinder is 2.0 m.
  - And we are targeting for a $Re = 10000$.

\[
\nu = \frac{\mu}{\rho} \quad Re = \frac{\rho \times U \times D}{\mu} = \frac{U \times D}{\nu}
\]
The `turbulenceProperties` dictionary file

- This dictionary file is located in the directory `constant`.
- In this dictionary file we select what model we would like to use (laminar or turbulent).
- In this case we are interested in modeling turbulence, therefore the dictionary is as follows

```
17  simulationType  RAS;  ← RANS type simulation
18
19  RAS  ← RANS sub-dictionary
20  {
21       RASModel        kOmegaSST;  ← RANS model to use
22  
23       turbulence      on;  ← Turn on/off turbulence. Runtime modifiable
24  
25       printCoeffs     on;  ← Print coefficients at the beginning
26  }
```

- If you want to know the models available use the banana method.
This case will start from the last saved solution (*startFrom*). If there is no solution, the case will start from time 0 (*startTime*).

It will run up to 500 seconds (*endTime*).

The initial time step of the simulation is 0.001 seconds (*deltaT*).

It will write the solution every 1 second (*writeInterval*) of simulation time (*runTime*).

It will keep all the solution directories (*purgeWrite*).

It will save the solution in ascii format (*writeFormat*).

The write precision is 8 digits (*writePrecision*).

And as the option *runTimeModifiable* is on, we can modify all these entries while we are running the simulation.

In line 64 we turn on the option *adjustTimeStep*. This option will automatically adjust the time step to achieve the maximum desired courant number *maxCo* (line 66).

We also set a maximum time step *maxDeltaT* in line 67.

Remember, the first time step of the simulation is done using the value set in line 28 and then it is automatically scaled to achieve the desired maximum values (lines 66-67).

The feature *adjustTimeStep* is only present in the PIMPLE family solvers, but it can be added to any solver by modifying the source code.
The `fvSchemes` dictionary

- In this case, for time discretization (`ddtSchemes`) we are using the blended `CrankNicolson` method. The blending coefficient goes from 0 to 1, where 0 is equivalent to the `Euler` method and 1 is a pure `Crank Nicolson`.

- For gradient discretization (`gradSchemes`) we are using as default option the `leastSquares` method. For `grad(U)` we are using `Gauss linear` with slope limiters (`cellLimited`). You can define different methods for every term in the governing equations, for example, you can define a different method for `grad(p)`.

- For the discretization of the convective terms (`divSchemes`) we are using `linearUpwindV` interpolation method with slope limiters for the term `div(phi,U)`.

- For the terms `div(phi,k)` and `div(phi,omega)` we are using `linearUpwind` interpolation method with no slope limiters. These terms are related to the turbulence modeling.

- For the term `div((nuEff*dev2(T(grad(U))))))` we are using `linear` interpolation (this term is related to turbulence modeling).

- For the discretization of the Laplacian (`laplacianSchemes` and `snGradSchemes`) we are using the `Gauss linear limited 1` method.

- To compute the distance to the wall and normals to the wall, we use the method `meshWave`. This only applies when using wall functions (turbulence modeling).

- This method is second order accurate.
The `fvSolution` dictionary

- To solve the pressure ($p$) we are using the **GAMG** method, with an absolute **tolerance** of $1e-6$ and a relative tolerance **relTol** of 0.001. Notice that we are fixing the number of minimum iterations (**minIter**).

- To solve the final pressure correction ($pFinal$) we are using the **PCG** method with the **DIC** preconditioner, with an absolute **tolerance** of $1e-6$ and a relative tolerance **relTol** of 0.

- Notice that we can use different methods between $p$ and $pFinal$. In this case we are using a tighter tolerance for the last iteration.

- We are also fixing the number of minimum iterations (**minIter**). This entry is optional.

- To solve $U$ we are using the solver **PBiCGStab** with the **DILU** preconditioner, an absolute **tolerance** of $1e-8$ and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).
The `fvSolution` dictionary

- To solve `UFinal` we are using the solver `PBiCGStab` with an absolute tolerance of 1e-8 and a relative tolerance `relTol` of 0. Notice that we are fixing the number of minimum iterations (`minIter`).

- To solve `omega` and `omegaFinal` we are using the solver `PBiCGStab` with an absolute tolerance of 1e-8 and a relative tolerance `relTol` of 0. Notice that we are fixing the number of minimum iterations (`minIter`).

- To solve `k` we are using the solver `PBiCGStab` with an absolute tolerance of 1e-8 and a relative tolerance `relTol` of 0. Notice that we are fixing the number of minimum iterations (`minIter`).
The `fvSolution` dictionary

- To solve `kFinal` we are using the solver `PBiCGStab` with an absolute tolerance of `1e-8` and a relative tolerance `relTol` of 0. Notice that we are fixing the number of minimum iterations (`minIter`).

- In lines 123-133 we setup the entries related to the pressure-velocity coupling method used (`PIMPLE` in this case). Setting the keyword `nOuterCorrectors` to 1 is equivalent to running using the `PISO` method.

- To gain more stability we are using 1 outer correctors (`nOuterCorrectors`), 3 inner correctors or `PISO` correctors (`nCorrectors`), and 1 correction due to non-orthogonality (`nNonOrthogonalCorrectors`).

- Remember, adding corrections increase the computational cost.

- In lines 135-147 we setup the under relaxation factors used during the outer corrections (pseudo transient iterations). If you are working in `PISO` mode (only one outer correction or `nOuterCorrectors`), these values are ignored.
The following dictionaries are new

- $0/k$
- $0/omega$
- $0/nut$

These are the field variables related to the closure equations of the turbulent model.

As we are going to use the $\kappa - \omega \; SST$ model we need to define the initial conditions and boundaries conditions.

To define the IC/BC we will use the free stream values of $\kappa$ and $\omega$

In the following site, you can find a lot information about choosing initial and boundary conditions for the different turbulence models:

- https://turbmodels.larc.nasa.gov/
Flow past a cylinder – From laminar to turbulent flow

$\kappa - \omega \, SST$  Turbulence model free-stream boundary conditions

- The initial value for the turbulent kinetic energy $\kappa$ can be found as follows

$$\kappa = \frac{3}{2} (UI)^2$$

- The initial value for the specific kinetic energy $\omega$ can be found as follows

$$\omega = \frac{\rho \kappa \mu_t}{\mu}$$

- Where $\frac{\mu_t}{\mu}$ is the viscosity ratio and $I = \frac{u'}{u}$ is the turbulence intensity.

- If you are working with external aerodynamics or virtual wind tunnels, you can use the following reference values for the turbulence intensity and the viscosity ratio. They work most of the times, but it is a good idea to have some experimental data or a better initial estimate.

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$</td>
<td>1.0 %</td>
<td>5.0 %</td>
<td>10.0 %</td>
</tr>
<tr>
<td>$\mu_t/\mu$</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>
We are using uniform initial conditions (line 19).

For the in patch we are using a fixedValue boundary condition.

For the out patch we are using an inletOutlet boundary condition (this boundary condition avoids backflow).

For the cylinder patch (which is base type wall), we are using the kqRWallFunction boundary condition. This is a wall function, we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of base type wall.

The rest of the patches are constrained.

FYI, the inlet velocity is 1 and the turbulence intensity is equal to 1%.
We are using uniform initial conditions (line 19).

For the in patch we are using a fixedValue boundary condition.

For the out patch we are using an inletOutlet boundary condition (this boundary condition avoids backflow).

For the cylinder patch (which is base type wall), we are using the omegaWallFunction boundary condition. This is a wall function, we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of base type wall.

The rest of the patches are constrained.

FYI, the inlet velocity is 1 and the eddy viscosity ratio is equal to 10.
The file 0/nut

- We are using uniform initial conditions (line 19).
- For the in patch we are using the calculated boundary condition (nut is computed from kappa and omega).
- For the out patch we are using the calculated boundary condition (nut is computed from kappa and omega).
- For the cylinder patch (which is base type wall), we are using the autkWallFunction boundary condition. This is a wall function, we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of base type wall.
- The rest of the patches are constrained.
- Remember, the turbulent viscosity $\nu_t$ (nut) is equal to $\frac{\kappa}{\omega}$.}

Flow past a cylinder – From laminar to turbulent flow
Flow past a cylinder – From laminar to turbulent flow

Running the case – Setting a turbulent case

• You will find this tutorial in the directory `$PTOFC/101OF/vortex_shedding/c14`
• Feel free to use the Fluent mesh or the mesh generated with `blockMesh`. In this case we will use `blockMesh`.
• To run this case, in the terminal window type:

1. `$> foamCleanTutorials`
2. `$> blockMesh`
3. `$> renumberMesh -overwrite`
4. `$> pimpleFoam | log`
   You will need to launch this script in a different terminal
5. `$> pyFoamPlotWatcher.py log`
   You will need to launch this script in a different terminal
6. `$> gnuplot scripts0/plot_coeffs`
   You will need to launch this script in a different terminal
7. `$> pimpleFoam -postprocess -func yPlus -latestTime -noFunctionObjects`
8. `$> paraFoam`
In step 3 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.

In step 4 we run the simulation and save the log file. Notice that we are sending the job to background.

In step 5 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.

In step 6 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, it is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.

In step 7 we use the utility `postProcess` to compute the $y^+$ value of each saved solution (we are going to talk about $y^+$ when we deal with turbulence modeling).
Flow past a cylinder – From laminar to turbulent flow

Courant Number mean: 0.088931706 max: 0.90251464
deltaT = 0.040145538
Time = 499.97
PIMPLE: iteration 1
DILUPBiCG: Solving for Ux, Initial residual = 0.0028528538, Final residual = 9.5497298e-11, No Iterations 3
DILUPBiCG: Solving for Uy, Initial residual = 0.0068876991, Final residual = 7.000938e-10, No Iterations 7
GAMG: Solving for P, Initial residual = 0.25644342, Final residual = 0.00022585963, No Iterations 7
GAMG: Solving for P, Initial residual = 0.0073871161, Final residual = 5.2798526e-06, No Iterations 8
time step continuity errors: sum local = 3.2664019e-10, global = -1.3568363e-12, cumulative = -9.8446438e-08
GAMG: Solving for P, Initial residual = 0.16889316, Final residual = 0.00014947209, No Iterations 7
time step continuity errors: sum local = 2.2950163e-10, global = -8.0710768e-13, cumulative = -9.8447245e-08
PIMPLE: iteration 2
DILUPBiCG: Solving for Ux, Initial residual = 0.0013482181, Final residual = 4.1395468e-10, No Iterations 3
DILUPBiCG: Solving for Uy, Initial residual = 0.0032433196, Final residual = 3.3969121e-09, No Iterations 3
GAMG: Solving for P, Initial residual = 0.10067317, Final residual = 8.9325549e-05, No Iterations 7
GAMG: Solving for P, Initial residual = 0.0042844521, Final residual = 3.0190597e-06, No Iterations 8
time step continuity errors: sum local = 1.735023e-10, global = -2.0653335e-13, cumulative = -9.8447452e-08
GAMG: Solving for P, Initial residual = 0.0050231165, Final residual = 3.7123156e-06, No Iterations 8
time step continuity errors: sum local = 5.4344408e-11, global = 4.0060595e-12, cumulative = -9.8443445e-08
DICPCG: Solving for P, Initial residual = 0.00031459519, Final residual = 8.9325549e-07, No Iterations 36
bounding k, min: -3.6865398e-05 max: 0.055400108 average: 0.0015914926
ExecutionTime = 1689.51 s ClockTime = 1704 s
fieldAverage fieldAverage output:
Calculating averages
forceCoeffs forceCoeffs_object output:
Cm = 0.0023218797
Cd = 1.1832452
Cl = -1.3927646
Cl(f) = -0.69406044
Cl(r) = -0.6987042
fieldMinMax minmaxdomain output:
min(p) = -1.5466372 at location (-0.040619337 -1.0334080)
max(p) = 0.54524589 at location (-1.0334080 0.040619337 1.4015759e-17)
min(U) = (0.94205232 -1.0407426 -5.0319219e-19) at location (-0.70200781 -0.75945224 -1.3630525e-17)
max(U) = (1.8458167 0.004763607 4.473279e-19) at location (-0.12989625 -1.0971865 2.4694467e-17)
min(k) = 1e-15 at location (1.0972618 1.3921931 -2.3239989e-17)
max(k) = 0.055400108 at location (2.1464795 0.42727634 0)
min(omega) = 0.2355751 at location (29.403674 19.3304 0)
max(omega) = 21.477072 at location (1.0334080 0.040619337 1.3245285e-17)

Message letting you know that the variable is becoming unbounded
kappa and omega residuals
minimum and maximum values
Force coefficients

Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the yPlus utility.

Time = 500.01
Reading field U

Reading/calculating face flux field phi

Selecting incompressible transport model Newtonian
Selecting RAS turbulence model kOmegaSST

kOmegaSSTCoeffs

{ }

alphaK1 0.85;
alphaK2 1;
alphaOmega1 0.5;
alphaOmega2 0.856;
gamma1 0.55555556;
gamma2 0.44;
bet1 0.075;
beta2 0.0828;
betaStar 0.09;
al 0.31;
b1 1;
c1 10;
F3 false;

Patch 4 named cylinder y+ : min: 0.94230389 max: 12.696632 average: 7.3497345

Writing yPlus to field yPlus

The output screen
Using a compressible solver

- So far we have only used incompressible solvers.
- Let us use the compressible solver `rhoPimpleFoam`, which is a transient solver for laminar or turbulent flow of compressible fluids for HVAC and similar applications. Uses the flexible PIMPLE (PISO-SIMPLE) solution for time-resolved and pseudo-transient simulations.

- When working with compressible solver we need to define the thermodynamical properties of the working fluid and the temperature field (we are also solving the energy equation)
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c24
```
Flow past a cylinder – From laminar to turbulent flow

- The following dictionaries remain unchanged
  - `system/blockMeshDict`
  - `constant/polyMesh/boundary`

- Reminder:
  - The diameter of the cylinder is 0.002 m.
  - The working fluid is air at 20°C Celsius and at a sea level.
  - Isothermal flow.
  - And we are targeting for a \( Re = 200 \).

\[
\nu = \frac{\mu}{\rho} \quad Re = \frac{\rho \times U \times D}{\mu} = \frac{U \times D}{\nu}
\]
Flow past a cylinder – From laminar to turbulent flow

The constant directory

• In this directory, we will find the following compulsory dictionary files:
  
  • `thermophysicalProperties`
  • `turbulenceProperties`  
  
  • `thermophysicalProperties` contains the definition of the physical properties of the working fluid.
  • `turbulenceProperties` contains the definition of the turbulence model to use.
Flow past a cylinder – From laminar to turbulent flow

The `thermophysicalProperties` dictionary file

- This dictionary file is located in the directory `constant`. Thermophysical models are concerned with energy, heat and physical properties.
- In the sub-dictionary `thermoType` (lines 18-27), we define the thermophysical models.
- The `transport` modeling concerns evaluating dynamic viscosity (line 22). In this case the viscosity is constant.
- The thermodynamic models (`thermo`) are concerned with evaluating the specific heat $C_p$ (line 23). In this case $C_p$ is constant.
- The `equationOfState` keyword (line 24) concerns to the equation of state of the working fluid. In this case

\[ \rho = \frac{p}{RT} \]

- The form of the energy equation to be used in the solution is specified in line 26 (`energy`). In this case we are using enthalpy (`sensibleEnthalpy`).

```plaintext
18  thermoType
19  {
20    type            hePsiThermo;
21    mixture         pureMixture;
22    transport       const;
23    thermo          hConst;
24    equationOfState perfectGas;
25    specie          specie;
26    energy          sensibleEnthalpy;
27  }
28
29  mixture
30  {
31    specie
32    {
33      nMoles        1;
34      molWeight    28.9;
35    }
36    thermodynamics
37    {
38      Cp            1005;
39      Hf           0;
40    }
41    transport
42    {
43      mu           1.84e-05;
44      Pr           0.713;
45    }
46  }
```
• In the sub-dictionary `mixture` (lines 29-46), we define the thermophysical properties of the working fluid.

• In this case, we are defining the properties for air at 20°C Celsius and at a sea level.

```plaintext
thermoType
{
  type hePsiThermo;
  mixture pureMixture;
  transport const;
  thermo hConst;
  equationOfState perfectGas;
  specie specie;
  energy sensibleEnthalpy;
}

mixture
{
  specie
  {
    nMoles 1;
    molWeight 28.9;
  }
  thermodynamics
  {
    Cp 1005;
    Hf 0;
  }
  transport
  {
    mu 1.84e-05;
    Pr 0.713;
  }
}
```
Flow past a cylinder – From laminar to turbulent flow

The `turbulenceProperties` dictionary file

- In this dictionary file we select what model we would like to use (laminar or turbulent).
- This dictionary is compulsory.
- As we do not want to model turbulence, the dictionary is defined as follows,

```plaintext
17 simulationType laminar;
```
The 0 directory

- In this directory, we will find the dictionary files that contain the boundary and initial conditions for all the primitive variables.
- As we are solving the compressible laminar Navier-Stokes equations, we will find the following field files:
  - $P$ (pressure)
  - $T$ (temperature)
  - $U$ (velocity field)
• This file contains the boundary and initial conditions for the scalar field pressure \( p \). We are working with absolute pressure.

• Contrary to incompressible flows where we defined relative pressure, this is the absolute pressure.

• Also, pay attention to the units (line 17). The pressure is defined in Pascal.

• We are using uniform initial conditions (line 19).

• For the \texttt{in} patch we are using a \texttt{zeroGradient} boundary condition.

• For the \texttt{outlet} patch we are using a \texttt{fixedValue} boundary condition.

• For the \texttt{cylinder} patch we are using a \texttt{zeroGradient} boundary condition.

• The rest of the patches are constrained.
This file contains the boundary and initial conditions for the scalar field temperature ($T$).

Also, pay attention to the units (line 17). The temperature is defined in Kelvin.

We are using uniform initial conditions (line 19).

For the **in** patch we are using a **fixedValue** boundary condition.

For the **out** patch we are using a **inletOutlet** boundary condition (in case of backflow).

For the **cylinder** patch we are using a **zeroGradient** boundary condition.

The rest of the patches are constrained.
Flow past a cylinder – From laminar to turbulent flow

The file $0/U$

- This file contains the boundary and initial conditions for the dimensional vector field $\mathbf{U}$.
- We are using uniform initial conditions and the numerical value is $(1.5 \, 0 \, 0)$ (keyword `internalField` in line 19).
- For the `in` patch we are using a `fixedValue` boundary condition.
- For the `out` patch we are using a `inletOutlet` boundary condition (in case of backflow).
- For the `cylinder` patch we are using a `zeroGradient` boundary condition.
- The rest of the patches are constrained.

```plaintext
dimensions [0 1 -1 0 0 0];
internalField uniform (1.5 0 0);
boundaryField {
in {
  type           fixedValue;
  value          uniform (1.5 0 0);
}
out {
  type           inletOutlet;
  phi            phi;
  inletValue     uniform (0 0 0);
  value          uniform (0 0 0);
}
cylinder {
  type           fixedValue;
  value          uniform (0 0 0);
}
sym1 {
  type           symmetryPlane;
}
sym2 {
  type           symmetryPlane;
}
back {
  type           empty;
}
front {
  type           empty;
}
}
```

This file contains the boundary and initial conditions for the dimensional vector field $\mathbf{U}$.

We are using uniform initial conditions and the numerical value is $(1.5 \, 0 \, 0)$ (keyword `internalField` in line 19).

For the `in` patch we are using a `fixedValue` boundary condition.

For the `out` patch we are using a `inletOutlet` boundary condition (in case of backflow).

For the `cylinder` patch we are using a `zeroGradient` boundary condition.

The rest of the patches are constrained.
The *system* directory

- The *system* directory consists of the following compulsory dictionary files:
  - *controlDict*
  - *fvSchemes*
  - *fvSolution*

- *controlDict* contains general instructions on how to run the case.
- *fvSchemes* contains instructions for the discretization schemes that will be used for the different terms in the equations.
- *fvSolution* contains instructions on how to solve each discretized linear equation system.

Flow past a cylinder – From laminar to turbulent flow
The `controlDict` dictionary

- This case will start from the last saved solution (`startFrom`). If there is no solution, the case will start from time 0 (`startTime`).
- It will run up to 0.3 seconds (`endTime`).
- The initial time step of the simulation is 0.00001 seconds (`deltaT`).
- It will write the solution every 0.0025 seconds (`writeInterval`) of simulation time (`adjustableRunTime`). The option `adjustableRunTime` will adjust the time-step to save the solution at the precise intervals. This may add some oscillations in the solution as the CFL is changing.
- It will keep all the solution directories (`purgeWrite`).
- It will save the solution in ascii format (`writeFormat`).
- And as the option `runTimeModifiable` is on, we can modify all these entries while we are running the simulation.
- In line 49 we turn on the option `adjustTimeStep`. This option will automatically adjust the time step to achieve the maximum desired courant number (line 50).
- We also set a maximum time step in line 51.
- Remember, the first time step of the simulation is done using the value set in line 28 and then it is automatically scaled to achieve the desired maximum values (lines 66-67).
- The feature `adjustTimeStep` is only present in the PIMPLE family solvers, but it can be added to any solver by modifying the source code.
As usual, at the bottom of the controlDict dictionary file we define the functionObjects (lines 55-236).

Of special interest is the functionObject forceCoeffs_object.

As we changed the domain dimensions and the inlet velocity we need to update the reference values (lines 204-206).

It is also important to update the reference density (line 195).
• In this case, for time discretization (**ddtSchemes**) we are using the **Euler** method.

• For gradient discretization (**gradSchemes**) we are using the **leastSquares** method.

• For the discretization of the convective terms (**divSchemes**) we are using **linearUpwind** interpolation with no slope limiters for the term \( \text{div}(\phi, U) \).

• For the terms \( \text{div}(\phi, K) \) (kinetic energy) and \( \text{div}(\phi, h) \) (enthalpy) we are using **linear** interpolation method with no slope limiters.

• For the term \( \text{div}(((\rho*\nu_{\text{Eff}})*\text{dev2}(\text{T}(\text{grad}(U)))))) \) we are using **linear** interpolation (this term is related to the turbulence modeling).

• For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear limited 1** method.

• This method is second order accurate.
To solve the pressure ($p$) we are using the **PCG** method with an absolute **tolerance** of $1e-6$ and a relative tolerance **relTol** of 0.01.

The entry **pFinal** refers to the final correction of the **PISO** loop. Notice that we are using macro expansion ($p$) to copy the entries from the sub-dictionary **p**.

To solve **U** and **UFinal** (**U.***) we are using the solver **PBiCGStab** with an absolute **tolerance** of $1e-8$ and a relative tolerance **relTol** of 0.

To solve **hFinal** (enthalpy) we are using the solver **PBiCGStab** with an absolute **tolerance** of $1e-8$ and a relative tolerance **relTol** of 0.

To solve **rho** and **rhoFinal** (**rho.***) we are using the **diagonal** solver (remember rho is found from the equation of state, so this is a back-substitution).

FYI, solving for the velocity is relative inexpensive, whereas solving for the pressure is expensive.

Be careful with the enthalpy, it might cause oscillations.
The `fvSolution` dictionary

- The PIMPLE sub-dictionary contains entries related to the pressure-velocity coupling (in this case the PIMPLE method).

- Setting the keyword `nOuterCorrectors` to 1 is equivalent to running using the PISO method.

- Hereafter we are doing 2 PISO correctors (`nCorrectors`) and 1 non-orthogonal corrections (`nNonOrthogonalCorrectors`).

- In lines 95-96 we set the minimum and maximum physical values of rho (density).

- If we increase the number of `nCorrectors` and `nNonOrthogonalCorrectors` we gain more stability but at a higher computational cost.

- The choice of the number of corrections is driven by the quality of the mesh and the physics involve.

- You need to do at least one PISO loop (`nCorrectors`).
Running the case – Using a compressible solver

- You will find this tutorial in the directory $PTOFC/101OF/vortex_shedding/c24
- Feel free to use the Fluent mesh or the mesh generated with blockMesh. In this case we will use blockMesh.
- To run this case, in the terminal window type:

1. $> foamCleanTutorials
2. $> blockMesh
3. $> transformPoints -scale ‘(0.001 0.001 0.001)’
4. $> renumberMesh -overwrite
5. $> rhoPimpleFoam | tee log
   $> pyFoamPlotWatcher.py log
   You will need to launch this script in a different terminal
6. $> gnuplot scripts0/plot_coeffs
   You will need to launch this script in a different terminal
7. $> rhoPimpleFoam -postProcess -func MachNo
8. $> paraFoam
In step 3 we scale the mesh.

In step 4 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.

In step 5 we run the simulation and save the log file. Notice that we are sending the job to background.

In step 6 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.

In step 7 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, it is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.

In step 8 we use the utility `MachNo` to compute the Mach number.
**Flow past a cylinder – From laminar to turbulent flow**

**rhoPimpleFoam output screen**

- **Courant Number mean**: 0.1280224248, max: 0.9885863338
- **deltaT**: $3.816512052 \times 10^{-5}$
- **Time**: 0.3

**Solving for density (rho)**
- PIMPLE: iteration 1
  - DILUPBiCG: Solving for $U_x$, Initial residual = 0.003594731129, Final residual = $1.22326662 \times 10^{-11}$, No Iterations 5
  - DILUPBiCG: Solving for $U_y$, Initial residual = 0.01228951539, Final residual = $2.58323646 \times 10^{-9}$, No Iterations 4
  - DIPCG: Solving for $h$, Initial residual = 0.01228951539, Final residual = $8.797612158 \times 10^{-7}$, No Iterations 77
  - DIPCG: Solving for $p$, Initial residual = 0.01967621449, Final residual = $2.36774797 \times 10^{-9}$, No Iterations 0

**h residuals**
- diagonal: Solving for $rho$, Initial residual = 0, Final residual = 0, No Iterations 0

**pFinal**
- time step continuity errors: sum local = $6.835363016 \times 10^{-11}$, global = $4.328592697 \times 10^{-12}$, cumulative = $2.366774797 \times 10^{-9}$

**Max/min density values**
- rho max/min: $1.201420286 \times 10^{-11}$

**ExecutionTime**: 480.88 s  **ClockTime**: 490 s

**faceSource inMassFlow output:**
- sum(in) of phi = $-7.208447027 \times 10^{-5}$

**faceSource outMassFlow output:**
- sum(out) of phi = $7.208444452 \times 10^{-5}$

**fieldAverage fieldAverage output:**
- Calculating averages
- Writing average fields

**Force coefficients**
- $C_m$ = -0.001269886395
- $C_d$ = 1.419350733
- $C_l$ = 0.6247248606
- $C_l(f)$ = 0.3110925439
- $C_l(r)$ = 0.3136323167

**fieldMinMax minmaxdomain output:**
- min(p) = 101322.7878 at location (-5.556854517e-05 0.001412635233 0)
- max(p) = 101326.4972 at location (-0.001215826043 0.001027092827 0)
- min(U) = (-0.526856427 -0.09305459972 -8.110485132e-25) at location (0.02039092041 -0.004058872656 -3.893823418e-20)
- max(U) = (2.184751599 0.2867627526 4.83091257e-25) at location (0.001663574444 0.00140596295 3.016083257e-20)
- min(T) = 293.1487423 at location (-5.556854517e-05 0.001412635233 0)
- max(T) = 293.1509903 at location (-0.001215826043 0.001027092827 -4.627394552e-05)

**Minimum and maximum values**

---

**Flow past a cylinder – From laminar to turbulent flow**

- Courant Number mean: 0.1280224248, max: 0.9885863338
- deltaT = $3.816512052 \times 10^{-5}$
- Time = 0.3

- Solving for density (rho)
  - PIMPLE: iteration 1
    - DILUPBiCG: Solving for $U_x$, Initial residual = 0.003594731129, Final residual = $1.22326662 \times 10^{-11}$, No Iterations 5
    - DILUPBiCG: Solving for $U_y$, Initial residual = 0.01228951539, Final residual = $2.58323646 \times 10^{-9}$, No Iterations 4
    - DIPCG: Solving for $h$, Initial residual = 0.01228951539, Final residual = $8.797612158 \times 10^{-7}$, No Iterations 77
    - DIPCG: Solving for $p$, Initial residual = 0.01967621449, Final residual = $2.36774797 \times 10^{-9}$, No Iterations 0

- Solving for density (rhoFinal)
  - diagonal: Solving for $rho$, Initial residual = 0, Final residual = 0, No Iterations 0

**h residuals**
- time step continuity errors: sum local = $6.835363016 \times 10^{-11}$, global = $4.328592697 \times 10^{-12}$, cumulative = $2.366774797 \times 10^{-9}$

**Max/min density values**
- rho max/min: $1.201420286 \times 10^{-11}$

**ExecutionTime**: 480.88 s  **ClockTime**: 490 s

**faceSource inMassFlow output:**
- sum(in) of phi = $-7.208447027 \times 10^{-5}$

**faceSource outMassFlow output:**
- sum(out) of phi = $7.208444452 \times 10^{-5}$

**fieldAverage fieldAverage output:**
- Calculating averages
- Writing average fields

**Force coefficients**
- $C_m$ = -0.001269886395
- $C_d$ = 1.419350733
- $C_l$ = 0.6247248606
- $C_l(f)$ = 0.3110925439
- $C_l(r)$ = 0.3136323167

**fieldMinMax minmaxdomain output:**
- min(p) = 101322.7878 at location (-0.001215826043 0.001027092827 0)
- max(p) = 101326.4972 at location (-0.001027092827 0.001215826043 0)
- min(U) = (-0.526856427 -0.09305459972 -8.110485132e-25) at location (0.001201420286 0.001027092827 0)
- max(U) = (2.184751599 0.2867627526 4.83091257e-25) at location (0.001201420286 0.001027092827 0)
- min(T) = 293.1487423 at location (-5.556854517e-05 0.001412635233 0)
- max(T) = 293.1509903 at location (-0.001215826043 0.001027092827 -4.627394552e-05)
In the directory \$PTOFC/101OF/vortex_shedding, you will find 28 variations of the cylinder case involving different solvers and models. Feel free to explore all them.

This is what you will find in each directory,

- c7 = blockMesh – pimpleFoam – Re = 200 – No turbulent model.
- c8 = blockMesh – pisoFoam – Re = 200 – No turbulent model.
- c10 = blockMesh – simpleFoam – Re = 200 – No turbulent model.
- c11 = blockMesh – simpleFoam – Re = 40 – No turbulent model.
- c12 = blockMesh – pisoFoam – Re = 40 – No turbulent model.
- c14 = blockMesh – pimpleFoam – Re = 10000 – K-Omega SST turbulent model with wall functions.
- c15 = blockMesh – pimpleFoam – Re = 100000 – K-Omega SST turbulent model with wall functions.
Flow past a cylinder – From laminar to turbulent flow

- This is what you will find in each directory,
  - c16 = blockMesh – simpleFoam – Re = 100000 – K-Omega SST turbulent model with no wall functions.
  - c17 = blockMesh – simpleFoam – Re = 100000 – K-Omega SST turbulent model with wall functions.
  - c18 = blockMesh – pisoFoam – Re = 100000, LES Smagorinsky turbulent model.
  - c19 = blockMesh – pimpleFoam – Re = 1000000 – Spalart Allmaras turbulent model with no wall functions.
  - c20 = blockMesh – sonicFoam – Mach = 2.0 – Compressible – Laminar.
  - c21 = blockMesh – sonicFoam – Mach = 2.0 – Compressible – K-Omega SST turbulent model with wall functions.
  - c22 = blockMesh – pimpleFoam – Re = 200 – No turbulent model – Source terms (momentum)
  - c23 = blockMesh – pimpleFoam – Re = 200 – No turbulent model – Source terms (scalar transport)
  - c24 = blockMesh – rhoPimpleFoam – Re = 200 – Laminar, isothermal
  - c25 = blockMesh – rhoPimpleFoam – Re = 20000 – Turbulent, compressible
  - c28 = blockMesh – interFoam – Laminar, multiphase, free surface.
  - c29 = blockMesh – pimpleFoam laminar with source terms and AMR.