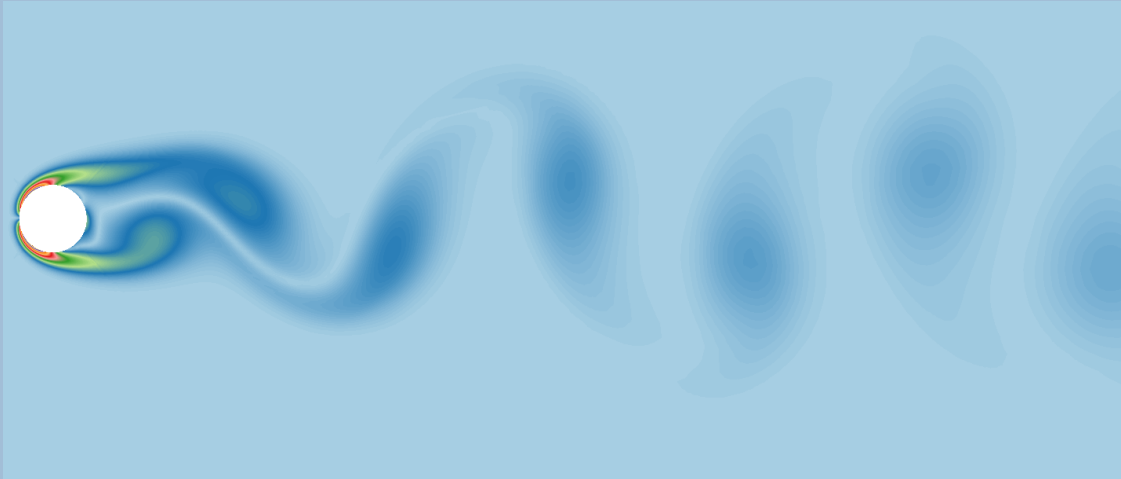


# Exploring OpenFOAM®



## Comflics

Sadiq Huq      Shuaib Ahmed Idrees  
Mehran Saeedi   Ramakrishna Nanjundiah

Beta Release, August 2014.

[www.comflics.de](http://www.comflics.de)

# 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.

DRAFT

# Preface

The response to OpenFOAM® tutorials in Youtube has been overwhelming. The videos were made to give a quick start to using OpenFOAM®. Continuing our efforts at making OpenFOAM® easy to understand, we now bring to you an ebook. Each chapter will demonstrate a key topic in Computational Fluid Dynamics, the simulation cases used in this book will also be made available to you. We encourage you to tweak the simulations, experiment new possibilities and whenever possible share your findings in a public domain.

You can download the files used in this ebook from <https://github.com/Comffics/comfficsTutor>. And you can also follow our blog, [www.comffics.de](http://www.comffics.de), for updates.

Munich, August 27, 2014.

DRAFT

# Contents

<b>1</b>	<b>Introduction to OpenFOAM - (coming soon)</b>	<b>2</b>
	S. Huq	
<b>2</b>	<b>Laminar Vortex Shedding</b>	<b>3</b>
	M. Saeedi	
2.1	Fluid Mechanics . . . . .	3
2.2	Numerics . . . . .	4
2.3	OpenFOAM . . . . .	7
	2.3.1 Preprocessing . . . . .	7
	2.3.2 Solver . . . . .	11
	2.3.3 Post processing . . . . .	14
2.4	Exercise . . . . .	16
2.5	References . . . . .	16

DRAFT

# **1 Introduction to OpenFOAM - (coming soon)**

S. HUQ

DRAFT

## 2 Laminar Vortex Shedding

M. SAEEDI

Flow around a cylinder is a classic problem in fluid mechanics. It is a problem of practical importance: Off-shore structures, bridge piers, single silos and to some extent cooling towers are typical cases where we face flow over cylindrical structures. Vortex shedding is an important phenomena which occurs in flow around these types of structures for a wide range of Reynolds numbers. The frequency of vortex shedding and analysis of vortex induced vibrations play an important role in design of these structures. In this tutorial laminar flow around a cylinder is simulated using OpenFOAM and the results are validated by comparing them with experimental data. First of all, physical aspect of flow around a cylinder is reviewed. The numerical aspects of simulating the case together with case setup in OpenFoam are discussed, the chapter ends with validating the results by comparing them with experimental data.

### 2.1 Fluid Mechanics

The most decisive non-dimensional parameter for describing the flow pattern around a cylinder is the Reynolds number:

$$Re = \frac{UL}{\nu} \quad (2.1)$$

It is defined as the ratio of inertial to viscous forces. At low Reynolds numbers viscous forces dominate the flow and the flow is laminar, as the Reynold number increases viscous forces cease to dominate the flow and the flow becomes turbulent. For very low Reynolds numbers  $Re < 5$  there is no flow separation, fluid particles from upstream of the flow can be divided to two groups flowing symmetrically around upper and lower part of the cylinder and reaching together the back of the cylinder without separation. As the Reynolds number is increased a small separation region is formed behind the cylinder where two eddies of opposite sign are formed (twin vortices), these vortices are not shed into the flow and the separation region does not extend downstream. With further increase in the Reynolds number the twin vortices elongate and a harmonic oscillation is observed in the wake region. At  $Re \approx 90$  eddies start to shed alternatively from the upper and lower part of the cylinder and form the so-called Kármán Vortex Street. At this range of Reynolds number, the flow is still laminar in the wake. As the Reynolds number increases ( $200 < Re < 300$ ) the flow in the wake becomes turbulent, while the boundary layer flow near the cylinder is still laminar and remains laminar up to  $Re \approx 3 \times 10^5$  (subcritical regime). We then have the critical regime for  $3 \times 10^5 < Re < 3.5 \times 10^5$  where transition to turbulence spreads to the boundary

layer region and the boundary layer becomes turbulent at the separation point at one side of the cylinder while the boundary layer separation at the other side is still laminar. The side where separation is turbulent occasionally switches from one side to the other. Sumer [1] reports a non-zero mean lift on the cylinder because of this asymmetry. In supercritical regime ( $3.5 \times 10^5 < Re < 1.5 \times 10^6$ ) the boundary layer is turbulent at the separation point for both sides but before the separation point it is partially laminar and partially turbulent. Transition from laminar to turbulent happens at some point between the stagnation point and the separation point. Then in the upper transition regime ( $1.5 \times 10^6 < Re < 4 \times 10^6$ ) the boundary layer is completely turbulent for one side while on the other side it is partially laminar and partially turbulent. Finally for  $Re < 4 \times 10^6$  boundary layer is fully turbulent for both sides of the cylinder (transcritical). The above described stages of flow around cylinder are depicted in figure 2.1.

Change of drag coefficient with Reynolds number can be explained using the above discussion on flow regimes at different Reynolds numbers. Figure 2.2 shows drag coefficient as a function of Reynolds number for a smooth cylinder. For  $Re > 3 \times 10^5$  transition to turbulence happens and the separation point moves backwards which decreases the size of the wake, as a result the total pressure force acting on the cylinder (and the drag coefficient) decreases abruptly. The frequency of vortex shedding is described by the dimensionless Strouhal number, after the Czech physicist Vincent Strouhal. Strouhal number is defined as:

$$St = \frac{fL}{V} \quad (2.2)$$

where  $L$  is the characteristic length. For flow around a cylinder the diameter is taken as the characteristic length. For  $200 < Re < 3 \times 10^5$  Strouhal numbers remains approximately constant and  $St \approx 0.2$  is a good estimate for it. Then there is a jump in Strouhal number at transition Reynolds number. Figure 2.3 shows how Strouhal number changes with Reynolds number for a smooth cylinder.

## 2.2 Numerics

Using discretization schemes the Navier-Stokes equations are transformed from partial differential equations to algebraic equations. Different schemes are available to discretize convection, diffusion and gradient terms. They have different properties in terms of accuracy and stability and should be chosen properly with respect to the specific problem. First order schemes are in general too diffusive and would under predict forces and gradients.

### Laplacian schemes

Laplacian terms are computed using Gaussian integration over the cell. In order to perform the Gaussian integration values of the gradients at Gauss points should be calculated first. These values are calculated via interpolation schemes and using the data at cell and face centers. An interpolation scheme should also be assigned to the chosen laplacian scheme, usually a linear interpolation is chosen. Other available interpolation schemes include: cubicCorrection, midPoint, upwind, linearUpwind, limitedLinear, vanLeer, etc.


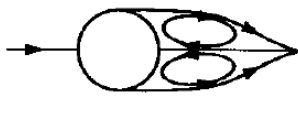


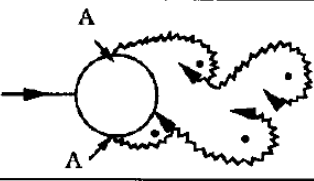
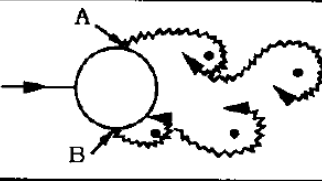
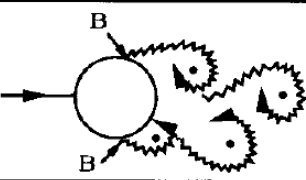
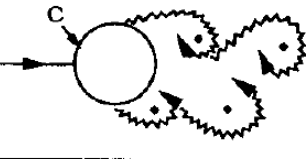
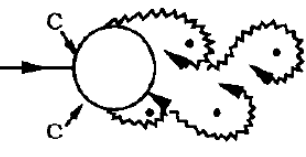
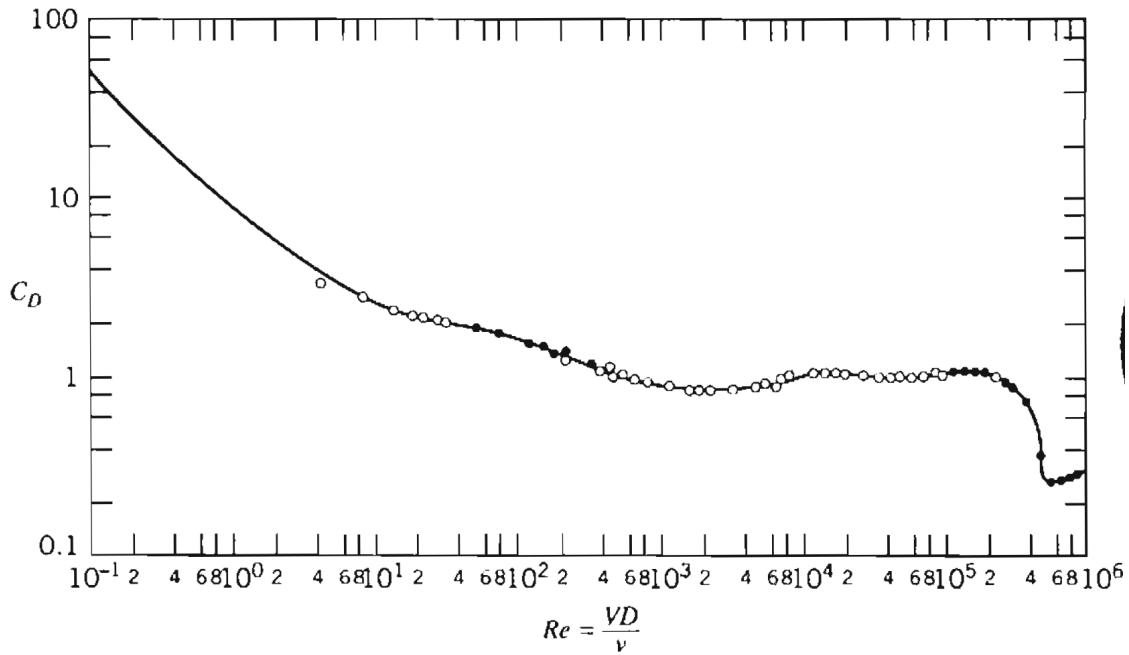
	No separation. Creeping flow	$Re < 5$
	A fixed pair of symmetric vortices	$5 < Re < 40$
	Laminar vortex street	$40 < Re < 200$
	Transition to turbulence in the wake	$200 < Re < 300$
	Wake completely turbulent. A: Laminar boundary layer separation	$300 < Re < 3 \times 10^5$  Subcritical
	A: Laminar boundary layer separation B: Turbulent boundary layer separation; but boundary layer laminar	$3 \times 10^5 < Re < 3.5 \times 10^5$ Critical (Lower transition)
	B: Turbulent boundary layer separation; the boundary layer partly laminar partly turbulent	$3.5 \times 10^5 < Re < 1.5 \times 10^6$ Supercritical
	C: Boundary layer com- pletely turbulent at one side	$1.5 \times 10^6 < Re < 4 \times 10^6$ Upper transition
	C: Boundary layer comple- tely turbulent at two sides	$4 \times 10^6 < Re$ Transcritical

Figure 2.1: Flow pattern at different Reynolds numbers, from [1]





**Figure 2.2:** Drag coefficient as a function of Reynolds number for a smooth cylinder, from [2]

## Gradient schemes

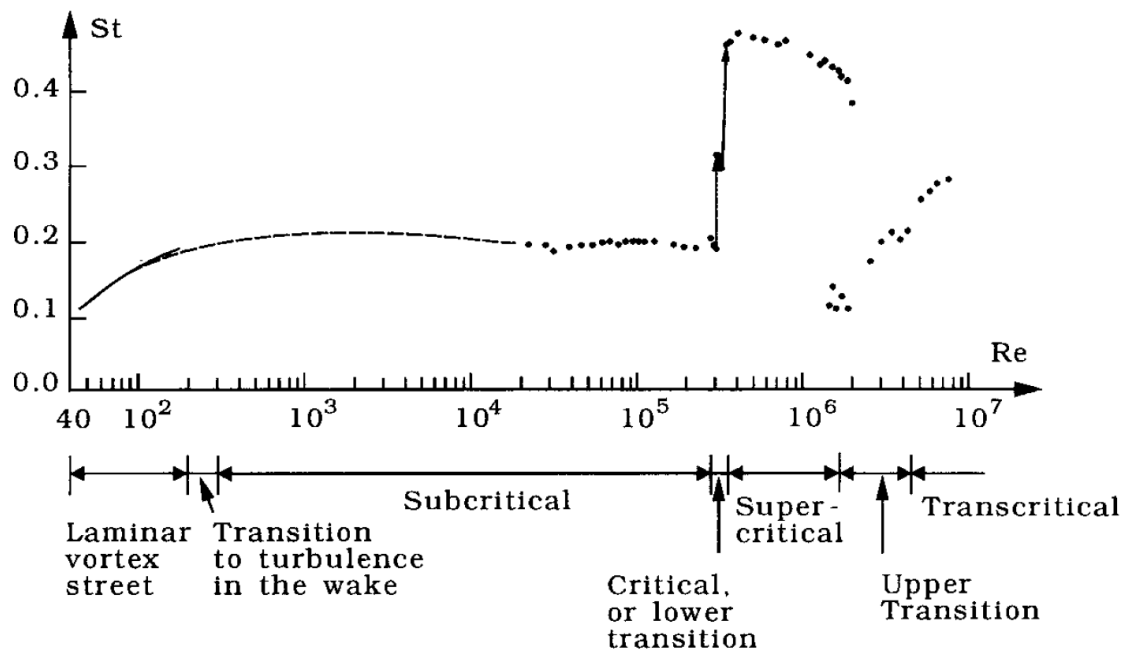
Gradient terms can be calculated by either Gauss integration or using leastSquares method. Second and fourth order least square methods are available. In contrast to the Gauss integration method, least square methods do not need an interpolation scheme. The schemes available for discretizing gradient terms are listed in table 2.1. Gradient limiters can also be used in discretizing gradient terms, they will avoid under shoots and over shoots in calculation of the gradients. faceLimited schemes are in general more diffusive than cellLimited ones.

## Divergence schemes

Similar to Laplacian schemes, Gauss integration is the only discretization method available for discretizing convective terms. A proper interpolation method should be chosen. The simplest method for calculating the divergence is the central difference scheme, which is equivalent to Gauss integration with linear interpolator. This method is however not stable,

**Table 2.1:** Discretization schemes available for gradients

Discretization scheme	Description
Gauss <interpolationScheme>	Second order, Gaussian integration
leastSquares	Second order, least squares
fourth	Fourth order, least squares
cellLimited <gradScheme>	Cell limited version of one of the above schemes
faceLimited <gradScheme>	Face limited version of one of the above schemes



**Figure 2.3:** Strouhal number for a smooth cylinder, from [1](#)

thus upwind scheme is normally used as the standard method for interpolation. Table [2.2](#) lists available interpolation schemes in OpenFOAM.

## 2.3 OpenFOAM

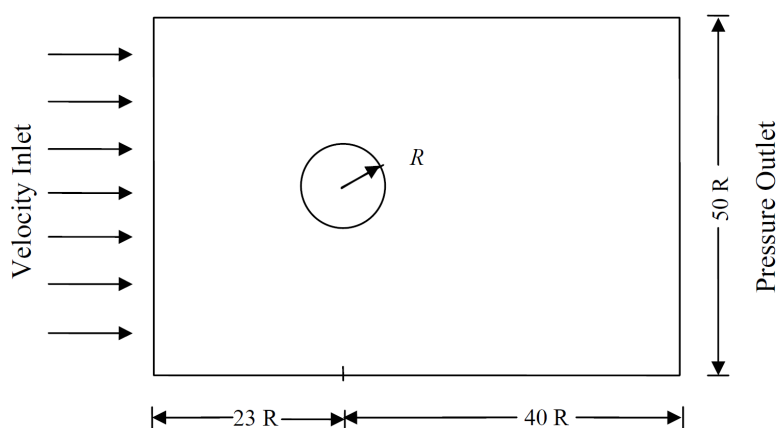
In the following the steps towards simulation of laminar flow around a cylinder are explained. The schematic of the problem is presented in figure [2.4](#).

### 2.3.1 Preprocessing

Preprocessing mainly consists of preparing the blockMeshDict to generate the block-structured mesh, setting the required boundary conditions in the 0 folder and setting the parameters in

**Table 2.2:** OpenFOAM interpolation schemes

linear	Second order, unbounded
skewLinear	Second order, (more) unbounded, skewness correction
cubicCorrected	Fourth order, unbounded
upwind	First order, bounded
linearUpwind	First/second order, bounded
QUICK	First/second order, bounded
TVD schemes	First/second order, bounded
SFCD	Second order, bounded
NVD schemes	First/second order, bounded

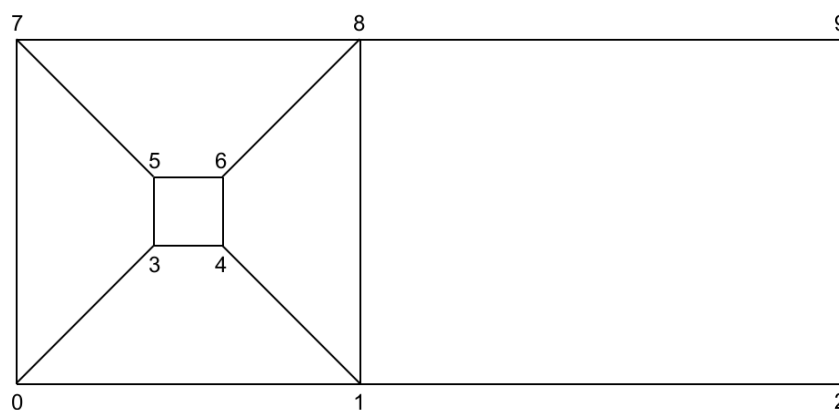


**Figure 2.4:** Schematic presentation of the computational domain

contolDict. Based on what you want to simulate you might also need to modify the schemes used within the solver for discretizing and solving the governing PDEs which is done in fvSchemes file in system directory.

### Meshing

The mesh can be generated using different blocking strategies. Here we use four blocks to generate an o-grid type mesh around the cylinder. A fifth block is attached to the o-grid mesh to extend the domain in the downstream region of the flow, figure 2.5. The four edges



**Figure 2.5:** Dividing the computational domain into 5 blocks

3 – 4, 4 – 6, 6 – 5 and 5 – 3 are straight lines by default. In order to represent cylinder edges using them, they should be changed to arcs. It can be done in the *edges* section of the blockMeshDict using *arc* command:

```
1 edges
2 (
3     arc 3 5 (-1 0 0)
4 );
```

The first two numbers indicate the start and end point of the arc. Infinite number of circular arcs can be defined between two points, we need a third point to specify the exact arc we need. The coordinate of this third point follows after start and end point numbers in the arc command,  $(-1, 0, 0)$  in this case. There are other options for defining curved edges between to vertices. Using splines for example, one can represent an airfoil profile using just 3 vertices and defining 3 splines for edges connecting them. Available options for defining curved edges in blockMeshDict are: arc, simpleSpline, PolyLine and polySpline. Their description is available in OpenFOAM user guide.

## Boundary conditions

Six patches are defined in this problem: inlet, outlet, top, bottom, cylinder, sides. Top and bottom patches are defined to be far enough from the cylinder, they are defined as slip walls. Since it is a 2D problem, side walls are defined in a single patch with type empty. The boundary conditions used are summarized in table 2.3. The *zeroGradient* boundary condition implies the gradient to be zero normal to the patch and *slip* is the same as zeroGradient for scalar parameters while for vectors it implies *fixedValue zero* to the normal to the patch component of the vector and *zeroGradient* to tangential components of the vector. Other used boundary conditions are more or less self-explanatory. The magnitude of the velocity at the inlet is set to a fixed value of  $1m/s$ , to fulfill  $Re = 100$  we can later play with dynamic viscosity.

**Table 2.3:** Boundary conditions

BC	Patch	Boundary condition
U	inlet	fixedValue, uniform (1 0 0)
	outlet	zeroGradient
	cylinder	fixedValue, uniform (0 0 0)
	top	slip
	bottom	slip
	sides	empty
p	inlet	zeroGradient
	outlet	fixedValue, uniform 0
	cylinder	zeroGradient
	top	zeroGradient
	bottom	zeroGradient
	sides	empty

## controlDict

We have already explained the controlDict in our tutorial #3. Here we explain how to define functions to calculate forces and force coefficients on a certain patch during run time. We can later use these outputs to validate our simulation.

A function block can be defined at the end of controlDict. Functions perform a post-processing step at the end of each time step. Different functions are available including: *faceZonesIntegration*, *fieldAverage*, *fieldMinMax*, *probes*, etc.

A generic function block looks like the following:

**Listing 2.1:** Definition of a general function in controlDict

```

1 functions
2 {
3     // a given name to distinguish between defined functions
4     name
5     {
6         // name of the function e.g. forces, probes,...
7         type          <functionName>;
8         // name of function's library
9         functionObjectLibs ("<libName>");
10        // control parameters for function output
11        outputControl    timeStep;
12        outputInterval    10;
13
14        ... // function-specific entries
15    }
16 }
```

In this tutorial we are going to use the two following ones:

- **forces**: calculates the forces and moments acting on a list of patches by integrating the pressure and shear forces acting on them. Additional entities like fluid density and center of rotation of the body should be specified by the user. Defining a force function in the controlDict is as follows:

**Listing 2.2:** Definition of force function in controlDict

```

1
2 forces          // given name
3 {
4     type          forces;
5     functionObjectLibs ("libforces.so");
6     outputControl    timeStep;
7     outputInterval    10;
8
9     patches          ( "cylinder" );
10    pName             p;
11    UName              U;
12    rhoName            rhoInf;
13    log                true;    // true writes the data to the
14                             // shell, false doesn't.
15    CofR               (0 0 0); // center of rotaion of the body
16    rhoInf             1.225;   // fluid density
17 }
```

- **forceCoeffs**: calculates lift, drag and pitching moment coefficients for a patch list. These coefficients are defined as:

$$C_l = \frac{L}{\frac{1}{2}\rho A U_\infty^2} \quad (2.3)$$

$$C_d = \frac{D}{\frac{1}{2}\rho AU_\infty^2} \quad (2.4)$$

$$C_M = \frac{M}{\frac{1}{2}\rho AU_\infty^2 c} \quad (2.5)$$

To calculate the coefficients the reference area and reference length (for moment coefficient) should be specified by the user. By definition, the lift force acts perpendicular to the velocity vector and drag acts parallel to it. These directions as well as pitching moment direction are declared in function block:

**Listing 2.3:** Definition of forceCoeff function in controlDict

```

1 forceCoeffs
2 {
3     type                forceCoeffs;
4     functionObjectLibs  ( "libforces.so" );
5     outputControl       timeStep;
6     outputInterval      10;
7
8     patches             ( "cylinder" );
9     pName               p;
10    UName                U;
11    rhoName              rhoInf;
12    log                  true;
13
14    liftDir               (0 1 0);
15    dragDir               (1 0 0);
16    CofR                 (0 0 0);
17    pitchAxis             (0 0 1);
18
19    magUInf               1;
20    rhoInf                1.225;
21    lRef                  1;
22    Aref                  2;
23 }
```

### 2.3.2 Solver

*pisoFoam* is a transient solver for incompressible flow. Both laminar and turbulent flows can be simulated using *pisoFoam*. It is based on the PISO algorithm, which stands for Pressure Implicit with Splitting of Operators, proposed by Issa in 1995. It is a pressure-velocity coupling algorithm with a predictor step and two corrector steps. In the predictor step the momentum equations are solved for an intermediate pressure field. The predicted velocity field at this stage do not satisfy continuity condition. Then during the two predictor steps, velocity and pressure fields are corrected in a way to fulfill both, momentum and continuity equations. The PISO algorithm consists of the following steps:

- Prediction of the intermediate velocity field from momentum equations
- PISO loop
  - Calculate mass fluxes on cell faces
  - Solve pressure field (*nNonOrthcorr* times)
  - Correct Velocity field
- Repeat the PISO loop *nCorr* times
- Solve the turbulence model
- Proceed to the next time step

Schematic presentation of the algorithm can be seen in figure 2.6. The way the algorithm is implemented in OpenFOAM can be seen in the source files of pisoFOAM solver:

**Listing 2.4:** Implementation of PISO loop in pisoFoam

```

1  // Pressure-velocity PISO corrector
2  {
3      // Momentum predictor
4
5      fvVectorMatrix UEqn
6      (
7          fvm::ddt(U)
8          + fvm::div(phi, U)
9          + turbulence->divDevReff(U)
10     );
11
12     UEqn.relax();
13
14     if (momentumPredictor)
15     {
16         solve(UEqn == -fvc::grad(p));
17     }
18
19     // --- PISO loop
20
21     for (int corr=0; corr<nCorr; corr++)
22     {
23         volScalarField rAU(1.0/UEqn.A());
24
25         U = rAU*UEqn.H();
26         phi = (fvc::interpolate(U) & mesh.Sf())
27             + fvc::ddtPhiCorr(rAU, U, phi);
28
29         adjustPhi(phi, U, p);
30
31         // Non-orthogonal pressure corrector loop
32         for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
33         {
34             // Pressure corrector
35
36             fvScalarMatrix pEqn
37             (
38                 fvm::laplacian(rAU, p) == fvc::div(phi)
39             );
40

```

```
41         pEqn.setReference(pRefCell, pRefValue);
42
43         if
44         (
45             corr == nCorr-1
46             && nonOrth == nNonOrthCorr
47         )
48         {
49             pEqn.solve(mesh.solver("pFinal"));
50         }
51         else
52         {
53             pEqn.solve();
54         }
55
56         if (nonOrth == nNonOrthCorr)
57         {
58             phi -= pEqn.flux();
59         }
60     }
61
62     #include "continuityErrs.H"
63     U -= rAU*fvc::grad(p);
64     U.correctBoundaryConditions();
65 }
66 }
67
68 turbulence->correct();
```



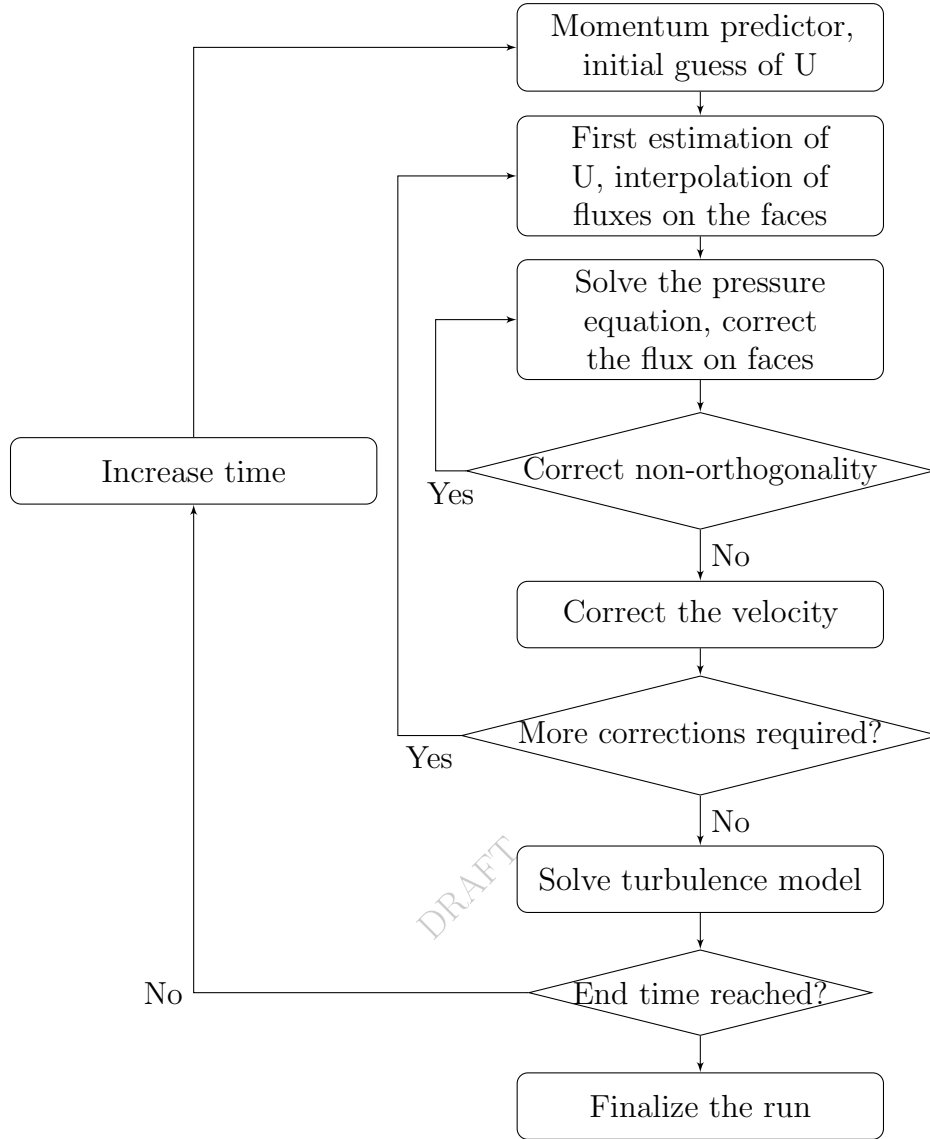
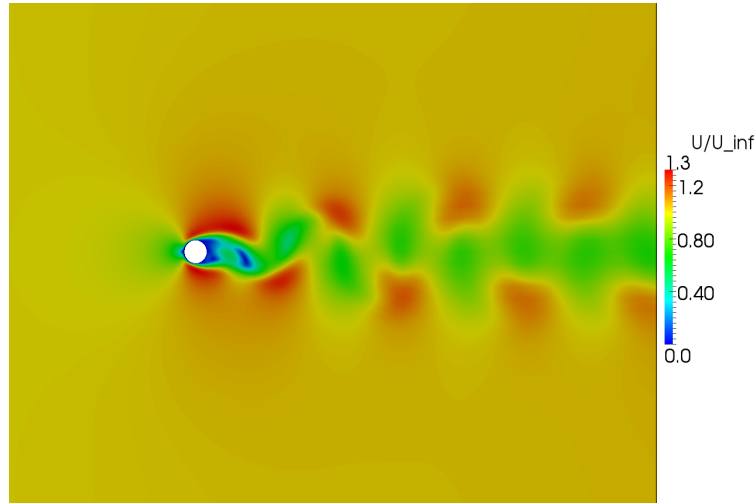


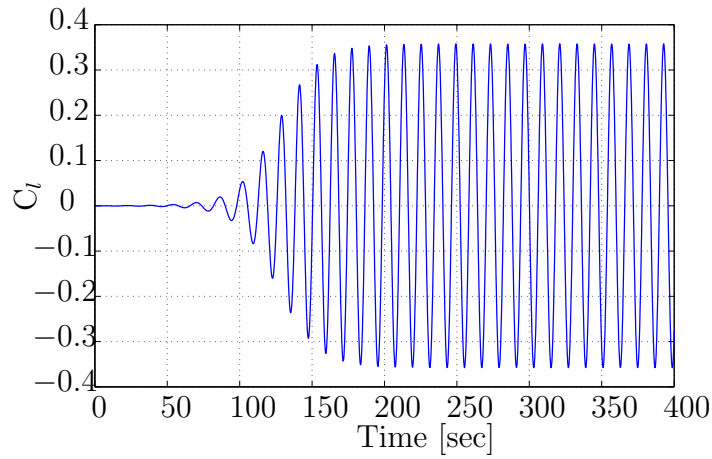
Figure 2.6: PISO algorithm

### 2.3.3 Post processing

In this part the results of the simulation are evaluated and compared with reference results reported in the literature. The initial flow needs some time to develop around the cylinder, what happens after a while is the onset of vortex shedding and development of the von Karman vortex street (figure 2.7). The Periodic shedding of the vortices into the wake results in an oscillating lift and drag force on the cylinder. The change in lift and drag coefficients in time can be seen in figures 2.8 and 2.9. With the emergence of vortex shedding and as flow develops itself around a cylinder the drag coefficient begins to oscillate about a mean value of 1.386 and the amplitude of lift coefficient is 0.358. They are in a very good agreement with the reported values for drag coefficient and the amplitude of lift coefficient from [3] which are 1.39 and 0.35.



**Figure 2.7:** Velocity distribution around the cylinder and the von Karman vortex street

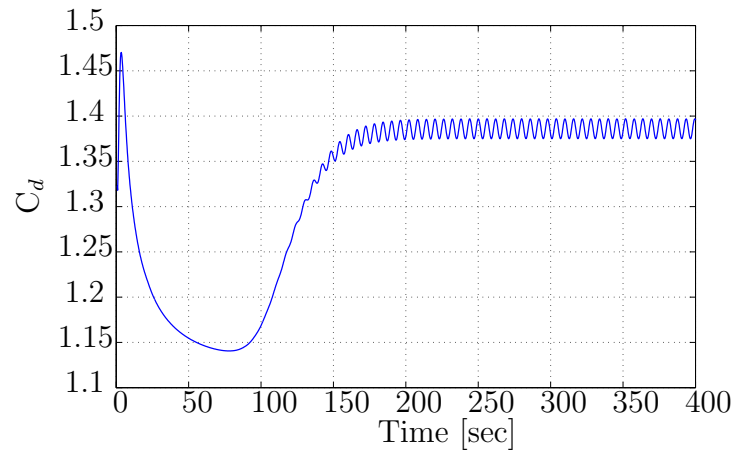


**Figure 2.8:** Lift coefficient

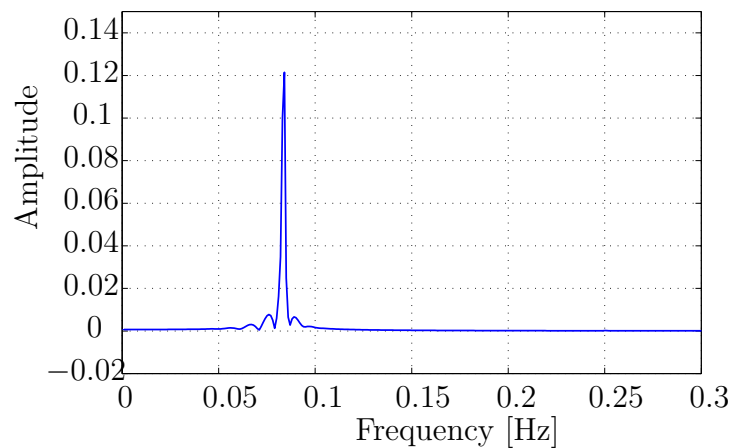
The next quantity to be validated is the frequency of vortex shedding. It can be done by Fourier analysis of the forces acting on the cylinder. Using Fourier transform the amplitude of different frequencies present in a signal (here lift coefficient) are calculated (figure 2.10). As can be seen in the figure the oscillations in lift coefficient is dominated by a single frequency at  $f = 0.084$  Hz, which is the frequency at which vortices are being shed into the wake. The corresponding Strouhal number can be calculated from equation 2.2:

$$St = \frac{fL}{V} = \frac{0.084 \times 2}{1} = 0.168 \quad (2.6)$$

It matches very well with the reported value of 0.17 from [3].



**Figure 2.9:** Drag coefficient



**Figure 2.10:** Fourier analysis of lift coefficient

## 2.4 Exercise

Laminar pipe flow is another benchmark for laminar fluid flow. Use the same fluid setup, to simulate laminar flow in pipes. Observe the development of velocity profile from the entrance to the region of fully developed flow. The velocity profile, together with the entrance length and pressure drop in the pipe could be good candidates to evaluate your results.

## 2.5 References

1. Sumer, B. Mutlu, et al. Hydrodynamics Around Cylindrical Structures. Singapore: World Scientific, 2006
2. Fox, Robert W., et al. Introduction to Fluid Mechanics. Hoboken, NJ: Wiley, 2010

3. Durbin, Paul A., et al. Fluid Dynamics with a Computational Perspective. Cambridge: Cambridge Univ. Press, 2007

DRAFT