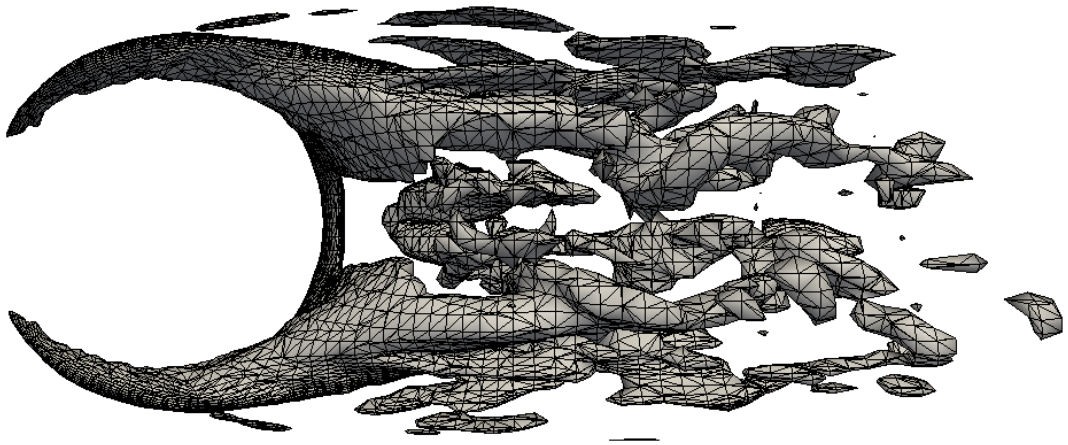
THESIS for the degree of MASTER OF SCIENCE

Master i Anvendt matematikk og mekanikk

CFD simulations of flow in bent pipe at high Reynolds numbers conditions

A set of CFD simulations are preformed, using two different turbulence models, on bent pipe sections. The CFD simulations is mainly for studying the pressure conditions at the pipe walls because of vortex shedding.

Sayed Millad Nour



Faculty of Mathematics and Natural Sciences

UiO : University of Oslo

November 20, 2014

Acknowledgments

First and foremost I would like thank Mikael Mortensen who is my main supervisor. I appreciate all the help I have received. So thank for all the information, thanks for all your time and thanks for all your patience.

Next I would like to thank Mr. Arnaud Sanchis, for taking time and answering questions I had about the simulations.

And finally I would like to thank my family for all the support and help. Not only through the process of writing a master thesis, but the last five years, from the beginning of my university education.

Contents

List of Figures

List of Tables

Abbreviations

JAEA: Japanese Atomic Energy Agency

JSFR: Japan Sodium-cooled Fast Reactor

FaCT: Fast reactor Cycle Technology development

RV: Reactor Vessel

IHX: Intermediate Heat eXchanger

LDV: Laser Doppler Velocimetry

RNG: Re-Normalization Group

CFD: Computational Fluid Dynamics

U-RANS: Unsteady Reynolds-Averaged Navier-Stokes equations

RMS: Reynolds Stress Model

LES: Large Eddy Simulations

FFT: Fast Fourier Transform

PSD: Power Density spectrum

Preface

I would like to give you some information about this document and my master project before you start reading.

- All the computers I used had 64 bit Linux operating systems.
- Tests and 2D simulations where done using my personal computer, while the main computing tasks where preformed on a supercomputer at the University of Oslo (The Abel Computer Cluster).
- I used different versions of OpenFOAM. On my home computer I had versions 2.3.0 and 2.2.2, while on The Abel Computer Cluster I had version 2.1.1 installed.
- Most programs and scripts made for this project is not included in this document. You can in Appendix A find a selection of flow simulation codes, code for implementing a mesh and some shell scripts. The rest of the material can be found at my `GitHub` repository: `https://github.com/sayedn/Master-`
- All the mathematical formulas, variables, functions, etc., in this thesis is for a standard Cartesian coordinate system with directions x, y and z.
- I've made some movies from the simulations. You can view them at my `GitHub` account: `https://github.com/sayedn/Master-/tree/master/mov`
- If you are reading this document on paper, I recommend that you also have the PDF-file open on your personal computer while reading. There is a lot of hyper-links to websites, movies and other things in this document.

Chapter 1

Introduction

Good understanding of fluid flow and behavior makes us able to design better equipment that last for a longer time, it will save ass a lot of money and is better at performing the task it is designed for. It will save as a lot of money by reducing construction cost (we don't need to scale design up to make it stronger). Therefor a well performed analysis of the fluid motion is a good investment.

Unfortunately turbulent fluid flows which are "chaotic " in nature are much more common in the physical world than the laminar "well behaved" flows.

All fluid motions can be described by the Navier-Stokes equations. Unfortunately there is no analytical solution to these equations at the present moment. And solving the Navier-Stokes equations numerically has a high computational cost. Meaning that if you want good accuracy you need a extremely fine mesh resolution.

So all of the points mentioned above, necessity of solving for turbulent flows, problems with solving the Navier-Stokes equations, ..., resulted in the development of turbulence models. Models that can describe and catch the main features for turbulent motion. Several turbulence models have been developed for describing turbulent flows. I will hare only mention two: RANS and LES. There is ups and downs with both models depending on what kind of problem wee want to solve.

In this thesis we will be focusing on finding pressure fluctuations and velocity fields in a bent pipe section with a short radius of curvature. We will be working alongside a article written by Tanaka[] at the Japanese Atomic Energy Agency (JAEA). The bent pipe section is a part of the cooling system of a new nuclear reactor design. You can see a illustration of the reactor in Figure (1.1). The design has a two loop cooling system. In

Boundary Conditions for Numerical Simulations

	U_m [m/s]	Re [-]	dt [ms]	Working Fluid	Mesh
B4	9.2	3.7×10^6	1.0	Water at 20°C	A
B7	9.2	3.7×10^6	0.1	Water at 20°C	B
B8	3.08	1.2×10^6	1.0	Water at 20°C	A
B9	0.8	0.3×10^6	1.0	Water at 20°C	A
B10	9.2	8.0×10^6	1.0	Water at 60°C	A
B11	9.2	1.4×10^7	1.0	Sodium at 550°C	A
D0	9.2	4.2×10^7	0.1	Sodium at 550°C	D

Table 1.1: Table of cases from TANAKA et al. [9]

each loop hot liquid sodium, which is the cooling liquid, will flow from the upper plenum of the reactor vessel, through the bent pipe [1] and into a heat exchanger (IHX) where the liquid will be cooled down. Finally the cold cooling liquid will be pumped back to bottom of the reactor core through the two "Cold-legs" and the cycle can restart.

At normal working capacity the averaged velocity of the cooling liquid at he beginning of the "Hot-pipe" elbow will be $\sim 9.2m/s$ with 5% turbulence intensity. And with a temperature of $\sim 550°C$ of the liquid sodium, we get a flow in the category of high Reynolds numbers flows. A simple estimate of the Reynolds number for circular pipe flow can be found by the formula

$$Re = \frac{\rho \mathbf{v} D}{\mu}. \tag{1.1}$$

Here ρ is the liquid density, $\mathbf{v}$ is the mean velocity, D is the pipe diameter and μ is the dynamic viscosity. In our case we get that

$$Re \approx .$$

Equation (9) can be found in most text books covering the subject turbulence.[2] If we next calculate the relative friction factor

$$\frac{\epsilon}{D}, \tag{1.2}$$

and then look at the diagram in Figure (1.2), we see that our flow is well inside the region for turbulent flows.

[1] Referred to as the hot-leg in TANAKA et al. [9]

[2] I found it at Wikipedia. http://en.wikipedia.org/wiki/Reynolds_number

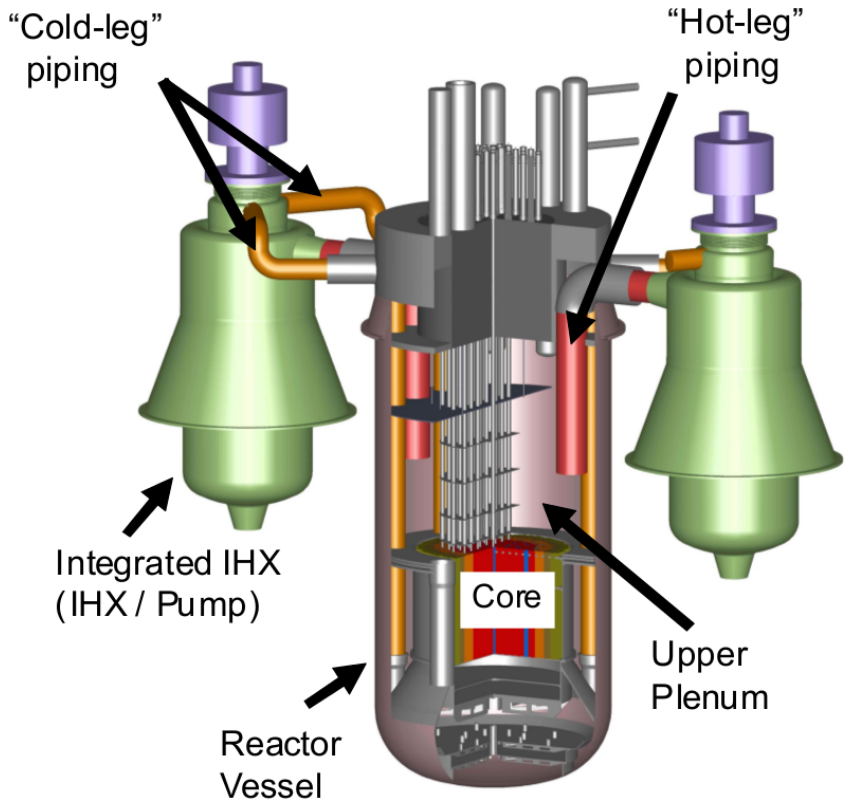


Figure 1.1: Japanese sodium-cooled fast reactor. The Figure is from Ono et al. [6].

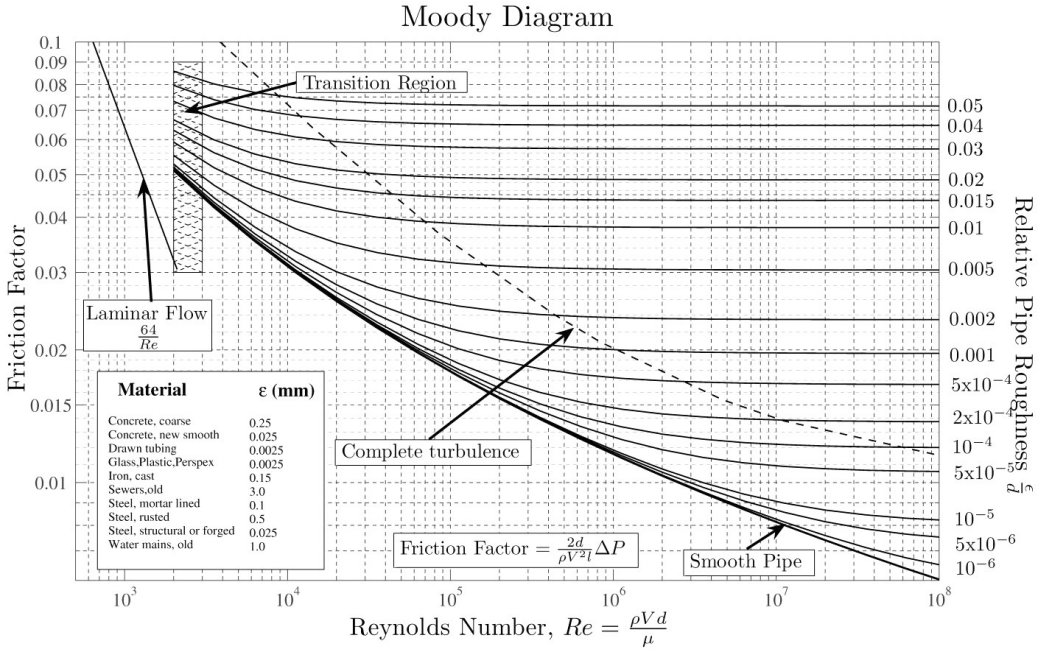


Figure 1.2: Moody Diagram. The Figure is from Wikipedia [2].

Chapter 2

Geometry and Mesh

In this chapter I am going to show how the different meshes, used for the simulations, where implemented. I will here just mention four different meshes, but I made several others. Most of them where for simple geometries like straight pipe sections and pipe elbows. And they were used for testing flow simulation code, mesh element shapes and other things. The main four meshes are:

- `Mesh-A_2D` is a 2D mesh for the test section, see Figure (2.1)
- `Mesh-A` is a 3D mesh for the test section
- `Mesh-B` is also a 3D for the test section, but with is finer mesh resolution then `Mesh-A`
- `Mesh-D` is a 3D mesh for the hot-leg piping, see Figure (2.2)

Meshes `Mesh-A`, `Mesh-B` and `Mesh-D` are similar to meshes Mesh-A, Mesh-B and Mesh-D described in TANAKA et al. [9]. The difference is that each of my meshes has a slightly finer mesh resolution then its counterpart in [9]. And this is because the information about the meshes in [9] is not complete.

Most of the content in this chapter is referred back to the user manual of Gmsh[1], witch is a free meshing software.

2.1 Geometries

In the article by TANAKA et al. [9], meshes where made for two different geometries. The first geometry was the hot-leg piping, witch is installed inside both primary cooling systems[2]. The second geometry was the test pipe. A pipe installed in the "1/3 scale water test". A experimental apparatus of the primary cooling system[3].

I will here use the two geometries described in [9]. In Figure (2.1) and (2.2) you see the geometries for the test pipe and the hot-leg piping.

[1] `http://geuz.org/gmsh/doc/texinfo/gmsh.pdf`
[2] See Figure (1.1)
[3] The experimental apparatus is a 1/3 scale model of the actual full size cooling system

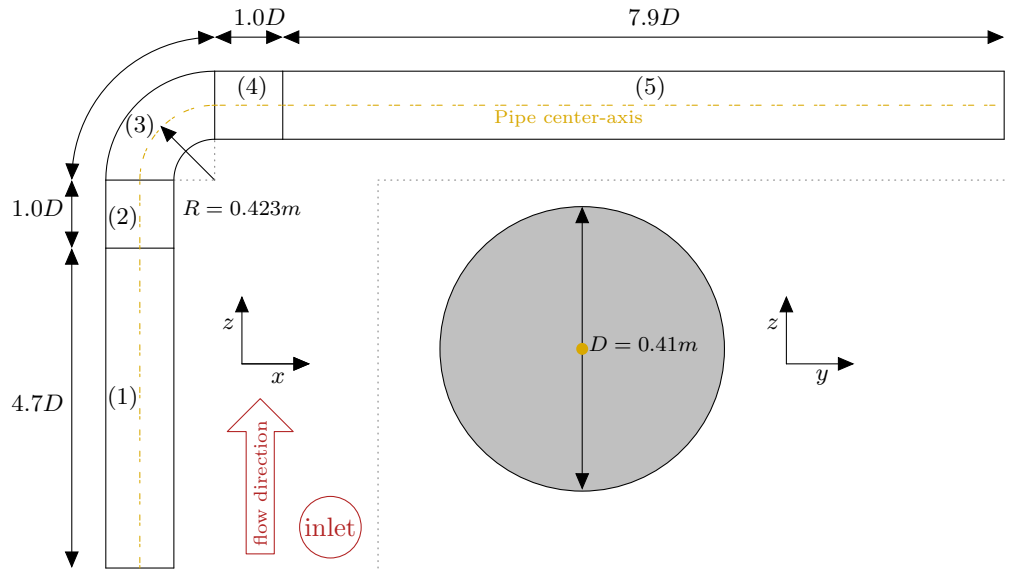


Figure 2.1: In this figure you see the geometry of the test pipe. (1)-(5) is the pipe section labels and R is the *radius of curvature*. The origin ($x = 0$, $z = 0$) in this figure is at the lower left corner of section (4). The vertical line between sections (3) and (4) is the pipe elbow outlet.

2.2 Mesh

As mentioned above, the meshing software I used was Gmsh, which is a finite element mesh generator.

2.2.1 Implementation of the meshes

You can implement meshes, using Gmsh, in several ways dependent of what you find most comfortable. Gmsh is supplied with a graphical interface witch is easy to use. But if you prefer working in a text-editor with source code, similar creating meshes with `blockMesh`[4], you can also do that. All the geometry and meshing information is stored in a `.geo` file, witch is a instruction file you can edit manually . The `.geo` file is written in Gmsh's own scripting language. The complete `.geo` files for mesh `Mesh-A` can be found in Appendix A.1. All the 3D meshes (`Mesh-A`, `Mesh-B` and `Mesh-D`) have the same basic form. What's different between the meshes is the mesh refinement, pipe section lengths and pipe diameters.

I will here just walk you through the implementation of `Mesh-A`. `Mesh-B` and `Mesh-D` is made using the same code as for `Mesh-A`, but with adjustments to the pipe diameters, pipe section lengths and mesh refinement (in different directions) parameters. Take a look at Table (2.1) for a overview of the different meshes.

We start at the pipe inlet. To create the same mesh arrangement as the meshes in

[4] `blockMesh` is the mesh generator supplied with OpenFOAM. When implementing meshes using `blockMesh`, you simply write all the mesh and geometry information (node positions, line segments, mesh refinement, etc.) in a `C++` dictionary class object file

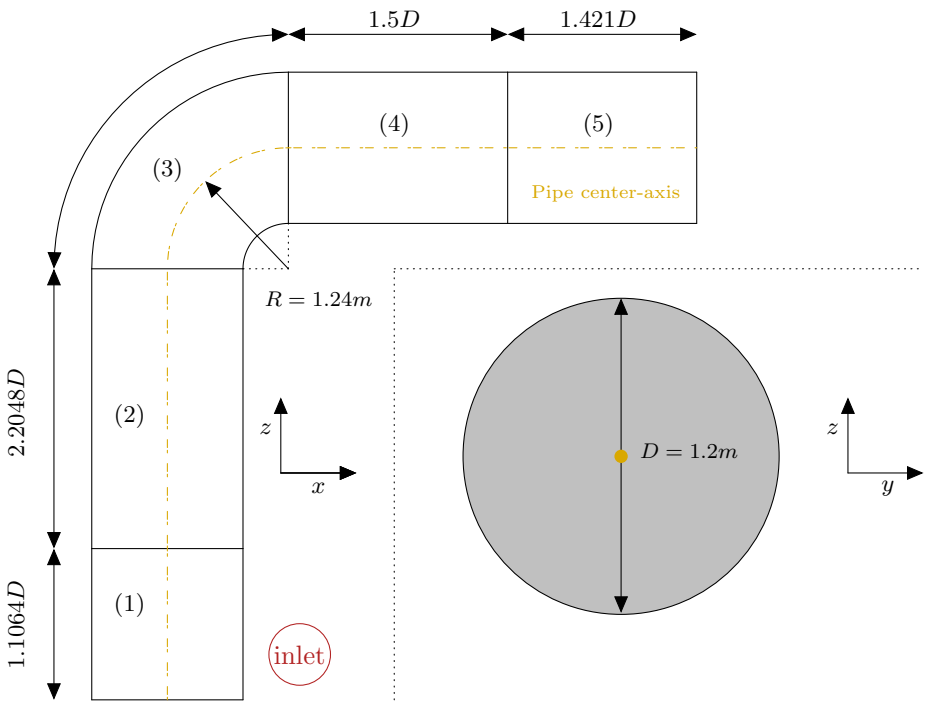


Figure 2.2: In this figure you see the geometry of the hot-leg piping. (1)-(5) is the pipe section labels and R is the *radius of curvature*. The origin ($x = 0$, $z = 0$) in this figure is at the lower left corner of section (4). The vertical line between sections (3) and (4) is the pipe elbow outlet.

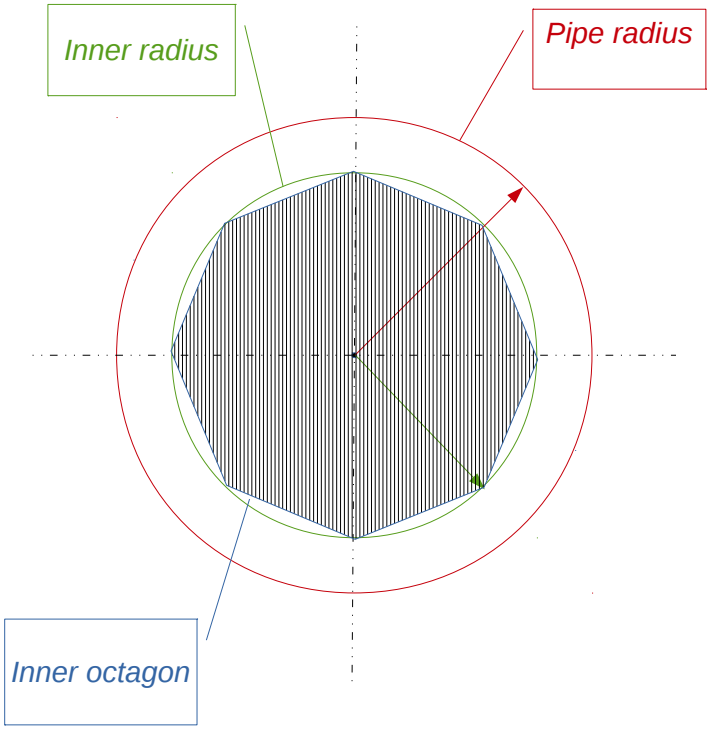


Figure 2.3: Pipe cross-section

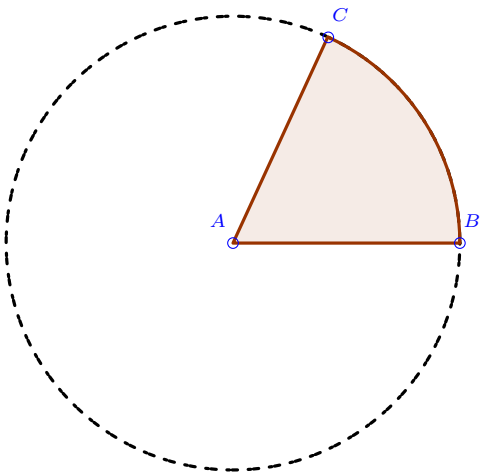


Figure 2.4: A illustration for the use of `Circle`. Point A is the circle center, point B is the arc starting point and point C is the arc ending point.

TANAKA et al. [9], we have to do something special. Looking at the pipe outlet in Figure (2.10), it is possible to divide the mesh in to two parts. A inner part, with a "inner" octagon, and a remaining "outer" part from the octagon to the pipe radius. Take a look at Figure (2.3). In the plane normal to the pipe center-axis position all the nodes/points like in Figure (2.5a). In the `.geo` file you define nodes and node positions like this

```
Point(1) = {xs, ys, zs, 1.0};
Point(2) = {r1 + xs, ys, zs, 1.0};
```

The number inside the parenthesis (round brackets) is the point label. To the right for the equality sign , inside the curly brackets, you have four values separated by commas. The three first values are the x,y,z positions of the node and the fourth value is a local mesh refinement parameter. The next step is to draw straight lines between the nodes. The code for this in the `.geo` file will be as

```
// lines for inlet
Line(1) = {1, 2};
Line(2) = {1, 3};
```

The values to the right for the equality sign, inside the curly brackets, are references to the specific nodes a line is drawn between. As an example, the first line segment `Line (1)` is drawn between points `Point (1)` and `Point (2)`. In Figure (2.5b) you see where the straight line segments should be placed on the inlet surface.

The outer parts of the mesh (pipe wall) consists of ruled surfaces. And the ruled surfaces are themselves made up of bent lines. The code for making circular arcs is

```
Circle(21) = {10, 1, 15};
Circle(22) = {15, 1, 11};
```

Inside the curly brackets, to the right for the equality sign, the numbers separated by commas are references to points. The first value is the arc starting point, the second value is the circle center and third value is the arch ending point. Take a look at Figure (2.4). With the curved lines in place, we now have a inlet-surface looking like in Figure (2.5c).

All the lines on the inlet plane is now positioned, but we haven still defined the closed curves as surfaces. This is done in two steps. First group a set of lines into a closed curve, then define the area inside the closed curve as a surface. The code for this is

```
// surfaces for inlet
Line Loop(29) = {1, 5, 6, -2};
Ruled Surface(30) = {29};
Line Loop(31) = {2, 7, 8, -3};
Ruled Surface(32) = {31};
```

`Line Loop` is your closed curve. The values inside the curly brackets are references to the specific lines the closed curve is made of. The actual defining of the surface happens with `Ruled Surface`, with the value inside the curly brackets being the reference to the specific closed curve (`Line Loop`).

A plane normal to the pipe center-axis is the same no matter where along the pipe we choose the plane. So we now basically need to make a continuous copy of the inlet-surface along the pipe center-axis. This can be obtained with the `Extrude` statement.

```
Extrude {0, 0, l1} {
  Surface{34, 32, 30, 36, 48, 46, 44, 42, 40, 38, 52, 50};
}
```

Above you see two curly brackets following each other after the `Extrude` keyword. The last curly bracket containing a `Surface` array. The first curly bracket is the displacement vector, and the second curly bracket holds surfaces you want to copy along the displacement vector.

One nice feature with `Extrude` is that, when you take a surface and "extrude" it, the `Extrude` statement will at the same time create volumes and volume surfaces. For a pipe section, for example, the pipe volume and pipe walls will be also implemented.

The mesh refinement along the pipe center-axis is not the same everywhere. So we need to divide the pipe into sections. As in [9] I have also divided the pipe into five sections along the pipe center-axis. And this, in terms of writing code, means that we have to use the `Extrude` statement five times. For each time, taking the newly created plane surface and extruding it. For the bent pipe section which you can see in Figure (2.6b), the `Extrude` statement has to be configured in a different way.

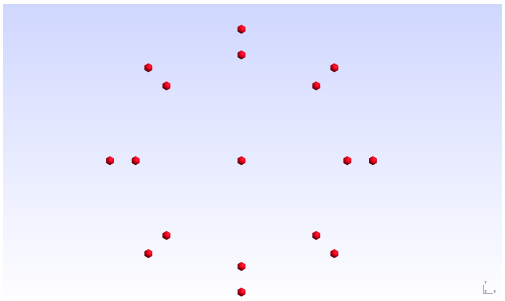
```
Extrude {{0, 1, 0}, {xs + R, 0, zs+l1+l2}, Pi/2} {
  Surface{404, 338, 360, 382, 558, 580, 426, 448, 470, 492, 514, 536};
}
```

The first of the two "outer" curly brackets contain within it, two other curly brackets and a value (all separated by commas). Of these two curly brackets, the first is the axis which the rotation is done about. The second curly bracket is the position of the rotational axis. And the value is the degree of rotation. You can see the final pipe geometry in Figure (2.6c).

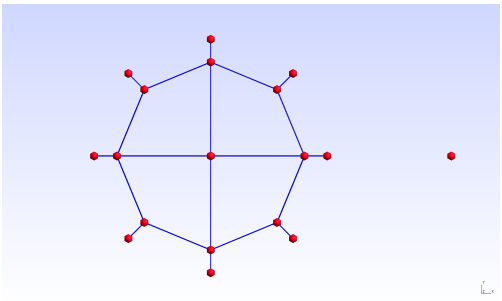
After we have finished creating surfaces and volumes, we need to define patches. Their is two types of patches, surface patches and volumes patches. A surface patch is simply all the faces of elements in contact with a physical surface defined into a group. A volume patch is a number of elements (with their faces) defined in to a group. Faces in a surface patch is not included in volume patches. This "marking" of element faces is used by the CFD-software to applying the right boundary conditions to the right surfaces.

For a pipe you need to define three surface patches and one volume patch. The surface patches are `inlet`, `outlet`, and `fixedwall`. The volume patch is `internal`. The code for this is

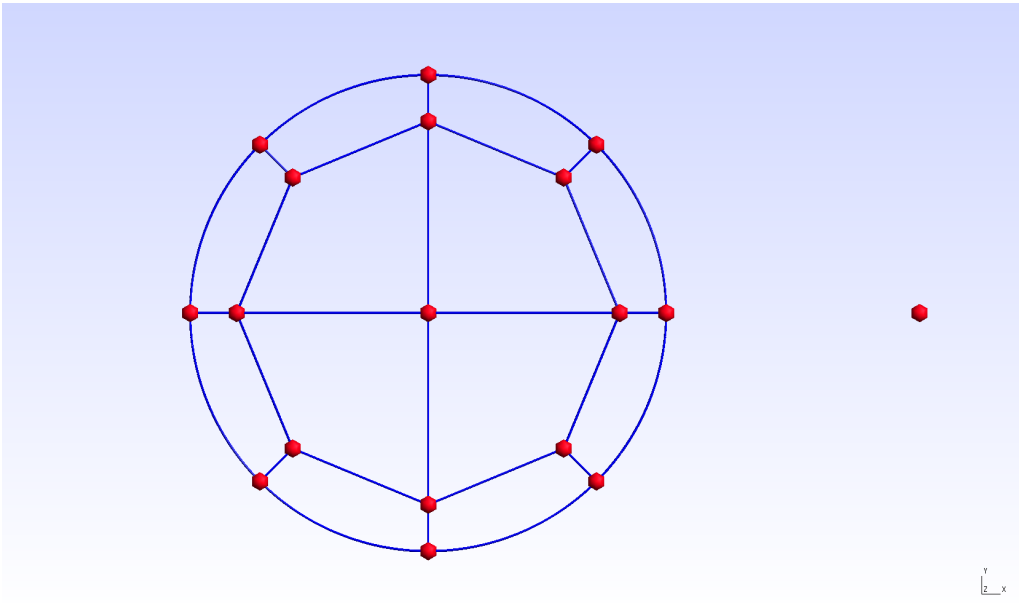
```
Physical Surface("inlet") = {30, 36, 34, 32, 40, 38, 52, 50, 48, 46, 44,
    42};
```



(a) Node distribution on the inlet-surface.

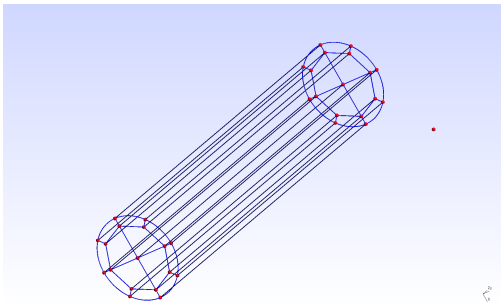


(b) Straight line segments on the inlet-surface.

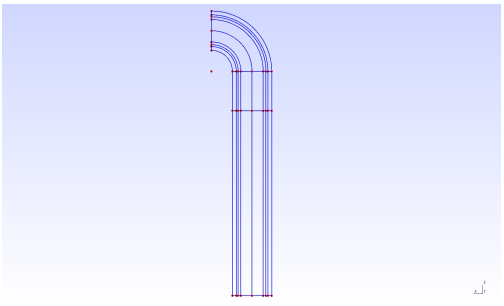


(c) A picture of the inlet-surface with all the nodes and line segments in place. The single node to the right in this picture is for implementation of the bent pipe section later on.

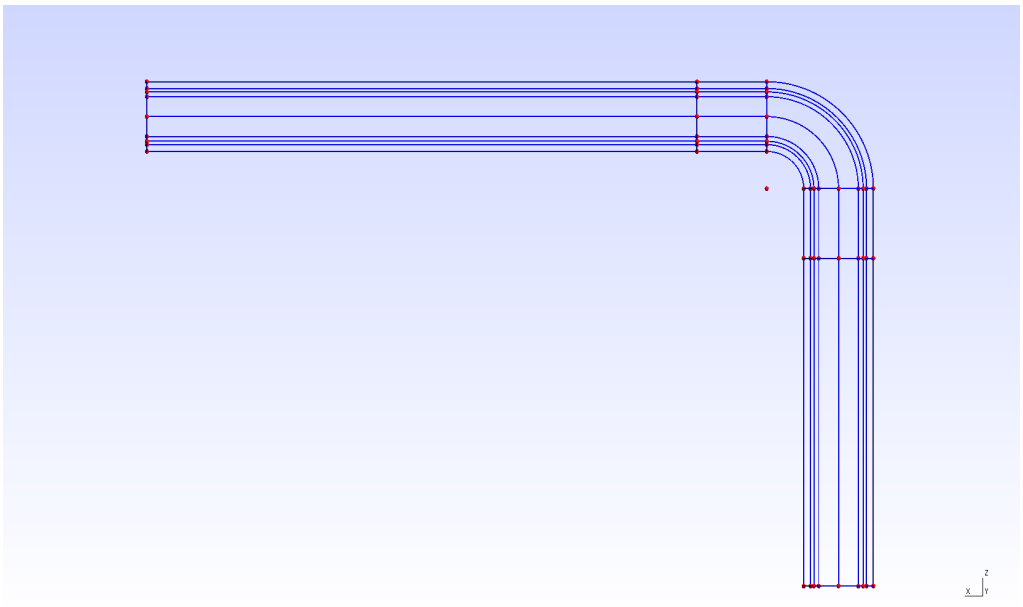
Figure 2.5: A picture of the inlet-surface. The inlet-surface is normal to the pipe center-axis.



(a) A pipe section created after using the `extrude` function on the inlet-surface.



(b) A part of the pipe geometry with the bent pipe section in place.



(c) The complete pipe geometry from the side ($xz - plane$).

Figure 2.6: The pictures illustrate the use of the `extrude` function in Gmsh. The complete pipe consist of 5 sections with four being straight and one bent.

The string inside the round brackets in the patch label. The numbers in side the curly brackets, to the right for the equality sign, is references to surfaces. For a volume patch you have to use the `Physical Volume` statement.

By default Gmsh will mesh with tetrahedral elements and the element distribution will be uniform. For producing hexahedral elements and controlling their shapes and sizes, we have use some of the more advanced algorithm supplied with Gmsh. For this task the transfinite algorithm comes in handy.

```
// axis lines (first part)
Transfinite Line {236, 108, 104, 280, 258, 130, 152, 68, 59, 82, 214, 64,
    148, 86, 60, 192, 170} = 41 ❶ Using Progression 1;
```

The numbers inside the curly brackets are references to lines. The first value after the equality sign is the number of mesh lines we want generate. `Using Progression` fallowed by a second value ❶ is for gradually increasing or decreasing the distances[5] between mesh lines as we move towards one of the end points of a `Transfinite Line`[6]. In the example above all the mesh lines are equidistant. The transfinite algorithm needs explanation, and I think it is easiest to illustrate the use with an example. There is math behind the algorithm, but I won't show it here. If we for example we have two line a certain distance from each other, call them *Reference line* ① and *Reference line* ②. Take a look a Figure (2.7a). And you want a uniform mesh splitting of X lines between *Reference line* ① end *Reference line* ②. Then the transfinite algorithm will draw X lines (equidistant) between *Reference line* ① and *Reference line* ②. The shapes and sizes of these X lines will depend on how close to a specific *Reference line* we are. See Figure (2.7b). So basically the transfinite algorithm sets up a set of lines and you have a gradual transformation in the the shapes of these lines from *Reference line* ① to *Reference line* ②. For a closed curve of four lines, two and two *Reference lines* are pared together with the transfinite algorithm. The result becomes like in Figure (2.7c).

The meshing instruction code for lines in radial direction from the pipe "inner" radius to the pipe wall[7] is

```
 // 'outer' radial lines
Transfinite Line {13, 14, 15, 16, 17, 18, 19, 20, 230, 208, 186, 164, 142,
    -144, 274, 252, 406, -408, -430, -452, -474, -496, -518, -540, -694,
    -716, -738, -760, -782, -804, 670, -672, -958, -980, -1002, -1024,
    -1046, -1068, 934, -936, -1200, -1222, -1244, -1266, -1288, -1310,
    -1332, 1198} = N2 Using Progression 0.9;
```

The minus sign in front of some numbers is for reversing the progression direction of the transfinite algorithm. The progression direction depends on the how a line is defined. For example if the line is defined as `Line (1) = {1, 2}` or `Line (1) = {2, 1}`.

The code for the "inner" octagon lines ❷ and lines from the pipe center and out to the pipe "inner" radius ❸ is

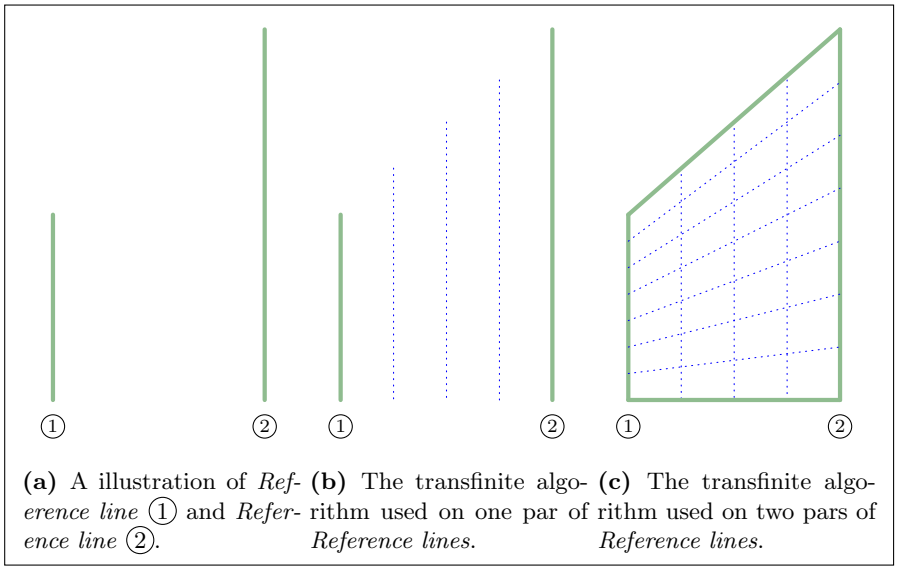
```
// octave lines
❷ Transfinite Line {5, 6, 7, 8, 9, 10, 11, 12, 99, 100, 77, 78, 55, 56, 121,
    122, 319, 320, 341, 342, 363, 364, 385, 386, 584, 605, 606, 627, 628,
    649, 650, 583, 848, 869, 870, 891, 892, 913,914, 847, 1112, 1133, 1134,
    1155, 1156, 1177, 1178, 1111} = N1 Using Progression 1;

// 'inner' radial lines
```

[5] Increasing for $value > 1$ and decreasing $value < 1$

[6] A line we have applied the transfinite algorithm on

[7] See Figure(2.3).



(a) A illustration of *Reference line* ① and *Reference line* ②.

(b) The transfinite algorithm used on one par of *Reference lines.*

(c) The transfinite algorithm used on two pars of *Reference lines.*

Figure 2.7: Illustration of the transfinite algorithm.

❸
```
Transfinite Line {1, 2, 3, 4, 98, 76, 54, 57, 318, 321, 343, 365, 585, 607,
    629, 582, 849, 871, 893, 846, 1113, 1135, 1157, 1110} = N1 Using
    Progression 1;
```

And the last peace of code that you need to write is

```
Transfinite Surface "*";
Recombine Surface "*";
Transfinite Volume "*";
```

This is for applying the transfinite algorithm on all the surfaces and volumes. The multiplication sign surrounded by the quotation marks means *all*. And explaining very simply, the `Recombine` command in this case, changes element shapes from tetrahedrals to hexahedrals.

In Figures (2.9) and (2.10) you can see pictures of `Mesh-A`. Figure (2.11) and (2.12) are pictures of `Mesh-B`. And in Figures (2.13) and (2.14) you see pictures of `Mesh-D`.

As mentioned above the `.geo` is only a instruction file and not a actual mesh file CFD softwares use. The detailed mesh file Gmsh generates is a `.msh` file. So the final thing you have to do is to generate and save this `.msh` file. This can be done trough the graphical interface supplied Gmsh.

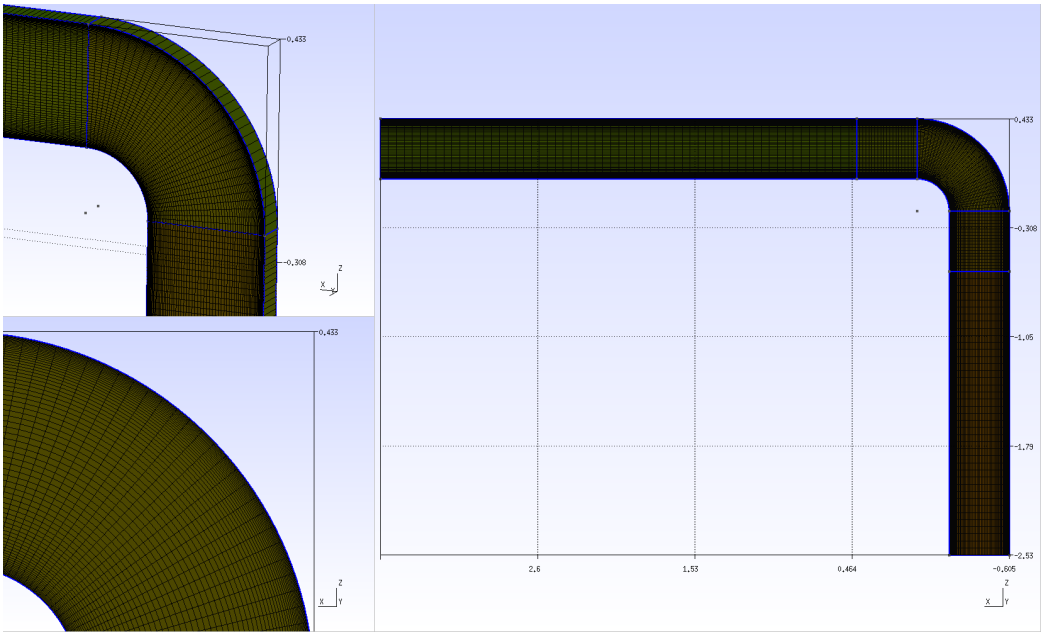


Figure 2.8: In this Figure you see `Mesh-A_2D`. The mesh is composed of 12975 hexahedral elements.

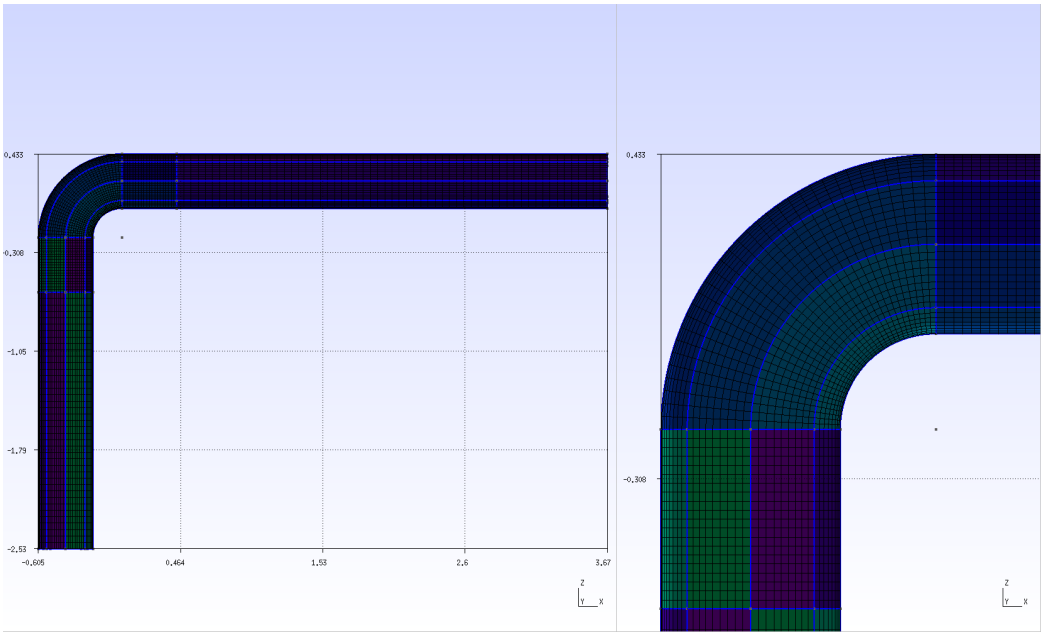


Figure 2.9: `Mesh-A` seen from the side. The mesh is composed of 392364 hexahedral elements.

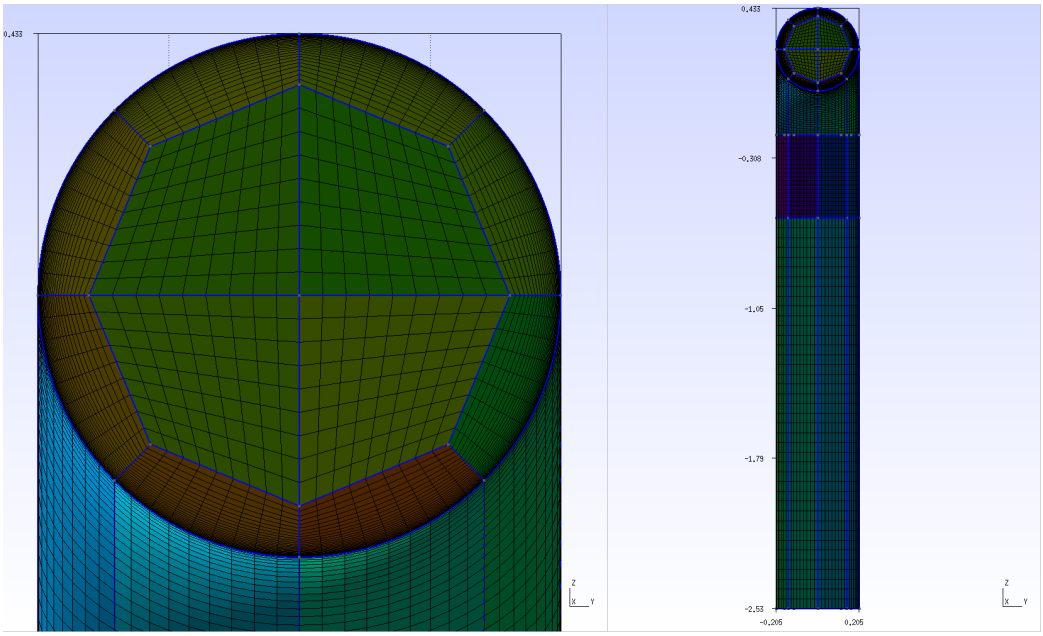


Figure 2.10: In this Figure you see `Mesh-A`. The flat circular surface in the picture to the left is the pipe outlet. The element mesh arrangement in radial direction is the same along the pipe center-axis. Minimum element length in radial direction is $\sim 0.27mm$ for elements at the wall.

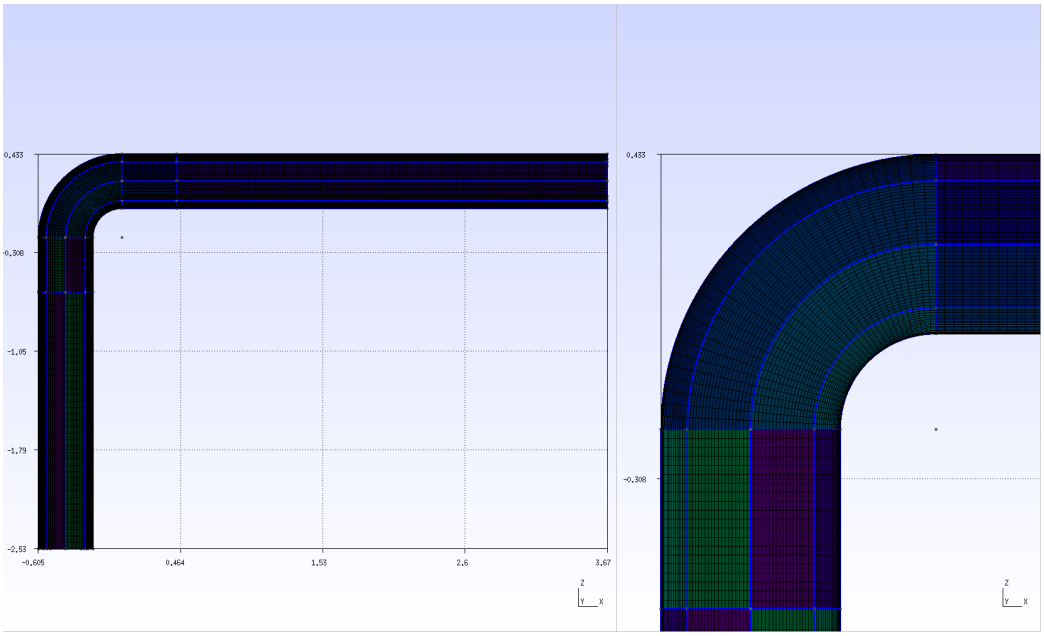


Figure 2.11: `Mesh-B` seen from the side. The mesh is composed of 959804 hexahedral elements.

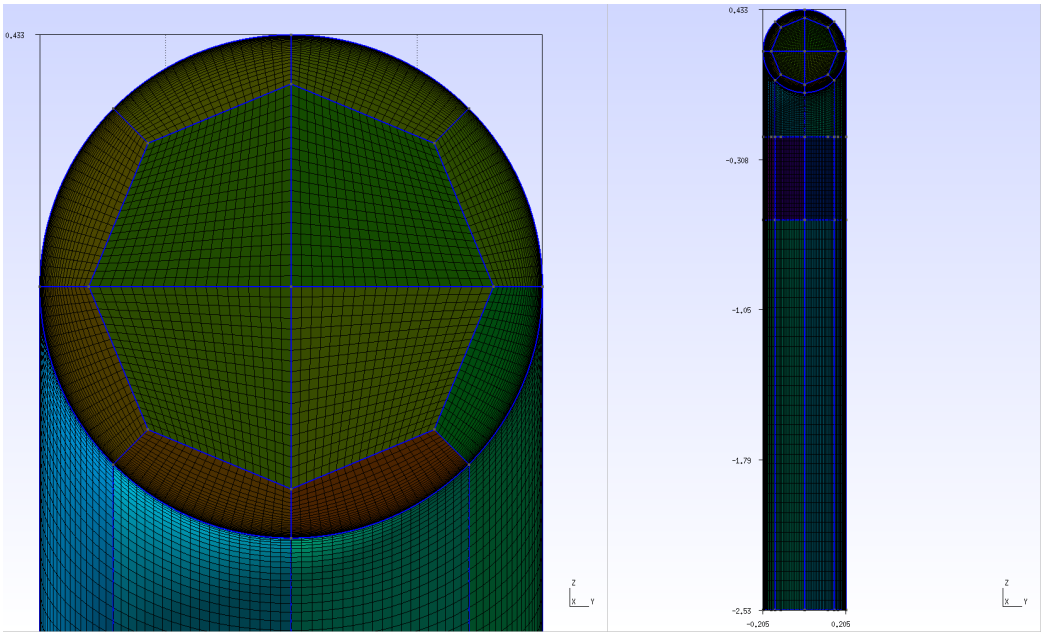


Figure 2.12: A picture of `Mesh-B`. `Mesh-B` has a higher density of elements in radial direction compared to `Mesh-A`. Witch also means that `Mesh-B` has more elements along the circumference of the pipe cross-section. The inner octagon is the same in `Mesh-B` and `Mesh-A`. Minimum element length in radial direction is $\sim 0.27mm$.

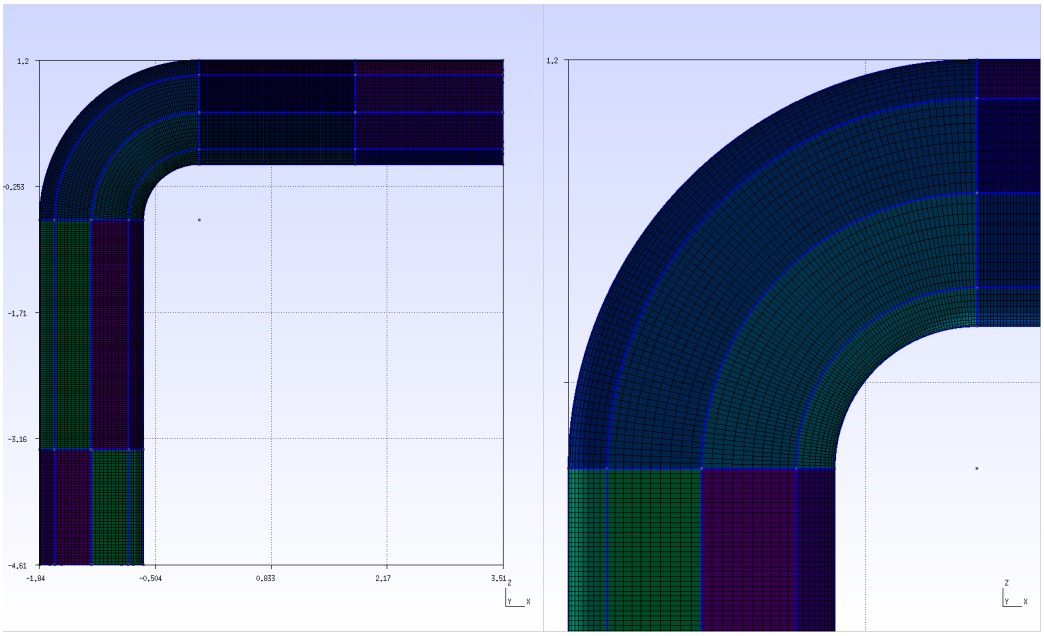


Figure 2.13: `Mesh-D` seen from the side. The mesh is composed of 1082880 hexahedral elements.

Mesh refinement for the different meshes.

	Along the center-axis of the pipe					In a plane normal to the pipe center-axis			Minimum element size in radial direction $[mm]$
Meshes/ Pipe sec.	(1)	(2)	(3)	(4)	(5)	Along the circumference	Inner radial	Outer radial	
`Mesh-A_2D`	40	20	33	20	60	-	9	27	~ 0.27
`Mesh-A`	40	20	33	20	60	9	9	27	~ 0.27
`Mesh-B`	40	20	33	20	60	20	20	27	~ 0.27
`Mesh-D`	40	120	83	120	60	10	10	27	~ 0.8

Table 2.1: In this table you see how the element distribution looks for the different meshes. The values in the table cells, except for the last column, stand for *number of elements*. Mesh `Mesh-A_2D`, which is a 2D mesh, has no radius. The values in the cells bellow *Inner radial* and *Outer radial* simply stand for *number of elements* from the "pipe" center-axis to the "inner" radius and *number of elements* from the "inner" radius to the "pipe-wall". See Figure (2.8) and (2.3).

Mesh	**Number of elements**
`Mesh-A_2D`	12975
`Mesh-A`	392364
`Mesh-B`	959804
`Mesh-D`	1082880

Table 2.2: In this table you see the total number of elements in each mesh. Each of my 3D meshes are bigger then their counterpart in TANAKA et al. [9].

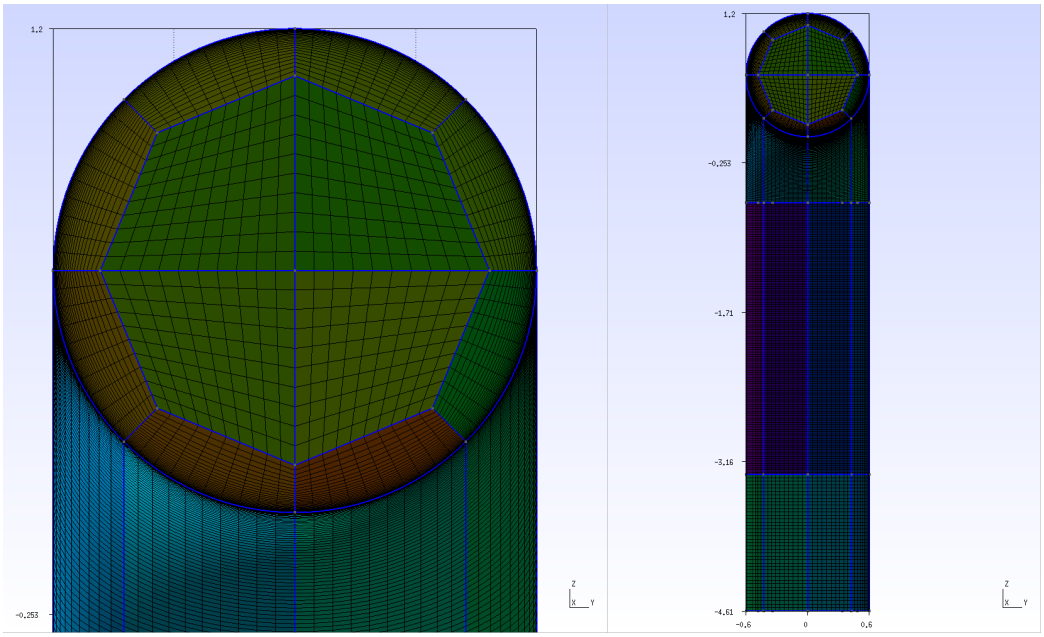


Figure 2.14: A picture of `Mesh-D`. The flat circular surface in the picture to the left is the pipe outlet. Minimum element length in radial direction is $\sim 0.8mm$ for elements at the wall.

Chapter 3

Mathematical models

Instead of solving the full Navier-Stokes equations[1], which is computationally heavy, I am going to use to different turbulence models. The first model will be a U-RANS (Unsteady Reynolds-Averaged Navier-Stokes) model and the second model will be a LES (Large-Eddy Simulation) model. Both models are well-known and used a lot in mechanical engineering communities today. So in this chapter I am going to give a short introduction to the two models and explain how the work. Both models is implemented in OpenFOAM. As mentioned earlier OpenFOAM is the software tool I am going to use for doing the simulations.

Dependent on the complexity of the model, turbulent flows are divided into categories. The complexity is graded after how many assumptions are done in the derivation of the different models. Therefor toping a complexity list will be a "real-life" flow with no simplifications. And at the other end of this list will be something called homogeneous isotropic flows. I have made a illustration that you can look at in Figure (3.1).

And finally before we start I want to inform you that the notation and content of this chapter is based on lecture notes from a course[2] in turbulence modeling at the University of Oslo. The lecture notes themselves is again based on Durbin and Pettersson-Reif [4], which is the course textbook.

3.1 U-RANS model

Let's start with the Navier-Stokes and the continuity equations

$$\partial_t u_i + u_j \partial_j u_i = -\frac{1}{\rho}\partial_i p + \nu \partial^2_{kk} u_i, \tag{3.1}$$

$$\partial_i u_i = 0. \tag{3.2}$$

As you probably know, incompressible flows are governed by these two equations. There is two things I want to inform the reader about before we continue. The first is that the equations above are on the standard index notation form. I assume that the reader is

[1] Doing fluid flow simulations by solving the full Navier-Stokes equations is often called doing DNS (Direct Numerical Simulations). DNS requires a very fine time and grid resolution refinement. Meaning that all spatial and temporal scales must be solved

[2] UNIK4900 - Advanced Turbulence Modeling and Simulations. http://www.uio.no/studier/emner/matnat/math/UNIK4900/index-eng.html. I took the course autumn 2013

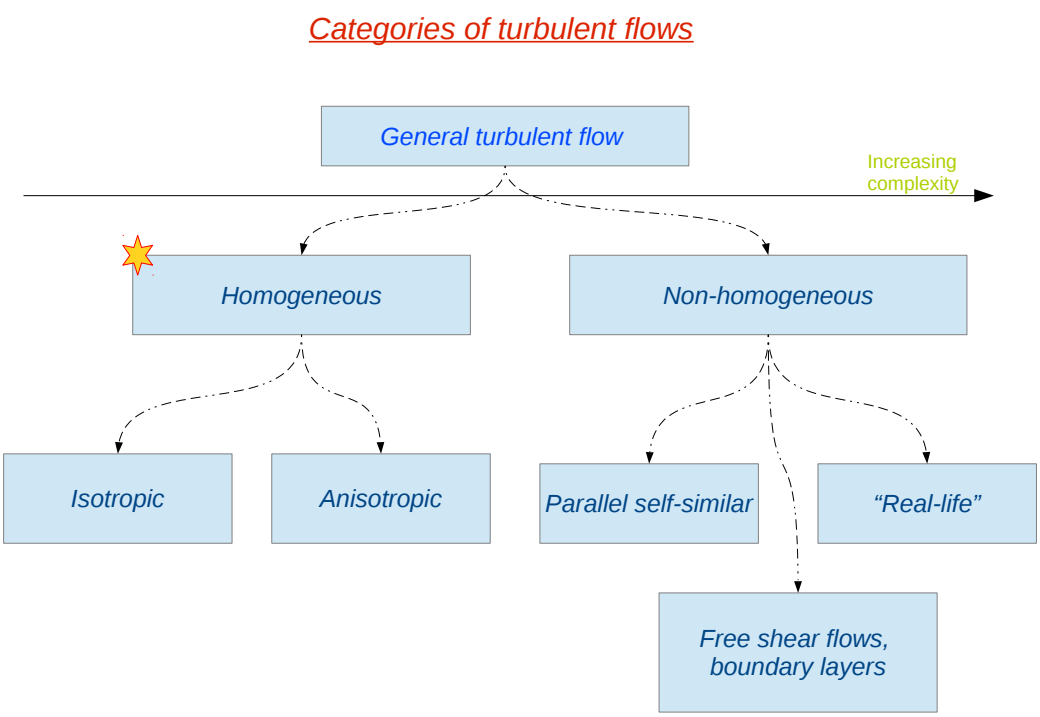


Figure 3.1: Categories of turbulent flows.

familiar with this notation type and will not explain the details of how it works here. And the second, which you probably already have figured out, is the meaning of the partial derivative terms:

$$\partial_t = \frac{\partial}{\partial t}, \quad \partial_i = \frac{\partial}{\partial x_i} \quad \text{and} \quad \partial^2_{kj} = \frac{\partial^2}{\partial x_k \partial x_j}. \tag{3.3}$$

You have probably heard about RANS (Reynolds Averaged Navier-Stokes) equations. So how is the U-RANS equations different from the RANS equations? Well, mathematically they aren't! The equations used for a U-RANS model is the same as in RANS model. The reason why the names are different has to do with the way the numerical simulation is done. When doing a U-RANS type of simulation the transient term in the RANS Equation (3.8):

$$\partial_t U_i$$

is discretized and starting from the initial condition the program moves forward in time using a small time step Δt until it reaches a end time where you have a steady-state solution. On the other hand when using e RANS model you jump directly to the final steady state solution.

The first step consist of decomposing the instantaneous velocity and pressure into two parts/components:

$$\underbrace{u_i(\underline{x}, t)}_{\text{instantaneous component}} = \underbrace{U_i(\underline{x}, t)}_{\text{mean component}} + \underbrace{u_i'(\underline{x}, t)}_{\text{fluctuating component}}, \tag{3.4}$$

$$\underbrace{p(\underline{x},t)}_{\text{instantaneous component}} = \underbrace{P(\underline{x},t)}_{\text{mean component}} + \underbrace{p'(\underline{x},t)}_{\text{fluctuating component}}, \tag{3.5}$$

where "mean" = ensemble average and $\underline{x} = \{x, y, z\}$ is the spatial position. The decomposition above is called a 'Reynolds decomposition' and is named after Osbourne Reynolds (1881). Next we substitute decomposed form of u_i and p from Equations (3.4) and (3.5) into Equations (3.1) and (3.2).

$$\partial_t(U_i + u_i') + (U_j + u_j')\partial_j(U_i + u_i') = -\frac{1}{\rho}\partial_i(P + p') + \nu\partial_{kk}^2(U_i + u_i'), \tag{3.6}$$

$$\partial_i(U_i + u_i') = 0. \tag{3.7}$$

To arrive at the RANS equations you have to take the ensemble average of Equations (3.6) and (3.7). The detailed derivation of the RANS equations consists of several steps where you have to use rules for ensemble averages of sums, derivatives and products. The derivation is not very difficult, but may take little bit of time. I don't see any point in showing it here and will therefor just referee this to [2][3]. Some literature covering turbulence modeling include the detailed procedure of Reynolds-averaging the Navier-Stokes and continuity equations.

Reynolds-averaged Navier–Stokes equations

$$\partial_t U_i + U_j\partial_j U_i = -\frac{1}{\rho}\partial_i P + \nu\partial_{kk}^2 U_i - \partial_j\overline{u_i'u_j'}, \tag{3.8}$$

$$\partial_i U_i = 0. \tag{3.9}$$

$\overline{u_i'u_j'} = \overline{u_i'u_j'}(\underline{x},t)$ in the last term of Equation (3.8) is called the *Kinematic Reynolds Stress Tensor*. And if you extend this term with the fluid density ρ you get the *Reynolds Stress Tensor* $\rho\overline{u_i'u_j'}$. Notice that the total number of unknowns in Equation (3.8) and (3.9) equals ten. Three are the velocity components U_x U_y and U_z, you have the pressure P and finally you have six unknowns from the Reynolds stress term $\overline{u_i'u_j'}$. The total number of equations, with Equation (3.8) being a vector equation, is four. So we have to many unknowns compared to equations.

3.1.1 Eddy-viscosity based models

The $\overline{u_i'u_j'}$ term in RANS equation (3.8) has to be modeled. So the idea is that instead of finding $\overline{u_i'u_j'}$ by solving a transport equation like Equation (3.14), we substitute $\overline{u_i'u_j'}$ by a expression consisting of known variables. The variable of choice is the *Mean Rate of Strain*:

$$S_{ij} = \frac{1}{2}(\partial_i U_j + \partial_j U_i).$$

Meaning that we want to replace the $\overline{u_i'u_j'}$ term by a function of S_{ij},

$$\overline{u_i'u_j'} = f(S_{ij}). \tag{3.10}$$

[3]The more exact location at Wikipedia is: http://en.wikipedia.org/wiki/Reynolds_stress

The dimension of $\overline{u_i' u_j'}$ is

$$\left[\frac{m^2}{s^2}\right] = \underbrace{\left[\frac{m^2}{s}\right]}_{(1)} \cdot \overbrace{\left[\frac{1}{s}\right]}^{(2)},$$

where term (1) has the same dimension as the kinematic viscosity ν and term (2) has the same dimension as the *Mean Rate of Strain* S_{ij}.

We continue by making an ansatz

$$\begin{aligned} \overline{u_i' u_j'} = f(\delta_{ij}, S_{ij}) &= \alpha\delta_{ij} + \beta S_{ij} + \gamma\delta_{ik}S_{kj} \\ &= \alpha\delta_{ij} + \beta S_{ij}. \end{aligned} \tag{3.11}$$

Above δ_{ij} is the ***Kronecker delta*** defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases},$$

and α and β are scalar variables that have to be decided. The last term $\gamma\delta_{ik}S_{kj}$ on the first line (in the function for $\overline{u_i' u_j'}$) is redundant and therefor been included into the βS_{ij} term.

Definition 1: Turbulence kinetic energy k

$$k = \frac{1}{2}\overline{u_i' u_i'} = \frac{1}{2}(\overline{u_1'^2} + \overline{u_2'^2} + \overline{u_3'^2})$$

Using the definition of the turbulence kinetic energy we can find a expression for for α by setting Equation (3.11), for $i = j$, equal to $2k$.

$$\begin{aligned} \overline{u_i' u_i'} &= \alpha\delta_{ii} + \beta S_{ii} \\ 2k &= 3\alpha + \beta \underbrace{(\partial_1 U_1 + \partial_2 U_2 + \partial_3 U_3)}_{\text{= 0, since we are working with incompressible flows}}, \end{aligned}$$

resulting in $\alpha = 2/3k$.

The handling of the β variable is done by substituting a expression for it. Since the dimension of β is $[m^2/s]$, the same as the kinematic viscosity, the suggestion was to set

$$\beta = -2\nu_T,$$

where ν_T is the *eddy viscosity*.

$$\boxed{\overline{u_i' u_j'} = \frac{2}{3}k\delta_{ij} - 2\nu_T S_{ij}} \tag{3.12}$$

3.1.2 Some exact transport equations

I here will just list up a set of equations governing the transport of the different variables, and explain very short how you can derive them. The purpose of this list is for referencing.

Transport equations for different variables

Transport equation for the fluctuating velocity field u_i':

$$\partial_t u_i' + U_k \partial_k u_i' + u_k' \partial_k U_i + \partial_k (u_k' u_i' + \overline{u_k' u_i'}) = -\frac{1}{\rho} \partial_i p' + \nu \partial_{kk}^2 u_i' \tag{3.13}$$

The transport equations for the *Reynolds stress tensor* $\overline{u_i' u_j'}$:

$$\begin{aligned} \partial_t \overline{u_i' u_j'} + u_k' \partial_k \overline{u_i' u_j'} = &-\frac{1}{\rho} (\overline{u_j' \partial_i p'}) - \frac{1}{\rho} (\overline{u_i' \partial_j p'}) - 2\nu (\overline{\partial_k u_i' \partial_k u_j'}) \\ &- \partial_k (\overline{u_k' u_i' u_j'}) - \overline{u_j' u_k'} \partial_k U_i - \overline{u_i' u_k'} \partial_k U_j + \nu \partial_{kk}^2 \overline{u_i' u_j'} \end{aligned} \tag{3.14}$$

Transport equation for the *Turbulence kinetic energy* k:

$$\begin{aligned} \partial_t k + U_i \partial_i k = &-\frac{1}{\rho} \partial_i \overline{u_i' p'} - \nu \overline{\partial_k u_i' \partial_k u_i'} - \frac{1}{2} \partial_k \overline{u_k' u_i' u_i'} - \overline{u_i' u_k'} \partial_k U_i \\ &+ \nu \partial_{ii}^2 k \end{aligned} \tag{3.15}$$

Transport equation for the *Mean kinetic energy* K:

$$\begin{aligned} \partial_t K + U_j \partial_j K = &-\frac{1}{\rho} U_i \partial_i P + \nu \partial_{kk}^2 K + \nu \partial_k U_i \partial_k U_i - \partial_k (U_i \overline{u_k' u_i'}) \\ &+ \overline{u_k' u_i'} \partial_k U_i \end{aligned} \tag{3.16}$$

Denote Equations (3.1) and (3.8) as

$$L(u_i) = 0 \quad \text{(Navier-Stokes eq.)} \quad \text{and} \quad L(U_i) = 0 \quad \text{(RANS eq.)}, \tag{3.17}$$

then the transport equation for the fluctuation velocity field can be derived as

$$L(u_i') = L(u_i) - L(U_i). \tag{3.18}$$

The full equation is Equation (3.13). For arriving at the Reynolds stress transport equation you have to set

$$L(u_i' u_j') = u_j' L(u_i') + u_i' L(u_j') \tag{3.19}$$

and then take the average of the right hand side of the equality sign.,

$$L(\overline{u_i' u_j'}) = \overline{u_j' L(u_i') + u_i' L(u_j')}. \tag{3.20}$$

Getting Equation (3.14). Using Definition (3.1.1) the transport equation for the turbulence kinetic energy can be derived from Equation (3.19) by setting $i = j$ and then dividing by 2. Equation (3.15) is the full form.

3.1.2.1 The k - ε turbulence model

The k - ε model is one of the eddy-viscosity based models. Their are other models, but the the k - ε model is by far the most famous one. It is the most widely used general purpose turbulence model there is, and the "standard" k - ε model was introduced by Jones & Launder in 1977.

A often used approximation for the *eddy viscosity* is

$$\nu_T = C_\mu \frac{k^2}{\varepsilon} \tag{3.21}$$

where ε is the *rate of viscous dissipation* of turbulence kinetic energy k[4] and C_μ is a constant. So in order to decide ν_T, and $\overline{u_i' u_j'}$, we need to find the k and ε fields. We have the exact transport equation for k, Equation (3.15). But rather then working with it, we will derive a template witch can be used to make new transport equations for k and ε. This is done roughly in four steps:

① Rewrite

$$\frac{\partial}{\partial t}\left(\frac{k}{\varepsilon}\right) = \frac{1}{\varepsilon}\partial_t k - \frac{k}{\varepsilon^2}\partial_t \varepsilon \tag{3.22}$$

② Set Equation (3.22) equal to 0 and get

$$\partial_t \varepsilon = \frac{\varepsilon}{k}\partial_t k \tag{3.23}$$

③ Insert for $\partial_t k = P_k - \varepsilon$[5] into Equation (3.23)

$$\frac{\partial \varepsilon}{\partial t} = \frac{P_k - \varepsilon}{k/\varepsilon} \tag{3.24}$$

(P_k is the production of turbulence kinetic energy)

④ Use Equation (3.24) as a template

Equation (3.28) and (3.29) are the new model equations for k and ε.

We have now arrived at the k-ε model. In the standard k-ε model there is six equations, Equations (3.25)-(3.30). You also have six unknowns that need initial and boundary conditions. And there is some model constant that have standard values.

Finally one little comment for the model equation of P_k, Equation (3.30). P_k is actually the fourth term, to the right for the equality sign, in Equation (3.15). To arrive at the form that you can see in Equation (3.30), you have to use that the flow is incompressible.

[4] The choice of approximation for the *eddy viscosity* is partially based on dimensional arguments. The dimension of k is $[m^2/s^2]$ and the dimension of ε is $[m^2/s^3]$

[5] $\partial_t k = P_k - \varepsilon$ simply says that the change in turbulence kinetic energy is equal to the production of k minus the amount of k turn into heat

$C_{\varepsilon 1}$	$C_{\varepsilon 2}$	σ_ε	C_μ
1.44	1.92	1.30	0.09

Table 3.1: k - ε model coefficients.

Standard k - ε model

$$\partial_t U_i + U_j \partial_j U_i = -\frac{1}{\rho}\partial_i P + \nu \partial^2_{kk} U_i - \partial_j \overline{u_i' u_j'}, \tag{3.25}$$

$$\overline{u_i' u_j'} = \frac{2}{3}k - \nu_T 2 S_{ij} \tag{3.26}$$

$$\nu_T = C_\mu \frac{k^2}{\varepsilon} \tag{3.27}$$

$$\frac{Dk}{Dt} = P_k - \varepsilon - \nu \partial^2_{jj} k + \partial_m \left[\nu_T \partial k\right] \tag{3.28}$$

$$\frac{D\varepsilon}{Dt} = \frac{\varepsilon}{k}\left(C_{\varepsilon 1} P_k - C_{\varepsilon 2}\varepsilon\right) + \nu \partial^2_{jj}\varepsilon + \partial_j \left[\nu_T \partial_j \varepsilon\right] \tag{3.29}$$

$$P_k = 2\nu_T S_{ij} S_{ij} \tag{3.30}$$

The values of the model constants in Table (3.1) is based on results arrived from experiments on a wide range of turbulent flows.

3.2 LES model

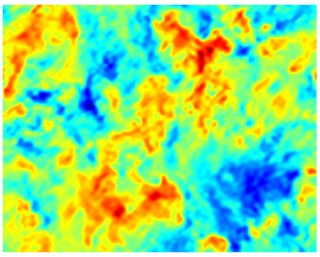
The LES (Large Eddy Simulation) mathematical turbulence model was first introduced in 1963 by Joseph Smagorinsky. It is, like the RANS model, a very popular turbulence model and used on a wide arrange of problems were you need to model turbulent fluid motion. Examples are combustion, acoustics, and simulations of the atmospheric boundary layer.

When doing DNS you resolve all the different length scales, making the simulation a very time-consuming and heavy procedure[6]. So the main idea behind LES modeling is to resolve length scales down to a certain size and model the remaining (smaller) length scales. And in this way get a simulation which is much smaller in size. Take a look at Figure (3.2)[7], maybe it makes things more clear.

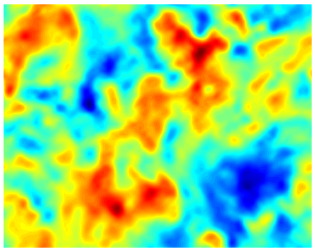
The removing of these smaller length scales happen by using something called a low-pass filter. You can preform a filtering operation in time (temporal filtering), space (spacial

[6] By "heavy" I mean that the total number of calculations, because of a very fine mesh resolution, is very big

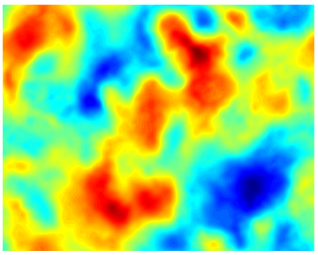
[7] The images are taken from Wikipedia: http://en.wikipedia.org/wiki/Large_eddy_simulation



(a) A velocity field produced by Direct Numerical Simulation (DNS).



(b) The same DNS velocity field filtered using a box filter with $\Delta = L/32$.



(c) The same DNS velocity field filtered using a box filter with $\Delta = L/16$.

Figure 3.2: Three pictures of the same velocity field. No low-pass filters have been used in Figure (3.2a). In Figure (3.2b) a filter removing the smallest and most high frequent velocity fluctuations have been used. And as you can see in Figure (3.2c) even lower frequencies of velocity fluctuations are removed (compared to Figure (3.2b)).

filtering), or both. But since I'm only going to use a spacial filtering in this thesis, I will only list up the definition for that:

Definition 2: Spatial filtering operation by means of a filter function $G(\mathbf{x}, \mathbf{x}', \Delta)$

$$\bar{\phi}(\mathbf{x}, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\mathbf{x}, \mathbf{x}', \Delta)\phi(\mathbf{x}', t)\, dx_1' dx_2' dx_3',$$

where $G(\mathbf{x}, \mathbf{x}', \Delta)$ is the filter kernel (filter function), $\bar{\phi}(\mathbf{x}, t)$ is the filtered function, $\phi(\mathbf{x}, t)$ is the unfiltered function and Δ is the cutoff width.

The way filters preform the selection, of what is kept and what is removed, is by using something called a cutoff width Δ. For a turbulent flow containing eddies [8], the size of these eddies are check against Δ. If the size of a certain eddie is smaller then Δ, then this eddie will be removed. So after the filtering process is done, only eddies of "desirable" sizes are left. The way the filter is implemented is by filtering the whole Navier-Stokes and continuity equations

$$\partial_t u_i + u_j \partial_j u_i = -\frac{1}{\rho}\partial_i p + \nu \partial_{kk}^2 u_i, \tag{3.31}$$

$$\partial_i u_i = 0. \tag{3.32}$$

There are many types of filtering functions with there own special area of use. Some good for theoretical and analysis work, while others are good for certain types of numerical methods, like for example the finite volume- and spectral methods. For the finite volume method with LES we need to use something called box filter defined as:

[8] Eddies are vortex like structures

Definition 3: Box filter (top hat filter)

$$G(\mathbf{x}, \mathbf{x}', \Delta) = \begin{cases} 1/\Delta^3 & |\mathbf{x} - \mathbf{x}'| \leq \Delta/2 \\ 0 & |\mathbf{x} - \mathbf{x}'| > \Delta/2 \end{cases}$$

With Definition (**??**) it is know possible to preform a splitting of the pressure p and velocity u_i fields

$$\underbrace{u_i(\underline{\mathrm{x}}, t)}_{\text{original velocity}} = \underbrace{\overline{u_i}(\underline{\mathrm{x}}, t)}_{\text{retained velocity}} + \underbrace{u_i'(\underline{\mathrm{x}}, t)}_{\text{rejected velocity}}, \tag{3.33}$$

$$\underbrace{p(\underline{\mathrm{x}}, t)}_{\text{original pressure}} = \underbrace{\overline{p}(\underline{\mathrm{x}}, t)}_{\text{retained pressure}} + \underbrace{p'(\underline{\mathrm{x}}, t)}_{\text{rejected pressure}}, \tag{3.34}$$

Almost like the Reynolds decomposition in Section (3.1).

The eddies kind of "depend" on each other. And by that I mean eddies of all sizes interact. The interaction happens by one eddie affecting surrounding eddies and other flow structures with forces, stresses, etc..So when filtering out the eddies which are smaller in size then Δ, we get a problem. The problem is that we are missing the interaction between eddies on the two sides of Δ. The missing interaction must be replaced some how. The effect of the eddies smaller in size then Δ has to modeled. And here is where the SGS (***sub-grid-scale stresses***) models come in.

3.3 Formulas for internal fields and boundaries

Below I have listed up some useful formulas that can be used for deciding internal and boundary values of different variables. Several of the formulas are only approximations and not exact definitions.

I found the formulas at a website of University of California Davis: `http://aerojet.engr.ucdavis.edu/fluenthelp/html/ug/node217.htm`

The turbulence intensity is defined as

$$I = \frac{u'}{u_{avg}} = 0.16 Re_{D_H}^{-1/8}, \tag{3.35}$$

where u_{avg} is the mean flow velocity and Re_{D_H} is the Reynolds number based on the pipe hydraulic diameter[9]. For single phase circular pipe flows, D_H is the same as the pipe diameter D.

Next we have

$$l = 0.07L. \tag{3.36}$$

The turbulence length scale, l, is a physical quantity related to the size of the large eddies that you can find in turbulent flows. L is the relevant dimension of the pipe[10].

The modified viscosity $\widetilde{\nu}$ can be calculated as

$$\widetilde{\nu} = \sqrt{\frac{3}{2}} u_{avg} I l. \tag{3.37}$$

[9] $Re_{D_H} = \rho \mathbf{v} D_H / \mu$. See: `http://en.wikipedia.org/wiki/Reynolds_number`

[10] For in my simulations $L = D$

And finally some formulas for the kinetic energy k and rate of viscous dissipation ε with the variables mentioned above

$$k = \frac{3}{2}(u_{avg}I)^2 \quad \text{and} \quad \varepsilon = C_\mu \frac{k^{3/2}}{l}.$$

C_μ is one of the four model coefficients in the k-ε turbulence model, See Table (3.1).

Some CFD software uses and another definition for the viscous dissipation

$$\varepsilon = C_\mu^{3/4} \left(\frac{k^{3/2}}{l} \right). \tag{3.38}$$

Chapter 4

Numerical methods

4.1 The finite volume method

In this chapter I am going to give a short introduction to the finite volume method. The finite volume method is simply a method for finding approximate solutions to differential equations and other mathematical problems.

This is the method I am going to use in this thesis for finding a solution to the set of PDE's governing the behavior of turbulent fluid motion. The method is implemented in through the software OpenFOAM[1] .

The examples and content in the different sections below is referred to Versteeg and Malalasekera [11].

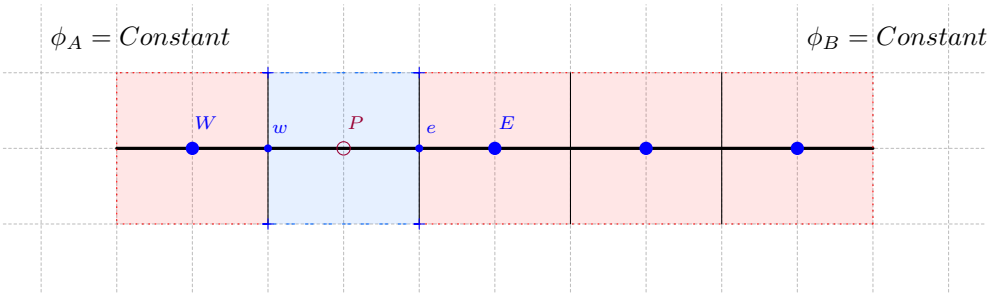


Figure 4.1: A one-dimensional domain divided into 5 control volumes. The blue square represents a general control volume with a node P at its center. W and E represent neighbor nodes and the lower case letters w and e represents control volume faces.

4.1.1 The finite volume method for 1D problems

To illustrate how the finite volume method works in one-dimensional space we will start by looking at a simple ordinary differential equation (ODE) governing the diffusion of some scalar function ϕ. Choosing the domain along the x-axis a diffusion equation would look

[1]OpenFOAM is a open-source computational fluid dynamics software implemented with the finite volume method. www.openfoam.org

like

$$\frac{d}{dx}\left(\gamma\frac{d}{dx}(\phi)\right)+S=0, \tag{4.1}$$

where γ will be the diffusion coefficient and S is a source term. It is quite usual with source terms in differential equations for diffusion and I have chosen to include one in this example. To get a boundary value problem we need to decide the domain and the value of ϕ at the boundaries of the domain. So accompanying Equation (4.1) is a par of boundary conditions which we will call ϕ_A and ϕ_B. The subscript A represents the west (left) boundary and B the east (right) boundary. In Figure (4.1) you can see a illustration of the domain and boundary conditions.

4.1.1.1 Grid generation

The first step in the finite volume method is to divide the domain into a number of control volumes[2], like in Figure (4.1). In Figure (4.1) you see an example of a one-dimensional domain which is divided into 5 pieces. The blue square is a general control volume and the solid vertical lines represent the control volume faces (boundaries). It is very usual that the size and shape of the control volumes vary over the domain[3], but in this example the control volumes will be of uniform length (Δx). In the middle of each control volume we will place a node. Looking just at a general control volume (blue square) the center node for this control volume will be denoted as P and the neighbor nodes as W (west) and E (east). The lower case letters w (west) and e (east) represent the control volume faces. Also notice that the distant between to adjacent nodes is Δx and the distance between a control volume node and one of it's faces is $\Delta x/2$.

4.1.1.2 Formal integration

The key step in the finite volume method is the control volume integration. When doing the integration we simply integrate the governing equations over each control volume in our domain. In this example the control volume integration of equation (4.1) becomes

$$\begin{aligned}\int_{V_c}\frac{d}{dx}\left(\gamma\frac{d}{dx}(\phi)\right)+S\,dV &= \int_{V_c}\frac{d}{dx}\left(\gamma\frac{d}{dx}(\phi)\right)dV+\int_{V_c}S\,dV\\ &= dydz\left[\gamma\frac{d}{dx}(\phi)\right]_e^w+\bar{S}\Delta V\\ &= dydz\underbrace{\gamma\frac{d}{dx}(\phi)\Big|_w}_{(1)}-dydz\underbrace{\gamma\frac{d}{dx}(\phi)\Big|_e}_{(2)}+\underbrace{\bar{S}\Delta V}_{(3)}\end{aligned} \tag{4.2}$$

where $dydz$ is the cross-section area, ΔV is the "volume" of the control volume and $\bar{S}$ is the average of the source over the control volume.

4.1.1.3 Discretization

The next step is the discretization of Equation (4.2). When doing the discretization we turn the governing equations into a useful form which makes us able to solve them numerically.

[2] In some literature control volumes are referred to as elements or cells

[3] Often in fluid flow problems it is normal to have higher mesh resolution close to domain boundaries like e.g. solid walls

In Equation (4.2) we need to find a substitution for terms (1) and (2). Assuming also that the diffusion coefficient γ is a function of x, the value of γ at the control volume faces w and e can be replaced by

$$\gamma_w = \frac{\gamma_W + \gamma_P}{\Delta x}, \tag{4.3}$$

$$\gamma_e = \frac{\gamma_P + \gamma_E}{\Delta x}. \tag{4.4}$$

The equations above are simple linear approximation, but it is also possible to use other type of approximations like for example a cubic approximation witch has a 3 node configuration.

For the gradient terms

$$\left.\frac{d\phi}{dx}\right|_w \quad \text{and} \quad \left.\frac{d\phi}{dx}\right|_e, \tag{4.5}$$

we will use central differencing. When using central differencing we basically set the gradient equal to the difference of ϕ at two adjacent nodes divided by the distance between the nodes. In our example we get that

$$\left.\frac{d\phi}{dx}\right|_w = \frac{\phi_E - \phi_P}{\Delta x} \quad \text{and} \quad \left.\frac{d\phi}{dx}\right|_e = \frac{\phi_P - \phi_W}{\Delta x}. \tag{4.6}$$

For term (3) we can either be kept as $\bar{S}\Delta V$ or be approximated as a linear form

$$\bar{S} = S_u + S_p\phi_P \tag{4.7}$$

Finally we substitute Equations (4.3), (4.4), (4.6) and (4.7) into Equation (4.2)

$$\gamma_e dydz\left(\frac{\phi_E - \phi_P}{\Delta x}\right) - \gamma_w dydz\left(\frac{\phi_P - \phi_W}{\Delta x}\right) + S_u + S_p\phi_P = 0, \tag{4.8}$$

and rearrange to get

$$\underbrace{\left(\frac{\gamma_e}{\Delta x}dydz + \frac{\gamma_w}{\Delta x}dydz - S_p\right)}_{a_P}\phi_P = \underbrace{\left(\frac{\gamma_w}{\Delta x}dydz\right)}_{a_W}\phi_W + \underbrace{\left(\frac{\gamma_e}{\Delta x}dydz\right)}_{a_E}\phi_E + S_u \tag{4.9}$$

4.1.1.4 Solution of equations

Finally the discretized equation(s) is applied to each control volume in our domain, resulting in a linear system of algebraic equations. For control volumes adjacent with the domain boundaries, the boundary conditions incorporated into the discretized equation.

4.1.2 The finite volume method for 2D and 3D problems

The same four steps:

- grid generation
- discretization
- control volume integration

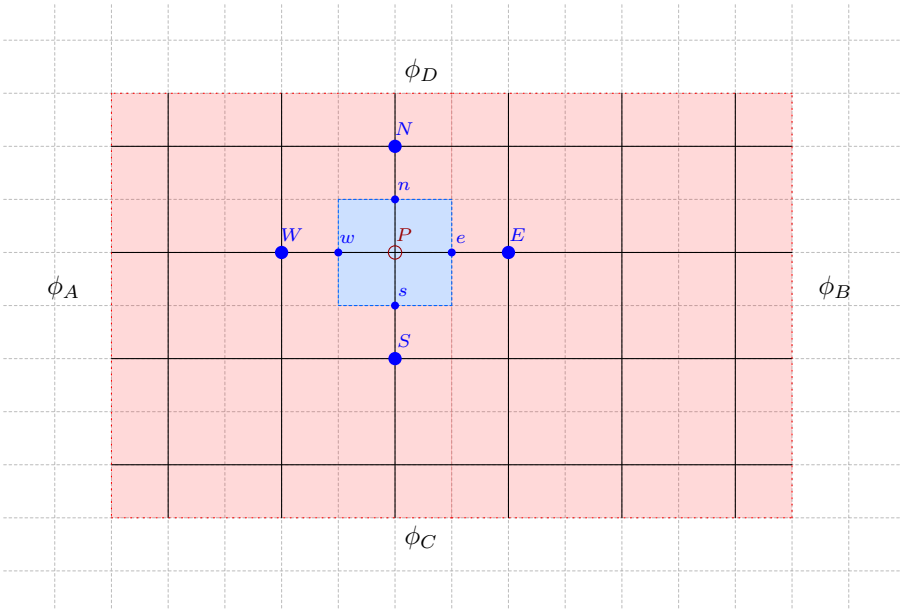


Figure 4.2: A two-dimensional domain. The blue square represents a general control volume with a node P at its center. W (west), E (east), S (south) and N (north) represent neighbor nodes and the lower case letters w, e, s and n represents control volume faces. ϕ_A, ϕ_B, ϕ_C and ϕ_D are the boundary conditions.

- solution of equations

used for solving ODE's (ordinary differential equations) in one-dimensional space is used for solving two-dimensional problems.

A two-dimensional steady state diffusion equation is given by

$$\frac{\partial}{\partial x}\left(\gamma\frac{\partial}{\partial x}(\phi)\right)+\frac{\partial}{\partial y}\left(\gamma\frac{\partial}{\partial y}(\phi)\right)+S=0, \tag{4.10}$$

where again γ is the diffusion coefficient, ϕ is some scalar function and S is the source term. Unlike the one-dimensional problem here γ, ϕ and S can be functions of x and y. In Figure (4.2) you see a example of a two-dimensional domain and grid. The difference compared to a one-dimensional problem is that a general control volume with a node P at it's center is has neighbor nodes and faces also in the y-direction. The extra nodes are denoted as N (north) and S (south) and the extra faces with lower case letters as n (north) and s (south). The grid refinement in the y-direction and the distance between nodes and faces is similar to that of the previous one-dimensional example. Meaning that the distance between two adjacent nodes in y-direction will be Δy and the distance between a node and one of it's faces is $\Delta y/2$. Below I will just list up the results, because the there is no need for much explantion. The calculations, discretization and substitutions are straight forward to do.

$$\int_{V_c}\frac{\partial}{\partial x}\left(\gamma\frac{\partial}{\partial x}(\phi)\right)\,dV+\int_{V_c}\frac{\partial}{\partial y}\left(\gamma\frac{\partial}{\partial y}(\phi)\right)\,dV+\int_{V_c}S\,dV=0 \tag{4.11}$$

$$\left[\gamma_e \Delta y \left(\frac{\partial \phi}{\partial x}\right)_e - \gamma_w \Delta y \left(\frac{\partial \phi}{\partial x}\right)_w\right] + \left[\gamma_n \Delta x \left(\frac{\partial \phi}{\partial y}\right)_n - \gamma_s \Delta x \left(\frac{\partial \phi}{\partial y}\right)_s\right] + \bar{S}\Delta V = 0 \quad (4.12)$$

$$\gamma_w \Delta y \left.\frac{\partial \phi}{\partial x}\right|_w = \gamma_w \Delta y \frac{(\phi_P - \phi_W)}{\Delta x} \quad \text{and} \quad \gamma_e \Delta y \left.\frac{\partial \phi}{\partial x}\right|_e = \gamma_e \Delta y \frac{(\phi_E - \phi_P)}{\Delta x} \quad (4.13)$$

$$\gamma_s \Delta x \left.\frac{\partial \phi}{\partial y}\right|_s = \gamma_s \Delta x \frac{(\phi_P - \phi_S)}{\Delta y} \quad \text{and} \quad \gamma_n \Delta x \left.\frac{\partial \phi}{\partial x}\right|_n = \gamma_n \Delta x \frac{(\phi_N - \phi_P)}{\Delta y} \quad (4.14)$$

$$\gamma_e \Delta y \frac{(\phi_E - \phi_P)}{\Delta x} - \gamma_w \Delta y \frac{(\phi_P - \phi_W)}{\Delta x} + \gamma_n \Delta x \frac{(\phi_N - \phi_P)}{\Delta y} - \gamma_s \Delta x \frac{(\phi_P - \phi_S)}{\Delta y} + \bar{S}\Delta V = 0 \quad (4.15)$$

$$\bar{S}\Delta V = S_u + S_p \phi_P \quad (4.16)$$

$$\underbrace{\left(\frac{\gamma_w \Delta y}{\Delta x} + \frac{\gamma_e \Delta y}{\Delta x} + \frac{\gamma_s \Delta x}{\Delta y} + \frac{\gamma_n \Delta x}{\Delta y} - S_p\right)}_{a_P} \phi_P = \underbrace{\left(\frac{\gamma_w \Delta y}{\Delta x}\right)}_{a_W} \phi_W + \underbrace{\left(\frac{\gamma_e \Delta y}{\Delta x}\right)}_{a_E} \phi_E + \underbrace{\left(\frac{\gamma_s \Delta x}{\Delta y}\right)}_{a_S} \phi_S + \underbrace{\left(\frac{\gamma_n \Delta x}{\Delta y}\right)}_{a_N} \phi_N + S_u \quad (4.17)$$

$$\frac{\partial}{\partial x}\left(\gamma \frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\gamma \frac{\partial \phi}{\partial y}\right) + \frac{\partial}{\partial z}\left(\gamma \frac{\partial \phi}{\partial z}\right) + S = 0 \quad (4.18)$$

$$\left[\gamma_e A_e \left(\frac{\partial \phi}{\partial x}\right)_e - \gamma_w A_w \left(\frac{\partial \phi}{\partial x}\right)_w\right] + \left[\gamma_n A_n \left(\frac{\partial \phi}{\partial y}\right)_n - \gamma_s A_s \left(\frac{\partial \phi}{\partial y}\right)_s\right] + \left[\gamma_t A_t \left(\frac{\partial \phi}{\partial z}\right)_t - \gamma_b A_b \left(\frac{\partial \phi}{\partial z}\right)_b\right] + \bar{S}\Delta V = 0 \quad (4.19)$$

$$\left[\gamma_e A_e \frac{\phi_E - \phi_P}{\Delta x} - \gamma_w A_w \frac{\phi_P - \phi_W}{\Delta x}\right] + \left[\gamma_n A_n \frac{\phi_N - \phi_P}{\Delta y} - \gamma_s A_s \frac{\phi_P - \phi_S}{\Delta y}\right] + \left[\gamma_t A_t \frac{\phi_T - \phi_P}{\Delta z} - \gamma_b A_b \frac{\phi_P - \phi_B}{\Delta z}\right] + (S_u + S_P \phi_P) = 0 \quad (4.20)$$

$$a_P \phi_P = a_w \phi_W + a_E \phi_E + a_S \phi_S + a_N \phi_N + a_B \phi_B + a_T \phi_T + S_u$$

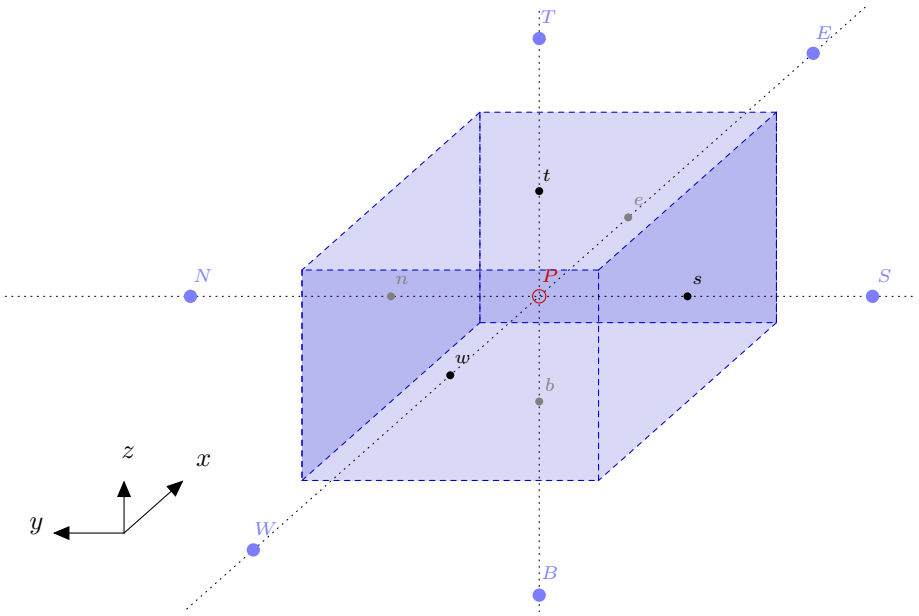


Figure 4.3: Here you see an example of a three dimensional control volume with a node P at its center. W (west), E (east), S (south), N (north), B (bottom) and T (top) are neighbor nodes and the lower case letters w, e, s, n, b and t are the control volume faces.

Chapter 5

OpenFOAM implementation

In OpenFOAM a simulation is run by making a `<case>` directory, then starting the simulation by executing a series of statements in a terminal window[1].

A general `<case>` directory for a incompressible flow looks like in Figure (5.1). The `<case>` directory has a tree structure with subdirectories and files. The information[2] you have to set in order to preform the simulation, are organized into these files[3].

When implementing a new OpenFOAM `<case>`, it is normal practice to copy a existing tutorial `<case>`[4] and make changes to it. Meaning that you change the `<case>` files, by commenting out the parts don't want and add new code, rather then make and write all the folders and files from scratch. So note that when looking through files, in one of the cases I've run, you find more code then what's written personally by me.

A short overview over initial and boundary conditions, for different cases I've implemented, can bee seen in Tables (5.1)-(5.5). Detailed information[5] about each case setup can be found in the `Master-/cases` folder at my Github account: `https://github.com/sayedn/Master-`. All the sub-folders are cases I've done simulations with.

You can roughly divide my simulations into two parts. The first part is a set of test cases with the k-ε turbulence model. The second part is two sets of LES simulations, see Table (5.6). The different between the two sets is implementation of the boundary conditions. Cases having a `-u` ending, like for example `Case-B4-u` belong to the first set of LES simulations. While cases having a `-u_a` ending belong to second set of LES simulations.

In Figure (5.3) and (5.2) you see the case directory structure for a U-RANS simulation and a LES simulation.

[1] There is no graphical interface

[2] The information is things like initial and boundary conditions, fluid properties, discretization of governing equations and etc.

[3] All the files are written in `C++`

[4] OpenFOAM is supplied with a tutorial containing complete cases for different simulations types

[5] By detailed information I mean thing like discretization of therms in the governing equations, values set for turbulence model, etc.

		Patches							
		inlet		outlet		internal		fixedwall	
		type	value [-s]	type	value [-s]	type	value	type	value
variables / files	U	turbulentInlet	See Table (5.2)	inletOutlet	(0 0 0)/(0 0 0)	-	(0 0 0)	fixedValue	(0 0 0)
	p	zeroGradient	-	fixedValue	0	-	0	zeroGradient	-
	nuSgs	zeroGradient	-	zeroGradient	-	-	0	zeroGradient	-
	nuTilda	fixedValue	0	inletOutlet	0/0	-	0	fixedValue	0
	k	fixedValue	See Table (5.2)	inletOutlet	0/0	-	0	fixedValue	0

Table 5.1: This table together with Table (5.2) gives a overview over initial and boundary conditions of the different variables in the LES simulations. - sign means that you don't have to give/set any value or statement. These "names" in the table cells below type is OpenFOAM boundary types. For the inletOutlet boundary type you have to give two values, one is for the inletvalue and the other (after the / sign) is for the normal value.

variables / files										
	U	turbulentInlet	referenceField	(0 0 9.2)	(0 0 9.2)	(0 0 3.08)	(0 0 0.8)	(0 0 9.2)	(0 0 9.2)	(0 0 9.2)
			fluctuationScale	(0.01 0.01 0.03)	"	"	"	"	"	"
			value	(0 0 9.2)	(0 0 9.2)	(0 0 3.08)	(0 0 0.8)	(0 0 9.2)	(0 0 9.2)	(0 0 9.2)
	K	inletOutlet	value	0.08	0.08	0.01	0.01	0.06	0.05	0.04
				B4-u	B7-u	B8-u	B9-u	B10-u	B11-u	D0-u
				Cases						

Table 5.2: This Table is a extension of Table (5.1) above. Here you see what OpenFOAM boundary types and values where used on the inlet boundaries of U and k. The fluctuationScale is the same for all the cases (marked by ").

		Patches							
		inlet		outlet		internal		fixedwall	
		type	value	type	value	type	value	type	value
Variables /files	U	fixedValue	(0 0 9.2)	zeroGradient	-	internalField	(0 0 0)	fixedValue	(0 0 0)
	p	zeroGradient	-	fixedValue	0	internalField	0	zeroGradient	-
	epsilon	fixedValue	0.214	zeroGradient	-	internalField	0.0702|	epsilonWallFunction	0.214
	nut	calculated	0	calculated	0	internalField	0	nutWallFunction	0
	nuTilda	fixedValue	0	zeroGradient	-	internalField	0	zeroGradient	-
	k	tubulentIntesity -KineticEnergyInlet	U/0.05/1	zeroGradient	0	internalField	0	kqRWallFunction	0.103

Table 5.3: In this table you see a overview over initial and boundary conditions for one test I did U-RANS simulations with.

		Patches							
		inlet		outlet		internal		fixedwall	
		type	value [-s]	type	value [-s]	type	value	type	value
Variables / files	U	fixedValue	See Table (5.5)	zeroGradient	-	-	(0 0 0)	fixedValue	(0 0 0)
	p	zeroGradient	-	fixedValue	0	-	0	zeroGradient	-
	nuSgs	zeroGradient	-	zeroGradient	-	-	0	zeroGradient	-
	nuTilda	fixedValue	0	zeroGradient	-	-	0	fixedValue	0
	k	turbulentIntensity -KineticEnergyInlet	U/0.05/1	zeroGradient	-	-	0	fixedValue	0

Table 5.4: This table together with Table (5.5) gives a overview over initial and boundary conditions for the different variables of the second set of LES simulations. - sign means that you don't have to give/set any value or statement. These "names" in the table cells below type is OpenFOAM boundary types.

		type	value					
Variable / file	U	fixedValue	(0 0 9.2)	(0 0 9.2)	(0 0 3.08)	(0 0 0.8)	(0 0 9.2)	(0 0 9.2)	(0 0 9.2)
			B4-u_a	B7-u_a	B8-u_a	B9-u_a	B10-u_a	B11-u_a	D0-u_a
			Cases						

Table 5.5: This Table is a extension of Table (5.4) above. Here you see what OpenFOAM boundary types and values where used on the inlet boundaries of U.

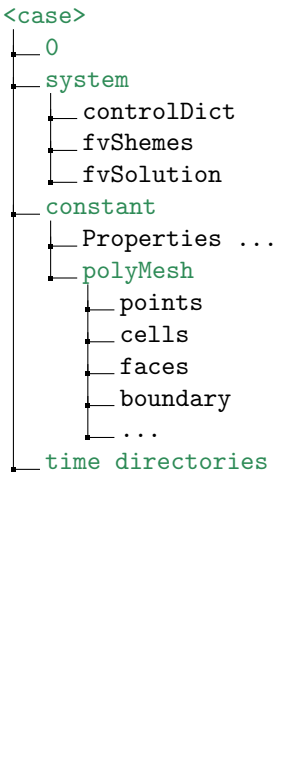


Figure 5.1: General case directory structure.

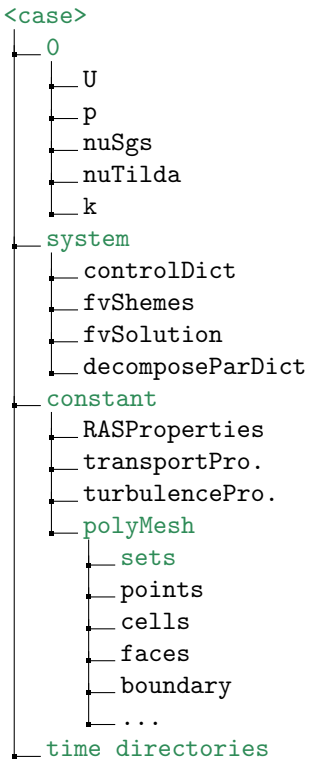


Figure 5.2: Case directory structure for LES simulations.

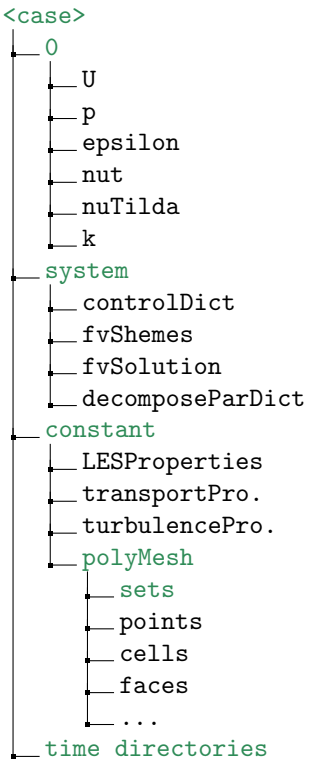


Figure 5.3: Case directory structure for U-RANS simulations.

5.1 Starting and running a OpenFOAM <case>

In OpenFOAM, preforming different tasks is done by executing statements and functions in a terminal window. To do this you have to first be located inside a OpenFoam `<case>` directory or one of its subdirectories. All the commands listed bellow is executed from the `<case>` directory.

Normally the first thing you do is to generate your mesh. And since I made my meshes using Gmsh, I need to generate a mesh compatible with OpenFOAM. The new mesh is based on the information in the `.msh` file. This is done with the command

```
$ gmshToFoam *.msh
```

`*.msh` is some `.msh` file. The new mesh and mesh data will be stored in files in the `constant/polyMesh` directory. See Figure (5.1).

As a optional thing, it is recommended to check the mesh for flaws. You can do this with

```
$ checkMesh
```

`checkMesh` tests different things like geometry and topology.

	My main LES simulations				
	Case	*Mesh*	*Working fluid*	Δt [s]	U_m [m/s]
First set / part	Case-B4-u	Mesh-A	Water at 20°	0.25e-03	9.2
	Case-B7-u	Mesh-B	Water at 20°	0.25e-04	9.2
	Case-B8-u	Mesh-A	Water at 20°	0.25e-03	3.08
	Case-B9-u	Mesh-A	Water at 20°	0.25e-03	0.8
	Case-B10-u	Mesh-A	Water at 60°	0.25e-03	9.2
	Case-B11-u	Mesh-A	Sodium at 550°	0.25e-03	9.2
	Case-D0-u	Mesh-D	Sodium at 550°	0.25e-04	9.2
Second set / part	Case-B4-u_a	Mesh-A	Water at 20°	0.25e-03	9.2
	Case-B7-u_a	Mesh-B	Water at 20°	0.25e-04	9.2
	Case-B8-u_a	Mesh-A	Water at 20°	0.25e-03	3.08
	Case-B9-u_a	Mesh-A	Water at 20°	0.25e-03	0.8
	Case-B10-u_a	Mesh-A	Water at 60°	0.25e-03	9.2
	Case-B11-u_a	Mesh-A	Sodium at 550°	0.25e-03	9.2
	Case-D0-u_a	Mesh-D	Sodium at 550°	0.25e-04	9.2

Table 5.6: In this table you see a list over all the cases I plane to preform LES simulations with. The cases that have a `-u` ending has the setup in Table (5.1), while cases with a `-u_a` ending has the setup in Table (5.4). For all the cases the simulation time length is 5 seconds.

We are know ready to start the simulation. In the `<case>` directory type the name of the solver [6] and press enter.

```
$ pisoFoam
```

As the program is running, folders for different time steps will be created inside the `<case>` directory. Inside these time folder[7] will be files containing field information. The time step between each "writing" is set in the `system/controlDict` file.

The command above was for doing the simulation with a single processing unit. For doing the simulation in parallel, with several CPU's, you have to first split the mesh into peaces. Each processing units will be assigned a mesh peace and a `processor*` folder [8] will be created for that processing unit. Inside the `processor*` folder, simulation results at different time steps will be saved. These results are just for the mesh area assigned a specific processing unit. The first of two commands you have to execute is

```
$ decomposePar
```

`decomposePar` does the mesh splitting and creates the `processor*` folders. The second command is for starting the simulation

```
$ mpirun -np 4   pisoFoam  -parallel
```

If you have done the simulation in parallel, the final thing you have to do is to "glue" together results from all the `processor*` folders. The execution command for this is

[6] You have to use same solver as defined in the `system/controlDict` file. See Figure (5.1)
[7] These time folders are for example named as `0`, `0.1` and so on, if the result "writing" is set to 0.1
[8] These folders are named as `processor0`, `processor1`, ...

```
$ reconstructPar
```

The visualization of the results and post-processing is done mainly with ParaView[9]. And if you like working with VTK files, you can generate them with

```
$ foamToVTK
```

5.2 Simulations on The Abel Computer Cluster

I will here really fast explain how you preform a simulation with OpenFOAM using Abel, short for The Abel Computer Cluster. The way described bellow is one way of of doing this and the way I did it. The content of this section is refereed back to the user-manual of Abel [10].

All communication with Abel, like sending and receiving files, running programs, etc., is done over the Internet using a secure network connection like `ssh`.

Every person who has a user-account on Abel get their own separate disk space. This disk space, witch is the users personal home directory, is where he/she can save files, install programs and do other similar tasks.

The way you preform a "job" on Abel is by applying simple four step:

① starting your program by sourcing the program functions

② next copy relevant files and folders to a special work area, which is on a faster file system

③ their run your simulation

④ and finally copy back the simulation results to your user home directory

All the steps above is preformed by Abel, but you have to handover the information and instructions in a shell script.

Say now you have OpenFOAM installed in your user home directory at Abel, the same way you have it installed on your personal computer. You log on to Abel, enter your OpenFOAM folder and move down to the subdirectory containing your finished OpenFOAM `<case>`[10], create the shell script mentioned above, and submit it to Abel.

The commands and functions you would have executed in a terminal window for running a simulation with OpenFOAM, on your home computer, has to now be placed inside this shell script. This shell script , called a job script, has in addition to the OpenFOAM statements a set of other instructions. These additional instructions can be roughly divided in to 3 categories. The first category is the computational resources[11] needed for the task. The second category is what files and folders you want to copy to the scratch area, and what results and other material you want to copy back to the folder where you submitted the job script. And the final third category is basic shell commands for moving between folders, compressing files, etc. In Appendix (A.2.2) you see a example of a job scripts. This job script was made for `Case-B4`.

---

9 ParaView

10 By finished I mean that OpenFOAM `<case>` is ready to be run. Everything like initial and boundary conditions, solver type, etc., is set

11 By computational resources I mean number of processing units you want to use, maximum simulation time, memory usage per CPU, etc.

Chapter 6

Results

I didn't finished preforming simulations with all the cases I had planed to work on. `Case-B7-u`, `Case-B7-u_a`, `Case-D0-u` and `Case-D0-u_a` has large meshes, with fine mesh refinement close to wall boundary areas, that requires a small time step Δt. A full five second simulation will take more then 96 hours and I didn't have the time for that. So I haven't produced any results for them and their not a part of the discussion in this chapter.

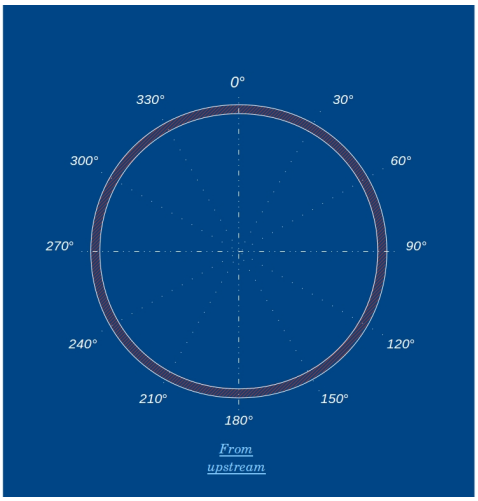
All the results I have will be compared against results in TANAKA et al. [9].

6.1 The U-RANS simulations

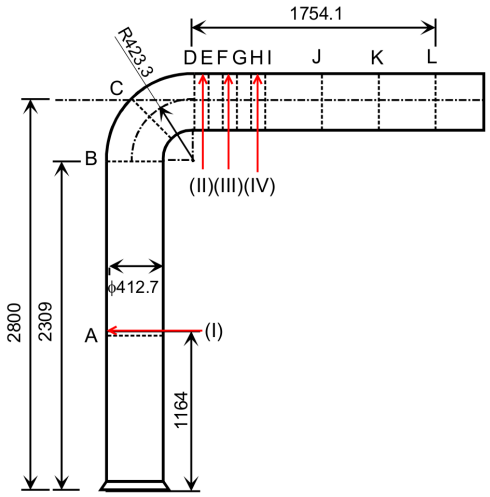
In TANAKA et al. [9] U-RANS simulations, with the k-ε turbulence model, where preformed on a set of different cases, varying mesh refinement, the simulation time-step and working fluids. For all the cases, the simulations converged towards a "repeated" solution where you have a flow field that produces the same flow structures at regular time intervals. The flow structure of importance in [9] was horseshoe shaped eddy's that where shedding from the pipe wall bottom in the area right after the pipe elbow outlet. The shedding was happening at a frequency of $20Hz$ and the movement of the eddy's was not only straight forwards towards the pipe outlet, but the eddy's where also oscillating slightly from side to side. And the effect of this was a pressure field, in the area close to the pipe bottom wall, that was fluctuating.

The results from my U-RANS simulations became very different then the results in [9]. None of my simulations converged to a "repeated" solution. There where no sign of vortex shedding in any of the test cases I preformed. There where no flow structures that where fluctuating or rotating. From a initial stat, with no fluid movement, the flow fields gradually changes to a steady state like the one you see in Figure (6.3a).

In Figure (6.3) you see pictures of a steady state solution from one test case where I was trying to reproduce Case-B4 in [9]. If you look closer at the `U` field in Figure (6.3a), you see that in the area after the pipe elbow outlet, the fluid velocity will increase as we move from the pipe bottom to the pipe top. In Figure (6.2) you see the velocity profile for the u_x component at three different position after the pipe elbow outlet. Compared to any of the velocity profiles you find in [9], my velocity profiles are different in shape and velocity magnitude. Their is also no area with back flow in positions $x/D = 0.18$ for $z/D < 0.2$.



(a) In this picture you see how I have defined the angles on the horizontal pipe sections of both the test pipe and the hot-leg piping. This is the same as in [9].



(b) The picture is from TANAKA et al. [8]

Since I have a steady state pressure field it is difficult to make a comparison to the instantaneous fields you find in [9]. One thing that indicates a difference between my pressure fields and the ones in [9], is the range of pressure magnitudes. The range in [9] extends much further both in positive and negative direction.

Different test cases produced other solutions then the ones you see in Figure (6.3), but the end result of all the cases was a steady state solution. Take a look at these movies from couple of the test cases

- https://github.com/sayedn/Master-/blob/master/mov/u-rans_CK.ogv
- https://github.com/sayedn/Master-/blob/master/mov/u-rans_movie.ogv

All the U-RANS simulations where preformed in 2D, and never extended to 3D since I couldn't get pass the steady state solutions.

6.2 The LES simulations

The reason why I did the LES simulations was because all my U-RANS simulations resulted in steady state solutions. My main LES simulations is divided into two sets, see Table (5.6). The discussion in this section will only be for the result of the main simulations, and not for any the preliminary test cases conducted.

6.2.1 The first set of LES simulations

I couldn't find horseshoe shaped eddy's in any of the cases in the first set of LES simulations. In Figure (6.4) you see instantaneous pictures of iso-surface contours defined by

$$Q = (W_{ij}W_{ij} - D_{ij}D_{ij})/2, \tag{6.1}$$

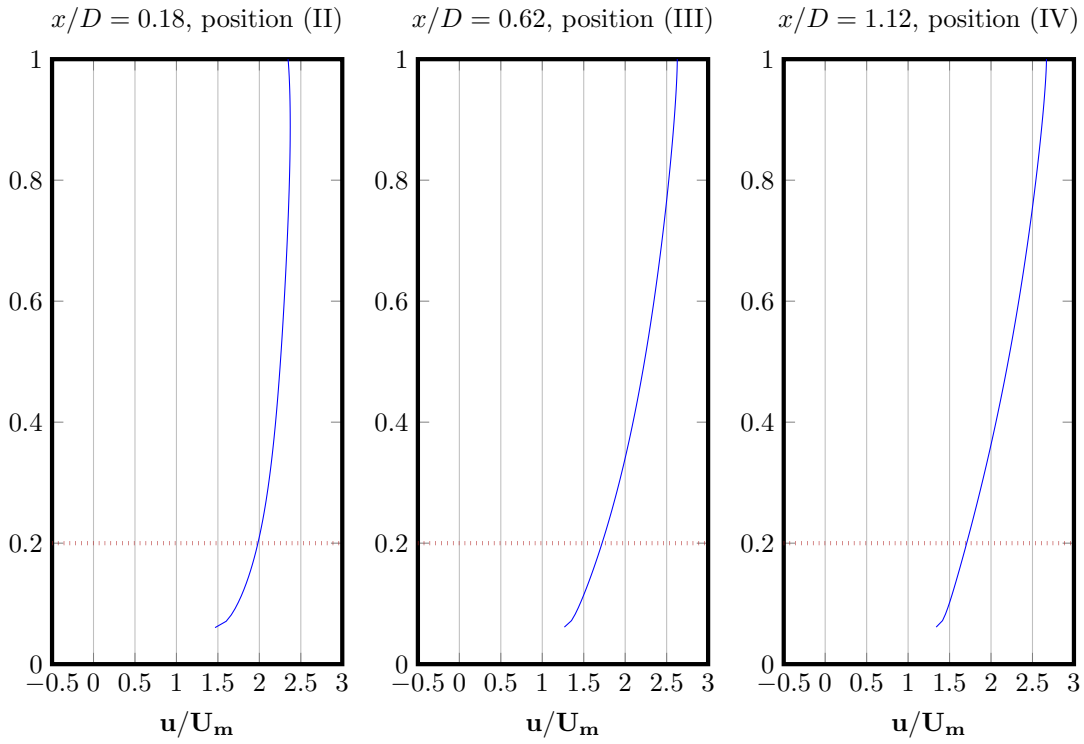


Figure 6.2: Time-averaged velocity profile of u_x component at three different positions downstream of pipe elbow outlet, see Figure (6.1b). This is for one of the U-RANS simulations, and at these positions u_x is parallel to the pipe walls.

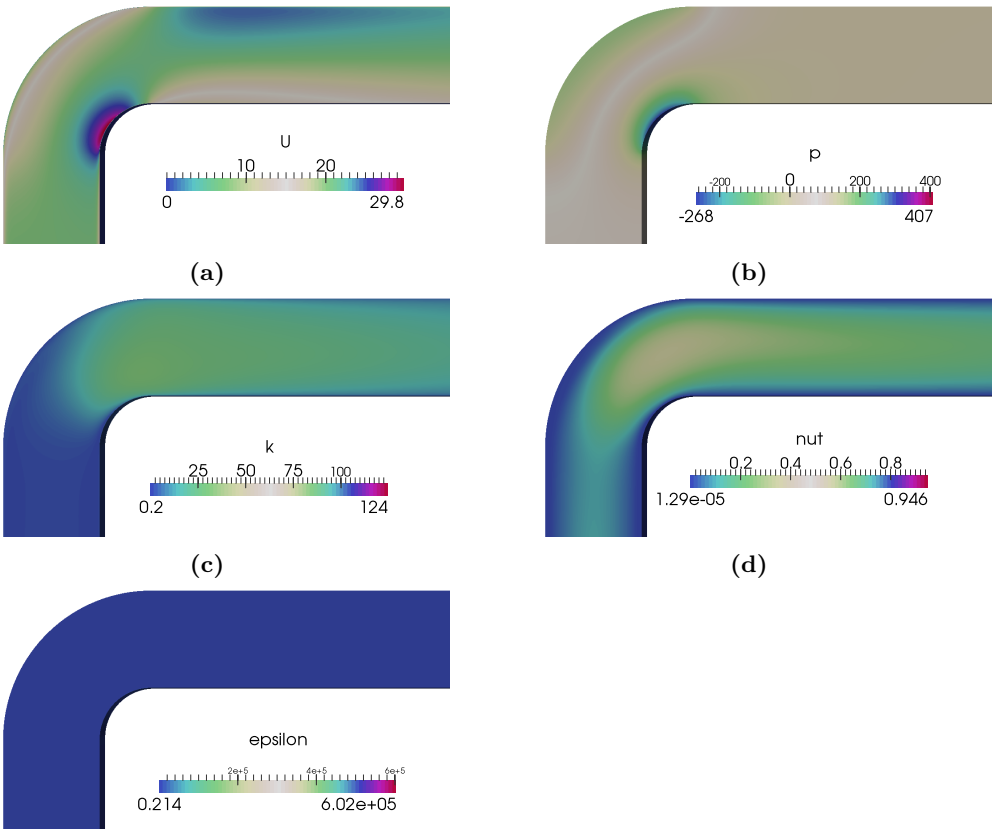


Figure 6.3: In this Figure you see the different fields from a two dimensional U-RANS simulation at t = 5 s. This is a steady state solution.

where

$$D_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad \text{and} \quad W_{ij} = \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial u_i}\right).$$

This type of iso-surfaces was used in TANAKA et al. [9] to illustrate eddy's. For all the cases I plotted Q over a wide range of different values, but didn't find any horseshoe shaped eddy's as described in [9]. Here is a movie of the iso-surface contours, with $Q = 5000s^{-2}$, for `Case-B4-u`.

- https://github.com/sayedn/Master-/blob/master/mov/iso_B4.ogv

In Figure (6.5) you see time-averaged velocity profiles of the velocity component parallel to the the pipe-center axis. There is three plots in the figure, at three different positions down stream of the pipe elbow outlet. These are the same measuring positions as in [9]. None of my velocity profiles look as their counterpart in [9]. At $x/D = 0.18$ and $x/D = 0.62$ the difference between my velocity profiles and the velocity profiles in [9] is big. The main difference is in the region $z/D < 0.6$, where the profiles in [9] has a bigger velocity magnitude. Also another thing is that I don't have any back flow in the region $z/D < 0.2$ at $x/D = 0.18$.

Here is a movie of the velocity field in `Case-B4-u`

- https://github.com/sayedn/Master-/blob/master/mov/Case-B4-u-LES_3D_2.ogv

There was a problem with the pressure data for all the cases in this set. Take a look at the movie

- https://github.com/sayedn/Master-/blob/master/mov/les_mixed.ogv

The pressure data is ruined. So I can't do a comparison against the pressure results in [9]. In Figure (6.6) you see the velocity magnitudes for the first set of LES simulations.

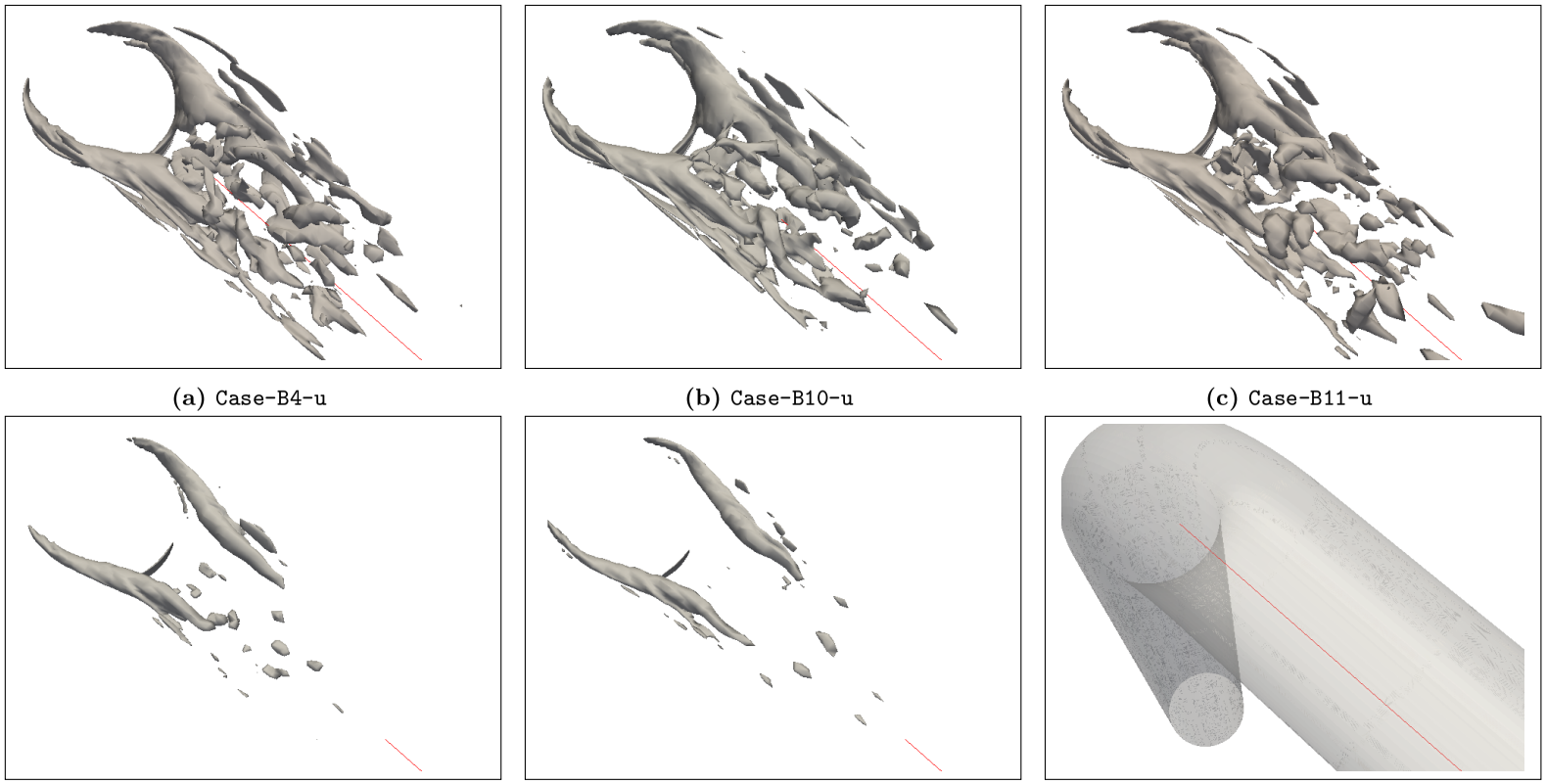
6.2.2 The second set of LES simulations

Let's again start with the iso-surface contours. In Figure (6.7) you can see a selection of surfaces for the different cases in the second set of LES simulations. Like with the first set of LES simulations, I plotted Q over a wide range of different values, but did not find any horseshoe shaped eddy's. The flow structures in Figure (6.7e) is the closest thing to a horseshoe shaped eddy I could find.

In Figure (6.8) you see velocity profiles of cases in the second set of LES simulations. Again there is a big difference between my velocity profiles and the velocity profiles in [9]. As with the first set of LES simulations the biggest difference is in the area $z/D < 0.6$ at $x/D = 0.18$ and $x/D = 0.62$. The velocity profiles at $x/D = 1.12$ is closer to the ones you find in [9], then the profiles at $x/D = 0.18$ and $x/D = 0.62$.

In Figure (6.9) and (6.10) you see frequency analysis of pressure fluctuations at two different positions $0.5D$ downstream of the pipe elbow outlet. The sampling of the pressure is done at $100Hz$ over a period of 5 seconds. At both positions, 150° and 180°, there is a peak at $St = 1$ ($20Hz$). The frequency analysis in [9] showed two peaks, one at $St = 1$ and one at $St = 0.5$ ($10Hz$). My plots don't have a peak at $St = 0.5$. Also notice that the magnitude of my PSD plots is smaller the ones in [9]. Here is a movie of the pressure fluctuations on the pipe wall for `Case-B-u_a`

- https://github.com/sayedn/Master-/blob/master/mov/Case-B4-u_a-LES_3D.ogv



(a) `Case-B4-u` **(b)** `Case-B10-u` **(c)** `Case-B11-u`

(d) `Case-B8-u` **(e)** `Case-B9-u` **(f)** View angel. The circle you see in this picture is the pipe inlet.

Figure 6.4: Instantaneous pictures at t = 5 of flow structures described by iso-surface contour of second invariant of velocity gradient. Q = 5000 s^{-2} for `Case-B4-u`, `Case-B10-u` and `Case-B11-u`. Q = 2100 s^{-2} for `Case-B8-u` and Q = 180 s^{-2} for `Case-B9-u`. I could not see any vortexes.

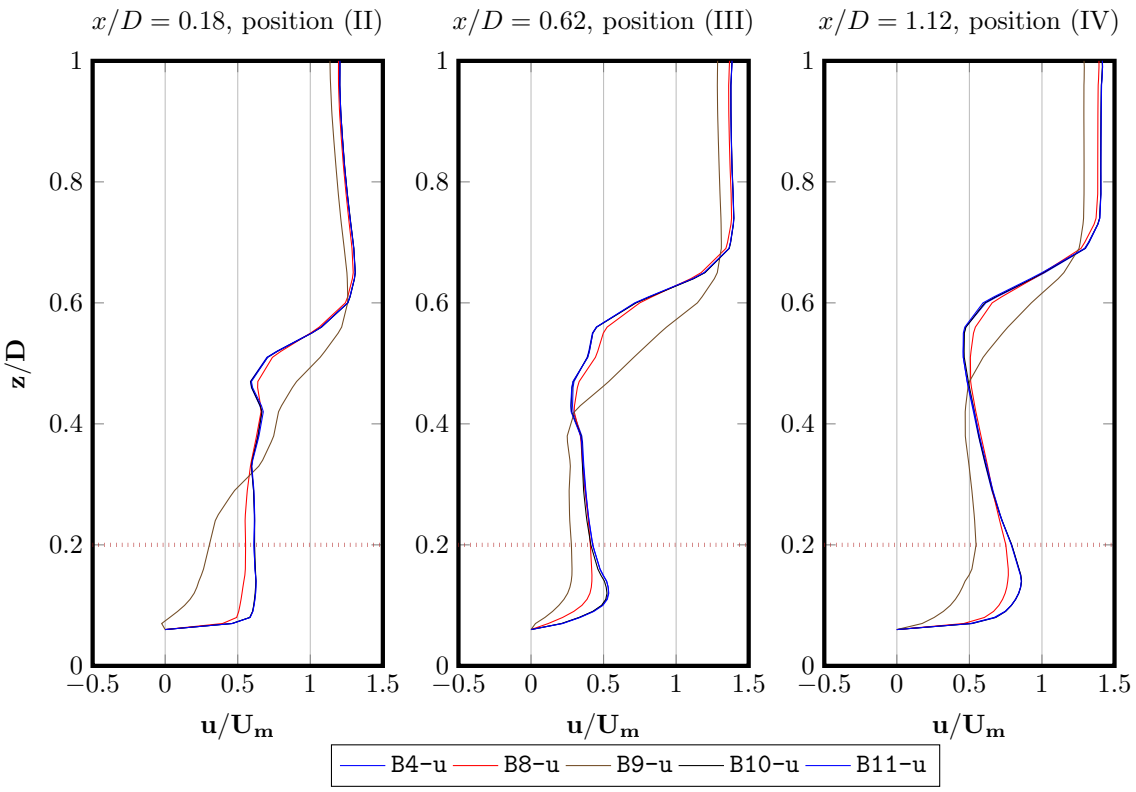
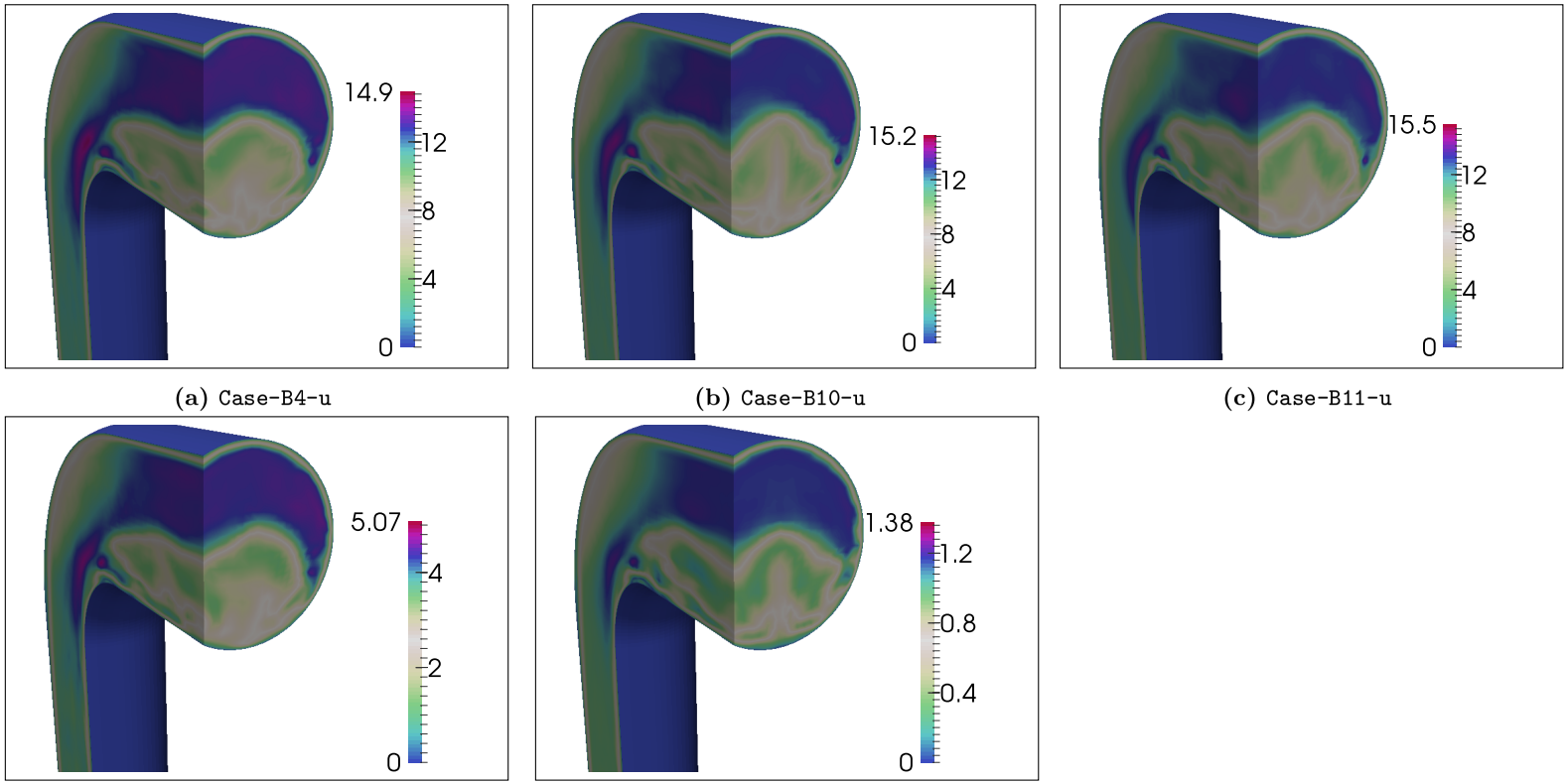


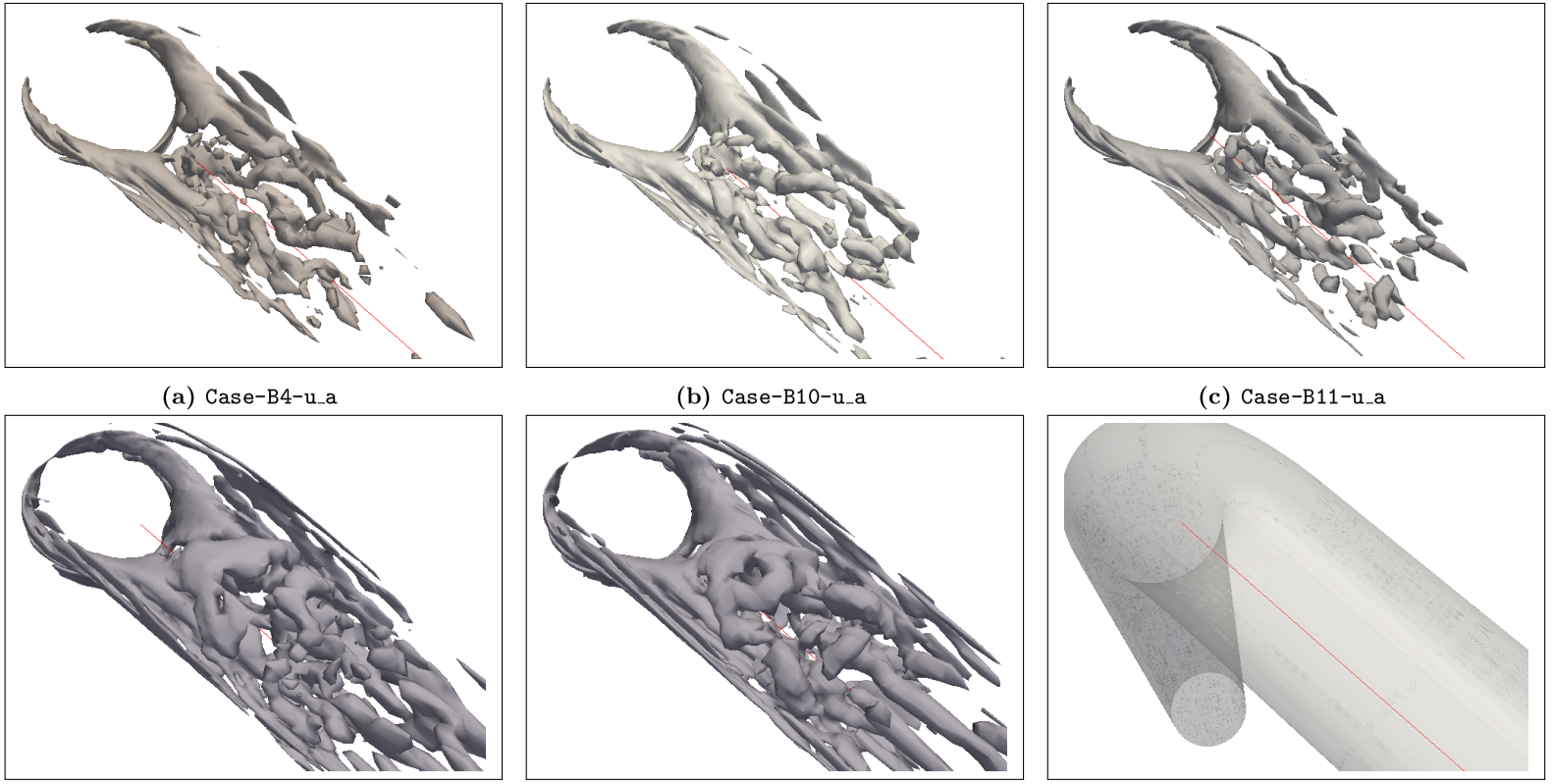
Figure 6.5: Time-averaged velocity profile of u_x component at three different positions downstream of pipe elbow outlet, see Figure (6.1b). This is for the first set of LES simulations, and at these positions u_x is parallel to the pipe walls.



(a) `Case-B4-u` **(b)** `Case-B10-u` **(c)** `Case-B11-u`

(d) `Case-B8-u` **(e)** `Case-B9-u`

Figure 6.6: Instantaneous velocity magnitudes, in $[m/s]$, at $t = 5s$. The cross-section cut, normal to the pipe center axis, is $0.5D$ downstream of the pipe elbow outlet. These velocity magnitudes is for the first set of LES simulations.



(a) `Case-B4-u_a` **(b)** `Case-B10-u_a` **(c)** `Case-B11-u_a`

(d) `Case-B8-u_a` **(e)** `Case-B9-u_a` **(f)** View angel. The circle you see in this picture is the pipe inlet.

Figure 6.7: Instantaneous pictures at t = 5 of flow structures described by iso-surface contour of second invariant of velocity gradient. Q = 5000 s^{-2} for `Case-B4-u_a`, `Case-B10-u_a` and `Case-B11-u_a`. Q = 200 s^{-2} for `Case-B8-u_a` and Q = 15 s^{-2} for `Case-B9-u_a`. I could not see any vortexes.

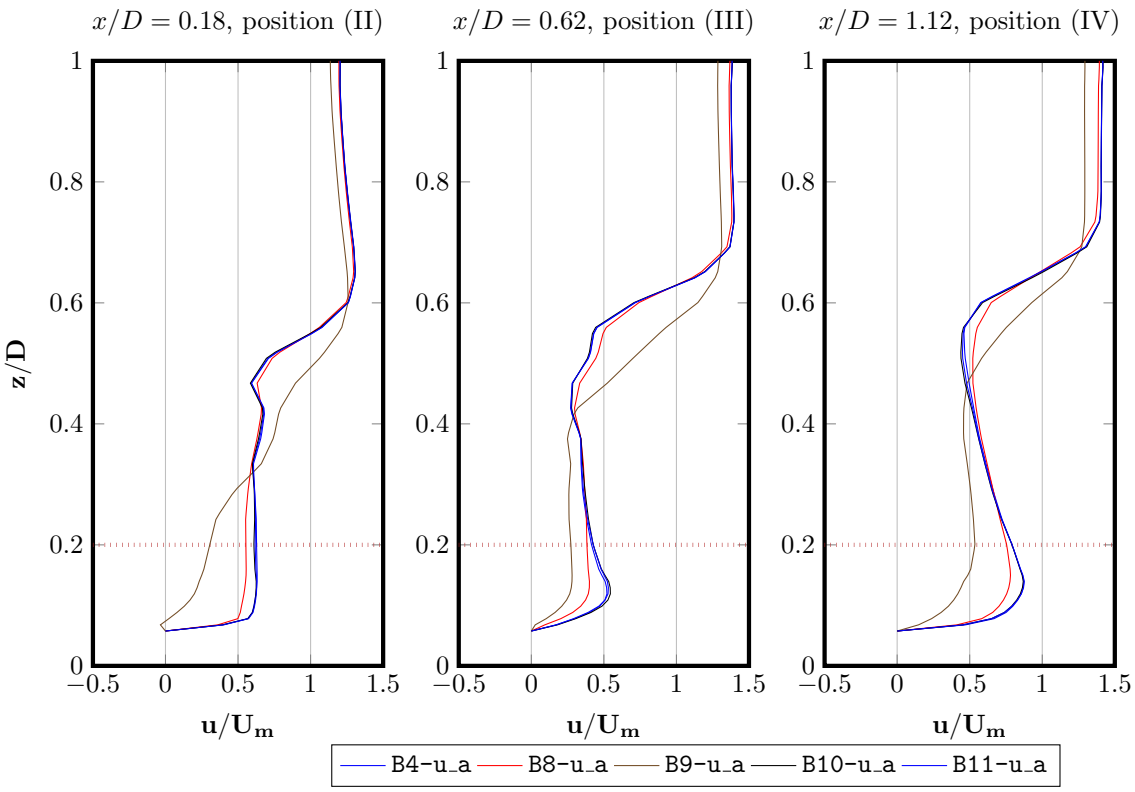


Figure 6.8: Time-averaged velocity profile of u_x component at three different positions downstream of pipe elbow outlet, see Figure (6.1b). This is for the second set of LES simulations, and at these positions u_x is parallel to the pipe walls.

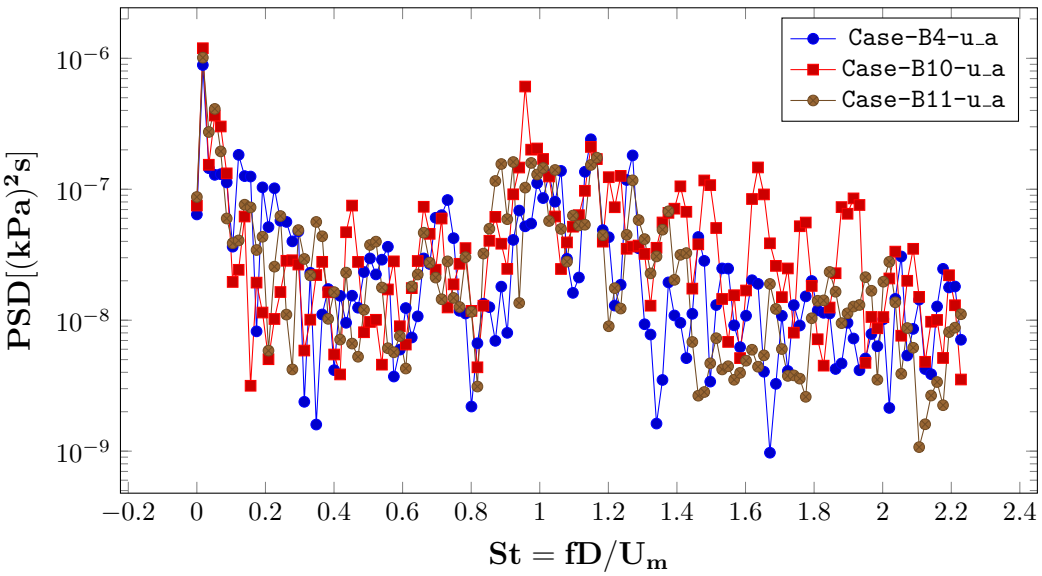


Figure 6.9: Frequency analysis of pressure fluctuations at 150° (on the pipe wall), see Figure (6.1a). The distance from the pipe elbow outlet is $0.5D$.

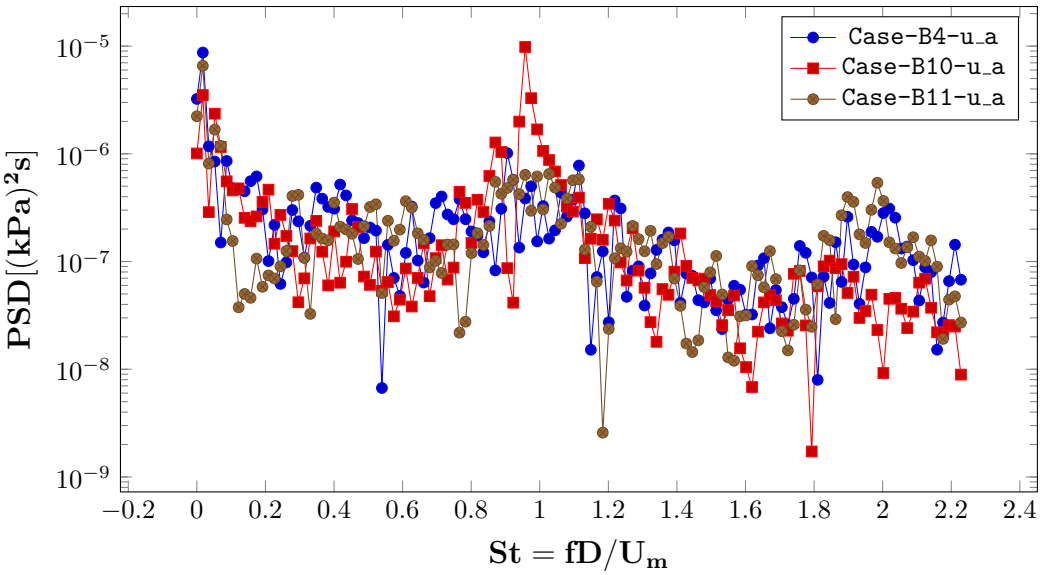
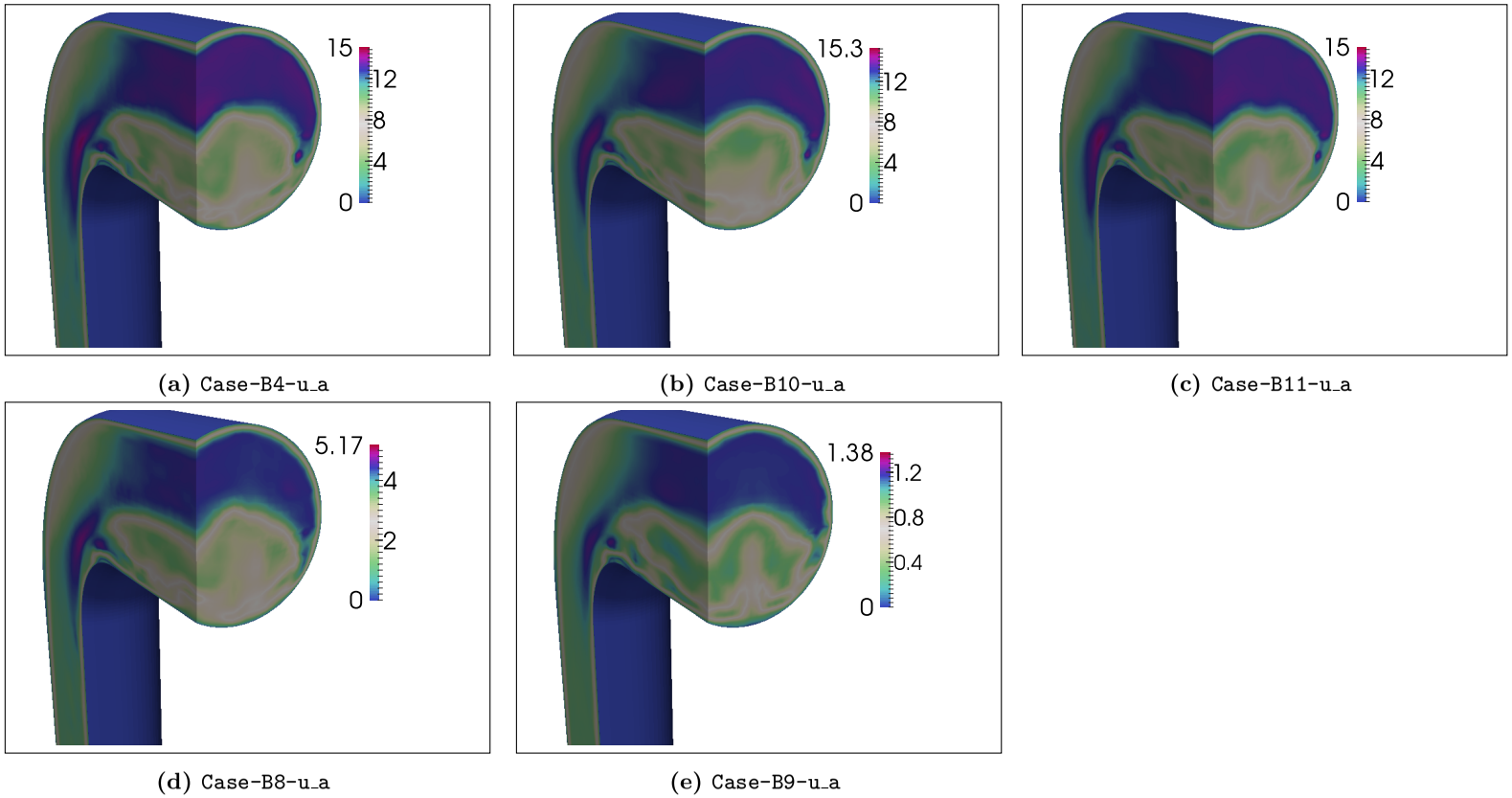


Figure 6.10: Frequency analysis of pressure fluctuations at 180° (on the pipe wall), see Figure (6.1a). The distance from the pipe elbow outlet is $0.5D$.



(a) Case-B4-u_a (b) Case-B10-u_a (c) Case-B11-u_a

(d) Case-B8-u_a (e) Case-B9-u_a

Figure 6.11: Instantaneous velocity magnitudes, in $[m/s]$, at $t = 5s$. The cross-section cut, normal to the pipe center axis, is $0.5D$ downstream of the pipe elbow outlet. These velocity magnitudes is for the second set of LES simulations.

In Figure (6.11) you see the velocity magnitudes for the second set of LES simulations.

Chapter 7

Conclusions

I didn't manage to reproduce the results in TANAKA et al. [9] and I didn't manage to make my k-ε turbulence model work properly. These are the two main down point of my master thesis project.

After having a discussion with my main supervisor Mikael Mortensen, he pointed out that maybe I need to change the discretization of some terms in the governing equations. Specially going up to higher order types of discretization both in time and space.

The choice to use LES modeling on this type of flows is problematic. Article [9] is one of many articles, in a series of publications, for finding a numerical method that can be used for studying the flow conditions inside the full scale hot-leg piping of JSFR. LES modeling was tested in earlier publications, but didn't manage to produce very good results.

A suggestion to improving the LES simulations is use a much finer mesh refinement in the pipe elbow section. And like with the U-RANS simulations trying different types discretization for the terms in the governing equations.

Other thing I could have done differently is the sampling of data. I for example extracted the pressure data from the files saved when preforming the simulation, while I could have placed out sampling probes and mange to pick up more frequencies of pressure fluctuations.

Appendix A

Source code

A.1 Gmsh code

A.1.1 .geo file for Mesh-A

```
// pipe diameter
D = 0.41;

// inlet center point
xs = -0.4; ys = 0; zs = -2.532;

// lenght of first pipe section
l1 = 4.7*D;

// length of second pipe section
l2 = 1*D;

// Radius of curvature for third pipe
    section
R = 0.4233;

// length of fourth pipe section
l4 = 1*D;

// length fifth pipe section
l5 = 7.9*D;

// radius for outer circle
r = D/2.0;

// "radius" for inner octave points
r1 = (165/205)*r;

mid1 = Sqrt((r1*r1)/2.0);

N1 = 10;

N2 = 28;

mid2 = Sqrt((r*r)/2.0);

Point(1) = {xs, ys, zs, 1.0};
Point(2) = {r1 + xs, ys, zs, 1.0};
Point(3) = {xs, ys - r1, zs, 1.0};
Point(4) = {xs - r1, ys, zs, 1.0};
Point(5) = {xs, r1 + ys , zs,1.0};

Point(6) = {mid1 + xs, mid1 + ys, zs, 1.0};
Point(7) = {mid1 + xs, ys - mid1, zs, 1.0};
Point(8) = {xs - mid1, ys - mid1, zs, 1.0};
Point(9) = {xs - mid1, ys + mid1, zs, 1.0};

Point(10) = {r + xs, ys, zs, 1.0};
Point(11) = {xs, ys - r, zs, 1.0};
Point(12) = {xs - r, ys, zs, 1.0};
Point(13) = {xs, r + ys , zs, 1.0};

Point(14) = {mid2 + xs, mid2 + ys, zs, 1.0};
Point(15) = {mid2 + xs, ys - mid2, zs, 1.0};
Point(16) = {xs - mid2, ys - mid2, zs, 1.0};
Point(17) = {xs - mid2, ys + mid2, zs, 1.0};
Point(18) = {xs + R, 0, zs + l1 + l2, 1.0};

// lines for inlet
Line(1) = {1, 2};
Line(2) = {1, 3};
Line(3) = {1, 4};
Line(4) = {1, 5};

Line(5) = {2, 7};
Line(6) = {7, 3};
Line(7) = {3, 8};
Line(8) = {8, 4};
Line(9) = {4, 9};
Line(10) = {9, 5};
Line(11) = {5, 6};
Line(12) = {6, 2};

/*
Circle(5) = {2, 1, 7};
Circle(6) = {7, 1, 3};
Circle(7) = {3, 1, 8};
Circle(8) = {8, 1, 4};
Circle(9) = {4, 1, 9};
```

```
Circle(10) = {9, 1, 5};
Circle(11) = {5, 1, 6};
Circle(12) = {6, 1, 2};
*/

Line(13) = {2, 10};
Line(14) = {7, 15};
Line(15) = {3, 11};
Line(16) = {8, 16};
Line(17) = {4, 12};
Line(18) = {9, 17};
Line(19) = {5, 13};
Line(20) = {6, 14};

Circle(21) = {10, 1, 15};
Circle(22) = {15, 1, 11};
Circle(23) = {11, 1, 16};
Circle(24) = {16, 1, 12};
Circle(25) = {12, 1, 17};
Circle(26) = {17, 1, 13};
Circle(27) = {13, 1, 14};
Circle(28) = {14, 1, 10};

// surfaces for inlet
Line Loop(29) = {1, 5, 6, -2};
Ruled Surface(30) = {29};
Line Loop(31) = {2, 7, 8, -3};
Ruled Surface(32) = {31};
Line Loop(33) = {3, 9, 10, -4};
Ruled Surface(34) = {33};
Line Loop(35) = {4, 11, 12, -1};
Ruled Surface(36) = {35};
Line Loop(37) = {13, 21, -14, -5};
Ruled Surface(38) = {37};
Line Loop(39) = {14, 22, -15, -6};
Ruled Surface(40) = {39};
Line Loop(41) = {15, 23, -16, -7};
Ruled Surface(42) = {41};
Line Loop(43) = {16, 24, -17, -8};
Ruled Surface(44) = {43};
Line Loop(45) = {17, 25, -18, -9};
Ruled Surface(46) = {45};
Line Loop(47) = {18, 26, -19, -10};
Ruled Surface(48) = {47};
Line Loop(49) = {19, 27, -20, -11};
Ruled Surface(50) = {49};
Line Loop(51) = {20, 28, -13, -12};
Ruled Surface(52) = {51};

Extrude {0, 0, l1} {
  Surface{34, 32, 30, 36, 48, 46, 44, 42,
    40, 38, 52, 50};
}

Extrude {0, 0, l2} {
  Surface{118, 96, 74, 140, 272, 250, 228,
    206, 184, 162, 316, 294};
}

Extrude {{0, 1, 0}, {xs + R, 0, zs+l1+l2},
    Pi/2} {
  Surface{404, 338, 360, 382, 558, 580, 426,
     448, 470, 492, 514, 536};
}
Extrude {14, 0, 0} {
  Surface{602, 624, 646, 668, 690, 712, 734,
     756, 778, 800, 822, 844};
}
Extrude {15, 0, 0} {
  Surface{866, 888, 910, 932, 954, 976, 998,
     1020, 1042, 1064, 1086, 1108};
}

Physical Surface("inlet") = {30, 36, 34, 32,
     40, 38, 52, 50, 48, 46, 44, 42};

Physical Surface("fixedwall") = {241, 263,
    285, 307, 153, 175, 197, 219, 439, 417,
    571, 549, 527, 505, 483, 461, 791, 769,
    747, 725, 703, 681, 835, 813, 945, 967,
    989, 1011, 1033, 1055, 1077, 1099, 1231,
     1253, 1297, 1275, 1319, 1341, 1363,
    1209};

Physical Surface("outlet") = {1130, 1152,
    1174, 1196, 1218, 1240, 1262, 1284,
    1306, 1328, 1350, 1372};

Physical Volume("internal") = {1, 2, 3, 4,
    5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
    26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
    36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
    46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
    56, 57, 58, 59, 60};

// axis lines (first part)
Transfinite Line {236, 108, 104, 280, 258,
    130, 152, 68, 59, 82, 214, 64, 148, 86,
    60, 192, 170} = 41 Using Progression 1;

// axis lines (second part)
Transfinite Line {548, 412, 394, 324, 416,
    328, 526, 376, 323, 332, 438, 372, 504,
    350, 354, 460, 482} = 21 Using
    Progression 1;

// axis lines (third part)
Transfinite Line {724, 746, 746, 746, 614,
    618, 702, 596, 768, 768, 636, 587, 592,
    592, 680, 640, 790, 588, 676, 658, 812}
    = 34 Using Progression 1;

// axis lines (fourth part)
Transfinite Line {988, 878, 1010, 966, 860,
    882, 851, 1032, 856, 944, 1032, 1032,
    900, 904, 1054, 852, 922, 940, 1076} =
    21 Using Progression 1;

// axis lines (fifth part)
Transfinite Line {1252, 1230, 1142, 1124,
    1274, 1146, 1208, 1120, 1115, 1164,
    1296, 1116, 1204, 1168, 1186, 1318,
```

```
    1340} = 61 Using Progression 1;

// circular lines
Transfinite Line {28, 21, 22, 23, 24, 25,
    26, 27, 231, 209, 187, 165, 143, 297,
    275, 253, 451, 473, 495, 517, 539, 561,
    407, 429, 671, 693, 715, 737, 759, 781,
    803, 825, 935, 957, 979, 1001, 1023,
    1045, 1067, 1089, 1199, 1221, 1243,
    1265, 1287, 1309, 1331, 1353} = N1 Using
     Progression 1;

// 'outer' radial lines
Transfinite Line {13, 14, 15, 16, 17, 18,
    19, 20, 230, 208, 186, 164, 142, -144,
    274, 252, 406, -408, -430, -452, -474,
    -496, -518, -540, -694, -716, -738,
    -760, -782, -804, 670, -672, -958, -980,
     -1002, -1024, -1046, -1068, 934, -936,
    -1200, -1222, -1244, -1266, -1288,
    -1310, -1332, 1198} = N2 Using
    Progression 0.9;

// octave lines
Transfinite Line {5, 6, 7, 8, 9, 10, 11, 12,
     99, 100, 77, 78, 55, 56, 121, 122, 319,
     320, 341, 342, 363, 364, 385, 386, 584,
     605, 606, 627, 628, 649, 650, 583, 848,
     869, 870, 891, 892, 913, 914, 847,
    1112, 1133, 1134, 1155, 1156, 1177,
    1178, 1111} = N1 Using Progression 1;

// 'inner' radial lines
Transfinite Line {1, 2, 3, 4, 98, 76, 54,
    57, 318, 321, 343, 365, 585, 607, 629,
    582, 849, 871, 893, 846, 1113, 1135,
    1157, 1110} = N1 Using Progression 1;

Transfinite Surface "*";
Recombine Surface "*";
Transfinite Volume "*";
```

A.2 Abel Computing Cluster Job Scripts

A.2.1 Simple Serial Job

```
#!/bin/bash

# Job name:
#SBATCH --job-name=test01
#
# Project:
#SBATCH --account=uio
#
# Wall clock limit:
#SBATCH --time=00:01:00
#
# Max memory usage:
#SBATCH --mem-per-cpu=1000M

## Set up job environment
source /cluster/bin/jobsetup

## Source run functions
source $HOME/.bashrc

## Copy input files to the work directory:
cp -r /usit/abel/u1/sayedn/OpenFOAM/sayedn
    -2.1.1/run/tutorials/incompressible/
    icoFoam/cavity/ $SCRATCH

## Make sure the results are copied back to
    the submit directory (see Work Directory
     below):
cleanup "cp -r $SCRATCH/cavity/0.*
    $SUBMITDIR/cavity/"

## Do some work:
cd $SCRATCH/cavity
blockMesh
icoFoam
```

A.2.2 Parallel Job

```
#!/bin/bash
# Job name:
#SBATCH --job-name=test
#
# Project:
#SBATCH --account=uio
#
# Wall clock limit:
#SBATCH --time=12:00:00
#
# Max memory usage per core (MB):
#SBATCH --mem-per-cpu=1G
#
# Number of tasks (cores):
#SBATCH --ntasks=12 # Number of cores:

## Set up job environment
source /cluster/bin/jobsetup

## module load openmpi.gnu/1.8.1
export OMPI_MCA_mpi_warn_on_fork=0

## Source run functions
source $HOME/.bashrc

## Copy input files to the work directory:
cp -r /usit/abel/u1/sayedn/OpenFOAM/sayedn
    -2.1.1/run/tutorials/incompressible/
    pisoFoam/les/Case-B4-u/ $SCRATCH

## Make sure the results are copied back to
    the submit directory (see Work Directory
     below):
```

```
cleanup "cp -r $SCRATCH/Case-B4-u/processor
    *.zip $SUBMITDIR/Case-B4-u/"

## Run command
cd $SCRATCH/Case-B4-u/

## (For non-OpenMP-programs, you must
    control the number of threads manually,
    using $OMP_NUM_THREADS.)
decomposePar
mpirun -np 12 pisoFoam -parallel

mpirun -np 1 zip -r processor0.zip
    processor0 : \
-np 1 zip -r processor1.zip processor1 : \
-np 1 zip -r processor2.zip processor2 : \
-np 1 zip -r processor3.zip processor3 : \
-np 1 zip -r processor4.zip processor4 : \
-np 1 zip -r processor5.zip processor5 : \
-np 1 zip -r processor6.zip processor6 : \
-np 1 zip -r processor7.zip processor7 : \
-np 1 zip -r processor8.zip processor8 : \
-np 1 zip -r processor9.zip processor9 : \
-np 1 zip -r processor10.zip processor10 : \
-np 1 zip -r processor11.zip processor11
```

A.3 OpenFoam code

A.3.1 Case-B4-u decomposeParDict -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}

numberOfSubdomains 12;

method          scotch;
```

A.3.2 Case-B4-u fvShemes -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}

ddtSchemes
{
    default         backward;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
    div(phi,k)      Gauss limitedLinear 1;
    div(phi,B)      Gauss limitedLinear 1;
    div(phi,nuTilda) Gauss limitedLinear 1;
    div(B)          Gauss linear;
    div((nuEff*dev(T(grad(U))))) Gauss
    linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

A.3.3 Case-B4-u fvSolution -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.1;
        smoother        GaussSeidel;
        nPreSweeps      0;
```

```
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels     1;
    }

    pFinal
    {
        $p;
        smoother        DICGaussSeidel;
        tolerance       1e-06;
        relTol          0;
    }

    "(U|k|B|nuTilda)"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

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

A.3.4 Case-B4-u LESProperties -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      LESProperties;
}

LESModel        oneEqEddy;

delta           cubeRootVol;

printCoeffs     on;

cubeRootVolCoeffs
{
    deltaCoeff      1;
}

PrandtlCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }
    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }

    Cdelta          0.158;
}

vanDriestCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }

    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }

    Aplus           26;
    Cdelta          0.158;
}

smoothCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }

    maxDeltaRatio   1.1;
}/
```

A.3.5 Case-B4-u transportProperties -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
```

```
transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 1.004e
    -06;

CrossPowerLawCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1
    e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ]
     1e-06;
    m               m [ 0 0 1 0 0 0 0 ] 1;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}

BirdCarreauCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1
    e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ]
     1e-06;
    k               k [ 0 0 1 0 0 0 0 ] 0;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}
```

A.3.6 Case-B4-u turbulenceProperties -file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}

simulationType  LESModel;
```

A.4 Python code

A.4.1 Program for finding the Power Spectral Density (PSD)

```
from scipy import signal, fftpack
import matplotlib.pyplot as plt
import csv

D = 0.41 # Pipe diameter
Um = 9.2 # Mean velocity
fs = 100 # Sampling frequency


# Open and read pressure file
data = csv.reader(open("Case-B11-u_a_p1800
    .0.csv","rb"), delimiter=",")

column1 = []
column2 = []

firstline = True

for row in data:

  if firstline:    #skip first line
          firstline = False
          continue

  column1.append(float(row[0])/1000.0)
        column2.append(float(row[2]))

# Find PSD and frequencies
f, Pxx_den = signal.welch(column1, fs)
St = [(x*D)/Um for x in f]

l = len(St)

# Write to new file
out = open('Case-B11-u_a_pds180.csv', 'w')
out.write('"St","Pxx_den" \n')

for i in range(l):
    out.write('%.16f,%.16f' % (St[i],
    Pxx_den[i]))
    out.write('\n')
out.close()
```

A.4.2 Program for calculating initial and boundary conditions for different field variables

```
from math import *

# Mean velocity
U = 9.2

# Pipe diameter
D = 0.41

# Characteristic length scale
L = D

C_nu = 0.09

l = 0.07*L

# Kinematic viscosity
nu = 1.004*10**(-6)

# Density of fluid
rho = 1000

# Reynolds number
```

```
Re_DH = (U*D)/nu

print
print "Re_DH = %f" % Re_DH
print
# Turbulence intensity
Re_DH = 50000
I = 0.16*(Re_DH)**(-1.0/8)

print "I = %f" % I
print
# Turbulence kinematic energy at different boundarys
print "
    ----------------------------------------"
print "                 k's                "
print "
    ----------------------------------------"
k_i =  (3.0/2)*(0*I)**2
print "k_i = %f" %  k_i
k_wall =  (3.0/2)*(0*I)**2
print "k_wall = %f" %  k_wall
k_out =  (3.0/2)*(0*I)**2
print "k_out= %f" % k_out
k_in = (3.0/2)*(U*I)**2
print"k_in = %f" % k_in

print "
    ----------------------------------------"
print "                 Epsilon's          "
      "
print "
    ----------------------------------------"
Epsilon_i = (C_nu**(3.0/4))*((k_i**(3.0/2))/l)
print "Epsilon_i = %f" % Epsilon_i
Epsilon_wall = (C_nu**(3.0/4))*((k_wall**(3.0/2))/l)
print "Epsilon_wall = %f" % Epsilon_wall
Epsilon_out = (C_nu**(3.0/4))*((k_out**(3.0/2))/l)
print "Epsilon_out = %f" % Epsilon_out
Epsilon_in = (C_nu**(3.0/4))*((k_in**(3.0/2))/l)
print "Epsilon_in = %f" % Epsilon_in

print "
    ----------------------------------------"
print "                 nuTilda's          "
      "
print "
    ----------------------------------------"
nuTilda_in = sqrt(3.0/2)*U*I*l
print "nuTilda_in = %f" % nuTilda_in
#Epsilon_wall = (C_nu**(3.0/4))*((k_wall**(3.0/2))/l)
#print "Epsilon_wall = %f" % Epsilon_wall
#Epsilon_out = (C_nu**(3.0/4))*((k_out**(3.0/2))/l)
#print "Epsilon_out = %f" % Epsilon_out
#Epsilon_in = (C_nu**(3.0/4))*((k_in**(3.0/2))/l)
#print "Epsilon_in = %f" % Epsilon_in


print "
    ----------------------------------------"
print "                 nut's
      "
print "
    ----------------------------------------"
nut_in = (C_nu*(k_in**2))/(Epsilon_in)
print "nut_in = %f" % nut_in
#nut_i = (C_nu*(k_i**2))/(Epsilon_i)
#print "nut_i = %f" % nut_i
#nut_wall = (C_nu*(k_wall**2))/(Epsilon_wall)
#print "nut_wall = %f" % nut_wall
#nut_out =  (C_nu*(k_out**2))/(Epsilon_out)
#print "nut_out = %f" % nut_out
```

Bibliography

[1] Bash scripting tutorial, . URL http://linuxconfig.org/bash-scripting-tutorial. Accessed: 2014-09-23.

[2] Wikipedia, . URL http://www.wikipedia.org/. Accessed: 2014-09-23.

[3] Kosuke AIZAWA, Shigeyuki NAKANISHI, Hidemasa YAMANO, Shoji KOTAKE, Satoshi HAYAKAWA, Osamu WATANABE, and Kazuhiro FUJIMATA. Study on flow-induced-vibration evaluation of large-diameter pipings in a sodium-cooled fast reactor (1) Sensitivity analysis of turbulent flow models for unsteady short-elbow pipe flow. In *NTHAS6: Sixth Japan-Korea Symposium on Nuclear Thermal Hydraulics and Safety*, November 2008.

[4] P. A. Durbin and B. A. Pettersson-Reif. *Statistical Theory and Modeling for Turbulent Flows*. John Wiley & Sons Ltd, second edition, 2011.

[5] Kazuo Hirota, Yoshihide Ishitani, Tomomichi Nakamura, Tadashi Shiraishi, and Hiromi Sago. Flow-induced vibration of a large-diameter elbow piping in high reynolds number range; random force measurement and vibration analysis. 2008.

[6] A. Ono, N. Kimura, H. Kamide, and A. Tobita. Influence of elbow curvature on flow structure at elbow outlet under high reynolds number condition. *Nuclear Engineering and Design*, June 2010.

[7] Tadashi SHIRAISHI, Hisato WATAKABE, Hiromi SAGO, and Hidemasa YAMANO. Pressure fluctuation characteristics of the short-radius elbow pipe for fbr in the post-critical reynolds regime. *Journal of Fluid Science and Technology*, 4(2), 2009.

[8] Masaaki TANAKA, Hiroyuki OHSHIMA, Hidemasa YAMANO, Kosuke AIZAWA, and Tatsuya FUJISAKI. Application of U-RANS to Elbow Pipe Flow with Small Curvature Radius under High Reynolds number Conditions. In *International Congress on Advances in Nuclear Power Plants 2009 (ICAPP 2009)*. Proceedings of a meeting held 10-14 May 2009, Shinjuku, Tokyo, Japan.

[9] Masaaki TANAKA, Hiroyuki OHSHIMA, Hidemasa YAMANO, Kosuke AIZAWA, and Tatsuya FUJISAKI. U-rans simulation of unsteady eddy motion in pipe elbow at high reynolds number conditions. *Proceedings of ICAPP '10*, 2010.

[10] Department of Informatics University of Oslo. Abel User Guide, September 2012. URL http://www.uio.no/english/services/it/research/hpc/abel/help/user-guide/. Accessed: 2014-09-23.

[11] H. K. Versteeg and W. Malalasekera. *An Introduction to COMPUTATIONAL FLUID DYNAMICS The Finite Volume Method*. Prentice-Hall Inc., 2nd edition, 2007.

[12] Hidemasa Yamano, Masaaki Tanaka, Nobuyuki Kimura, and Hiroyuki Ohshima. Development of flow-induced vibration evaluation methodology for large-diameter piping with elbow in japan sodium-cooled fast reactor. *Nuclear Engineering and Design*, 2011.

[13] Hidemasa YAMANO, Masaaki TANAKA, Takahiro MURAKAMI, Yukiharu IWAMOTO, Kazuhisa YUKI, Hiromi SAGO, and Satoshi HAYAKAWA. Unsteady Elbow Pipe Flow to Develop a Flow-Induced Vibration Evaluation Methodology for Japan Sodium-Cooled Fast Reactor. *Journal of NUCLEAR SCIENCE and TECHNOLOGY*, 48(4):677–687, 2011.

Index