Politechnika Warszawska
Wydział Mechaniczny Energetyki i Lotnictwa

Praca przejściowa magisterska

Utilisation of space-varying boundary conditions in OpenFOAM numerical package

Wojciech Sadowski

Opiekun pracy:
dr hab. inż. Sławomir Kubacki

February 7, 2019

Contents

List of Figures

Chapter 1

Introduction

1.1 Goal of the project

This work aims to achieve several goals. First one is to learn **OpenFOAM** package. Get to know it's abilities and use it comfortably despite of it's rather intimidating text and console based user interface. Finally use it extensively for other projects since it is freely available as open source code.

Second goal is to understand how to impose space-varying boundary conditions in **OpenFOAM** . In many CFD problems there is a need to impose an non uniform boundary condition, based for example on a measured data of certain quantities, in order to achieve the same conditions as in the experiment. One example of such situation could be validation of a simulation code or a model against collected data.

Third goal would be to assert if conducting simulations with such boundary conditions is appropriate enough for **OpenFOAM** to be used as validation tool for various CFD models. In order to do that, a simple two dimensional test case will be considered and flow simulation will be performed. Next domain will be trimmed. Appropriate space-varying boundary conditions will be imposed on this new domain (generated from solution of the first simulation). Solution from both cases will be compared and studied.

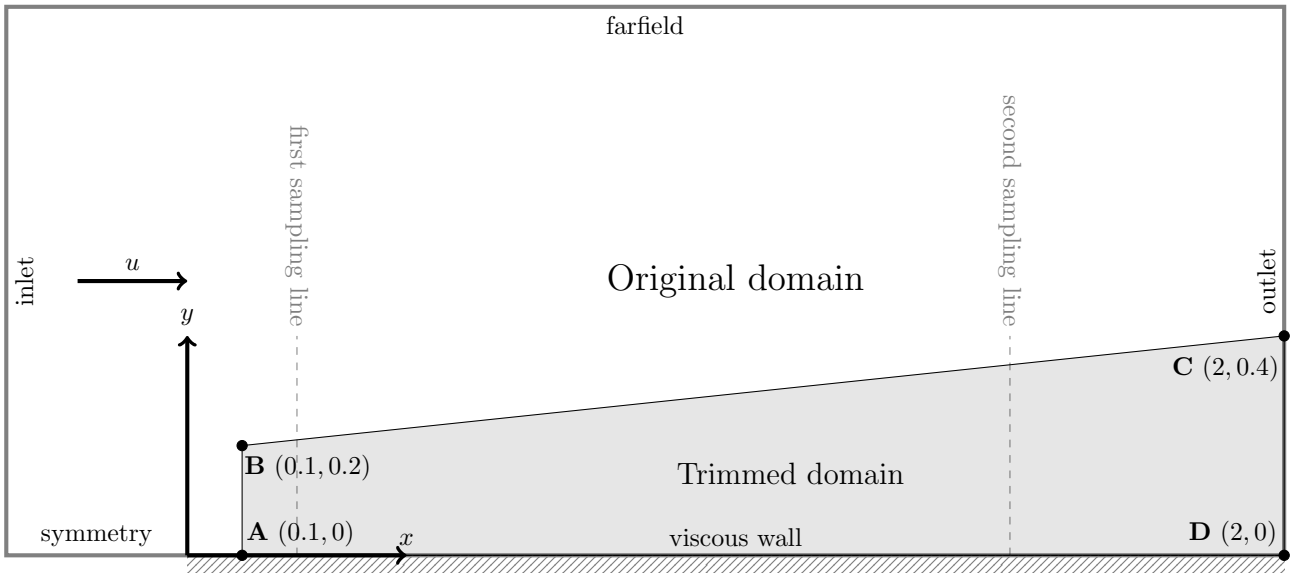

Figure 1.1: Depiction of physical domains for first and second simulation with a descriptions of the boundary conditions of the original domain (including the placements of two sampling lines used for comparison of conducted simulations)

1.2 Description of the test case

In order to grasp wide spectrum of **OpenFOAM** possibilities, test case with fully turbulent flow was chosen: flow over a plat plate. It is a simple and very popular case, used often for turbulent model validation (the best resources describing this specific flow and an archive of validation data for turbulence models is available here [2]).

Simulated flow is two dimensional and incompressible (see fig. 1.1). Fluid has kinematic viscosity of the air:

$$\nu = 1.5 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$$

Reynolds number Re of the flow (see chapter 2.1) is equal 5×10^6. Characteristic length that is used to calculate Re is exactly 1 m. Knowing those parameters velocity of the fluid at the inlet can be calculated:

$$u = 75\,\mathrm{m/s}$$

Turbulent intensity t_i, at the inlet (defined as a fraction of turbulent velocity fluctuations to mean velocity value ${}^{u'}/_{u}$) is equal to 0.05. Characteristic length scale l_T of those fluctuations is 1 m.

Second simulation will be conducted on the trimmed domain. Fields computed from the first one will be sampled along lines describing new area of calculations. Extracted data will be used to impose correct boundary conditions in the latter case.

Chapter 2

Theoretical background

2.1 Turbulence and RANS equations

Each flow can be characterised by a non dimensional number called the *Reynolds number*:

$$Re = \frac{uL}{\nu} \tag{2.1}$$

In the above expression symbols u, L and ν denotes:

u - velocity scale of the flow (e.g inlet velocity);

L - characteristic length of the domain (e.g height of the step in backward facing step case);

ν - kinematic viscosity of the fluid.

Experiments conducted by Osborne Reynolds [10] prove that for low Reynolds numbers flow will remain stable even under the influence of big disturbances (e.g. separation, wall surface roughness). However, above certain Reynolds number each flow will lose it's laminar nature and become unstable even under the influence of very small disturbances.

Resulting oscillating flow is called turbulent. It can be described by three distinct features:

- presence of eddies and swirling motion;
- chaotic movement of the flow (small changes in initial conditions will cause drastic changes in flow appearance);
- greatly increased diffusion of any flow quantity by turbulent oscillations.

Turbulence can be simulated directly. Navier-Stokes equations fully describes flow of any fluid, turbulent motion included. Such numerical experiments are named *DNS* (Direct Numerical Simulation). Unfortunately such approach has many disadvantages:

- in order to simulate any turbulent flow very fine mesh is necessary [1]
- simulation of that kind can not be simplified to a two dimensional case - turbulence is inherently a three dimensional phenomenon [9];
- DNS simulation generates a lot of data and full simulation may take days or months.

As of today DNS is not a viable option option for industrial CFD applications [12]. Luckily there are other means to describe effects that turbulence has on fluid flow. In each turbulent flow two components of movement can be easily distinguished - one describing mean motion of he flow and the other corresponding to turbulent oscillations and eddies. Given that, any field ϕ describing such flow can be decomposed into two separate fields:

$$\phi = \overline{\phi} + \phi' \tag{2.2}$$

In above expression $\overline{\phi}$ denotes time averaged value [1] of ϕ and ϕ' describes turbulent fluctuations (see fig 2.1).

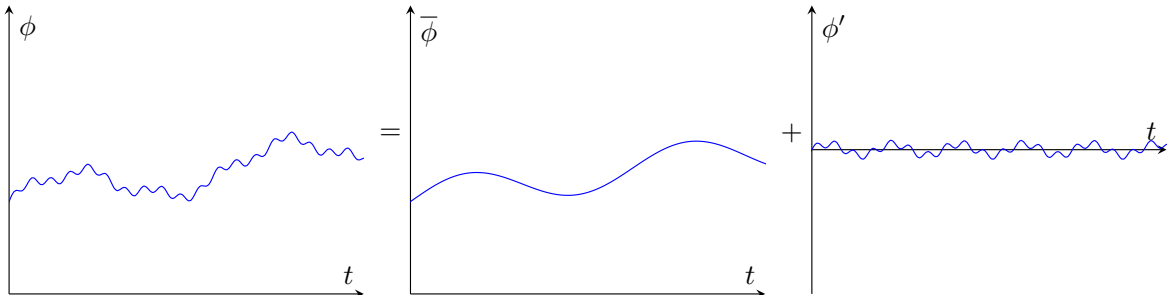

Figure 2.1: Decomposition of field variable ϕ into time average and fluctuation part

In general data describing fluctuations ϕ' is not useful. Mean properties of the flow, on the other hand, provide much more advantageous information. In two similar flows, instantaneous image of two turbulent flows will be drastically different as fluctuations are chaotic in nature. Mean values of e.g. velocity fields will be comparable.

In order to achieve quantitative description of fluid flow in terms of averaged variables $\overline{\boldsymbol{u}}$ and $\overline{p}$ two operations on Navier-Stokes equations must be made:

1. Each variable must be decomposed into it's averaged and oscillating parts.
2. Whole system of Navier-Stokes equations must be averaged in time.

Having performed them, several of arising terms can be dropped and simplified resulting in *Reynolds Averaged Navier-Stokes* (RANS) equations:

$$\begin{cases} \dfrac{\partial \overline{\boldsymbol{u}}}{\partial t} + (\overline{\boldsymbol{u}} \cdot \nabla)\overline{\boldsymbol{u}} = -\nabla\left(\dfrac{\overline{p}}{\rho}\right) + \nabla \cdot \left(\nu \nabla \overline{\boldsymbol{u}} + \nu \nabla^T \overline{\boldsymbol{u}} + \boldsymbol{\tau}\right) \\ \nabla \cdot \overline{\boldsymbol{u}} = 0 \end{cases} \tag{2.3}$$

The tensor $\boldsymbol{\tau}$ is called Reynolds stress tensor. It is symmetric and its components are defined in the following way:

$$\tau_{ij} = -\overline{u_i' u_j'} \tag{2.4}$$

It represents effect turbulence has on the flow. Observing the above equation two remarks can be made. Firstly $\boldsymbol{\tau}$ is present in the exact part of equation that is "responsible" for viscous dissipation i.e. diffusion of momentum. It corresponds with one of the aforementioned features of turbulence: eddies increase mixing of the velocity field. Secondly, if Reynolds tensor is dropped from the equation original Navier-Stokes formulation is recovered.

Unfortunately components of $\boldsymbol{\tau}$ are dependent on fluctuating parts of velocity field. Since no assumptions can be made about turbulent fluctuations, values of all six components of this tensor must be modelled. Te process of modelling components of $\boldsymbol{\tau}$ is often denoted as "closure".

First step in closing RANS equation is assuming that Boussinesq eddy viscosity theory holds. It states that momentum transfer caused by turbulent eddies can be modelled by adding artificial eddy viscosity. Tensor $\boldsymbol{\tau}$ is then defined as:

$$\tau_{ij} = 2\nu_T S_{ij} - \frac{2}{3}\overline{u_\alpha' u_\alpha'}\delta_{ij} = 2\nu_T\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) - \frac{2}{3}\overline{u_\alpha' u_\alpha'}\delta_{ij} \tag{2.5}$$

The last term on the right can be ignored for non-supersonic speed flows [1]. The newly introduced viscosity must be calculated by a chosen turbulent model. RANS equation after substituting Boussinesq assumption looks as follows:

$$\begin{cases} \dfrac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \dfrac{\partial \overline{u}_i}{\partial x_j} = -\dfrac{\partial}{\partial x_i}\left(\dfrac{\overline{p}}{\rho}\right) + \dfrac{\partial}{\partial x_j}\left[(\nu + \nu_T)\left(\dfrac{\partial \overline{u}_i}{\partial x_j} + \dfrac{\partial \overline{u_j}}{\partial x_i}\right)\right] \\ \dfrac{\partial \overline{u}_k}{\partial x_j} = 0 \end{cases} \tag{2.6}$$

2.2 Wilcox k-ω turbulence model

In order to calculate turbulent viscosity ν_T system of two partial differential equations is introduced

$$\begin{cases} \dfrac{\partial k}{\partial t} + \overline{u}_j \dfrac{\partial k}{\partial x_j} = P_k - \beta^* \omega k + \dfrac{\partial}{\partial x_j}\left[(\nu + \sigma_k \dfrac{k}{\omega})\dfrac{\partial k}{\partial x_j}\right] \\ \dfrac{\partial \omega}{\partial t} + \overline{u}_j \dfrac{\partial \omega}{\partial x_j} = \dfrac{\gamma\omega}{k} P_k - \beta\omega^2 + \dfrac{\partial}{\partial x_j}\left[(\nu + \sigma_\omega \dfrac{k}{\omega})\dfrac{\partial \omega}{\partial x_j}\right] \end{cases} \tag{2.7}$$

First one is the equation for transport of kinetic energy of turbulent fluctuations k, that is defined in a following way:

$$k = \frac{1}{2}\overline{u_i' u_i'} \tag{2.8}$$

Another scalar variable that is solved for is ω - the rate at which turbulence kinetic energy is dissipated per unit volume and time. Sometimes it is also referred to as the mean frequency of the turbulence. Using those two quantities ν_T can be computed:

$$\nu_T = \frac{k}{\omega} \tag{2.9}$$

First term on the right in the first equation is denoted as production of kinetic energy and calculated with following formula:

$$P_k = \nu_T S^2, \quad S = \sqrt{2 S_{ij} S_{ij}} \tag{2.10}$$

Constants present in the model have following values:

$$\sigma_k = \sigma_\omega = 0.5, \quad \gamma = \frac{5}{9}$$

$$\beta = \frac{3}{40}, \quad \beta^* = 0.09$$

In order to solve this system of equations appropriate boundary conditions must be specified. Boundary condition for k can be easily deduced from it's definition (2.8):

$$k_{wall} = 0 \tag{2.11}$$

The same boundary condition for ω can be obtained by asymptotic analysis of the model behaviour near the solid wall [1]:

$$\omega_{wall} = \frac{6\nu}{\beta d_1^2} \tag{2.12}$$

Variable d_1 is the distance from solid wall to the nearest integration point. Additionally inlet boundary conditions must be specified; for k:

$$k_{inlet} = \frac{3}{2}(|\boldsymbol{u}|t_i)^2$$

and for ω:

$$\omega inlet = \frac{\sqrt{k_{inlet}}}{l_T}$$

Chapter 3

OpenFOAM case definition

Case in **OpenFOAM** is defined in a very segregated way. Each piece of the case is defined in a separate text file placed in the specific directory in case directory tree (see figure 3.1). The most important thing to remember while editing those files is that they are essentially C++ code. Which means that standard C++ programming rules apply. For example each line must be ended with a semicolon. This section will describe in detail how to set up a full **OpenFOAM** case starting from mesh generation and finishing on post-processing. Input files for both cases featured in this work can be viewed and downloaded from [11].

```
case ..................................................... main directory of the case
├── constant ..................... constant properties of the case e.g. mesh description
├── system ....................... numerical schemes, solvers and simulation settings
├── 0 ............................ initial conditions for all fields including boundary conditions
├── 100 ......................................................... solution for timestep t = 100
└── processor* ................... part of the case that was decomposed for core "*"
```

Figure 3.1: Typical **OpenFOAM** case folder structure

3.1 Turbulent flow over flat plate

First step in defining the case is selecting properties of transported fluid. This can be done by creating appropriate file in `constant` folder:

```
mkdir constant
touch constant/transportProperties
```

and editing it so it looks like this:

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

transportModel  Newtonian;

nu              [ 0 2 -1 0 0 0 0 ] 1.5e-05;


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

Code starting with `FoamFile` is a definition of chosen file. It has file's location specified and it's name. Most of the files created will be instances of `dictionary` class and will have the same version and format.

Below transport model is declared. Simulated fluid has Newtonian constitutive relation. Next fluid kinematic viscosity is defined. Value 1.5×10^{-5} selected for this simulation is appended by description of its physical units (which is optional). Seven fields in braces corresponds to powers of specific SI units:

```
[kg m s K mol A cd]
```

Kinematic viscosity has a unit of m^2/s which yields `[ 0 2 -1 0 0 0 0 ]` as **OpenFOAM** unit description.

Since simulated flow is turbulent in nature, solver also needs information about turbulent modelling technique and chosen turbulence model. Those settings are in file named `turbulenceProperties` placed in the same folder. Contents of this file should look like this:

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

RAS
{
    RASModel        kOmega;

    turbulence      on;

    printCoeffs     on;
}
```

First `simulationType` field must be defined. Here `RAS` is **OpenFOAM** keyword for RANS simulation. Next `RAS` dictionary must be defined which includes choosing turbulence model, switching on the turbulence and additional options.

3.1.1 Mesh generation

Next step is mesh generation using `blockMesh` utility. It is capable generating structural hexagonal meshes in three dimensions. Input for `blockMesh` is also defined in text file named `blockMeshDict` and placed in `case/system` directory. Detailed explanation and description of `blockMesh` format is available here [3].

For this case `blockMeshDict` should look like this:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale 0.333333333333333;    //scale of the whole mesh

vertices                    //definition of vertices needed for defining
(                           //  two hexes (numbering starts from 0)
    (-1 0 0.3)
    (0 0 0.3)
    (6 0 0.3)
    (6 3 0.3)
    (0 3 0.3)
    (-1 3 0.3)

    (-1 0 0)
```

```
    (0 0 0)
    (6 0 0)
    (6 3 0)
    (0 3 0)
    (-1 3 0)
);

blocks                          //defing two blocks
(
    hex (6 7 10 11 0 1 4 5)         //definition of a hex by 8 vertices
        ( 96 385 1)                 //divisions of mesh in x, y and z directions
        simpleGrading (((1 9 0.15)(0.1 3 0.9)) 50000 1) //grading of the mesh
    hex (7 8  9 10 1 2 3 4)
        (449 385 1)
        simpleGrading (((0.1 22 1.5)(0.35 23 7)(0.515 12 1)) 50000 1)
);

edges                           //definition of the edges (not used here)
(
);


boundary                        //definition of boundary patches
(
    inlet                       //patch name
    {
        type patch;
        faces                   //faces defining patch (by vertex number)
        (
            (6 0 5 11)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (8 9 3 2)
        );
    }
    topWall
    {
        type wall;              //if patch is of type wall than postprocessing
        faces                   //   tools such as yPlus will be used on this
        (                       //   patches
            (11 5 4 10)
            (10 4 3 9)
        );
    }
    bottomWall
    {
        type wall;
        faces
        (
            (7 8 2 1)
        );
    }
    symmetry
    {
        type symmetryPlane;
        faces
        (
```

```
            (6 7 1 0)
        );
    }
    frontAndBack //empty boundary condition - reducing 3D to 2D
    {
        type empty;
        faces
        (
            (0 1 4 5)
            (1 2 3 4)
            (6 11 10 7)
            (7 10 9 8)
        );
    }
);

mergePatchPairs
(
);

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

The last boundary patch is of type `empty`. Since **OpenFOAM** can only simulate three dimensional flows. Empty boundary condition is used to reduce case to two dimensional. It should be specified in two dimensional cases on faces perpendicular to z axis.

3.1.2 Defining boundary and initial conditions

Initial and boundary conditions must be specified for each field that is calculated during run time: $\boldsymbol{u}$, p, k, ω and ν_T. Every simulation in **OpenFOAM** is iterated in time (even steady ones; time is than just iteration number). Definitions for each of those fields is specified in the `0` folder (see figure 3.2) which can be interpreted as flow data from time 0. Each of the files has the same structure (and files in directories describing further times e.g. `100/U` will have the same structure as well) that can be divided into 4 sections:

`FoamFile` - standard file description. Class of each field is defined here: `volVectorField` is a vector field e.g. $\boldsymbol{u}$ and `volScalarField` is a typical scalar quantity, for example p.

`dimension` - definition of physical units of the quantity.

`internalField` - values of field variables inside the domain. In the initial time step it is connivent to use `uniform` keyword to set one value for each cell, for example `internalField uniform (75 0 0);` initialises velocity vector with x component equal to 75.

`boundaryField` - definitions of boundary conditions. For each `patch` defined in `blockMeshDict` user must specify boundary condition type and value (if needed). Several of possible types are:

- `fixedValue` - condition of Dirichlet type imposing uniform value;
- `zeroGradient` - zero Neumann boundary condition;
- `symmetryPlane` - condition imposing symmetry;
- `empty` - condition placed in two dimensional cases on unnecessary patches.

```
case
└─ 0 ..................................... initial time folder
   ├─ U ..................................... velocity field
   ├─ p ..................................... pressure field
   ├─ k ..................................... turbulent kinetic energy field
   ├─ omega ................................. turbuelnt dissipation rate
   └─ nut ................................... turbulent visocsity
```

Figure 3.2: File structures defining initial and boundary conditions

Exemplary text file for velocity field is shown below:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (75 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }

    outlet
    {
        type            zeroGradient;
    }

    topWall
    {
        type            fixedValue;
        value           $internalField;
    }

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

    symmetry
    {
        type            symmetryPlane;
    }

    frontAndBack
    {
        type            empty;
    }
}

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

3.1.3 Setting up the simulation

In order to run any simulation a file with solver setting must be specified. It stores information about time step, iteration numbers and other options. File must be named `controlDict` and must be placed in `case/system` directory.

Again it has `FoamFile` field in the begging. Than it is appended by following entries:

application - in this place numerical solver is specified, in this simulations `simpleFoam` was chosen. It is a solver for steady turbulent problems.

iterations - specification of iterations or time stepping (as here they are interchangeable) is rather self explanatory and is achieved in the following way:

```
startFrom       startTime;
startTime       41000;
stopAt          endTime;
endTime         50000;
deltaT          1;
```

saving solutions - there are several options for saving solutions while the simulation is running. Again they are mostly self explanatory:

```
writeControl    timeStep;
writeInterval   2000;
purgeWrite      0;
```

The last entry in the above list describes when solutions from previous time steps should be deleted (integer specified in `purgeRight` field describes maximal number of time directories; `0` means no overwriting, each solution is saved).

I/O options - fields describing writing precisions for fields and time:

```
writeFormat     ascii;
writePrecision  8;
writeCompression off;
timeFormat      general;
timePrecision   8;
runTimeModifiable true;
```

Last entry in the above list enables altering case files during simulation time i.e. changing of postprocessing settings, time step etc.

In depth description of options available in `controlDict` can be found in [8].

3.1.4 Selecting numerical schemes

Each differential operator is in any equation must be simplified into its discrete form in order to solve such problem. Description of numerical schemes used to represent those operators is in the file `case/system/fvSchemes`.

Another reason behind **OpenFOAM** being noticeably harder to use than other software packages that are available because is, it does not specify any standard numerical schemes for a chosen problem. User must specify a discrete scheme for each differential operator manually. For example time derivative can be defined in the following way:

```
ddtSchemes
{
    default steadyState; //default scheme (used for every field variable)
    ddt(k)  backward;    //scheme for k equation (overrides default scheme)
}
```

If simulation requires calculation of wall distance y that method should also be specified here:

```
wallDist
{
    method          meshWave;
}
```

Additional documentation about numerical schemes can be found here [4].

3.1.5 Setting up the linear solvers

After the discretisation each equation is represented by system of linear equations, that is solved iteratively. User can specify a solver type and its parameters inside `case/system/fvSolution`. Since `simpleFoam` is a segregated solver (it solves for each component of $\boldsymbol{u}$ and p separately) user must specify one solver for components velocity and the other one for pressure. Another two solvers for turbulent variables are also necessary. For example settings for p may look like this:

```
p
{
    solver          GAMG;           //type of the solver
    tolerance       1e-16;          //final tolerance of the residual
    relTol          0.001;          //relative tolerance of the residual
    maxIter         1000;           //maximum number of iterations
    smoother        GaussSeidel;    //type of smoother
    nPreSweeps      1;              //-----------------------------
    nPostSweeps     3;              //
    nFinestSweeps   1;              //
    cacheAgglomeration on;          //settings for GAMG solver
    agglomerator    faceAreaPair;   //
    nCellsInCoarsestLevel 128;      //
    mergeLevels     1;              //-----------------------------
}
```

Fields `tolearnce`, `relTol` and `maxIter` are stop criteria - when solver reaches any of those it will stop iterating. The other fields are options for chosen solver.

In the same file settings for SIMPLE algorithm implemented in `simpleFoam` are placed:

```
SIMPLE
{
    nNonOrthogonalCorrectors 0; //specifies repeated solutions of the
                                //pressure equation, used to update the
                                //explicit non-orthogonal correction

    residualControl             //stopping criteria for the algorithm
    {                           //based on residual
        p               1e-8;
        U               1e-8;
        "(k|epsilon|omega)" 1e-8; //one criterion for multiple fields
    }
}
```

One setting or solver can be applied on all the fields at once, by using parenthesis and "or" operator ("|") as seen in above example.

Another very important simulation parameter is also specified here - under-relaxation coefficients. User can specify a value for each of the variables:

```
relaxationFactors
{
    equations
    {
        U               0.5;
        p               0.3;
        k               0.5;
        omega           0.5;
    }
}
```

More in depth documentation of `fvSolution` file can be found here [7].

3.1.6 Mesh decomposition

As in most CFD codes in **OpenFOAM** mesh can be decomposed into many parts in order to enable palatalisation of necessary calculations. It is achieved by creating file named `decomposeParDict` in `system` directory and running `decomposePar` utility. Insides of this file describes how mesh should be divided:

```
FoamFile
{
    version     2.3;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


numberOfSubdomains  2;  //number of resulting domains

method  hierarchical;   //method of subdivision

coeffs
{
    n (2 1 1);          //number of divisions in each physical dimension
}

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

After parallel calculations are completed mesh and saved results can be recombined to original state by calling `reconstructPar`

3.1.7 Running the solver

After mesh is generated, initial and boundary conditions imposed and simulation settings set up, a solver can be run. It is achieved by using `simpleFoam` command. It will run SIMPLE segregated solver along with appropriate solvers for chosen turbulence models in serial mode (on one computation core).

Calculations can be run in parallel mode using MPI. In order to do that first host file must be created. In case of running solver on one computer with 2 cores it should look like this:

```
localhost cpu=2
```

Than solver is called with following command:

```
mpirun --hostfile <path to host file> -np 2 simpleFoam -parallel
```

It is often useful to redirect output from `simpleFoam` to log file and run it in the background by appending above command with:

```
> <name of log file> &
```

3.1.8 Understanding and plotting the residuals

Example output from `simpleFoam` solver can look in the following way:

```
Time = 5006

smoothSolver:
Solving for Ux, Initial residual = 2.539e-05, Final residual = 1.787e-07, No Iterations 4
smoothSolver:
Solving for Uy, Initial residual = 3.608e-05, Final residual = 2.802e-07, No Iterations 3
GAMG:
Solving for p, Initial residual = 5.525e-05, Final residual = 5.510e-10, No Iterations 586
time step continuity errors:
sum local = 2.544e-09, global = -1.575e-10, cumulative = -6.343e-10
DILUPBiCGStab:
Solving for omega, Initial residual = 3.645e-09, Final residual = 9.800e-17, No Iterations 3
```

```
smoothSolver:
Solving for k, Initial residual = 1.321e-05, Final residual = 7.552e-17, No Iterations 21
ExecutionTime = 11.74 s  ClockTime = 12 s
```

Algorithm outputs two residual values for each variable:

Initial residual - measure of how well old solution fits new system of algebraic equations.

Final residual - measure of convergence of linear solver.

The first one is, at the same time, a measure of convergence of solution of nonlinear partial differential equation.

In order to monitor solution convergence, plot of initial residuals against number of iterations is often useful. It can be achieved by using `gnuplot`:

```
gnuplot residuals
```

File `residuals` is a bash script[1] reading residual data from file named `log`:

```
set logscale y
set title "Residuals"
set ylabel 'Residual'
set xlabel 'Iteration'
plot "< cat log | grep 'Solving for Ux' | cut -d' ' -f9 | tr -d ','" title 'Ux' with lines,\
"< cat log | grep 'Solving for Uy' | cut -d' ' -f9 | tr -d ','" title 'Uy' with lines,\
"< cat log | grep 'Solving for Uz' | cut -d' ' -f9 | tr -d ','" title 'Uz' with lines,\
"< cat log | grep 'Solving for omega' | cut -d' ' -f9 | tr -d ','" title 'omega' with lines,\
"< cat log | grep 'Solving for k' | cut -d' ' -f9 | tr -d ','" title 'k' with lines,\
"< cat log | grep 'Solving for p' | cut -d' ' -f9 | tr -d ','" title 'p' with lines
pause 1
reread
```

It can be called while the simulation is running in the background and solver's output is redirected to a file.

3.1.9 Postprocessing

OpenFOAM has many postprocessing tools built in e.g. an utility used to calculate y^+ value in cells adjacent to viscous walls. Postprocessing routines can be used in three ways:

1. after the simulation - using `postProcess` command
2. after the simulation - running chosen solver with `-postProcess` option
3. during run time

The first option is the simplest. In order to calculate any field variable, for example Courant number or y^+ user must call:

```
postProcess -func <field name>
```

List of fields that can be calculated can be found by typing:

```
postProcess -list
```

For vorticity field, appropriate command will look like this:

```
postProcess -func vorticity
```

Program that is executed will calculate values of chosen field in each time step directory.

Another option is to include postprocessing tools inside `controlDict` file. It is useful in situations when postprocessing requires some additional code i.e. turbulence model as in the case of wall shear stress calculation. In order to do that user can append `controlDict` with the following code:

```
functions
{
    #includeFunc "wallShearStress"
}
```

[1] Courtesy of user *wolle1982* of `www.cfd-online.com` forum [13].

Any function that is listed here will be evaluated during run time (evaluation time is selected by `writeControl` option) and its outcome will be saved in appropriate time step directory. Functions can be also evaluated after simulation using `<chosen solver> -postProcess` command (more information about postprocessing and available utilities can be found here [5]).

Postprocessing can also be done in third party software **ParaView**. It is shipped with most of **OpenFOAM** distributions and can be lunched by calling `paraFoam` utility inside case directory.

3.1.10 Sampling simulation data

Various kinds of data, for example components of velocity field, can be extracted along specified line or surface. It is achieved by introducing a new line in `functions{...}` block in `controlDict` file:

```
#includeFunc "sample"
```

Next new file `case/system/sample` must be created with definitions of sampling surfaces. There are many options available. In case of this two dimensional simulation only extracting data along line is necessary (as seen in figure 1.1). Example `sample` file can look like this:

```
libs            ("libsampling.so"); //inclusion of executable library
type            sets;               //sets means sampling along lines
writeControl    writeTime;
interpolationScheme cellPoint;      //selection of interpolation scheme
setFormat       raw;                //output data format
sets                                //set definitions
(
    plateHorizontal           //first set, defined as a line
    {                         //to end with 1000 points
        type    face;
        axis    xyz;                //specifying what coordinates
                                    //should be outputted
        start   (0.1 0.2 0);
        end     (2 0.4 0);
        nPoints 1000;
    }

    plateBegin                      //second set defined as cloud of points
    {
        type    cloud;
        axis    xyz;
        points ( #include "points" ); //inclusion of file
                                      //with points definitions
    }
);

fields          (p U k omega);      //selection of fields to be sampled
```

As dictionary files are essentially a code in C++ when non uniform spacing is needed a claud of points can be simply "pasted" into `sample` file using standard `#include` preprocessor directive. Data in `case/system/points` files should has following format:

```
(x1 y1 z1)  //standard openFOAM vectors, not separated by any delimiter
(x2 y2 z2)
```

More in detail description of sampling abilities of **OpenFOAM** can be found [6]

3.2 Trimmed domain - case setup

Trimmed domain has four distinct boundaries (see figure 1.1):

A-B : horizontal inlet boundary that is situated 0.1 m from the start of the plate. Velocity profile should vary across this boundary.

B-C : skew boundary above plate surface. Velocity field is also not uniform on this boundary.

C-D : domain outlet.

D-A : plate surface modelled as viscous wall.

Boundary conditions on edges **C-D** and **D-C** are straight forward and set up in the same way as in previous case. In order to set appropriate velocity profiles on rest of boundaries `timeVaryingMappedFixedValue` type of boundary condition has been used.

3.2.1 Velocity and turbulent fields BCs - `timeVaryingMappedFixedValue`

In order to use `timeVaryingMappedFixedValue` type, first correct boundary condition should be defined in `0/U` file in following manner:

```
inlet
{
    type            timeVaryingMappedFixedValue;
    offset          (0 0 0);     //offset for three dimensional vector field
    setAverage      off;         //scale field to have an certain
                                 //average value
}
```

As name suggests this type of boundary condition lets user to specify fields that are both changing in time and space. It is achieved by creating a directory named the same as chosen patch (in this case it was `inlet`) in `constant/boundaryData` directory. Files inside this folder have similar structure as definitions of boundary conditions in `case` directory (see figure 3.3). For each chosen time step patch field values must be defined along with interpolation points (that are time invariant). If only initial time is specified boundary conditions will not change in time.

```
case
└─ constant ..................................... constants folder
   └─ boundaryData ............... data for timeVaryingMappedFixedValue boundary condition
      ├─ inlet ..................... file with space varying data for inlet patch
      │  ├─ points ..... file describing points where sampling was made
      │  ├─ 0 ..................................... initial time step
      │  │  ├─ U ..................... velocity field
      │  │  ├─ k ..... turbulent kinetic energy field
      │  │  └─ omega ..... turbulent dissipation rate
      │  └─ <some other time step>
      └─ <name of other patch>
```

Figure 3.3: File structures defining `timeVaryingMappedFixedValue` conditions

In order to define points and values of fields in those points sampling along lines were used. In this case flow is two dimensional. Having in mind that **OpenFOAM** is a three dimensional solver, it is better to provide tho sets of points for each patch - laying on both of `empty` planes[2]. It will ensure that values are interpolated correctly across the "empty" dimension. File with points definition should look in the following way:

```
4 //number of points
(
(1.0000000000e-01 0.0000000000e+00 0.0000000000e+00) //coordinates
(1.0000000000e-01 2.0000000000e-06 0.0000000000e+00)
(1.0000000000e-01 4.4430910000e-06 0.0000000000e+00)
(1.0000000000e-01 7.4274379000e-06 0.0000000000e+00)
)
```

Velocity field file `U` is very similar in structure:

[2]It can be achieved for example by copying and processing sampled data in a simple Matlab script.

```
4 //number of points
(
(0.0000000000e+00 0.0000000000e+00 0.0000000000e+00) //vector components
(1.5774773000e+00 2.9655442000e-06 0.0000000000e+00)
(3.5008039000e+00 1.4358657000e-05 0.0000000000e+00)
(5.8499814000e+00 4.0020820000e-05 0.0000000000e+00)
)
```

3.2.2 Pressure boundary conditions

A careful reader might have noticed that **OpenFOAM** is somewhat contradictory in one place. Providing boundary conditions for both pressure and velocity at the inlet to the domain is necessary. If those boundary conditions are not exactly complementary, it can be seen as overdetermining the Navier-Stokes equations.

In the first of described cases this problem can be easily avoided since both zero Neumann condition for pressure and uniform velocity value are correct boundary conditions. Unfortunately neither **A-B**, nor **B-C** boundaries are defined along streamlines, and velocity field is not uniform in their vicinity, which means that zero Neumann boundary condition for pressure cannot be applied here. Instead `fixedFluxExtrapolatedPressure` type of boundary condition should be chosen. It is described in the **OpenFOAM** documentation as: "This boundary condition sets the pressure gradient to the provided value such that the flux on the boundary is that specified by the velocity boundary condition."

Chapter 4

Results

In this chapter results of both simulations are compared and presented as graphs of velocity components and turbulent quantities along lines $x = 0.2$ (figures 4.1 - 4.5) and $x = 0.2$ (figures 4.6 - 4.10) . The first sampling line is located 0.1 m past the inlet to the trimmed domain (and 0.2 m from flat plate starting point, see figure 1.1). The second one is located near the outlet of both domains.

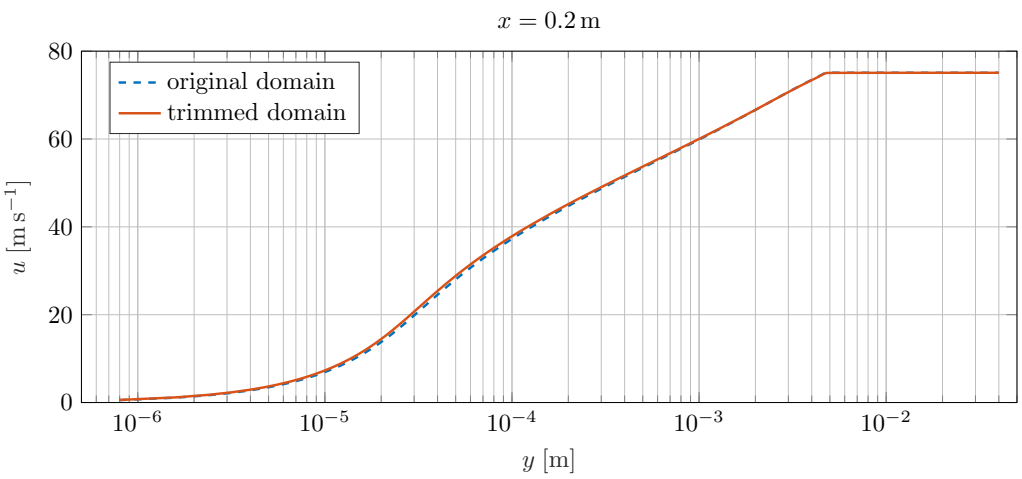

Figure 4.1: Graph of x component of velocity field along the $x = 0.2$ line from both original domain and the trimmed one.

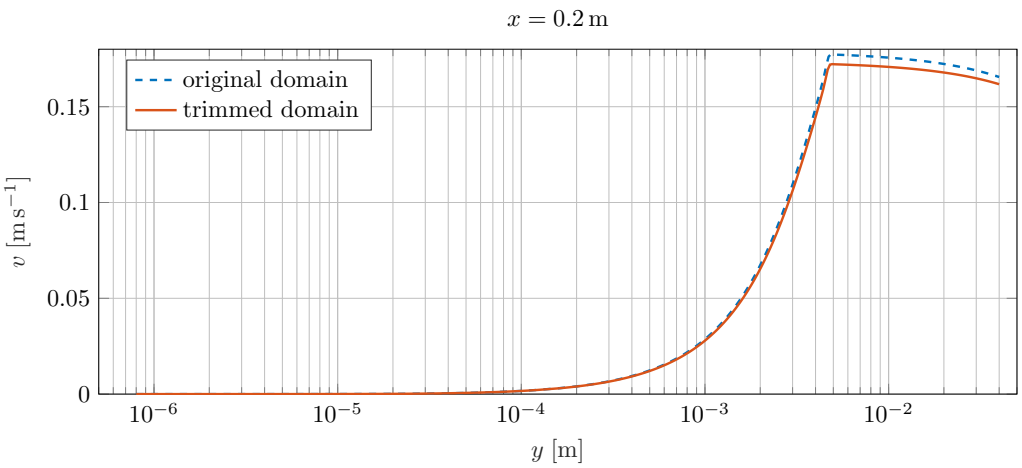

Figure 4.2: Graph of y component of velocity field along the $x = 0.2$ line from both original domain and the trimmed one.

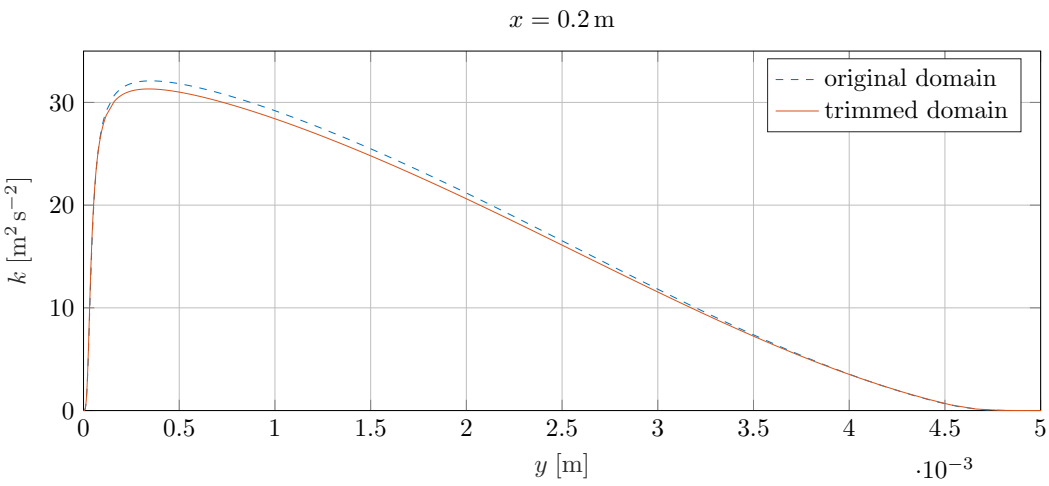

Figure 4.3: Graph of turbulent kinetic energy field along the $x = 0.2$ line from both original domain and the trimmed one.

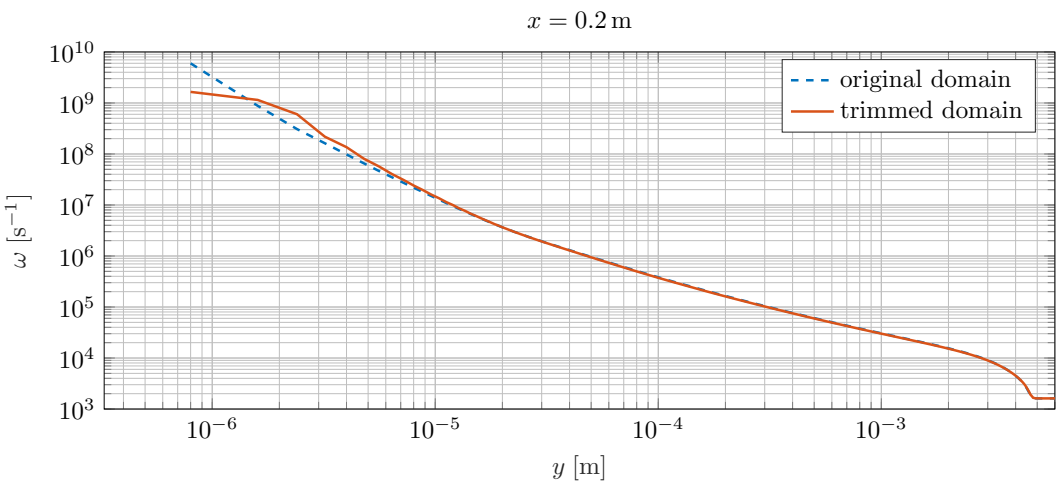

Figure 4.4: Graph of specific dissipation rate field along the $x = 0.2$ line from both original domain and the trimmed one.

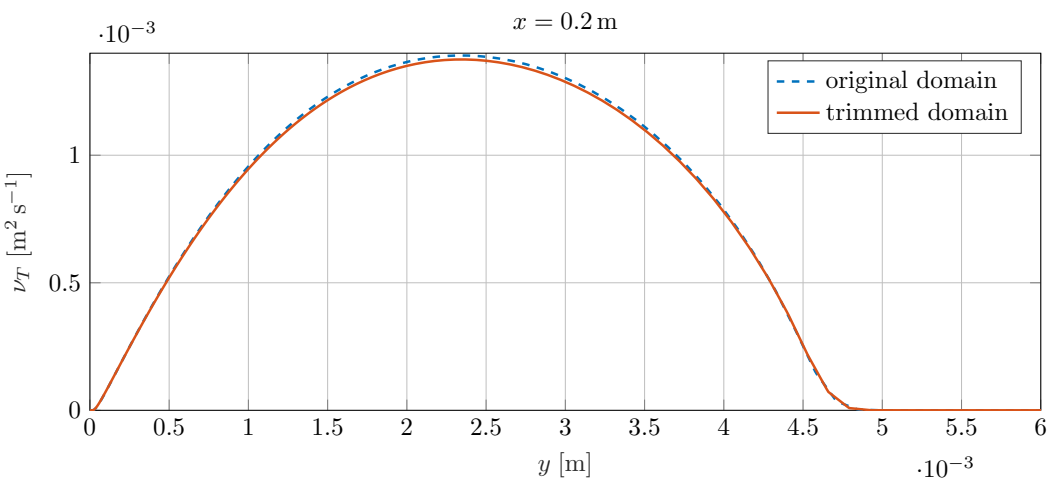

Figure 4.5: Graph of turbulent viscosity field along the $x = 0.2$ line from both original domain and the trimmed one.

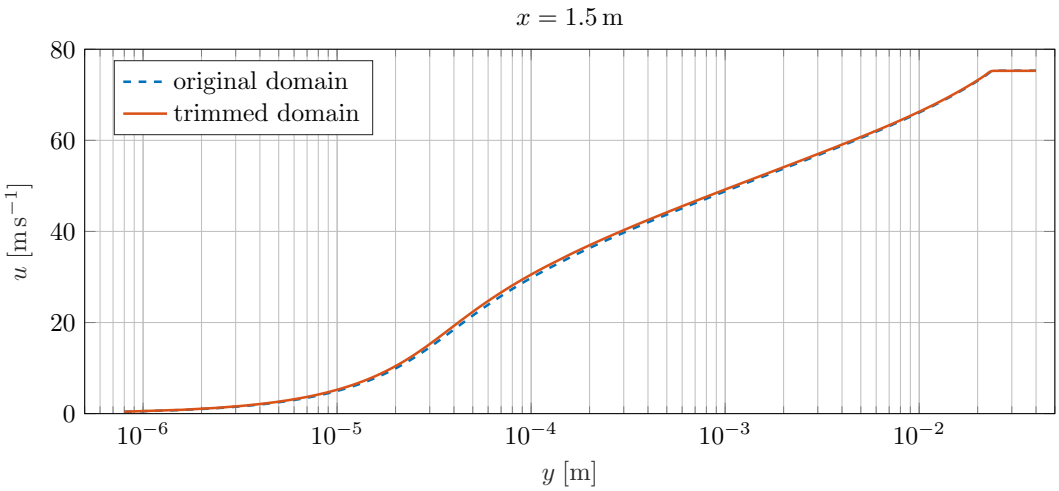

Figure 4.6: Graph of x component of velocity field along the $x = 1.5$ line from both original domain and the trimmed one.

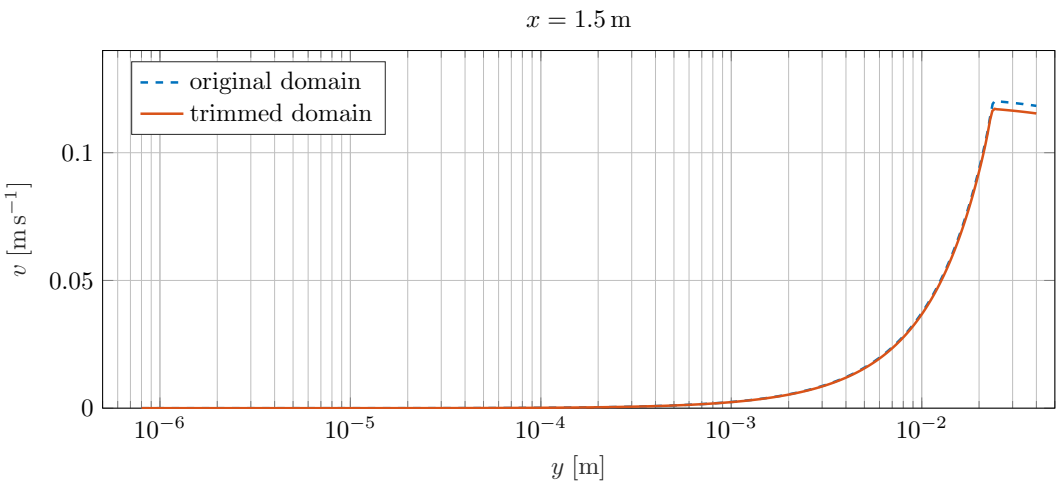

Figure 4.7: Graph of y component of velocity field along the $x = 1.5$ line from both original domain and the trimmed one.

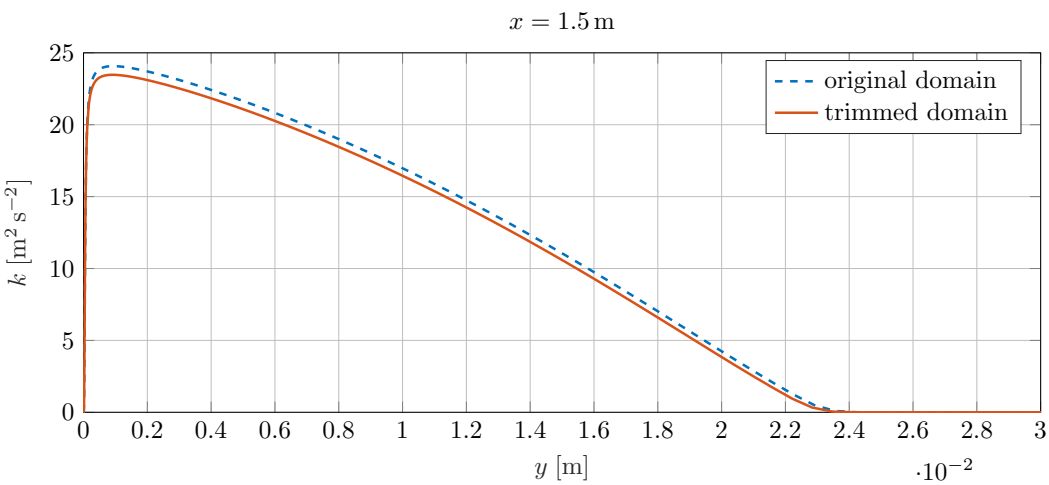

Figure 4.8: Graph of turbulent kinetic energy field along the $x = 1.5$ line from both original domain and the trimmed one.

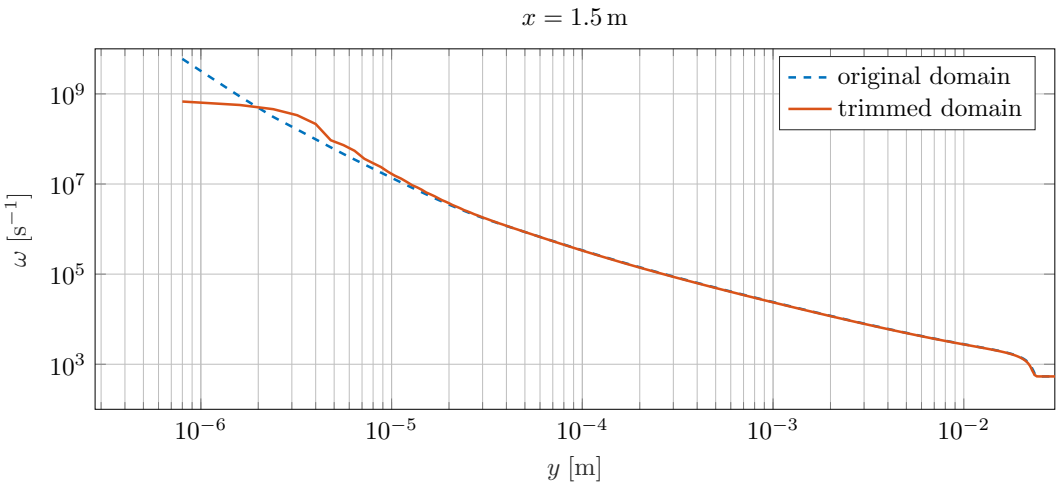

Figure 4.9: Graph of specific dissipation rate field along the $x = 1.5$ line from both original domain and the trimmed one.

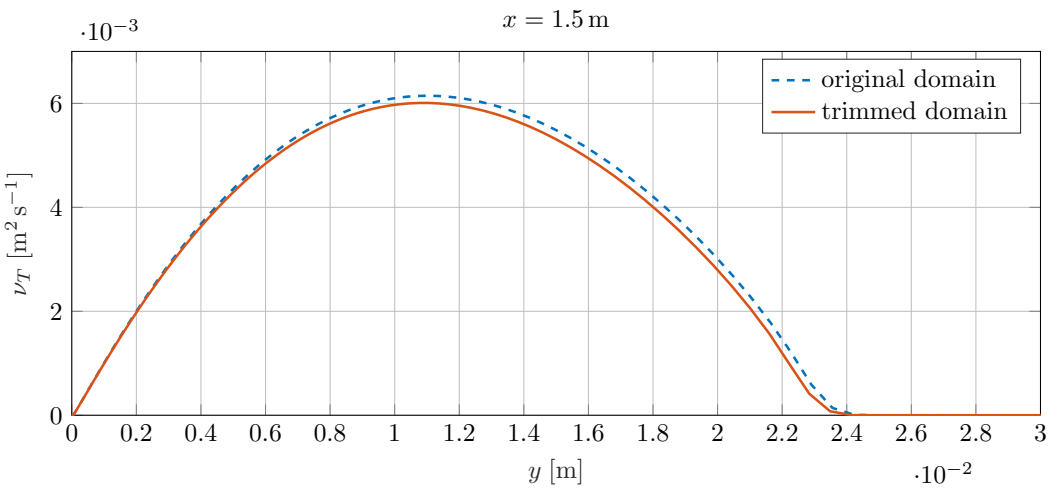

Figure 4.10: Graph of turbulent viscosity field along the $x = 1.5$ line from both original domain and the trimmed one.

Chapter 5

Conclusions

Both simulations achieved very similar results differing very slightly (relative differences in field quantities were 2 % at most on both of sampled lines). Those differences can be attributed mostly to two factors:

- Different mesh resolutions in both simulations. Each mesh was graded in the direction of the plates wall in order to achieve correct y^+ value (around 0.05 in case of original domain and 0.68 in the latter one), however mesh created for the second simulation is noticeably coarser. The reasoning behind this was to increase convergence rate by limiting high values of ω that can arise very close to the viscous wall. Nevertheless both meshes have adequate resolution to represent physical phenomena that arise in that type of flow.
- In the boundary layer region sampling points were separated by logarithmically rising intervals in order to account for sharply changing turbulent quantities. Sampled data was then linearly interpolated in order to impose boundary conditions. This interpolation might have introduced slight error in the fields that was than convected along plates surface.

In conclusion, goals that ware stated in the begging of this work were achieved. Author learned basic usage of **OpenFOAM** along with some other advanced practices i.e. tweaking linear solver settings or utilising advanced time and space varying boundary conditions.

In the same time, this work has proven that **OpenFOAM** can be used as a tool for validation of various CFD models against experiments. Sampled data from those experiments can be used as boundary conditions in order to observe flow behaviour and assert validity of chosen model.

Bibliography

[1] David C Wilcox. *Turbulence Modeling for CFD (Third Edition) (Hardcover)*. 01 2006.

[2] Langley Research Center. Turbulence modeling resource. `https://turbmodels.larc.nasa.gov`, Accessed: 05.01.2019.

[3] CFD Direct. Mesh generation with the `blockMesh` utility. `https://cfd.direct/openfoam/user-guide/v6-blockmesh/`, Accessed: 05.01.2019.

[4] CFD Direct. Numerical schemes. `https://cfd.direct/openfoam/user-guide/v6-fvSchemes/#x20-1130159`, Accessed: 05.01.2019.

[5] CFD Direct. Post-processing cli. `https://cfd.direct/openfoam/user-guide/v6-post-processing-cli/`, Accessed: 05.01.2019.

[6] CFD Direct. Sampling data. `https://www.openfoam.com/documentation/user-guide/userse21.php`, Accessed: 05.01.2019.

[7] CFD Direct. Solution and algorithm control. `https://cfd.direct/openfoam/user-guide/v6-fvsolution/`, Accessed: 05.01.2019.

[8] CFD Direct. Time and data input/output control. `https://cfd.direct/openfoam/user-guide/v6-controldict/`, Accessed: 05.01.2019.

[9] L.D. Landau and E.M. Lifshitz. *Fluid Mechanics*. Number v. 6. Elsevier Science, 2013.

[10] Osborne Reynolds. An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels. *Philosophical Transactions of the Royal Society of London*, 174:935–982, 1883.

[11] Wojciech Sadowski. Git respository with openfoam case files. `https://github.com/szynka12/oFoamSVBC`, Accessed: 05.01.2019.

[12] Philippe R. Spalart. On the role and challenges of cfd in the aerospace industry. 2016.

[13] wolle1982. How to plot the residuals (and forces) graphically on screen on-the-fly. `https://www.cfd-online.com/Forums/openfoam-community-contributions/64146-tutorial-how-plot-residuals.html`, Accessed: 05.01.2019.