CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

---

Project work:

# Modeling of bed roughness using a geometry function and forcing terms in the momentum equations

---

Developed for OpenFOAM-2.2.x

*Author:*
Jonatan Margalit

*Peer reviewed by:*
Håkan Nilson
Thomas Vyzikas
Simon Lindberg



January 5, 2015

# Contents

**1 Introduction** **2**

1.1 Theoretical background . . . . . 2

1.2 Getting started . . . . . 3

**2 Preparing the solver** **3**

2.1 Copying the pimpleFoam solver . . . . . 3

2.2 Modifying the solver code . . . . . 4

**3 Preparing the case** **6**

3.1 Copying a case from the Tutorials directory . . . . . 6

3.2 Defining the domain and the mesh . . . . . 7

3.3 Setting initial conditions in the 0 folder . . . . . 10

3.4 Generating the rough bed . . . . . 14

3.5 Adding source terms with fvOptions . . . . . 18

3.6 Configuring simulation controls with controlDict . . . . . 19

3.7 The fvSchemes and fvSolution dictionaries . . . . . 21

3.8 Configuring the turbulence parameters . . . . . 23

3.9 Configuring the flow parameters . . . . . 25

3.10 Running the case . . . . . 25

**4 Post-processing** **25**

4.1 postChannel utility for spatial averaging . . . . . 25

4.2 Plotting with Gnuplot . . . . . 28

# 1 Introduction

This tutorial guides how to set up an OpenFOAM case with a rough bed using porosity to replicate the bed elements. The case set-up is based on the channel395 tutorial; an LES modeled turbulent channel-flow with periodic boundary conditions in the streamwise and spanwise directions. The solver used is the pimpleFoam with additional source terms for the pressure gradient and the porosity, using the fvOptions. Post-processing is done by averaging the fields in time and space, in order to have mean depth profiles.

The tutorial is proposing a way to implement Stoesser's [1] rough bed simulation, in OpenFOAM. The idea is that, since wall functions have a somewhat limited success when applied in LES modelings, a rough-bed geometry is better as the near-bed flow is resolved. Stoesser proposed to simply model the bed within the domain by blocking the flux through the cells that represent the bed. This simplifies the computation since a structured mesh can be used, as long as the near-bed cells are in the order of magnitude of the mean grain diameter. Stoesser demonstrated that for averaged quantities, the result show a good representation of laboratory measured quantities.

The user of this tutorial does not need to have much understanding of the structure of OpenFOAM as it is a step by step guide for the rough-bed implementation. However, for detailed explanation of the dictionaries and case structure, as well as the underlying equations, it is recommended that the user completes the tutorials in the OpenFOAM user guide first and consults the text-books about CFD.

## 1.1 Theoretical background

The following is a brief summary of the governing equations.

The Navier-Stokes equation with index notations is given by

$$\rho\left(\frac{\partial u_i}{\partial t}+u_j\frac{\partial u_i}{\partial x_j}\right)=\rho g_i+\frac{\partial}{\partial x_j}\left[-p\delta_{ij}+\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\right]. \tag{1}$$

For a pressure driven channel flow, the time-averaged equation in the flow direction, $x$, is reduced into

$$\frac{\partial}{\partial y}\left(\mu\frac{\partial \overline{u}}{\partial y}-\rho\overline{u'v'}\right)=\frac{\partial \overline{p}}{\partial x}, \tag{2}$$

where the overline denotes time-averaged quantities. In Newtonian fluids the shear is related linearly to the velocity gradient. In turbulence modeling it is customary to consider the effective shear as comprising of a viscous part, and the Reynolds stresses. This is defined as

$$\overline{\tau}\equiv\overline{\tau}_{eff}=\mu\frac{\partial \overline{u}}{\partial y}-\rho\overline{u'v'}, \tag{3}$$

which implies that Eq. (2) may be written as

$$\frac{\partial \overline{\tau}}{\partial y}=\frac{\partial \overline{p}}{\partial x}. \tag{4}$$

Integrating Eq. (4) and applying the boundary condition $\tau(h)=0$ ($h$ being the depth), leads to the solution of the shear stress

$$\overline{\tau}=-\frac{\partial \overline{p}}{\partial x}h\left(1-\frac{y}{h}\right). \tag{5}$$

For a flow over a smooth bed the velocity is in the logarithmic layer given by

$$\frac{u}{U_f}=\frac{1}{\kappa}\ln\left(y^+\right)+5.5, \tag{6}$$

where $U_f = \sqrt{\tau_b/\rho}$ is the friction velocity, $\tau_b$ is the bed shear stress, $\kappa = 0.4$ is the von Karman constant and $y^+ = yU_f/\nu$ is the depth wise wall units. Similarly for a rough-bed the flow the velocity is given by

$$\frac{u}{U_f} = \frac{1}{\kappa} \ln\left(\frac{30y}{k_s}\right), \tag{7}$$

where $k_s$ is Nikuradse equivalent sand roughness.

For LES the governing equation is the spatially averaged Navier-Stokes equation:

$$\frac{\partial \rho \overline{u}_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho \overline{u}_i \overline{u}_j\right) = -\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\mu + \mu_{SGS}\right)\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)\right] + \rho \overline{g}_i \tag{8}$$

where overline denotes spatial-averaged quantities and $\mu_{SGS}$ is the Sub-grid scale viscosity which has to be modeled.

### 1.2 Getting started

The commands in the tutorial are intended to be executed in a Linux terminal where OpenFOAM-2.2.x has been installed. All framed lines are intended to be executed sequentially in the order of appearance.

Start by sourcing your OpenFOAM-2.2.x. For example, if OpenFOAM has been installed into the user `$HOME` directory, you can source it by the line:

```
source $HOME/OpenFOAM/OpenFOAM-2.2.x/etc/bashrc
```

## 2 Preparing the solver

The channel flow in this tutorial is solved using the pimpleFoam solver, which is included in the OpenFOAM installation directories. A small modification is applied to the solver in order to make it write out the viscous and SGS stresses at every time step.

The pimpleFoam solver is a large time-step transient solver for incompressible flow using the PIMPLE (merged PISO-SIMPLE) algorithm [2]. It includes turbulence modeling and run-time selection finite volume options.

### 2.1 Copying the pimpleFoam solver

Make a copy of the pimpleFoam solver in the user directory:

```
cd $FOAM_APP
cp -r --parents solvers/incompressible/pimpleFoam $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/solvers/incompressible
```

Rename the copied pimpleFoam solver:

```
mv pimpleFoam pimpleFoam_mod
cd pimpleFoam_mod
mv pimpleFoam.C pimpleFoam_mod.C
```

Cleanup the directory and delete the unnecessary folders and files:

```
wclean
rm -rf SRFPimpleFoam/ pimpleDyMFoam/ Allwmake Make/linux*
```

Now execute the Linux command `tree` from within the solver directory and verify that the output is as the following:

```
.
├── createFields.H
├── Make
│   ├── files
│   └── options
├── pEqn.H
├── pimpleFoam_mod.C
└── UEqn.H
```

## 2.2 Modifying the solver code

The next step is to modify the solver code in order to make it output the viscous (`nuGradU`) and the SGS (`B`) stress tensors. Start by opening the file `pimpleFoam_mod.C` in your preferred text editor. For example, the following command will open the file in the `gedit` editor:

```
gedit pimpleFoam_mod.C
```

Now have a look at the code within the time iteration loop. After the PIMPLE loop, make the solver write the stress tensors `nuGradU` and `B`, so your script should look like this:

Time iteration loop in pimpleFoam_mod.C

```
    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }
```

```
            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        nuGradU = laminarTransport.nu()*-2*dev(symm(fvc::grad(U)));
        B = turbulence->R();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
```

Save and exit the editor.

Next, you need to initialize the two new fields `nuGradU` and `B`, so that they are pre-allocated in the memory before the simulation starts. This is done in `createFields.H`, which is a header file that is called at the beginning of a simulation to initialize all the necessary fields. Open `createFields.H`:

```
gedit createFields.H
```

Add the following lines to the end of `createFields.H`:

Additions to createFields.H

```
volSymmTensorField nuGradU
    (
         IOobject
        (
            "nuGradU",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
            laminarTransport.nu()*-2*dev(symm(fvc::grad(U)))
    );

volSymmTensorField B
    (
         IOobject
        (
            "B",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
```

```
            laminarTransport.nu()*-2*dev(symm(fvc::grad(U)))
    );
```

Notice that `nuGradU` and `B` are created with the same expression using `laminarTransport.nu()`. For `B` this expression is simply used to initiate the field with the correct structure and dimensions. The correct values are calculated in each time step.

Save and exit the editor and now open `Make/files`:

```
gedit Make/files
```

Modify the content to the following:

Content of Make/files

```
pimpleFoam_mod.C

EXE = $(FOAM_USER_APPBIN)/pimpleFoam_mod
```

Save and exit. This is done to tell the OpenFOAM compiler to compile the `pimpleFoam_mod.C` file and put the executable in a directory where it is found by the OpenFOAM environment. Then it can be called any time, as any other solver.

Finish by compiling the solver with the command `wmake` from the main solver directory:

```
wmake
```

If no error message appears, proceed to the next section.

# 3 Preparing the case

The modified solver will in the following be used to simulate the flow in an open channel. For this the tutorial case `channel395` will be used. This case is made for running with LES in a closed channel with $Re_\tau = U_f h/\nu = 395$. The case is run, copied and modified so it can be used to compare the flow over a smooth and a rough bed.

## 3.1 Copying a case from the Tutorials directory

Copy the tutorial case `channel395` into the `run` directory:

```
cp -r $FOAM_TUTORIALS/incompressible/pimpleFoam/channel395 $FOAM_RUN
run
```

In the `run` folder make another copy of `channel395` and call it `roughChannel`:

```
cp -r channel395 roughChannel
```

Now run the `channel395` case while you configure the `roughChannel`:

```
blockMesh -case channel395
pimpleFoam_mod -case channel395 > channel395/log.run &
```

Verify that `channel395` is running correctly by peeking at the `log.run` file:

```
tail -f channel395/log.run
```

Exit `tail` with Ctrl+C.

Head over to the `roughChannel` case and cleanup the directory:

```
cd roughChannel
rm -rf 0.org Allrun
```

View the directory organization with the Linux `tree` command:

```
tree
```

```
.
├── 0
│   ├── B.gz
│   ├── k.gz
│   ├── nuSgs.gz
│   ├── nuTilda.gz
│   ├── p.gz
│   └── U.gz
├── constant
│   ├── LESProperties
│   ├── polyMesh
│   │   ├── blockMeshDict
│   │   └── boundary
│   ├── postChannelDict
│   ├── transportProperties
│   └── turbulenceProperties
└── system
    ├── controlDict
    ├── decomposeParDict
    ├── fvOptions
    ├── fvSchemes
    └── fvSolution
```

## 3.2 Defining the domain and the mesh

Edit the channel in order to create an open-channel. Open `constant/polyMesh/blockMeshDict` in a text editor and edit the `vertices` section as the following:

vertices definition in blockMeshDict

```
vertices
(
    (0 0 0)
    (4 0 0)
    (0 0.2 0)
    (4 0.2 0)
    (0 1 0)
    (4 1 0)
    (0 0 2)
    (4 0 2)
    (0 0.2 2)
    (4 0.2 2)
    (0 1 2)
    (4 1 2)
);
```

Then edit the `blocks` section in the following way:

blocks definition in blockMeshDict

```
blocks
(
    hex (0 1 3 2 6 7 9 8) (60 25 40) simpleGrading (1 1 1)
    hex (2 3 5 4 8 9 11 10) (60 35 40) simpleGrading (1 6 1)
);
```

Finally, change the boundary type of the `topWall` into a `patch`:

topWall definition in blockMeshDict

```
topWall
    {
        type            patch;
        faces           ((4 10 11 5));
    }
```

Save and exit the editor.

Run `blockMesh` from the main case directory:

```
blockMesh
```

Your terminal output should look like this:

Terminal output from the command blockMesh

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
```

```
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 2.2.x-61b850bc107b
Exec   : blockMesh
Date   : Nov 28 2014
Time   : 15:43:31
Host   : "COMPUTER_NAME"
PID    : 3665
Case   : /home/USER/OpenFOAM/USER-2.2.x/run/roughChannel
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Disallowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Creating block mesh from
    "/home/USER/OpenFOAM/USER-2.2.x/run/roughChannel/constant/polyMesh/blockMeshDict"
Creating curved edges
Creating topology blocks
Creating topology patches

Creating block mesh topology

Check topology

        Basic statistics
                Number of internal faces : 1
                Number of boundary faces : 10
                Number of defined boundary faces : 10
                Number of undefined boundary faces : 0
        Checking patch -> block consistency

Creating block offsets
Creating merge list .

Creating polyMesh from blockMesh
Creating patches
Creating cells
Creating points with scale 1

Writing polyMesh
----------------
Mesh Information
----------------
  boundingBox: (0 0 0) (4 1 2)
  nPoints: 152561
  nCells: 144000
  nFaces: 440400
  nInternalFaces: 423600
```

```
----------------
Patches
----------------
  patch 0 (start: 423600 size: 2400) name: bottomWall
  patch 1 (start: 426000 size: 2400) name: topWall
  patch 2 (start: 428400 size: 1500) name: sides1_half0
  patch 3 (start: 429900 size: 1500) name: sides1_half1
  patch 4 (start: 431400 size: 2100) name: sides2_half0
  patch 5 (start: 433500 size: 2100) name: sides2_half1
  patch 6 (start: 435600 size: 1000) name: inout1_half0
  patch 7 (start: 436600 size: 1000) name: inout1_half1
  patch 8 (start: 437600 size: 1400) name: inout2_half0
  patch 9 (start: 439000 size: 1400) name: inout2_half1

End
```

Figure 1 shows the meshes of `channel395` and `roughChannel`.

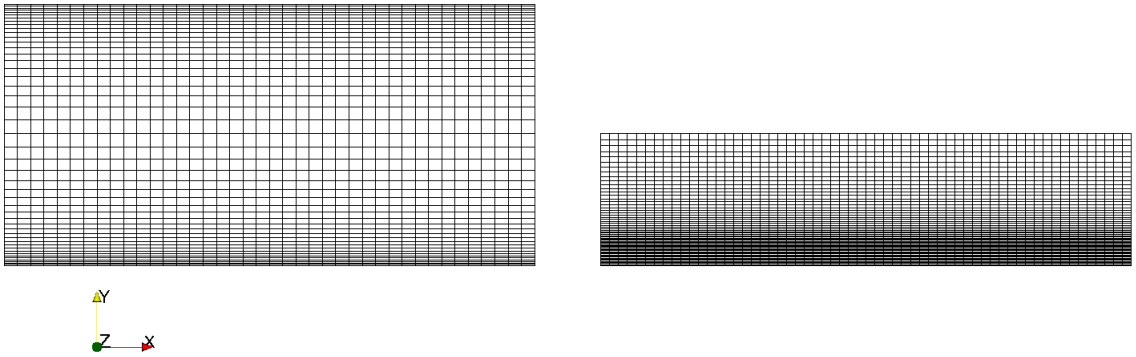


Figure 1: **Left**: Mesh of channel395, **Right**: Mesh of roughChannel.

## 3.3 Setting initial conditions in the 0 folder

The `0` folder contains the boundary and initial conditions of the simulated fields. Since we changed the mesh in `blockMeshDict`, the current field values do not match the mesh any more. We can either start all fields from 0, which will require very long simulation time to develop the flow, or map the developed fields using interpolation from `channel395`. In the following it is shown how set the fields to 0, and then map the lower half field values of `channel395`, into the `roughChannel` fields.

To modify the fields in the 0 folder, we can conveniently use the `changeDictionaryDict`, which allows the modification of the fields altogether. Lets start by copying the `changeDictionaryDict` dictionary from another tutorial, into the system folder:

```
cp -r $FOAM_TUTORIALS/multiphase/interFoam/ras/damBreakPorousBaffle/system/ \
changeDictionaryDict system/
```

Open up `changeDictionaryDict` and edit the contents to be as in the frame below.

What we do is specify the internal fields to be uniform 0 for all the files. For vectors this is specified as `internalField uniform (0 0 0);`, and for scalars `internalField uniform 0;`.

Furthermore, for the velocity set the boundary condition at the `topWall` to be of the type `slip`, in order to have a rigid lid surface. Similarly, set the `topWall` boundary condition of `k` to `zeroGradient`.

Contents of changeDictionaryDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      changeDictionaryDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dictionaryReplacement
{
    p
    {
        internalField uniform 0;
    }

    U
    {
        internalField uniform (0 0 0);
        boundaryField
        {
            topWall
            {
                type slip;
            }
        }
    }

    k
    {
        internalField uniform 0;
        boundaryField
        {
            topWall
            {
                type zeroGradient;
            }
        }
    }

    nuSgs
    {
        internalField uniform 0;
```

```
    }
}

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

Now execute the changes by running

```
changeDictionary
```

from the case directory.

Note that this step is necessary before mapping fields, because the `mapFields` utility is not able to map fields between two cases with different meshes and non-uniform internal fields. Alternatively, you could use the `0.org`, where all fields are uniform 0.

You will now map the fields of the lower half of the channel in `channel395` into your rough channel. This is done by copying the `mapFieldsDict` into the `system` folder:

```
cp -r $FOAM_APP/utilities/preProcessing/mapFields/mapFieldsDict system/
```

Now inside the `mapFieldsDict` you specify the patches that coincide for the two cases in the `patchMap` section. Then enter the name of the patch in the current case that cuts the geometry. This would be `topWall` since it is laying exactly at the middle of the `\channel395` geometry. Your `mapFieldsDict` should now look like this:

mapFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      mapFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// List of pairs of target/source patches for mapping
patchMap
(
    bottomWall bottomWall
    sides1_half0 sides1_half0
    sides2_half0 sides2_half0
    inout1_half0 inout1_half0
    inout2_half0 inout2_half0
```

```
);

// List of target patches cutting the source domain (these need to be
// handled specially e.g. interpolated from internal values)
cuttingPatches
(
    topWall
);

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

Now map the fields from `channel395` into the rough channel:

```
mapFields ../channel395/
```

This should produce the following terminal output:

Terminal output from mapFields

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 2.2.x-61b850bc107b
Exec   : mapFields ../channel395
Date   : Nov 28 2014
Time   : 15:50:54
Host   : "COMPUTER_NAME"
PID    : 3791
Case   : /home/USER/OpenFOAM/USER-2.2.x/run/roughChannel
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Disallowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Source: ".." "channel395"
Target: "/home/USER/OpenFOAM/USER-2.2.x/run" "roughChannel"

Create databases as time

Source time: 0
Target time: 0
Create meshes

Source mesh size: 60000        Target mesh size: 144000


Mapping fields for time 0
```

```
    interpolating p
    interpolating k
    interpolating nuSgs
    interpolating nuTilda
    interpolating U
    interpolating B

End
```

Open up the field files again and verify that the internalFields have been created and that the boundary conditions are correct. Give a special attention to the boundary conditions that you modified earlier.

## 3.4 Generating the rough bed

In this section you will generate a zone of cells near the channel-bed, which are to be used to simulate the bed roughness, in a similar approach to the one demonstrated by Stoesser [1].

The following script is used to generate a random field of 3D bars, of which heights follow a normal distribution. The script is quite poorly coded in C++ and is expected to by optimized for use as a tool in the future. Comments have been added for the sake of clarity.

The main idea behind the script is to define the geometry of the domain in the spanwise and streamwise directions. Then a set of 3D bars are generated, where the height is a function of the mean grain diameter. This is arranged in a list of coordinates in a new file, which the OpenFOAM `topoSetDict` is able to read.

For optimal results, the bars should be larger than the mesh cell sizes, i.e. so the roughness is well discretized.

Start by creating a new file called `boxes.C` in the `system` folder and paste the code lines into it:

```
cd system
gedit boxes.C
```

Rough bed generation script to be pasted in boxes.C

```
// C++ script that generates a file 'boxes.txt' that contains the coordinates
// to 3D bars, of which heights follow a normal distribution

#include <iostream>
#include <fstream>
#include <random>
#include <math.h>
using namespace std;

int main()
{
    const double d50=0.024;     // mean grain diameter
    const double c=3;           // scaling factor
    const double x=4;           // streamwise domain size
```

```
const double z=2;           // spanwise domain size
const double mean=0;        // mean
const double sigma=0.5*d50; // standard deviation
double dx=c*d50;            // streamwise spacing
double dz=c*d50;            // spanwise spacing
const int nx=ceil(x/dx);    // number of boxes streamwise
const int nz=ceil(z/dz);    // number of boxes spanwise

// outputs to terminal
cout << "Mean grain diameter: " << d50 << endl;
cout << "Number of boxes in streamwise direction: " << nx << endl;
cout << "Number of boxes in spanwise direction: " << nz << endl;
cout << "Streamwise bar size: " << dx << endl;
cout << "Spanwise bar size: " << dz << endl;

// generate coordinates with the depthwise being random
default_random_engine generator;
normal_distribution<double> distribution(mean,sigma);

double X[nx+1];
double Z[nz+1];
double number[nx][nz];
double small = number[0][0];

for (int i=0; i<nx; i++)
{
    for (int j=0; j<nz; j++)
    {
        number[i][j] = distribution(generator);
    }
}

// find the lowest depthwise coordinate and scale all so it becomes 0
for (int i=0; i<nx; i++)
{
    for (int j=0; j<nz; j++)
    {
        if(small>number[i][j])
            small = number[i][j];
    }
}
cout << "Mean 0 bed located " << abs(small) << " above bottomWall" << endl;
for (int i=0; i<nx; i++)
{
    for (int j=0; j<nz; j++)
    {
            number[i][j] += abs(small) ;
    }
}

// output coordinates to file
for (int i=0; i<nx+1; i++)
{
    X[i] = i*dx;
```

```
    }
    for (int i=0; i<nz+1; i++)
    {
        Z[i] = i*dz;
    }

    ofstream myfile;
    myfile.open ("boxes.txt");
    for (int i=0; i<nx; i++)
    {
        for (int j=0; j<nz; j++)
        {
            myfile << "(" << X[i] << " 0 " << Z[j] << ")(" << X[i+1] << " " <<
             number[i][j] << " " << Z[j+1] << ")\n";
        }
    }
    myfile.close();
  return 0;
}
```

Save and exit the editor.

Now compile the script in order to create an executable file:

```
g++ -std=c++11 boxes.C -o boxes
```

and now execute `boxes` to create the coordinate list file `boxes.txt`:

```
./boxes > log.boxes
```

The `topoSet` tool creates a list of sets at `constant/polyMesh/sets` based on it's dictionary input. Here we want it to read the coordinates in the file `boxes.txt` and create a set of all cells that are confined by the bars.

Copy the `topoSetDict` file into the `system` folder:

```
cp -r $FOAM_APP/utilities/mesh/manipulation/topoSet/topoSetDict .
```

Open and edit the file:

```
gedit topoSetDict
```

You will notice that all the various options for the inputs are commented. You can either leave those commented or clean up, so that your code looks like the following:

Content of topoSetDict

```
actions
(
```

```
    {
        name    bed;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            boxes
            (
                #include "boxes.txt"
            );
        }
    }
);
```

As can be seen, `topoSet` creates a `cellSet` called `bed`, and it is generated using the source `boxToCell`, which reads the coordinates from `boxes.txt`.

Save and exit `topoSetDict`. To execute `topoSet`, simply run the command from the case directory:

```
cd ..
topoSet
```

And now you want to convert the set of cells into a zone, in order to be able to assign the bed properties to it. This can be done by simply executing the command:

```
setsToZones
```

You should now successfully have defined a zone of cells which can be referred to by it's name: `bed`. The `topoSet` created the file `constant/polyMesh/sets/bed` and `setsToZones` used it to create the zone in `constant/polyMesh/cellZones`.

Figure 2 shows the cells of the `bed` zone.

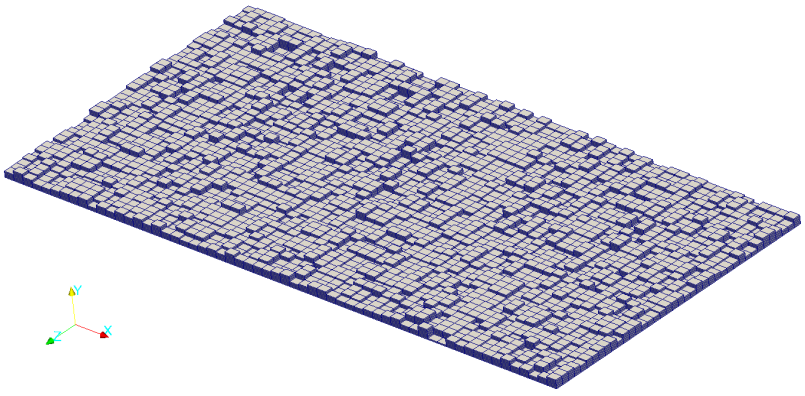

Figure 2: Cells representing the rough bed

## 3.5 Adding source terms with fvOptions

The `fvOptions` framework is used in order to allow users to select any physics that can be represented as sources or constraints on the governing equations, e.g. porous media, MRF and body forces [3]. For our case we will use it in order to give the bed-cells a porosity and to drive the flow by a pressure gradient.

Edit the `system/fvOptions` file and add the following porosity specification such that your file should look like this:

Content of fvOptions

```
momentumSource
{
    type            pressureGradientExplicitSource;
    active          off;            //on/off switch
    selectionMode   all;       //cellSet // points //cellZone

    pressureGradientExplicitSourceCoeffs
    {
        fieldNames  (U);
        Ubar        ( 0.1335 0 0 );
    }
}

porosity
{
    type explicitPorositySource;
    active true;
    selectionMode cellZone;
    cellZone bed;

    explicitPorositySourceCoeffs
    {
        type DarcyForchheimer;

        DarcyForchheimerCoeffs
        {
            d d [0 -2 0 0 0 0 0] (1e12 1e12 1e12);
            f f [0 -1 0 0 0 0 0] (1e12 1e12 1e12);

            coordinateSystem
            {
                e1 (1 0 0);
                e2 (0 1 0);
            }
        }
    }
}
```

The `pressureGradientExplicitSource` adds a momentum source to the momentum equation. It can be applied to specific areas or to the whole domain by specification in `selectionMode`. In this case it is adjusting the pressure gradient over the whole domain, to achieve the specified average flow velocity `Ubar`.

For this case we are interested in completely blocking the flow through the bed-cells. Therefore we use the implementation of the Darcy-Forchheimer equations, where the coefficients `d` and `f` should be specified with 3 components. A coordinate system is defined for each coefficient, `e1` and `e2`, which sets the local orientation of the components of the coefficients.

In the actual implementation we assign this to the `cellZone bed`, and we simple give very high values to the coefficients, such that the cells become practically impermeable.

## 3.6 Configuring simulation controls with controlDict

Open and edit `system/controlDict` to adjust the time steps such that a specified Courant number withheld. This is achieved by adding the following after `runTimeModifiable`:

```
adjustTimeStep  yes;

maxCo           0.5;
```

Since the time step will be adjusted during run-time, we have to make sure that `writeControl` is set to `adjustableRunTime`, so that the simulation will write out results at the time steps specified by `deltaT` and `writeInterval`.

Furthermore, we want get time-averaged quantities from the simulation, so we specify which fields have to be average during runtime. You can see that the velocity `U` and pressure `p` fields are already averaged under the `fieldAverage` function. Add the viscous and SGS stress fields to it, so your `controlDict` should look like this:

Content of controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pimpleFoam_mod;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1000;
```

```
deltaT          0.2;

writeControl    adjustableRunTime;

writeInterval   100;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep        yes;

maxCo                 0.5;

functions
{
    fieldAverage1
    {
        type            fieldAverage;
        functionObjectLibs ( "libfieldFunctionObjects.so" );
        enabled         true;
        outputControl   outputTime;
        resetOnRestart  true;

        fields
        (
            U
            {
                mean        on;
                prime2Mean  on;
                base        time;
                window      125;
            }

            p
            {
                mean        on;
                prime2Mean  on;
                base        time;
                window      125;
            }

            nuGradU
```

```
            {
                mean        on;
                prime2Mean  on;
                base        time;
                window      125;
            }

            B
            {
                mean        on;
                prime2Mean  on;
                base        time;
                window      125;
            }
        );
    }
}

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

The `window` option allows setting a moving average based the specification in `base`. The `resetOnRestart` option lets you determine whether averaging should be reset when restarting the simulation, or if it should continue averaging using the previous average.

Note that in some versions it has been reported that `fieldAverage` cannot average `prime2Mean` for `volSymmTensorField`. If that is the case, you can simply set it to `off` for `nuGradU` and `B`.

## 3.7 The fvSchemes and fvSolution dictionaries

The `fvSchemes` dictionary located in the `system` folder is where the finite volume schemes of each quantity are defined. For the current case there is no need to modify it.

The `fvSolution` dictionary is used to define the numerical solver of the different fields as well as the tolerances. There is a section dedicated for specific inputs for the PIMPLE solver, where you have to specify a reference point for the pressure. In this case rename `pRefCell` into `pRefPoint` and specify the coordinates of the point `(0 1 0);`. This means that the reference pressure at the top of channel is set to be 0.

Content of fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
```

```
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.05;
    }

    pFinal
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }

    "(U|k)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0.1;
    }

    "(U|k)Final"
    {
        $U;
        tolerance       1e-05;
        relTol          0;
    }
}

PIMPLE
{
    nOuterCorrectors 1;
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefPoint       (0 1 0);
    pRefValue       0;
}


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

## 3.8 Configuring the turbulence parameters

In order to configure which turbulence model should be used, if any, the `constant/turbulenceProperties` dictionary should be used. There you will find a single entry, `simulationType`, followed by the type. For this case we use LES, hence the entry is `LESModel`. Alternatives can be `RASModel` or `laminar`.

Having specified the turbulence model, it is necessary to have a dictionary which defines the properties of the chosen model. This is for the LES model done in the `constant/LESProperties` dictionary, where a variety of LES models can be selected, and their coefficients chosen.

In this case we use the default LES model which is the `oneEqEddy` model. It uses one $k$ equation to model the turbulence in the sub-grid scale. The spatial filter `delta` is chosen to be the cube root of each cell, with no van-Driest damping because the bed cells are not regarded as a wall and therefore the `vanDriest delta` would not work as it is intended to.

Make sure you edit the third line of the code in `constant/LESProperties` from `delta vanDriest` into `delta cubeRootVol`.

Content of LESProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      LESProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

LESModel        oneEqEddy;

printCoeffs     on;

delta           cubeRootVol;

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;
}


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

### 3.9 Configuring the flow parameters

The last thing to do before the simulation starts is to reduce the viscosity due to the reduction of the channel depth with the creation of the bed inside the domain. Open `constant/transportProperties` and reduce the viscosity of the fluid to say:

Viscosity specification in transportProperties

```
nu              nu [ 0 2 -1 0 0 0 0 ] 1.9e-05;
```

Another thing you would notice in `transportProperties` is the presence of `Ubar`, like in `fvOptions`. The value that is used for the average velocity is the one specified in `fvOptions`.

You can also see inputs for `CrossPowerLawCoeffs` and `BirdCarreauCoeffs`. These are used for non-Newtonian viscosity models, and since this case is Newtonian they are simply ignored.

### 3.10 Running the case

Your case should be fully set-up by now and ready for simulation. Run the case with your modified solver `pimpleFoam_mod`:

```
pimpleFoam_mod > log.run &
```

Verify that the case is running:

```
tail -f log.run
```

## 4 Post-processing

While the simulation is running, you can prepare the post-processing tools.

### 4.1 postChannel utility for spatial averaging

In the `constant` folder you will find the file `postChannelDict`. This is used for post-processing so the user can specify the direction of the depth and whether or not the domain is symmetric. In the `channel395` case the domain is symmetric thus the post-processing is generating of half the depth by averaging the top and bottom. In our case we have an open top and therefore the symmetry should be turned off:

Content of postChannelDict

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
```

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

// Seed patches to start layering from
patches         ( bottomWall );

// Direction in which the layers are
component       y;

// Is the mesh symmetric? If so average(symmetric fields) or
// subtract(asymmetric) contributions from both halves
symmetric       false;


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

Copy the `postChannel` utility into the utility directory in the user directory (if you don't have a utility folder in the user directory you can create it with `mkdir`!):

```
cp -r $FOAM_UTILITIES/postProcessing/miscellaneous/postChannel \
 $WM_PROJECT_USER_DIR/utilities
cd $WM_PROJECT_USER_DIR/utilities
```

Now rename the folder and files:

```
mv postChannel postChannelRough
cd postChannelRough
wclean
mv postChannel.C postChannelRough.C
mv channelIndex.C channelIndexRough.C
```

The `postChannel` utility does a spatial averaging of fields over 2 dimensions. Here it is configured to average the quantities over the streamwise and spanwise directions such that depth profiles are generated.

Recall that we used the `fieldAverage` function in the `controlDict` dictionary, thus we have some time averaged quantities. By default `postChannel` creates spatially averaged profiles from the mean velocity and pressure fields. In the following we will add averaging of our added viscous and SGS stress fields.

Open and edit `readFields.H` and add the following lines to the end of the file:

Additions to readFields.H

```
volSymmTensorField BMean
(
    IOobject
    (
        "BMean",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ
    ),
    mesh
);
volScalarField Bxx(BMean.component(symmTensor::XX));
volScalarField Byy(BMean.component(symmTensor::YY));
volScalarField Bzz(BMean.component(symmTensor::ZZ));
volScalarField Bxy(BMean.component(symmTensor::XY));

volSymmTensorField nuGradUMean
(
    IOobject
    (
        "nuGradUMean",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ
    ),
    mesh
);
volScalarField nuGradUxx(nuGradUMean.component(symmTensor::XX));
volScalarField nuGradUyy(nuGradUMean.component(symmTensor::YY));
volScalarField nuGradUzz(nuGradUMean.component(symmTensor::ZZ));
volScalarField nuGradUxy(nuGradUMean.component(symmTensor::XY));
```

Now we want to generate the profiles from the read fields. This is done by adding the following lines to `collapse.H`:

Additions to collapse.H

```
scalarField BxyValues(channelIndexing.collapse(Bxy));
scalarField nuGradUxyValues(channelIndexing.collapse(nuGradUxy));

makeGraph(y, BxyValues, "tauSgs", path, gFormat);
makeGraph(y, nuGradUxyValues, "tauVisc", path, gFormat);
```

Last thing we need to do is add the renamed entries to `Make/files`:

files

```
postChannelRough.C
channelIndexRough.C
```

```
EXE = $(FOAM_USER_APPBIN)/postChannelRough
```

Compile the utility with `wmake`.

Now you can navigate back to the `run` directory to check on the simulation progress of `channel395` and `roughChannel`. Take a peak at the `log` file of each case:

```
run
tail -f channel395/log.run
tail -f roughChannel/log.run
```

If the log file shows that the simulations have reached the last time step specified in the `controlDict`, you can run the post-processing utility on the cases.

For post processing the cases type:

```
postChannel -case channel395
postChannelRough -case roughChannel
```

Each case should now have a folder named `graphs` that includes averaged profiles of various quantities in the saved time steps.

## 4.2 Plotting with Gnuplot

The profiles can easily be visualized in gnuplot.

Notice that when you tailed the logs of the simulations, the pressure gradient in each iteration was listed. We will now run a command that will scan through the log file of `roughChannel` for the value of the pressure gradient, write it into a file `gradP.txt` and then calculate the mean pressure gradient over time:

```
cd roughChannel
cat log.run | grep 'pressure gradient' | cut -d' ' -f11 | tr -d ',' > gradP.txt
awk '{total += $1} END { print total/NR }' gradP.txt
```

You should now have output to the terminal the mean value of the pressure gradient, which you should insert in the following gnuplot script, in order to calculate the friction velocity:

```
gedit plotProfiles.gplt
```

plotProfiles.gplt

```
# analytical profile data
D2 = 1                  # depth channel395
offset = 0.0449899      # mean 0 bed of roughChannel found in boxes.txt
D1 = D2-offset          # mean depth roughChannel
```

```
kappa = 0.41           # von karman
d50 = 0.024            # mean grain diameter
Ks = 2.0*d50           # Nikuradse roughness
gradP = 0.000116894    # pressure gradient roughChannel
Uf1 = sqrt(gradP*D1)   # friction velocity roughChannel
Uf2 = 0.0079           # friction velocity channel395
nu1 = 2e-5             # viscosity roughChannel
nu2 = 2e-5             # viscosity channel395
Retau = Uf1*D1/nu1
KsP=Ks*Uf1/nu1

set terminal pngcairo
set pointsize 1.5

########################## Ux velocity plot ##########################
set output 'U.png'
set key left top
set xlabel "y^+"
set ylabel "U^+"
set logscale x

plot [yP=1:Retau] \
"graphs/1000/Uf.xy" u (($1-offset)*Uf1/nu1):($2/Uf1) t "roughChannel", \
"../channel395/graphs/1000/Uf.xy" u ($1*Uf2/nu2):($2/Uf2) t "channel395", \
1/kappa*log(30*yP/KsP) lw 2 t "1/kappa*ln\(30y^+/K_s^+)"

########################## shear stress plot ##########################
set output 'tau.png'
set key right bottom
set xlabel "y/D"
set ylabel "tau/Uf^2"
unset logscale x
set xrange [0:1]

plot \
"graphs/1000/uv.xy" u (($1-offset)/D1):($2/Uf1**2) t "uv", \
"graphs/1000/tauVisc.xy" u (($1-offset)/D1):($2/Uf1**2) t "tauvisc", \
"graphs/1000/tauSgs.xy" u (($1-offset)/D1):($2/Uf1**2) t "tauSgs", \
'< paste graphs/1000/uv.xy graphs/1000/tauVisc.xy graphs/1000/tauSgs.xy' \
u (($1-offset)/D1):(($2+$4+$6)/Uf1**2) t "tauTot"


system('display U.png &')
system('display tau.png &')
```

Generate plots with gnuplot:

```
gnuplot plotProfiles.gplt
```

You now have the files `U.png` and `tau.png`, which show the velocity profiles in `roughChannel` and `channel395`, and the shear stress in `roughChannel`.

Figures 3 and 4 show these profiles as obtained with the current set-up. It can be seen that the velocity has the parallel shift due to the roughness, exactly as it should be, and also confirmed by the log law for a rough bed. It should be noted that this is a very coarse simulation, hence a finer mesh should preferably be used for more accurate results, nevertheless the results are quite impressive for such a low computational cost. The shear stress profiles show how the Reynolds stresses dominate in the outer flow, while the SGS stress and viscous stress become gradually the dominating factors near the bed in the boundary layer. The total shear stress should ideally vary linearly and become unity at the bed. The unity seems to be obtained suggesting that the simulation has converged to a mean. The kinks in the total shear profile near the bed might indicate that the components haven't been computed to a satisfactory degree in that region, thus mesh refinement should be tested.

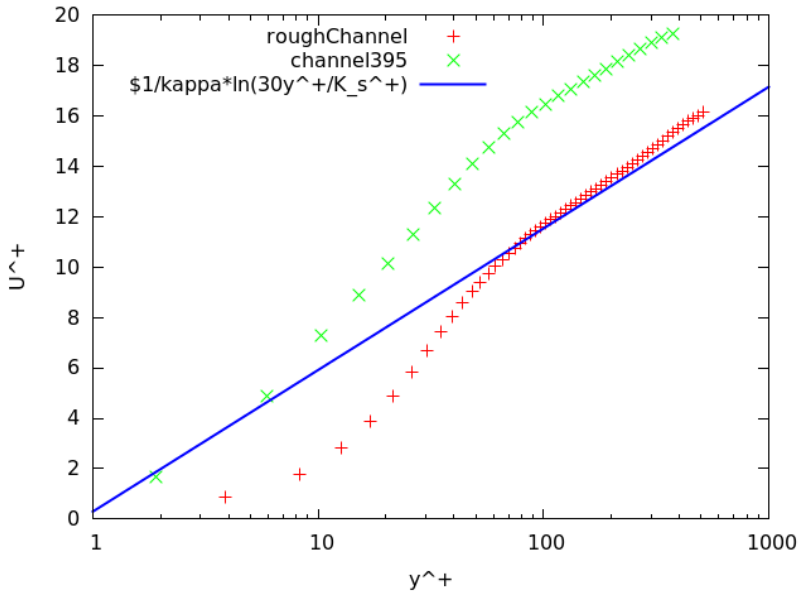


Figure 3: Velocity profiles

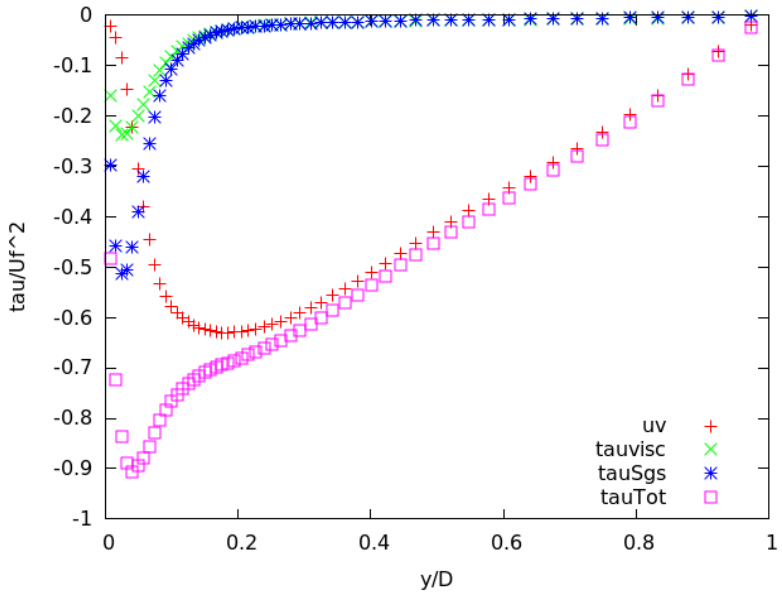


Figure 4: Shear stress distribution in roughChannel

# References

[1] Stoesser, T. (2010). "Physically Realistic Roughness Closure Scheme to Simulate Turbulent Channel Flow over Rough Beds within the Framework of LES." *J. Hydraul. Eng.*, 136(10), 812-819.

[2] http://www.openfoam.org/features/standard-solvers.php

[3] http://www.openfoam.org/version2.2.0/fvOptions.php