This course material has been developed at SCC (Steinbuch Centre for Computing at the Karlsruhe Institute of Technology). If you use it, please cite that the source is developed at SCC-Institute.

This course material is free: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation. It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

More details about the GNU General Public License can be seen at: <http://www.gnu.org/licenses/>.

Tutorial: BwUniCluster 2.0/HoreKa
Large Eddy Simulation (LES) in OpenFOAM

In this tutorial we will learn how to set up a LES case in OpenFOAM.

1. Inlet boundary condition

In Reynolds-Averaged Navier-Stokes (RANS) simulations, the effects of turbulence are modeled using turbulence models, which are based on empirical relationships between the mean flow properties and turbulence quantities. These models assume that the turbulence is statistically steady and homogeneous, which means that the turbulence structures do not vary significantly in space and time. As a result, generating turbulent structures at the inlet is not necessary in RANS simulations because the turbulence models are designed to simulate the averaged effects of turbulence on the mean flow. In Large Eddy Simulation (LES) or Direct Numerical Simulation (DNS) of fluid flows, it is important to accurately capture the turbulent structures present in the flow. In order to capture these turbulent structures, it is necessary to specify appropriate boundary conditions at the inlet of the computational domain. This is because turbulence is an unsteady and chaotic process, and the statistical properties of the turbulence vary in both space and time.

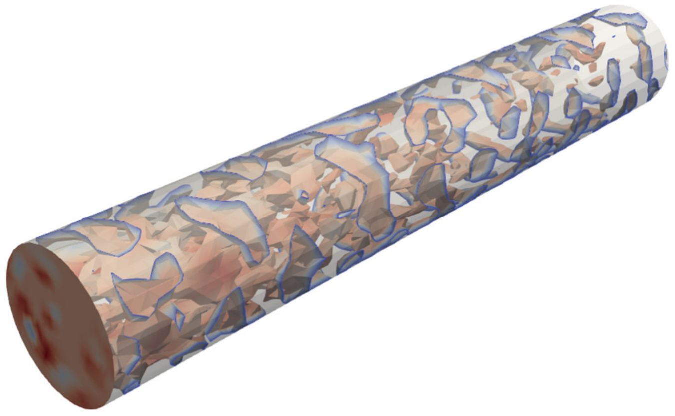
Possible approaches for generation of turbulent fluctuations in OpenFOAM:

a) Synthetic Turbulence Generation:

Divergence-Free Synthetic Eddy Method (turbulentDFSEMInlet) is a velocity boundary condition including synthesised eddies for use with DNS, LES and DES turbulent flows. It can be used as,

```
inlet
{
    type                turbulentDFSEMInlet;
    delta               1; //Characteristic length scale
    U                   uniform (0 0 1); //mean velocity
    R                   uniform (0.2 0 0 0.2 0 0.2); // Reynolds stress: <Rxx> <Rxy> <Rxz>
<Ryy> <Ryz> <Rzz>
    L                   uniform 0.4; //Integral length scale
    nCellPerEddy        1; //Minimum eddy length in units of number of cells
    value               uniform (0 0 1);
}
```

Results:



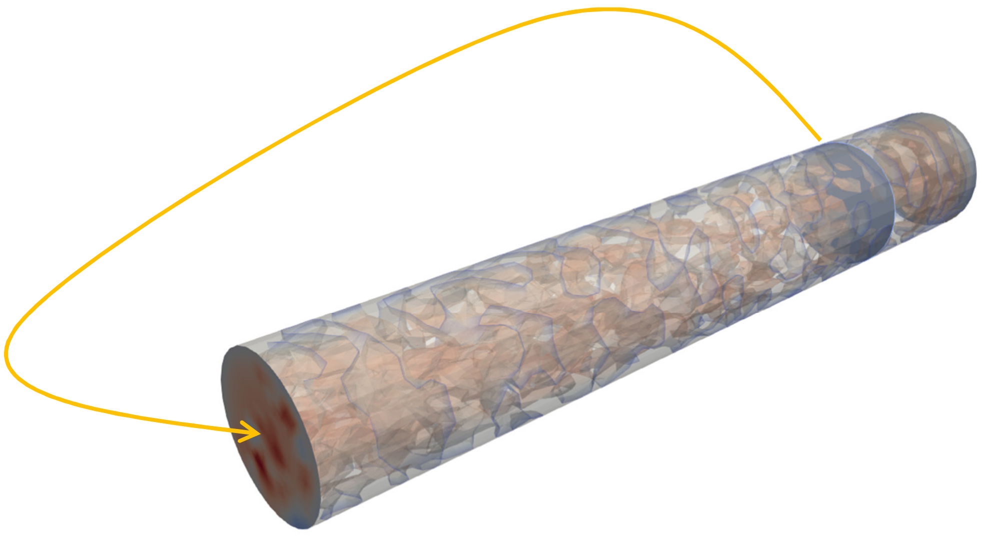
It is possible to use a field for U, R and L in turbulentDFSEMInlet. To do that first use codedFixedValue to generate the velocity field in the inlet and write the data just for a time step. Then this generated field can be pasted in turbulentDFSEMInlet boundary condition. Below is an example of codedFixedValue boundary condition:

```
inlet
{
    type            codedFixedValue;
    value           uniform (0 0 0);

    name    increaseToFixedValue;

    code
    #{
      scalar U_max = 2;
      const fvPatch& boundaryPatch = this->patch();
      const vectorField& Cf = boundaryPatch.Cf();
      vectorField& field = *this;

      forAll(boundaryPatch, i)
      {
           scalar r = sqrt(Cf[i].y()*Cf[i].y() + Cf[i].x()*Cf[i].x())/0.5;
           field[i] = vector(0, 0, U_max*Foam::pow(1.0-r, 1.0/7.0));
      }
    #};
}
```

One of the main problems of the turbulentDFSEMInlet is that it needs additional data for Reynolds stresses and integral length scale, which is not available in many cases.

b) **Recycling-Method (mapped boundary condition):**

Extension of domain upstream and extraction of turbulent velocities (and other fields if needed) from the interior domain as the picture below.



To use this method, the *boundary* file in the *polyMesh* directory should be modified in following way:

```
inlet
{
    type            mappedPatch; //modified
    nFaces          245;
    startFace       50225;
    sampleMode      nearestCell; //added
    samplePatch     none; //added
    sampleRegion    region0; //added
    offsetMode      uniform; //added
    offset          (0 0 5); //added
}
```

then the boundary condition is set for U and k as bellow,

U:

```
inlet
{
    type                mapped;
    value               uniform (1 0 0);
    interpolationScheme cell;
    setAverage          true;
    average             (1 0 0);
}
```

k:

```
  inlet
  {
    type             mapped;
    value            uniform 0.0;
    interpolationScheme cell;
    setAverage       false;
  }
```

One benefit of using this approach is that it does not require any parameters. However, it is important to note that the internal field needs to be agitated initially, as otherwise, it may take a significant amount of time for turbulent structures to form. Therefore, a possible solution is to utilize the *turbulentDFSEMInlet* method to generate vortices throughout the pipe (with a rough estimation of *R* and *L*) before switching to the mapped boundary condition (test it).

2. Numerical dissipation in LES

Numerical dissipation in Large Eddy Simulation (LES) refers to the artificial damping of the resolved turbulent scales due to the discretization of the governing equations on a numerical grid. Numerical dissipation arises from the truncation error in the numerical scheme used to solve the equations, and can lead to a loss of accuracy in the resolved scales.
In LES, the resolved turbulent scales are computed on a grid with finite resolution, which means that small-scale turbulent structures cannot be fully resolved and must be modeled or subgrid-scale (SGS) resolved. The numerical dissipation in the LES model can cause additional damping of the resolved scales, which can impact the accuracy of the subgrid-scale models.
In OpenFOAM, there are several discretization schemes available for the solution of the Navier-Stokes equations, each with different levels of numerical dissipation and accuracy. The choice of discretization scheme depends on the specific flow problem and the desired level of accuracy. However, central differencing schemes (Linear) are less diffusive than the upwind schemes, but they can introduce numerical oscillations in regions with strong gradients.

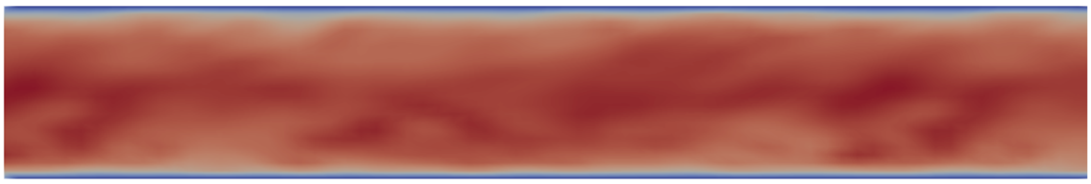
An example of a suitable discretization for LES is shown below,

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
```

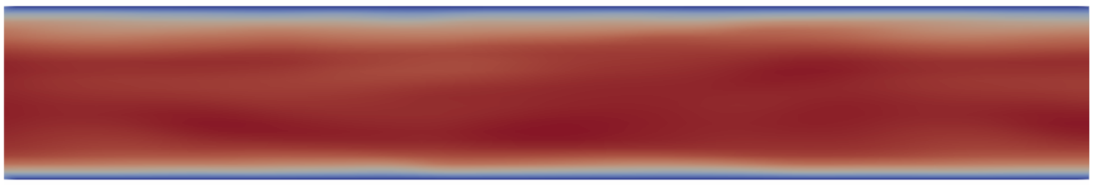
```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         backward;
}

gradSchemes
{
    default         leastSquares; //  "Gauss linear" is more stable
}

divSchemes
{
    default         none;

    div(phi,U)       Gauss linear; // use "LUST" For low quality grids
    div(phi,k)       Gauss linear; // use "imitedLinear" For low quality grids
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}
// ************************************************************************* //
```

Let us to examine the effect of discretization in LES. To do that apply the following changes in fvSchemes of testCase2:

div(phi,U)   Gauss linear;   →   div(phi,U)   Gauss linearUpwindV grad(U);



div(phi,U)   Gauss linear;



div(phi,U)   Gauss linearUpwindV grad(U);

1. **Post-processing**

The **"fieldAverage"** is a utility, that is used to compute time-averaged scalar and vector fields from the transient data generated by OpenFOAM solvers. It can also compute the root-mean-square (RMS) values of the fluctuating components of the fields. The time-averaged fields can be used for further analysis, such as computing turbulence statistics, or for validation against experimental data. To use this utility, the following code should be added in the *cotrolDict*,

```
functions
{
    myFieldAverage
    {
        type            fieldAverage;
        libs            (fieldFunctionObjects);
        writeControl    writeTime;

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

            p
            {
                mean        on;
                prime2Mean  on;
                base        time;
            }
        );
    }
}
```

The "**probes**" utility in OpenFOAM is a diagnostic tool used to extract information about the flow field at a particular point or location during the simulation. It can be used to monitor the evolution of various flow parameters such as velocity, pressure, temperature, and turbulence at a given point or a set of points in the computational domain. To use this utility, the following code should be added in the *cotrolDict*,

```
functions
{
    probes
    {
        type    probes;
        libs    (sampling);

        name    probes;

        writeControl   timeStep;
        writeInterval  1;

        fields
        (
            U
        );

        probeLocations
        (
            (0   0    5)
            (0.025  0    5)
            (0.05   0    5)
            (0.075  0    5)
            (0.1    0    5)
        );
    }
}
```

In OpenFOAM, the "**surfaces**" utility can be used to perform surface sampling of various flow parameters such as velocity, pressure, and temperature on a definded surfaces. To perform surface sampling using the surfaces utility, a user needs to first define the surface(s) of interest using a surface definition input. This definition specifies the location and geometry of the surface(s) in the computational domain. To use this utility, the following code should be added in the *cotrolDict*,

```
functions
{
    cuttingPlane
    {
        type            surfaces;
        libs            (sampling);

        writeControl    timeStep;
        writeInterval   5;

        surfaceFormat   vtk;
        fields          ( U );

        interpolationScheme cellPoint;

        surfaces
        {
            zNormal
            {
                type            cuttingPlane;
                planeType       pointAndNormal;
                pointAndNormalDict
                {
                    point    (0 0 0);
                    normal   (0 1 0);
                }
                interpolate     true;
            }
        }
    }
}
```

It is possible to utilize a Python script known as "vtkAnim.py" to create an animation of the output files that have been generated. The “vtkAnim.py” can be downloaded from the following link:
https://openfoamwiki.net/index.php/VtkAnim