Tutorial

J. Roller, C. Song

Large Eddy Simulation

modified on May 14, 2020

Version 1.3

http://www.hiflow3.org/

Contents

Using HiFlow^3 for simulating turbulent incompressible flows

1 Introduction

HiFlow^3 is a multi-purpose finite element software providing powerful tools for efficient and accurate solution of a wide range of problems modeled by partial differential equations (PDEs). Based on object-oriented concepts and the full capabilities of C++ the HiFlow^3 project follows a modular and generic approach for building efficient parallel numerical solvers. It provides highly capable modules dealing with the mesh setup, finite element spaces, degrees of freedom, linear algebra routines, numerical solvers, and output data for visualization. Parallelism - as the basis for high performance simulations on modern computing systems - is introduced on two levels: coarse-grained parallelism by means of distributed grids and distributed data structures, and fine-grained parallelism by means of platform-optimized linear algebra back-ends.

1.1 How to Use the Tutorial?

You find the example code (les_tutorial.cc, les_tutorial.h) and a parameter file for the first numerical example (les_tutorial.xml) in the folder `/hiflow/examples/les`. The geometry data (*.inp, *.vtu) is stored in the folder `/hiflow/examples/data`.

1.1.1 Using HiFlow^3 as a Developer

First build and compile HiFlow^3. Go to the directory `/build/examples/les`, where the binary **les_tutorial** is stored. Type **./les_tutorial**, to execute the program in sequential mode. To execute in parallel mode with four processes, type **mpirun -np 4 ./les_tutorial**. In both cases, you need to make sure that the default parameterfile les_tutorial.xml is stored in the same directory as the binary, and that the geometry data specified in the parameter file is stored in `/hiflow/examples/data`. Alternatively, you can specify the path of your own xml-file with the name of your xml-file (first) and the path of your geometry data (second) in the comment line, i.e. **./les_tutorial** `/"path_to_parameterfile"/"name_of_parameterfile".xml /"path_to_geometry_data"/`.

2 Mathematical Setup

2.1 Problem

We consider the simulation of an instationary flow in a two-dimensional cavity with the moving lid Γ_{in}. We assume the liquid to be an incompressible Newtonian fluid, and model the flow by the Navier-Stokes equations. We have to distinguish two different boundary conditions, namely, (1c) the inflow condition at Γ_{in} and (1d) the no-slip condition at all remaining parts of the boundary Γ_0. We consider the following initial-boundary value problem for the unknown velocity

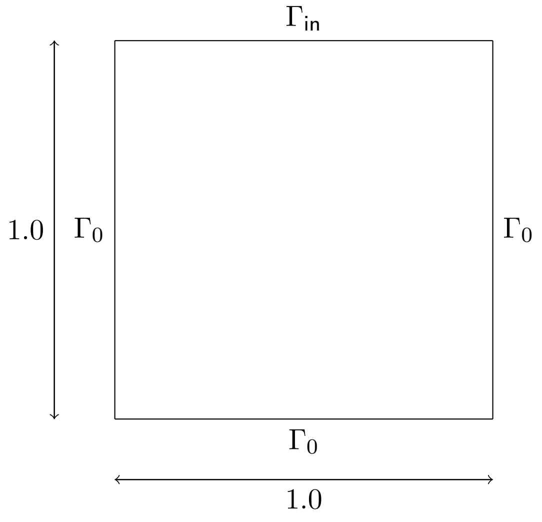

Figure 1: Unit square with moving lid Γ_{in}.

field $u = (u_1(x_1, x_2), u_2(x_1, x_2))^T$ and the unknown pressure $p = p(x_1, x_2)$ of the velocity field:

$$\begin{aligned}
\partial_t u + (u \cdot \nabla)u - \nu \Delta u + \frac{1}{\rho}\nabla p = 0, &\quad \text{in } \Omega, && \text{(1a)}\\
\nabla \cdot u = 0, &\quad \text{in } \Omega, && \text{(1b)}\\
u = g, &\quad \text{on } \Gamma_{\text{in}}, && \text{(1c)}\\
u = 0, &\quad \text{on } \partial\Omega \backslash \overline{\Gamma_{\text{in}}} =: \Gamma_0, && \text{(1d)}\\
u(0, \cdot) = u_0, &\quad \text{in } \Omega && \text{(1e)}
\end{aligned}$$

where Ω is the domain specified in Fig 1. The viscosity $\nu = \nu(x_1, x_2)$ is a known material constant and the density $\rho = \rho(x_1, x_2)$ is also constant due to the incompressibility of the fluid, $g : \Gamma_{\text{in}} \to \mathbb{R}$ is a given Dirichlet data.

2.2 Turbulence models

For the case of high Reynolds numbers , we have to take into account the occurrence of turbulent phenomena (eddies). The complexity of these can be characterized by their dependence on many different spatial scales. A finite element simulation of the Navier-Stokes equations can only simulate eddies up to the associated mesh size, which implies that an extremely fine mesh size is generally needed to directly simulate turbulent flows. However, on a fixed-size mesh, we can construct a splitting of the solution

$$u = \bar{u} + u', \quad p = \bar{p} + p', \tag{2}$$

such that $(\bar{u}, \bar{p})$ are regarded as the *large* or *resolved* scale parts and (u', p') are the *small* or *subgrid-scale* (SGS) parts of (u, p). The large eddy simulation (LES) approach then consists of applying a convolution with a *spatial filter* g to the Navier-Stokes equations with the aim of "filtering out" the small scales (which are then *modeled*, not simulated) and simulating the filtered equations . These are given by

$$\partial_t \bar{u} + \nabla \cdot (\overline{uu^T}) - \nu \Delta \bar{u} + \nabla \frac{1}{\rho} \bar{p} = \bar{f} \quad \text{in } (0,T] \times \mathbb{R}^d, \tag{3a}$$

$$\nabla \cdot \bar{u} = 0 \quad \text{in } (0,T] \times \mathbb{R}^d, \tag{3b}$$

$$\bar{u}(0,\cdot) = \bar{u}_0 \text{ in } \mathbb{R}^d. \tag{3c}$$

The advective term $\nabla \cdot (\overline{uu^T})$ in (3a) gives rise to the LES *closure problem* because we need to find a model that expresses it in terms of $\bar{u}$. Assuming linearity of the filter, it can be written as

$$\overline{uu^T} = \overline{\bar{u}\bar{u}^T} + \overline{\bar{u}u'^T} + \overline{u'\bar{u}^T} + \overline{u'u'^T}. \tag{4}$$

On the right-hand size the second and third term are the large-scale and *cross terms*, and the last term corresponding to SGS advection. The approach followed here assumes a Gaussian filter and consists of Fourier transforming each term, applying a rational approximation and then inverting the Fourier transform [1, p.541-551]. With a second-order approximation, we get for the large-scale and cross terms

$$\overline{\bar{u}\bar{u}^T} + \overline{\bar{u}u'^T} + \overline{u'\bar{u}^T} \approx \bar{u}\bar{u}^T + \big(I - \frac{\delta^2}{24}\Delta\big)^{-1} \frac{\delta^2}{12} \nabla\bar{u}\nabla\bar{u}^T, \tag{5}$$

where δ is the *characteristic filter width* which should refer to all eddies of size $\mathcal{O}(\delta)$ being filtered out. Commonly $\delta := 2h$ with the local mesh size h. We set

$$h := h_{K,\text{min}}, \tag{6}$$

where $h_{K,\text{min}}$ is the shortest edge on an individual mesh cell K. The second term on the right-hand side of (5) defines a tensor-valued elliptic auxiliary problem

$$-\frac{\delta^2}{24}\Delta\mathbb{X} + \mathbb{X} = \frac{\delta^2}{12}\nabla\bar{u}\nabla\bar{u}^T \tag{7}$$

which we equip with homogeneous Neumann boundary conditions. Equations (5),(7) then give the (second order) **rational LES model** . As the term $(I - \frac{\delta^2}{24}\Delta)^{-1}$ approximates convolution with a Gaussian filter, the model can alternatively be defined by

$$\overline{\bar{u}\bar{u}^T} + \overline{\bar{u}u'^T} + \overline{u'\bar{u}^T} \approx \bar{u}\bar{u}^T + \frac{\delta^2}{12} g * (\nabla\bar{u}\nabla\bar{u}^T). \tag{8}$$

A major drawback of this model is that it approximates the SGS term $\overline{u'\bar{u}'^T}$ as zero. Because completely neglecting this term is physically unreasonable, we need to add an SGS model which we call the *turbulent* or *eddy viscosity* $\nu_T(\bar{u})$. The most popular choice is the **Smagorinsky model** [2]

$$\nu_T(\bar{u}) = C_S \delta^2 ||\nabla\bar{u}||_F \tag{9}$$

with the *Smagorinsky constant* C_S which is not generally known. It has been estimated to be roughly 0.17 for isotropic turbulence. Another model derived by Iliescu and Layton (**IL**) in [3] is given by

$$\nu_T(\bar{u}) = C_S \delta ||\bar{u} - g * \bar{u}||_2. \tag{10}$$

This model can also be defined with a rational approximation of the convolution with the Gaussian filter, yielding

$$\nu_T(\bar{u}) = C_S\delta||\bar{u} - (I - \frac{\delta^2}{24}\Delta)^{-1}\bar{u}||_2 \tag{11}$$

and another auxiliary problem with right-hand side $\bar{u}$. It is one of three models proposed in [3], two of which have been implemented in this tutorial. In numerical experiments, the scaling factor C_S seems to need a larger value than for the Smagorinsky model. The likely reason is that the Smagorinsky model, unlike the Iliescu-Layton model, is often defined with a squared constant instead. Thus, it roughly holds that

$$C_{S,\text{IL}} \approx \sqrt{C_{S,\text{Smag.}}}. \tag{12}$$

Because we replace $\bar{u}$ with the previous time step velocity $\bar{u}_k$ in the auxiliary problems, these only have to be solved once in each time step. In the paper [4], the **streamline-upwind Petrov-Galerkin (SUPG)** method was compared to the Smagorinsky model. We can define a corresponding eddy viscosity by

$$\nu_{T,K}(\bar{u}) = \tau_{\text{SUPG}}||\bar{u}||^2_{\infty,K} \tag{13}$$

with the SUPG stabilization parameter τ_{SUPG} and the point-wise maximum of u in the Euclidean norm

$$||\bar{u}||_{\infty,K} = \max\left\{\sqrt{\bar{u}\cdot\bar{u}},\ x \in K\right\} \tag{14}$$

for any mesh cell K. This model, unlike all previous models, is based on the weak formulation, as it arises from the SUPG method. Vreman [5] introduced a sophisticated eddy viscosity model in 2004. It vanishes identically for over 300 types of non-turbulent flow structures, a considerable improvement over the Smagorinsky model, which is overly diffusive for laminar flows. In three spatial dimensions, Vreman's eddy viscosity is given by

$$\nu_T := \begin{cases} C_S\delta^2\sqrt{\frac{B}{|\nabla\bar{u}|_F^4}}, & |\nabla\bar{u}|_F^4 > 0 \\ 0, & \text{else.} \end{cases} \tag{15}$$

Using the shorthand β for the velocity gradient times its transpose

$$\beta_{i,j} := (\nabla\bar{u}\nabla\bar{u}^T)_{i,j}, \tag{16}$$

the scalar function $B(\beta)$ is defined as

$$B(\beta) := \beta_{1,1}\beta_{2,2} - \beta_{1,2}^2 + \beta_{1,1}\beta_{3,3} - \beta_{1,3}^2 + \beta_{2,2}\beta_{3,3} - \beta_{2,3}^2. \tag{17}$$

Vreman estimated the scaling factor C_S to be roughly

$$C_{S,\text{Vreman}} \approx 2.5C_{S,\text{Smagorinsky}}. \tag{18}$$

For isotropic turbulence, it is therefore $C_{S,\text{Vreman}} \approx 0.07$. Because this tutorial deals with two-dimensional problems only, we adapt Vreman's model by simply dropping all terms depending on the third dimension. As for the numerical approach, this model contains many more terms than the other eddy viscosity models discussed here. For efficiency, we thus treat it by explicit approximation as opposed to including it in the nonlinear solving process. Additionally, we set a very small cut-off value for the denominator $|\nabla\bar{u}|_F^4$ instead of requiring it to exactly vanish. The final, most complex eddy viscosity model in this tutorial is the *dynamic* Smagorinsky model. It has no free parameters because the scaling factor $C_S(x,t)$, which is not assumed to be constant anymore, is calculated from known quantities and can be determined at every timestep for every quadrature point x_q. The dynamic model introduces a second filter with width $\hat{\delta} = 2\delta$. Upon filtering the space-filtered Navier-Stokes equations (3a-3c) again with this filter and inserting the Smagorinsky model, the following approximation can be established after several steps of calculation:

$$\mathbf{0} \approx -\widehat{\bar{u}\bar{u}^T} + \hat{\bar{u}}\hat{\bar{u}}^T + \frac{1}{3}\text{trace}\Big(\widehat{\bar{u}\bar{u}^T} - \hat{\bar{u}}\hat{\bar{u}}^T\Big)\mathbb{I} \tag{19}$$

$$+C_S(t,x)\left(\delta^2\Big(\widehat{||\mathbb{D}(\bar{u})||_F\mathbb{D}(\bar{u})}\Big) - \hat{\delta}^2||\mathbb{D}(\hat{\bar{u}})||_F\mathbb{D}(\hat{\bar{u}})\right)$$

$$=: \mathbb{L} + C_S(x,t)\mathbb{M}, \tag{20}$$

where the hat symbol $(\hat{\ })$ denotes quantities filtered by the second filter and $\mathbb{I}$ the identity tensor. A solution $C_S(x,t)$ is obtained by replacing the approximation (19) with an equality. The corresponding system of equations is overdetermined; no value of C_S satisfies it exactly. Lilly [6] first suggested approximating a solution by the least squares approach. It requires minimization of $||\mathbb{L} + C_S(x,t)\mathbb{M}||_F^2$. Setting the first derivative of this expression with regard to C_S to zero gives

$$C_S(x,t) = -\frac{\mathbb{L} : \mathbb{M}}{\mathbb{M} : \mathbb{M}}(x,t). \tag{21}$$

An important property of this choice of C_S is that – unlike the standard Smagorinsky constant – it can become negative, allowing for backscatter of energy. However, it can rapidly vary in space and time. We thus apply a cell-wise averaging of the numerator and the denominator of (21). Additionally, we clip the total viscosity $\nu + \nu_T$ to zero if it becomes negative (by setting $\nu_T = -\nu$). The reason is both physical (negative viscosity is thermodynamically impossible) and practical because allowing negative viscosity strongly destabilizes numerical simulations. The resulting models can be summarized by the equations

$$\partial_t\bar{u} - \nabla\cdot((\nu + \nu_T(\bar{u}))\nabla\bar{u}) + (\bar{u}\cdot\nabla)\bar{u}$$

$$+\frac{1}{\rho}\nabla\bar{p} + \nabla\cdot\Big(A_{\text{LES}}\big(\frac{\delta^2}{12}\nabla\bar{u}_k\nabla\bar{u}_k^T\big)\Big) = \bar{f} \quad \text{in } (0,T]\times\Omega, \tag{22a}$$

$$\nabla\cdot\bar{u} = 0 \quad \text{in } (0,T]\times\Omega, \tag{22b}$$

$$\bar{u}(0,\cdot) = \bar{u}_0 \text{ in } \Omega. \tag{22c}$$

On a bounded domain Ω, $\bar{u}$ is subject to problem-dependent boundary conditions. The operator A_{LES} can be defined as

- $A_{LES} := 0$ for a pure SGS model, as is the standard Smagorinsky model,
- $A_{LES} := I$ for the *Taylor LES* model (also called the *gradient model*), defined by a Taylor series approximation of the convolution with the Gaussian filter. As that approximation is very inaccurate, we advise against using this model.
- $A_{LES} := \left(I - \frac{\delta^2}{24}\Delta\right)^{-1}$ for the rational LES model with auxiliary problem (5, 7),
- $A_{LES} := g*$ for the rational LES model with convolution (8). Especially in three dimensions, it is impractically expensive to compute.

The turbulent viscosity ν_T can be chosen from the SGS models described above, i.e. Equations 9 10 11 13 21.

Remark 2.1. *Not all choices of ν_T produce a stable model when combined with the rational LES model. For the Smagorinsky and Iliescu-Layton models, C_S should be chosen smaller than for a pure SGS model because otherwise, these dominate the rational LES terms, making its contribution irrelevant.*

2.3 Weak Formulation and linearization

To solve a problem using finite element methods, a variational formulation of the problem must be given. It can be derived by multiplying the equation with some test functions, integrating over the domain, applying integration by parts and the Gauss theorem. Therefore the domain Ω has to be a Lipschitz domain [7, p.89-96]. The finite element spaces we use are the Taylor-Hood pairs Q_2/Q_1 (on quadrilateral and hexahedral meshes) and P_2/P_1 (on simplicial meshes). Because these spaces are only discretely divergence-free, the weak divergence-free property is not guaranteed. Therefore, we penalize divergence by adding the additional term

$$-\mu\nabla(\nabla\cdot u) \tag{23}$$

to the momentum equation, where the estimate $\mu \approx 0.25$ has been reported in the literature [8]. This is called *grad-div stabilization*. As we are *not* solving the Navier-Stokes equations, but some version of the turbulence model (22a)-(22c), we will denote the solution from now on as (w, r) as to not confuse it with the solution (u, p) of the Navier-Stokes equations. The solution space for the approximated velocity w is defined as

$$\mathcal{U}(\Omega) := \{u \in (H^1(\Omega))^2 : \; u|_{\Gamma_{\text{in}}} = g, \; u|_{\Gamma_0} = 0\}.$$

The space of test and trial functions has homogeneous Dirichlet boundaries. Therefore we have to find $\bar{u} = u + u_d$, where $u_d \in \mathcal{U}(\Omega)$ and

$$w \in V(\Omega) := \{u \in (H^1(\Omega))^2 : \; u|_{\Gamma_0 \cup \Gamma_{\text{in}}} = 0\}.$$

These spaces are exactly the same as in the standard Galerkin formulation of the Navier-Stokes equations. To discretize the instationary LES boundary value problem, we need the weak formulation of the linearized problem which is then solved in each Newton step. For this we need to linearize the additional nonlinear terms included in the models. In the case of the Smagorinsky model the respective term $G(w) = \nu_T(w)\nabla w$ can be linearized as

$$\begin{aligned}\nabla_w G(w)\cdot c_w &= C_S\delta^2\frac{\nabla w : \nabla c_w}{||\nabla w||_F}\nabla w + \nu_T(w)\nabla c_w \\ &=: \bar{\nu}_T(w, c_w)\nabla w + \nu_T(w)\nabla c_w.\end{aligned} \tag{24}$$

For the Iliescu-Layton model with convolution $\nu_T(w) = C_S\delta||w - g*w||_2$, we use the approximation

$$C_S\delta||w - g*w||_2 \approx C_S\delta||w - g*w_k||_2, \tag{25}$$

where w_k is taken from the last time step. The same approximation can be applied when the convolution operator is itself approximated as $g* \approx \left(I - \frac{\delta^2}{24}\Delta\right)^{-1}$. We then proceed analogously to arrive at

$$\begin{aligned}\nabla_w G(w)\cdot c_w &= C_S\delta\frac{(w - g*w_k)\cdot c_w}{||w - g*w_k||_2}\nabla w + \nu_T(w)\nabla c_w \\ &=: \bar{\nu}_T(w, c_w)\nabla w + \nu_T(w)\nabla c_w\end{aligned} \tag{26}$$

The nonlinear gradient term is, as previously mentioned, treated explicitly:

$$\nabla w \nabla w^T \approx \nabla w_k \nabla w_k^T \tag{27}$$

After multiplying with test functions and integrating by parts, we arrive at the problem:

Problem 2.1. *Find $c_w \in \mathcal{U}(\Omega)$ and $c_r \in \mathcal{L}^2(\Omega)$ such that*

$$\begin{aligned}
&\int_\Omega c_w \circ \phi + \alpha_1 \Big[(\nu + \nu_T(\bar{w}))\nabla c_w : \nabla \phi \\
&\quad + \bar{\nu}_T(\bar{w}, c_w)\nabla \bar{w} : \nabla \phi + (\bar{w} \cdot \nabla) c_w \circ \phi \\
&\quad + (c_w \cdot \nabla)\bar{w} \circ \phi + \mu(\nabla \circ w)(\nabla \circ \phi) \Big] - \alpha_2 \frac{c_r}{\rho}(\nabla \circ \phi) && = \int_\Omega F_1(\bar{w}, \bar{r}) \circ \phi \mathrm{dx} && (28)
\end{aligned}$$

$$\int_\Omega (\nabla \cdot c_w)\psi \mathrm{dx} = \int_\Omega F_2(\bar{w}, \bar{r})\psi \mathrm{dx} \qquad (29)$$

for all $\phi \in V(\Omega), \psi \in \mathcal{L}^2(\Omega)$, where $(\overline{w}, \overline{r})^T$ denotes the linearization point or previous Newton step. The right-hand side is given by

$$\begin{aligned}
\int_\Omega F_1(\overline{w}, \overline{r}) \circ \phi \mathrm{dx} = &\int_\Omega (\overline{w} - w_k) \circ \phi && (30)\\
&+ [\alpha_1(\nu + \nu_T(\bar{w}))\nabla \overline{w} + \alpha_3(2\nu + \nu_T(w_k))\nabla w_k] : \nabla \phi \\
&+ [\alpha_1(\overline{w} \cdot \nabla)\overline{w} + \alpha_3(\overline{w}_k \cdot \nabla)\overline{w}_k] \circ \phi \\
&- \alpha_2 \Big[\frac{\overline{r}}{\rho}(\nabla \circ \phi) + \Big(A_{\mathrm{LES}}\big(\frac{\delta^2}{12}\nabla \bar{w}_k \nabla \bar{w}_k^T\big)\Big)\Big] : \nabla \phi \mathrm{dx} && (31)\\
\int_\Omega F_2(\overline{w}, \overline{r})\psi \mathrm{dx} = &\int_\Omega (\nabla \cdot \overline{w})\psi \mathrm{dx} && (32)
\end{aligned}$$

The parameters $\alpha_1, \alpha_2, \alpha_3$ define the Crank-Nicolson discretization in time:

$$\begin{aligned}
\alpha_1 &= \frac{1}{2}\Delta t \\
\alpha_2 &= \Delta t \qquad (33)\\
\alpha_3 &= \frac{1}{2}\Delta t.
\end{aligned}$$

The advantage of this formulation is that the auxiliary problems only have to be solved once in each time step. The scalar versions of their weak formulations are given by

$$\int_\Omega \nabla \mathbb{X}_{i,j} \cdot \nabla \mathbb{Y}_{i,j} + \int_\Omega \mathbb{X}_{i,j}\mathbb{Y}_{i,j} = \int_\Omega \frac{\delta^2}{12}(\nabla w_k \nabla w_k^T)_{i,j}\mathbb{Y}_{i,j} \qquad (34)$$

for the rational LES model and

$$\int_\Omega \nabla \mathbb{X}_i \cdot \nabla \mathbb{Y}_i + \int_\Omega \mathbb{X}_i \mathbb{Y}_i = \int_\Omega (w_k)_i \mathbb{Y}_i \qquad (35)$$

for the first Iliescu-Layton model. The space of test and trial functions for the auxiliary problems $V^{\mathsf{aux}}(\Omega)$ is the same as $V(\Omega)$ up to boundary conditions [9]. Using the finite element method for the discretization, the linear variational formulation results in a stiffness matrix, which corresponds to the Jacobian in Newton's method. During the discretization process the stiffness matrix and residual vector have to be assembled.

3 The Commented Program

3.1 Preliminaries

HiFlow^3 is designed for high performance computing on massively parallel machines. So it applies the Message Passing Interface (MPI) library specification for message-passing. The LES tutorial needs the following two input files:

- A parameter file: The parameter file is an xml-file, which contains all parameters needed to execute the program. It is read in by the program. Parameters for example defining the termination condition of the non-linear and linear solver are listed as well as parameters needed for the linear algebra. It is not necessary to recompile the program, when parameters in the xml-file are changed. By default the flow tutorial reads in the parameter file les_tutorial.xml, see section 3.2, which contains the parameters of the two-dimensional numerical example, see section 5.1.This file is stored in `/hiflow/examples/les/`.

- Geometry data: The file containing the geometry is specified in the parameter file. For the numerical results in two dimensions in section 5.1 we choose cavity_les.inp. You can find different meshes in the folder
`/hiflow/examples/data` .

HiFlow^3 does not generate meshes for the domain Ω. Meshes in *.inp and *.vtu format can be read in. There exists a function in /build/utils/ called 'inp2vtu' which converts *.inp format to *.vtu format. Type **/build/utils/inp2vtu 2 cavity_les.inp** to convert cavity_les.inp to cavity_les.vtu. Additionally a file cavity_les_bdy.vtu is created which shows the body of the domain.
It is possible to extend the reader for other formats. Furthermore it is possible to generate other geometries by using external programs (Mesh generators) or by hand. Both formats provide the possibility to mark cell or facets by material numbers.

3.2 Parameter File

The needed parameters are initialized in the parameter file les_tutorial.xml. The scaling factor C_S is set by the parameter `TurbulenceParameter`. One can choose the eddy viscosity model from `Smagorinsky`, `IliescuLayton1`, `IliescuLayton2` or `SUPG` and the LES model between `Taylor` and `RationalAux`. Please note that the Taylor LES model is generally unstable and the rational LES model without an eddy viscosity model also eventually becomes unstable in simulations. C_S should also not be smaller than $\mathcal{O}(0.001)$ to have a meaningful impact of the SGS model. It is possible to leave out the LES model entirely and only specify an eddy viscosity model.

```
'
<Param>
    <OutputPrefix>LESTutorial</OutputPrefix>
    <OutputPath>""</OutputPath>
    <Mesh>
        <Filename>cavity_les.inp</Filename>
        <InitialRefLevel>2</InitialRefLevel>
    </Mesh>
    <UseBoundaryDomainDescriptor>0</UseBoundaryDomainDescriptor>
    <FlowModel>
        <Density>1.0</Density>
        <Viscosity>1.e-4</Viscosity>
        <InflowSpeed>1.0</InflowSpeed>
        <InflowHeight>0.41</InflowHeight>
        <InflowWidth>0.41</InflowWidth>
    </FlowModel>
```

```
<DFGbenchmark>0</DFGbenchmark>
<BenchQuantities>0</BenchQuantities>
<QuadratureOrder>6</QuadratureOrder>
<FiniteElements>
    <VelocityDegree>2</VelocityDegree>
    <PressureDegree>1</PressureDegree>
</FiniteElements>
<Instationary>
    <SolveInstationary>1</SolveInstationary>
    <Method>CrankNicolson</Method>
    <Timestep>0.005</Timestep>
    <Endtime>20.0</Endtime>
    <TurbulenceParameter>0.005</TurbulenceParameter>
    <EddyViscosityModel>DynamicSmagorinsky</EddyViscosityModel>
    <LESModel>None</LESModel>
</Instationary>
<Boundary>
    <InflowMaterial>15</InflowMaterial>
    <OutflowMaterial>16</OutflowMaterial>
    <CylinderMaterial>14</CylinderMaterial>
</Boundary>
<NonlinearSolver>
    <UseHiFlowNewton>1</UseHiFlowNewton>
    <MaximumIterations>20</MaximumIterations>
    <AbsoluteTolerance>1.e-15</AbsoluteTolerance>
    <RelativeTolerance>1.e-6</RelativeTolerance>
    <DivergenceLimit>1.e6</DivergenceLimit>
    <ArmijoUpdate>0</ArmijoUpdate>
    <ThetaInitial>1.0</ThetaInitial>
    <ThetaMinimal>1.e-16</ThetaMinimal>
    <ArmijoDecrease>0.5</ArmijoDecrease>
    <SufficientDecrease>1.e-2</SufficientDecrease>
    <MaxArmijoIteration>5</MaxArmijoIteration>
    <ForcingStrategy>EisenstatWalker1</ForcingStrategy>
    <ConstantForcingTerm>1.e-4</ConstantForcingTerm>
    <InitialValueForcingTerm>1.e-3</InitialValueForcingTerm>
    <MaxValueForcingTerm>1.e-2</MaxValueForcingTerm>
    <GammaParameterEW2>0.9</GammaParameterEW2>
    <AlphaParameterEW2>2</AlphaParameterEW2>
</NonlinearSolver>
<LinearSolver>
    <MaximumIterations>1000</MaximumIterations>
    <AbsoluteTolerance>1.e-15</AbsoluteTolerance>
    <RelativeTolerance>1.e-6</RelativeTolerance>
    <DivergenceLimit>1.e6</DivergenceLimit>
    <BasisSize>500</BasisSize>
    <Preconditioning>1</Preconditioning>
</LinearSolver>
<Schur>
    <NumIter>50</NumIter>
    <RelativeTolerance>1.e-1</RelativeTolerance>
    <PrintLevel>0</PrintLevel>
    <SolverBlockA>
        <NumIter>1000</NumIter>
        <BasisSize>100</BasisSize>
        <RelativeTolerance>5.e-3</RelativeTolerance>
    </SolverBlockA>
    <PrecondBlockA>
        <MaxLevels>25</MaxLevels>
        <MaxCoarseSize>9</MaxCoarseSize>
        <CycleType>1</CycleType>
        <InterpType>6</InterpType>
```

```
            <CoarsenType>10</CoarsenType>
            <RelaxType>3</RelaxType>
            <RelaxWt>0.5</RelaxWt>
            <StrongThreshold>0.6</StrongThreshold>
            <NumSweeps>5</NumSweeps>
            <AggNumLevels>25</AggNumLevels>
            <AggInterpType>4</AggInterpType>
            <Nodal>0</Nodal>
            <SmoothNumLevels>0</SmoothNumLevels>
            <Variant>3</Variant>
            <Overlap>1</Overlap>
            <DomainType>1</DomainType>
            <SchwarzUseNonSymm>1</SchwarzUseNonSymm>
        </PrecondBlockA>
        <SolverPrecond>
            <NumIter>500</NumIter>
            <RelativeTolerance>1.e-1</RelativeTolerance>
        </SolverPrecond>
        <PrecondPrecond>
            <MaxLevels>25</MaxLevels>
            <MaxCoarseSize>9</MaxCoarseSize>
            <CycleType>1</CycleType>
            <RelaxType>6</RelaxType>
            <RelaxWt>0.5</RelaxWt>
            <NumFunctions>1</NumFunctions>
            <InterpType>6</InterpType>
            <CoarsenType>10</CoarsenType>
            <StrongThreshold>0.6</StrongThreshold>
            <NumSweeps>3</NumSweeps>
            <AggNumLevels>25</AggNumLevels>
            <AggInterpType>4</AggInterpType>
            <SmoothNumLevels>0</SmoothNumLevels>
            <Variant>3</Variant>
            <Overlap>1</Overlap>
        </PrecondPrecond>
    </Schur>
    <UsePressureFilter>0</UsePressureFilter>
    <Backup>
        <Restore>0</Restore>
        <LastTimeStep>0</LastTimeStep>
        <Filename>backup.h5</Filename>
    </Backup>
</Param>
```

3.3 Member functions

The main function starts the simulation of the LES problem (les_tutorial.cc). Most other functions are taken from the HiFlow^3 Navier-Stokes tutorial; they will not be discussed here. The boundary conditions are defined in the class ChannelFlowBC (les_tutorial.h). Note that since the LES auxiliary problems have homogeneous Neumann boundary conditions, these do not have to be explicitly imposed on the weak solution.

3.3.1 solve()/solve_eddy_viscosity_auxiliary()/solve_rational_les_auxiliary()

The member functions solve(), solve_eddy_viscosity_auxiliary() and solve_rational_les_auxiliary() read in some parameters and solve the non-linear flow problem and the LES auxiliary problems at the current time step. These are implemented in les_tutorial.cc. The instationary matrix is of course only assembled after setting the parameter C_S of the SGS model and potentially including the solutions of the auxiliary problems. For the first Iliescu-Layton SGS model (11), we filter each velocity component separately and thus define our local assembly (for one version of the model) as follows:

```
class IliescuLayton1Assembler : private AssemblyAssistant< DIMENSION, double > {
public:
  IliescuLayton1Assembler(const CVector &time_sol) : prev_time_sol_(time_sol) {}

  void set_index(const int var) {
    var_ = var;
  }

  void operator()(const Element< double > &element,
                  const Quadrature< double > &quadrature, LocalMatrix &lm) {
    AssemblyAssistant< DIMENSION, double >::initialize_for_element(element,
        quadrature);

    compute_delta(element);
    // compute local matrix
    const int num_q = num_quadrature_points();
    for(int q = 0; q < num_q; ++q) {
      const double wq = w(q);
      const double dJ = std::abs(detJ(q));
      for(int i = 0; i < num_dofs(0); ++i) {
        for(int j = 0; j < num_dofs(0); ++j) {
          lm(dof_index(i, 0), dof_index(j, 0)) +=
            wq * dot((0.25*delta_*delta_/6.)*grad_phi(j, q), grad_phi(i, q)) * dJ;
        }
      }

      for(int var = 0; var < DIMENSION+1; ++var) {
        const int n_dofs = num_dofs(var);
        for(int i = 0; i < n_dofs; ++i) {
          for(int j = 0; j < n_dofs; ++j) {
            lm(dof_index(i, var), dof_index(j, var)) +=
              wq * phi(j, q, var) * phi(i, q, var) * dJ;
          }
        }
      }
    } // end loop q
  }

  void operator()(const Element< double > &element,
                  const Quadrature< double > &quadrature, LocalVector &lv) {
    AssemblyAssistant< DIMENSION, double >::initialize_for_element(element,
        quadrature);
    // recompute previous timestep velocity for rhs
    for(int d = 0; d < DIMENSION; ++d) {
      vel_ts_[d].clear();
      evaluate_fe_function(prev_time_sol_, d, vel_ts_[d]);
    }

    const int num_q = num_quadrature_points();
    for(int q = 0; q < num_q; ++q) {
      const double wq = w(q);
      const double dJ = std::abs(detJ(q));

      for(int i = 0; i < num_dofs(0); ++i) {
        lv[dof_index(i, 0)] += wq * vel_ts_[var_][q] * phi(i, q) * dJ;
      }
    }
  }
private:

  void compute_delta(const Element< double > &element) {
    const mesh::Entity cell = element.get_cell();
    const mesh::TDim edge_dim = 1;

    delta_ = h();
    std::vector<double> edge_coords;
    for(mesh::IncidentEntityIterator it = cell.begin_incident(edge_dim);
        it != cell.end_incident(edge_dim); ++it) {
      it->get_coordinates(edge_coords);
      delta_ = std::min(delta_,sqrt(std::pow(edge_coords[0]-edge_coords[2],
                                       2)+std::pow(edge_coords[1]
                                       -edge_coords[3],2)));
    }
    delta_ *= 2.; // 2 * shortest edge of cell
  }

  int var_; // velocity component for which the scalar Helmholtz equation is solved
  const CVector &prev_time_sol_;
  double delta_;
  FunctionValues < double > vel_ts_[DIMENSION]; // velocity at previous timestep
};
```

As the rational LES auxiliary problem is actually tensor-valued, we would in theory have to solve d^2 scalar Helmholtz problems. Fortunately, the tensor $\nabla w \nabla w^T$ is symmetric. Therefore, only $\frac{d(d+1)}{2}$ auxiliary equations have to be solved. The class of the local assembler for the rational LES term is then

```
,
class RationalLESAssembler : private AssemblyAssistant< DIMENSION, double > {
public:
  RationalLESAssembler(const CVector &time_sol) : prev_time_sol_(time_sol) {}

  void set_indices(int var_i, int var_j) {
    var_i_ = var_i;
    var_j_ = var_j;
  }

  void operator()(const Element< double > &element,
                  const Quadrature< double > &quadrature, LocalMatrix &lm) {
    AssemblyAssistant< DIMENSION, double >::initialize_for_element(element,
        quadrature);

    compute_delta(element);
    // compute local matrix
    const int num_q = num_quadrature_points();
    for(int q = 0; q < num_q; ++q) {
      const double wq = w(q);
      const double dJ = std::abs(detJ(q));
      for(int i = 0; i < num_dofs(0); ++i) {
        for(int j = 0; j < num_dofs(0); ++j) {
          lm(dof_index(i, 0), dof_index(j, 0)) +=
            wq * dot((0.25*delta_*delta_/6.)*grad_phi(j, q), grad_phi(i, q)) * dJ;
        }
      }

      for(int var = 0; var < DIMENSION+1; ++var) {
        const int n_dofs = num_dofs(var);
        for(int i = 0; i < n_dofs; ++i) {
          for(int j = 0; j < n_dofs; ++j) {
              lm(dof_index(i, var), dof_index(j, var)) +=
                wq * phi(j, q, var) * phi(i, q, var) * dJ;
            }
          }
        }
      } // end loop q
    }

    void operator()(const Element< double > &element,
                    const Quadrature< double > &quadrature, LocalVector &lv) {
      AssemblyAssistant< DIMENSION, double >::initialize_for_element(element,
          quadrature);
      // recompute previous timestep velocity for rhs
      for(int d = 0; d < DIMENSION; ++d) {
        grad_vel_ts_[d].clear();
        evaluate_fe_function_gradients(prev_time_sol_, d, grad_vel_ts_[d]);
      }

      compute_delta(element);
      const int num_q = num_quadrature_points();

      for(int q = 0; q < num_q; ++q) {
        const double wq = w(q);
        const double dJ = std::abs(detJ(q));
        // Get previous timestep velocity gradient times its transpose
        for(int i = 0; i < num_dofs(0); ++i) {
          lv[dof_index(i, 0)] += wq * (0.5*delta_*delta_/6.) * dot(
                                 grad_vel_ts_[var_i_][q],grad_vel_ts_[var_j_][q])
                                 * phi(i, q) * dJ;
        }
      }
    }
  private:

    void compute_delta(const Element< double > &element) {
      const mesh::Entity cell = element.get_cell();
      const mesh::TDim edge_dim = 1;

      delta_ = h();
      std::vector<double> edge_coords;
      for(mesh::IncidentEntityIterator it = cell.begin_incident(edge_dim);
          it != cell.end_incident(edge_dim); ++it) {
        it->get_coordinates(edge_coords);
        delta_ = std::min(delta_,sqrt(std::pow(edge_coords[0]-edge_coords[2],
                                       2)+std::pow(edge_coords[1]
                                       -edge_coords[3],2)));
      }
      delta_ *= 2.; // 2 * shortest edge of cell
    }

    int var_i_;
    int var_j_; // component of velocity gradient times transpose
                // for which the scalar Helmholtz equation is solved
    const CVector &prev_time_sol_;
    double delta_;
    FunctionValues < Vec < DIMENSION, double > >
    grad_vel_ts_[DIMENSION]; // gradient of velocity at previous timestep

  };
```

4 Program Output

HiFlow^3 can be executed in a parallel or sequential mode which influence the generated output data. Note that the log files can be viewed by any editor.

4.1 Parallel Mode

Executing the program in parallel, for example with four processes by **mpirun -np 4 ./les_tutorial** generates following output data.

- Mesh/geometry data:
  - **mesh_local.pvtu** Global mesh (parallel vtk-format). It combines the local meshes of sequential vtk-format owned by the different processes to the global mesh.
  - **mesh_local_X.vtu** local mesh owned by process X for X=0, 1, 2 and 3 (vtk-format).
- Solution data. Since it is only possible to visualize data of polynomial degree 1 (Q_1 on quads in 2 dimensions), only the data corresponding to the degrees of freedom of a Q_1 element are written out. It means the information of the degrees of freedom of higher order is lost due to the fact that it cannot be visualized using the vtk-format.
  - **LESTutorial_solution_.pvtu** Solution of the velocity field and the pressure variable (parallel vtk-format). It combines the local solutions owned by the different processes to a global solution.
  - **LESTutorial_solution_X.vtu** Local solution of the velocity field and the pressure variable of the degrees of freedoms which belong to cells owned by process X, for X=0, 1, 2 and 3 (vtk-format).
- Log files:
  - **LESTutorial_debug_log** Log file listing errors helping to simplify the debugging process. This file is empty if the program runs without errors.
  - **LESTutorial_info_log** Log file listing parameters and some helpful informations to control the program as for example information about the residual of the linear and non-linear solver used.

4.2 Sequential Mode

Executing the program sequentially by **./les_tutorial** following output data is generated.

- Mesh/geometry data:
  - **mesh_local.pvtu** Global mesh (parallel vtk-format).
  - **mesh_local_0.vtu** Global mesh owned by process 0 (vtk-format) containing the mesh information.
- Solution data. Since it is only possible to visualize data of polynomial degree 1 (Q1 on quads in 2 dimensions), only the data corresponding to the degrees of freedom of a Q1-element are written out. It means the information of the degrees of freedom of higher order are lost due to the fact that this information cannot be visualized using the vtk-format.
  - **LESTutorial_solution.vtu** Solution of the velocity field and the pressure variable (vtk-format).

- Log files:
  - **LESTutorial_debug_log** is a list of errors helping to simplify the debugging process. This file keeps empty if the program runs without errors.
  - **LESTutorial_info_log** is a list of parameters and some helpful informations to control the program as for example information about the residual of the linear and non-linear solver used.

4.3 Visualizing the Solution

HiFlow^3 only generates output data but does not visualize. The mesh/geometry data as well as the solution data can be visualized by any external program which can handle the vtk data format as e.g. the program ParaView [10].

5 Numerical Example

5.1 Driven cavity

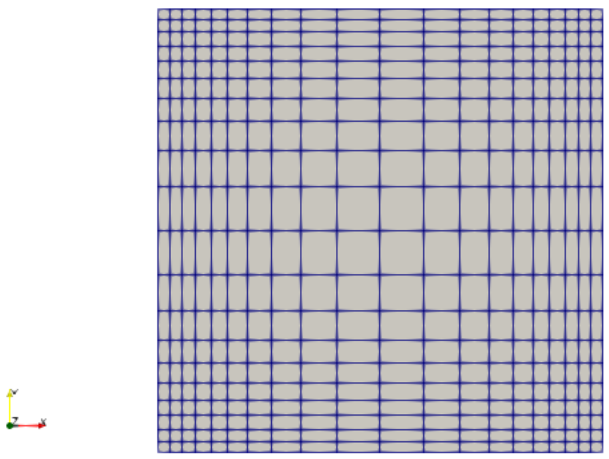

Figure 2: The mesh (cavity_les.inp) used for the simulation is finer toward the walls. For our example simulation, it is refined twice.

We set $U_m = 1.0ms^{-1}$ and $H = 1.0m$ in the parameter file. As geometry data we choose the square mesh with refinement towards the walls given by cavity_les.inp. The inflow condition $g(x, 1)$ at the top (inflow boundary Γ_{in}) is a regularized version of the well-known lid-driven cavity problem (see [11] for details) and defined as

$$g(x,1) = \begin{pmatrix} g_1(x) \\ 0 \end{pmatrix}, \tag{36}$$

$$g_1(x) = \begin{cases} 1 - \frac{1}{4}\big(1 - \cos\big(\frac{x_1 - x}{x_1}\pi\big)\big)^2 & \text{for} x \in [0, x_1], \\ 1 & \text{for} x \in (x_1, 1 - x_1), \\ 1 - \frac{1}{4}\big(1 - \cos\big(\frac{1-(1-x_1)}{x_1}\pi\big)\big)^2 & \text{for} x \in [1 - x_1, 1] \end{cases} \tag{37}$$

with $x_1 = 0.1$. All other walls have no-slip / homogeneous Dirichlet boundary conditions. Because these boundary conditions are not implemented by material numbers, you do not have to change the material numbers in the parameter file. As one example, fig. 2-4 show the mesh, the solution on refinement level 2 for this example illustrated by the magnitude of the velocity vector and the pressure for the parameters Re $= 10^5$ and $C_S = 0.1$ with the first Iliescu-Layton model.

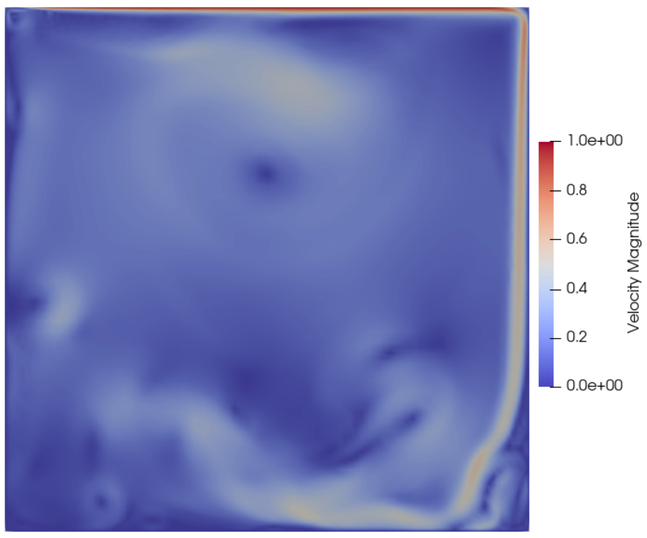

Figure 3: The magnitude of the velocity vector field $w_h \in U_h(\Omega)$ of the simulation.

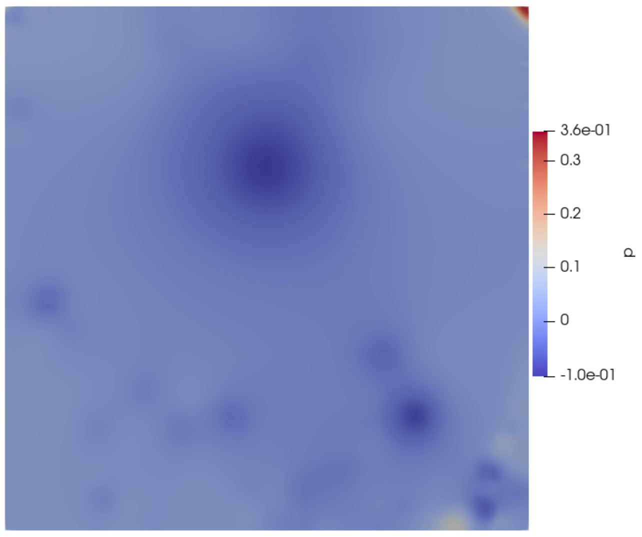

Figure 4: The solution for the pressure.

References

[1] Volker John. *Finite Element Methods for Incompressible Flow Problems*. Springer International Publishing, Cham, 2016.

[2] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963.

[3] T Iliescu and WJ Layton. Approximating the larger eddies in fluid motion. iii. the boussinesq model for turbulent fluctuations. *Analele Stiintifice ale Universitatii Al. I. Cuza, tomul*, 44:245–261, 1998.

[4] J. E. Akin, Tayfun E. Tezduyar, M. Ungor, and S. Mittal. Stabilization parameters and smagorinsky turbulence model. *Journal of Applied Mechanics, Transactions ASME*, 70(1):2–9, 1 2003.

[5] A. W. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications. *Physics of Fluids*, 16(10):3670–3681, 2004.

[6] D. K. Lilly. A proposed modification of the germano subgrid-scale closure method. *Physics of Fluids A: Fluid Dynamics*, 4(3):633–635, 1992.

[7] W.McLean. *Strongly Elliptic Systems and Boundary Integral Equations*. Cambridge University Press, 2000.

[8] Javier de Frutos, Bosco García-Archilla, Volker John, and Julia Novo. Grad-div stabilization for the evolutionary oseen problem with inf-sup stable finite elements. *Journal of Scientific Computing*, 66(3):991–1024, Mar 2016.

[9] Volker John. *Large Eddy Simulation of Turbulent Incompressible Flows. Analytical and Numerical Results for a Class of LES Models*, volume 34. 01 2004.

[10] Amy Henderson Squillacote. *The ParaView guide: a parallel visualization application*. Kitware, 2007.

[11] Javier de Frutos, Volker John, and Julia Novo. Projection methods for incompressible flow problems with weno finite difference schemes. *Journal of Computational Physics*, 309:368 – 386, 2016.