Training:
Relaxation finite element schemes for the incompressible Navier-Stokes equations

Ruslan Krenzler
Training Coordinator: Dr. Chiara Simeoni

16. July. 2009

Abstract

The target of this training is to understand the role of the relaxation inside the numerical process. In particular it focuses on analysis and numerical simulation of relaxation Navier-Stokes for incompressible fluids using finite element methods in two space dimensions.

Contents

1 Introduction **3**
- 1.1 Relaxation finite element schemes for the incompressible Navier-Stokes equation. . . . 4
- 1.2 Methodology . . . 4
- 1.3 Implementation . . . 5
- 1.4 Notations . . . 5

2 Stokes equations for incompressible fluids[3] **6**
- 2.1 Variational Formulation . . . 7
- 2.2 Existence and Uniqueness of the Solution and Meaning of Inf-Sup condition . . . 7
- 2.3 Numerical Simulations . . . 8
- 2.4 Postprocessing of the Results of Numerical Simulations . . . 9

3 Steady State Navier-Stokes equations[1] **12**
- 3.1 Variational Formulation Oseen problem . . . 13
- 3.2 Numerical Simulations . . . 13

4 Navier-Stokes equations **15**
- 4.1 Variational Formulation . . . 15
- 4.2 Numerical Simulations . . . 16

5 Navier-Stokes equations with relaxation. **16**
- 5.1 Variational Formulation . . . 17
- 5.2 Numerical Simulations . . . 17

6 Tests of FreeFem++ Solvers **18**

7 Software **20**
- 7.1 LyX. . . . 21
- 7.2 *FreeFem++* . . . 21
- 7.3 *FreeFem++-cs* . . . 21
- 7.4 Subversion . . . 22
- 7.5 Noweb . . . 22
- 7.6 Eclipse . . . 23
- 7.7 Inkscape . . . 23
- 7.8 TextText . . . 24
- 7.9 Gnuplot . . . 24
- 7.10 R . . . 24
- 7.11 Python . . . 24

A Nonlinear, bilinear and linear functions, long forms and identities **25**

B General Remarks to Simulations **27**

C Taylor-Green-Vortex **28**

D Useful macros **34**
D.1 Norm macros 34
D.2 Timer macros 35
D.3 Data storage macros 37

E Numerical Simulations of Stokes-Equations **38**

F Numerical simulation of the Oseen problem **45**

G Numerical simulation Taylor-Green-Vortex as Navier-Stokes equations **53**

H Numerical simulation Taylor-Green-Vortex as Navier-Stokes equations with relaxation. **59**

I Custom Libraries: **64**

1 Introduction

The Navier-Stokes equations describe dynamical behavior of incompressible fluid in space and time. This behavior is modeled as a system of partial differential equations

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = f(x,t) \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \end{cases} \tag{1}$$

- d is the dimension of the space, in this training we consider $d = 2$
- $\mathbb{R}^d \supset \Omega$ is the space, where the fluid is situated.
- $\mathbb{R} \supset [0,T]$ is time period of observation.
- $u : \Omega \times [0,T] \longrightarrow \mathbb{R}^d$, $(x,t) \mapsto u(x,t)$ is the velocity of the fluid in point x at time t
- $p : \Omega \times [0,T] \longrightarrow \mathbb{R}$, $(x,t) \mapsto p(x,t)$ is the pressure of the fluid in point x at time t
- $g(x,t) : \partial\Omega \times [0,T] \longrightarrow \mathbb{R}^d$ is the boundary condition, which describes the behavior of the velocity u on the boundary $\partial\Omega$.[1]
- $f(x,t) : \Omega \times [0,T] \longrightarrow \mathbb{R}^d$ describes body forces of the fluid.
- ν is viscosity of the fluid.

[1] We focus on the Dirichlet boundary conditions, other conditions are also possible.

- The equation $\mathrm{div} u = 0$ describes the incompressibility of the fluid.
- Ω, T, f, g, ν are given, the functions u and p need to be calculated.

1.1 Relaxation finite element schemes for the incompressible Navier-Stokes equation.

Following [4] we extend the incompressible Navier-Stoke equations (2) by adding a relaxation approximation $\varepsilon\partial_{tt}u + \varepsilon\partial_t\nabla p$ as following

$$\begin{cases} \partial_t u + u\cdot\nabla u - \nu\Delta u + \nabla p & +\varepsilon\partial_{tt}u + \varepsilon\partial_t\nabla p = f(x,t) \\ \mathrm{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \end{cases} \tag{2}$$

We will focus on low viscosity values and we will analyze the behavior of the numerical simulation depending on the parameters ν , ε and the choice of finite element space. Since the incompressibility of the fluid, that means the expression $\mathrm{div} u = 0$, could be the source of instability of numerical solution, we will try to apply special finite element spaces described in the paper [5], to see how they influence the numerics.

1.2 Methodology

The idea of the training is to collect general ideas from various sources and try to implement them. To learn to work with various sources, to do it quickly and focus only on essential parts is an important part of the scientific work.

The studying sequence was firstly to read articles [4, 5], in order to understand, what are the final goals of the training and what is required to achieve these goals. Then it was necessary to go back to simple mathematical models until the point, where knowledge received in the MathMods master courses could be applied. After the choice of starting point we can go step by step to the final goal, increasing the complexity of the mathematical models, introducing new numerical methods and improving software.

The numerical implementation is an essential part of the training. To join mathematics with numerical implementation we will often use ideas from Literate Programing paradigm. This paradigm was introduced by Donald Knuth as a way of producing program documentation. The difference to the common programing process is following: instead of just to add comments into the source code we will put source code in the documentation. Actually the largest part of the report, which is put into appendix, is documentation to the various numerical implementations in FreeFem++. More information about Literate Programming could be found in `http://en.wikipedia.org/wiki/Literate_programming`.

1.3 Implementation

For the numerical simulation, tests and visualization we will work only with free and open source software, and open standards. This software is of height quality and additionally it provides us with the following advantages

- Working with worldwide communities.
- Height flexibility, due to possibility to extend or to combine already existing solutions.
- More independence from the original software developers.
- Working on different platforms: Unix, OSX, Windows.
- Avoiding of "black boxes".
- No additional license costs for private, scientific and commercial use.
- Freely available documentation.

The list of software with a short description and short information about the license will be given in the section 7. For FEM calculation we will use FreeFem++.

During the project I used 4 programming languages:

- FreeFem++, for numerical calculations.
- Python, for automatization of simulation tests with different parameters and postprocessing (visualization and format conversion).
- C++, to write a special library for data interchange between FreeFem++ and external applications.
- R, for postprocessing (analyzing and visualization) of experiment series.

1.4 Notations

For our calculations we will use Hilbert spaces $L_2(\Omega)$ and $L(\Omega)^2$ spaces with

$$L_2(\Omega) = \{f : \Omega \to \mathbb{R}| \qquad ||f||_{L^2(\Omega)} < \infty\}$$

with appropriate scalar product and norms

$$\begin{aligned} < u, v >_{L^2(\Omega)} &= \int_\Omega uvdx \\ ||u||_{L^2(\Omega)} &= \sqrt{< u, u >_{L^2(\Omega)}} \end{aligned}$$

For the vector fields on Ω we will use

$$L_2(\Omega)^2 = \{f = (f_1, f_2) : \Omega \to \mathbb{R}^{\not\subset}| \qquad ||f||_{L^2(\Omega)^2} < \infty\}$$

subspaces with appropriate scalar products and norms defined for $f := (f_1, f_2), \quad g := (g_1, g_2)$ as

$$\begin{aligned} < f, g >_{L^2(\Omega)^2} &= \int_\Omega h_1 g_1 d\lambda + \int_\Omega f_2 g_2 d\lambda \\ ||f||_{L^2(\Omega)^2} &= \sqrt{< f, f >_{L^2(\Omega)^2}} = \sqrt{< f_1, f_1 >_{L^2(\Omega)} + < f_2 f_2 >_{L^2(\Omega)}} \end{aligned}$$

In the most cases we will write just $< u, v >$ for $< u, v >_{L^2(\Omega)}$ or $< u, v >_{L^2(\Omega)^2}$

For the numerical analysis we mainly use Sobolev spaces

$$H^m(\Omega) = \{f \in L^2(\Omega) : D^\alpha f \in L^2(\Omega) \forall |\alpha| \leq k\}$$

with the seminorms

$$|f|_{H^m(\Omega)} = \left(\sum_{|k|=m} ||\partial^k f||_{L^2(\Omega)} \right)^{1/2}$$

and with the norm

$$||f||_{H^m} = \left(\sum_{0 \leq |k| \leq m} |f|^2_{H^m(\Omega)} \right)^{1/2}$$

We will use the names (w, q) for the *Taylor-Green Vortex* to wich we will refer very often. The Taylor-Green Vortex as an analytical solution of Navier-Stokes equations. The more precise information about it can be found in the appendix section C

2 Stokes equations for incompressible fluids[2][3]

As already mentioned in section 1.2, Methodology , we will start with simpler equations. In this section we will introduce many important ideas, which we will often use later. We start with Stokes equations

$$\begin{cases} -\nu \Delta \tilde{u} + \nabla p & = \tilde{f}(x,t) \\ \text{div} \tilde{u} & = 0 \\ \tilde{u}|_{\partial\Omega} & = g(x,t) \end{cases} \tag{3}$$

The Stokes equations model the behavior of a fluid with very height viscosity in steady state.

[2]The theoretical part of this chapter is mainly took from the 12. chapter of the book [3]

2.1 Variational Formulation

To solve the Stokes equations (4) means to find function u and p. These functions are from the different functional spaces. The function space for velocity we will call

$$X = H^1(\Omega)^2$$

and the functional space for pressure we will call

$$M = \{q \in L^2(\Omega) : \int_\Omega q dx = 0\}$$

The additional constraint $\int_\Omega q dx = 0$ is necessary for the uniqueness of the solution p.

For the FEM approximation, we will use finite subpsaces $X_h \subset X$ and $M_h \subset M$.

The first step is to bring the system of equations to the form

$$\begin{cases} -\nu\Delta u + \nabla p & = f(x,t) \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = 0 \end{cases} \tag{4}$$

To get variational formulation of the problem, we multiply the first equations with a test function $\phi \in X$ and constraint equation $\text{div} u = 0$ with a test function $\psi \in M$ and integrate over Ω. The result is the variational formulation of the problem:

$$\begin{cases} \int_\Omega -\nu\Delta u\phi dx + \int_\Omega \phi\nabla p dx & = \int_\Omega f u\phi dx \\ \int_\Omega u\psi dx & = 0 \end{cases}$$

$$\iff \begin{cases} a(u,\phi) + b(\phi,p) & = F(\phi) \qquad \forall\phi \in X \\ b(u,\psi) & = 0 \qquad \forall\psi \in M \end{cases} \tag{5}$$

with

$$a(v,\phi) = \nu < \nabla v, \nabla\phi >, \qquad b(\phi,\psi) = - < \text{div}\phi, \psi >$$

Both bilinear forms are continuous, that means

$$\begin{aligned} a(v,\phi) &< C||v||_X||\phi||_X, \qquad \forall v,\phi \in X \\ b(\phi,\psi) &< C||\phi||_X||\psi||_M, \qquad \forall\phi \in X, \forall\psi \in M \end{aligned}$$

2.2 Existence and Uniqueness of the Solution and Meaning of Inf-Sup condition

In order to solve the equations above, we defined a subspace Z of X s.t.

$$Z = \{\phi \quad \in \quad X | b(\phi,\psi) = 0 \quad \forall\psi \in M\}$$

To find solution u in the subspace Z of X, means to solve an equation

$$a(u,\phi) = F(\phi) \qquad \forall \phi \in Z$$

Such a solution exists and it is unique, due to coercivity and continuity of the bilinear form a on the Hilbert space Z, according to the Lax-Milgram theorem.

The next step is to find the pressure p. From the variational formulation it follows

$$b(\phi,p) = F(\phi) - a(u,\phi) \qquad \forall \phi \in X \tag{6}$$

The u is taken from the previous step. We cannot use Lax-Milgram theorem here strait forward, but another, related to the coercivity, concept . To show this relationship, we take a coercive bilinear form $c(u,v)$ on some a Hilbert space H, scalar product $<>_H$ and appropriate norm $||\cdot||_H$ and we transform it

$$\begin{aligned} c(u,v) &\geq \alpha||u||_H||v||_H \quad \forall u,v \in H \\ \Longrightarrow \frac{c(u,v)}{||v||_H} &\geq \alpha||u||_H \quad \forall u,v \in H \\ \Longrightarrow \sup_{v\in H} \frac{c(u,v)}{||v||_H} &\geq \alpha||u||_H \quad \forall u \end{aligned}$$

Fortunately it is enough for to have existence and uniqueness of the solution for the equation (6) a less weak condition

$$\Longrightarrow \sup_{v\in X} \frac{b(\phi,\psi)}{||\phi||_H} \geq \alpha||\psi||_M \quad \forall v \in X$$

with continuity of a and b.

2.3 Numerical Simulations

We will perform some numerical simulation, in order to find out, how different parameters, like viscosity ν, mesh refinement N and choice of the finite dimensional Spaces $(X_h, M_h) \subset (X, M)$ influence the solution. For analysis of quality of the simulation we will need some exact solution as a reference. For this purpose we will use *Taylor-Green Vortex* function described in the section C. The Taylor-Green Vortex is an exact solution of the Navier-Stokes equations. To use it as Stokes equations it needs to be transformed. This method we will use frequently.

Let (w,q) be a Taylor-Green Vortex solution, that is

$$\begin{cases} \partial_t w + w\cdot\nabla w - \nu\Delta w + \nabla q &= 0 \qquad \text{on } \Omega = (0,\pi)^2 \times [0,\infty) \\ \text{div} w &= 0 \\ w|_{\partial\Omega} &= w|_{\partial\Omega} \end{cases}$$

For the fixed time $t = 0$ it follows

$$\begin{cases} -\nu\Delta w + \nabla q & = (\partial_t w - w \cdot \nabla w)|_{t=0} \\ \mathrm{div} w & = 0 \\ w|_{\partial\Omega} & = w|_{\partial\Omega, t=0} \end{cases}$$

We define $f(x)$ and $g(x)$ as

$$f(x) : \quad = \quad (\partial_t w(x,0) - w \cdot \nabla w)|_{t=0}$$

and

$$g(x) := w(x,0)$$

Thus $u(x) = w(x,0)$, $p = q(x,0)$ is the exact solution of

$$\begin{cases} -\nu\Delta u + \nabla p & = f \qquad \text{on } \Omega \\ \mathrm{div} u & = 0 \\ u|_{\partial\Omega} & = g \qquad \text{on } \partial\Omega \end{cases}$$

2.4 Postprocessing of the Results of Numerical Simulations

A series of the Stokes was solving by an FreeFem++ script described in Appendix E with different values of ν and different mesh refinement N. Mesh refinement means that for the rectangle domain $[0, \pi]^2$ we use $N \times N$ grid. This process was automatized by using of a python script *stokes.py*.

ν	10	1.0	1/10	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	0

N	10	15	20	25	30	40

This simulations where done with Taylor-Hood elements $(X_h, M_h) = (P_2(\Omega)^2, P_1(\Omega))$ (continues, partially quadratic, and partially linear functions). As a solver was selected default FreeFem++ solver UMFPACK. The choice of he UMFPACK was done after a series of tests, where it proved to be robust and fast. (See the test results in the section 6). The simulations results where collected in an .csv file and later analyzed with an R-script *stokes.r*. In order see relation between different parameters we used standard plot function for data frames.

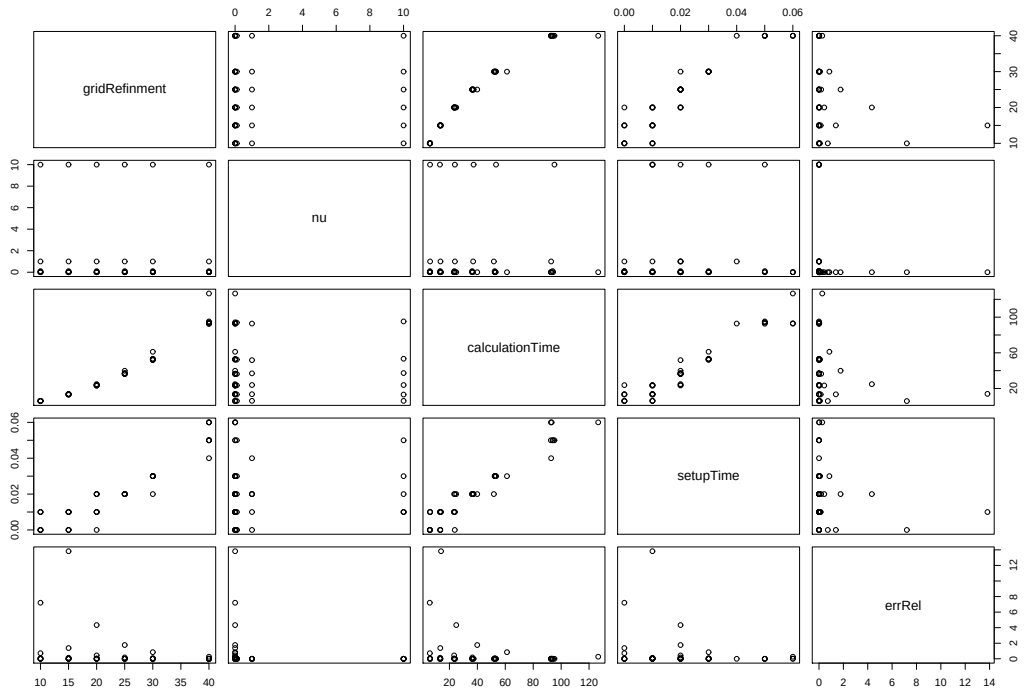

Figure 1: Parameter relations

It is important to notice the influence of viscosity ν on the calculation precision and calculation time. Mainly it means the low viscosity the less precise is the calculation. The solutions with $\nu = 0$ failed. Beginning with viscosity greater than 1 the precision decreases again.

We need take into account, that the relationship between viscosity and precision also depends on many technical factors, like choice of the solver, implementation of the linear solver, solving methods, problem definition in FreeFem++, break condition and so on. But mention of this relation has at least two reasons.

- There are analytically estimators for the convergence speed with the same relationship.
- We need to know numerical properties of tools we working with.

The next plots demonstrate the influence of grid refinement on precision and calculation time.

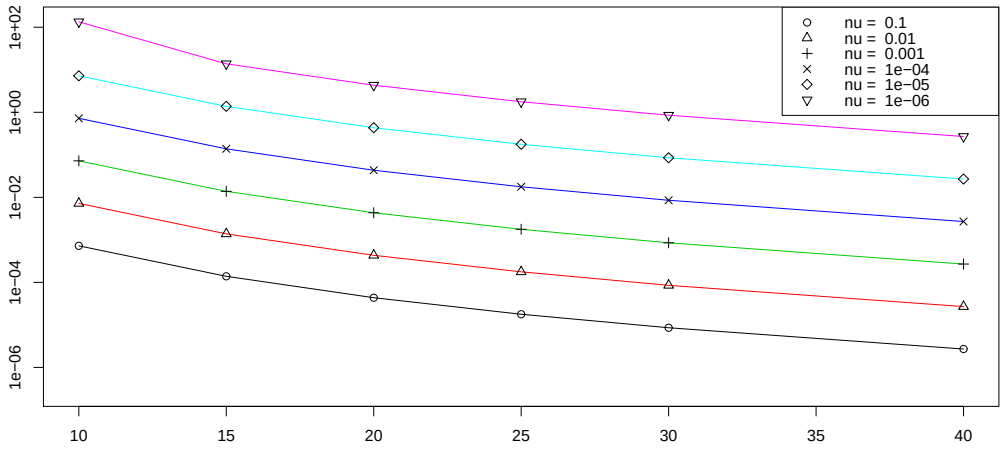

Figure 2: grid refinement and relative error

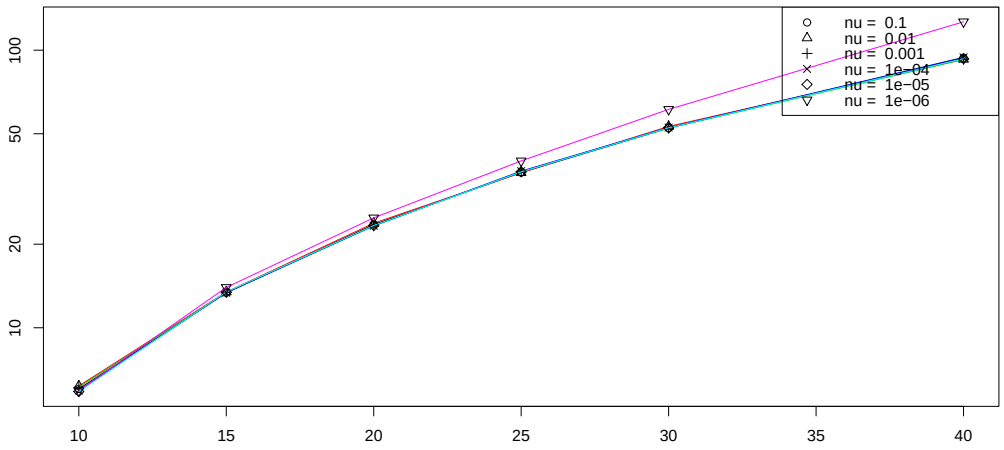

Figure 3: grid refinement and calculation time

3 Steady State Navier-Stokes equations[3][1]

If we take Navier-Stokes equations, and make an assumption it is in steady state, the expression u_t disappears:

$$\begin{cases} u \cdot \nabla u - \nu \Delta u + \nabla p & = f(x,t) \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \end{cases} \tag{7}$$

This equation system describes fluid properties in steady state.

Comparing to the Stokes equations we get an additional non-linear expression $u \times \nabla u$, which makes the solving more complicated.

[3]The mathematical part of this chapter is based on the 2.1 chapter of the book [1]

3.1 Variational Formulation Oseen problem

Solving of steady state Navier-Stokes equation with FEM methods requires the linearisation of the $u \cdot \nabla u$ expression. One popular method is based on Picard iteration method. We choose some starting point u^0, then in each step we interactively solve so called Oseen problem

$$\begin{cases} u^n \cdot \nabla u^{n+1} - \nu \Delta u^{n+1} + \nabla p^{n+1} & = f(x,t) \\ \mathrm{div} u^{n+1} & = 0 \\ u^{n+1}|_{\partial\Omega} & = g(x,t) \end{cases} \tag{8}$$

For uniqueness of the pressure p we add some additional condition

$$\int_\Omega p dx = 0$$

Under certain condition: ν is not too small, f is not too large the steady state Navier-Stokes equation 7 has a unique solution $(\hat{u}, \hat{p})$ to which (u^k, p^k) converges with $k \to \infty$

To get the variational formulation of the problem, we use the same approach like in Stokes equations. Additionally we add a new expression $< u^k \nabla u, \phi >$ to it.

$$\begin{cases} < u^k \cdot \nabla u^{k+1}, \phi > + \nu < \nabla u^{k+1}, \nabla \phi > - < p^{k+1}, \mathrm{div}\phi > & \\ - \int_{\partial\Omega} \frac{\partial}{\partial n} u \phi ds - + \int_{\partial\Omega} (\phi \cdot n) p ds & =< f, \phi > \quad \text{on } \Omega \\ < \mathrm{div} u^{k+1}, \psi > & = 0 \\ u^{k+1}|_{\partial\Omega} & = g(x,t) \text{ on } \partial\Omega \end{cases} \tag{9}$$

3.2 Numerical Simulations

As an numerical test we choose a $\Omega = (0,1) \times (0,1)$, $f = 0$, and a boundary condition $u_1 = 0.1$ on $\Gamma_2 \cup \Gamma_4$,

and $u_2 = 0$ on $\partial\Omega$, $\nu = 1$

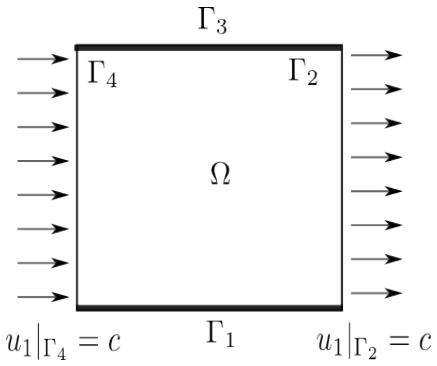

Figure 4: Simulation of a steady state Navier-Stoke equations, $c = 0.1$

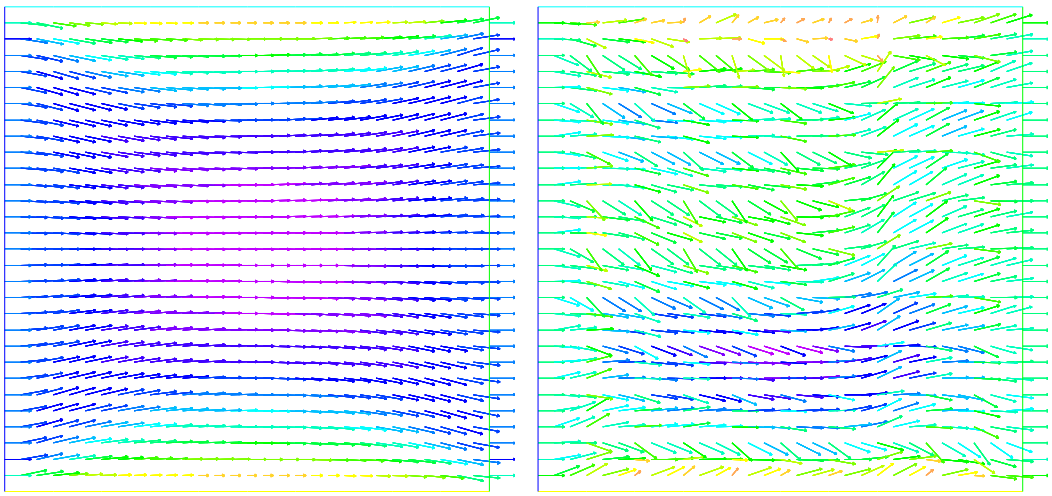

Figure 5: Simulation with $k = 20$, grid=10x10: $\nu = 0.1$ (left), $\nu = 10^{-7}$(right)

We mentioned previously, that, according to the book [1], the convergence depends on viscosity ν and body force function f. To test this behavior we performed simulations with very low ν, to "stress" the solver until the solver breaks down. During for $\nu = 10^{-6}$ there were still good results, with $\nu = 10^{-7}$ the solver stops to work properly, even increasing of iteration number for example $k = 200$ did not lead to better results. Just to test that the errors were not caused by round-off errors because of low ν, or the resulting "strange" solution is a real solution, we just increase the grid refinement and the problem was fixed.

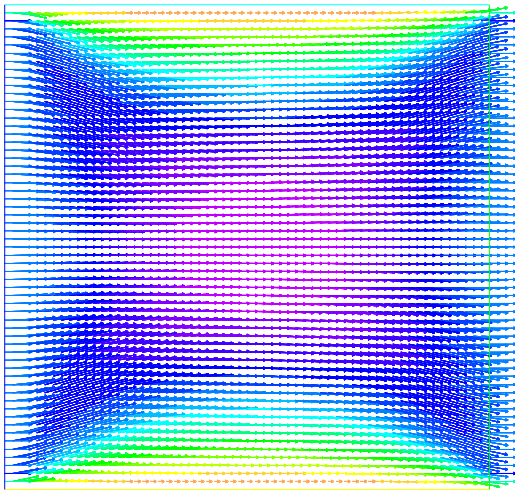

Figure 6: Simulation with $k = 20$, grid=20x20: $\nu = 10^{-7}$

Remark: The control method in simulation was not sufficient, since just apply the PDE to P2 solution is not possible. The problem is, that the second derivative of u_h does not exists (in not weak sense). The P3 approximation is not implement in FreeFem and approximation of the derivative with P2 does not lead to sufficient results. This simulation script should be impoved.

4 Navier-Stokes equations

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = f(x,t) \qquad \text{on } \Omega \times [0,T] \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases} \tag{10}$$

The Navier-Stokes equations describe dynamical behavior of a fluid.

4.1 Variational Formulation

In order to apply the FEM method to it we need to discretize the time derivative of u. We will spit the time domain $[0,T]$ in Δt uniform time steps. We will use an upper index n do describe the function value at time $(n \cdot \Delta t)$. For example $u^n(x) = u(x, n \cdot \Delta t)$.

We will also choose following discretization scheme

$$\partial_t u + u^n \cdot \nabla u^n - \nu \Delta u^{n+1} + \nabla p^{n+1} = f^{n+1}$$

The time discretization is an important part of the numerical solution. Just applying finite differences schemes to u_t is not enough. Fortunately FreeFem++ provides a special operator *convect* for time discretization of the expression $\partial_t u + u^n \cdot \nabla u^n$ [4]. We will use the FreeFem++ name *convect* for this operator.

$$\partial_t u + u^n \cdot \nabla u^n \approx \frac{1}{\Delta t}(u^{n+1} - \text{convect}(u^n, -\Delta t)) = \frac{1}{\Delta t}(u^{n+1} - u^n \circ X^n)$$

The variational formulation of the problem is similar to the variational formulation of steady state Navier-Stoke equations. The only difference is the additional time derivative u_t

$$\begin{cases} < \frac{1}{\Delta t}(u^{n+1} - u^n \circ X^n), \phi > + \nu < \nabla u^{n+1}, \nabla \phi > & \\ - \int_{\partial\Omega} \frac{\partial}{\partial n} u^{n+1} \phi ds - < p^{n+1}, \text{div}\phi > + \int_{\partial\Omega} (\phi \cdot n) p ds & =< f^{n+1}, \phi >, \quad \text{on } \Omega \\ < \text{div} u^{n+1}, \psi > & = 0 \\ u^{n+1}|_{\partial\Omega} & = g(x,t), \qquad \text{on } \partial\Omega \end{cases}$$

This equation system is to be solved iteratively going from the initial step u_0 towards u_T

4.2 Numerical Simulations

As already mentioned, the Taylor-Green Vortex is an analytical solution of Navier-Stoke equations

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = 0 \qquad \text{on } \Omega \times [0,T] \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases}$$

on domain $\Omega = (0,\pi)^2$.

For the simulation we will use Taylor-Hood Mixed elements $(P_2(\Omega)^2, P_1(\Omega))$. A simulation FreeFem++ script and description can be find in the appendix G. Additionaly to simulation in the appendix section C contains explicite formulas, pictures and a gnuplot script of the analytical Taylor-Green Vortex solution.

5 Navier-Stokes equations with relaxation.

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & + \varepsilon \partial_{tt} u + \varepsilon \partial_t \nabla p = f(x,t) \qquad \text{on } \Omega \times [0,T] \\ \text{div} u & = \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases} \tag{11}$$

[4] See FreeFem++ documentation [?], p. 198,

In order to model physics in a better way we will extend the Navier-Stokes equations with relaxation expression $\varepsilon_2 \partial_{tt} u + \varepsilon_2 \partial_t \nabla p$. It is also to notice that, from mathematically point of view, this extension changes the structure of the equations from parabolic to hyperbolic.

5.1 Variational Formulation

For the numerical simulation, of this equations, with FEM methods, we need to discretize u_{tt} and $\partial \nabla p_t$ in time. To do it we use finite differences methods[5]. In particular, in the variational formulation problem we use

$$< \partial_t \nabla p^{n+1}, \phi > \quad \approx \quad \frac{1}{\Delta t} < \nabla p^{n+1}, \phi > - \frac{1}{\Delta t} < \nabla p^n, \phi >$$

and

$$< \partial_{tt} u^{n+1}, \phi > \quad \approx \quad < \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2}, \phi > .$$

Finally the resulting variation formulation for one iteration step in time is

$$\begin{aligned}
0 \quad &= \quad (\partial_t u^{n+1} - u^n \cdot \nabla u^n, \phi) + v(\nabla u^{n+1}, \nabla \phi) - (\nabla \phi, p^{n+1}) - (u^{n+1}, \nabla \psi) - \varepsilon (p^{n+1}, \psi) \\
&\approx \quad \frac{1}{\Delta t} < u^{n+1}, \phi > + \frac{\varepsilon}{\Delta t^2} < u^{n+1}, \phi > + \nu < \nabla u^{n+1}, \nabla \phi > \\
&\quad - < \mathrm{div} \phi, p^{n+1} > + \frac{\varepsilon}{\Delta t} < \mathrm{div} \phi, p^{n+1} > \\
&\quad - \frac{1}{\Delta t} < \mathrm{convect}(u^n, -\Delta t), \phi > \\
&\quad + \frac{\varepsilon}{\Delta t^2} < -2u^n + u^{n-1}, \phi > - \frac{\varepsilon}{\Delta t} < \mathrm{div} \phi, p^{n+1} > \\
&\quad + \int_{\partial \Omega} \frac{\partial}{\partial n} u^{n+1} \phi ds + \int_{\partial \Omega} (\phi \cdot n) p ds + \frac{\varepsilon}{\Delta t^2} \int_{\partial \Omega} \frac{\partial}{\partial n} (-2u^n + u^{n-1}) \phi ds - \frac{\varepsilon}{\Delta t} \int_{\partial \Omega} (\phi \cdot n) p^n ds
\end{aligned}$$

5.2 Numerical Simulations

We will again use the Taylor-Green Vortex solution (w, q) as reference to check the quality of the the simulation. We chose again the domain $\Omega = (0, \pi)^2$, $g = w(x, t)$. Then we transform the equations in the form of the Navier-Stokes equations with relaxation (11)

$$\begin{cases}
\partial_t w + w \cdot \nabla w - \nu \Delta w + \nabla q & + \varepsilon \partial_{tt} w + \varepsilon \partial_t \nabla p = \varepsilon \partial_{tt} w + \varepsilon \partial_t \nabla q \qquad \text{on } \Omega \times [0, T] \\
\mathrm{div} w & = 0 \\
w|_{\partial \Omega} & = g(x, t) \qquad \text{on } \partial \Omega \times [0, T]
\end{cases}$$

[5] See Section A for more information

That means that the Taylor-Green-Vortex is the exact solution of the Navier-Stokes equation with relaxation. with $f(x,t) := \varepsilon_2 \partial_{tt} w(w,t) + \varepsilon_2 \partial_t \nabla q(w,t)$

6 Tests of FreeFem++ Solvers

At the beginning of the training we made some calculation of Navier-Stokes equations with and without relaxation in order to test if FreeFem++ is a right choice for this training. After the good results got by solving method[6] suggested in FreeFem documentation [2], section 9.6, for Navier-Stokes equations. we tested other solvers, to find the most suitable one for this training. Although these first experiments were done in very rough way, the results influenced much the subsequent work.

If there is no special remarks, we used $T = 5$, $\nu = \frac{1}{1000}$, $dt = \frac{T}{100}$, $\Delta x = \Delta y = \frac{\pi}{20}$. As reference we took the Taylor-Green-Vortex.

For the simulation quality measurement we used the absolute error

$$err_{abs} = ||u(T) - u_h(T)||_{L^2(\Omega)}$$

and the relative error

$$err_{rel} = \frac{||u(T) - u_h(T)||_{L^2(\Omega)}}{||u(T)||_{L^2(\Omega)}}$$

Default UMFPACK solver, dependence on ν:

ν	1	10^{-1}	10^{-2}	10^{-3}	10^{-4}	0
CT[7] [s]	28.3	30.82	32.44	32.16	32.39	32.72
err_{abs}	2.76698e-06	0.00428494	0.0819085	0.119019	0.125909	0.12806
err_{rel}	0.0267864	0.00526776	0.0423797	0.057081	0.0599728	0.0609257

UMFPACK solver with relaxation of Navier-Stokes equations: Here we tried out if the relaxation which subject of this training can improve the solution of Navier-Stokes equations similar to relaxation of the divergence constraint. That is why the reference point was Taylor-Green-Vortex with $f(x,t) = 0$

ν	10^{-3}	0
CT[8] [s]	45.43	44.94
err_{abs}	0.120235	0.129117
err_{rel}	0.057696	0.0614583

[6] The FreeFem++ documentation provides with various solving methods for Navier-Stokes equations. It was just one of them.

Relative error of Crout solver and divergence constraint relaxation[9]: The divergence constraint relaxation, means here adding $\varepsilon_2 u$ to the constraint. The resulting equation is

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = 0 \qquad \text{on } \Omega \times [0,T] \\ \text{div} u + \varepsilon_2 u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases}$$

Doing so, we got rid of the divergence constraint, which can be a source of instabilities or slow convergence.

Crout solver, erroDependence on viscosity ν:

ν	1	10^{-1}	10^{-2}	10^{-3}	10^{-4}	0
CT [s]	26.59	29.03	30.14	30.42	30.27	30.65
err_{abs}	$2.76698e-06$	0.00428505	0.0819095	0.11902	0.12591	0.128061
err_{rel}	0.0267864	0.00526789	0.0423802	0.0570815	0.059973	0.0609261

UMFPACK solver vs crout solver with different viscosities ν: We will compare calculation time and relative errors of both methods. To distinguish between Standard and Crout solver, we will use indexes S and C appropriately.

ν	1	10^{-1}	10^{-2}	10^{-3}	10^{-4}	0
$\frac{CT_S}{CT_C}$	106.431 %	106.166 %	107.631 %	105.72 %	107.004%	106.754 %
$err_{S,rel} - err_{C,rel}$	0	$-1.3 \cdot 10^{-8}$	$-5 \cdot 10^{-7}$	$-5 \cdot 10^{-7}$	$-2 \cdot 10^{-7}$	$-4 \cdot 10^{-7}$

Conclusions:

- As one can notice, the relative error grows if ν decreases. Additionally the numerical solutions become less "smooth" if ν decrease. Here some graphical examples for $err_1 = u_1 - u_h$ with $\nu = 1$ (left) $\nu = 10^{-4}$ (right) to compare. The using of the P2b (P2 elements + bubble function) does not improve the situation.

- The precision of the Crout solver with an standard relaxation grows with decreasing of the ε_2 parameter.

- Without relaxation expression the Crout solver does not work.

- The Crout solver does not work, if we replace the constraint relaxation with relaxation of Navier-Stokes equations.

- From the numerical results we can conclude, that the Crout solver with relaxation is about 6 % faster during it stay nearly as precise as the standard method. That means it is a good choice for the solving of Navier-Stokes equations.

[9]The relaxation mentioned here as *constraint relaxation*, is not Navier-Stokes equation relaxation, which is the subject, of this training. It is only necessary for the Crout solver.

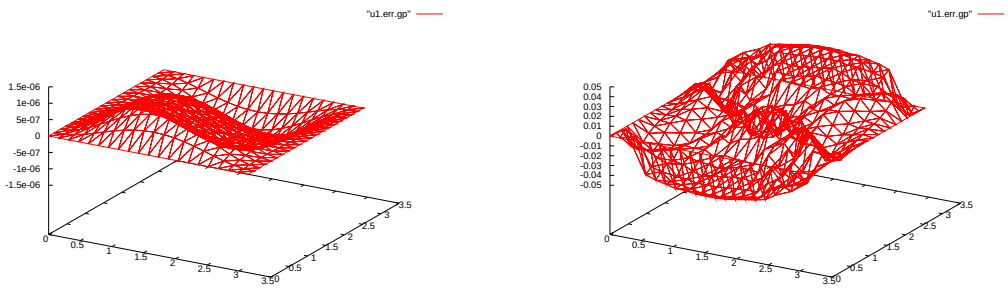

Figure 7:

- Navier-Stokes equations relaxation does not improve numerical solving of Navier-Stokes equations.
- UMFPACK provides good results, without adding of special additional relaxation expression. This will make our test less "biased" by a constraint relaxation. The UMFPACK test was chosen for subsequent tests.
- Manual performing of test series is very slow there is a need for automatizing of this process.

Remark: Besides the Crout and UMFPACK solvers I also tested LU, CG, Crout, Cholesky and GMRES FreeFem++ solvers. The only one which works was GMRES, but it was about 175 % slower than UMFPACK without any advantage in precision.

7 Software

In this training following Software was used

- *LyX*, A The WYSIWYM Document Processor with LaTeX back-end
- *FreeFem++* as, for FEM-Calculations.
- *FreeFem++-cs,* an integrated environment for FreeFem++
- *subversion,* a version control system
- noweb, a tool for Literate Programming
- Eclipse, an IDE for C++ and Python development
- Inkscape, vector graphic editor.
- TextText Inkscape plug-in, for LaTeX formulas.

- Gnuplot, plotting tool for post processing.

7.1 LyX.

LyX is an graphical document processor using "what you see is what you mean" paradigm (WYSIWYM). LyX uses LaTeX to produces documents. Additionally it supports noweb.

- Homepage of the project: http://www.lyx.org.
- License: LyX is released under the GNU General Public License, version 2 or later.
- Platforms: LyX runs on Linux/Unix, Windows, and Mac OS X.

7.2 *FreeFem++*

FreeFem++ is an implementation of a language dedicated to the finite element method. It enables to solve Partial Differential Equations (PDE). The choice of the FreeFem was motivated by the idea, to use already existing solvers for Navier-Stokes equation, replacing the standard finite elements by an divergence preserving Crouzeix-Raviart element. This approach will make possible to reuse all nice features provided with FreeFem packages, like determination of the domain geometry, triangulation, solvers, and own scripting language. The default solution of the Navier-Stoke equation can be also used to validate the implementation of the Crouzeix-Raviart element.

- Homepage of the project: `http://www.freefem.org/ff++/`.
- License: GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999 Except files : Coming form COOOL software (all files in directory src/Algo, mailto:coool@coool.mines.edu), no license. and the file mt19937ar.cpp (mailto:m-mat@math.sci.hiroshima-u.ac.jp) with own license.
- Platforms: FreeFem++ runs on Linux, FreeBSD, NetBSD, Solaris 10, Windows, and Mac OS X.

7.3 *FreeFem++-cs*

FreeFem++-cs an integrated environment for FreeFem++.

- Homepage: `http://www.ann.jussieu.fr/~lehyaric/ffcs/index.htm`
- License: is distributed under the FreeFem++ license.
- Platforms: FreeFem++-cs runs on Linux/Unix, Windows, and Mac OS X.

Remark: To work with FreeFem++-cs and FreeFem++ under Ubuntu, it was necessary just to down load a source tarball, extract it and execute

```
./configure --enable-debug
```

the –enable-debug option will add debugging information the the library. All necessary libraries including FreeFem++ will be down loaded automatically.

7.4 Subversion

During the traing I produced a big amount of documentation and sources. To log the changes there was a need in a version control system. For this task I chosen an open source version control system *Subversion*. Another very important feature of subversion is a possibility of collaboration work, in the case if someone will join to the project.

- Homepage: `http://subversion.tigris.org`
- License: `http://subversion.tigris.org/license-1.html`
- Platforms: subversion runs on Linux, FreeBSD, Solaris, Windows, and Mac OS X.

7.5 Noweb

The numerical software is a essential part of this training. To join a proper mathematical documentation with source code in FreeFem++ and other programming languages I used the Literate Programing paradigm. That means that the documentation in LaTeX format and source code will be stored in one file. This approach has following advantages:

- The source code is better to understand. It is possible to use mathematical formulas in the documentation and all other LaTeX features.
- The appendix of this report contains source code, which can be extracted from the documentation file special noweb tool.

Here some information about the noweb project:

- Homepage `http://www.cs.tufts.edu/~nr/noweb/`
- License: Noweb is copyrighted 1989–2008 by Norman Ramsey. All rights reserved. You may use and distribute noweb for any purpose, for free. You may modify noweb and create derived works, provided you retain the copyright notice, but the result may not be called noweb without my written consent. You may do anything you like with programs created with noweb. You may even sell noweb itself, for example, as part of a CD-ROM distribution, provided that what you sell is the true, complete, and unmodified noweb.
- Platforms: Linux, Windows.

Remark: I never used the Literature Programing before and this internship is a good opportunity to learn it.

7.6 Eclipse

Eclipse is a multi-language, open source software development platform with an IDE and a plug-in system to extend it. It is written in Java, and can be used to develop in Java, C++, FORTRAN, Python and other languages. In this project I used it for C++ and Python development.

- Homepage: `http://www.eclipse.org/`
- License: Eclipse Public License (EPL), the plug-ins may have different license.
- Platforms: Linux, Mac, Windows

Remark: To compile an extension for FreeFem++ a pretty complex make file is necessary. To simplify that task we will use an standard make file from the FREE_FEM_DIR/src/femlib directory. Assume the FreeFem++ sources reside in FREE_FEM_DIR directory, RPOJECT_DIR is the directory with custom libraries for FreeFem++, ItemStorage.cpp is a file with c++ source custom FreeFem++ functions. The steps below can be used to create an eclipse project for the convinient development of custom FreeFem++ objects and functions.

1. Create an PROJECT_DIR directory for new custom FreeFem++ elements
2. Select File->New->Project in eclipse. In the wizard select "make file empty project". Uncheck "use default location" and set PROJECT_DIR/src/femlib as location.
3. Create a mekafile in PROJECT directory with content:

```
INCFF="FREE_FEM_DIR/ff/local/examples++-load/Include"
all: ItemStorage
ItemStorage:
    ItemStorage.cpp ff-c++ ItemStorage.cpp
```

7.7 Inkscape

Inkscape is a free and open source vector graphics editor application

- Homepage of the project: http://www.inkscape.org/
- License: GPL.
- Platforms: LyX runs on Linux, Mac OS X and Windows.

7.8 TextText

Is an Inkscape extension (plug-in), which makes possible to insert LaTeX text and formulas in Inkscape. It is written in python.

- Homepage of the project: http://www.elisanet.fi/ptvirtan/software/textext/.
- License: Free (See LICENSE.txt inside the packages for more information).
- Platforms: LyX runs on Linux, Mac OS X and Windows.

7.9 Gnuplot

Gnuplot is free and open source visualization application.

- Homepage of the project: http://www.gnuplot.info
- License: Custom one (Modification are restricted, some distribution restrictions, check website for more information).
- Platforms: Gnuplot runs on Unix, GNU/Linux, Microsoft Windows, Mac OS, and others.

7.10 R

For the postprocessing of the simulation. That means analyzing and visualization of the results I used R. The free, open source programming language R very powerful tool for statistic calculations

- Homepage of the project: http://www.r-project.org/
- License: GPL
- Platforms: Gnuplot runs on Unix, Linux, BSD, Microsoft Windows, Mac OS.

7.11 Python

Python is a general purpose high level programming language, which I used to automatize simulations and for postprocessing. It is free and open source. There is a lot libraries written for it, including interface to gnuplot.

- Homepage of the project: http://www.python.org/
- License: OSI-approved open source license, the Python License.
- Platforms: Gnuplot runs on Unix, Linux, BSD, Microsoft Windows, Mac OS and others.

Appendix

A Nonlinear, bilinear and linear functions, long forms and identities

In order too keep calculation in the report small, we will often use a short form of different expressions like this $(u \cdot \nabla u, \phi)$, but for the transformation, calculation and programing we often need a more explicit representation. For this purpose we put all that calculations and representations in this appendix. Here we use dx for integration over volume, and ds for integration over surface. Note that in FreeFem++ we need to use x for x_1 and y for x_2.

$u \times \nabla u$ and $< u \cdot \nabla u, \phi > - \int_\Omega u \cdot \nabla u \phi dx$

$$u \times \nabla u = \begin{pmatrix} u_1 \frac{\partial}{\partial x_1} u_1 + u_2 \frac{\partial}{\partial x_2} u_1 \\ u_1 \frac{\partial}{\partial x} u_2 + u_2 \frac{\partial}{\partial x_2} u_2 \end{pmatrix}$$

$$< u \cdot \nabla u, \phi > \quad = \quad < u_1 \frac{\partial}{\partial x_1} u_1, \phi_1 > + < u_2 \frac{\partial}{\partial x_2} u_1, \phi_1 > + < u_1 \frac{\partial}{\partial x_1} u_2, \phi_2 > + < u_2 \frac{\partial}{\partial x_2} u_2, \phi_2 >$$

Δu and $- < \Delta u, \phi > = - \int_\Omega \Delta u \phi dx$

$$\Delta u = \begin{pmatrix} \Delta u_1 \\ \Delta u_2 \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1^2} u_1 + \frac{\partial}{\partial x_2^2} u_1 \\ \frac{\partial}{\partial x_1^2} u_1 + \frac{\partial}{\partial x_2^2} u_1 \end{pmatrix}$$

$$\begin{aligned} - < \Delta u, \phi > &= - < \Delta u_1, > - < \Delta u_2, \phi_2 > \\ &= < \nabla u_1, \nabla \phi_1 > - \int_{\partial\Omega} \phi_1 \frac{\partial u_1}{dn} ds + < \nabla u_2, \nabla \phi_2 > - \int_{\partial\Omega} \phi_2 \frac{\partial u_1}{dn} ds \qquad \text{Green's identity} \\ &= < \frac{\partial}{\partial x_1} u_1 \frac{\partial}{\partial x_1} \phi_1 + < \frac{\partial}{\partial x_2} u_1, \frac{\partial}{\partial x_2} \phi_1 > + < \frac{\partial}{\partial x_1} u_2, \frac{\partial}{\partial x_1} \phi_2 > + < \frac{\partial}{\partial x_1} u_2, \frac{\partial}{\partial x_2} \phi_2 > \\ &\quad - \int_{\partial\Omega} \phi_1 \frac{\partial u_1}{dn} ds - \int_{\partial\Omega} \phi_2 \frac{\partial u_1}{dn} ds \end{aligned}$$

$$\Longleftrightarrow - < \Delta u, \phi > = < \nabla u, \nabla \phi > - \int_{\partial\Omega} \phi_1 \frac{\partial u_1}{dn} ds - \int_{\partial\Omega} \phi_2 \frac{\partial u_1}{dn} ds$$

$< \phi, \nabla p >$ according to divergence theorem

$$\begin{aligned} \int_{\partial\Omega} (\phi p) \cdot n ds &= \int_\Omega \text{div}(\phi p) dx \\ \Longleftrightarrow \int_{\partial\Omega} (\phi \cdot n) p ds &= \int_\Omega \text{div}(\phi) p dx + \int_\Omega \phi \times \nabla p dx \\ \Longleftrightarrow < \phi, \nabla p > &= - < \text{div} \phi, p > + \int_{\partial\Omega} (\phi \cdot n) p ds \end{aligned}$$

$$< \phi, \nabla p >= - < \frac{\partial}{\partial x_1}\phi_1 + \frac{\partial}{\partial x_2}\phi_2, p > + \int_{\partial\Omega} (\phi \cdot n) p ds$$

$$(u_t - u^n \cdot \nabla u^n) \approx \tfrac{1}{\Delta t}(u^{n+1} - u^n \circ X^n) \text{ with } u^n \circ X^n = \text{convect}(u^n, -\Delta t)$$

$$u^n \circ X^n$$

$$\frac{1}{\Delta t}(u^{n+1} - u^n \circ X^n) = \begin{pmatrix} u_1^{n+1} - \text{convect}((u_1^n, u_2^n), -\Delta t, u_1^n) \\ \\ u_2^{n+1} - \text{convect}((u_1^n, u_2^n), -\Delta t, u_2^n) \end{pmatrix}$$

$$< u_t - u^n \cdot \nabla u^n, \phi > \approx < \tfrac{1}{\Delta t}(u^{n+1} - u^n \circ X^n), \phi >$$

$$\begin{aligned} < \frac{1}{\Delta t}(u^{n+1} - u^n \circ X^n), \phi > &= \frac{1}{\Delta t}(u_1^{n+1}, \phi_1) + \frac{1}{\Delta t}(u_1^{n+1}, \phi_1) \\ &+ \frac{1}{\Delta t}(\text{convect}((u_1^n, u_2^n), -\Delta t, u_1^n), \phi_1) \\ &+ \frac{1}{\Delta t}(\text{convect}((u_1^n, u_2^n), -\Delta t, u_2^n), \phi_2) \end{aligned}$$

$$< \partial_t \nabla p, \phi >$$

$$\begin{aligned} \partial_t \nabla p &= \frac{\nabla p^{n+1} - \nabla p^n}{\Delta t} + O(\Delta t) \\ \Longrightarrow < \partial_t \nabla p, \phi > &\approx \frac{1}{\Delta t} < \nabla p^{n+1}, \phi > - \frac{1}{\Delta t} < \nabla p^n, \phi > \\ &= \frac{1}{\Delta t}\left[-(p^{n+1}, \frac{\partial \phi_1}{\partial x_1}) - (p^{n+1}, \frac{\partial \phi_2}{\partial x_2}) + (p^n, \frac{\partial \phi_1}{\partial x_1}) + (p^n, \frac{\partial \phi_2}{\partial x_2}) \right] \end{aligned}$$

$$< \partial_{tt} u, \phi >$$

$$\partial_{tt} u = \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + O(\Delta t)$$

$$\begin{aligned} \Longrightarrow < \partial_{tt} u, \phi > &\approx < \frac{u^{n+1} + u^{n-1} - 2u^n}{\Delta t^2}, \phi > \\ &= \frac{1}{\Delta t^2} < u_1^{n+1}, \phi_1 > + \frac{1}{\Delta t^2} < u_2^{n+1}, \phi_2 > - \frac{2}{\Delta t^2} < u_1^n, \phi_1 > - \frac{2}{\Delta t^2} 2 < u_2^n, \phi_2 > \\ &+ \frac{1}{\Delta t^2}(u_1^{n-1}, \phi_1) + \frac{1}{\Delta t^2}(u_2^{n-1}, \phi_2) \end{aligned}$$

B General Remarks to Simulations

The FreeFEM scripts were derived from the script *cavity.edp* from the section 9.6 of the FreeFem++ documentation .

In order to make a lot of tests with different parameters automatically, I developed special libraries which make possible to pass parameters to the FreeFem++ simulation through a file *simulation.params.txt*.

27a ⟨*Initialize Simulation* 27a⟩≡ (44 52 53a 59a) 27b▷

```
load "ItemStorage"; // Some items constant coud be overwritten by an external file.
string [string] params; // Variable to store simulation parameters.
// load alternative settings from an external file.
ItemStorageLoad(params, "simulation.params.txt");
string resultDataDir="."; // here we will store results of simulations;
ItemStorageGet(params, "resultDataDir", resultDataDir);
```

To reuse code in different simulations, we will use the same names for constants, spaces and some parameters.

27b ⟨*Initialize Simulation* 27a⟩+≡ (44 52 53a 59a) ◁27a 27c▷

```
// Define some constatnt
real nu=1.0; // viscosity
real T=5; // Time period where u will be observed
```

We will overwrite these default parameters with special testing parameters, if they are defined in *simulation.params.txt* file in the same directory with FreeFem++ script. Overwrite ν and T.

27c ⟨*Initialize Simulation* 27a⟩+≡ (44 52 53a 59a) ◁27b

```
// load alternative settings from an external file.
ItemStorageGet(params,"nu",nu);
ItemStorageGet(params,"T",T);
```

The code can be extracted from this file using a bash script *build.sh*.

```
#!/bin/sh
# Extract FreeFem++ source code for:
# stokes.numeric.vs.exact.edp
# navier.stokes.ss.edp
# navier.stokes.num.vs.exact.edp
# navier.stokes.rx.num.vs.exact.edp
notangle -Rstokes.numeric.vs.exact.edp presentation.report.nw >edp/stokes.numeric.vs.ex
notangle -Rnavier.stokes.ss.edp presentation.report.nw >edp/navier.stokes.ss.edp
notangle -Rnavier.stokes.num.vs.exact.edp presentation.report.nw >edp/navier.stokes.num
notangle -Rnavier.stokes.rx.num.vs.exact.edp presentation.report.nw >edp/navier.stokes.
```

C Taylor-Green-Vortex

According to wikipedia: `http://en.wikipedia.org/wiki/Taylor%E2%80%93Green_vortex` the Taylor-Green-Vortex function is one of the exact solution of Navier-Stokes equation

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = 0 \\ \mathrm{div} u & = 0 \\ u|_{\partial\Omega} & = w(x,y,t) \end{cases}$$

on the space domain $\Omega = (0,\pi)^2$ time domain $[0,\infty)$.
We will call this solution (w,q) to distinguish it from other soltions.

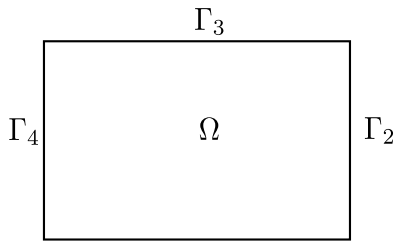

Figure 8:

For the description of the boundary of the space domain we will use $\partial\Omega = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$, where Γ_i are straight lines. This is a standard description of a rectangle boundary in FreeFem[10].

The explicit definition of the Taylor-Green Vortex solution is following:

$$\begin{aligned} w_1(x,y,t) &= \sin(x)\cos(y)F(t) \\ w_2(x,y,t) &= -\cos(x)\sin(y)F(t) \\ q((x,y),t) &= \frac{1}{4}(\cos 2x + \cos 2y)F^2(t) \\ F(t) &:= e^{-2\nu t} \end{aligned}$$

28 ⟨ *Taylor-Green-Vortex* 28 ⟩≡ (44 53a 59a)

```
// TGV prefix means Taylor-Green-Vortex
func real TGVF(real t)
{
    return exp(-2*nu*t);
}

func real TGVU1(real t)
{
```

[10] See documentation of the command *square* for more information.

```
    sin(x)*cos(y)*TGVF(t);
}

func real TGVU2(real t)
{
    -cos(x)*sin(y)*TGVF(t);
}

func real TGVP(real t)
{
    0.25*(cos(2*x)+cos(2*y))*TGVF(t)^2;
}
```

We will also calculate $\partial_t u$, $\frac{\partial}{dx} u_i$, $\frac{\partial}{dy} u_i$ and $u \times \nabla u$ for future use

$$\begin{aligned} \partial_t u_1(x,y,t) &= -2\nu u_1(x,y,t) = -2\nu \sin(x)\cos(y) e^{-2\nu t} \\ \partial_t u_2(x,y,t) &= -2\nu u_2(x,y,t) = 2\nu \cos(x)\sin(y) e^{-2\nu t} \end{aligned}$$

29 ⟨ *Taylor-Green-Vortex-dt* 29⟩≡ (44)

```
func real TGVU1dt(real t)
{
    -2*nu*sin(x)*cos(y)*TGVF(t);
}

func real TGVU2dt(real t)
{
    2*nu*cos(x)*sin(y)*TGVF(t);
}
```

$$
\begin{aligned}
\frac{\partial}{dx}u_1 &= \cos(x)\cos(y)e^{-2\nu t} \\
\frac{\partial}{dy}u_1 &= -\sin(x)\sin(y)e^{-2\nu t} \\
\frac{\partial}{dx}u_2 &= \sin(x)\sin(y)e^{-2\nu t} \\
\frac{\partial}{dy}u_2 &= -\cos(x)\cos(y)e^{-2\nu t}
\end{aligned}
$$

30 ⟨ *Taylor-Green-Vortex-dxdy* 30 ⟩≡ (44)

```
func real TGVU1dx(real t)
{
    cos(x)*cos(y)*TGVF(t);
}

func real TGVU1dy(real t)
{
    -sin(x)*sin(y)*TGVF(t);
}

func real TGVU2dx(real t)
{
    sin(x)*sin(y)*TGVF(t);
}

func real TGVU2dy(real t)
{
    -cos(x)*cos(y)*TGVF(t);
}
```

$$
\begin{aligned}
u \cdot \nabla u &= \begin{pmatrix} u_1 \frac{\partial}{\partial x} u_1 + u_2 \frac{\partial}{\partial y} u_1 \\ u_1 \frac{\partial}{\partial x} u_2 + u_2 \frac{\partial}{\partial y} u_2 \end{pmatrix} \\
&= \begin{pmatrix} u_1 \frac{\partial}{\partial x} u_1 + u_2 \frac{\partial}{\partial y} u_1 \\ u_1 \frac{\partial}{\partial x} u_2 + u_2 \frac{\partial}{\partial y} u_2 \end{pmatrix} F^2(t)
\end{aligned}
$$

$$
\begin{aligned}
\partial_{tt} u_1(x,y,t) &= 4\nu^2 u_1(x,y,t) = 4\nu^2 \sin(x)\cos(y) e^{-2\nu t} \\
\partial_{tt} u_2(x,y,t) &= 4\nu^2 u_2(x,y,t) = -4\nu^2 \cos(x)\sin(y) e^{-2\nu t}
\end{aligned}
$$

31a ⟨ *Taylor-Green-Vortex-u-dtdt* 31a ⟩≡ (64)

```
func real TGVU1dtdt(real t)
{
    4*(nu^2)*sin(x)*cos(y)*TGVF(t);
}

func real TGVU2dtdt(real t)
{
    -4*(nu^2)*cos(x)*sin(y)*TGVF(t);
}
```

$\partial_t \nabla p(x,y,t)$

$$
\nabla p = \begin{pmatrix} -\frac{1}{2}(\sin 2x) e^{-4\nu t} \\ -\frac{1}{2}(\sin 2y) e^{-4\nu t} \end{pmatrix}
$$

$$
\partial_t \nabla p = \begin{pmatrix} 2\nu(\sin 2x) e^{-4\nu t} \\ 2\nu(\sin 2y) e^{-4\nu t} \end{pmatrix}
$$

31b ⟨ *Taylor-Green-Vortex-grad-p-dt* 31b ⟩≡ (64)

```
func real TGVPdxdt(real t)
{
    2*nu*sin(x)*TGVF(t)^2;
};

func real TGVPdydt(real t)
{
    2*nu*sin(y)*TGVF(t)^2;
}
```

$\varepsilon \partial_t \nabla p(w,t)$

In order to visualize the Taylor-Green Vortex we can use a gnuplot script

32 ⟨*taylor.green.vortex.gp* 32⟩≡

```
# This is a gnu script to plot velocity vector field
# and pressure of a Taylor-Green-Vortex

# Set space domain
set xrange [0:pi]
set yrange [0:pi]

# define parameters for vortex functions
nu = 0.1
t = 0

# Define vortex functions.
TGVF(t) = exp(-2*nu*t)
TGVU1(x,y)=sin(x)*cos(y)*TGVF(t)
TGVU2(x,y)=-cos(x)*sin(y)*TGVF(t)
TGVP(x,y)=(0.25*(cos(2*x)+cos(2*y))*TGVF(t)**2)

# in order to prevent different resolution on x and y
# set samples and isosamples values to the same small value
# to reduce number of arrows
set samples 20, 20
set isosamples 20,20

# create a temporary file with x y coordinates (work around)
# Warning! if you have a file fieldxy.tmp it will be overwritten
set table "fieldxy.tmp"
splot 1
unset table

set title "Velocity of Taylor-Green-Vortex"
plot "fieldxy.tmp" u 1:2:(TGVU1($1,$2)):(TGVU2($1,$2)) w vec notitle
pause -1 "Hit any key to continue.\n"

# set result to a file
set terminal postscript enhanced color
set output "taylor.green.vortex.u.eps"
replot


# return to the dislpay view again
set terminal x11
```

```
# Plot pressure.
# Increase resolution.
set samples 50, 50
set isosamples 50,50
set cntrparam levels 10

set title "Pressure of Taylor-Green-Vortex"
set contour
splot TGVP(x,y) notitle
pause -1 "Hit any key to continue."

# Save result to a file.
set terminal postscript enhanced color
set output "taylor.green.vortex.p.eps"
replot

set terminal x11 # return to the display
```

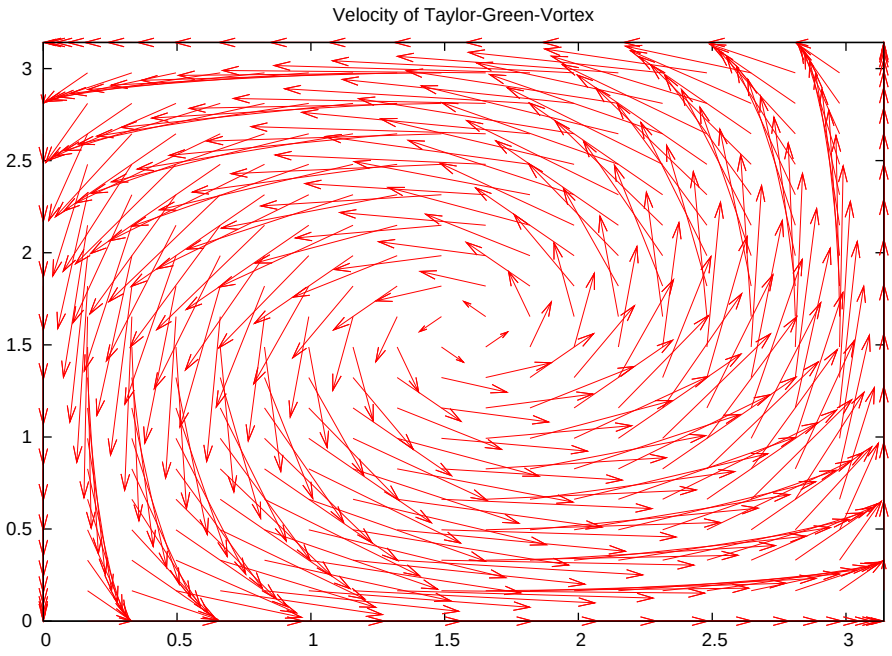

Figure 9: Taylor-Green-Vortex velocity u

D Useful macros

D.1 Norm macros

We define an L_2 norm on $\Omega \subset \mathbb{R}^2$

34 ⟨*Norm macros* 34⟩≡ (44 52 58b 64)

```
macro ML2D2Norm(u1,u2,Th) (sqrt(int2d(Th)(u1^2 + u2^2)))
// end of macro ML2D2Norm
```

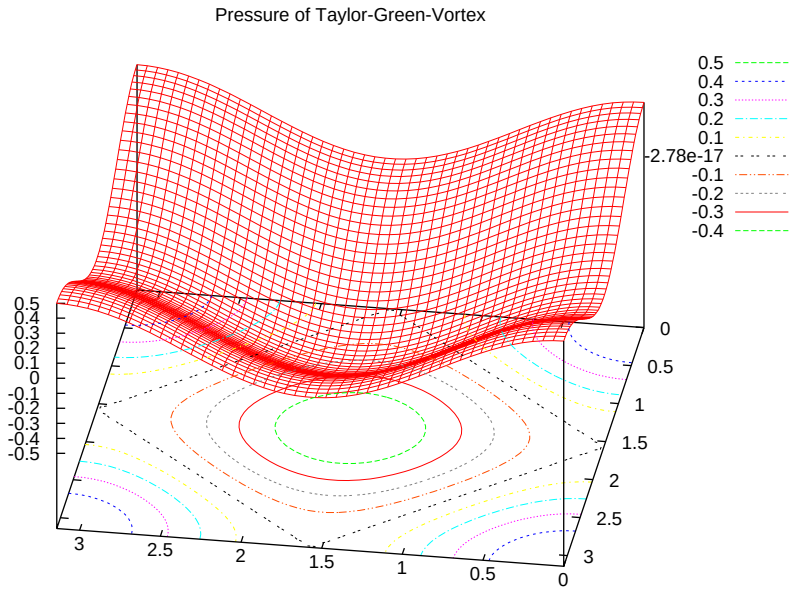

Figure 10: Taylor-Green-Vortex pressure p

D.2 Timer macros

In order to measure execution time, we need a FreeFem variable of type real[int] variable(2), containing of two real values, which must be initialized with 0.

35 ⟨*Timer macros* 35⟩≡ (44 52 53a 59a)

```
// In order to store execution time in a variable "timer", declare it as
// real[int] timer(2)
// and initialize it with 0
// The spent time will be stored in timer[1]

macro TimerStart(timer)
{
    timer[1]=clock();
}
// End of macro TimerStart

macro TimerStop(timer)
```

```
{
    if(timer[1]!=0)
    {
        timer[0] = timer[0]+(clock()-timer[1]);
        timer[1] = 0;
    }
}// End of macro TimerStop
```

D.3 Data storage macros

The next macro is used to store function data for visualization with gnuplot (This is modified code from the [2], example 3.2.)

37 ⟨*Data storage macros* 37⟩≡ (44 52 58b 64)

```
// MESH is a mesh where finite elements of type FEM are defined.
// F is a function RxR -> R of type FEM.
// FILENAME is a name of the file where data will be stored.
// The data will be stored as lines in format " x y u(x,y) "
macro StorePlotData(Th, f, filename)
{
    ofstream ff(filename);
    fespace P1d1(Th,P1);
    P1d1 interpF = f;

    for (int i=0;i<Th.nt;i++)
    {
        for (int j=0; j <3; j++)
        {
            ff«Th[i][j].x « " "« Th[i][j].y« " " « interpF[][P1d1(i,j)]«endl;
        }
        ff « Th[i][0].x « " "« Th[i][0].y« " "«interpF[][P1d1(i,0)]«"\n\n\n";
    }
}
// Create CSV file with results
macro CreateFunctionValuesCSV(path)
{
    string sep = " ";
    ofstream of(path);
    of « "x" « sep « "y" « sep « "t";
    of « sep « "u1" « sep « "u2" « sep « "p" « sep « "fem_i" « endl;
}

// EOM

// end of macro StorePlotData
// Store vector filed [u1,u2] and pressure into output stream as lines of format
// x y t u1(x,y,t) u2(x,y,t) p(x,y,t) fem_i
// x,y, are space coordinates, u1, u2, p functions values at time t
// fem_i index of finite element, in order to split data on blocks later
macro AppendFunctionVauesToCSV(Th, u1,u2, p, t, filename)
{
    ofstream of(filename,append);
    fespace P1d1(Th,P1);
    P1d1 interpF1 = u1;
    P1d1 interpF2 = u2;
```

```
    P1d1 interpP = p;

    for (int i=0;i<Th.nt;i++)
    {
         for (int j=0; j <3; j++)
         {
             of « Th[i][j].x « " " « Th[i][j].y« " "« t;
             of « " " « interpF1[][P1d1(i,j)];
             of « " " « interpF2[][P1d1(i,j)];
             of « " " « interpP[][P1d1(i,j)];
             of « " " « i « endl;
         }
     }
 }
// End of StoreVectorField macro
```

E Numerical Simulations of Stokes-Equations

We will perform some numerical simulation, in order to find out how different parameters, like viscosity ν, Mesh refinement N and choice of the finite dimensional Spaces $(X_h, M_h) \subset (X, M)$ influence the solution. As reference we will take a *Taylor-Green Vortex* solution w, q at time $t = 0$. This function is a exact solution of the Stokes equation

$$\begin{cases} -\nu\Delta u + \nabla p & = f \qquad \text{on } \Omega \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g \qquad \text{on } \partial\Omega \end{cases}$$

On $\Omega = [0,\pi] \times [0,\pi]$, $f = -(\partial_t w(x,y,0) + w \cdot \nabla w)|_{t=0})$ and $g(x,y) := w(x,y,0)$.

We also will use (x,y) instead of (x_1,x_2), in order not to get confused between mathematical calculation and definitions required by FreeFem++.

We define body forces $f(x,y) = (f_1, f_2)(x,y)$

$$f(x,y) = \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix} = -\left(\begin{pmatrix} \partial_t w_1(x,0) \\ \partial_t w_2(x,0) \end{pmatrix} + \begin{pmatrix} w_1 \frac{\partial}{\partial x} w_1 + w_2 \frac{\partial}{\partial y} w_1 \\ w_1 \frac{\partial}{\partial x} w_2 + w_2 \frac{\partial}{\partial y} w_2 \end{pmatrix} \right)(x,y,0)$$

38 ⟨*Stokes, define f* 38⟩≡ (44)

```
func f1=-(TGVU1dt(0)+TGVU1(0)*TGVU1dx(0)+TGVU2(0)*TGVU1dy(0));
func f2=-(TGVU2dt(0)+TGVU1(0)*TGVU2dx(0)+TGVU2(0)*TGVU2dy(0));
```

We define boundary conditions $g(x,y) := (g_1, g_2)(x,y) = w(x,y,0)$

39a ⟨*Stokes, define g* 39a⟩≡ (44)
```
func g1=TGVU1(0);
func g2=TGVU2(0);
```

For the performance measurement we will use two timers

39b ⟨*Stokes, define timers* 39b⟩≡ (44 53a 59a)
```
// Declare timers (See timer macros for more information)

// One setup timer
real[int] setupTimer(2);
setupTimer = 0;
// One calculation timer
real[int] calculationTimer(2);
calculationTimer = 0;
```

Now we will construct an appropriate FEM space. First we define space domain $\Omega = (0,\pi)^2$ with uniform mesh and appropriate space refinement.

39c ⟨*Stokes, mesh* 39c⟩≡ (44 53a 59a)
```
// Set default refinement values for the Mesh
int gridRefinment=20; // Space grid refinement.

// Change it by the values stored in parameter simulation (if necessary)
ItemStorageGet(params,"gridRefinment",gridRefinment);

TimerStart(setupTimer);
// define square domain [0,pi]x[0,pi]
mesh Th=square(gridRefinment,gridRefinment, [pi*x,pi*y]);
TimerStop(setupTimer);
```

Then we define functional FEM-spaces X_h and M_h on Ω. The velocity space X_h is a vector field consisting of continuous of partially quadratic two dimensional polynomial.

$$X_h = \{v \in H^1(\Omega)^2 | v_h|_T = (\mathcal{P}_2)^2 \forall T \in \mathcal{T}_h\}$$

and the pressure space M_h consist of partial functions

$$M_h = \{v \in H^1(\Omega) | v_h|_T = (P_1) \forall T \in \mathcal{T}_h\}$$

Our solution consists of the velocity function u

$$\begin{aligned} X_h \quad \ni \quad & u : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^2 \\ & u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \end{aligned}$$

and pressure functions p

$$M_h \ni p : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$$

We will also use test functions $\phi = (\phi_1, \phi_2) \in X_h$ and $\psi \in M_h$

40 ⟨*Stokes, FEM space and test function definitions* 40⟩≡ (44)

```
TimerStart(setupTimer);
fespace Xh(Th,[P2,P2]); // define finite space for u (velocity)
fespace Mh(Th,P1); // define finite space for p (pressure)

Xh [u1,u2]; // Define velocity function.
Xh [phi1,phi2]; // Define test functions for velocity space.
Mh p; // Define pressure function.
Mh psi; // Define test function for pressure space.
TimerStop(setupTimer);
```

Now we will define a variational formulation of the problem

$$\begin{cases} a(u,\phi) + b(\phi,p) & = f(\phi) \qquad \forall \phi \in X \\ b(u,\psi) & = 0 \qquad \forall \psi \in M \end{cases}$$

with

$$\begin{aligned} a(v,\phi) &= -\nu < \Delta v, \phi > \\ &= \nu \int_\Omega \left[\frac{\partial}{\partial x} u_1 \frac{\partial}{\partial x} \phi_1 + \frac{\partial}{\partial y} u_1 \frac{\partial}{\partial y} \phi_1 + \frac{\partial}{\partial x} u_2 \frac{\partial}{\partial x} \phi_2 + \frac{\partial}{\partial y} u_2 \frac{\partial}{\partial y} \phi_2 \right] d\lambda + \int_{\partial\Omega} R_1 ds \end{aligned}$$

$$\begin{aligned} b(\phi,p) &= < \phi, \nabla p > \\ &= -\int_\Omega \frac{\partial \phi_1}{\partial x} p d\lambda - \int_\Omega \frac{\partial \phi_2}{\partial y} p d\lambda + \int_{\partial\Omega} R_2 ds \end{aligned}$$

The appropriate boundary integrals $\int_{\partial\Omega} R_1 ds$ and $\int_{\partial\Omega} R_2 ds$ will be calculated automatically by FreeFem++ from boundary conditions g

41a ⟨*Stokes, variation formulation* 41a⟩≡ (44)

```
TimerStart(setupTimer);
problem  Stokes ([u1,u2,p],[phi1,phi2,psi]) =
   int2d(Th)(
             + nu * ( dx(u1)*dx(phi1) + dy(u1)*dy(phi1)
             +  dx(u2)*dx(phi2) + dy(u2)*dy(phi2) )
             - dx(phi1)*p - dy(phi2)*p  // +b([phi1,phi2],p)
             - dx(u1)*psi - dy(u2)*psi  // +b([u1,u2],psi)
            )
   - int2d(Th) ( f1*phi1+f2*phi2)
   // + on(1,2,3, 4,u1=0,u2=0) ;
   + on(1,2,3,4, u1=g1,u2=g2);

TimerStop(setupTimer);
```

W will solve equations and log calculation time.

41b ⟨*Stokes, solve* 41b⟩≡ (44)

```
TimerStart(calculationTimer);
// Calculate u
Stokes;
// Stop the timer
TimerStop(calculationTimer);
```

After calculation we will calculate the absolute L2 error of our numerical solution u_h , which is $u_{err} = ||u - u_h||$.

42a ⟨*Stokes, Calculate absolute L2 error for u* 42a⟩≡ (44)

```
// Calculate error function, which is difference between calculated and exact solutions
Xh [u1Err,u2Err]=[u1,u2]-[TGVU1(0),TGVU2(0)];
// Calculate and print L2 norm of the error.
real errNorm = ML2D2Norm(u1Err,u2Err,Th);
// Calculate interpolation of the exact solution.
real interpolatedUNorm = ML2D2Norm(TGVU1(0),TGVU1(0),Th);

// Print results
cout « "L2 norm of the absolute error = " « errNorm « endl;
cout « "L2 norm of the u = " « interpolatedUNorm « endl;
cout « "L2 norm of the relative error = " « errNorm/interpolatedUNorm « endl;
```

We store the absolute error u_{err}, calculation time, information about solver and other parameters for simulation into special file resultDataDir/resultFilename for postprocessing. The default path of this file is "./simulation.result.txt", .

42b ⟨*Stokes, Store simulation results results* 42b⟩≡ (44)

```
string resultFilename="simulation.result.txt";
ItemStorageGet(params,"resultFilename",resultFilename);
ItemStorageAdd(params,"errUL2Norm",errNorm);
ItemStorageAdd(params,"interpolatedUL2Norm",interpolatedUNorm);
ItemStorageAdd(params,"solver","UMFPACK");
ItemStorageAdd(params,"Xh","P2");
ItemStorageAdd(params,"Mh","P1");
// Print setup time.
ItemStorageAdd(params,"setupTime",setupTimer[0]);
cout « "Setup time: " « setupTimer[0] « endl;
// Print calculation time.
cout « "Calculation time: " « calculationTimer[0] « endl;
ItemStorageAdd(params,"calculationTime",calculationTimer[0]);
ItemStorageStore(params,",",resultDataDir+"/"+resultFilename);
```

If it is required by simulation parameters we will also store the calculated function values

43a ⟨*Stokes, Store function values* 43a⟩≡ (44)

```
bool storeFunctionValues=false;
string functionValuesFilename="u.simulated.csv";
string meshFilename="mesh";

ItemStorageGet(params,"storeFunctionValues",storeFunctionValues);
ItemStorageGet(params,"meshFilename",meshFilename);
ItemStorageGet(params,"functionValuesFilename",functionValuesFilename);


string meshPath = resultDataDir+"/"+meshFilename;
string functionValuesPath = resultDataDir+"/"+functionValuesFilename;

// Store mesh remove previous values.
if(storeFunctionValues)
{
    cout « "Store Mesh data to " « meshPath « "[.points,.faces]" « endl;
    savemesh(Th,meshPath, [x,y,0]);
    cout « "Create CVS with results  " « functionValuesPath « endl;
    CreateFunctionValuesCSV(functionValuesPath);
    AppendFunctionVauesToCSV(Th,u1,u2,p,0,functionValuesPath);
}
```

If it is required we can plot the result of the simulation:

43b ⟨*Stokes, plot result* 43b⟩≡ (44 52 64)

```
// Flag indicating the at the end of the simulation the result must be plotted
bool usePlot = false;
ItemStorageGet(params,"usePlot",usePlot);
// Plot vector field.
if(usePlot)
{
    plot([u1,u2]); // plot velocity vector field
}
```

44 ⟨*stokes.numeric.vs.exact.edp* 44⟩≡

⟨*Initialize Simulation* 27a⟩
⟨*Taylor-Green-Vortex* 28⟩
⟨*Taylor-Green-Vortex-dt* 29⟩
⟨*Taylor-Green-Vortex-dxdy* 30⟩
⟨*Stokes, define f* 38⟩
⟨*Stokes, define g* 39a⟩
⟨*Timer macros* 35⟩
⟨*Stokes, define timers* 39b⟩
⟨*Stokes, mesh* 39c⟩
⟨*Stokes, FEM space and test function definitions* 40⟩
⟨*Stokes, variation formulation* 41a⟩
⟨*Stokes, solve* 41b⟩
⟨*Norm macros* 34⟩
⟨*Stokes, Calculate absolute L2 error for u* 42a⟩
⟨*Stokes, Store simulation results results* 42b⟩
⟨*Data storage macros* 37⟩
⟨*Stokes, Store function values* 43a⟩
⟨*Stokes, plot result* 43b⟩

Additional files:

- *stokes.py.* is a python-script for to produce automatically experiment series with FreeFem++ script *stokes.numeric.vs.exact.edp.* For each experiment the *stokes.py* script creates a separate directory such that it is possible also to store function values for further analysis and visualization. The main results of all stokes tests be stored in cvs stokes.result.csv for postprocessing.
- *stokes.r* is a R-script was used for analyzing data of the experiment series produce by stokes.py. Some results were visualized.

F Numerical simulation of the Oseen problem

The goal of this simulation is to solve a problem using an iterative method described in the section Steady-State-Navier-Stokes equations

$$\begin{cases} u \cdot \nabla u - \nu \Delta u + \nabla p & = 0 \\ \mathrm{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \end{cases}$$

on domain $\Omega = (0,1)^2$ with $g_1 = 0.1$ on $\Gamma_2 \cup \Gamma_4$ and $u_2 = 0$ on $\partial\Omega$, $\nu = 0.1$.
For the performance measurement we will use a timer

45a ⟨*Steady state Navier-Stokes, define timers* 45a⟩≡ (52)

```
// Declare and initialize the timer (See timer macros for more information)

// One calculation timer
real[int] calculationTimer(2);
calculationTimer = 0;
```

On the domain Ω , we will construct an appropriate FEM mesh. We begin with definition of the space domain $\Omega = (0,1)^2$ and appropriate grid refinements

45b ⟨*Steady state Navier-Stokes, mesh* 45b⟩≡ (52)

```
// Set default refinement values for the Mesh
int gridRefinment=20; // Space grid refinement.

// Change it by the values stored in simulation parameters (if necessary)
ItemStorageGet(params,"gridRefinment",gridRefinment);

// define square domain [0,1]x[0,1]
mesh Th=square(gridRefinment,gridRefinment);
```

Then we define a mixed FEM-spaces (X_h, M_h) on Ω. The velocity space X_h is a vector field consists of continues functions, which are two dimensional polynomial of second degree.

$$X_h = \{v \in H^1(\Omega)^2 | v_h|_T = (\mathcal{P}_2)^2\}$$

and the pressure space M_h con sits of partially linear functions

$$M_h = \{v \in H^1(\Omega)^2 | v_h|_T = (P_1)\}$$

Our solution consists of the velocity function u

$$\begin{aligned} X_h \quad \ni \quad & u : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^2 \\ & u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \end{aligned}$$

and pressure functions p

$$M_h \ni p : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$$

We will also use test functions $\phi = (\phi_1, \phi_2) \in X_h$ and $\psi \in M_h$ and functions u_p, p_p for termporal storage of the results from the previous iteration step.

46 ⟨*Steady state Navier-Stokes, FEM space and test function definitions* 46⟩≡ (52)

```
fespace Xh(Th,[P2,P2]); // Define finite element space for u (velocity)
fespace Mh(Th,P1); // Define finite element space for p (pressure)
fespace Yh(Th,P2); // Define finite element space for interpolations

Xh [u1,u2]; // Define velocity function.
Xh [up1,up2]; // Define velocity function for storage of previous step.
Xh [phi1,phi2]; // Define test functions for velocity space.
Mh p; // Define pressure function.
Mh pp; // Define pressure function for storage of previous step.
Mh psi; // Define test function for pressure space.
```

Now we will define a variation formulation of the problem

$$\begin{cases} N(u,u,\phi) + a(u,\phi) + b(\phi,p) & = 0 \qquad \forall \phi \in X_h \\ b(u,\psi) & = g(x,y) \qquad \forall \psi \in M_h \end{cases}$$

with

$$\begin{aligned} N(v,u,\phi) &= < u \times \nabla u, \phi > \\ a(v,\phi) &= -\nu < \Delta v, \phi > \\ b(\phi,p) &= < \phi, \nabla p > \end{aligned}$$

Which we will solve iteratively.
We will start the iteration with $u = u^0 = 0$.
Then in each integration step we will solve an Oseen problem

$$\begin{cases} N(u^k,u^{k+1},\phi) + a(u^{k+1},\phi) + b(\phi,p) & = 0 \qquad \forall \phi \in X_h \\ b(u,\psi) & = g(x,y) \qquad \forall \psi \in M_h \end{cases}$$

with given u^k and

$$N(u_p,u,\phi) = \int_\Omega \left[u_{p,1} \frac{\partial}{\partial x} u_1 \phi_1 + u_p \frac{\partial}{\partial y} u_1 \phi_1 + u_{p,1} \frac{\partial}{\partial x} u_2 \phi_2 + u_{p,2} \frac{\partial}{\partial y} u_2 \phi_2 \right] dV$$

$$a(u,\phi) = \nu \int_\Omega \left[\frac{\partial}{\partial x} u_1 \frac{\partial}{\partial x} \phi_1 + \frac{\partial}{\partial y} u_1 \frac{\partial}{\partial y} \phi_1 + \frac{\partial}{\partial x} u_2 \frac{\partial}{\partial x} \phi_2 + \frac{\partial}{\partial y} u_2 \frac{\partial}{\partial y} \phi_2 \right] dV + \int_{\partial\Omega} R_1 ds$$

$$b(\phi,p) = -\int_\Omega \frac{\partial \phi_1}{\partial x} p dV - \int_\Omega \frac{\partial \phi_2}{\partial y} p dV + \int_{\partial\Omega} R_2 ds$$

The appropriate boundary integrals $\int_{\partial\Omega} R_1 ds$ and $\int_{\partial\Omega} R_2 ds$ will be calculated automatically by FreeFem++ from boundary conditions g.

u_p stays for the previous step

47 ⟨*Steady state Navier-Stokes, Oseen problem* 47⟩≡ (52)

```
int k=0; // iteration index
problem OseenProblem([u1,u2,p],[phi1,phi2,psi],init=k)
           = int2d(Th)(
               + up1*dx(u1)*phi1+up2*dy(u1)*phi1 // Non linear part.
               + up1*dx(u2)*phi2+up2*dy(u2)*phi2 // Non linear part.
               + nu*(
                     dx(u1)*dx(phi1) + dy(u1)*dy(phi1) // Laplace operator.
                   + dx(u2)*dx(phi2) + dy(u2)*dy(phi2) // Laplace operator.
               ) // laplace operator.
               - p*dx(phi1) - p*dy(phi2) // +b([phi1,phi2],p)
```

```
- dx(u1)*psi - dy(u2)*psi) // + b([u1,u2],psi)
+ on(2,4,u1=0.1,u2=0)  // Boundary conditions.
+ on(1,3,u1=0,u2=0); // Boundary conditions.
```

We will define maximal iteration number, which can be change using *simulation.params.txt* file from an external application.

48a ⟨*Steady state Navier-Stokes, setup iteration parameters* 48a⟩≡ (52)

```
int maxIterationNumber=20;
ItemStorageGet(params,"maxIterationNumber",maxIterationNumber);
```

In order to control quality of approximation we will compute L2 norm of the function

$$f^n(x) \quad = \quad N(u^n, u^n, \phi) + a(u^n, \phi) + b(\phi, p^n)$$

48b ⟨*Steady state Navier-Stokes, define F norm* 48b⟩≡ (52)

```
real[int] fNorm(maxIterationNumber+1);
fNorm=-1;

// Note usually dx of u1 is not more continuous, try to find
// a better way to solve, my be through a high order interpolation
// of u1 and u

Xh [f1,f2]=[u1*dx(u1)+u2*dy(u1)-nu*(dxx(u1)+dyy(u1))+dx(p),
u1*dx(u2)+up2*dy(u2)-nu*(dxx(u2)+dyy(u2))+dy(p)];
```

If required, we will also save intermediate results of calculations, instead of time t we will store an iteration index k

49 ⟨*Steady state Navier-Stokes, Init function value storage* 49⟩≡ (52)

```
bool storeFunctionValues=false;
string functionValuesFilename="u.simulated.csv";
string meshFilename="mesh";

ItemStorageGet(params,"storeFunctionValues",storeFunctionValues);
ItemStorageGet(params,"meshFilename",meshFilename);
ItemStorageGet(params,"functionValuesFilename",functionValuesFilename);


string meshPath = resultDataDir+"/"+meshFilename;
string functionValuesPath = resultDataDir+"/"+functionValuesFilename;

// Store mesh, remove previous values.
if(storeFunctionValues)
{
    cout « "Store Mesh data to " « meshPath « "[.points,.faces]" « endl;
    savemesh(Th,meshPath, [x,y,0]);
    cout « "Create CVS with results  " « functionValuesPath « endl;
    CreateFunctionValuesCSV(functionValuesPath);
}
```

Finally we will perform required iteration steps, saving each time results to a file (if required).

50 ⟨*Steady state Navier-Stokes, perform iterations* 50⟩≡ (52)

```
// Init [u1,u2] with zero
[u1,u2]=[0,0];

Yh dxu1,dyu1,dxu2,dyu2; // Some functions for interpolation
TimerStart(calculationTimer);

for(k = 1; k <= maxIterationNumber;k++)
{
    [up1,up2] = [u1,u2]; // Save previous results
    OseenProblem; // Calculate ([u1,u2],p) depending on the previous values stored in ([up
    // Store result if required.
    if(storeFunctionValues)
    {
         TimerStop(calculationTimer);
         AppendFunctionVauesToCSV(Th,u1,u2,p,k,functionValuesPath);
         TimerStart(calculationTimer);
    }

     // Calculate Norm of the result.
     // [f1,f2]=[u1*dx(u1)+u2*dy(u1)-nu*(dxx(u1)+dyy(u1))+dx(p),
     // u1*dx(u2)+up2*dy(u2)-nu*(dxx(u2)+dyy(u2))+dy(p)];
     // fNorm[k]= ML2D2Norm(f1,f2,Th);

    // Because the derivatives of u1,u2 are not more continues we will
    // We will interpolate them.

    dxu1 = dx(u1);
    dyu1 = dy(u1);
    dxu2 = dx(u2);
    dyu2 = dy(u2);
    [f1,f2]=[u1*dx(u1)+u2*dy(u1)-nu*(dx(dxu1)+dy(dyu1))+dx(p),
    u1*dx(u2)+up2*dy(u2)-nu*(dx(dxu2)+dy(dyu2))+dy(p)];
    fNorm[k]= ML2D2Norm(f1,f2,Th);
}

TimerStop(calculationTimer);
```

The results of the simulation we will store into special file resultDataDir/resultFilename for postprocessing. The default path of this file is "./simulation.result.txt", . Additionally we will store a file with the name f.norm.csv in the resultDataDir directory to analyze convergence of the method.

51 ⟨*Steady state Navier-Stokes, store simulation results* 51⟩≡ (52)

```
string resultFilename="simulation.result.txt";
ItemStorageGet(params,"resultFilename",resultFilename);
ItemStorageAdd(params,"solver","UMFPACK");
ItemStorageAdd(params,"Xh","P2");
ItemStorageAdd(params,"Mh","P1");

// Print calculation time.
cout « "Calculation time: " « calculationTimer[0] « endl;
ItemStorageAdd(params,"calculationTime",calculationTimer[0]);
string fNormFilename = "f.norm.csv";
ItemStorageAdd(params,"fNormFilename",fNormFilename);
ItemStorageStore(params,",",resultDataDir+"/"+resultFilename);

// Store fNorm.
string fNormPath=resultDataDir+"/"+fNormFilename;

ofstream of(fNormPath);
string sep = " "; // Separator
of « "k" « sep « "fNorm" « endl;

for (int k=1;k<=maxIterationNumber;k++)
{
    of « k « sep « fNorm[k]«endl;
}
```

At the end we can plot the result. For this purpose we will reuse code from the Stoke Simulation.

52 ⟨*navier.stokes.ss.edp* 52⟩≡
⟨*Initialize Simulation* 27a⟩
⟨*Timer macros* 35⟩
⟨*Steady state Navier-Stokes, define timers* 45a⟩
⟨*Steady state Navier-Stokes, mesh* 45b⟩
⟨*Steady state Navier-Stokes, FEM space and test function definitions* 46⟩
⟨*Steady state Navier-Stokes, Oseen problem* 47⟩
⟨*Steady state Navier-Stokes, setup iteration parameters* 48a⟩
⟨*Norm macros* 34⟩
⟨*Steady state Navier-Stokes, define F norm* 48b⟩
⟨*Data storage macros* 37⟩
⟨*Steady state Navier-Stokes, Init function value storage* 49⟩
⟨*Steady state Navier-Stokes, perform iterations* 50⟩
⟨*Steady state Navier-Stokes, store simulation results* 51⟩
⟨*Stokes, plot result* 43b⟩

Remark: After some experiments I figured out, that it is hard to control the quality of interpolation, at least with methods I used, because the derivative of the resulting function is not more continuous. This part of the needs to be improved.

Additional: *navier.stokes.ss.py* is used for automatize testing with different parameters, and to visualize a certain iteration step of certain experiment.

G Numerical simulation Taylor-Green-Vortex as Navier-Stokes equations

The Taylor-Green-Vortex is an analytical solution of Navier-Stoke equations

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & = 0 \qquad \text{on } \Omega \times [0,T] \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases} \tag{12}$$

on the space domain domain $(0,\pi)^2$

For the simulation we will use Taylor-Hood Mixed elements $(P_2(\Omega)^2, P_1(\Omega))$

We will reuse a parts of the code containing mesh definitions, boundary conditions and timers from the previous sections.

53a ⟨*navier.stokes.num.vs.exact.edp* 53a⟩≡ 58b▷

```
⟨Initialize Simulation 27a⟩
⟨Taylor-Green-Vortex 28⟩
⟨Timer macros 35⟩
⟨Stokes, define timers 39b⟩
⟨Stokes, mesh 39c⟩
```

We also need a time discretization of the Time space. The refinement of the time space $[0,T]$ is described through the variable *timeStepNum*.

53b ⟨*Navier Stokes, time discretization* 53b⟩≡ (58b 59a)

```
int timeStepNum=100;
ItemStorageGet(params,"timeStepNum",timeStepNum);
```

We define appropriate FEM element spaces on domain Ω, velocity u, pressure p, test functions ϕ, ψ and some additional spaces to store results of the previous iteration step.

54a ⟨*Navier Stokes, FEM space and test function definitions* 54a⟩≡ (58b)

```
TimerStart(setupTimer);
fespace Xh(Th,[P2,P2]); // define finite space for u (velocity)
fespace Mh(Th,P1); // define finite space for p (pressure)

Xh [u1,u2]; // Define velocity function.
Xh [phi1,phi2]; // Define test functions for velocity space.
Xh [up1,up2]; // Storage for "Previous velocity" in time iterations
Mh p; // Define pressure function.
Mh psi; // Define test function for pressure space.
TimerStop(setupTimer);
```

In each time step we will need to solve variation formulation of the problem

$$\begin{cases} < \frac{1}{\Delta t}(u^{n+1} - u^n \circ X^n), \phi > + \nu < \nabla u^{n+1}, \nabla \phi > - < p^{n+1}, \mathrm{div}\phi > & = \\ - \int_{\partial\Omega} \frac{\partial}{\partial n} u^{n+1} \phi ds + \int_{\partial\Omega} (\phi \cdot n) p ds & = \quad \text{on } \Omega, \forall \phi \in X_h \\ < \mathrm{div} u^{n+1}, \psi > & = 0 \quad \forall \psi \in M_h \\ u^{n+1}|_{\partial\Omega} & = g(x,t) \text{ on } \partial\Omega \end{cases}$$

We will start with $u_0 = w(x,0)$ where w is an exact solution. The length of the step is Δt. Thus after timeStepNum iterations we will calculate final velocity $u(T)$ and pressure $p(T)$

54b ⟨*Navier-Stokes, Iteration variables* 54b⟩≡ (58b)

```
int n = 0; // Iteration index.

real dt = T/timeStepNum;
real t=0;
real alpha=1/dt;

[u1,u2]=[TGVU1(0),TGVU2(0)]; // Set initial value.
```

$$
\begin{aligned}
0 &= < \partial_t u^{n+1} - u^n \cdot \nabla u^n, \phi > + v < \nabla u^{n+1}, \nabla \phi > - < \nabla \phi, p^{n+1} > - < u^{n+1}, \nabla \psi > \\
&\approx \frac{1}{\Delta t} \int_\Omega \left[u_1^{n+1} \phi_1 + \frac{1}{\Delta t} u_1^{n+1} \phi_1 \right] dV \\
&\quad \nu \int_\Omega \left[\frac{\partial}{\partial x} u_1^{n+1} \frac{\partial}{\partial x} \phi_1^{n+1} + \frac{\partial}{\partial y} u_1^{n+1} \frac{\partial}{\partial y} \phi_1 + \frac{\partial}{\partial x} u_2^{n+1} \frac{\partial}{\partial x} \phi_2 + \frac{\partial}{\partial y} u_2^{n+1} \frac{\partial}{\partial y} \phi_2 \right] dV \\
&\quad - \int_\Omega \frac{\partial}{\partial x} \phi_1 p^{n+1} dV - \int_\Omega \frac{\partial}{\partial y} \phi_2, p^{n+1} dV \\
&\quad - \int_\Omega (\frac{\partial}{\partial x} u_1^{n+1} \psi dV - \int_\Omega \frac{\partial}{\partial y} u_2^{n+1} \psi dV \\
&\quad - \int_\Omega \mathrm{convect}((u_1^n, u_2^n), -\Delta t, u_1^n) \phi_1 dV - \int_\Omega \mathrm{convect}((u_1^n, u_2^n), -\Delta t, u_2^n) \phi_2) \\
&\quad - \int_{\partial\Omega} \frac{\partial}{\partial n} u^{n+1} \phi ds + \int_{\partial\Omega} (\phi \cdot n) p^{n+1} ds
\end{aligned}
$$

55 ⟨*Navier Stokes, step problem* 55⟩≡ (58b)

```
problem  NS ([u1,u2,p],[phi1,phi2,psi],init=n) =
   int2d(Th)(
             alpha*( u1*phi1 + u2*phi2)
             + nu * ( dx(u1)*dx(phi1) + dy(u1)*dy(phi1)
             +  dx(u2)*dx(phi2) + dy(u2)*dy(phi2) )
            - dx(phi1)*p - dy(phi2)*p  // +b([phi1,phi2],p)
            - dx(u1)*psi - dy(u2)*psi  // +b([u1,u2],psi)
            )
  + int2d(Th) ( -alpha*convect([up1,up2],-dt,up1)*phi1 -alpha*convect([up1,up2],-dt,up2)*p
  + on(1,2,3,4, u1=TGVU1(t),u2=TGVU2(t));
```

The computation of the boundary integrals $+\int_{\partial\Omega}\frac{\partial}{\partial n}u^{n+1}\phi ds + \int_{\partial\Omega}(\phi\cdot n)pds$ we will leave to FreeFem. The expression above we will rewrite in FreeFem suitable notation.

- The value u^{n+1} is the value to be calculated (In FreeFem++ $[u1, u2]$)
- u^n is the previous calculated values (In FreeFem++ $[up1, up2]$)
- $+\int_{\partial\Omega}\frac{\partial}{\partial n}u^{n+1}\phi ds + \int_{\partial\Omega}(\phi\cdot n)pds$ is replaced with *on* command.
- $\frac{1}{\Delta t}$ is in FreeFem++ α

If required, we will save intermediate calculation results, for each time t.

56 ⟨*Navier-Stokes, Init function value storage* 56⟩≡ (58b 64)

```
bool storeFunctionValues=false;
string functionValuesFilename="u.simulated.csv";
string meshFilename="mesh";

ItemStorageGet(params,"storeFunctionValues",storeFunctionValues);
ItemStorageGet(params,"meshFilename",meshFilename);
ItemStorageGet(params,"functionValuesFilename",functionValuesFilename);


string meshPath = resultDataDir+"/"+meshFilename;
string functionValuesPath = resultDataDir+"/"+functionValuesFilename;

// Store mesh remove previous values.
if(storeFunctionValues)
{
    cout « "Store Mesh data to " « meshPath « "[.points,.faces]" « endl;
    savemesh(Th,meshPath, [x,y,0]);
    cout « "Create CVS with results  " « functionValuesPath « endl;
    CreateFunctionValuesCSV(functionValuesPath);
    p=TGVP(0);
    AppendFunctionVauesToCSV(Th,u1,u2,p,0,functionValuesPath);
}
```

Now we will perform the time iterations, during the iterations we will measure the calculation time and if it is required we will store calculated values to the *functionValuesFilename* file

57a ⟨*Navier-Stokes, perform iterations* 57a⟩≡ (58b)

```
TimerStart(calculationTimer);

for(n = 1; n <= timeStepNum;n++)
{
    t=n*dt;
    [up1,up2] = [u1,u2]; // Save previous results
    NS; // Calculate ([u1,u2],p) depending on the previous values stored in ([up1,up2])
    // Store result if required.
    if(storeFunctionValues)
    {
         TimerStop(calculationTimer);
         AppendFunctionVauesToCSV(Th,u1,u2,p,t,functionValuesPath);
         TimerStart(calculationTimer);
    }

     // Calculate Norm of the result.
     // [f1,f2]=[u1*dx(u1)+u2*dy(u1)-nu*(dxx(u1)+dyy(u1))+dx(p),
     // u1*dx(u2)+up2*dy(u2)-nu*(dxx(u2)+dyy(u2))+dy(p)];
     // fNorm[k]= ML2D2Norm(f1,f2,Th);
}

TimerStop(calculationTimer);
```

To compute the absolute errors we compute the $L_2(\Omega)$ norm of the difference $u_h(T) - u(T)$. Where $u_h(T)$ is a simulated velocity $u(T)$ is the exact solution of the Navier-Stoke problem.

57b ⟨*Navier-Stokes, calculate error* 57b⟩≡ (58b 64)

```
Xh [u1Err,u2Err]=[u1,u2]-[TGVU1(T),TGVU2(T)];
// Calculate and print L2 norm of the error.
real uNorm = ML2D2Norm(u1,u2,Th);
real errNorm = ML2D2Norm(u1Err,u2Err,Th);
cout « "L2 norm of the absolute error = " « errNorm « endl;
cout « "L2 norm of the u = " « uNorm « endl;
cout « "L2 norm of the relative error = " « errNorm/uNorm « endl;
```

We will store result of simulation and simulation parameters to a file "dataDir/resultFilename", which is usually "./simulation.result.txt", for postprocessing.

58a ⟨*Navier-Stokes, Store parameters and results* 58a⟩≡ (58b 64)

```
string resultFilename="simulation.result.txt";
ItemStorageGet(params,"resultFilename",resultFilename);
ItemStorageAdd(params,"errL2Norm",errNorm);
ItemStorageAdd(params,"interpolatedUL2Norm",uNorm);
ItemStorageAdd(params,"solver","UMFPACK");
ItemStorageAdd(params,"Xh","P2");
ItemStorageAdd(params,"Mh","P1");
// Print setup time.
ItemStorageAdd(params,"setupTime",setupTimer[0]);
cout « "Setup time: " « setupTimer[0] « endl;
// Print calculation time.
cout « "Calculation time: " « calculationTimer[0] « endl;
ItemStorageAdd(params,"calculationTime",calculationTimer[0]);
ItemStorageStore(params,",",resultDataDir+"/"+resultFilename);
```

58b ⟨*navier.stokes.num.vs.exact.edp* 53a⟩+≡ ◁53a

⟨*Navier Stokes, time discretization* 53b⟩
⟨*Navier Stokes, FEM space and test function definitions* 54a⟩
⟨*Navier-Stokes, Iteration variables* 54b⟩
⟨*Navier Stokes, step problem* 55⟩
⟨*Data storage macros* 37⟩
⟨*Navier-Stokes, Init function value storage* 56⟩
⟨*Navier-Stokes, perform iterations* 57a⟩
⟨*Norm macros* 34⟩
⟨*Navier-Stokes, calculate error* 57b⟩
⟨*Navier-Stokes, Store parameters and results* 58a⟩

Additional files: *navier.stokes.py.* This python scripts is used for automatic simulations with different parameters. It also contains code for visualization of a certain simulation (experiment) and certain time t.

H Numerical simulation Taylor-Green-Vortex as Navier-Stokes equations with relaxation.

We will simulate a relaxation Navier-Stoke equation

$$\begin{cases} \partial_t u + u \cdot \nabla u - \nu \Delta u + \nabla p & +\varepsilon_2 \partial_{tt} u + \varepsilon_2 \partial_t \nabla p = f(x,t) \qquad \text{on } \Omega \times [0,T] \\ \text{div} u & = 0 \\ u|_{\partial\Omega} & = g(x,t) \qquad \text{on } \partial\Omega \times [0,T] \end{cases}$$

with $g(x,t) = w(x,t)$ and $f(x,t) = +\varepsilon_2 \partial_{tt} w(w,t) + \varepsilon_2 \partial_t \nabla q(w,t)$ on the space domain $\Omega := [0,\pi] \times [0,\pi]$, where m w, q is Taylor-Green Vortex solution. The Taylor-Green Vortex w, q solves this equation, so we can use it as a reference point, for analyzing the quality of the numerical solution.

Like in the previous simulation we initialize the simulation and define the mesh. Again, the refinement of the mesh is controlled over the *gridRefinment* variable. And the refinement of the time domain is controlled over the variable *timeStepNum*.

59a ⟨*navier.stokes.rx.num.vs.exact.edp* 59a⟩≡ 64▷

```
⟨Initialize Simulation 27a⟩
⟨Taylor-Green-Vortex 28⟩
⟨Timer macros 35⟩
⟨Stokes, define timers 39b⟩
⟨Stokes, mesh 39c⟩
⟨Navier Stokes, time discretization 53b⟩
```

For our simulation we will need an additional parameter ε which can be passed from an external program

59b ⟨*Navier Stokes rx, epsilon definition* 59b⟩≡ (64)

```
real epsilon=0.0001;
ItemStorageGet(params,"epsilon",epsilon);
```

We will define a body force function $f(x,t) = \varepsilon\partial_{tt}w(w,t) + \varepsilon\partial_t\nabla q(w,t)$

60a ⟨*Navier Stokes rx, f definition* 60a⟩≡ (64)

```
func real f1(real t)
{
    (epsilon*TGVU1dtdt(t)+epsilon*TGVPdxdt(t));
}

func real f2(real t)
{
     (epsilon*TGVU2dtdt(t)+epsilon*TGVPdydt(t));
}
```

For the simulation we will use Taylor-Hood mixed elements $(P_2(\Omega)^2, P_1(\Omega))$, but if necessary, they can be replaced by Crouzeix-Raviart elements. To do so, we need to write *P2b* instead of *P2*. For our simulation we will need velocity u, pressure p, test functions ϕ, ψ and some additional variables to store previous steps in iterations.

60b ⟨*Navier Stokes rx, FEM space and test function definitions* 60b⟩≡ (64)

```
TimerStart(setupTimer);
fespace Xh(Th,[P2,P2]); // define finite space for u (velocity)
fespace Mh(Th,P1); // define finite space for p (pressure)

Xh [u1,u2]; // Define velocity function.
Xh [phi1,phi2]; // Define test functions for velocity space.
Xh [up1,up2]; // Storage for "Previous velocity" in time iterations
Xh [upp1,upp2]; // another addition storage for velocity (previous to previous)
Mh p; // Define pressure function.
Mh pp; // Here the previous pressure will be stored
Mh psi; // Define test function for pressure space.
```

We will also initialize the function values u^0, u^{-1}, and p^0 using exact solution to keep the initial error as small as possible.

61 ⟨*Navier-Stokes rx, Iteration variables* 61⟩≡ (64)

```
int n = 0; // Iteration index.

real dt = T/timeStepNum;
real t=0;
real alpha=1/dt;

[u1,u2]=[TGVU1(0),TGVU2(0)]; // Set initial value.
[up1,up2]=[TGVU1(-dt),TGVU2(-dt)];
p=TGVP(0);
```

Similar to previous case we will need to solve in each time step the equation

$$
\begin{aligned}
0 \quad = \quad & (\partial_t u^{n+1} - u^n \cdot \nabla u^n, \phi) + v(\nabla u^{n+1}, \nabla \phi) - (\nabla \phi, p^{n+1}) - (u^{n+1}, \nabla \psi) - \varepsilon(p^{n+1}, \psi) \\
\approx (\quad & \frac{1}{\Delta t} + \frac{\varepsilon}{\Delta t^2}) < u^{n+1}, \phi > + \nu < \nabla u^{n+1}, \nabla \phi > \\
& -(1 + \frac{\varepsilon}{\Delta t}) < \mathrm{div} \phi, p^{n+1} > \\
& -\frac{1}{\Delta t} < \mathrm{convect}(u^n, -\Delta t), \phi > \\
& +\frac{\varepsilon}{\Delta t^2} < -2u^n + u^{n-1}, \phi > + \frac{\varepsilon}{\Delta t} < \mathrm{div} \phi, p^n > \\
& -\int_{\partial\Omega} \frac{\partial}{\partial n} u^{n+1} \phi ds + \int_{\partial\Omega} (\phi \cdot n) p ds + \frac{\varepsilon}{\Delta t^2} \int_{\partial\Omega} \frac{\partial}{\partial n} (-2u^n + u^{n-1}) \phi ds - \frac{\varepsilon}{\Delta t} \int_{\partial\Omega} (\phi \cdot n) p^n ds \\
& - < f, \phi >
\end{aligned}
$$

The long form of the equation, without boundary integrals is

$$
\begin{aligned}
0 \quad = \quad & < \partial_t u^{n+1} - u^n \cdot \nabla u^n, \phi > + v < \nabla u^{n+1}, \nabla \phi > - < \nabla \phi, p^{n+1} > - < u^{n+1}, \nabla \psi > \\
\approx \quad & (\frac{1}{\Delta t} + \frac{\varepsilon}{\Delta t^2}) \int_\Omega \left[u_1^{n+1} \phi_1 + u_2^{n+1} \phi_2 \right] dV \\
& \nu \int_\Omega \left[\frac{\partial}{\partial x} u_1^{n+1} \frac{\partial}{\partial x} \phi_1^{n+1} + \frac{\partial}{\partial y} u_1^{n+1} \frac{\partial}{\partial y} \phi_1 + \frac{\partial}{\partial x} u_2^{n+1} \frac{\partial}{\partial x} \phi_2 + \frac{\partial}{\partial y} u_2^{n+1} \frac{\partial}{\partial y} \phi_2 \right] dV \\
& -(1 + \frac{\varepsilon}{\Delta t}) \left[\int_\Omega \frac{\partial}{\partial x} \phi_1 p^{n+1} dV + \int_\Omega \frac{\partial}{\partial y} \phi_2, p^{n+1} dV \right] \\
& -\int_\Omega \frac{\partial}{\partial x} u_1^{n+1} \psi dV - \int_\Omega \frac{\partial}{\partial y} u_2^{n+1} \psi dV \\
& -\int_\Omega \mathrm{convect}((u_1^n, u_2^n), -\Delta t, u_1^n) \phi_1 dV - \int_\Omega \mathrm{convect}((u_1^n, u_2^n), -\Delta t, u_2^n) \phi_2) \\
& -\frac{2\varepsilon}{\Delta t^2} \int_\Omega \left[u_1^n \phi_1 dV + \frac{1}{\Delta t} u_2^n \phi_2 \right] dV + \frac{\varepsilon}{\Delta t^2} \int_\Omega \left[u_1^{n-1} \phi_1 dV + \frac{1}{\Delta t} u_2^{n-1} \phi_2 \right] dV \\
& +\frac{\varepsilon}{\Delta t} \int_\Omega \left[\frac{\partial}{\partial x} \phi_1 p^n + \frac{\partial}{\partial y} \phi p^{n+1} \right] dV
\end{aligned}
$$

62 ⟨*Navier Stokes rx, step problem* 62⟩≡ (64)

```
problem  NSrx ([u1,u2,p],[phi1,phi2,psi],init=n)
    = int2d(Th)
    (
        (alpha+epsilon*alpha^2)*( u1*phi1 + u2*phi2)
        + nu *
        (
            dx(u1)*dx(phi1) + dy(u1)*dy(phi1)
            + dx(u2)*dx(phi2) + dy(u2)*dy(phi2)
        )
```

```
        - (1+epsilon*alpha)*(p*dx(phi1) + p*dy(phi2))  // +b(phi,p)
        - dx(u1)*psi- dy(u2)*psi  // +b([u1,u2],psi) // +b(u,psi)

    )
  + int2d(Th)
    (
        -alpha*convect([up1,up2],-dt,up1)*phi1
        -alpha*convect([up1,up2],-dt,up2)*phi2
        -(2*epsilon*alpha^2)*up1*phi1
        -(2*epsilon*alpha^2)*up2*phi2
        +(epsilon*alpha^2)*upp1*phi1
        +(epsilon*alpha^2)*upp2*phi2
        +(epsilon*alpha)*(pp*dx(phi1) + pp*dy(phi2))
        -f1(t)*phi1-f2(t)*phi2
     )
   + on(1,2,3,4, u1=TGVU1(t),u2=TGVU2(t));
TimerStop(setupTimer);
```

Now we will perform the time iteration, during the iterations we will measure the calculation time and if it is required we will store calculated values to a file.

63 ⟨*Navier-Stokes nx, perform iterations* 63⟩≡ (64)

```
TimerStart(calculationTimer);

for(n = 1; n <= timeStepNum;n++)
{
    t=n*dt;
    [upp1,upp2] = [up1,up2]; // Save previous results
    [up1,up2] = [u1,u2];
    pp = p;
    NSrx; // Calculate ([u1,u2],p)
    // Store result if required.
    if(storeFunctionValues)
    {
         TimerStop(calculationTimer);
         AppendFunctionVauesToCSV(Th,u1,u2,p,t,functionValuesPath);
         TimerStart(calculationTimer);
    }
}

TimerStop(calculationTimer);
```

Finlay, we will compute $L_2(\Omega)^2$, norm of the error $u_h(T) - u(T)$ and store results like in previous simulation.

We will also plot a final result if required.

64 ⟨*navier.stokes.rx.num.vs.exact.edp* 59a⟩+≡ ◁59a
⟨*Navier Stokes rx, epsilon definition* 59b⟩
⟨*Taylor-Green-Vortex-u-dtdt* 31a⟩
⟨*Taylor-Green-Vortex-grad-p-dt* 31b⟩
⟨*Navier Stokes rx, f definition* 60a⟩
⟨*Navier Stokes rx, FEM space and test function definitions* 60b⟩
⟨*Navier-Stokes rx, Iteration variables* 61⟩
⟨*Navier Stokes rx, step problem* 62⟩
⟨*Data storage macros* 37⟩
⟨*Navier-Stokes, Init function value storage* 56⟩
⟨*Navier-Stokes nx, perform iterations* 63⟩
⟨*Norm macros* 34⟩
⟨*Navier-Stokes, calculate error* 57b⟩
⟨*Navier-Stokes, Store parameters and results* 58a⟩
⟨*Stokes, plot result* 43b⟩

Additional files: *navier.stokes.rx.py.* This python scripts is used for automatic simulations with different parameters. It also contains code for visualization of a certain simulation (experiment) and certain time t. Additionally it contains code to make a animation from the simulation and stored free codecs Dirac and Theora. It use gnuplot for visualization of a the function at certain point t and then combine them together using a free and open source program *ffmpeg.*

I Custom Libraries:

For the automatic tests and postprocessing I developed special libraries, which are used in the scripts stokes.py, navier.stokes.ss.py, navier.stokes.py and navier.stokes.rx.py.

ItemStorage: C++

ItemStorage war written for FreeFem in order to exchange data between FreeFem script and external programs in convenient way. The Idea of this library is to be able to produce a large number of simulation with different parameters and then store result for postprocessing. Technically parameter to be pass, are stored in a file in name,"value" format.

ItemStorage: Python

The python module *itemstorage.py* is used to interchange data with FreeFem.

Experiment: Python

The module experiment is used to launch a FreeFEM script with certain parameters. It is used in all automatic scripts.

ExperimentResult: Python

The module *experimentresult.py* is used in python postprocessing scripts, to search directories and collect the results of the previous experiments. This data can be than stored as a .csv file for data analysis.

Plot: Python

The module *plot* is used for data visualization. It helps for example to visualize velocity u or pressure p of certain experiment at certain time t and to produce animations for dynamical simulations.

References

[1] Michele Benzi. *Numerical solution of saddle point problems*. Cambridge University Press, 2005.

[2] F. Hecht. Freefem++, third edition, version 3.3-2.

[3] L. Ridgway Scott Susanne C. Brenner. *The Mathematical Theory of Finite Element Methods, Third Edition*. Springer, 2008.

[4] Chiara Simeoni Theodoros Katsaounis, Charalambos Makridakis. Relaxation finite element schemes for the incomplressible navier-stokes equations.

[5] L.R. Schott V. Girault. A quasi-local interpolationoperator preserving thediscrete divergence. *CALCOLO*, 40:1-19, 2003.