



EUROPEAN CONSORTIUM FOR MATHEMATICS IN INDUSTRY

28th ECMI Modelling Week Final Report

19.07.2015—26.07.2015
Lisboa, Portugal



Departamento de Matemática
Universidade de Coimbra

Group 5

Patient-specific blood flow modelling

Anastasiia Bozhok

Grenoble INP (ENSIMAG)

Gaetano Formato

Lappeenranta University of Technology

Christoph Sadée

University College Dublin

Milena Mandic

University of Novi Sad

Pavel Iliev

University of Sofia

Janvier Ukwizagira

Lappeenranta University of Technology

Instructor: Dr. Jorge Tiago

CEMAT, IST-ULisboa

Abstract

The purpose of this work is to develop a 2D mathematical model of blood flow and to forecast some cardiovascular diseases behavior such as aneurysms and stenosis. In order to complete our model, we should consider the variability in design and size of cardiovascular systems between individuals.

We compute a blood flow model based on Stokes and Navier-Stokes equations. Furthermore, we would like to simulate the behaviour of aneurysms and stenosis in the cardiovascular system by using the model made up. Our primary interests are in the mechanical properties of the system, which will be the principal target of the models discussed in this paper.

Finally, we would like to minimize the misfit between the computed blood flow and the real cross-sections data of specific patients by controlling the choice of the inflow boundary conditions.

5.1 Introduction

Medical images allow us to obtain good representations of blood vessels, even in pathological cases. However concerning the dynamics of the blood itself, medical images can only give us some sparse velocity measurements. Medical doctors would make a good use of accurate blood flow simulations in predicting either the evolution of certain pathologies, or the effect of some therapies. Our project consists in finding a model which allows one to reconstruct the blood flow in the complete domain by using the information obtained from the velocity measurements.

We have largely ignored the enormous and medically very important subject of cardiovascular pharmacology. Drug delivery and action is undoubtedly influenced by haemodynamics but little is known about these interactions.

5.2 Cardiovascular system and diseases

The cardiovascular system (Fig. 5.1) is composed of the heart, which pumps the blood through the body and a network of blood vessels which transport blood to the body and drain it from the body tissues to the heart. All parts of the system work together, but will be considered individually. Blood is ejected from the heart in discrete pulses under relatively high pressure into the main arteries (at a lower pressure in the pulmonary circulation than in the systemic circulation) where it flows through a network of branching arteries of decreasing size to the arterioles and then the capillaries where it delivers oxygen and nutrients to the tissues and removes carbon dioxide. Blood is collected from the capillaries through merging venules and returns to the heart at low pressure through a network of veins.

The most common diseases, affecting the cardiovascular system are aneurysm's (Fig. 5.3) and stenosis' (Fig. 5.2).

A stenosis is an obstruction of blood flow caused by the development of plaque of atherosclerosis.

An aneurysm is a gradual dilation of an arterial segment over a period of years. The aneurysm wall stretches and becomes thinner and weaker than normal arterial walls. This phenomenon could cause the rupture of the vein wall, causing massive haemorrhage, which is often lethal.

5.3 Model

Assumptions

In order to develop a 2D mathematical model for blood flow, we fix some assumptions:

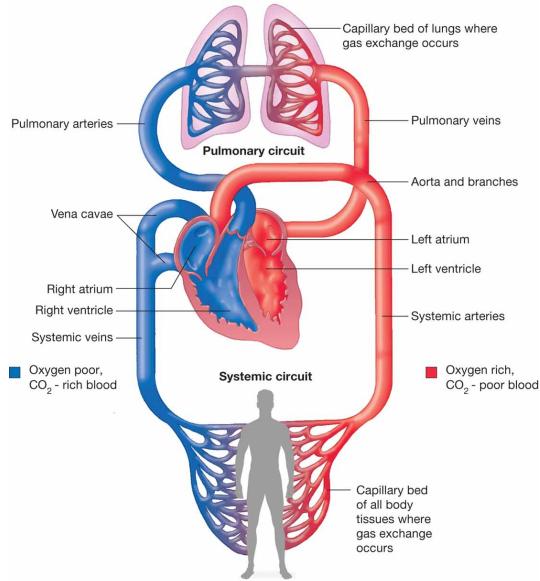


Figure 5.1: A sketch of the cardiovascular system.

- 1) Blood is modeled as homogeneous fluid flow .
- 2) We consider in our simulation the stationary case. Therefore the pulsatile nature of blood is neglected .
- 3) The temperature is constant at 37 °C.
- 4) Blood is assumed to be a Newtonian fluid with constant viscosity equal to 3.5 mPa · s .
- 5) Blood density is considered to be constant $1.06 \cdot 10^3 \text{ Kg/m}^3$.
- 6) We assume the non-slip boundary at the artery wall, that is, the velocity is set to be zero in every direction.
- 7) The Reynolds number for blood is 100 – 400. This corresponds to a non turbulent motion.
- 8) We consider blood to be an incompressible fluid. From the mass conservation equation, we get:

$$\nabla \cdot \mathbf{u}(x; y) = 0$$

where \mathbf{u} is the 2D velocity field of the fluid.

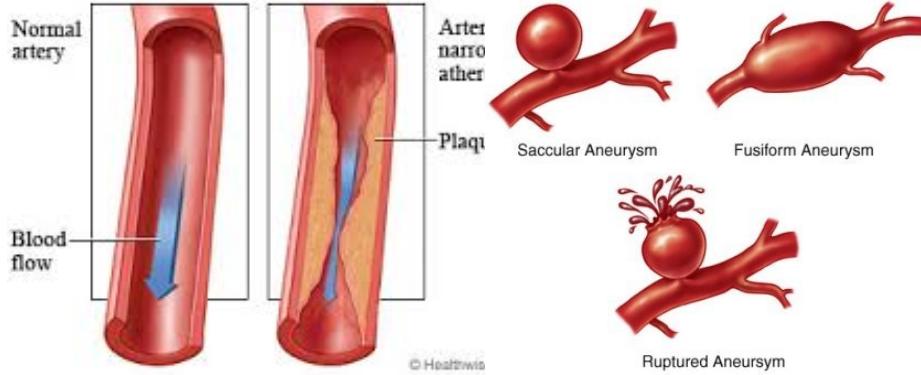


Figure 5.2: Stenosis development

Figure 5.3: Aneurism development

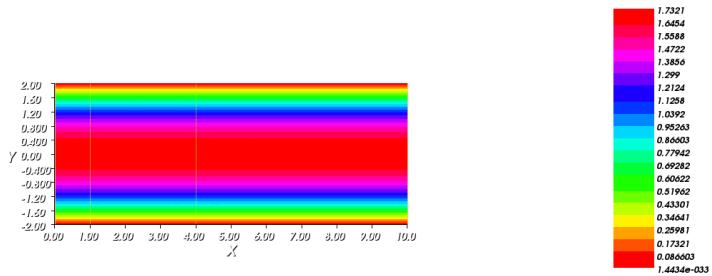


Figure 5.4: Steady state solution to non-stress Stokes problem with Dirichlet boundary condition on one boarder.

Mathematical model for blood flow

In order to obtain patient specific blood flow from measurements we have to define a model for the blood flow. Under the assumptions that have been discussed in the previous chapter the for steady blood flow is expressed by the steady Navier-Stokes equations. Any type of flow is generally described by Navier-Stokes equation (PDE). The assumptions are used to adjust the PDE to fit our model. The first two equation in (5.1) correspond to linear momentum and mass conservation respectively. Here \mathbf{u} denotes the velocity field, p is the pressure, ρ and μ are density and viscosity of the blood. We assume that Ω represents the fluid domain, and the boundaries of Ω consist of Γ_{Wall} , which represents the inside wall of the artery and two artificial boundaries Γ_{In} and Γ_{Out} . Γ_{In} corresponds to the border of inlet flow and Γ_{Out} to the border of outlet flow of the considered section of the artery. We consider Dirichlet boundary condition on Γ_{In} and Neumann boundary condition on Γ_{Out} and no-slip Dirichlet boundary condition on Γ_{Wall} .

$$\begin{cases} \rho(\mathbf{u} \cdot \nabla \mathbf{u}) - \nabla \cdot \mathbf{T}(\mathbf{u}, p) = 0 & \text{on } \Omega \\ \nabla \cdot \mathbf{u} = 0 & \text{on } \Omega \\ \mathbf{u} = 0 & \text{on } \Gamma_{Wall} \\ \mathbf{n} \cdot \mathbf{T} = 0 & \text{on } \Gamma_{Out} \\ \mathbf{u} = \mathbf{c} & \text{on } \Gamma_{In} \end{cases} \quad (5.1)$$

Using the Cauchy stress principle

$$\mathbf{T}(\mathbf{u}, p) = -p\mathbf{I} + 2\mu D(\mathbf{u}) \quad (5.2)$$

where

$$D(\mathbf{u}) = \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^T}{2} \quad (5.3)$$

and when $\nabla \cdot \mathbf{u} = 0$, it can be shown that

$$\text{div}(D(\mathbf{u})) = \Delta \mathbf{u} \quad (5.4)$$

hence, the first equation in (5.1) takes the form:

$$\rho(\mathbf{u} \cdot \nabla \mathbf{u}) - \mu \Delta \mathbf{u} + \nabla p \mathbf{I} = 0 \quad \text{on } \Omega \quad (5.5)$$

We use this system to predict the behaviour of the velocity magnitudes and pressure in the artery for cases of stenosis and aneurism. The function \mathbf{c} , defining the inlet flow is set to be a parabolic function for simulations, while in reality it depends on patient specific blood flow, it's importance will be discussed later.

Weak Formulation

To obtain a weak formulation of the system of differential equations (5.1) we define the function spaces

$$\mathbb{S} = \{\mathbf{u} \in [H^1(\Omega)], \mid \mathbf{u} = \mathbf{c} \quad \text{on } \Gamma_{In}, \mathbf{u} = \mathbf{0} \quad \text{on } \Gamma_{Wall}\} \quad (5.6)$$

$$\mathbb{V} = \{\mathbf{u} \in [H^1(\Omega)], \mid \mathbf{u} = \mathbf{0} \quad \text{on } \Gamma_{In} \cup \Gamma_{Wall}\} \quad (5.7)$$

The weak form of (5.1) is to find $\mathbf{u} \in \mathbb{S}$ and $p \in Q$, ($Q = L^2(\Omega)$) such that for every $\mathbf{v} \in \mathbb{V}$ and $q \in Q$

$$\int_{\Omega} \rho(\mathbf{u} \cdot \nabla \mathbf{u}) \cdot \mathbf{v} \, d\mathbf{x} + \int_{\Omega} \mu \nabla \mathbf{u} : \nabla \mathbf{v} \, d\mathbf{x} - \int_{\Omega} p \nabla \cdot \mathbf{v} \, d\mathbf{x} - \int_{\Omega} q \nabla \cdot \mathbf{u} \, d\mathbf{x} = 0 \quad (5.8)$$

This expression is obtained by multiplying the two governing equations in (5.1) by test functions \mathbf{v} and q , respectively, integrating over the domain Ω and subtracting one equation from the other. Because the integral is a linear functional it can be further simplified to the form

$$\int_{\Omega} [\rho(\mathbf{u} \cdot \nabla \mathbf{u}) \cdot \mathbf{v} + \mu \nabla \mathbf{u} : \nabla \mathbf{v} - p \nabla \cdot \mathbf{v} - q \nabla \cdot \mathbf{u}] = 0 \quad (5.9)$$

We denote this problem (WP). Obtaining the weak form of the system (5.1) is a key part in the present study. Solving (WP) form leads to weaker restrictions for the smoothness of the solution than solving directly the differential form (5.1). We define finite dimensional subspaces $\mathbb{V}_h \subset \mathbb{V}$ and $Q_h \subset Q$ and take finite dimensional approximations $\mathbf{u}_h \in \mathbb{V}_h$ and $p_h \in Q_h$:

$$\mathbf{u}_h = \sum_{j=1}^{N_u} u_j \phi_j \quad , \phi_j \in \mathbb{V}_h \quad (5.10)$$

$$p_h = \sum_{l=1}^{N_p} p_l \psi_l \quad , \psi_l \in Q_h \quad (5.11)$$

We use finite element spaces \mathbb{V}_h and Q_h corresponding to the Taylor-Hood elements (P2,P1). One method for numerically solving (5.9) is by replacing \mathbf{u} and q in (5.9) with their finite approximations (5.10) and (5.11) and obtain a discrete system. But since the expression in the integral is non-linear the discrete system is not going to be a linear system of equations. In the present study the numerical algorithm that we use to solve the weak form of (5.1) is an iterative method based on the Newton method that we will discuss in details in the next chapter. This method allows us to solve a non-linear problem that models the blood flow and hence to solve the optimization problem.

5.4 The Newton Method

The numerical algorithm that we use to solve (5.9) is based on the classical Newton's method for solving non-linear equations based on Taylor series expansion. Let

$$F(\mathbf{u}, p) = \int_{\Omega} [\mathbf{v} \cdot \rho(\mathbf{u} \cdot \nabla \mathbf{u}) + \mu \nabla \mathbf{v} \cdot \nabla \mathbf{u} - \nabla \mathbf{v} \cdot p \mathbf{I} - q \nabla \cdot \mathbf{u}] \quad (5.12)$$

The problem can be formulated as: find (\mathbf{u}, p) such that $F(\mathbf{u}, p) = 0$. This is a non-linear problem and for the formulation of the numerical algorithm that we use to solve it we define the Gateaux derivative of F for all the directions $(\delta \mathbf{u}, \delta p)$ as

$$F'(\mathbf{u}, p).(\delta \mathbf{u}, \delta p) = \frac{\delta F}{\delta \mathbf{u}}(\mathbf{u}, p).(\delta \mathbf{u}) + \frac{\delta F}{\delta p}(\mathbf{u}, p).(\delta p) \quad (5.13)$$

where

$$\frac{\delta F}{\delta \mathbf{u}}(\mathbf{u}, p).(\delta \mathbf{u}) = \lim_{\epsilon \rightarrow 0} \frac{F(\mathbf{u} + \epsilon \delta \mathbf{u}, p) - F(\mathbf{u}, p)}{\epsilon} \quad (5.14)$$

and

$$\frac{\delta F}{\delta p}(\mathbf{u}, p).(\delta p) = \lim_{\epsilon \rightarrow 0} \frac{F(\mathbf{u}, p + \epsilon \delta p) - F(\mathbf{u}, p)}{\epsilon} \quad (5.15)$$

Therefore the Gateaux derivative of (5.12) takes the form

$$F'(\mathbf{u}, p).(\delta \mathbf{u}, \delta p) = \int_{\Omega} [\mathbf{v} \cdot \rho(\delta \mathbf{u} \cdot \nabla \mathbf{u}) + \mathbf{v} \cdot \rho(\mathbf{u} \cdot \nabla \delta \mathbf{u}) + \mu \nabla \mathbf{v} \cdot \nabla \delta \mathbf{u} - q \nabla \cdot \delta \mathbf{u} - \nabla \mathbf{v} \cdot \delta p \mathbf{I}] \quad (5.16)$$

The Newton algorithm can be formulated as the following iterative process

1. Choose $(\mathbf{u}_0, p_0) \in (\mathbb{V}, Q)$
2. For $i=1 \dots n$
 - (a) Solve $F'(\mathbf{u}_i, p_i).(\mathbf{w}_i, q_i) = F(\mathbf{u}_i, p_i)$
 - (b) $(\mathbf{u}_{i+1}, p_{i+1}) = (\mathbf{u}_i, p_i) - (\mathbf{w}_i, q_i)$

Break when $\|(\mathbf{w}_i, q_i)\| < \epsilon$

For the initial guess (\mathbf{u}_0, p_0) we use the solution obtained from solving the weak form of the linear Stokes equations:

$$\int_{\Omega} [\mu \nabla \mathbf{u} : \nabla \mathbf{v} - p \nabla \cdot \mathbf{v} - q \nabla \cdot \mathbf{u}] = 0 \quad (5.17)$$

Then for each following step we solve linear problem for (\mathbf{w}_i, q_i) . In order to solve (5.17) the linear problem for each step we use finite elements method in the subspaces (\mathbb{V}_h, Q_h) , defined in the previous chapter.

5.5 Time dependent Stokes Equation

Consider the following time dependent Stokes equation:

$$\rho \frac{\partial \mathbf{u}}{\partial t} - \mu \nabla^2 \mathbf{u} + \nabla p \mathbf{I} = 0 \quad (5.18)$$

This is a simplified version of Navier-Stokes equation, neglecting $\mathbf{u} \cdot \nabla \mathbf{u}$. The time derivative can be discretized using the Crank-Nicolson scheme:

$$\frac{\rho}{\tau} (\mathbf{u}^{n+1} - \mathbf{u}^n) + \frac{1}{2} (-\mu \nabla^2 \mathbf{u}^{n+1} + \nabla p^{n+1} \mathbf{I}) + \frac{1}{2} (-\mu \nabla^2 \mathbf{u}^n + \nabla p^n \mathbf{I}) = 0 \quad (5.19)$$

The same is done for $\nabla \cdot \mathbf{u} = 0$

$$\frac{1}{2} \nabla \cdot \mathbf{u}^{n+1} + \frac{1}{2} \nabla \cdot \mathbf{u}^n = 0 \quad (5.20)$$

The next step is to derive the weak form:

$$\begin{aligned} \int \left[\frac{\rho}{\tau} (\mathbf{u}^{n+1} - \mathbf{u}^n) + \frac{1}{2} (-\mu \nabla^2 \mathbf{u}^{n+1} + \nabla p^{n+1} \mathbf{I}) + \frac{1}{2} (-\mu \nabla^2 \mathbf{u}^n + \nabla p^n \mathbf{I}) \right] \cdot \mathbf{v} d\Omega \\ - \int \left[\frac{1}{2} \nabla \cdot \mathbf{u}^{n+1} + \frac{1}{2} \nabla \cdot \mathbf{u}^n \right] \cdot q d\Omega = 0 \quad (5.21) \end{aligned}$$

Note that the following holds:

$$\int_{\Omega} \mathbf{v} \nabla^2 u^{n+1} d\Omega = \left[\int_{\Gamma} u \nabla \mathbf{v} \cdot \mathbf{v} d\Gamma - \int_{\Omega} \nabla \mathbf{u} \cdot \nabla \mathbf{v} d\Omega \right] \quad (5.22)$$

Hence applying Equation (5.22) to Equation (5.21) and grouping $n+1$ and n terms together yields:

$$\begin{aligned} & \int_{\Omega} \frac{\rho}{\tau} \mathbf{u}^{n+1} \cdot \mathbf{v} + \frac{1}{2} \mu \nabla \mathbf{u}^{n+1} \cdot \nabla \mathbf{v} - \frac{1}{2} p^{n+1} \nabla \cdot \mathbf{v} - \frac{1}{2} q \nabla \cdot \mathbf{u}^{n+1} d\Omega + \\ & \int_{\Omega} -\frac{\rho}{\tau} \mathbf{u}^n \cdot \mathbf{v} + \frac{1}{2} \mu \nabla \mathbf{u}^n \cdot \nabla \mathbf{v} - \frac{1}{2} p^n \nabla \cdot \mathbf{v} - \frac{1}{2} q \nabla \cdot \mathbf{u}^n d\Omega = 0 \end{aligned} \quad (5.23)$$

Equation (5.23) is now the final version which can be numerically solved using FreeFEM++.

5.6 Results

We would like to investigate the behaviour of the velocity magnitude and the blood pressure for stenosis and aneurysm. As stated before, we use parabolic function \mathbf{c} for the Dirichlet boundary condition at the inlet flow and its magnitude c is determined from the relation $c = \Re \mu / \rho l$, where l is the diameter of the artery. Figures 5.5 and 5.6 show plots of the velocity magnitude and pressure respectively for four different values of Reynolds numbers for domain Ω that resembles the case of stenosis (the direction of the blood flow is from left to right). By increasing the Reynolds number, the magnitude of the velocity field increases and the pressure in the region where the artery is narrow decreases. One can see from (Fig. 5.5 (c) and (d)) that regions close to the boundary after the stenosis have lower velocity magnitudes.

From the pressure plots one can conclude that higher velocities of the blood leads to bigger pressure drops and hence the affected part of the vessel can be narrowed even more. Furthermore, the magnitude of these pressure drops is not linearly dependent of the velocity of the inlet flow.

Now we simulate two different rates of stenosis. Figure 5.7 compares the magnitudes of the velocity fields for two cases: diameter of stenosis = $0.4l$ (on the left side) and diameter of stenosis = $0.3l$ (on the right side). We see that in the case where the stenosis is more narrow the velocity magnitude is greater. Again there can be seen regions after the stenosis with lesser velocity magnitudes and they are much bigger in the second case.

On (Fig. 5.8), the pressures for the cases from (Fig. 5.7) are compared in the same order. We see that in the second case the magnitudes of the pressure drops in the narrowed part are precisely two times more than in the first case. In the current study we do not model the artery as an elastic

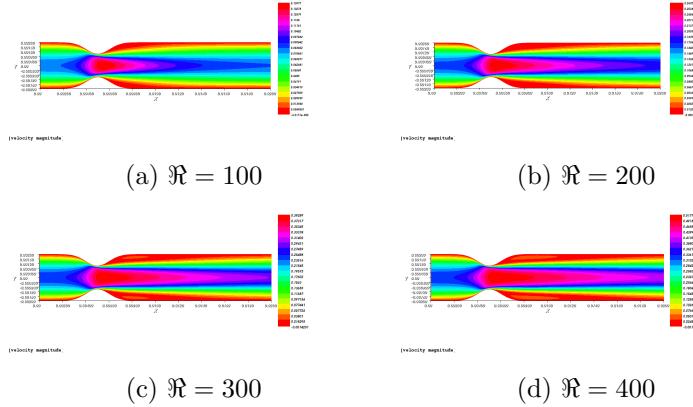


Figure 5.5: Blood flow velocity magnitude for different Reynolds numbers.

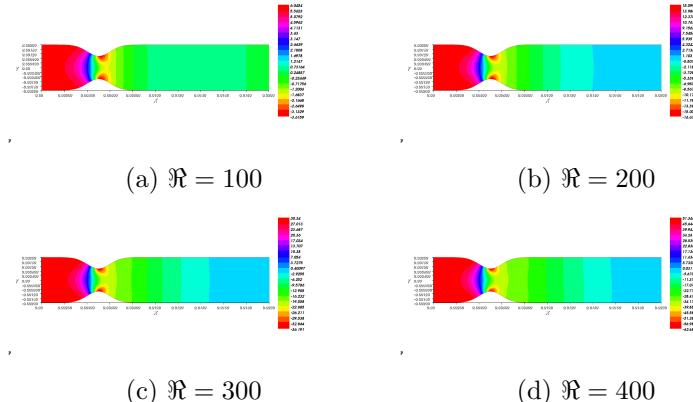


Figure 5.6: Blood flow pressure for different Reynolds numbers.

body, but we can conclude that once there is a region where the artery is narrow there are pressure drops and therefore the risk of the affected part getting even more narrow.

Let us now investigate in more detail, the behaviour of the blood flow after the stenosis. In (Fig. 5.8) the x coordinate of the velocity field \mathbf{u} for one of the previous cases is shown. We have previously seen that after the stenosis there are isolated regions with comparably smaller velocities. From (Fig. 5.8) one can see that, in these regions, the x coordinate of \mathbf{u} is negative. This means that we observe regions where the blood flows in a direction opposite to the direction of the general blood flow. This is an indication of the formation of vortices and we can conclude that in a non-ideal case turbulent behavior of the blood flow after the stenosis can be observed.

All of the previous results are obtained for the case of rotational sym-

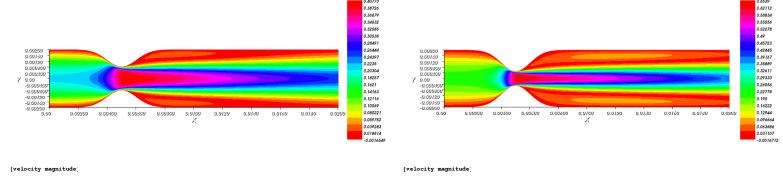


Figure 5.7: Blood flow velocity magnitude for different rates of stenosis.

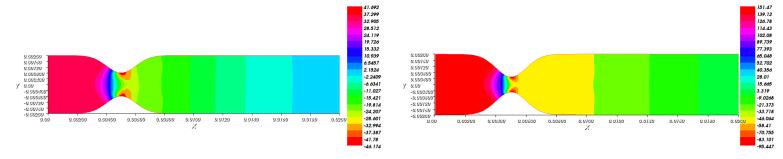


Figure 5.8: Blood flow pressure for different rates of stenosis.

metry, now let's consider the case when there is a shift between the upper and the lower narrowings. Figure 5.10 shows the blood flow velocity magnitude for a simulated non-symmetrical case of stenosis. One can see from the figure that the behaviour of the blood flow with this slight shift changes considerably. Since in reality there is always non-symmetry it is important to be able to obtain results for such cases. In all simulations for the case of stenosis the numerical algorithm based on the Newton method shows fast convergence.

Lastly, we simulate the blood flow for a domain that resembles the other pathological case we are interested in investigating - the case of aneurysm. Plot for the blood flow velocity magnitude is shown in Figure 5.11. The domain in this case consists of one boundary Γ_{In} in the lower right corner of the figure with Dirichlet boundary condition and two boundaries Γ_{Out} in the lower left and upper right corners with Neumann boundary conditions. One can see that in this domain, the blood flow has complex behaviour, however fast convergence of the numerical algorithm was again observed.

Time dependent Stokes Equation

A time varying source was placed on the lower boundary in order to simulate the pulsating blood flow into the artery. This is basically a time dependent Dirichlet boundary. The boundaries are described as follows:

$$\begin{aligned} \mathbf{n} \cdot \mathbf{T} &= 0 \text{ on } \Gamma_{Out} \\ \mathbf{u} &= A \sin^2(wt) \end{aligned}$$

The evolution of the solution can be seen in Figure 5.12

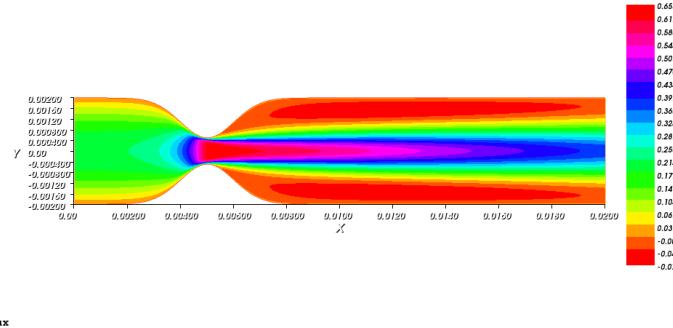
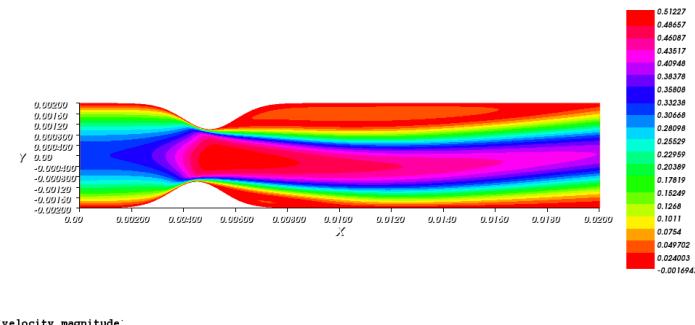
Figure 5.9: x coordinate of the velocity field.

Figure 5.10: Blood flow velocity magnitude for non-symmetric case of stenosis.

5.7 The Observation Error Minimization

Problem Motivation

In practice, patient-specific simulations of the aneurysms and stenoses can be constructed by using computational fluid dynamic techniques and image-based vascular models through segmentation of medical images from MRI (magnetic resonance imaging) or CTA (computed tomography angiography) (Fig. 5.13).

Moreover, some imaging techniques, mainly MRI-based, allow not only to obtain the geometry of the computational domain but also to measure velocity of the blood along cross-sections of the domain (Fig. 5.14). We want to use the velocity data from those extra observations to set a model and obtain a numerical simulation which can be considered reliable.

It is clear that the impact of the choice of the boundary conditions on simulation can be highly important in matters of reliability. Therefore,

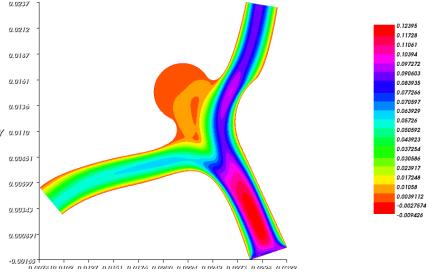


Figure 5.11: Blood flow velocity magnitude in case of aneurysm.

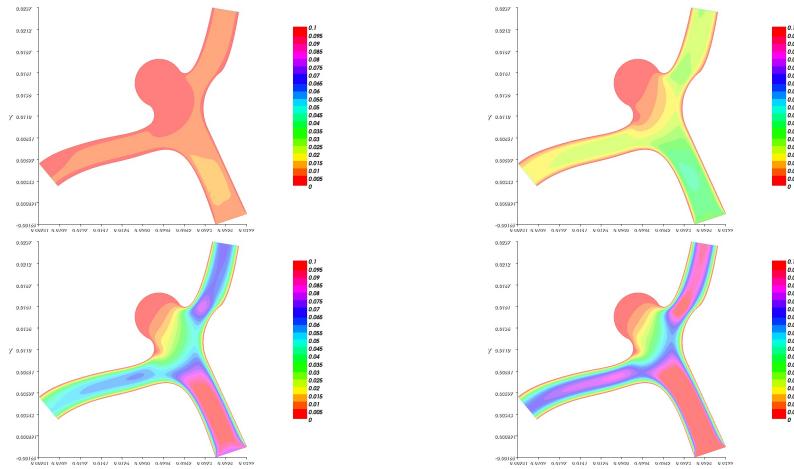


Figure 5.12: Pulsating in-flow of blood

we consider a control of the velocity values at the inflow initial Dirichlet boundary of the approximated computational domain.

The associated optimization problem aims to minimize the misfit between the given cross-sections velocity data and the computed solution, thus, reducing the uncertainty associated to the reconstructed domain.

Problem Formulation

We first simplify the problem discussed in the previous chapters from the case of Navier-Stokes equations to the case of 2D Stokes equation by cause of the limited time allotted in the context of the conference. This simplification reduces both the running time of error minimization program and implementation difficulties. However, the same theory could be applied as well to a more general model based on Navier-Stokes equations.

The weak form of the corresponding Stokes model is to find $u \in \mathbb{S}_u$ and $p \in L^2(\Omega)$, such that

$$\int_{\Omega} [\mu \nabla u : \nabla v - p \nabla \cdot v - q \nabla \cdot u] \, dx = 0 \quad (5.24)$$

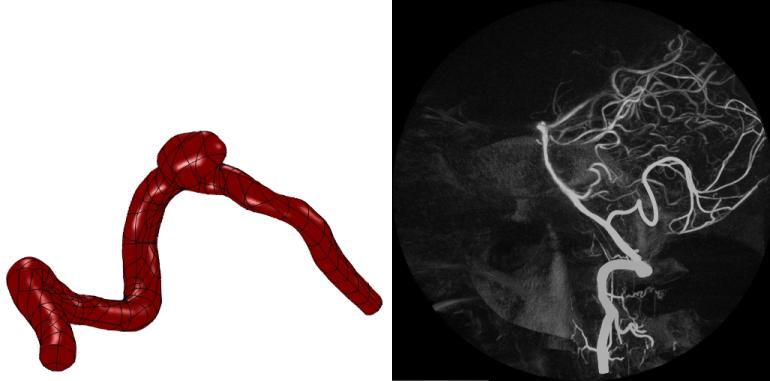


Figure 5.13: Segmentation from Medical Image. Blood through a brain artery with an aneurysm. Image from CTA.

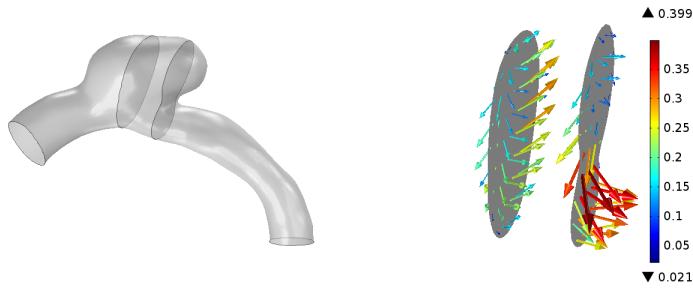


Figure 5.14: The Geometry and the velocity of the matter captured by 3D MRI.

We denote this problem (WPS).

We expect to know the velocity cross-sections data in some internal observation lines that are parallel to Γ_{In} . We denote the union of such lines as Ω_{Obs} .

Our goal is to find a vector of the initial Dirichlet boundary condition on Γ_{In} such that a given objective function J , which depends on \mathbf{u}, p is minimized.

Thus, the error minimization problem can be formulated as follows:

$$\begin{aligned} & \text{Minimize} && J(\mathbf{u}, p, \mathbf{c}) \\ & \mathbf{c} \in A_{ad} && (5.25) \\ & \text{subject to} && (WPS) \end{aligned}$$

where

\mathbf{c} is the control parameter, consisting of the unknown velocity profiles at Dirichlet boundary Γ_{In} ,

A_{ad} is the admissible function space for the control function and

J is the considered cost functional such that:

$$J(\mathbf{u}, p, \mathbf{c}) = \|\mathbf{u}(\mathbf{c}) - \mathbf{u}_{Obs}\|_{L^2(\Omega_{Obs})} + \alpha \|\nabla \mathbf{u}(\mathbf{c})\|_{L^2(\Gamma_{In})}^2 \quad (5.26)$$

The first term of the cost functional J is responsible for minimizing the misfit between the observation data and the computed solution.

The second term is a regularization term that is included in the minimization in order to ensure smoothness of the solution at the controlled boundary Γ_{In} . α is a free parameter that needs to be tuned empirically.

The reason to introduce this additional penalization information is that the problem of fitting the observation data is known to be ill-posed and generally requires some form of regularization. In this work, we use Tikhonov regularization which is essentially a trade-off between fitting the data and smoothness of the solution at the controlled boundary.

Discretization and Algorithm

To discretize the governing equations (WPS), we apply a finite element discretization. We choose a triangulation of the domain and the discretization of the functional spaces. We use linear finite elements for the velocities and the pressure and replace the functions by their approximations similarly to the case of Navier-Stokes equations. We also discretize the cost function 5.26.

The discretized optimization problem may now be written as

$$\begin{aligned} & \text{Minimize} && \hat{J}^h(\mathbf{c}) \\ & \text{subject to} && \mathbf{c} \in A_{ad} \end{aligned} \quad (5.27)$$

with

$$\hat{J}^h(\mathbf{c}) = J^h(\mathbf{u}^h(\mathbf{c}), p^h(\mathbf{c}), \mathbf{x}^h(\mathbf{c}), \mathbf{c}) \quad (5.28)$$

Here, the velocities and pressure, $\mathbf{u}^h(\mathbf{c})$ and $p^h(\mathbf{c})$, are implicit functions of the control vector $\mathbf{c} \in A_{ad} \subset \mathbb{R}^2$. These implicit functions are defined as the solution of the discretized version of the Stokes problem (WPS) with \mathbf{c} as Dirichlet boundary condition. This is referred to as the black-box or nested analysis and design (NAND) approach.

In (5.25), the velocities and pressures as well as the control vector are optimization variables. This is referred to as the all-at-once or simultaneous analysis and design (SAND) approach.

SAND formulations combined with sequential quadratic programming (SQP) methods is a very good approach, however, they require more implementation efforts than the gradient-free method CMAES that we use to solve the error minimization problem (5.27).

CMAES (Covariance Matrix Adaptation Evolution Strategy) is an a derivative free optimizer implementing an evolutionary algorithm. This algorithm works with a normal multivariate distribution in the parameters

space and try to adapt its covariance matrix using the information provided by the successive function evaluations. Although gradient-based methods are usually faster than the CMAES, we have chosen the latter because it is a convenient tool provided by FreeFem++.

On the whole, the algorithm to perform our optimization problem is given as following steps:

Algorithm 1: Blood flow modeling with observation error minimization

- 1: Get the observation data \mathbf{u}_{obs} (from a simulation).
- 2: Initialize \mathbf{c} randomly on Dirichlet boundary Γ_{In} .
- 3: While stopping criteria not satisfied:
 - (a) Solve no-stress Stokes problem (WPS) with \mathbf{c} fixed as a Dirichlet boundary condition.
 - (b) Evaluate the cost functional $J(\mathbf{c})$, measuring the misfit between the solution obtained above and observation data.
 - (c) Perform an optimization step by updating \mathbf{c} . CMAES optimizer is used in the current work.

For the purpose of simplifying implementation, we performed the error minimization analyses by assuming the considered part of cardiovascular system to be rectangular.

Observation Data

In the context of the current work, the real observation data obtained from MRI were not available, instead, we simulated the velocity data on observation lines by running the solver of the Stokes problem (WPS) with an inclined parabola as an initial Dirichlet boundary condition \mathbf{c} .

The formula for this parabola is given by:

$$u|_{\Gamma_{In}} = \left(-\frac{1}{2}(-y^2 + l^2) \cos\left(\frac{\pi}{6}\right); \quad \frac{1}{2}(-y^2 + l^2) \sin\left(\frac{\pi}{6}\right) \right)$$

with l – rectangular domain height.

The result of this simulation at observation lines (the solution $\mathbf{u}|_{\Omega(Obs)}$) was afterwards saved and referred as the observation data. This procedure provides the first step of the Algorithm (Alg. 1). The magnitude plot of the obtained simulated data is shown on the (Fig. 5.15).

Although data sets obtained by real medical imaging technique often contain noise and artifacts caused by signal degradation, those potential errors are out of the scope of the current work.

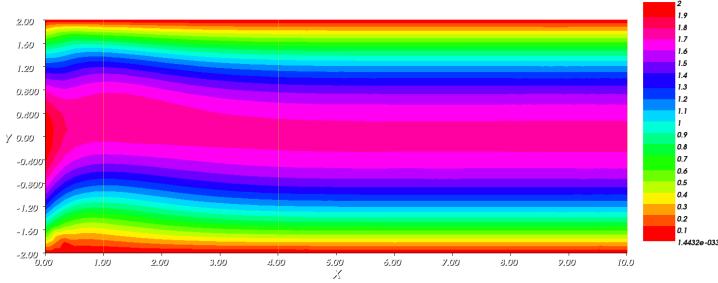


Figure 5.15: The simulated solution referred as the observation data.

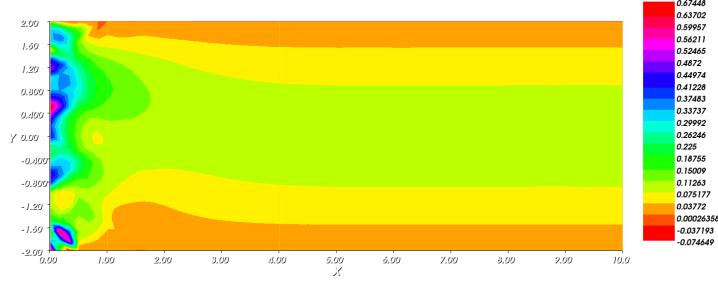


Figure 5.16: The solution obtained with an initial randomized Dirichlet boundary condition \mathbf{c} .

Optimization Procedure and Results

The optimization procedure is started by a random initialization of the Dirichlet boundary control parameter \mathbf{c} , which is the second step of the Algorithm (Alg. 1). The result of running the solver of the Stokes problem with such an initialization is shown in (Fig. 5.16).

In order to evaluate the minimal error solution obtained from a proposed optimal control algorithm, we compare this solution with a simulated data referred here as a ground truth solution \mathbf{u}_{Sim} .

For a numerical evaluation, three types of errors are introduced:

- Error over the whole domain:

$$\epsilon_{\Omega} = \frac{\|\mathbf{u}_{Sim} - \mathbf{u}\|_{L^2(\Omega)}}{\|\mathbf{u}_{Sim}\|_{L^2(\Omega)}}$$

- Error on the inflow (control) boundary

$$\epsilon_{\Gamma_{In}} = \frac{\|\mathbf{u}_{Sim} - \mathbf{u}\|_{L^2(\Gamma_{In})}}{\|\mathbf{u}_{Sim}\|_{L^2(\Gamma_{In})}}$$

- Error on the observation lines

$$\epsilon_{\Omega_{Obs}} = \frac{\sum_i \|\mathbf{u}_{Sim} - \mathbf{u}\|_{L^2(\Gamma_{Obs_i})}}{\sum_i \|\mathbf{u}_{Sim}\|_{L^2(\Gamma_{Obs_i})}}, \quad \text{where } \cup_i \Gamma_{Obs_i} = \Omega_{Obs}$$

As it was mentioned above, the free parameter α , responsible for the weight of the regularization term, is needed to be tuned empirically. The influence of the choice of α on the error values is demonstrated in the Table below (Fig. 5.17).

	ϵ_{Ω}	$\epsilon_{\Gamma_{In}}$	$\epsilon_{\Omega_{Obs}}$
$\alpha = 0$	2.94	0.125	0.005
$\alpha = 10^{-3}$	0.43	0.034	0.004
$\alpha = 10^{-2}$	0.286	0.050	0.002
$\alpha = 10^{-1}$	2.072	0.616	0.058

Figure 5.17: Error values with different regularization weight parameter α . Number of iterations 500 for all the cases. Number of fitness evaluations 7000.

In the case of small α or $\alpha = 0$ the solution at the inflow boundary is not smooth enough and, even though the error on the observation lines may be small, the error over the whole domain shows that the computed optimized solution is far away from the simulated solution. The velocity magnitude plots for $\alpha = 0$ and $\alpha = 10^{-3}$ are shown on the (Fig. 5.18) and (Fig. 5.19). Empirically obtained optimal regularization term weight is $\alpha = 10^{-2}$ which is confirmed by the smallest errors, producing the following values: $\epsilon_{\Omega} = 0.286$, $\epsilon_{\Gamma_{In}} = 0.050$, $\epsilon_{\Omega_{Obs}} = 0.002$. The velocity magnitude plots for $\alpha = 10^{-1}$ and $\alpha = 10^{-2}$ are shown on the (Fig. 5.20) and (Fig. 5.21).

5.8 Conclusions

In the current work we have presented a 2D mathematical model based on the Navier-Stokes equations for steady state blood flow inside a segment of an artery. We proposed an iterative numerical method for obtaining a solution of the non-linear problem, based on the Newton method. Furthermore we have investigated the behavior of the blood flow for simulated cases of the two most common cardiovascular diseases - aneurysm and stenosis. We have shown that the numerical method we use allows us to obtain results for non-symmetric cases resembling the case of stenosis and also for complex domain that resembles the case of aneurysm. Also we have formulated a minimization problem for the recovery of boundary conditions from given cross-sections measurements. However, due to time limitations the optimization problem was based on the simplified case of a Stokes flow. A

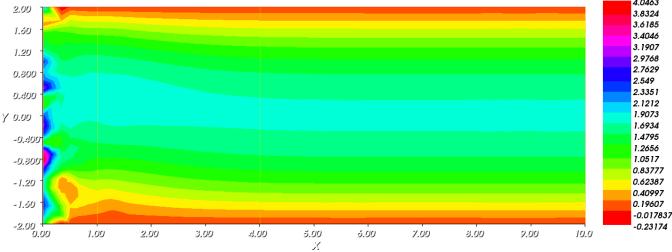


Figure 5.18: Minimization with no regularization term imposed ($\alpha = 0$).

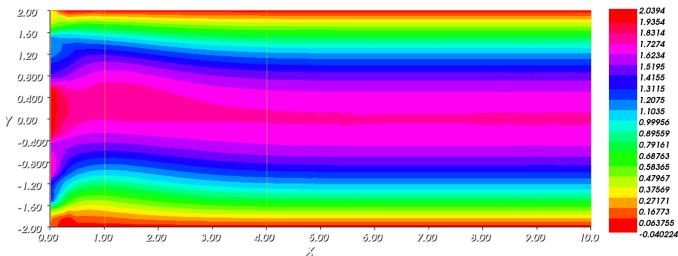


Figure 5.19: Minimization with small-weight regularization term ($\alpha = 10^{-3}$).

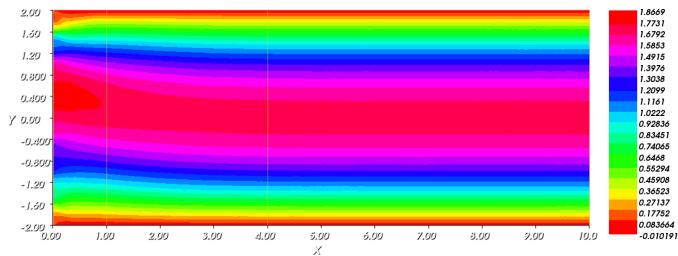


Figure 5.20: Minimization with big-weight regularization term ($\alpha = 0.1$).

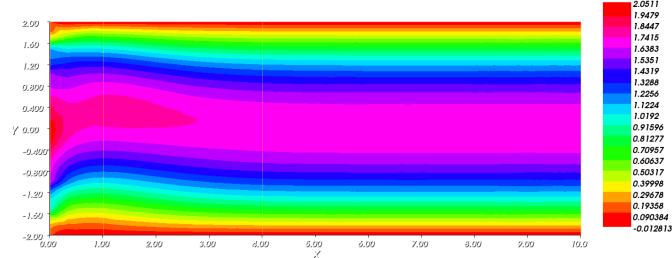


Figure 5.21: Minimization with $\alpha = 0.01$.

numerical algorithm for the minimization problem was also proposed and simulations were performed. Results from the carried simulations led us to a conclusion that the minimization algorithm allows us to recover simulated data with a sufficient error from only two cross-section measurements. For future work one should consider a 3D model for the blood flow without the assumption of the blood being a Newtonian fluid and a minimization problem for Navier-Stokes flow.

Bibliography

- [1] ABRAHAM, F AND BEHR, M. Shape optimization in steady blood flow: A numerical study of non-newtonian effects. In *Computer Methods in Biomechanics and Biomedical Engineering*, Vol.8, 127–137. 2005.
- [2] AUTHOR, U. Tikhonov regularization. https://en.wikipedia.org/wiki/Tikhonov_regularization.
- [3] FORMAGGIA, L; QUARTERONI, A AND VENEZIANI, A. *Cardiovascular Mathematics: Modeling and simulation of the circulatory system*, volume 1. Springer Science & Business Media, 2010.
- [4] HANSEN, N. The cma evolution strategy: A tutorial 2011.
- [5] HECHT, F. New development in freefem++. *J Numer Math* **20**(3-4), 251–265, 2012. ISSN 1570-2820.
- [6] TIAGO, J; GAMBARUTO, A AND SEQUEIRA, A. Patient-specific blood flow simulations: setting dirichlet boundary conditions for minimal error with respect to measured data. *Mathematical Modelling of Natural Phenomena* **9**(06), 98–116, 2014.

.1 Code

Navier-Stokes Stenosis.edp

```
1 // Stationary incompressible Navier-Stokes Equation
  with Newton method.
2
3 // build the Mesh
4 real L = 0.020, l = 0.004;
5 real l2=l/2;
6
7 real ag =0.0010 , bg=L/4 , cg=0.001, shiftparam = 0*
  L/40;
8 real numb = 40;
9
```

```

10 border gamma1(tt=0,L) { x=tt ; y=-12+ag*exp( - ( ( tt-L/2 + bg + shiftparam)^2)/(2*cg^2) ) ); label =1; }
11 border gamma4(tt=-12,12) { x=L ; y=tt; label=3; }
12 border gamma2(tt=L,0) { x=tt ; y=12-ag*exp( - ( ( tt-L/2 + bg)^2)/(2*cg^2) ) ); label=1; }
13 border gamma3(tt=12,-12) { x=0 ; y=tt; label=2; }
14
15 mesh Th=buildmesh(gamma1(numb*3)+gamma2(numb*3)+ gamma3(numb)+gamma4(numb));
16 plot(Th);
17
18
19 // macro operators
20 macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]// 
21 macro UgradV(u1,u2,v1,v2) [ [u1,u2] *[dx(v1),dy(v1)] , [u1,u2] *[dx(v2),dy(v2)] ]// 
22 macro div(u1,u2) (dx(u1)+dy(u2))// 
23
24
25 // FE Space
26 fespace Xh(Th,P2);fespace Mh(Th,P1);
27 Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
28 Mh p,q,dp,pp;
29
30 Xh psi, phi;
31
32 real c ; // velocity of in flow
33 Xh vel, vor;
34
35 // Physical properties
36 real mu = 0.0035, rho = 1060;
37 real Vmean;
38 real Re = 200;
39 c = Re*mu/(rho*1);
40
41 // Initial guess for (u1, u2) with B.C.
42 solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
43 int2d(Th)( mu*(Grad(u1,u2) *Grad(v1,v2) )
44 - div(u1,u2)*q - div(v1,v2)*p
45 )
46 + on(1,u1=0,u2=0)
47 + on(2,u1=(-c*(y^2)+c*(12^2))/(12^2),u2=0);
48
49 //plot([u1,u2] );
50
51
52 solve streamlines(psi,phi) =
53 int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))

```

```

54      +  int2d(Th)( -phi*(dy(u1)-dx(u2)))
55 //   +  on(1,psi=0)
56 //   +  on(2,psi=0)
57 //   +  on(3,psi=0)
58 ;
59
60 plot(psi,wait=1);
61
62 vel = sqrt(u1^2+u2^2);
63 plot(vel, fill=1, value=1);
64 plot([u1,u2]);
65
66
67 // stop test for Newton
68 real eps=1e-6;
69
70 verbosity = 2;
71
72 int n;
73 real err=0;
74 for( n=0;n<20;n++) // Newton Loop
75 {
76   solve Oseen([du1,du2,dp],[v1,v2,q]) =
77     int2d(Th) ( mu*(Grad(du1,du2)'*Grad(v1,v2)
78                 )
79                 + rho*UgradV(du1,du2, u1, u2)'*[v1
80                               ,v2]
81                 + rho*UgradV( u1, u2,du1,du2)'*[v1
82                               ,v2]
83                 - div(du1,du2)*q - div(v1,v2)*dp
84                 - 1e-8*dp*q // stabilization term
85                 )
86   - int2d(Th) ( mu*(Grad(u1,u2)'*Grad(v1,v2) )
87                 + rho*UgradV(u1,u2, u1, u2)'*[v1,
88                               v2]
89                 - div(u1,u2)*q - div(v1,v2)*p
90                 - 1e-8*p*q
91                 )
92   + on(1,du1=0,du2=0)
93   + on(2, du1=0, du2=0)
94 ;
95
96   u1[] -= du1[];
97   u2[] -= du2[];
98   p[] -= dp[];
99   real Lu1=u1[].linfty, Lu2 = u2[].linfty , Lp = p
100    [].linfty;
101   err= du1[].linfty/Lu1 + du2[].linfty/Lu2 + dp[] .
102    linfty/Lp;

```

```

97
98 cout << n << " err = " << err << endl;
99 if(err < eps) break; // converge
100 if( n>3 && err > 10.) {
101     cout << " not converge " << endl;
102     break;
103 } // Blowup ?
104
105 // plot([u1,u2]);
106 }
107
108 streamlines;
109 plot(cmm="streamlines", psi, wait=1);
110
111 vel = sqrt(u1^2+u2^2);
112 plot(cmm=" [velocity magnitude]", vel, fill=1, value
113 =1);
114 //plot([u1,u2]);
115 plot (cmm="p" ,p, fill=1, value=1);
116
117 vor = abs(dx(u2)-dy(u1));
118 plot (cmm="vortex" ,vor, fill=1, value=1);
119
120 Xh u1n = u1 / vel;
121 Xh u2n = u2 / vel;
122 plot(cmm=" [u1n,u2n]" , [u1n,u2n]);

```

Time-dependent Stokes equation.edp

```

1 border f1(t=8.277579470048165,3.122414148982325){x=
  ;y=0.002358371815112*t^5 -
2 0.065655720087747*t^4 + 0.729627361800546*t^3 -
  4.08197654392561*t^2 + 11.856566776041069*t -
3 11.709295757087336; label=1;}
4 border f2(t=0.859452741468984,-0.981740566055334){x=
  (0.341400771151288^(1./2.))*cos(t) +
  7.952959033840158; y=
5 (0.341400771151288^(1./2.))*sin(t)
  +4.461148557368587; label=1;}
6 border f3(t=0.529128582979014,2*pi
  -1.952566680840888){x=((1.281091616344505)^(1./2.))
  )*cos(t)+8.756105018206105; y =
7 ((1.281091616344505)^(1./2.))*sin(t) +
  5.954110697006632; label=1;}
8 border f5(t=12.880766163863337,11.008092762240402){x
  =t;y= -2.1851851851852*t + 27.97381491146139; label

```

```

=1; }

9 border f6(t=4,11.41942696370471){x=t;y
=0.000141915629507*t^7 - 0.007078659134381*t^6 +
0.14697826936389*t^5 -
10 1.653476604852379*t^4 + 10.936743462092085*t^3 -
42.81088073494685*t^2 +
11 92.93661032718971*t - 86.13879673351471; label=1;}
12 border f7(t=11.265838454747,9.733174187725922){x=t;
y= 1.863904540127596*t^2 -
13 37.203740366653165*t + 192.0595346933998; label=1;}
14 border f8(t=11.688120335844724,12.5) {x=t; y=
0.818080357142857*t^3 - 26.75577731092437*t^2 +
15 291.4957983193277*t - 1051.6302521008404; label=1;}
16 border f9(t=2*pi
-2.739677193250460,2.225248502466470) {x=
(4.763099785545185^(1./2.))*cos(t) +
17 13.016633203294978; y= (4.763099785545185^(1./2.))*sin(t)+4.772829646050528; label=1;}
18 border g1(t=3.122414148982325,4){x=t;y
=-1.200267257468735*t + 5.932898265695426;}
19 border g2(t=11.41942696370471,12.880766163863337){x=
t;y = 0.332421039275865*t - 4.454882158508418;label
=2;}
20 border g4(t=12.5, 11.265838454747){x=t; y=
-0.164567526760309*t + 11.347313687969192;}
21
22
23 plot(f1(50)+f2(50)+f3(50)+f5(50)+f6(50)+f7(50)+f8
(50)+g1(50)+g2(50)+g4(50)+f9(50));
24
25
26 mesh Th1=buildmesh(f1(50)+f2(10)+f3(50)+f5(30)+f6
(70)+f7(30)+f8(35)+f9(25)+g1(20)+g2(20)+g4(20));
27
28 mesh Th=movemesh(Th1,[0.001*2.5*x,0.001*2.5*y]);
29
30 plot(Th);
31
32
33 //macro operators
34 macro Grad(u1,u2) [dx(u1),dy(u1), dx(u2),dy(u2)]//
35 macro UgradV(u1,u2,v1,v2) [ [u1,u2]'*[dx(v1),dy(v1)]
, [u1,u2]'*[dx(v2),dy(v2)] ]//
36 macro div(u1,u2) (dx(u1)+dy(u2))//
37
38 //macro Gradn(u1n,u2n) [ dx(u1n),dy(u1n), dx(u2n),dy
(u2n)]//
39 //macro UgradVn(u1n,u2n,v1,v2) [ [u1n,u2n]'*[dx(v1),
dy(v1)] , [u1n,u2n]'*[dx(v2),dy(v2)] ]//

```

```

40 //macro divn(u1n,u2n) (dx(u1n)+dy(u2n))//
41
42 // FE Space
43 fespace Xh(Th,P2);fespace Mh(Th,P1);
44 Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
45 Mh p,q,dp,pp;
46
47 Xh vel;
48
49 real c; // velocity of in flow
50
51 //real tau=0.01; //time
52
53 // Physical properties
54 real mu = 0.0035, rho = 1060;
55 real Vmean;
56 real Re = 100;
57 real Lc=0.003425;
58 c = Re*mu/(rho*Lc);
59 cout << c << endl;
60
61 real t;
62 real T=2;
63 real n=200;
64 real tau=T/n;
65
66 real[int] viso(2);
67 for(int ii=0;ii<viso.n;ii++)
68   viso[ii]=ii*0.1;
69
70
71 Xh u1o=0,u2o=0,po=0,u1plot,u2plot;
72 problem Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK)
    =
73   int2d(Th)( (rho/tau)*[u1,u2] *[v1,v2]+0.5*( mu*( 
    Grad(u1,u2) *Grad(v1,v2) )
74           - div(u1,u2)*q - div(v1,v2)*p))+ 
75   int2d(Th)( -(rho/tau)*[u1o,u2o] *[v1,v2]
76           +0.5*( mu*(Grad(u1o,u2o) *Grad(v1,v2) )
77           - div(u1o,u2o)*q - div(v1,v2)*po)
    )
78
79   + on(1,u1=0,u2=0)
80 //+ on(2,u1=0,u2=0);
81 //+ on(2,u1=-c*(N.x),u2=-c*(N.y));
82 + on(2,u1=-c*(N.x)*sin(10*t)*sin(10*t),u2=-c*(N.y)
    *sin(10*t)*sin(10*t));
83
84

```

```

85 t=0;
86 for (int iii = 0; iii < n; iii++) {
87     t+=tau;
88     cout << "t" << t << endl;
89
90     u1o=u1;
91     u2o=u2;
92     po=p;
93
94     Stokes;
95     vel = sqrt(u1^2+u2^2);
96
97     u1plot=u1/70;
98     u2plot=u2/70;
99
100
101    //plot([u1plot,u2plot],ps="Magnetude"+iii+".jpg");
102    plot([u1plot,u2plot]);
103    //plot(vel, fill=1, value=1,ps="Plot"+iii+".jpg",
104    //      viso=viso(0:viso.n-1));
105    //plot(vel,fill=1,value=1,ps="Plot.jpg", viso=viso
106    //      (0:viso.n-1));
107 }

```

Optimization.edp

```

1 // Observation Error Minimization for Stationnary
   imcompressible Stokes Equation with Newton method.
2
3 // -----
4 // I. Load modules:
5 // -----
6
7 load "ff-cmaes"
8
9 // -----
10 // II. Global variables:
11 // -----
12
13 // Mesh properties
14 real L = 10., l = 4.;
15 real l2=l/2;
16 mesh Th;
17
18 // FE Space
19 fespace Xh(Th,P2);
20 fespace Mh(Th,P1);

```

```

21 Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
22 Mh p,q,dp,pp;
23 Xh boundaryC1,boundaryC2;
24
25 // Physical properties
26 real mu = 0.8, rho = 0.5;
27 real Vmean = 1.;
28 real Re = 100;
29
30 // Simulation and Observation properties and
31 // variables (for optimization part)
31 real c = 0.5; // velocity of in flow
32 real alpha = 0.01; // weight of the regularization
33 // term
33 int numObs=2;
34
35 // Arrays numNodesInflowBoundary, nodesObsLines, etc
36 // . don't have a lot of physical/mathematical sence,
36 // they are introduced only to read/compare
37 // observations at the observation lines
37 int numNodesInflowBoundary; // number of dots
38 // on the inflow boundary line
38 int[int] nodesInflowBoundary(0); // to get indices
39 // of inflow boundary dofs
39 int[int] numNodesObsLines(numObs); // (cnumber)
40 // number of dots on the observation lines
40 int[int,int] nodesObsLines(numObs, 0); // (cMatrix)
41 // to get indices of observation lines dofs
41
42 Xh u1Sim, u2Sim; // stores the data from observation
43 // lines
43 real [int, int] uObsMat1 (numObs,2000); // another
44 // format / stores the data from observation lines (x
44 // component)
44 real [int, int] uObsMat2 (numObs,2000); // /
45 // another format / stores the data from observation
45 // lines (y component)
46
46 real[int] control1, control2;
47
48 // Plotting properties
49 int itNum = 0;
50 int plotEvery = 5000; // intermediate plots
51
52 // -----
53 // III. Mesh
54 // -----
55
56 // build the Mesh

```

```

57 border gamma1(tt=0,L) { x=tt ; y=-1/2; label=1; }
58 border gamma4(tt=-12,12) { x=L ; y=tt; label=3; }
59 border gamma2(tt=L,0) { x=tt ; y=1/2; label=1; }
60 border gamma3(tt=12,-12) { x=0 ; y=tt; label=2; }
61 border gamma6(tt=12,-12){ x=4 ; y=tt; label=5; }
62 border gamma5(tt=12,-12){ x=1 ; y=tt; label=4; }
63
64 int[int] labels = [2,4,5];
65
66 Th=buildmesh(gamma1(L*2)+gamma2(L*2)+gamma3(1*2)+  

67   gamma4(1*2)+ gamma5(1*2)+gamma6(1*2));
68 plot(Th);
69 // -----
70
71 Xh[int] xh (2);
72 xh[0] = x;
73 xh[1] = y;
74 int n = Xh.ndof; // velocity dfom
75
76 // -----
77 // IV. Auxilary functions
78 // (GO TO to the next chapter to skip the details  

79 // of implementation)
80 // -----
81 // -----
82 // IV.a Standard macro operators
83 // -----
84
85 macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]//  

86 macro UgradV(u1,u2,v1,v2) [ [u1,u2]'*[dx(v1),dy(v1)]  

87   , [u1,u2]'*[dx(v2),dy(v2)] ]//  

88 macro div(u1,u2) (dx(u1)+dy(u2))//  

89 // -----
90 // IV.b Functions and macros needed
91 // -----
92
93 // the function return the indeces of nodes located
94 // at the border with a label 'label'
95 func int getNodesOnLine(int iLine)
96 {
97   int label = labels(iLine);
98
99   // to generate a vector with 1 on boundary dofs
100  varf von(u,v)= on(label,u=1);
101  real[int] on1=von(0,Xh, tgv=1); // one on boundary
      dofs

```

```

102
103     int numNodesOnBorder = on1.sum; // number of nodes
104     on the line 'label'
105     int[int] nodesOnBorder(numNodesOnBorder);
106
107     int boundaryCounter = 0;
108     for (int i=0;i<n;++i)
109     {
110         if( on1[i])
111         {
112             nodesOnBorder[boundaryCounter]=i;
113             boundaryCounter++;
114         }
115     }
116
117     cout << "# nodes " << Xh.ndof << " ; "
118     << "# nodes on the border " << numNodesOnBorder
119     << endl;
120     cout << "Indeices and positions of nodes at the
121     observation line "
122     << iLine << ":" << endl;
123     for (int n = 0; n < numNodesOnBorder; n++) {
124         cout << nodesOnBorder[n];
125         for (int d = 0; d < 2; d++)
126             cout << " " << xh[d] [] [nodesOnBorder[n]]; //  

127             x, y
128         cout << endl;
129     }
130
131     // -----
132     // save the results to global arrays
133
134     // Inflow boundary
135     if (iLine == 0) {
136         nodesInflowBoundary.resize(numNodesOnBorder);
137         nodesInflowBoundary = nodesOnBorder;
138         numNodesInflowBoundary = numNodesOnBorder;
139     }
140
141     // Observation lines
142     if (iLine >= 1) {
143         if (nodesObsLines.m < numNodesOnBorder)
144             nodesObsLines.resize (numObs, numNodesOnBorder
145             );
146         nodesObsLines(iLine-1,:) = nodesOnBorder;
147         numNodesObsLines(iLine-1) = numNodesOnBorder;
148     }
149
150     return 0;

```

```

146 }
147
148 // solve Stokes equation
149 macro solvePDE(control)
150 {
151   convertControlToBoundary(control); /*changes
152   boundaryC1, boundaryC2*/
153
154 /* Initial guess for (u1, u2) with B.C.*/
155 solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK)
156 =
157   int2d(Th)(
158     (1/Re)*(Grad(u1,u2)''*Grad(v1,v2) )
159     - div(u1,u2)*q - div(v1,v2)*p
160     )
161   + on(1,u1=0,u2=0)
162   + on(2,u1=boundaryC1,u2=boundaryC2);
163
164 /* change u1 and u2 on control */
165 }[u1,u2]//
166
167 // the function convert a vector control into
168 // compatible for the current program format of
169 // boundary
170 macro convertControlToBoundary(control)
171 {
172   splitControlVectorIntoTwoVectors(control, control1
173   , control2);
174   for (int n = 0; n < numNodesInflowBoundary; n++) {
175     boundaryC1 [] [nodesInflowBoundary [n]] =
176       control1 [n];
177     boundaryC2 [] [nodesInflowBoundary [n]] =
178       control2 [n];
179   }
180 }[boundaryC1, boundaryC2]//
181
182 // auxiliary function for convertControlToBoundary(
183 // control)
184 func int splitControlVectorIntoTwoVectors(
185   /* input */ real[int] & control,
186   /* output */ real[int] & control1, real[int] &
187   control2)
188 {
189   int obsLength = control.n/2;
190   control1.resize(obsLength);
191   control2.resize(obsLength);
192
193   for (int i = 0; i < obsLength; i++)
194   {

```

```

187     control1[i] = control[i];
188     control2[i] = control[i + obsLength];
189 }
190
191     return 0;
192 }
193
194 // plot the current solution u
195 func int plotSolution()
196 {
197     Xh mag=sqrt(u1^2+u2^2); // for a magnitude plot
198     plot([u1,u2],value=1);
199     plot(mag,value=1, fill=1);
200     return 0;
201 }
202
203 // run the simulation with an inclined parabola at
204 // the inflow border
205 func int runSimulation() {
206
207     real[int] cInitX(numNodesInflowBoundary);
208     real[int] cInitY(numNodesInflowBoundary);
209     real[int] cInit2(2*numNodesInflowBoundary);
210
211     real xx, yy;
212     for (int n = 0; n < numNodesInflowBoundary; n++) {
213         yy = xh[1][][nodesInflowBoundary[n]];
214         //xx = xh[0][][nodesInflowBoundary[n]];
215
216         cInitX(n) = (-c*(yy^2)+c*(12^2))*cos(-pi/6);
217         cInitY(n) = -(-c*(yy^2)+c*(12^2))*sin(-pi/6)
218         ;
219
220         cInit2(n) = cInitX(n);
221         cInit2(n+numNodesInflowBoundary) = cInitY(n);
222     }
223
224     solvePDE(cInit2);
225     plotSolution();
226
227
228 // save the results of simulation as observation
229 func int getObservationsFromSimulation()
230 {
231     real yy;
232     for (int k = 0; k<numObs; k++){
233         for (int j = 0; j < numNodesObsLines[k]; j++) {

```

```

234         u0bsMat1(k,j) = u1 [] [nodes0bsLines(k,j)];
235         u0bsMat2(k,j) = u2 [] [nodes0bsLines(k,j)];
236     }
237 }
238
239 u1Sim = u1;
240 u2Sim = u2;
241 return 0;
242 } // [u1Sim, u2Sim, u0bsMat1, u0bsMat2]
243
244 // compute errors on the whole domain, inflow and on
245 // observation lines
246 func real[int] computeError() {
247     real[int] error(3);
248
249     error(0) = int2d(Th)
250         (
251             sqrt(
252                 ( u1Sim - u1 )^2
253                 + ( u2Sim - u2 )^2
254             )
255         /
256             sqrt(
257                 ( u1Sim )^2
258                 + ( u2Sim )^2
259             )
260         );
261
262     error(1) = int1d(Th, gamma3)
263         (
264             sqrt(
265                 ( u1Sim - u1 )^2
266                 + ( u2Sim - u2 )^2
267             )
268         /
269             sqrt(
270                 ( u1Sim )^2
271                 + ( u2Sim )^2
272             )
273         );
274
275     error(2) = int1d(Th, gamma5)
276         (
277             sqrt(
278                 ( u1Sim - u1 )^2
279                 + ( u2Sim - u2 )^2
280             )
281         /
282             sqrt(

```

```

282             ( u1Sim )^2
283             + ( u2Sim )^2
284         )
285     )
286     + int1d(Th, gamma6)
287     (
288     sqrt(
289         ( u1Sim - u1 )^2
290         + ( u2Sim - u2 )^2
291     )
292     /
293     sqrt(
294         ( u1Sim )^2
295         + ( u2Sim )^2
296     )
297 );
298     return error;
299 }
300
301 func real[int] setInitialControlToZero() {
302     itNum = 0;
303     real[int] cInit(2*numNodesInflowBoundary);
304     for (int n = 0; n < numNodesInflowBoundary; n++) {
305         cInit(n) = 0;
306         cInit(n+numNodesInflowBoundary) = 0;
307     }
308     return cInit;
309 }
310
311 // returns J = ||u_Obs - u|| + alpha * || grad(u) ||_c
312 // ||^2
313 func real evaluateCostFunctionalJ() {
314     real cost=0;
315
316     Xh du1x = dx(u1);
317     Xh du1y = dy(u1);
318     Xh du2x = dx(u2);
319     Xh du2y = dy(u2);
320
321     for (int k = 0; k < numObs; k++)
322     {
323         for (int j=0;j<numNodesObsLines[k];j++)
324         {
325             cost +=
326             abs(
327                 sqrt(
328                     (uObsMat1(k,j) - u1[] [nodesObsLines(k,j)])
329                     ^2
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
25
```

```

329         + (uObsMat2(k, j) - u2[] [nodesObsLines(k, j)])
330         ^2
331     )
332     + alpha *
333     abs(
334         (du1x [] [nodesInflowBoundary[j]])^2
335         + (du1y [] [nodesInflowBoundary[j]])^2
336         +
337         (du2x [] [nodesInflowBoundary[j]])^2
338         + (du2y [] [nodesInflowBoundary[j]])^2
339     )
340     ;
341 }
342 }
343
344     return cost;
345 }
346
347 // -----
348 // V. Cost Function
349 // -----
350
351 func real J(real[int] & control)
352 {
353     // -----
354
355     // solves PDE (Stokes here) with the inflow
356     // boundary condition 'control'
357     solvePDE(control);
358     // evaluate the cost functional J using
359     //     * the solution obtained by solvePDE(control)
360     //     ) above
361     //     and * the observation data (obtained from a
362     //         simulation here)
363     real cost = evaluateCostFunctionalJ();
364
365     // -----
366     // print and plot:
367
368     cout << "J =" << cost << " control =" << control1
369     (0) << " " << control1(1) << "...\\n" ;
370     if (itNum % plotEvery == 0) plotSolution();
371     itNum++;
372
373     // -----
374
375     return cost;
376 }

```

```

373
374 // -----
375 // VI. Main program
376 // -----
377
378 // -----
379 // VI.0) Get indeces of nodes at observation lines
380 // and the inflow border.
381 // It is a technical solution with no
382 // mathematical sence in it.
383
384 for (int i=0 ; i<=numObs; i++){
385     getNodesOnLine(i);
386 }
387
388 // -----
389 // VI.1) SIMULATION
390
391     runSimulation();
392     getObservationsFromSimulation();
393
394 // -----
395 // VI.2) OPTIMIZATION
396
397     // We start from cInit (the control, initial flow)
398     // being a random vector around 0
399     real[int] cInit = setInitialControlToZero();
400     // The function cmaes performs optimization given
401     // * a cost function J (solvePDE with cInit +
402     //   compute the cost functional)
403     // and * a control vector cInit
404     real min = cmaes(J,cInit,stopTolFun=1e-3,
405                      stopMaxIter=500,seed=1);
406
407 // -----
408 // VI.3) RESULTS
409
410     cout << "minimal value is J = " << min << endl;
411     plotSolution();
412     real[int] error = computeError();
413     cout << "Error whole domain: " << error(0) << endl
414     ;
415     cout << "Error inflow boundary: " << error(1) <<
416     endl;
417     cout << "Error observation lines: " << error(2) <<
418     endl;

```