Bachelor's Degree Thesis

Modelling the Cardiovascular System: Pulsating Flow Through a Collapsible Channel

Author:
Xavier Cabanes Bosacoma*

Thesis advisors:
Marc Àvila†

Degree in Physical Engineering
Escola Tècnica Superior d'Enginyeria de Telecomunicacions de Barcelona
Universitat Politècnica de Catalunya

*Centre de Formacio Interdisciplinària Superior (CFIS) from Universitat Politècnica de Catalunya (UPC)
†Zentrum für Angewandte Raumfahrttechnologie und Mikrogravitation (ZARM)

A totes les Bosacoma i Cortada

Abstract

The interaction between fluid flows and the deformation of solid structures is important in many engineering applications, as in turbochargers, wind turbines or airplane wings. The flow of blood in the body is also an example involving very complex fluid-structure interactions. Here the flow rate pulsates cyclically, the fluid is non-Newtonian and the precise material properties are difficult to measure accurately. Hence the development of appropriate simplified models and experiments is necessary in order to determine the importance of each of these ingredients and their nature.

In this project we will be dealing with a basic model for the behaviour of medium sized blood vessels consisting of a partially elastic, partially rigid channel with a Poiseuille flow as initial condition. With this model we try to simulate real life cases and try to obtain results that could bring to better understanding of the circulatory system, while also covering the basis of solid mechanics, fluid mechanics and Partial differential equations solving methods necessary to fully understand the purpose of this project.

Keywords: Lagrangian Mapping, Finite Elements Method, Newtonian Fluid, Large blood vessels, Pulsating flow, Driven Damped Oscillator. Oomph-lib, Fluid Structure Interaction.

Contents

1 Introduction

1.1 Motivation

The introduction of new numerical methods as well as the development of new experimental measuring techniques that produce large amount of data retrieved by biologists, chemists and other scientists has induced a growing interest in interpreting and understanding of many biological systems. These results allow mathematicians, engineers and physicians to verify the correctness of their models, in addition to giving important measurements to describe their behaviour.

For my part, I wanted to work on a project that involved the characterisation of a biological system but using the concepts I learned in both numerical calculus and fluid mechanics courses. Thus, I took advantage of the opportunity of being part of one of these projects with prof Marc Avila at the Center for Applied Space Tecnology in Bremen.

More specifically, the modelling of the behaviour of the circulatory system is of high interest since cardiovascular disease is the leading cause of death in Europe [1]. It is crucial then to invest in the development and improvement of the existing models in order to fully understand the functioning of the system, which would turn into the ability of preventing and treating heart diseases.

1.2 Objectives

The aim of this project is to analyze the response of a pulsating flow through a channel with partially elastic, partially rigid walls in order to retrieve significant biological data. Both the transient and steady solutions want to be described, as well as the different phenomena that arise from the different values of the governing parameters.

In order to do so, an introduction to the theoretical background is presented, so the reader can fully understand the faced problem. After that, the problem to be faced is presented as well as all of its features and conditions. Then, the results are obtained, analyzed and discussed. Finally, the conclusions from the analysis of the response of our system are presented and also some possible future work in the same research line.

Another personal objective set for this project is to get a first contact with the modelling of biological systems through mathematical and physical tools, as well as being able to properly analyze and present the results in a complete thesis.

1.3 State of the Art

Due to the importance of the topic, many researchers are carrying out studies on the behaviour of some cardiovascular pathologies. The most investigated are the influence of the fluid shear stress on Fluid Structure Interaction problems, mainly with blood vessels

presenting aneurysms. Aneurysms are balloon shaped bulges of the vessels that can appear at any point of the circulatory system, but the most known and lethal in the brain, thoracic aorta and abdominal aorta. Other common studied pathologies are atherosclerosis and thrombosis.

In [2] a comparison between the behaviour of a cerebral aneurysm with high and normal blood pressure is made. The authors reconstructed a realistic geometry taken from Computer Tomography of a 59 year old female patient and studied the effects of a pulsating pressure and velocity on the bleb of the aneurysm, which is a small saccular aneurysm on the aneurysmal wall whose existance is one of the known indicators in rupture-risk estimation of the aneurysm. Results shown that the stress from high pressure where two times those obtained using normal pressure.

For other pathologies like stenosis, it has been shown that critical mechanical conditions for the wall are very sensitive to the geometry rather than the wall mechanical characteristics[3].

More tools for patient-specific models have been developed in [4] in order to study the effect of the fluid shear stress on the blood vessels. Others used another approach for the modelling of the vessel wall, adding more than one layer with different elastic parameters for a model of the aortic inlet [5].

Studies are done not only to describe pathologies, but also to understand other cardiovascular phenomena like the behaviour of the heart valves. In [6], a 2D model of a heart valve is solved using a Finite Elements Method by considering the effect of the flow on an elastic leaflet.

2 Foundations

In this chapter, the basic notions necessary to understand the description of our problem and code are presented divided in two parts: the first one corresponds to the derivation of the fluid equations as well as the definition of some necessary Continuum Medium Mechanics concepts necessary to understand the proposal of the faced problem.

The second section covers the basic notions on the numerical method used to solve the governing equations: the Finite Elements Method. Only the general scheme of the method is presented, in order for the reader to get a glimpse of numerical solving for Partial Derivatives Equations.

2.1 Continuum Medium Mechanics

In order to understand the derivation of the equations that govern the motion of our media, some kinematic concepts and properties must be introduced. The Continuum Medium Mechanics studies the motion of a medium that continuously occupies a portion of space at each time; we will call this "portion of space" our domain of the problem, and we will be able to deal with the motion of the media through the use of standard methods of analysis.

This section covers exhaustively the concepts necessary to infer the very well known Navier Stokes Equations (NSE). The structure and length of this section is chosen to be such that covers the concepts from the very basics until the final equations are obtained. Most of the information of it comes from [7], which I found very interesting and even more useful for the development of this project, as it faces the theory behind Continuum Medium Mechanics in a very mathematical way.

2.1.1 The Lagrangian and Eulerian frames

Let Ω_0 be a domain (i.e. an open, bounded and connected set) in R^d which we will call the "reference" configuration. The motion of the particles in the media through a time period $I = (t_0, t_1)$ will be described by a set of diffeomorphisms (differentiable, invertible and with differentiable inverse) $\mathcal{L}_t$ that will relate our reference configuration Ω_0 with the configuration Ω_t at a current time $t \in I$.

These mappings associate the position of a "labelled" particle $\boldsymbol{\xi} \in \Omega_0$ in the reference configuration to the position of that particle $\mathbf{x} \in \Omega_t$ at time $t \in I$, that is

$$\mathcal{L}_t : \Omega_0 \to \Omega_t, \quad \boldsymbol{\xi} = \mathbf{x}(t, \boldsymbol{\xi}) = \mathcal{L}_t(\boldsymbol{\xi}),$$

which are called the "Eulerian" and the "Lagrangian" variables, respectively.

Since the Lagrangian mapping is surjective, a field f associated with the media may be described as a function of the Lagrangian or the Eulerian variables. In order to know where is this function defined, we will follow the next rule: we will use the same letter

or symbol for both of the configurations but we will mark the function defined in the Lagrangian frame with the tilde sign, $\sim$. Thus, if we have $f : I \times \Omega_t \to \mathsf{R}$, then the following equality holds:

$$f(\mathbf{x}, t) = f(\mathcal{L}_t(\boldsymbol{\xi}), t) = \tilde{f}(\boldsymbol{\xi}, t)$$

.

A similar approach will be used for the spatial differential operators that use the nabla sign. We will use ∇ when we refer to the partial derivatives with respect to the Eulerian variable $\mathbf{x}$, while for the Lagrangian ones we will use $\nabla_{\boldsymbol{\xi}}$. That is

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right), \qquad \nabla_{\boldsymbol{\xi}} \tilde{f} = \left(\frac{\partial \tilde{f}}{\partial \xi_1}, \frac{\partial \tilde{f}}{\partial \xi_2}, \frac{\partial \tilde{f}}{\partial \xi_3} \right)$$

and the same applies for the other spatial differential operators, such as the divergence, the Laplacian, etc.

Furthermore, we will use the usual symbol for the composition even if it only applies to the spatial variables. That is, the composition of a field $f : I \times \Omega_t \to \mathsf{R}$ with the Lagrangian mapping is to be understood as

$$f \circ \mathcal{L}_t(t, \boldsymbol{\xi}) = f(t, \mathcal{L}_t(\boldsymbol{\xi})) = f(t, \mathbf{x}(t, \boldsymbol{\xi}))$$

, and the same applies for its inverse.

2.1.2 Basic quantities and properties

Velocity

When facing fluid mechanics problems, one of the quantities that plays a central role is the fluid velocity. In the Lagrangian frame, the velocity field $\tilde{\mathbf{u}}(t, \boldsymbol{\xi})$ is defined as

$$\tilde{\mathbf{u}}(t, \boldsymbol{\xi}) = \frac{\partial}{\partial t} \mathbf{x}(t, \boldsymbol{\xi}) \tag{2.1}$$

and it can be understood as the time derivative of the trajectory $T_\xi = \{(t, \mathbf{x}(t, \boldsymbol{\xi})),\ t \in I\}$ of the fluid particle labelled $\boldsymbol{\xi}$.

In the Eulerian frame, the velocity $\mathbf{u}$ is defined in $I \times \Omega_t$ as $\mathbf{u} = \tilde{\mathbf{u}} \circ \mathcal{L}_t^{-1}$, that is

$$\mathbf{u}(t, \mathbf{x}) = \tilde{\mathbf{u}}(t, \boldsymbol{\xi}), \quad \boldsymbol{\xi} = \mathcal{L}_t^{-1}(\mathbf{x})$$

.

The material derivative

Since our domain is continuously moving, and the rate of the movement of the particles is described by the velocity of the fluid, the velocity plays an important role when taking into account the variation of quantities in our domain with respect to time. For that, let us define the "material derivative" of a function $f : I \times \Omega_t \to \mathsf{R}$, commonly denoted by $\frac{Df}{Dt}$, as the derivative in the Lagrangian frame expressed as function of the Eulerian variables, that is

$$\frac{Df}{Dt} : I \times \Omega_t \to \mathsf{R}, \quad \frac{Df}{Dt}(t, \mathbf{x}) = \frac{\partial \tilde{f}}{\partial t}(t, \boldsymbol{\xi}), \tag{2.2}$$

where

$$\boldsymbol{\xi} = \mathcal{L}_t^{-1}(\mathbf{x}).$$

This derivative can be understood as the rate of variation of the quantity f along the trajectory of the particle $\boldsymbol{\xi}$. By the chain rule of derivation of composed functions,

$$\frac{Df}{Dt} = \left[\frac{\partial}{\partial t}(f \circ \mathcal{L}_t)\right] \circ \mathcal{L}_t^{-1} = \left[\frac{\partial f}{\partial t} \circ \mathcal{L}_t + \nabla f \cdot \frac{\partial x}{\partial t}\right] \circ \mathcal{L}_t^{-1} = \frac{\partial f}{\partial t} + \nabla f \cdot \frac{\partial x}{\partial t} \circ \mathcal{L}_t^{-1}$$

and recalling the definition of the velocity in the Lagrangian frame (Equation (2.1)), we finally get the form of the material derivative of a scalar function f, that is

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + \nabla f \cdot \mathbf{u} \tag{2.3}$$

Note that a quantity that satisfies $\frac{\partial f}{\partial t}$ (also-called "stationary") does not have to be constant along the trajectory, because of the term related with the velocity and the gradient of f. For example, imagine a 2D room where each point of the floor is set to have a constant temperature $T(\mathbf{x})$ with time and a person moving around the room. The variation of the temperature with respect to the time is zero, but the variation of temperature along the trajectory of that person does not have to be necessary zero.

Acceleration

Another key quantity in fluid mechanics is the "acceleration". In the Lagrangian frame, the acceleration is the field $\tilde{\mathbf{a}} : I \times \Omega_0 \to \mathsf{R}^3$ defined as the rate of variation of the velocity with respect to the time. That is

$$\tilde{\mathbf{a}} = \frac{\partial \tilde{\mathbf{u}}}{\partial t} = \frac{\partial^2 \mathbf{x}}{\partial t^2}$$

and for the Eulerian frame, recalling Equations (2.2) and (2.3) for each of the components of the acceleration

$$a_i = \frac{Du_i}{Dt} = \frac{\partial u_i}{\partial t} + \nabla u_i \cdot \mathbf{u} = \frac{\partial u_i}{\partial t} + \sum_{j=1}^{3} u_j \frac{\partial u_i}{\partial x_j} \tag{2.4}$$

so for the whole vector

$$\mathbf{a} = \frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \tag{2.5}$$

Deformation Gradient Tensor

The last quantity that we will talk about is the "deformation gradient", $\tilde{\mathbf{F}}_t$. This quantity is defined as a $N \times N$ matrix at each point of the reference configuration Ω_t for a time $t \in I$ as follows

$$\tilde{\mathbf{F}}_t : \Omega_0 \to \mathsf{R}^N \times \mathsf{R}^N, \qquad \tilde{\mathbf{F}}_t = \nabla_{\boldsymbol{\xi}} \mathbf{x}(t, \boldsymbol{\xi}) \qquad \left(\tilde{\mathbf{F}}_t\right)_{ij} = \frac{\partial x_j}{\partial \xi_i} \tag{2.6}$$

and characterizes the local deformation at a material point with position vector ξ.

Another important property of the deformation gradient tensor is how it relates the gradient of fields in the Lagrangian frame with those in the Eulerian frame. Let $\tilde{\mathbf{f}} : \Omega_0 \to$

R^3 be a vector field defined in the Lagrangian frame and $\mathbf{f}$ its relative in the Eulerian frame. By applying the chain rule and recalling the definition of the deformation gradient we can see

$$(\nabla_{\boldsymbol{\xi}}\tilde{\mathbf{f}})_{ij} = \frac{\partial \tilde{f}_i}{\partial \xi_j} = \sum_{k=1}^{3} \frac{\partial f_i}{\partial x_k}\frac{\partial x_k}{\partial \xi_j} = \sum_{k=1}^{3} (\nabla \mathbf{f})_{ik}\,\tilde{F}_{kj} \Rightarrow \nabla_{\boldsymbol{\xi}}\tilde{\mathbf{f}} = \nabla \mathbf{f}\cdot\tilde{\mathbf{F}} \tag{2.7}$$

The last property of the deformation gradient tensor we are interested in is the expression for its time derivative, which will be useful in the following sections. By applying Equation (2.7) and knowing that the Lagrangian mapping is differentiable, which allows us to swap derivatives, we get

$$\dot{\tilde{\mathbf{F}}} = \frac{\partial}{\partial t}\left(\nabla_{\boldsymbol{\xi}}\mathbf{x}\right) = \nabla_{\boldsymbol{\xi}}\left(\frac{\partial \mathbf{x}}{\partial t}\right) = \nabla_{\boldsymbol{\xi}}\tilde{\mathbf{u}} = \nabla \mathbf{u}\cdot\tilde{\mathbf{F}} \tag{2.8}$$

As for any vector field, the determinant of the gradient is called the Jacobian, $\tilde{J}_t$. In the Lagrangian frame

$$\tilde{J}_t = \det \tilde{\mathbf{F}}_t. \tag{2.9}$$

The Jacobian of the Lagrangian mapping $\mathcal{L}_t$ gives an idea of the expansion or contraction of an infinitesimal volume $dU_0 \in \Omega_0$ through the application of the Lagrangian mapping, that is $dU_t = \tilde{J}_t dU_0$. Since by hypothesis it is an invertible mapping,the Jacobian will be nonzero everywhere. Moreover, for our case we will only consider orientation-preserving deformations, thus

$$\tilde{J}_t(\boldsymbol{\xi}) > 0 \qquad \forall \boldsymbol{\xi} \in \Omega_0. \tag{2.10}$$

An example of a deformation that does not preserve orientation would be the turning of a tennis ball inside out.

Transport Equations

Since we are dealing with a set of particles that occupy a certain volume in space continuously with the time, we may be interested in computing macroscopic quantities and their rate of variation with respect to the time. For example, imagine we have the internal energy density in the Eulerian frame, $\beta(t, \mathbf{x}(t, \boldsymbol{\xi}))$, and we want to compute the rate of win/loss of energy of our system throgh time. Our aim would be, then, to compute the value of the following:

$$\frac{d}{dt}\int_{\Omega_t} \beta(t, \mathbf{x}(t, \boldsymbol{\xi}))\; d\mathbf{x}.$$

There is not a direct way of computing this derivative, since the domain of integration itself is varying with time. The aim in this section is to find a relation which will help us solve this problem.

First of all, we might want to change our domain of integration from Ω_t to Ω_0. As it is kown from first calculous courses, the Jacobian of a mapping plays an important role when regarding the integration of functions in two different frames. In our case, the following relation holds:

Theorem 2.1.1. *Let $V_t \in \Omega_t$ be a subdomain of Ω_t and let us consider the function $f : I \times V_t \to \mathsf{R}$. Then, f is integrable on V_t iff $(f \circ \mathcal{L}_t)\tilde{J}_t$ is integrable on $V_0 = \mathcal{L}_t^{-1}(V_t)$, and*

$$\int_{V_t} f(t, \mathbf{x})\; d\mathbf{x} = \int_{V_0} \tilde{f}(t, \boldsymbol{\xi})\tilde{J}_t(\boldsymbol{\xi})\; d\boldsymbol{\xi},$$

which can be recognized as the well known Integration by Substitution Formula adapted to our own scope.

For the derivation of the transport equations, we will need the following result from matrix analysis:

Lemma 2.1.1 (Jacobi's Formula)**.** *Let $A : \mathsf{R} \to \mathsf{R}^{n\times n}$ be a differentiable map from the real numbers to the set of $n \times n$ square matrices. Then,*

$$\frac{d}{dt}(\det \mathbf{A}) = \mathrm{tr}\left(\mathrm{adj}(A(t))\frac{dA(t)}{dt}\right) = \det A(t)\mathrm{tr}\left(A(t)^{-1}\frac{dA(t)}{dt}\right).$$

Proof. See [8] □

Lemma 2.1.2 (Euler Expansion Formula)**.** *Let $\dot{\tilde{J}}_t$ be the Jacobian in the Lagrangian frame. Then,*

$$\dot{\tilde{J}}_t = \tilde{J}_t \nabla \cdot \mathbf{u}.$$

Proof. By using Lemma 2.1.1 and (2.8), we have

$$\dot{\tilde{J}}_t = \frac{\partial}{\partial t}\left(\det \tilde{\mathbf{F}}\right) = \det \tilde{\mathbf{F}}\ \mathrm{tr}\left(\tilde{\mathbf{F}}^{-1}\dot{\tilde{\mathbf{F}}}\right) = \det \tilde{\mathbf{F}}\ \mathrm{tr}\left(\nabla \mathbf{u}\right) = \tilde{J}_t \nabla \cdot \mathbf{u} \tag{2.11}$$

□

Theorem 2.1.2 (Reynolds Transport Theorem)**.** *Let $V_0 \subset \Omega_0$ and $V_t \subset \Omega_t$ be its image under the Lagrangian mapping. Let $f : I \times \Omega_t \to \mathsf{R}$ be a continuously differentiable scalar function with respect to bot variables $\mathbf{x}$ and t. Then,*

$$\frac{d}{dt}\int_{V_t} f\ d\mathbf{x} = \int_{V_t}\left(\frac{Df}{Dt} + f\nabla\cdot\mathbf{u}\right)\ d\mathbf{x} = \int_{V_t}\left(\frac{\partial f}{\partial t} + \nabla\cdot(f\mathbf{u})\right)\ d\mathbf{x}.$$

Proof. By using Lemma 2.1.2, Theorem 2.1.1 and (2.3), we have

$$\begin{aligned}\frac{d}{dt}\int_{V_t} f d\mathbf{x} &= \int_{V_0}\frac{\partial}{\partial t}\left(\tilde{f}\tilde{J}_t\right)\ d\boldsymbol{\xi} = \int_{V_0}\left(\dot{\tilde{f}}\tilde{J}_t + \tilde{f}\dot{\tilde{J}}\right)\ d\boldsymbol{\xi} = \int_{V_0}\left(\dot{\tilde{f}}\tilde{J}_t + \tilde{f}\tilde{J}_t\nabla\cdot\mathbf{u}\right)\ d\boldsymbol{\xi}\\ &= \int_{V_0}\left(\dot{\tilde{f}} + \tilde{f}\nabla\cdot\mathbf{u}\right)\tilde{J}_t\ d\boldsymbol{\xi} = \int_{V_t}\left(\frac{Df}{Dt} + f\nabla\cdot\mathbf{u}\right)\ d\mathbf{x} = \int_{V_t}\left(\frac{\partial f}{\partial t} + \nabla\cdot(f\mathbf{u})\right)\ d\mathbf{x}.\end{aligned}$$

□

Remark. *By the application of the divergence theorem, the latter expression of the theorem becomes*

$$\frac{d}{dt}\int_{V_t} f\ d\mathbf{x} = \int_{V_t}\frac{\partial f}{\partial t}\ d\mathbf{x} + \int_{\partial V_t} f\ \mathbf{u}\cdot\mathbf{n}\ dS, \tag{2.12}$$

where ∂V_t is the boundary of V_t and $\mathbf{n}$ is the outward pointing unit normal field of the boundary ∂V_t.

2.1.3 The derivation of the Navier Stokes equations

From now on, the symbol $V_t \subset \Omega_t$ will stand for the material volume at time t, that means the image through the Lagrangian mapping of a subdomain $V_0 \subset \Omega_0$, i.e. $V_t = \mathcal{L}_t(V_0)$.

Continuity equation or mass conservation

We will assume that there exists a measurable function $\rho : I \times \Omega_t \to \mathsf{R}$ called density such that the mass of the material volume V_t can be obtained as the integration of it in the domain, that is

$$\int_{V_t} \rho \, d\mathbf{x} = m(V_t).$$

A commonly accepted principle in Physics is that the total mass of a volume is neither created nor destroyed during the motion of our material particles. That can be translated to the relation

$$\frac{dm(V_t)}{dt} = \frac{d}{dt} \int_{V_t} \rho \, d\mathbf{x} = 0,$$

which by applying the Transport Theorem yields

$$\int_{V_t} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) \right) \, d\mathbf{x} = 0. \tag{2.13}$$

Due to the arbitrariness of the volume V_t and assuming that the terms in the integral are continuous the latter expression can be written in differential form as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0. \tag{2.14}$$

In our case, we will be dealing with constant density fluids, that means

$$\nabla \cdot \mathbf{u} = 0. \tag{2.15}$$

This result acts as a kinematic constraint for the velocity field in the flows with constant density fluids. Recalling (2.1.2) we also obtain

$$\frac{\partial}{\partial t} \tilde{J}_t = \frac{D}{Dt} J_t = 0, \tag{2.16}$$

which is called the "incompressibility constraint". Note that for constant density fluids the incompressibility constraint is satisfied, while the fact of a fluid being incompressible does not imply it to have constant density.

Momentum conservation

Following the classical dynamics of Newton and Euler, the motion of a material body Ω_t is produced by the action of externally applied forces which are assumed to be of two kinds: surface forces $\mathcal{F}$ and body forces $\mathbf{b}$.

Body forces

Body forces are forces proportionals to the mass of the material volume at each point. They can be expressed by a vector field $\mathbf{f}^b : I \times \Omega_t \to \mathsf{R}^3$ called the "specific body force". Then, the resultant body force acting on a volume V_t is obtained by integrating this field times the density all over the volume, that is

$$\int_{V_t} \rho \mathbf{f}^b \, d\mathbf{x}.$$

Note that the units of the specific body force are those of force/mass, which are $[\mathbf{f}^b] = Ne/kg = m/s^2$, like the acceleration.

An example of a body force is the well known gravity force $\mathbf{f}^b = (0, 0, -g)$.

Surface forces

When a material volume Ω_t is subjected to external surface forces or contact forces $\mathcal{F}$ in a given measurable subset of the boundary $\Gamma_t^n \subset \partial\Omega_t$, internal contact forces and moments appear and are transmitted from point to point in the body due to the mechanical contact of one portion of the continuum onto the other. We will represent the external or "applied" forces through a vector field $\mathbf{t}^e : I \times \Gamma_t^n \to \mathsf{R}^3$ called applied stresses, with which we will be able to obtain the resultant force acting through the surface by integrating it on the surface, that is

$$\mathcal{F} = \int_{\Gamma_t^n} \mathbf{t}^e(t, \mathbf{x})\ d\sigma.$$

In order to determine these internal forces, we will follow the Cauchy's stress principle and the Cauchy stress tensor theorem.

(The Cauchy's stress principle)**.** *The balancing action of internal contact forces generates a contact force density or Cauchy traction field* $\mathbf{t}$*, called Cauchy stress,*

$$\mathbf{t} : I \times \Omega_t \times \mathbf{S}_1 \to \mathsf{R}^3$$

that represents a distribution of internal contact forces throughout the volume of the body in a particular configuration Ω_t*. Thus, the resultant of the material continuity forces acting on any material domain* $V_t \subset \Omega_t$ *is given by*

$$\int_{\partial V_t} \mathbf{t}(t, \mathbf{x}, \mathbf{n})\ d\sigma, \tag{2.17}$$

where $\mathbf{n} \in \mathbf{S}_1 = \{\mathbf{v} \in \mathsf{R}^3 \mid |\mathbf{v}| = 1\}$ *is the outward unitary vector of* ∂V_t*. In addition, we have that*

$$\mathbf{t} = \mathbf{t^e} \qquad \text{on } \partial V_t \cap \Gamma_t^n.$$

Remark. *The Cauchy traction field* $\mathbf{t}$ *is not a vector field because it depends not only on the position* $\mathbf{x} \in \Omega_t$ *of a particular material point, but also on the local orientation of the surface element as defined by its normal vector* $\mathbf{n} \in \mathbf{S}_1$*.*

With some assumptions on the regularity of the Cauchy stresses, we can relate the material continuity forces to a tensor field through the following result.

Theorem 2.1.3 (Cauchy stress theorem)**.** *Let us assume that for any* $t \in I$*, the body forces* $\mathbf{f}^b$*, the density* ρ *and* $\frac{D}{Dt}\mathbf{u}$ *are all bounded functions on* Ω_t *and that the Cauchy stress vector field* $\mathbf{t}$ *is continuously differentiable with respect to the variable* $\mathbf{x}$ *for each* $\mathbf{n} \in \mathbf{S}_1$*, and continuous with respect to* $\mathbf{n}$*. Then, there exists a continuously differentiable symmetric tensor field called "Cauchy stress tensor"*

$$\mathbf{T} : I \times \overline{\Omega_t} \to \mathsf{R}^{3\times 3}$$

such that

$$\mathbf{t}(t, \mathbf{x}, \mathbf{n}) = \mathbf{T}(t, \mathbf{x}) \cdot \mathbf{n}$$

Proof. Refer to [9]. □

Now that we defined all the forces acting on the material volume, we can state the principle of conservation of linear momentum.

(Principle of conservation of linear momentum)**.** *For any $t \in I$, the total variation of linear momentum with respect to time in a material volume V_t is equal to the sum of all body and surface forces acting on it, that is*

$$\frac{d}{dt}\int_{V_t} \rho(t,\mathbf{x})\mathbf{u}(t,\mathbf{x}) \;\; d\mathbf{x} = \int_{V_t} \rho(t,\mathbf{x})\mathbf{f}^b(t,\mathbf{x}) \; d\mathbf{x} + \int_{\partial V_t} \mathbf{T}(t,\mathbf{x})\cdot\mathbf{n} \; d\sigma. \tag{2.18}$$

By using the transport theorem on the left hand side and the divergence theorem on the surface forces term in (2.18) we obtain

$$\int_{V_t}\left(\frac{D}{Dt}(\rho\mathbf{u}) + \rho\mathbf{u}\nabla\cdot\mathbf{u}\right) \; d\mathbf{x} = \int_{V_t}\rho\mathbf{f}^b d\mathbf{x} + \int_{V_t}\nabla\cdot\mathbf{T} \; d\mathbf{x}.$$

Thanks to the arbitrariness of V_t and under the hypothesis that the terms under the integrals are continuous and the flow has constant density we can rewrite the latter integral expression into the differential equation

$$\rho\frac{\partial\mathbf{u}}{\partial t} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = \rho\mathbf{f}^b + \nabla\cdot\mathbf{T} \qquad \text{in } \Omega_t, \tag{2.19}$$

where we have used (2.3) for the expression of the material derivative.

Note that the extension from a volume $V_t \subset \Omega_t$ to Ω_t requires the value of the Cauchy stress tensor in Γ_t^n to be regarded as a boundary condition.

In order to close the system of equations (2.19) we need to find a relation between the Cauchy stress tensor and the kinematic quantities (in our case, the velocity field). These kind of relations are called constitutive laws and provide a characterization of the mechanical behaviour of the fluid in consideration.

As stated in the assumptions of our project, we will assume the fluid to have a Newtonian behaviour. For Newtonian incompressible fluids, the Cauchy stress tensor can be expressed as an afine function of the spatial derivatives of the velocity in the following way

$$\mathbf{T} = -P\mathbf{I} + \boldsymbol{\mu}(\nabla\mathbf{u} + \nabla\mathbf{u}^T), \tag{2.20}$$

where P is a scalar function called pressure, $\mathbf{I}$ stands for the identity matrix in $\mathsf{R}^{3\times3}$ and $\boldsymbol{\mu} > 0$ is the dynamic viscosity tensor of the fluid, which models how strong is the friction between particles.

Furthermore, Newtonian fluids assume the viscosity to be a tensor that does not depend on the stress state and velocity of the flow, but can depend on temperature or the spatial components. We will only deal with the case of isotropic Newtonian fluids, that is, we will assume μ is a single real value instead of a tensor.

To make the expression a little cleaner, let us denote by $\mathbf{L}$ the gradient of the velocity field. Then, the symmetric and skew-symmetric parts of $\mathbf{L}$ are

$$\mathbf{D}(\mathbf{u}) = \frac{\nabla\mathbf{u} + \nabla\mathbf{u}^T}{2} \qquad \text{and} \qquad \mathbf{W}(\mathbf{u}) = \frac{\nabla\mathbf{u} - \nabla\mathbf{u}^T}{2}. \tag{2.21}$$

These tensors are respectively called the strain rate tensor and the spin tensor, which are related to the change of length and orientation of the media. Then, Equation (2.20) can be rewritten as

$$\mathbf{T} = -P\mathbf{I} + 2\mu\mathbf{D}(\mathbf{u}).$$

If we now plug in the expression for the Cauchy stress tensor applied to Newtonian fluids in Equation (2.19), we obtain

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = \rho\mathbf{f}^b - \nabla P + 2\nabla\cdot(\mu\mathbf{D}(\mathbf{u})).$$

Since we have the density ρ in many of our terms in the equation, it is usually convenient to divide the whole equation by ρ and define the kinematic viscosity as $\nu = \mu/\rho$. Then,

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - 2\nabla\cdot(\nu\mathbf{D}(\mathbf{u})) = \mathbf{f}^b, \tag{2.22}$$

where $p = P/\rho$ is a scaled pressure and we have used that

$$\nabla\cdot(P\mathbf{I}) = \nabla P\mathbf{I} + P\nabla\cdot\mathbf{I} = \nabla P.$$

Remark. *The Momentum Equations can be simplified when ν is constant. Indeed, let us observe that*

$$\nabla\cdot(\nabla\mathbf{u}) = \nabla^2\mathbf{u} = \Delta\mathbf{u}.$$

Moreover, if the fluid is incompressible,

$$\nabla\cdot\left(\nabla\mathbf{u}^T\right) = \nabla\left(\nabla\cdot\mathbf{u}\right) = 0.$$

Therefore, recalling that

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

the Momentum Equations become

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} = \mathbf{f}^b. \tag{2.23}$$

The Navier Stokes equations

The set of differential equations formed by the Continuity Equation and the Momentum Equations derived in the previous sections are the so-called Navier-Stokes Equations for incompressible fluids.

These equations are valid for any fixed subset $\Omega \subset \Omega_t$ of the portion of space filled by the fluid. In most cases (as in the case of the flow around a car or an airplane), the flow motion is studied in the fixed domain Ω embodying the region of interest. When dealing with fluid-structure interaction for blood flow in a large artery, the wall of the vessel move as well, so that we will not be able to consider a fixed domain, but an intermediate approach is needed, namely the Arbitrary Lagrangian-Eulerian Frame (see 3.1.2).

However, dealing with this alternative formulation brings cumbersome equations into the model, which in fact are not needed in order to correctly understand our setting. Hence, we will stick to the assumption that our domain is fixed Ω.

For the problem to be well posed, the equations must be supplemented with a set of proper boundary and initial conditions. The classical boundary conditions are classified into

1. *Neumann boundary conditions* (prescribed boundary stresses). After discussing the Cauchy principle, we defined the stresses $\mathbf{t}^e$ as

$$\mathbf{t}^e = \mathbf{T} \cdot \mathbf{n} = -P\mathbf{n} + 2\mu\mathbf{D}(\mathbf{u})\mathbf{n} \quad \text{on} \quad \Gamma^n \subset \partial\Omega, \tag{2.24}$$

   where $\Gamma^n \subset \partial\Omega$ is a measurable subset and $\mathbf{t}^e : I \times \Omega \to \mathsf{R}^3$ is a smooth function whose value we prescribe.

2. *Dirichlet boundary conditions* (prescribed velocity). A given velocity $\mathbf{g} : I \times \Gamma^d \to \mathsf{R}^3$ is prescribed in the measurable set $\Gamma^d \subset \partial\Omega$, so that $\mathbf{u} = \mathbf{g}$ on Γ^d.

   Note that since $\nabla \cdot \mathbf{u} = 0$, if $\Gamma^d = \partial\Omega$ the following compatibility condition must be satisfied for each $t \in I$:

$$\int_{\partial\Omega} (\mathbf{g} \cdot \mathbf{n}) \, d\mathbf{x} = 0,$$

   which can be seen applying the divergence theorem.

For a problem involving only these two types of boundary conditions (which is usually the case), we require that $\Gamma^d \cup \Gamma^n = \partial\Omega$ and $\Gamma^d \cap \Gamma^n = \emptyset$.

The choice of these conditions is normally driven by physical considerations regarding each problem. For instance, for a viscous fluid ($\mu > 0$) like the ones we consider, is it normally considered the velocity of the fluid to be the same as a solid boundary (in our case, the vessel wall). However, for artificial boundaries, like inflow and outflow artery sections, the choice of more appropriate conditions can be more delicate in order to maintain the well-posedness of the problem.

Taking this into account, the problem of solving the Navier-Stokes Equations is

(Navier-Stokes Problem). *Find* $\boldsymbol{u} \in [C^2(\Omega)]^2$ *and* $p \in \boldsymbol{C}^1(\Omega)$ *such that*

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} = \mathbf{f}^b \ \textit{in} \ \Omega \\ \nabla \cdot \mathbf{u} = 0 \ \textit{in} \ \Omega \\ \mathbf{u} = \mathbf{g} \ \textit{on} \ \Gamma^d \\ \mathbf{T} \cdot \mathbf{n} = \mathbf{t}^e \ \textit{on} \ \Gamma^n \\ \mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0 \ \textit{in} \ \Omega, \end{cases}$$

with $\Gamma^d \cup \Gamma^n = \partial\Omega$, $\Gamma^d \cap \Gamma^n = \emptyset$ *and* $\mathbf{T} = -p\rho + 2\mu\mathbf{D}(\mathbf{u})$.

Non-dimensionalisation

Non-dimensionalisations is often useful when solving differential equations that involve physical quantities, for it brings to a characterisation of the system by a reduced amount of parameters or numbers, which can be obtained through the original conditions of the original problem.

In order to do so, one must first define the so-called dimensionless variables and choose the scale through which these are reduced. We denote by an upper star (*) the dimensionless variables. The scales that we will use are

- Characteristic length L: $\mathbf{x}^* = \mathbf{x}/L$
- Characteristic speed U: $\mathbf{u}^* = \mathbf{u}/U$
- Characteristic time T: $t^* = t/T = t/(L/U)$

- Pressure for dominant dynamic effects P_{dyn}: $p^* = P/P_{\text{dyn}} = P/(\rho U^2)$
- Pressure for dominant viscous effects P_{visc}: $p^* = P/P_{\text{visc}} = P/(\mu U/L)$

After defining these variables, we can rewrite the differential operators with respect to the scaled quantities as

$$\nabla = \frac{1}{L}\nabla^*, \qquad \frac{\partial}{\partial t} = \frac{\partial}{\partial t^*}\frac{\partial t^*}{\partial t} = \frac{U}{L}\frac{\partial}{\partial t^*},$$

where ∇^* stands for the spatial differential operator with respect to the reduced variables.

Now we plug in the corresponding scaled variables and operators to the Moment Equation (2.23) with zero body forces (i.e. $\mathbf{f}^b = 0$) and we obtain

$$\left[\frac{U^2}{L}\right]\frac{\partial \mathbf{u}^*}{\partial t^*} + \left[\frac{U^2}{L}\right](\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* + \left[\frac{U^2}{L}\right]\nabla^* p^* - \left[\frac{U}{L^2}\right]\nu\Delta^*\mathbf{u}^* = 0$$

for dominant dynamic effects, and

$$\left[\frac{U^2}{L}\right]\frac{\partial \mathbf{u}^*}{\partial t^*} + \left[\frac{U^2}{L}\right](\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* + \left[\frac{U}{L^2}\right]\nu\nabla^* p^* - \left[\frac{U}{L^2}\right]\nu\Delta^*\mathbf{u}^* = 0$$

for dominant viscous effects. Dividing both sides of these equations by U^2/L and defining the Reynolds number as $Re = LU/\nu$, the non-dimensionalised Momentum Equations can be written as

$$\frac{\partial u^*}{\partial t^*} + (\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* + \nabla^* p - \frac{1}{Re}\Delta^*\mathbf{u}^* = 0 \tag{2.25}$$

and

$$\frac{\partial u^*}{\partial t^*} + (\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* + \frac{1}{Re}\nabla^* p - \frac{1}{Re}\Delta^*\mathbf{u}^* = 0. \tag{2.26}$$

For the incompressibility constraint, the result is way more straightforward:

$$\nabla \cdot \mathbf{u} = \frac{U}{L}\nabla^* \cdot \mathbf{u}^* = 0 \Rightarrow \nabla^* \cdot \mathbf{u}^* = 0. \tag{2.27}$$

Remark. *Note that some of the scaling variables that we chose are also related to other known quantities or other variables. This reduces the amount of characteristic numbers in the non-dimensionalisation. However, other choices for the scales may lead to different expressions of the equations, as we will see in Chapter 3.1.*

With this reduction, we obtain a set of equations which only depend on one parameter, namely the Reynolds number. Hence the solution of the system is characterised by the quantities that affect the Reynolds number, in such a way that two different systems with the same Reynolds number show the same behaviour.

Let us abuse notation and drop the upper star (*) of the dimensionless variables. Then, the problem to be solved becomes

(Non-dimensional Navier-Stokes problem)**.** *Find* $\mathbf{u} \in [C^2(\Omega)]^2$ *and* $p \in \boldsymbol{C}^1(\Omega)$ *such that*

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \frac{1}{Re}\Delta\mathbf{u} = \mathbf{f}^b \ \textit{in}\ \Omega \\ \nabla \cdot \mathbf{u} = 0 \ \textit{in}\ \Omega \\ \mathbf{u} = \mathbf{g} \ \textit{on}\ \Gamma^d \\ \mathbf{T} \cdot \mathbf{n} = \mathbf{t}^e \ \textit{on}\ \Gamma^n \\ \mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0 \ \textit{in}\ \Omega, \end{cases}$$

where $\Gamma^d \cup \Gamma^n = \partial\Omega$, $\Gamma^d \cap \Gamma^n = \emptyset$ *and* $\mathbf{T} = -p\rho + 2\mu\mathbf{D}(\mathbf{u})$.

2.2 Methodology

2.2.1 The Finite Elements Method

The Finite Element Method (FEM) is one of the most used and popular methods to solve Partial Derivatives Equations (PDE) numerically. PDEs are equations that contain unknown multivariate functions and their partial derivatives like, for example, the Navier-Stokes equations. We usually write all PDEs in "residual form", obtained by moving all terms to the left-hand-side of the equation. Taking that into account, a way to express a general PDE problem is

(Strong Formulation of a PDE). *Find* $\mathbf{u} \in \boldsymbol{\mathcal{V}}(\Omega)$ *such that*

$$\mathcal{R}(x_1, x_2, \dots, x_n; \mathbf{u}, \partial_{x_1}\mathbf{u}, \dots, \partial_{x_n}\mathbf{u}; \partial^2_{x_1x_1}\mathbf{u}, \dots, \partial^2_{x_1x_n}\mathbf{u}, \dots) \equiv \mathcal{R}(\mathbf{u}) = 0 \qquad \textit{in } \Omega, \quad (2.28)$$

where we have used the standard notation

$$\partial_{x_i}\mathbf{u} \equiv \frac{\partial \mathbf{u}}{\partial x_i}.$$

Higher-order partial derivatives are denoted in the obvious way. The function must satisfy the boundary conditions

$$\mathbf{u}_{|\Gamma^d} = \mathbf{g}, \quad \mathbf{u}_{|\Gamma^n} \cdot \mathbf{n} = h.$$

The set $\boldsymbol{\mathcal{V}}(\Omega)$ denotes the functional space containing the function which we seek. The boundary $\partial\Omega$ is divided into two disjoint parts, namely the Dirichlet part, Γ^d, and the Neumann part, Γ^n. These fulfill $\Gamma^d \cup \Gamma^n = \partial\Omega$ and $\Gamma^d \cap \Gamma^n = \emptyset$. The boundary conditions could be of any type of combination, as long as they maintain the well-posedness of the problem.

The weak formulation

The FEM method consists in solving the weak formulation of our PDE problem, i.e, reformulating the problem in an integral form by taking the scalar product $(\cdot, \cdot)$ in $\mathcal{L}^2(\Omega)$ to (2.28) forcing the identity for all the functions of a properly chosen space. This formulation can be understood as if the solution to the weak formulation is going to fulfill the strong equation in an averaged way for any weighting function.

Thus, the new problem becomes

(Weak Formulation of a PDE). *Find* $\mathbf{u} \in \boldsymbol{\mathcal{V}}(\Omega)$ *such that*

$$\int_\Omega \mathcal{R} \cdot (\mathbf{u})\mathbf{w} \; d\Omega = 0, \qquad \forall \mathbf{w} \in \boldsymbol{\mathcal{V}}_0(\Omega) \quad (2.29)$$

and satisfies the boundary conditions

$$\mathbf{u}_{|\Gamma^d} = \mathbf{g}, \quad \mathbf{u}_{|\Gamma^n} \cdot \mathbf{n} = h$$

with $\boldsymbol{\mathcal{V}}_0(\Omega) = \{\mathbf{w} \in \boldsymbol{\mathcal{V}}(\Omega) \mid \mathbf{w}(\mathbf{x}) = 0 \;\; \forall \mathbf{x} \in \Gamma^d\}$.

It is easy to see that a solution of the strong problem is going to fulfill the weak problem. However, the inverse is not always true, but given certain conditions it is. For a more detailed discussion of it, see [10].

This integral can be normally exploited by using the integration by parts rule, reducing the order of the derivatives of $\mathbf{u}$ and transferring them to $\mathbf{w}$.

In order to better understand the concept of the weak formulation, we will discuss its applications to the very well known Poisson's Equation. The strong formulation of the problem is

(Poisson's Equation). *Find* $u \in \boldsymbol{\mathcal{V}}(\Omega)$ *such that*

$$\mathcal{R}(u) = \Delta u - f = 0 \qquad in\ \Omega \subset \mathsf{R}^d \tag{2.30}$$

and

$$u_{\partial\Omega} = g$$

where Δ *is the Laplace operator* $\Delta = \nabla^2 = \sum_{i=1}^{d} \partial^2_{x_i x_i}$.

We will obtain the weak formulation of the problem by taking the scalar product of the residual by a test function w in a certain space $\boldsymbol{\mathcal{V}}_0(\Omega)$. But first, let us exploit the integral by using the integration by parts rule, so that

$$\int_\Omega (\Delta u - f) w \, d\Omega = \int_\Omega \Delta u w \, d\Omega - \int_\Omega f w \, d\Omega = \int_{\partial\Omega} w \nabla u \cdot \mathbf{n} \, d\Gamma - \int_\Omega \nabla u \cdot \nabla v \, d\Omega - \int_\Omega f w \, d\Omega =$$

$$= \{w = 0 \ \ \forall \mathbf{x} \in \partial\Omega\} = - \int_\Omega \nabla u \cdot \nabla v \, d\Omega - \int_\Omega f w \, d\Omega$$

This strategy is often used since it allows us to transfer orders of derivatives from our unknown function u to the test function w, also using the fact that w vanishes on the boundary.

Note: for a problem with both Dirichlet and Neumann boundary conditions, the surface term in the integration by parts rule is split into two parts: one over Γ^d, which will still vanish, and one over Γ^n, in which the value of $\mathbf{u} \cdot \mathbf{n}$ is known as boundary condition.

Note that we can see that in order to make our calculations reasonable, our functions must be differentiable and their 0-th and first derivatives must be square integrable. These type of functions are members of a (well-known) function space called Sobolev space that is usually denoted by $H^1(\Omega)$. Bearing this in mind, we can write the weak formulation of the Poisson's Equation as follows:

(Weak formulation of the Poisson's Equation). *Find* $u \in H^1(\Omega)$ *such that*

$$\int_\Omega \nabla w \cdot \nabla u \, d\Omega + \int_\Omega f w d\Omega = 0 \qquad \forall w \in H^1_0 \tag{2.31}$$

and

$$u_{|\partial\Omega} = g,$$

with $H^1_0(\Omega) = \{w \in H^1(\Omega) \mid w(\mathbf{x}) = 0, \ \forall \mathbf{x} \in \partial\Omega\}$.

The choice of the functional spaces is problem dependent. However, a pragmatic method to find the most suitable space is to determine the minimal constraints that must be set to the unknown and test function for the weak formulation to be consistent. The most common spaces used in these situation are the Sobolev spaces, $H^i(\Omega)$, whose index i stands for the fact that their functions are i times differentiable and that their differentials are square integrable up to the i-th.

The FEM uses the definition of the weak solution to develop a numerical method that can be used to determine approximate solutions to our problem. In order to do so, let us first split our solution u into two parts

$$u = u_h + u_p,$$

where u_p is an arbitrary function that satisfies the Dirichlet boundary conditions. Therefore, the unknown function u_h has to satisfy homogeneous Dirichlet boundary condition. Then, we expand our solution in terms of a given infinite set of basis $\phi_j(\mathbf{x}) \in \mathcal{V}(\Omega)$. The expression of the solution is

$$u(\mathbf{x}) = u_p(\mathbf{x}) + \sum_{i=1}^{\infty} U_i \phi_i(\mathbf{x}). \tag{2.32}$$

Since the set $\{\phi_j\}_j$ is a complete basis of $\mathcal{V}(\Omega)$, every test function w can be written as $w = \sum_{i=1}^{\infty} W_i \phi_i(\mathbf{x})$ and the condition of the weak formulation for the integral to be accomplished by any test function can be changed to: for any basis function ϕ_j.

In practice, the series is truncated, and only a finite amount of functions N is taken. Still, they must fulfill that $\mathcal{V}_h(\Omega) = \text{span}\{\phi_i \mid i = 1 \dots N\} \subset \mathcal{V}$ and $\mathcal{V}_h(\Omega) \to \mathcal{V}(\Omega)$ as $N \to \infty$. The new problem obtained after the truncation is made is called the Galerkin problem.

By inserting our new approximation

$$u(\mathbf{x}) \approx u^h(\mathbf{x}) = u_p(\mathbf{x}) + \sum_{i=1}^{N} U_i \phi_i(\mathbf{x})$$

of the solution into our original equation we obtain:

(FEM equation)**.** *Find* $\mathbf{U} \in \mathsf{R}^N$ *such that*

$$r_j(U_1, ..., U_M) = \int_\Omega R\left(u_p + \sum_{i=1}^{N} U_i \phi_i\right) \phi_j(\mathbf{x})\ d\Omega = 0, \qquad \forall j = 1, \dots, M. \tag{2.33}$$

The equations defined in (2.33) form a set of algebraic equations that can be solved using the Newton's method. Let us recall the steps of this method:

1. Set $i = 0$ and give an initial guess $\mathbf{U}^i = (U_1^i, ..., U_N^i)$.
2. Compute the residuals $r_k^i(U_1^i, ..., U_N^i)$. If the norm of the residuals is less than our tolerance, we accept the approximation $\mathbf{U}^i$ and stop the process.
3. Compute the Jacobian matrix

$$J_{kj} = \left.\frac{\partial r_k}{\partial U_j}\right|_{(U_1^i, ..., U_M^i)} \qquad j, k = 1, ..., N$$

4. Solve the linear system

$$\underline{\mathbf{J}}\delta\mathbf{U} = -\mathbf{r}$$

5. Update the approximation

$$\mathbf{U}^{i+1} = \mathbf{U}^i + \delta\mathbf{U}$$

6. Set $i = i + 1$ and go back to 2.

For a good enough initial guess, Newton's method converges quadratically towards the solution. Furthermore, if the equation is linear, Newton's Method converges in a single step. For instance, recall the expression of the residual in our latter example of Poisson's Equation,

$$r_j(\mathbf{U}) = \int_\Omega \nabla u_p \cdot \nabla \phi_j \; d\Omega + \sum_{i=1}^{N} U_i \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \; d\Omega + \int_\Omega f \phi_j \; d\Omega, \tag{2.34}$$

whose solution by simply forcing $\mathbf{r} = 0$ would be to solve the linear system of equations

$$\sum_{i=1}^{N} U_i \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \; d\Omega = -\int_\Omega \nabla u_p \cdot \nabla \phi_j \; d\Omega - \int_\Omega f \phi_j \; d\Omega$$

which is equivalent to compute the update of our initial guess $\mathbf{U}^0$ using the Jacobian matrix $J_{ij} = \int_\Omega \nabla \phi_i \nabla \phi_j \; d\Omega$:

$$\mathbf{U}^1 = \mathbf{U}^0 + \delta \mathbf{U} = \mathbf{U}^0 - \mathbf{r}^0 \underline{\mathbf{J}}^{-1} = \left(\underline{\mathbf{J}} \mathbf{U}^0 - \mathbf{r}^0 \right) \underline{\mathbf{J}}^{-1}$$

The true power of FEM comes from the way these entries in the Jacobian matrix are computed. Since they involve solving an integral of the generators of our desired functional space $\mathcal{V}_h$, a smart choice on these functions is crucial for the simplification of the algorithm. In the following section, we discuss how we can choose these functions in order to do so.

Shape Functions

The key feature of the finite element method is that the used functions that form the basis have bounded support, being zero over most of the domain, and have the same functional form. This renders a sparse Jacobian matrix, which helps in the calculations in steps 3 and 4.

In order to introduce these functions, let us first introduce a set of nodes $\mathbf{x}_i \in \Omega$ with $i = 1 \dots N$, which will discretize our domain, The family of functions will be indexed by the set of nodes, and will have the property of being Kronecker's delta on the nodes, i.e. $\phi_i(\mathbf{x}_j) = \delta_{ij} \; \forall i, j = 1 \dots N$. They are called shape functions, and are normally chosen as piecewise continuous degree-k polynomials which vanish out of the vicinity of their corresponding node.

To better understand this concept, figure 2.1 shows the sketch of two shape functions of degree 1 (linear) for a 1 dimensional domain $\Omega = (0, 1)$ with a number of nodes $N = 5$.

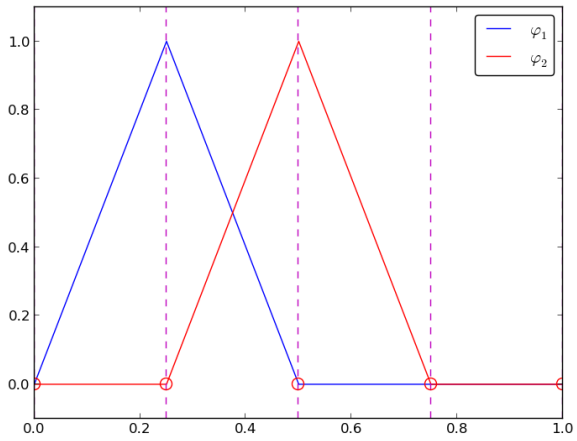


Figure 2.1: Two shape functions ϕ_1 and ϕ_2 for a 1 dimensional domain $\Omega = (0, 1)$ with a number of nodes $N = 5$

In other words, our solution is going to be an interpolation of the resulting values on the nodes, with degree k polynomials matchig the degree of the shape functions. Thus, a higher chosen order of the interpolation allows for a greater accuracy of the solution, as well as the density of nodes we assign to each part of the domain. Including the set of nodes $I \subset \{1, \ldots, N\}$ on the boundary, our sought solution is finally going to be

$$u^{FEM}(\mathbf{x}) = \sum_{i=1}^{N} U_i \phi_i(\mathbf{x})$$

with $U_i = g(\mathbf{x}_i)$ for $i \in I$ which is now going to be consistent with the specifications given before.

Figure 2.2 shows an example of the interpolation of a solution, with both the shape functions and real solution.

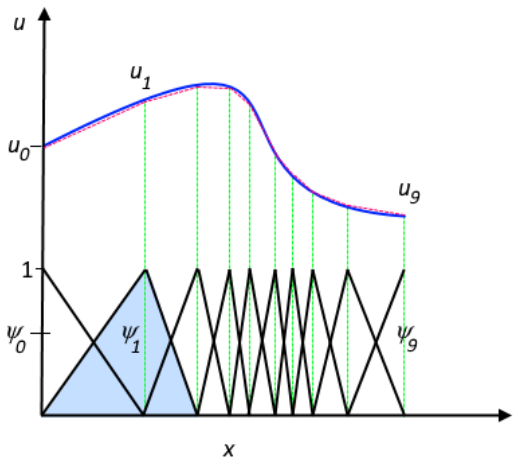


Figure 2.2: Real solution in blue; approximation in red. The shape functions are shown below the solutions.

Note that the set of nodes is not necessarily equally spaced, which is one of the strengths of the FEM method. However, the expression of each shape function is still node dependent, which forces us to represent them individually instead of globally. This problem is going to be tackled in the following section.

Elements

As told in the previous section, the compact support corresponding to each of the shape functions is going to simplify the computation of the integrals in the Jacobian matrix. Taking a closer look at the definition we gave for the shape functions, we can see that the integrals are going to vanish out of the neighbourhood of every node. In order to better exploit this property, let us define a partition of our domain Ω into M elements ω_i such that $\Omega = \cup_{i=1}^{M} \omega_i$ and $\int_{\omega_i \cap \omega_j} d\Omega = 0$ for $i \neq j$. A given set of elements in our domain defines what we call a mesh.

Each element w_i consists of a given number of nodes that depends upon the order of local approximation within the element. For example, for 1D problems we have segments, for 2D problems we can have triangles or quadrilaterals, while for 3D problems tetrahedrons or cubes are the mostly used. For more information on the formal definition of an element and some examples, see [11]. Some examples are shown in the following figure:

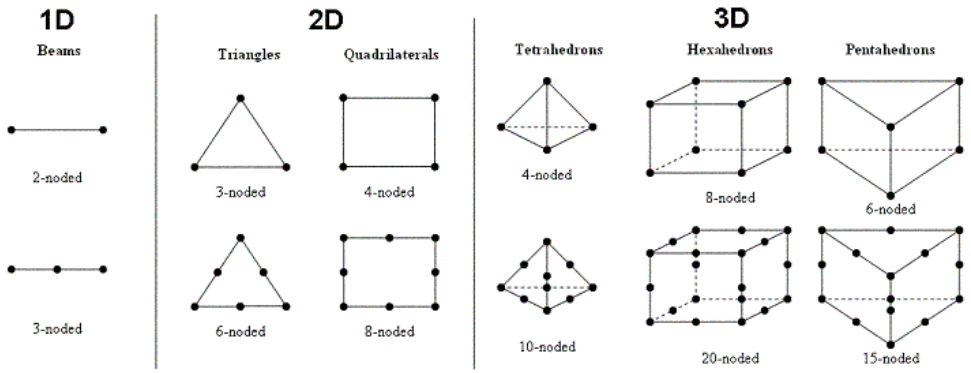


Figure 2.3: Different types of elements for 1,2 and 3 dimensional domains.

With that in mind, the residuals and Jacobian matrix for a given node is now going to be contribution of the resulting integral of the neighbouring elements, that is

$$r_j(\mathbf{U}) = \sum_{j=1}^{M} \int_{\omega_j} R\left(\sum_{i=1}^{N} U_i \phi_i\right) \phi_j(\mathbf{x})\ d\Omega.$$

For that, we will compute the contribution of the integral in each element in our domain to each of the entries of the Jacobian matrix and residuals that correspond to the nodes inside the element. The process of adding the contribution of the elements to the nodes in them is called **assembly.**

However, we still have not solved the problem of having node dependent shape functions. In order to do so, we shall now develop an alternative, local representation of the shape functions that involves only quantities that are intrinsic to each element. For this purpose, we introduce the concept of the reference element $\mathcal{E}$, which is a local expression of the element and the shape functions defined in it.

The integral over every element will be computed using a simple change of variables called the *isoparametric transformation* $\mathcal{I} : \mathcal{E} \to \Omega$ which is basically the interpolation of the coordinates in our domain Ω of the nodes using the reference shape functions.

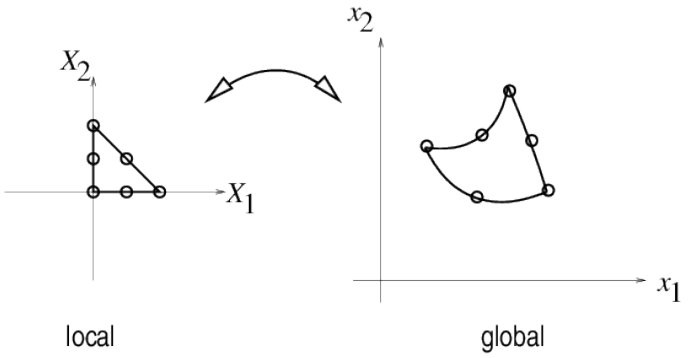


Figure 2.4: Example of 2D, Isoparametric transformation with triangular elements and quadratic interpolation.

Once we have the mapping that relates our domain with the reference frame, an integral over an element $\omega_i \subset \Omega$ can be computed via the integration by substitution rule as

$$\int_{w_i} f \; d\Omega = \int_{\mathcal{E}} fJ \; d\mathcal{E}, \tag{2.35}$$

where f is any of the functions that we are considering and $J = \frac{dx}{ds}$ is the Jacobian of the isoparametric transformation. This transformation is very useful to evaluate the integrals numerically, because it allows us to define the Gaussian points for the integral quadrature as well as the weights in order to make our integrals to be computed exactly in the reference frame. The degree of the quadrature is going to depend not only on the degree of the interpolation we are using but the equation itself. For more information on that aspect, see [12].

Take for example a 1D Poisson problem with linear interpolation, i.e. having 2 node elements. Let x_i and x_{i+1} be the coordinates of two consecutive nodes, and let us define our local coordinate $s \in [-1, 1]$ such that it parametrises our coordinates $x(s)$ and has the property that $s = \pm 1$ corresponds to x_i and x_{i+1}, respectively. The local shape functions,

$$\phi_1(s) = \frac{1}{2}(1 - s) \qquad \phi_2(s) = \frac{1}{2}(1 + s)$$

are going to fulfill the condition of being Kronecker's delta on the nodes as well. The mapping $x(s)$ must interpolate the coordinates of the two nodal positions in Ω. The smartest choice for the Isoparametric transformation is then

$$x(s) = x_i\phi_1(s) + x_{i+1}\phi_2(s)$$

Since the computation of the integrals involves the product of the derivatives of the shape funcionts, the Gauss quadrature must be such that is able to compute exactly polynomials of degree 0.

We have now built a method that is independent of the domain and computes the solution of a given PDE problem via the interpolation of the solution on a set of nodes with a certain degree of interpolation. To sum up, the process of solving a PDE problem via the Finite Element Method is

1. Find the weak formulation of the PDE, reducing the global order of derivatives as much as possible, transferring derivatives of the uknowns to the test functions via the integration by parts rule.
2. Discretize the domain Ω into elements, which at the same time will have nodes in it.
3. Assign boundary values to the unknowns U_i that belong to nodes on the boundary.
4. Give an initial guess of the solution $\mathbf{U}^0$
5. Update the solution via the Newton's method until a desired tolerance of error is reached, by computing the contribution of each element to the Jacobian matrix and the residual vector, by knowing the nodes that belong to it.
6. Build the solution of the problem defined in Ω as a linear combination of the shape functions and the nodal values found in the previous step.

3 Description of the work done

3.1 Pulsating flow through a collapsible channel

3.1.1 The model

Geometry

Consider a flow through a 2D channel of width H and length $L_T = L_{up} + L_{col} + L_{down}$ with a pressure drop between the inflow and the outflow $p_{up} - p_{down}$ whose upstream and downstream walls are rigid, whereas the upper wall in the central section is an elastic membrane whose shape is driven by the motion of the fluid and an external pressure p_{ext}.

The elastic membrane is parametrised by a Lagrangian coordinate $\xi \in (0, L_{col})$ and the position of the moving wall is given by $\mathbf{R}_w(\xi, t)$ (see Figure 3.1 for a geometrical sketch of our problem).

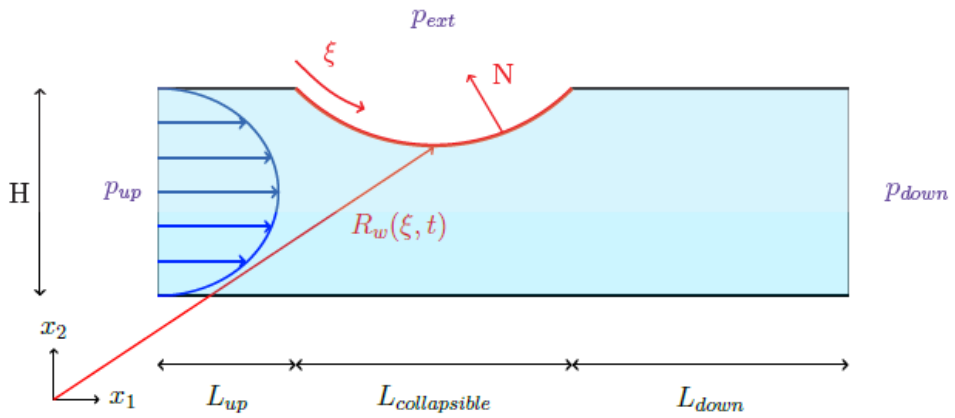


Figure 3.1: Sketch of the problem, extracted from [13]

As it is a coupled problem with two domains, governed by their own equations, we will be dealing simultaneously with solid and fluid equations. In the following sections we will cover the expression for the equations, as well as their non-dimensionalisation and initial and boundary conditions.

Fluid Mechanics equations

The fluid equations to be solved are the non-dimensional Navier-Stokes equations with dominant viscous effects. However, the chosen scales are not exactly the same as we used in 2.26. Thus, the resulting equations are slightly different but the method in order to obtain them is exactly the same.

The chosen scales for our problem are:

- Characteristic length H: $\mathbf{x}^* = \mathbf{x}/H$.
- Characteristic speed: U: $\mathbf{u}^* = \mathbf{u}/U = \mathbf{u}^*/(p_0(H)^2/12\mu L_{total})$.
- Characteristic frequency f: $t^* = tf$.
- Characteristic pressure p: $P^* = P/p = P/(\mu U/H)$

Note: as we did in section 2.1.3, after the non-dimensionalisation is performed, the stars over the variables are removed.

Then, the resulting equations for the fluid domain are

$$Re\left(St\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\,\mathbf{u}\right) = -\nabla P + 2\nabla\cdot(\mathbf{D}(\mathbf{u})) \tag{3.1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{3.2}$$

Comparing with the equations obtained in Section 2.1.3, we can see that the time derivative term on the LHS iw now multiplied by the Strouhal number $St = fH/U$, which describes the oscillating flow mechanisms in fluid mechanics. This is due to the new time scale chosen as a characteristic frequency of the system f, that is now not related with either the velocity nor the length scale. Also notice that the scale for the velocity is the median velocity of the Poiseuille flow, which means that our Reynolds number is going to be the median Reynolds number of the system.

As explained in Section 2.1.3, a proper set of initial and boundary conditions must be given in order to make the problem well posed.

The initial condition in our case is the well known Poiseuille flow, which has the form

$$\mathbf{u}(x_1, x_2, 0) = (6x_2(1-x_2), 0)).$$

The Poiseuille flow is the result of solving the Navier-Stokes equations in a channel of width H and length L_T, considering some assumptions on the flow characteristics. The resulting flow has only horizontal component and depends quadratically to the distance from the centerline of the pipe.

As for the set of boundary conditions, we have

- Inflow ($x = 0$)
  - Parallel inflow: $\mathbf{u}\cdot\mathbf{e}_2 = 0$
  - Applied traction: $\mathbf{t}\cdot\mathbf{e}_1 = p_{up} = p_0(1+\sin(St\cdot t))$
- Outflow ($x = L_T$)
  - Parallel outflow: $\mathbf{u}\cdot\mathbf{e}_2 = 0$
  - Applied traction: $\mathbf{t}\cdot\mathbf{e}_1 = p_{down} = 0$
- Rigid wall
  - Non-slip condition: $\mathbf{u} = \mathbf{u}_{wall} = 0$
- Elastic wall
  - Non-slip condition: $\mathbf{u} = \mathbf{u}_{wall} = \frac{\partial \mathbf{R}_w}{\partial t}$

In order for the initial pressure p_0 to be consistent with the Poiseuille flow, we must have $p_0 = 12L_T$.

Solid equations

The elastic membrane is modelled as massless elastic Kirchhoff-Love beam of wall thickness h and length L_{col} with Young modulus E and Poisson ratio ν. Thus, the effective Young modulus is given by $E_{eff} = E/(1-\nu^2)$. The beam is subject to a pre-axial 2nd Piola-Kirchhoff stress σ_0 large enough to allow us to assume incrementally linear elastic behaviour, which implies that $\sigma = \sigma_0 + \gamma$.

The deformation of the beam is governed by the principle of virtual displacements. This principle states that in a deformable body in equilibrium the work exerted by all internal forces is equal to the one exerted by external forces. In our case, the bending and strain energy (internal) will be equal to the work exerted by external forces. The principle undergoes a non-dimensionalisation as well using the same scales as for the fluid, except for the external pressure p_{ext}, which is instead scaled by the effective Young modulus E_{eff}. The resulting equation for the beam is

$$\int_0^{L_{col}} \left[(\sigma_0 + \gamma)\delta\gamma + \frac{1}{12}h^2\kappa\delta\kappa \right] = \int_0^{L_{col}} \left[\left(\frac{1}{h}\sqrt{\frac{A}{a}}\mathbf{f} - \Lambda^2 \frac{\partial^2 \mathbf{R}_w}{\partial t^2} \right) \cdot \delta\mathbf{R}_w \right]. \tag{3.3}$$

The left-hand side of the equation corresponds to the work of the internal forces: the strain energy and the bending energy. They are related to the strain tensor γ and bending tensor κ, which are defined as

$$\gamma = \frac{1}{2}(A - a), \qquad \kappa = -(B - b),$$

with

$$a = \frac{\partial \mathbf{r}_w}{\partial \xi} \cdot \frac{\partial \mathbf{r}_w}{\partial \xi}, \quad A = \frac{\partial \mathbf{R}_w}{\partial \xi} \cdot \frac{\partial \mathbf{R}_w}{\partial \xi}, \quad b = \mathbf{n} \cdot \frac{\partial^2 \mathbf{r}_w}{\partial \xi^2}, \quad B = \mathbf{N} \cdot \frac{\partial^2 \mathbf{R}_w}{\partial \xi^2},$$

where a and A are the squares of the length of infinitesimal material lines in the undeformed and deformed configuration, b and B are the curvature of the beam's centerline and $\mathbf{n}$ and $\mathbf{N}$ are the normal vectors to the beam's centerline.

The right hand side of the equation corresponds to the work done by external forces and the inertia forces. For the external forces, we have the traction done by the difference of pressure between the flow inside the channel and the external pressure and also the shear stresses done by the gradient of velocities inside the fluid. Thus, the dimensional load vector $\mathbf{f}$ is

$$\mathbf{f} = -(P_{ext} - P)\mathbf{N} - 2\mu\mathbf{D}(\mathbf{u}) \cdot \mathbf{N}. \tag{3.4}$$

As we have said, the pressures (and thus, the loads) are scaled by the Effective young modulus E_{eff}, but the pressures in the fluid are scaled by the viscous pressure. In order to rescale the fluid pressure to the walls scale, a new parameter Q is introduced in the equations. The resulting equations are

$$\mathbf{f} = -P_{ext}\mathbf{N} + Q\left(P\mathbf{N} - 2\mathbf{D}(\mathbf{u}) \cdot \mathbf{N}\right), \tag{3.5}$$

where $Q = \frac{\mu U}{H^* E_{eff}}$ represents the strength of the fluid structure interaction, as it is multiplying the load terms coming from the fluid in the expression of the load upon the solid.

The inertia forces are those related to the energy needed to move the mass of the beam, and Λ^2 may be interpreted as the adimensional wall density. However, since we are

considering a massless beam we have $\Lambda = 0$. Thus, the final equations become

$$\int_0^{L_{col}} \left[(\sigma_0 + \gamma)\delta\gamma + \frac{1}{12}h^2\kappa\delta\kappa \right] = \int_0^{L_{col}} \left[\frac{1}{h}\sqrt{\frac{A}{a}}\mathbf{f} \cdot \delta\mathbf{R}_w \right]. \tag{3.6}$$

The initial condition of the wall is the undeformed state, that is, a flat line between $(1, L_{up})$ and $(1, L_{up} + L_{col})$:

$$\mathbf{R}_w(\xi, 0) = (1, L_{up} + \xi).$$

The set of boundary conditions for the solid equations are

- Left end ($\xi = 0$)
  - Pinned: $\mathbf{R}_w(0, t) = (L_{up}, 1)$
- Right end ($\xi = L_{col}$)
  - Pinned: $\mathbf{R}_w(0, t) = (L_{up} + L_{col}, 1)$

Parameters and dimensionless numbers.

After the non-dimensionalisation of our problem, we have seen that our equations can be characterized by four parameters: the external pressure p_{ext}, the fluid-structure interaction parameter (also wall elasticity) Q, the Reynolds number Re and the Strouhal number St. These are the parameters that will be taken into account when interpreting the results. Other factors that could be studied but **are not** studied in this project are the geometry of the system, i.e. L_{up}, $L_{collapsible}$ and L_{down}.

The following table covers all the dimensionless numbers that appear in our equations.

Parameter	Definition	Meaning
Reynolds number	UH/μ	Ratio between dynamic and viscous forces
Strouhal number	fH/U	Frequency of pulsating oscillations
Q	$\mu U/E_{eff}H$	Wall stiffness / FSI parameter

Table 3.1

In order to get a glimpse of the order of magnitude of our parameters in the cardiovascular system, table 3.2 shows the value of the average Reynolds and Strouhal number for different blood vessels in the human body as well as values for the diameter of the vessel and mean velocity in the channel.

Vessel	Diameter (cm)	Mean velocity (cm/s)	Re	St
Arteriole	$\sim 10^{-3}$	$\sim$0.5	0.09	$4 \cdot 10^{-4}$
Femoral artery	$\sim$0.5	$\sim$12.5	190	$4 \cdot 10^{-3}$
Ascending aorta	$\sim$1.7	$\sim$25	1300	0.023

Table 3.2: Some values of the parameters and scales for human body blood vessels. Results taken from [14]

The type of vessels that we will be dealing with in the simulations are medium-large sized, like the Femoral artery, as we set Reynolds to sweep values from 1 to 200. The studied range of Strouhal numbers is not the common one (see Table 3.2), for our aim is to model anomalies due to particular values of the frequency, which may lead to resonance (see Section 4.1).

3.1.2 The code

In order to solve these equations, a modification of a code from the example list of the Oomph-lib is made. Oomph-lib is an object-oriented, open-source finite-element library for the simulation of multi-physics problems developed and maintained by Matthias Heil and Andrew Hazel of the School of Mathematics at The University of Manchester.

The main aim of the library is to provide an environment that facilitates the robust, adaptive solution of multi-physics problems by monolithic discretisations, while maximising the potential for code re-use. Monolithic solvers are those that solve both the fluid and solid equations at the same time, while those that use separated solvers for fluid and solid mechanics are called partitioned. Further information of the library and its functioning can be found in [15].

Different boundary conditions, parameter values and time features are set in the code in order to retrieve different solutions for multiple values of these parameters, as well as element types and mesh settings.

The code implements a Finite Elements method with quadratic Crouzeix-Raviart elements, which are fluid mechanics oriented triangular elements with different nodes for the pressure and the velocity used for discontinuous formulation of the finite element method. That means that the solution is continuous only in the mid point of the edges between two contiguous elements.

The code follows an Arbitrarian Lagrangian-Eulerian (ALE) frame, which means that a part of our domain is driven through the Lagrangian mapping explained in Section 2.1.1, while another part of our domain is considered "static", such as the boundary inflow and outflow. This focus is commonly used in the literature and research, since it allows to have a partially fixed domain while taking into account the movement of the particles in it. As the expression found in Section 2.1.3 for the Navier-Stokes equations follow the Eulerian frame and not an ALE frame, the resulting equations from an ALE formulation are slightly different, but will not be covered in this project. The interested reader may refer to [7].

The compilation and running is done via modified makefiles given by the creators and example lists. The main code has a lot of different ways to be executed, thus some flags must be given in order to choose the exact settings, such as element types.

The simulations are run in a computer with an Intel Core i5-4200U CPU 1.60GHz $\times$ 4 and 4 Gb of RAM. Each simulation took about five hours to be completed, depending on the Reynolds number, the time step dt and the total time.

Code Validation

In order to validate the correctness of the code, some results from [16] are reproduced. The article is written by the same creators of the code and covers deeply the study of pulsatile flows through collapsible channels.

The results we will reproduce are those found in the article's figure 10, where the displacement of the midpoint of the elastic wall is represented for a geometry of $L_{up} = 5$, $L_{col} = 10$ and $L_{down} = 30$ in the same length scale explained in section 3.1.1. However, the chosen scale for the pressure is the dynamic pressure, $p_{dyn} = \rho U^2$ while in our code the non dimensionalisation is done through the viscous pressure, $p_{visc} = \mu U/L$. Thus, to compute the upstream pressure to set we must multiply by the ratio of both the pressures. Since the pressure in the article $p_0 = L_{total}/Re$ is exactly the same as used in our code except it is divided by the Reynolds number and the ratio of the pressures

$$\frac{p_{dyn}}{p_{visc}} = \frac{\rho U}{\mu L} = Re$$

is exactly the Reynolds number, our expression for the pressure stays as stated in 3.1.1. Note that we could already see that by comparing equations 2.26 and 2.25.

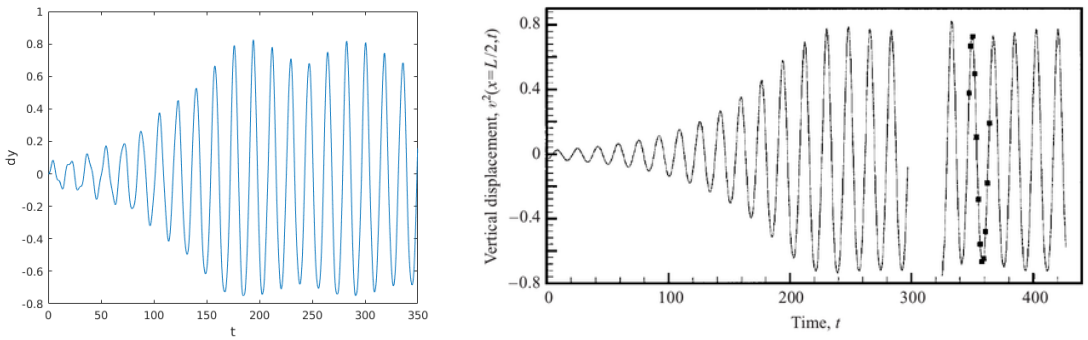
For the other governing parameters, the Reynolds number is set to 450 and the Strouhal number to 1. To characterize the wall elasticity, another parameter T is introduced in the paper, which in our case corresponds to a value of $Q = 0.222 \cdot 10^{-3}$. Finally, for the external pressure $p_{ext} = 0.96677$, which in the paper is in the dynamic pressure scale, we must multiply times Reynold (dynamic→viscous) and Q (viscous→wall stiffness). So, in our code $p_{ext} = 0.222 \cdot 10^{-3} \cdot 450 \cdot 0.96677 = 0.0966$.

The previously discussed parameters and other that do not require any transformation are shown in the following table:

Parameter	Symbol	Value
Reynolds number	Re	450
Strouhal number	St	1
Wall's elasticity	Q	$0.22 \cdot 10^{-3}$
External pressure	p_{ext}	0.0966
Upstream median pressure	p_0	$12L_{total}$
Wall thickness	h	0.01
Axial pre-stress	σ_0	1000

Table 3.3: Value of the parameters of the system for the checking of the code.

Once all the parameters are set, the simulation is done (see Figure 3.2).



(a) Own results

(b) Results in the referenced paper.

Figure 3.2: Comparison of the obtained results versus the results in [16].

We can see that the transient state is not the same for both cases, that is because the initial conditions set in [16] are not Poiseuille flow but other conditions for the total inflow q. However, the same steady state is obtained, with the same frequency and amplitude, checking the well performance of our code.

4 Results and discussion

4.1 Induced oscillations: driven damped oscillator

Our aim in this section is to characterize the response of our system with respect to the different parameters that govern it: the external pressure p_{ext}, the Reynolds and Strouhal numbers Re and St, and the fluid-structure parameter, Q. All the results have been analyzed using Matlab and the fittings using its curve fit tool, `cftool`.

Note: Not all of the parameters have been studied deeply for each of the phenomena described in the following sections, due to the lack of data for many values of some of them.

Let us recall the different conditions of our system: a pulsating pressure driven flow, modelled as a sine function of frequency St, goes through a channel with partially elastic and partially fixed wall. The initial state of the membrane is its undeformed state, i.e. parallel to the lower wall. The initial condition is a Poiseulle flow and the median value of the pressure is chosen so the driven flow in the undeformed channel is that Poiseulle flow.

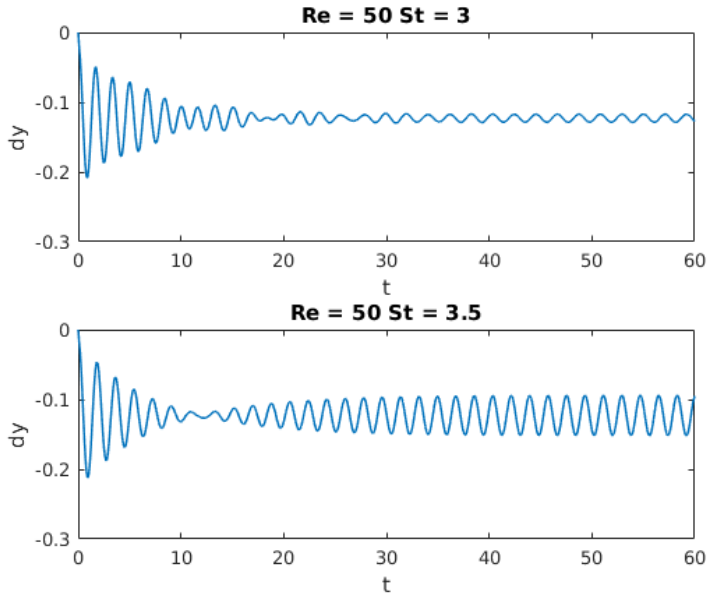


Figure 4.1: Vertical displacement $< dy >$ of the midpoint of the elastic wall.

In order to characterize the response of the system, the vertical position of the midpoint in the elastic membrane has been tracked through time. In all of the simulations, the response follows the same pattern as a very well known physical system: a driven damped oscillator (DDO). In figure 4.1, the displacement of the midpoint of the elastic membrane $dy = y_{mid} - 1$ with respect to time is shown for different values of St with fixed Re, Q and p_{ext}. As in the response for the driven damped oscillations, we can see that for

different value of the frequency of the driven force the system can exhibit resonance, in which the amplitude of the oscillations reach their maximum value. This phenomena will be discussed in depth in Section 4.5.

The main interesting phenomena of both the transient and steady response of a DDO are the phase shift and induced frequency, the amplitude of the oscillations and the transient time. Since our system is driven by a constant force p_{ext} as well as an oscillating one p_{up}, we will also be interested in characterizing the mean deviation of the elastic wall. All these concepts are explained in detail in the following sections.

4.2 Phase and frequency

The response of the system to the forced oscillations driven by the time-varying pressure drop between the outflow and inflow is first a transient phase, where the phase shift and frequency between the elastic wall and the pressure is not computable but chaotic. This is because of the sum of a transient and steady state forms with different frequencies. However, after a certain amount of time $t_{transient}$, the system presents a steady state which presents the same frequency but a shifted phase with respect to the forcing term.

In figure 4.2 we can see the phase difference between the system and the pressure gradient for certain values of the parameters. The figure shows the response for a St number, i.e. frequency, near the resonance point.

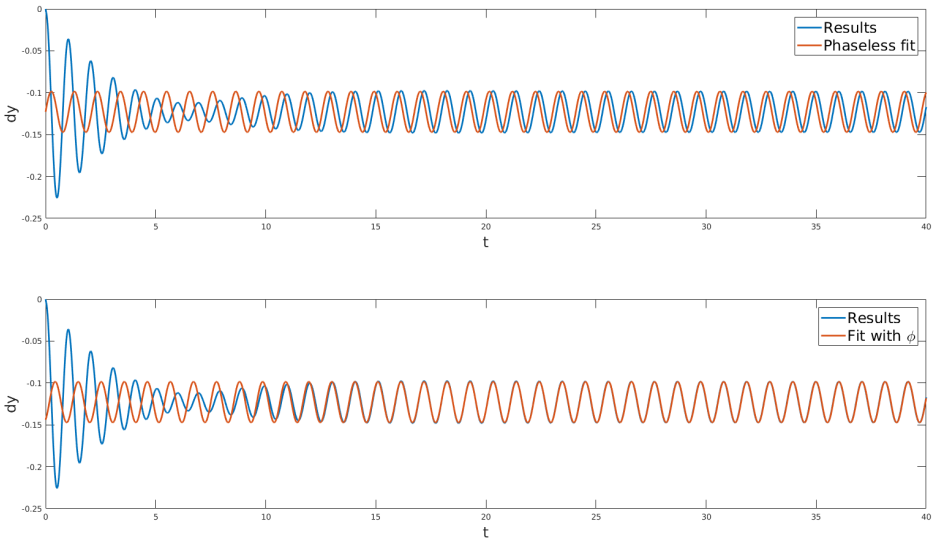


Figure 4.2: Up: comparison between the response of the system and pressure drop without the shifting phase; Down: correction taking the shifting phase into account. $Q = 10^{-5}$, $p_{ext} = 0.1$, $Re = 10$ and $St = 6$

In order to check that the results do follow a similar response as the one from a DDO, figure 4.3 shows the different values of phase found for several values of the St and the fitting of the phase from a common DDO.

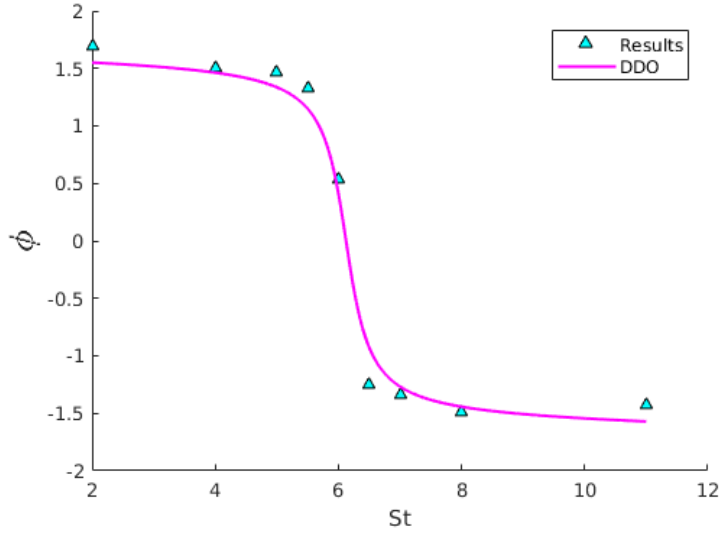


Figure 4.3: Phase between the pressure drop and oscillations of the elastic membrane for $Re = 50$, varying St, $Q = 10^{-5}$ and $p_{ext} = 0.1$. Fitting of the form $\tan\phi = \frac{b\omega}{k - m\omega^2}$.

If we compare the results for different values of the external pressure, we can see no phase shift nor affectation to the forced frequency. This can be seen in figure 4.4

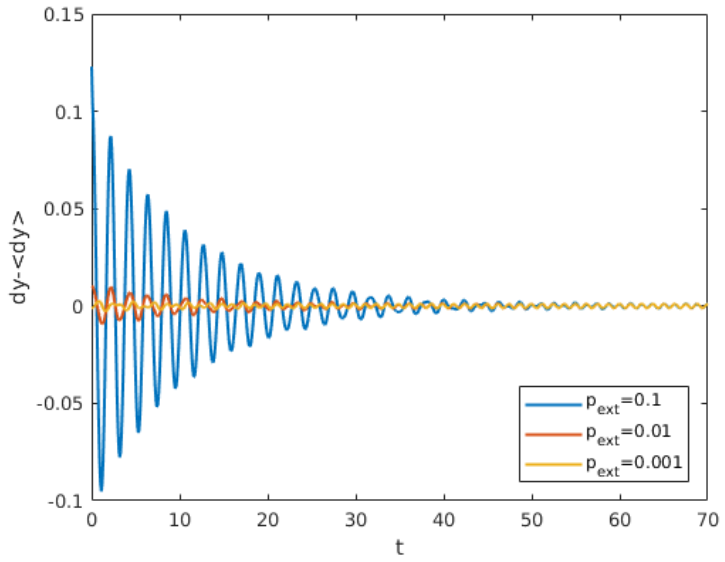


Figure 4.4: Comparison of the response for $Re = 50$, $St = 5$ and $Q = 10^{-5}$ and different external pressures. In all of the three responses the average deviation has been substracted in order to appreciate the phase shift or frequency difference.

Since our aim was to check that the response does follow a similar solution as the one for a DDO, and due to the lack of time for so many simulations, no variation of the fluid-structure interaction parameter Q has been done. However, since it represents the elasticity of the elastic membrane, it would be expected to affect the system as an

increasing value of the spring constant in the DDO. Other factors assumed to affect the phase shift are the Reynolds number, due to its relation with the ratio of advective and diffusive forces, and the geometry of the system, i.e. the lengths of the channel.

4.3 Mean deviation

Another important phenomena of the response is the mean deviation of the elastic membrane. Due to the fact that we want to be able to give biological useful information of the system, not only the oscillations but the mean deviation of the membrane is important, since it can induce thrombosis and cause rupture [17].

Since the mean deviation of the membrane is not expected to be related to the oscillatory nature of our problem, the first parameter to be studied is the external pressure p_{ext}. Because it constantly affects the bending of the elastic wall and is linear in terms of the load vector affecting it, we expect to find larger deformation for larger values. Figure 4.5 shows the mean deviation of the elastic membrane for varying values of p_{ext}.

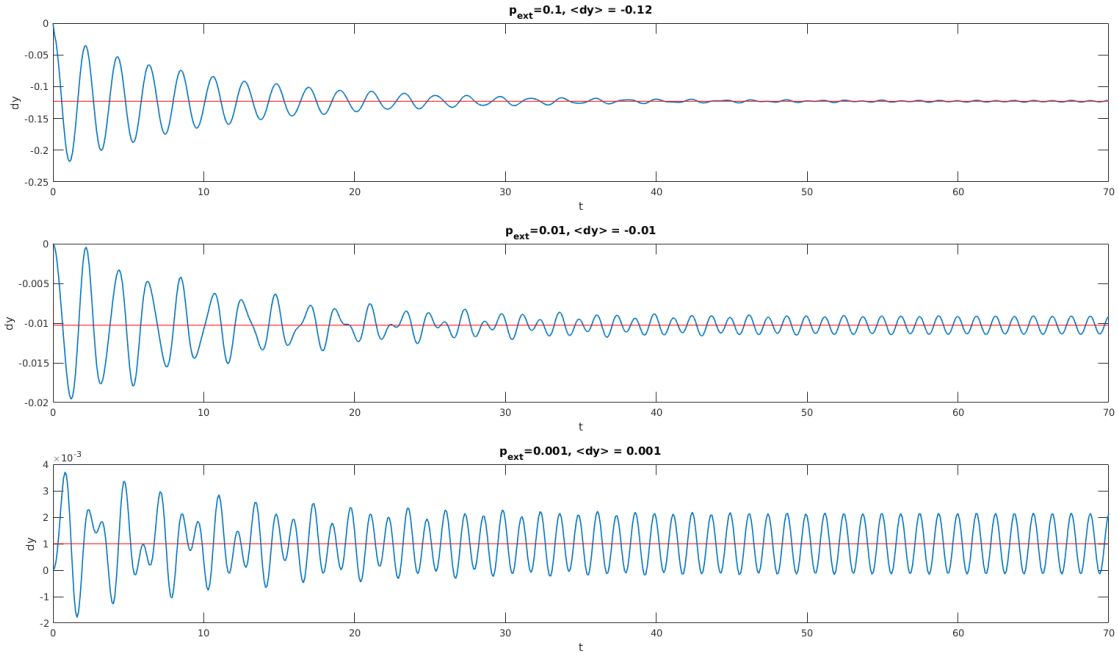


Figure 4.5: Deviation of the elastic membrane (blue) versus mean deviation (orange). Note: the scale of the vertical axis in each plot is different.

As we can see, an increasing value of the external pressure induces a larger inward buckling of the channel.

Having a look at the influence of Re, St and Q, the mean is not affected by Re nor St. Nonetheless, as it can be expected, a greater value of the elasticity constant Q induces a smaller (in absolute value) deformation of the membrane.

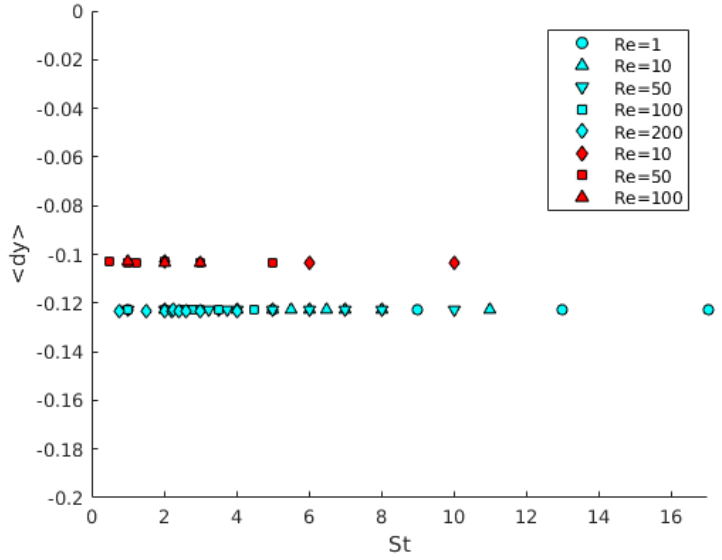


Figure 4.6: Different values of the mean deviation for varying Re and St. Cyan markers refer to $Q = 10^{-5}$, with mean deviation $< dy >= -0.123$ while red markers refer to $Q = 10^{-4}$, with mean deviation $< dy >= -0.103$.

4.4 Transient time

In a DDO, the transient time $t_{transient}$ of the system does not depend on the frequency of the forcing term but on the damping coefficient. Thus, we do not expect to find a dependence of $t_{transient}$ with respect to the frequency of the oscillations, i.e., the Strouhal number. However, as St depends linearly on the time scale, we must take that into account and check if $t_{transient}/St = const$.

The computation of the transient time has been carried out by fitting a sinusoidal signal with amplitude equal to the one in the steady state in advancing time, until a certain tolerance (0.5%) is reached from the norm of the difference between the sinusoidal signal and our results.

In figure 4.7 we can see that the relation previously discussed holds. The data points that are more deviated from the expected shape are those who are near the resonance point, where the transient state is more chaotic and the implemented function that computes the transient time had trouble to properly capture its value. It can also be seen that for greater values of the Reynolds number, the transient time increases, which is to be expected as the flow becomes more turbulent for larger Reynolds number and more unstabilities arise.

As for the influence of the external pressure, in figure 4.10 we can see that for greater values of p_{ext} the transient time is greater as well.

Due to the lack of data, the effect of the walls elasticity Q is not discussed. However, it is expected to have a strong relation with it, since the time transient in the DDO is related with the damping ratio.

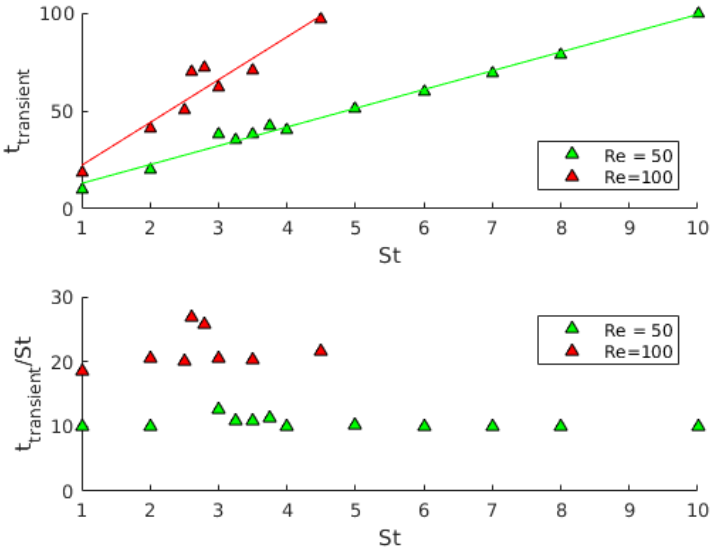


Figure 4.7: Plot of the transient time and normalized transien time with respect to the Strouhal number, for $Q = 10^{-5}$ and $p_{ext} = 0.1$.

4.5 Amplitude

The amplitude of the oscillations in a DDO as a variable of the frequency is modified by the damping ratio b, the spring constant k, the mass of the body m and the mean value of the driven force F_0 via the relation

$$A(\omega) = \frac{F_0}{\sqrt{(k - m\omega^2)^2 - \omega^2 b^2}} \tag{4.1}$$

Thus, we expect to find maximums of the amplitude by varying the frequency of the oscillations, i.e. by varying St.

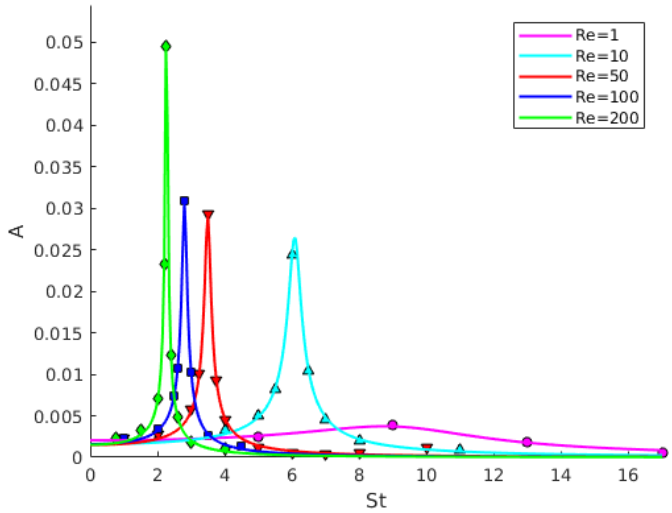


Figure 4.8: Amplitude of the steady state oscillations for fixed values of $Q = 10^{-5}$ and $p_{ext} = 0.1$.

Figure 4.8 shows the value of the amplitude found for the steady state for different values of St for fixed Re, Q and p_{ext}. The markers refer to the results obtained via simulations, while the fitting is done according to the shape given by Equation (4.1).

In order to bee able to predict large oscillations due to resonance, a relation between the position of the maximum of amplitude for each Re and St can be useful. Figure 4.9 shows an exponential relation between the values of Re and St for which the amplitude of the oscillations is greater, which we will call Re_{max} and St_{max}.

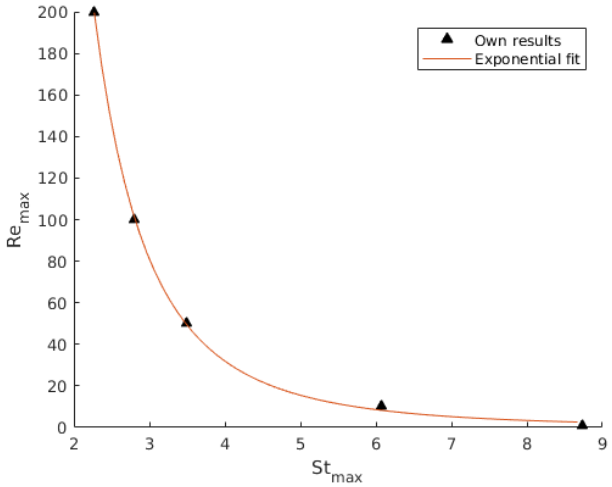


Figure 4.9: Fitting of the form $Re_{max} = a \cdot St_{max}^{b}$, with $a = 2832$ and $b = -3.235$ and same value of Q and p_{ext} as 4.8.

In the case of the external pressure, no relation with the amplitude is expected, since the response of a damped oscillator driven by a constant force (or step force) is the same as the homogeneous solution with just a shift in the equilibrium point. This can be seen in both Figures 4.4 and 4.10.

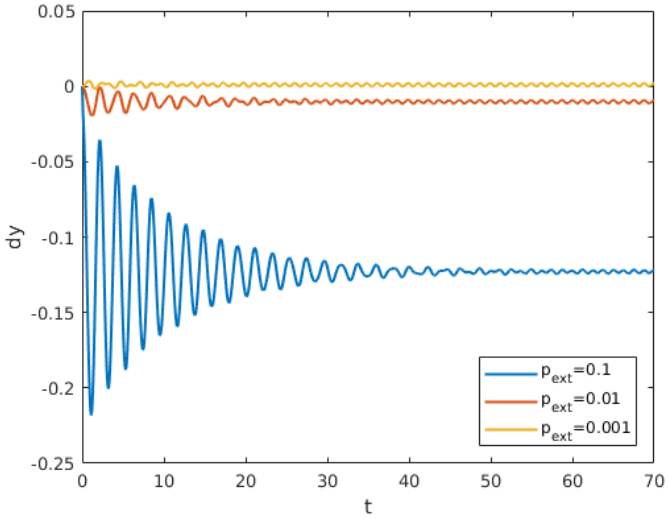


Figure 4.10: Vertical displacement of the wall for different values of p_{ext} at $Re = 50$, $St = 5$ and $Q = 10^{-5}$.

5 Conclusion and future work

The understanding of the motion of the flows through elastic channels is of great interest for the scientific community, since it plays a big role in the development of many processes occurring inside the human body, such as blood flow or the air through the vocal chords. In these processes, the conduits undergo inward buckling of the walls, which turns into a strong fluid interaction between the flow and its channel. Several models have been developed, ranging up to 3D with very complex geometries and numerical methods in order to fully capture the behaviour of these systems.

In this project, a 2D model of a partially elastic partially rigid channel wall with a pulsating inflow pressure of frequency St is tackled. The system is governed by a set of parameters, which includes the Reynolds number Re, the Strouhal numberSt , the fluid-structure interaction parameter Q and the external pressure p_{ext}. Although geometry conditions play an important role, we have not studied their effect on the response of our system due to a lack of computational time.

In order to analyse the response of our system, the vertical displacement of the midpoint lying on the elastic wall is tracked through time. Results show that the response of the elastic wall follows that of a Driven Damped Oscillator with both a step and a periodic forcing term due to the presence of an external constant pressure and a time-varying inflow pressure. This response consists of a first phase called transient, characterized by a damped oscillation with large modulated amplitude, and a second phase called stationary, with oscillations of constant frequency and amplitude.

The first phenomenon to be discussed is the transient time. The analysis of the results shows that it is not frequency-dependent, as in the Driven Damped Oscillator, but it does depend on the external pressure, the FSI parameter and the Reynolds number. That means that these parameters are somehow linked to the damping phenomena of the Driven Damped Oscillator parallelism.

The frequency of the steady oscillations is the same as the frequency of the inflow pressure. Nonetheless, there is a phase shift between the oscillations and the pressure, given that the "information" (in this case, the pressure drop) must travel through the channel until it reaches the elastic membrane. This phase shift is frequency-dependent, and has the same shape as the phase shift in a Driven Damped Oscillator. The effects of the Reynolds number have not been studied, but one would expect a shorter phase shift for larger values of the Reynolds number. This is because the Reynolds number measures the ratio between the dynamic and viscous (i.e. diffusive) effects, and the transport through dynamic effects tends to be faster than the diffusive spreading.

The displacement of the elastic wall in the steady state is the sum of a constant median displacement (equilibrium point) caused by the external pressure and an oscillatory term due to the pulsating inflow pressure. The amplitude of the oscillations can exhibit resonance for a certain value of the inflow frequency. A dependence between the Reynolds and Strouhal numbers that induce resonance has been found to be exponential of the form $Re \cdot St^{3.2} = \text{const}$. The peak value of the amplitude of this oscillations increases with

the Reynolds number, and is found to be up to 80% of the vessel width for large enough Reynolds numbers (see Figure3.2), which implies a huge fluid structure interaction. In those cases, even reverse flows might be detected. These are the cases where more physiological problems could arise, as the large deformations could cause rupture of the wall. However, the simulations performed in this project do not deal with them. Finally, the wall stiffness or fluid-structure interaction parameter Q has been found to affect these displacements as well, as would happen with a Driven Damped Oscillator.

The model dealt with in this project is certainly not the best or most accurate representation of a blood vessel. Nonetheless, the pulsating nature of the response found in this project should not be neglected since it would still apply in more complex models, and the fact of finding resonance phenomena in pulsating flows is of great interest when considering physiological processes. Further analysis of the response of the system for larger values of the Reynolds number might lead to quasi-stationary states where the solution becomes a sum of more than one frequency components. It would be physiologically interesting to see if this semi-stable states could become stable after the system has undergone such oscillations by modifying the governing parameters, thus creating a hysteresis.

In addition, geometry factors of the system (e.g. the ratio of elastic wall versus total wall) could be analysed in our own formulation. More accurate models could account for more complex geometries, such as axisymmetrical or 3D ones, resembling known human vessel channels. Developing such models and understanding more general types of systems in human physiology would not only allow the prediction of cardiovascular diseases, but also the simulations of surgical interventions, which could help train doctors without the need of patients.

Bibliography

1. Nichols, M., Townsend, N., Scarborough, P. & Rayner, M. Cardiovascular disease in Europe 2014: epidemiological update. *European Heart Journal* **35,** 2950 (2014).
2. Torii, R., Oshima, M., Kobayashi, T., Takagi, K. & Tezduyar, T. E. Fluid–structure interaction modeling of aneurysmal conditions with high and normal blood pressures. *Computational Mechanics* **38,** 482–490 (2006).
3. Tang, D., Yang, C., Kobayashi, S., Zheng, J. & Vito, R. P. Effect of stenosis asymmetry on blood flow and artery compression: a three-dimensional fluid-structure interaction model. *Annals of biomedical engineering* **31,** 1182–1193 (2003).
4. Torii, R., Oshima, M., Kobayashi, T., Takagi, K. & Tezduyar, T. E. Computer modeling of cardiovascular fluid–structure interactions with the deforming-spatial-domain/stabilized space–time formulation. *Computer Methods in Applied Mechanics and Engineering* **195,** 1885–1895 (2006).
5. Gao, F., Guo, Z., Sakamoto, M. & Matsuzawa, T. Fluid-structure interaction within a layered aortic arch model. *Journal of biological physics* **32,** 435–454 (2006).
6. Van Loon, R., Anderson, P. D. & van de Vosse, F. N. A fluid–structure interaction method with solid-rigid contact for heart valve dynamics. *Journal of computational physics* **217,** 806–823 (2006).
7. Quarteroni, A. & Formaggia, L. Mathematical modelling and numerical simulation of the cardiovascular system. *Handbook of numerical analysis* **12,** 3–127 (2004).
8. Bellman, R. *Introduction to Matrix Analysis: Second Edition* ISBN: 9780898713992. <`https://books.google.de/books?id=QVCflvTPYE8C`> (Society for Industrial and Applied Mathematics, 1997).
9. Serrin, J. in *Fluid Dynamics I/Strömungsmechanik I* 125–263 (Springer, 1959).
10. Brezis, H. *Functional analysis, Sobolev spaces and partial differential equations* (Springer Science & Business Media, 2010).
11. Logg, A., Mardal, K.-A. & Wells, G. *Automated solution of differential equations by the finite element method: The FEniCS book* (Springer Science & Business Media, 2012).
12. Quarteroni, A., Sacco, R. & Saleri, F. *Numerical mathematics* (Springer Science & Business Media, 2010).
13. Seeböck, T. *Fluid-structure interaction of pulsatile flows with elastic conduits* MA thesis (Friedrich-Alexander-Universität Erlangen-Nürnberg, 2016).
14. Caro, C., Fitz-Gerald, J. & Schroter, R. Atheroma and arterial wall shear observation, correlation and proposal of a shear dependent mass transfer mechanism for atherogenesis. *Proceedings of the Royal Society of London B: Biological Sciences* **177,** 109–133 (1971).

15. Heil, M. & Hazel, A. L. oomph-lib – An Object-Oriented Multi-Physics Finite-Element Library. *Lecture Notes in Computational Science and Engineering* **53,** 19–49 (2006).

16. Jenser, O. E. & Heil, M. High-frequency self-excited oscillations in a collapsible-channel flow. *Journal of Fluid Mechanics* **481,** 235–268 (2003).

17. Han, H., J.K.W., G. & et al., J. Artery Buckling: New Phenotypes, Models, and Applications. *Annals of Biomedical Engineering* **41,** 1399–1410 (2013).

Appendices

1 The solver code

```
//LIC// ====================================================================
//LIC// This file forms part of oomph-lib, the object-oriented,
//LIC// multi-physics finite-element library, available
//LIC// at http://www.oomph-lib.org.
//LIC//
//LIC//    Version 1.0; svn revision $LastChangedRevision: 1097 $
//LIC//
//LIC// $LastChangedDate: 2015-12-17 11:53:17 +0000 (Thu, 17 Dec 2015) $
//LIC//
//LIC// Copyright (C) 2006-2016 Matthias Heil and Andrew Hazel
//LIC//
//LIC// This library is free software; you can redistribute it and/or
//LIC// modify it under the terms of the GNU Lesser General Public
//LIC// License as published by the Free Software Foundation; either
//LIC// version 2.1 of the License, or (at your option) any later version.
//LIC//
//LIC// This library is distributed in the hope that it will be useful,
//LIC// but WITHOUT ANY WARRANTY; without even the implied warranty of
//LIC// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
//LIC// Lesser General Public License for more details.
//LIC//
//LIC// You should have received a copy of the GNU Lesser General Public
//LIC// License along with this library; if not, write to the Free Software
//LIC// Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
//LIC// 02110-1301 USA.
//LIC//
//LIC// The authors may be contacted at oomph-lib@maths.man.ac.uk.
//LIC//
//LIC//====================================================================
#include <iostream>
// Generic oomph-lib includes
#include "generic.h"
#include "navier_stokes.h"
#include "beam.h"

// The wall mesh
#include "meshes/one_d_lagrangian_mesh.h"

//Include the fluid mesh
#include "meshes/collapsible_channel_mesh.h"

using namespace std;

using namespace oomph;
```

```
//==========start_of_BL_Squash ============================================
/// Namespace to define the mapping [0,1] -> [0,1] that re-distributes
/// nodal points across the channel width.
//=========================================================================
namespace BL_Squash
{

 /// Boundary layer width
 double Delta=0.1;

 /// Fraction of points in boundary layer
 double Fract_in_BL=0.5;

 /// \short Mapping [0,1] -> [0,1] that re-distributes
 /// nodal points across the channel width
 double squash_fct(const double& s)
 {
  // Default return
  double y=s;
  if (s<0.5*Fract_in_BL)
   {
    y=Delta*2.0*s/Fract_in_BL;
   }
  else if (s>1.0-0.5*Fract_in_BL)
   {
    y=2.0*Delta/Fract_in_BL*s+1.0-2.0*Delta/Fract_in_BL;
   }
  else
   {
    y=(1.0-2.0*Delta)/(1.0-Fract_in_BL)*s+
      (Delta-0.5*Fract_in_BL)/(1.0-Fract_in_BL);
   }

  return y;
 }
}// end of BL_Squash



////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


//====start_of_underformed_wall=============================================
/// Undeformed wall is a steady, straight 1D line in 2D space
///  \f[ x = X_0 + \zeta \f]
///  \f[ y = H \f]
```

```
//=======================================================================
class UndeformedWall : public GeomObject
{

public:

 /// \short Constructor: arguments are the starting point and the height
 /// above y=0.
 UndeformedWall(const double& x0, const double& h): GeomObject(1,2)
  {
   X0=x0;
   H=h;
  }


 /// \short Position vector at Lagrangian coordinate zeta
 void position(const Vector<double>& zeta, Vector<double>& r) const
  {
   // Position Vector
   r[0] = zeta[0]+X0;
   r[1] = H;
  }


 /// \short Parametrised position on object: r(zeta). Evaluated at
 /// previous timestep. t=0: current time; t>0: previous
 /// timestep. Calls steady version.
 void position(const unsigned& t, const Vector<double>& zeta,
               Vector<double>& r) const
  {
   // Use the steady version
   position(zeta,r);

  } // end of position


 /// \short Posn vector and its  1st & 2nd derivatives
 /// w.r.t. to coordinates:
 /// \f$ \frac{dR_i}{d \zeta_\alpha}\f$ = drdzeta(alpha,i).
 /// \f$ \frac{d^2R_i}{d \zeta_\alpha d \zeta_\beta}\f$ =
 /// ddrdzeta(alpha,beta,i). Evaluated at current time.
 virtual void d2position(const Vector<double>& zeta,
                         Vector<double>& r,
                         DenseMatrix<double> &drdzeta,
                         RankThreeTensor<double> &ddrdzeta) const
  {
   // Position vector
   r[0] = zeta[0]+X0;
   r[1] = H;
```

```
    // Tangent vector
    drdzeta(0,0)=1.0;
    drdzeta(0,1)=0.0;

    // Derivative of tangent vector
    ddrdzeta(0,0,0)=0.0;
    ddrdzeta(0,0,1)=0.0;

   } // end of d2position

  private :

  /// x position of the undeformed beam's left end.
  double X0;

  /// Height of the undeformed wall above y=0.
  double H;

}; //end_of_undeformed_wall




//====start_of_physical_parameters=======================
/// Namespace for phyical parameters
//=======================================================
namespace Global_Physical_Variables
{
 /// Reynolds number
 double Re=10.0;

 /// Womersley = Reynolds times Strouhal
 double ReSt=10;

 /// Default pressure on the left boundary
 double P_up=0.0;

 /// Traction applied on the fluid at the left (inflow) boundary, chosen to be
 void prescribed_traction(const double& t,
                          const Vector<double>& x,
                          const Vector<double>& n,
                          Vector<double>& traction)
 {
  traction.resize(2);
  traction[0]=P_up*(1+sin(t*14.5));
```

```
  traction[1]=0.0;

 } //end traction

 /// Non-dimensional wall thickness. As in Jensen & Heil (2003) paper.
 double H=1.0e-2;

 /// 2nd Piola Kirchhoff pre-stress. As in Jensen & Heil (2003) paper.
 double Sigma0=1.0e3;

 /// External pressure
 double P_ext=0.0;

 /// \short Load function: Apply a constant external pressure to the wall.
 /// Note:  This is the load without the fluid contribution!
 /// Fluid load gets added on by FSIWallElement.
 void load(const Vector<double>& xi, const Vector<double>& x,
           const Vector<double>& N, Vector<double>& load)
 {
  for(unsigned i=0;i<2;i++)
   {
    load[i] = -P_ext*N[i];
   }
 } //end of load


 /// \short Fluid structure interaction parameter: Ratio of stresses used for
 /// non-dimensionalisation of fluid to solid stresses.
 double Q=1.0e-5;


} // end of namespace




//====start_of_problem_class=============================================
///Problem class
//=======================================================================
template <class ELEMENT>
class FSICollapsibleChannelProblem : public Problem
{

 public :

/// \short Constructor: The arguments are the number of elements and
/// the lengths of the domain.
 FSICollapsibleChannelProblem(const unsigned& nup,
                              const unsigned& ncollapsible,
```

```
                                const unsigned& ndown,
                                const unsigned& ny,
                                const double& lup,
                                const double& lcollapsible,
                                const double& ldown,
                                const double& ly);

  /// Destructor (empty)
  ~FSICollapsibleChannelProblem(){}

#ifdef MACRO_ELEMENT_NODE_UPDATE

  /// Access function for the specific bulk (fluid) mesh
  MacroElementNodeUpdateRefineableCollapsibleChannelMesh<ELEMENT>* bulk_mesh_pt
   {
    // Upcast from pointer to the Mesh base class to the specific
    // element type that we're using here.
    return dynamic_cast<
     MacroElementNodeUpdateRefineableCollapsibleChannelMesh<ELEMENT>*>
     (Bulk_mesh_pt);
   }

#else

  /// Access function for the specific bulk (fluid) mesh
  RefineableAlgebraicCollapsibleChannelMesh<ELEMENT>* bulk_mesh_pt()
   {
    // Upcast from pointer to the Mesh base class to the specific
    // element type that we're using here.
    return dynamic_cast<
     RefineableAlgebraicCollapsibleChannelMesh<ELEMENT>*>
     (Bulk_mesh_pt);
   }

#endif


  /// Access function for the wall mesh
  OneDLagrangianMesh<FSIHermiteBeamElement>* wall_mesh_pt()
   {
    return Wall_mesh_pt;

   } // end of access to wall mesh


  /// Actions before adapt: Wipe the mesh of prescribed traction elements
  void actions_before_adapt();

  /// Actions after adapt: Rebuild the mesh of prescribed traction elements
```

```
 /// and reset FSI
 void actions_after_adapt();

 /// \short Update the problem specs before solve (empty)
 void actions_before_newton_solve() {}

 /// Update the problem after solve (empty)
 void actions_after_newton_solve(){}

 /// \short Update before checking Newton convergence: Update the
 /// nodal positions in the fluid mesh in response to possible
 /// changes in the wall shape
 void actions_before_newton_convergence_check()
  {
   Bulk_mesh_pt->node_update();
  }

 /// Doc the solution
 void doc_solution(DocInfo& doc_info,ofstream& trace_file);

 /// Apply initial conditions
 void set_initial_condition();

private :

 /// Create the prescribed traction elements on boundary b
 void create_traction_elements(const unsigned &b,
                               Mesh* const &bulk_mesh_pt,
                               Mesh* const &traction_mesh_pt);

 /// Delete prescribed traction elements from the surface mesh
 void delete_traction_elements(Mesh* const &traction_mesh_pt);

 ///Number of elements in the x direction in the upstream part of the channel
 unsigned Nup;

 /// \short Number of elements in the x direction in the collapsible part of
 /// the channel
 unsigned Ncollapsible;

 ///Number of elements in the x direction in the downstream part of the channe
 unsigned Ndown;

 ///Number of elements across the channel
 unsigned Ny;

 ///x-length in the upstream part of the channel
 double Lup;
```

```
 ///x-length in the collapsible part of the channel
 double Lcollapsible;

 ///x-length in the downstream part of the channel
 double Ldown;

 ///Transverse length
 double Ly;

#ifdef MACRO_ELEMENT_NODE_UPDATE

 /// Pointer to the "bulk" mesh
 MacroElementNodeUpdateRefineableCollapsibleChannelMesh<ELEMENT>* Bulk_mesh_pt

#else

 /// Pointer to the "bulk" mesh
 RefineableAlgebraicCollapsibleChannelMesh<ELEMENT>* Bulk_mesh_pt;

#endif

 /// \short Pointer to the "surface" mesh that applies the traction at the
 /// inflow
 Mesh* Applied_fluid_traction_mesh_pt;

 /// Pointer to the "wall" mesh
 OneDLagrangianMesh<FSIHermiteBeamElement>* Wall_mesh_pt;

 ///Pointer to the left control node
 Node* Left_node_pt;

 ///Pointer to right control node
 Node* Right_node_pt;

 /// Pointer to control node on the wall
 Node* Wall_node_pt;

};//end of problem class




//=====start_of_constructor=======================================
/// Constructor for the collapsible channel problem
//================================================================
template <class ELEMENT>
FSICollapsibleChannelProblem<ELEMENT>::FSICollapsibleChannelProblem(
 const unsigned& nup,
 const unsigned& ncollapsible,
```

```
 const unsigned& ndown,
 const unsigned& ny,
 const double& lup,
 const double& lcollapsible,
 const double& ldown,
 const double& ly)
{
 // Store problem parameters
 Nup=nup;
 Ncollapsible=ncollapsible;
 Ndown=ndown;
 Ny=ny;
 Lup=lup;
 Lcollapsible=lcollapsible;
 Ldown=ldown;
 Ly=ly;


 // Overwrite maximum allowed residual to accomodate bad initial guesses
 Problem::Max_residuals=1000.0;

 // Allocate the timestepper for the Navier-Stokes equations
 BDF<2>* fluid_time_stepper_pt=new BDF<2>;

 // Add the fluid timestepper to the Problem's collection of timesteppers.
 add_time_stepper_pt(fluid_time_stepper_pt);

 // Create a dummy Steady timestepper that stores two history values
 Steady<2>* wall_time_stepper_pt = new Steady<2>;

 // Add the wall timestepper to the Problem's collection of timesteppers.
 add_time_stepper_pt(wall_time_stepper_pt);

 // Geometric object that represents the undeformed wall:
 // A straight line at height y=ly; starting at x=lup.
 UndeformedWall* undeformed_wall_pt=new UndeformedWall(lup,ly);

 //Create the "wall" mesh with FSI Hermite beam elements, passing the
 //dummy wall timestepper to the constructor
 Wall_mesh_pt = new OneDLagrangianMesh<FSIHermiteBeamElement>
  (Ncollapsible,Lcollapsible,undeformed_wall_pt,wall_time_stepper_pt);

 // Build a geometric object (one Lagrangian, two Eulerian coordinates)
 // from the wall mesh
 MeshAsGeomObject* wall_geom_object_pt=
  new MeshAsGeomObject(Wall_mesh_pt);

#ifdef MACRO_ELEMENT_NODE_UPDATE
```

```
  //Build bulk (fluid) mesh
  Bulk_mesh_pt =
   new MacroElementNodeUpdateRefineableCollapsibleChannelMesh<ELEMENT>
   (nup, ncollapsible, ndown, ny,
    lup, lcollapsible, ldown, ly,
    wall_geom_object_pt,
    fluid_time_stepper_pt);

  // Set a non-trivial boundary-layer-squash function...
  Bulk_mesh_pt->bl_squash_fct_pt() = &BL_Squash::squash_fct;

  // ... and update the nodal positions accordingly
  Bulk_mesh_pt->node_update();

#else

  //Build bulk (fluid) mesh
  Bulk_mesh_pt =
   new RefineableAlgebraicCollapsibleChannelMesh<ELEMENT>
   (nup, ncollapsible, ndown, ny,
    lup, lcollapsible, ldown, ly,
    wall_geom_object_pt,
    &BL_Squash::squash_fct,
    fluid_time_stepper_pt);

#endif

  if (CommandLineArgs::Argc>1)
   {
    // One round of uniform refinement
    Bulk_mesh_pt->refine_uniformly();
   }

  // Create "surface mesh" that will contain only the prescribed-traction
  // elements. The constructor just creates the mesh without
  // giving it any elements, nodes, etc.
  Applied_fluid_traction_mesh_pt = new Mesh;

  // Create prescribed-traction elements from all elements that are
  // adjacent to boundary 5 (left boundary), but add them to a separate mesh.
  create_traction_elements(5,Bulk_mesh_pt,Applied_fluid_traction_mesh_pt);

  // Add the sub meshes to the problem
  add_sub_mesh(Bulk_mesh_pt);
  add_sub_mesh(Applied_fluid_traction_mesh_pt);
  add_sub_mesh(Wall_mesh_pt);

  // Combine all submeshes into a single Mesh
  build_global_mesh();
```

```
//Set errror estimator
Z2ErrorEstimator* error_estimator_pt=new Z2ErrorEstimator;
bulk_mesh_pt()->spatial_error_estimator_pt()=error_estimator_pt;


// Complete build of fluid mesh
//------------------------------

// Loop over the elements to set up element-specific
// things that cannot be handled by constructor
unsigned n_element=Bulk_mesh_pt->nelement();
for(unsigned e=0;e<n_element;e++)
 {
  // Upcast from GeneralisedElement to the present element
  ELEMENT* el_pt = dynamic_cast<ELEMENT*>(Bulk_mesh_pt->element_pt(e));

  //Set the Reynolds number
  el_pt->re_pt() = &Global_Physical_Variables::Re;

  // Set the Womersley number
  el_pt->re_st_pt() = &Global_Physical_Variables::ReSt;

 } // end loop over elements


// Pin redudant pressure dofs
RefineableNavierStokesEquations<2>::
 pin_redundant_nodal_pressures(Bulk_mesh_pt->element_pt());


// Apply boundary conditions for fluid
//------------------------------------


//Pin the velocity on the boundaries
//x and y-velocities pinned along boundary 0 (bottom boundary) :
unsigned ibound=0;
unsigned num_nod= bulk_mesh_pt()->nboundary_node(ibound);
for (unsigned inod=0;inod<num_nod;inod++)
 {
  for(unsigned i=0;i<2;i++)
   {
    bulk_mesh_pt()->boundary_node_pt(ibound, inod)->pin(i);
   }
 }

//x and y-velocities pinned along boundaries 2, 3, 4 (top boundaries) :
for(ibound=2;ibound<5;ibound++)
```

```
 {
  num_nod= bulk_mesh_pt()->nboundary_node(ibound);
  for (unsigned inod=0;inod<num_nod;inod++)
   {
    for(unsigned i=0;i<2;i++)
     {
      bulk_mesh_pt()->boundary_node_pt(ibound, inod)->pin(i);
     }
   }
 }

//y-velocity pinned along boundary 1 (right boundary):
ibound=1;
num_nod= bulk_mesh_pt()->nboundary_node(ibound);
for (unsigned inod=0;inod<num_nod;inod++)
 {
  bulk_mesh_pt()->boundary_node_pt(ibound, inod)->pin(1);
 }


//y-velocity pinned along boundary 5 (left boundary):
ibound=5;
num_nod= bulk_mesh_pt()->nboundary_node(ibound);
for (unsigned inod=0;inod<num_nod;inod++)
 {

  bulk_mesh_pt()->boundary_node_pt(ibound, inod)->pin(1);

 }//end of pin_velocity



// Complete build of applied traction elements
//=====================================================

// Loop over the traction elements to pass pointer to prescribed
// traction function
unsigned n_el=Applied_fluid_traction_mesh_pt->nelement();
for(unsigned e=0;e<n_el;e++)
 {
  // Upcast from GeneralisedElement to NavierStokes traction element
  NavierStokesTractionElement<ELEMENT> *el_pt =
   dynamic_cast< NavierStokesTractionElement<ELEMENT>*>(
    Applied_fluid_traction_mesh_pt->element_pt(e));

  // Set the pointer to the prescribed traction function
  el_pt->traction_fct_pt() = &Global_Physical_Variables::prescribed_traction;

 }
```

```
// Complete build of wall elements
//-------------------------------------

//Loop over the elements to set physical parameters etc.
n_element = wall_mesh_pt()->nelement();
for(unsigned e=0;e<n_element;e++)
 {
  // Upcast to the specific element type
  FSIHermiteBeamElement *elem_pt =
   dynamic_cast<FSIHermiteBeamElement*>(wall_mesh_pt()->element_pt(e));

  // Set physical parameters for each element:
  elem_pt->sigma0_pt() = &Global_Physical_Variables::Sigma0;
  elem_pt->h_pt() = &Global_Physical_Variables::H;

  // Set the load vector for each element
  elem_pt->load_vector_fct_pt() = &Global_Physical_Variables::load;

  // Function that specifies the load ratios
  elem_pt->q_pt() = &Global_Physical_Variables::Q;

  // Set the undeformed shape for each element
  elem_pt->undeformed_beam_pt() = undeformed_wall_pt;

  // The normal on the wall elements as computed by the FSIHermiteElements
  // points away from the fluid rather than into the fluid (as assumed
  // by default)
  elem_pt->set_normal_pointing_out_of_fluid();

 } // end of loop over elements



// Boundary conditions for wall mesh
//----------------------------------

// Set the boundary conditions: Each end of the beam is fixed in space
// Loop over the boundaries (ends of the beam)
for(unsigned b=0;b<2;b++)
 {
  // Pin displacements in both x and y directions
  wall_mesh_pt()->boundary_node_pt(b,0)->pin_position(0);
  wall_mesh_pt()->boundary_node_pt(b,0)->pin_position(1);
 }
```

```
//Choose control nodes
//---------------------

// Left boundary
ibound=5;
num_nod= bulk_mesh_pt()->nboundary_node(ibound);
unsigned control_nod=num_nod/2;
Left_node_pt= bulk_mesh_pt()->boundary_node_pt(ibound, control_nod);

// Right boundary
ibound=1;
num_nod= bulk_mesh_pt()->nboundary_node(ibound);
control_nod=num_nod/2;
Right_node_pt= bulk_mesh_pt()->boundary_node_pt(ibound, control_nod);


// Set the pointer to the control node on the wall
num_nod= wall_mesh_pt()->nnode();
Wall_node_pt=wall_mesh_pt()->node_pt(Ncollapsible/2);


// Setup FSI
//----------

// The velocity of the fluid nodes on the wall (fluid mesh boundary 3)
// is set by the wall motion -- hence the no-slip condition needs to be
// re-applied whenever a node update is performed for these nodes.
// Such tasks may be performed automatically by the auxiliary node update
// function specified by a function pointer:
ibound=3;
num_nod= bulk_mesh_pt()->nboundary_node(ibound);
for (unsigned inod=0;inod<num_nod;inod++)
 {
  bulk_mesh_pt()->boundary_node_pt(ibound, inod)->
   set_auxiliary_node_update_fct_pt(
    FSI_functions::apply_no_slip_on_moving_wall);
 }



// Work out which fluid dofs affect the residuals of the wall elements:
// We pass the boundary between the fluid and solid meshes and
// pointers to the meshes. The interaction boundary is boundary 3 of the
// 2D fluid mesh.
FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
 (this,3,Bulk_mesh_pt,Wall_mesh_pt);
```

```
 // Setup equation numbering scheme
 cout <<"Number of equations: " << assign_eqn_numbers() << std::endl;


}//end of constructor




//====start_of_doc_solution=================================================
/// Doc the solution
//==========================================================================
template <class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>:: doc_solution(DocInfo& doc_info,
                                                          ofstream& trace_file
{

#ifdef MACRO_ELEMENT_NODE_UPDATE

 // Doc fsi
 if (CommandLineArgs::Argc>1)
  {
   FSI_functions::doc_fsi<MacroElementNodeUpdateNode>
    (Bulk_mesh_pt,Wall_mesh_pt,doc_info);
  }

#else

 // Doc fsi
 if (CommandLineArgs::Argc>1)
  {
   FSI_functions::doc_fsi<AlgebraicNode>(Bulk_mesh_pt,Wall_mesh_pt,doc_info);
  }

#endif

 ofstream some_file;
 char filename[100];

 // Number of plot points
 unsigned npts;
 npts=5;

//Do not output the whole solution for the fluid and the wall
 /*
 // Output fluid solution
 sprintf(filename,"%s/soln%i.dat",doc_info.directory().c_str(),
         doc_info.number());
 some_file.open(filename);
```

```
 bulk_mesh_pt()->output(some_file,npts);
 some_file.close();

 // Document the wall shape
 sprintf(filename,"%s/beam%i.dat",doc_info.directory().c_str(),
         doc_info.number());
 some_file.open(filename);
 wall_mesh_pt()->output(some_file,npts);
 some_file.close();

 // Loop over all elements do dump out previous solutions
 // (get the number of previous timesteps available from the wall
 // timestepper)
 unsigned nsteps=time_stepper_pt(1)->nprev_values();
 for (unsigned t=0;t<=nsteps;t++)
  {
   sprintf(filename,"%s/wall%i-%i.dat",doc_info.directory().c_str(),
           doc_info.number(),t);
   some_file.open(filename);
   unsigned n_elem=wall_mesh_pt()->nelement();
   for (unsigned ielem=0;ielem<n_elem;ielem++)
    {
     dynamic_cast<FSIHermiteBeamElement*>(wall_mesh_pt()->element_pt(ielem))->
      output(t,some_file,npts);
    }
   some_file.close();
  } // end of output of previous solutions
  */




 // Write trace file
 trace_file << time_pt()->time() << " "
            << Wall_node_pt->x(1) << " "
            << Left_node_pt->value(0) << " "
            << Right_node_pt->value(0) << " "
            << Global_Physical_Variables::P_ext  << " "
            << std::endl;

} // end_of_doc_solution




//=====start_of_create_traction_elements=================================
/// Create the traction elements
//========================================================================
```

```
template <class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>::create_traction_elements(
 const unsigned &b, Mesh* const &bulk_mesh_pt, Mesh* const &traction_mesh_pt)
{

 // How many bulk elements are adjacent to boundary b?
 unsigned n_element = bulk_mesh_pt->nboundary_element(b);

 // Loop over the bulk elements adjacent to boundary b?
 for(unsigned e=0;e<n_element;e++)
  {
   // Get pointer to the bulk element that is adjacent to boundary b
   ELEMENT* bulk_elem_pt = dynamic_cast<ELEMENT*>
    (bulk_mesh_pt->boundary_element_pt(b,e));

   //What is the index of the face of element e along boundary
   int face_index = bulk_mesh_pt->face_index_at_boundary(b,e);

   // Build the corresponding prescribed-traction element
   NavierStokesTractionElement<ELEMENT>* flux_element_pt =
    new NavierStokesTractionElement<ELEMENT>(bulk_elem_pt,face_index);

   //Add the prescribed-traction element to the surface mesh
   traction_mesh_pt->add_element_pt(flux_element_pt);

  } //end of loop over bulk elements adjacent to boundary b

} // end of create_traction_elements



//==========start_of_delete_traction_elements============================
/// Delete traction elements and wipe the surface mesh
//========================================================================
template<class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>::
delete_traction_elements(Mesh* const &surface_mesh_pt)
{
 // How many surface elements are in the surface mesh
 unsigned n_element = surface_mesh_pt->nelement();

 // Loop over the surface elements
 for(unsigned e=0;e<n_element;e++)
  {
   // Kill surface element
   delete surface_mesh_pt->element_pt(e);
  }

 // Wipe the mesh
```

```
 surface_mesh_pt->flush_element_and_node_storage();

} // end of delete_traction_elements



//====start_of_apply_initial_condition=========================================
/// Apply initial conditions
//============================================================================
template <class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>::set_initial_condition()
{
 // Check that timestepper is from the BDF family
 if (time_stepper_pt()->type()!="BDF")
  {
   std::ostringstream error_stream;
   error_stream << "Timestepper has to be from the BDF family!\n"
                << "You have specified a timestepper from the "
                << time_stepper_pt()->type() << " family" << std::endl;

   throw OomphLibError(error_stream.str(),
                       OOMPH_CURRENT_FUNCTION,
                       OOMPH_EXCEPTION_LOCATION);
  }

 // Update the mesh
 bulk_mesh_pt()->node_update();

 // Loop over the nodes to set initial guess everywhere
 unsigned num_nod = bulk_mesh_pt()->nnode();
 for (unsigned n=0;n<num_nod;n++)
  {
   // Get nodal coordinates
   Vector<double> x(2);
   x[0]=bulk_mesh_pt()->node_pt(n)->x(0);
   x[1]=bulk_mesh_pt()->node_pt(n)->x(1);

   // Assign initial condition: Steady Poiseuille flow
   bulk_mesh_pt()->node_pt(n)->set_value(0,6.0*(x[1]/Ly)*(1.0-(x[1]/Ly)));
   bulk_mesh_pt()->node_pt(n)->set_value(1,0.0);
  }

 // Assign initial values for an impulsive start
 bulk_mesh_pt()->assign_initial_values_impulsive();
 wall_mesh_pt()->assign_initial_values_impulsive();

} // end of set_initial_condition
```

```
//========start_of_actions_before_adapt=======================================
/// Actions before adapt: Wipe the mesh of prescribed traction elements
//============================================================================
template<class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>::actions_before_adapt()
{
 // Kill the traction elements and wipe surface mesh
 delete_traction_elements(Applied_fluid_traction_mesh_pt);

 // Rebuild the global mesh.
 rebuild_global_mesh();

} // end of actions_before_adapt



//========start_of_actions_after_adapt=======================================
/// Actions after adapt: Rebuild the mesh of prescribed traction elements
//============================================================================
template<class ELEMENT>
void FSICollapsibleChannelProblem<ELEMENT>::actions_after_adapt()
{
 // Create prescribed-flux elements from all elements that are
 // adjacent to boundary 5 and add them to surface mesh
 create_traction_elements(5,Bulk_mesh_pt,Applied_fluid_traction_mesh_pt);

 // Rebuild the global mesh
 rebuild_global_mesh();

 // Unpin all pressure dofs
 RefineableNavierStokesEquations<2>::
  unpin_all_pressure_dofs(Bulk_mesh_pt->element_pt());

 // Pin redundant pressure dofs
 RefineableNavierStokesEquations<2>::
  pin_redundant_nodal_pressures(Bulk_mesh_pt->element_pt());

 // Loop over the traction elements to pass pointer to prescribed
 // traction function
 unsigned n_element=Applied_fluid_traction_mesh_pt->nelement();
 for(unsigned e=0;e<n_element;e++)
  {
   // Upcast from GeneralisedElement to NavierStokesTractionElement element
   NavierStokesTractionElement<ELEMENT> *el_pt =
    dynamic_cast<NavierStokesTractionElement<ELEMENT>*>(
     Applied_fluid_traction_mesh_pt->element_pt(e));
```

```
    // Set the pointer to the prescribed traction function
    el_pt->traction_fct_pt() = &Global_Physical_Variables::prescribed_traction;
   }

  // The functions used to update the no slip boundary conditions
  // must be set on any new nodes that have been created during the
  // mesh adaptation process.
  // There is no mechanism by which auxiliary update functions
  // are copied to newly created nodes.
  // (because, unlike boundary conditions, they don't occur exclusively
  // at boundaries)

  // The velocity of the fluid nodes on the wall (fluid mesh boundary 3)
  // is set by the wall motion -- hence the no-slip condition needs to be
  // re-applied whenever a node update is performed for these nodes.
  // Such tasks may be performed automatically by the auxiliary node update
  // function specified by a function pointer:
  unsigned ibound=3;
  unsigned num_nod= bulk_mesh_pt()->nboundary_node(ibound);
  for (unsigned inod=0;inod<num_nod;inod++)
   {
    bulk_mesh_pt()->boundary_node_pt(ibound, inod)->
     set_auxiliary_node_update_fct_pt(
      FSI_functions::apply_no_slip_on_moving_wall);
   }

  // (Re-)setup fsi: Work out which fluid dofs affect wall elements
  // the correspondance between wall dofs and fluid elements is handled
  // during the remeshing, but the "reverse" association must be done
  // separately. We need to set up the interaction every time because the fluid
  // element adjacent to a given solid element's integration point may have
  // changed.We pass the boundary between the fluid and solid meshes and
  // pointers to the meshes. The interaction boundary is boundary 3 of
  // the Fluid mesh.
  FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
   (this,3,Bulk_mesh_pt,Wall_mesh_pt);


} // end of actions_after_adapt



//==========start_of_main=============================================
/// Driver code for a collapsible channel problem with FSI.
/// Presence of command line arguments indicates validation run with
/// coarse resolution and small number of timesteps.
//====================================================================
int main(int argc, char* argv[])
```

```
{

 // Store command line arguments
 CommandLineArgs::setup(argc,argv);

 // Reduction in resolution for validation run?
 unsigned coarsening_factor=1;
 if (CommandLineArgs::Argc>1)
  {
   coarsening_factor=4;
  }

 // Number of elements in the domain
 unsigned nup=20/coarsening_factor;
 unsigned ncollapsible=40/coarsening_factor;
 unsigned ndown=40/coarsening_factor;
 unsigned ny=16/coarsening_factor;

 // Length of the domain
 double lup=5.0;
 double lcollapsible=10.0;
 double ldown=10.0;
 double ly=1.0;

 // Set external pressure (on the wall stiffness scale).
 Global_Physical_Variables::P_ext = 1.0e-1;

 // Pressure on the left boundary: This is consistent with steady
 // Poiseuille flow
 Global_Physical_Variables::P_up=12.0*(lup+lcollapsible+ldown);



//Sweep for Strouhal parameter
for(int i = 0; i < 5; ++i){
     Global_Physical_Variables::ReSt = Global_Physical_Variables::Re*(2.6+0.2*i

      // Timestep. Note: chose to have about 40 points per period
     double dt=Global_Physical_Variables::Re/(7*Global_Physical_Variables::ReSt

     // Initial time fo0.0r the simulation
     double t_min=0.0;

     // Maximum time for simulation
     double t_max= 50;

     #ifdef MACRO_ELEMENT_NODE_UPDATE

     #ifdef TAYLOR_HOOD
```

```
// Build the problem with QTaylorHoodElements
FSICollapsibleChannelProblem
<MacroElementNodeUpdateElement<RefineableQTaylorHoodElement<2> > >
problem(nup, ncollapsible, ndown, ny,
        lup, lcollapsible, ldown, ly);

#else

// Build the problem with QCrouzeixRaviartElements
FSICollapsibleChannelProblem
<MacroElementNodeUpdateElement<RefineableQCrouzeixRaviartElement<2> > >
problem(nup, ncollapsible, ndown, ny,
        lup, lcollapsible, ldown, ly);

#endif

#else

#ifdef TAYLOR_HOOD

// Build the problem with QTaylorHoodElements
FSICollapsibleChannelProblem
<AlgebraicElement<RefineableQTaylorHoodElement<2> > >
problem(nup, ncollapsible, ndown, ny,
        lup, lcollapsible, ldown, ly);

#else

// Build the problem with QCrouzeixRaviartElements
FSICollapsibleChannelProblem
<AlgebraicElement<RefineableQCrouzeixRaviartElement<2> > >
problem(nup, ncollapsible, ndown, ny,
        lup, lcollapsible, ldown, ly);

#endif

#endif



// Initialise timestep
problem.time_pt()->time()=t_min;
problem.initialise_dt(dt);

// Apply initial condition
problem.set_initial_condition();

//Set output directory
```

```
DocInfo doc_info;
char direcu[100];
//sprintf(direcu,"RESLT%d",i);
sprintf(direcu,"RESLT");

doc_info.set_directory(direcu);

// Open a trace file
ofstream trace_file;
char filename[100];
sprintf(filename,"%s/trace_Re%g_ReSt%g.dat",doc_info.directory().c_str(),C
trace_file.open(filename);

// Output the initial condition
problem.doc_solution(doc_info, trace_file);

// Increment step number
doc_info.number()++;

// Find number of timesteps (reduced for validation)
unsigned nstep = unsigned((t_max-t_min)/dt);
if (CommandLineArgs::Argc>1)
{
nstep=3;
}


// Set targets for spatial adaptivity
problem.bulk_mesh_pt()->max_permitted_error()=1.0e-3;
problem.bulk_mesh_pt()->min_permitted_error()=1.0e-5;

// Overwrite for validation run
if (CommandLineArgs::Argc>1)
{
// Set targets for spatial adaptivity
problem.bulk_mesh_pt()->max_permitted_error()=0.5e-2;
problem.bulk_mesh_pt()->min_permitted_error()=0.5e-4;
}

// When performing the first timestep, we can adapt the mesh as many times
// as we want because the initial condition can be re-set
unsigned max_adapt=3;
bool first=true;

// Timestepping loop
for (unsigned istep=0;istep<nstep;istep++)
{
// Solve the problem
problem.unsteady_newton_solve(dt,max_adapt,first);
```

```
      // Outpt the solution
      problem.doc_solution(doc_info, trace_file);

      // Step number
      doc_info.number()++;

      // We've done the first step
      first=false;
      max_adapt=1;
      }


      // Close trace file.
      trace_file.close();
}

}//end of main
```