Testing and optimization of Unicorn Fluid-Structure Interaction solver for simulating an industrial problem

THI THANH TRUC VU

Master of Science Thesis
Stockholm, Sweden 2014

Testing and optimization of Unicorn Fluid-Structure Interaction solver for simulating an industrial problem

T H I T H A N H T R U C VU

Master's Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Computer simulation for Science and Engineering (120 credits)
Royal Institute of Technology year 2014
Supervisor at KTH was Johan Hoffman
Supervisor at Vattenfall was Luca Facciolo
Examiner was Michael Hanke

TRITA-MAT-E 2014:01
ISRN-KTH/MAT/E--14/01--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

In industry applications, such as power supply plants, the issue of interaction between fluid and structure is always presented. More precisely, the fluid flow affects the structure by applying force(s) on it and vice versa. As a result, the structure can move (vibrate) or deform. A good understanding of this problem can help to design the system in term of safety, stability and efficiency.

This project aims to optimize and test the Unicorn FSI solver from the FEniCS project [1] to simulate the interaction of fluid and structure in an experiment, which was carried out at Vattenfall Research and Development. The target is to improve the Unicorn FSI solver to cope with a real industrial problem. Moreover, some results of the simulation can be used as a tool to predict the behaviour of a system under the effect of fluid flow.

Referat

Testning och optimering av Unicorn Fluid-Structure Interaction lösare för att simulera ett industriellt problem

I industriapplikationer, såsom kraftverk, är frågan om samspelet mellan fluid och struktur alltid närvarande. Närmare bestämt påverkar fluiden kraftverkets struktur genom att applicera en kraft på det och vice versa. Som ett resultat av fluidens kraftpåverkan, kan kraftverkets struktur vibrera eller deformeras. En god förståelse för detta FSI problem kan bidra till att utforma system ifråga om säkerhet, stabilitet och effektivitet.

Detta projekt syftar till att optimera och testa Unicorn FSI lösaren från FEniCS projektet. Denna FSI lösare ska därefter användas till att simulera samspelet mellan vätska och struktur i ett experiment, som utförts på Vattenfalls forsknings och utvecklingsavdeling. Målet är att förbättra Unicorn FSI-lösaren för att klara av ett verkligt industriellt problem. Dessutom kan vissa resultaten av simuleringen användas som ett verktyg för att förutsäga beteendet hos ett system under inverkan av en fluid.

Acknowledgement

First I would like to thank Prof. Johan Hoffman for support, help and valuable advices during this project, for kindly reading my report and offering detailed suggestions on the outline, content and grammar of this report.
Second I would like to express my appreciation to Rodrigo Vilela De Abreu for encouragement, help with the software platform, and useful discussions during the project.
In addition, I sincerely thank Jeannette Spühler and Cem Degirmenci for invaluable discussions and advices in the coding phase.
Furthermore, a thank you to Dr. Luca Facciolo for giving me a good opportunity to work with this industrial project and for discussions in the mechanic side as well as the numerical side of the project.

Contents

Chapter 1

Introduction

1.1 Fluid-Structure Interaction

Depending on the velocity of the flow, structures immersed into fluid flow are caused to vibrate. If the vibration amplitude is small, this eventually can lead to structure cracking in the long term. If this amplitude is large, damage may happen in the short term. Furthermore, self-excited vibrations [2] may also occur if the flow velocity overpasses some threshold. Self-excited vibration means that the oscillation amplitude grows exponentially in time until it reaches to a limit of further increase.

Fluid-Structure Interaction (FSI) can be analysed by experimental measurements or numerical simulations. There are different computational methods to treat the fluid and structure: simultaneously (a monolithic method) or partitioned (a coupled method). In [3] readers can find the description of two approaches together with their drawbacks and advantages. The main idea of a coupled method is to use the boundary conditions between the two phases as an interface to update input from the fluid solver and the solid solver in turn, i.e. in each time step one uses the solution of the *fluid* solver as boundary conditions to update and solve the *solid* equations and vice versa. While in a monolithic method, both governing equations of the fluid and the structure are solved by one single solver.

In this work we used a Unified Continuum model as a monolithic method together with an Arbitrary Lagrangian Eulerian (ALE) formulation in the Unicorn FSI solver under FEniCS [1]. The project focuses on testing and optimization of the solver to adapt to the requirements of simulating a real industrial problem.

1.2 Problem introduction

In real engineering applications fluids and structures seldom operate in isolation. The interaction between fluids and structures, i.e. Fluid-Structure Interaction (FSI), are of great importance at many different levels in the safety and performance analysis. FSI is a very important issue in the power industry, which experiences increasing attention especially within the context of serviceability, durability and

long-term behaviour of the structures.

In nuclear power plants the working fluid (steam or water) exerts continuous loads on the structures that bound and interfere with it. Typical examples are flow-induced vibrations, flow accelerated corrosion, thermal loads, water and steam hammers, valve operations, jet and sprinkling systems, pool and tank dynamics, pipe breaks, etc. Many of the mentioned structures are connected and supported by concrete structures. Therefore it is essential to understand vibratory behaviour of the structure in order to determine the impacts on the supports and the supporting concrete structures. Many of these fluid-structure interaction problems are related to safety aspects of the power plant: the structures undergo stresses that can cause in a short or long time period degradation or a failure of their function.

In wind power farms the loads calculation is of primary importance for the structural integrity and the power output. The forces and the moments acting on the fundaments and on the turbine body are time and place dependent, affecting the performance of the whole wind farm. Furthermore, in the offshore configuration, the sea waves and streams add a supplementary load to the structure. The load and vibration induced to the structure are caused by local impacts on the structural elements. How locally induced impacts are transferred to other parts of the structure is an important issue to address. Moreover, how the impacts affect the fatigue behaviour of the structural elements and their connection to the adjacent elements should be considered.

In hydropower applications the water coming from the turbines exerts powerful dynamic loads on the structures including the draft tubes, outlets and spillways. For instance, it has been observed that vibrations due to the turbulent water flow may damage the concrete structure of draft tubes. Another important issue in this context is vibrations induced to the spillway gates. The gates are normally supported by the surrounding concrete structures. Improved modelling of the dynamic behaviour of the gates may improve the tools for designing the connection between concrete and steel structures in a spillway.

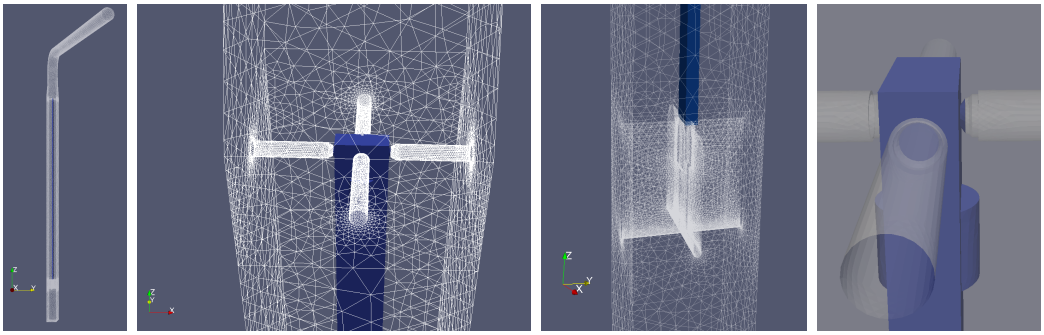
Since Fluid-Structure Interaction is a central aspect in the safety and performance analysis for the present and future power plants, there is a strong need to be able to make more reliable and realistic predictions with the use of proper tools that include a better description of the fluids and their interaction. It is important to calculate the variable loads and the vibration frequencies that the structures have to withstand during their life. A better result will allow a better knowledge of the margin to be adopted in the safety analysis.

1.3 Problem statement

The experiments have been carried out at Vattenfall Research & Development AB. The geometry of this experiment contains a vertical channel, cross section $80mm \times 80mm$, in which there is a thin long bar of size $20mm \times 8mm \times 1500mm$. The bottom part of the bar is fixed in a cross. The top of the bar is free. There are

four cylinders around the top of the bar, each consists of a small ball on a tip. The role of these cylinders and the balls is to keep the bar fixed in the two horizontal directions but to be free to slide in the vertical direction. The distance between the cylinders tip to the bar is $0.6mm$ (visible part of the balls).

Fig. 1.1 shows the geometry of the experiment.



(a) (b) (c) (d)

Figure 1.1: Geometry design: (a) Overview of the structure; (b) Upper part, the top of the bar lays centered between four cylinders; (c) Lower part, the bottom of the bar is fixed in a cross; (d) The ball part between the cylinder and the bar

The following experiments were carried out:

1. The channel was filled with still water,
2. The channel was filled partly with still water,
3. Water was pumped into the channel from below with velocity $1m/s$ and $3m/s$,
4. The channel was filled with still air.

An external force is applied exactly in the middle of the bar by pulling it 10mm away from the center (bending) then let go. The vibration of the bar at the centre will be measured with a laser sensor.

Experimental measurements will be used to validate the simulation result.

In this project, some experiments are simulated. Outputs of interest from numerical simulations are the flow velocity and the pressure inside the channel; frequencies, amplitudes and damping factors of the vibrating bar.

Chapter 2

Unicorn FSI Solver

In this chapter we review the main theory of the Unicorn FSI solver such as the Arbitrary Lagrangian Eulerian coordinate systems, the Unified Continuum model, the Finite Element Method, the Unicorn FSI algorithm and its implementation.

2.1 Arbitrary Lagrangian Eulerian (ALE)

The name Arbitrary Lagrangian Eulerian comes from formulating the solid part in a material (Lagrangian) coordinate system; and the fluid part in a fixed spatial (Eulerian) coordinate system. The mixed description is usually referred to as Arbitrary Lagrangian Eulerian or ALE. The work of Stöckli et. al. [4] gives some examples and comparisons between these two descriptions. In short, a Lagrangian description means that the observer is attached to a material point, he travels along this point and capture the coordinate changes of the point, while in an Eulerian description the observer is anchored somewhere among the flow, and captures through his fixed position: how the flow passes by.

2.1.1 Classical Lagrangian and Eulerian description

The explanation of ALE method below is referenced from the book of Erwin Stein, Ren de Borst and Thomas J.R. Hughes et. al. [5].

The Eulerian coordinate system is fixed in space while the Lagrangian coordinate system moves with the continuum (mesh nodes move).

We introduce the domain $R_X \subset R^n$ with n is spatial dimension, which consists of material particles X, shown as $\triangle$ in Fig. 2.1. The domain $R_x \subset R^n$ consists of spatial points x, shown as $\bigcirc$.

In the Lagrangian description, the material particles X are permanently connected to the spatial points (mesh nodes) x. As time passes, the particles moves. Therefore the mesh nodes should be moved accordingly. Hence, there is a map, namely φ, which links X to x in time. Note that here the motion in time interval $I = [0, T)$ is considered.

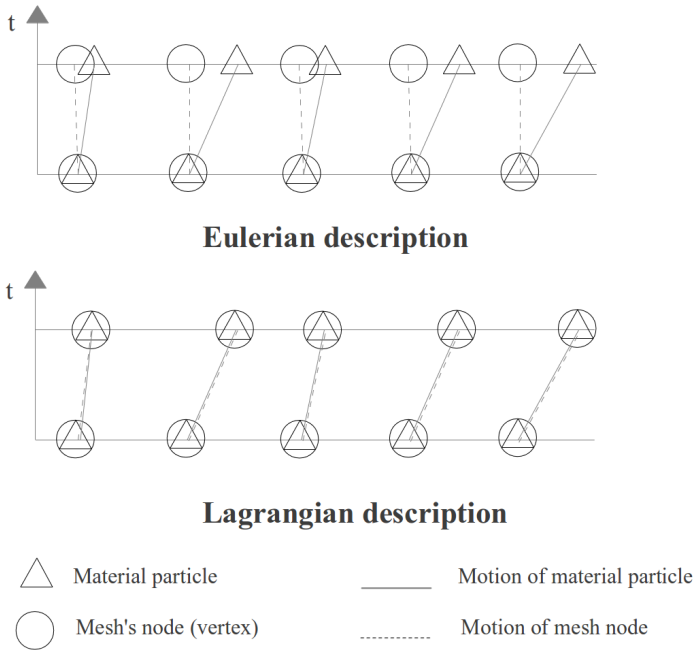


Figure 2.1: 1D example of material motion and mesh motion in the Eulerian and Lagrangian descriptions

$$\begin{aligned} \varphi : R_X \times I &\longrightarrow R_x \times I \\ (X,t) &\longmapsto \varphi(X,t) = (x,t) \end{aligned} \tag{2.1}$$

In one time instant t with given material motion X, the mapping φ defines the shape of the mesh.

In the Eulerian formulation, the velocity u at a given mesh node at time t is the same as the velocity of the material particle, being coincident in that node at that time. Consequently, the velocity u at a given node does not depend on the initial configuration of the continuum nor the material coordinates X, so that $u = u(x,t)$. In [5], the drawbacks and gains of each descriptions are discussed in more details.

2.1.2 ALE description

In an ALE description, the mesh nodes will not be fixed as in Eulerian. They will move in the direction of material particles but not necessarily coincide with them.

The ALE description introduces the third domain R_χ as a Reference domain, where reference coordinates χ are represented to keep track of the mesh nodes. The map from the Reference domain R_χ to R_X and R_x are Ψ and Φ respectively.

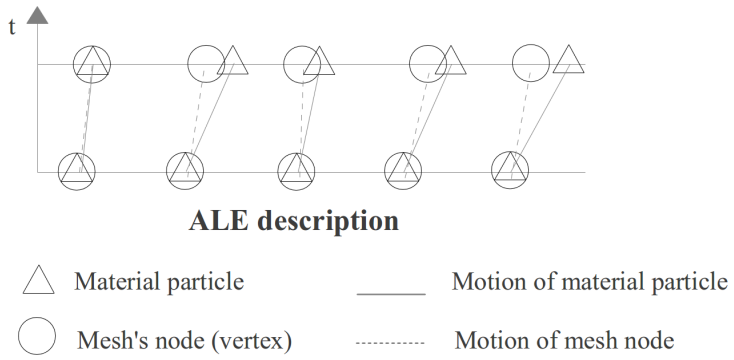


Figure 2.2: 1D example of material motion and mesh motion in the ALE description

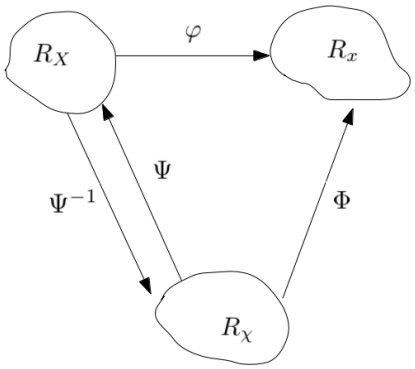


Figure 2.3: Maps between coordinate systems

$$\begin{aligned} \Psi : R_\chi \times I &\longrightarrow R_X \times I \\ (\chi, t) &\longmapsto \Psi(\chi, t) = (X, t) \end{aligned} \tag{2.2}$$

$$\begin{aligned} \Phi : R_\chi \times I &\longrightarrow R_x \times I \\ (\chi, t) &\longmapsto \Phi(\chi, t) = (x, t) \end{aligned} \tag{2.3}$$

The map Φ can be understood as the motion of grid nodes χ in the spatial domain R_x.

To make it easier to understand, here is the summary for some notations:

χ *is the coordinates of one mesh point;*
R_χ *is the mesh points domain*
X *is the coordinates of a material particle;*

R_X is the material particles domain
x is a fixed point coordinate in space;
R_x is the spatial domain.

The inverse map of Ψ is $\Psi^{-1}(X,t) = (\chi, t)$. From Fig. 2.3 it is obvious that

$$\varphi = \Phi \circ \Psi^{-1} \tag{2.4}$$

The gradients of the maps are:

$$\frac{\partial \varphi}{\partial (X,t)} = \begin{pmatrix} \frac{\partial x}{\partial X} & u \\ 0 & 1 \end{pmatrix} \tag{2.5}$$

$$\frac{\partial \Psi^{-1}}{\partial (X,t)} = \begin{pmatrix} \frac{\partial \chi}{\partial X} & \omega \\ 0 & 1 \end{pmatrix} \tag{2.6}$$

$$\frac{\partial \Phi}{\partial (\chi,t)} = \begin{pmatrix} \frac{\partial x}{\partial \chi} & \beta \\ 0 & 1 \end{pmatrix} \tag{2.7}$$

where

$u(X,t) = \left.\frac{\partial x}{\partial t}\right|_X$ *is the velocity of a material particle in the spatial domain*
$\omega(X,t) = \left.\frac{\partial \chi}{\partial t}\right|_X$ *is the velocity of a material particle in the reference domain*
$\beta(\chi,t) = \left.\frac{\partial x}{\partial t}\right|_\chi$ *is the velocity of a grid point in the spatial domain*

From the equation $\varphi = \Phi \circ \Psi^{-1}$ (2.4), we differentiate to get:

$$\begin{aligned} \frac{\partial \varphi}{\partial (X,t)}(X,t) &= \frac{\partial \Phi}{\partial (\chi,t)}(\Psi^{-1}(X,t)) \cdot \frac{\partial \Psi^{-1}}{\partial (X,t)}(X,t) \\ &= \frac{\partial \Phi}{\partial (\chi,t)}(\chi,t) \cdot \frac{\partial \Psi^{-1}}{\partial (X,t)}(X,t) \end{aligned} \tag{2.8}$$

In matrix representation:

$$\begin{pmatrix} \frac{\partial x}{\partial X} & u \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \chi} & \beta \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial \chi}{\partial X} & \omega \\ 0 & 1 \end{pmatrix} \tag{2.9}$$

The inner product in the right hand side is:

$$\begin{pmatrix} \frac{\partial x}{\partial X} & u \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial X} & \frac{\partial x}{\partial \chi}\omega + \beta \\ 0 & 1 \end{pmatrix} \tag{2.10}$$

it follows that

$$\begin{aligned} u &= \frac{\partial x}{\partial \chi}\omega + \beta \\ c := u - \beta &= \frac{\partial x}{\partial \chi}\omega \end{aligned} \tag{2.11}$$

where c is the material velocity relative to the mesh.

The choice $\Psi = I$ makes the Reference domain be identical to the material domain and $X \equiv \chi$. The equation (2.5) and equation (2.7) will then coincide and it yields $u = \beta$, $c = 0$. Physically, this situation means that the mesh nodes move with the same velocities as material points.

In order to characterize a scalar physical quantity, it is described by different functions $f(x,t)$, $f^*(\chi,t)$ and $f^{**}(X,t)$ in spatial, referential and material coordinate systems respectively. According to the mapping of domains above, it holds that:

$$f^{**}(X,t) = f(\varphi(X,t),t) \qquad or \qquad f^{**} = f \circ \varphi \tag{2.12}$$

The gradient of the above equation is:

$$\frac{\partial f^{**}}{\partial (X,t)}(X,t) = \frac{\partial f}{\partial (x,t)}(x,t) \;\; \frac{\partial \varphi}{\partial (X,t)}(X,t) \tag{2.13}$$

In matrix representation:

$$\begin{aligned} \begin{pmatrix} \frac{\partial f^{**}}{\partial X} & \frac{\partial f^{**}}{\partial t} \end{pmatrix} &= \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial t} \end{pmatrix} \begin{pmatrix} \frac{\partial x}{\partial X} & u \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f}{\partial x}\frac{\partial x}{\partial X} & \frac{\partial f}{\partial x}\cdot u + \frac{\partial f}{\partial t} \end{pmatrix} \end{aligned} \tag{2.14}$$

Hence

$$\frac{\partial f^{**}}{\partial t} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\cdot u \tag{2.15}$$

We rewrite the functions f^{**} and f in term of their domains, so that:

$$\left.\frac{\partial f}{\partial t}\right|_X = \left.\frac{\partial f}{\partial t}\right|_x + u \cdot \frac{\partial f}{\partial x} \tag{2.16}$$

This equation expresses the relation of the time derivatives of a function in material and spatial domains. Thus the variation of a physical quantity for a given material point X equals the local variation plus the convective term of change in spatial domain.

Analogously, it also holds:

$$f^{**}(X,t) = f^{*}(\Psi^{-1}(X,t),t) \qquad or \qquad f^{**} = f^{*} \circ \Psi^{-1} \tag{2.17}$$

Gradient:

$$\frac{\partial f^{**}}{\partial (X,t)}(X,t) = \frac{\partial f^{*}}{\partial (\chi,t)}(\chi,t) \;\; \frac{\partial \Psi^{-1}}{\partial (X,t)}(X,t) \tag{2.18}$$

$$\begin{aligned} \begin{pmatrix} \frac{\partial f^{**}}{\partial X} & \frac{\partial f^{**}}{\partial t} \end{pmatrix} &= \begin{pmatrix} \frac{\partial f^{*}}{\partial \chi} & \frac{\partial f^{*}}{\partial t} \end{pmatrix} \begin{pmatrix} \frac{\partial \chi}{\partial X} & \omega \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f^{*}}{\partial \chi}\frac{\partial \chi}{\partial X} & \frac{\partial f^{*}}{\partial \chi}\cdot\omega + \frac{\partial f^{*}}{\partial t} \end{pmatrix} \end{aligned} \tag{2.19}$$

Thus:

$$\frac{\partial f^{**}}{\partial t} = \frac{\partial f^{*}}{\partial t} + \frac{\partial f^{*}}{\partial \chi}\cdot\omega \tag{2.20}$$

This equation expresses the relation between time derivatives in material and reference domains. Using the definition of ω in (2.11) gives:

$$\frac{\partial f^{**}}{\partial t} = \frac{\partial f^{*}}{\partial t} + \frac{\partial f}{\partial x}\cdot c \tag{2.21}$$

$$\boxed{\left.\frac{\partial f}{\partial t}\right|_{X} = \left.\frac{\partial f}{\partial t}\right|_{\chi} + c\cdot\nabla f} \tag{2.22}$$

Taking $\Psi \equiv I$ implies $\chi \equiv X$ and $u = \beta$, $c = 0$: the mesh nodes move with the same velocity as material particles, hence there is the Lagrangian description. The choice $\Phi \equiv I$ implies $\chi \equiv x$ and $\beta = 0$, $c = u$: the mesh nodes do not move, which characterizes the Eulerian description.

We take an example of the momentum conservation equation in Eulerian coordinate systems:

$$\rho \left.\frac{\partial u}{\partial t}\right|_{X} = \rho\left(\left.\frac{\partial u}{\partial t}\right|_{x} + u\cdot\boldsymbol{\nabla}u\right) = f + \boldsymbol{\nabla}\cdot\sigma$$

If we describe this momentum conservation equation in the ALE description, the equation is:

$$\rho \left.\frac{\partial u}{\partial t}\right|_{X} = \rho\left(\left.\frac{\partial u}{\partial t}\right|_{\chi} + (u-\beta)\cdot\boldsymbol{\nabla}u\right) = f + \boldsymbol{\nabla}\cdot\sigma$$

2.2 Unified Continuum Model

This section is a review of the Unified Continuum (UC) model for FSI in Euler coordinates. The fluid is assumed to be incompressible.

2.2.1 Principles of UC model

The main idea of the Unified Continuum model is to consider the fluid and the solid as one (single) continuum, where we solve the same system of equations for the whole continuum [6]. The two main differences of the fluid/solid parts are 1. how we calculate the stress force acting on the fluid and solid, and 2. the density difference between the fluid and solid parts.

The system of equations in the UC model includes conservation equations for mass and momentum, which can be found in any Computational Fluid Dynamics books, for instance [7]. We simplify this system of equations by assuming an incompressible continuum, in which the conservation equation for energy is decoupled in the model.

$$\begin{aligned} \dot{\rho} + \boldsymbol{\nabla} \cdot (\rho u) &= 0 \qquad \textit{Mass conservation} \\ \dot{m} + \boldsymbol{\nabla} \cdot (um) &= f + \boldsymbol{\nabla} \cdot \sigma \qquad \textit{Momentum conservation} \\ \dot{e} + \boldsymbol{\nabla} \cdot (eu) &= 0 \qquad \textit{Energy conservation} \end{aligned} \tag{2.23}$$

where:

ρ *is the density of continuum,* $[\frac{kg}{m^3}]$
u *is the velocity,* $[\frac{m}{s}]$
m *is the momentum,* $= \rho u$ $[\frac{kg}{m^2 s}]$
e *is the total energy,* $[J]$
f *is the volume force,* $[\frac{N}{m^3}] = [\frac{kg}{m^2 s^2}]$
$\boldsymbol{\nabla} \cdot \sigma$ *is the stress force,* $[\frac{N}{m^3}]$
σ *is so called the stress tensor.*
The dot (for example, in $\dot{\rho}$*) indicates the partial derivative by time (i.e.* $\dot{\rho} = \frac{\partial \rho}{\partial t}$*).*

Applying the chain rule for the term $\nabla \cdot (\rho u)$ gives $\nabla \cdot (\rho u) = \rho \nabla \cdot u + (u \cdot \nabla)\rho$. For an incompressible continuum, it holds that $\dot{\rho} + (u \cdot \nabla)\rho = 0$, which leads to:

$$\begin{aligned} \boldsymbol{\nabla} \cdot u &= 0 \qquad \textit{Continuity equation} \\ \rho(\dot{u} + u \cdot \boldsymbol{\nabla} u) &= f + \boldsymbol{\nabla} \cdot \sigma \qquad \textit{Momentum equation} \end{aligned} \tag{2.24}$$

where the energy equation is decoupled and omitted.

2.2.2 Stress forces calculation

As mentioned, the two differences regarding the fluid and solid phases are stress force and density differences. First, the stress force calculation is studied.

In general, the stress tensor can be written as a sum of an isotropic part and a deviatoric part.

$$\sigma_{ij} = \underbrace{d_{ij}}_{\text{Deviatoric part}} - \underbrace{p\delta_{ij}}_{\text{Isotropic part}} \tag{2.25}$$

where δ_{ij} is the Kronecker delta function.

For a moving fluid, the normal stress in each surface will change with direction. The pressure is defined here as the negative of mean normal stresses:

$$p = -\frac{1}{3}\sum_{i=1}^{3}\sigma_{ii}$$

For simplicity, the isotropic part is written as

$$\sigma_{ii} = -p\delta_{ii}$$

The deviatoric stress tensor d_{ij} contains the properties of shear stresses.

Different stress formulations for two phases

In order to distinguish the fluid and solid parts, a phase function θ is used, which identifies the material. More precisely, in the UC model, the phase function θ equals one for the fluid part and zero for the solid part.

Hence, total stress is defined as:

$$\sigma = \theta\sigma^f + (1-\theta)\sigma^s \tag{2.26}$$

where σ^f is the stress acting on the fluid part and σ^s is the stress acting on the solid part.

Fluid laws for stress force σ^f

For simplicity Newtonian fluids are considered here, which implies the Newtonian law [8] for the fluid phase:

$$\sigma^f = \underbrace{2\mu_v\epsilon}_{\text{Deviatoric part}} - \underbrace{pI}_{\text{isotropic part}} \tag{2.27}$$

where

μ_v *is the dynamic viscosity of fluid* $[\frac{kg}{ms}]$,
ϵ *is the rate of strain* $[\frac{1}{s}]$,
pI *is the normal stress* $[Pa] = [\frac{kg}{ms^2}]$,
p *is the pressure* $[Pa]$.

The rate of strain is defined by:

$$\epsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^T)$$

Solid laws for stress force σ^s

There are multiple choices of constitutive laws for the solid phase. In the solver Unicorn [1], the Neo-Hookean law is implemented:

$$\dot{\sigma}^s = 2\mu_s \frac{1}{2}(\nabla u + (\nabla u)^T) + \nabla u \sigma^s + \sigma^s (\nabla u)^T \tag{2.28}$$

where μ_s is the Young modulus (stiffness) of structure:

$$\mu_s = \frac{shearstress}{shearstrain}$$

2.2.3 Density difference

The second factor in the differences between the fluid and the solid is the density. Since this quantity is part of the momentum conservation equation, it is needed to be taken into account in the UC solver. Theoretically, for an incompressible flow and an elastic body, the momentum conservation equation becomes:

For the fluid:

$$\rho^f(\dot{u} + u \cdot \boldsymbol{\nabla} u) = f + \boldsymbol{\nabla} \cdot \sigma^f \tag{2.29}$$

For the solid:

$$\rho^s(\dot{u} + u \cdot \boldsymbol{\nabla} u) = f + \boldsymbol{\nabla} \cdot \sigma^s \tag{2.30}$$

2.2.4 Interface conditions

We define the interface between the solid phase and the fluid phase as Γ_{fs} - the boundary between the two subdomains Ω_s and Ω_f. For a detailed discussion, see [9].

Friction condition

In this work, a *no-slip* boundary condition is used for Γ_{fs}. The *no-slip* condition means that $u_s = u_f$ on Γ_{fs}. This condition provides a continuous velocity field over the whole domain.

Force-balance condition

In forming the weak formulation, which will be discussed in detail later in Section 2.3.3, we multiply the force term $\nabla \cdot \sigma$ by a test function and take the integral over the domain, then we apply the Green's first identity:

$$(\nabla \cdot \sigma, v) = -(\sigma, \nabla v) + \int_{\Gamma_{fs}} \sigma v \cdot n ds \tag{2.31}$$

Note that the inner product is defined as $(v, w) = \int_\Omega v \cdot w dx$, where Ω is the problem domain.

In this formula, a boundary term appears. If we apply this operator in both subdomains (fluid and solid), we obtain a jump term $\int_{\Gamma_{fs}}(\sigma^f - \sigma^s)v \cdot nds$. Enforcing the force-balance (according to the third law of Newton), we omit the jump term by simply set it to zero:

$$\int_{\Gamma_{fs}} (\sigma^f - \sigma^s)v \cdot nds = 0 \tag{2.32}$$

Equation (2.33) expresses a weak implementation of the *force-balance* condition for the interface Γ_{fs}:

$$\sigma^s \cdot n = \sigma^f \cdot n \quad \text{on } \Gamma_{fs} \tag{2.33}$$

Discontinuous Phase condition

The advection equation for the phase is defined as:

$$\dot{\theta} + \nabla \cdot (\theta u) = 0 \tag{2.34}$$

Using the chain rule for the term $\nabla \cdot (\theta u)$, we get:

$$\nabla \cdot (\theta u) = \theta \nabla \cdot u + u \cdot \nabla \theta \tag{2.35}$$

The flow is assumed to be incompressible, which means $\nabla \cdot u = 0$, so that:

$$\dot{\theta} + u \cdot \nabla \theta = 0 \tag{2.36}$$

Here u is the velocity vector field that advects the scalar quantity θ downstream. In the ALE description (explained in Section 2.1), $(u - \beta)$ is used instead of u, where β is the mesh velocity and u is the velocity.

In the UC model we compute U and β_h, the discrete versions of u and β, which give the discrete phase equation

$$\dot{\theta} + (U - \beta_h) \cdot \nabla \theta = 0 \tag{2.37}$$

On the interface Γ_{fs}, the phase function discontinuously changes from fluid ($\theta = 1$) to solid ($\theta = 0$) or vice versa. We want the phase function to have exactly the value 1 or 0 in each arbitrary cell (otherwise it provokes unphysical diffusion on the interface Γ_{fs}). On this interface, the velocity u is the same as the mesh velocity β (as we choose the solid part to move with the mesh velocity $u = \beta$). This gives a trivial solution to the phase convection equation and allows us to remove it from the system. Therefore, the phase interface can remain discontinuous.

2.2.5 Boundary conditions (BCs)

BC for momentum equation

First the boundary conditions for the momentum equation are considered. The boundary condition term w.r.t. space and time in weak form is:

$$\begin{aligned} \int_I \int_\Gamma (\sigma(\hat{u}) \cdot n) \cdot v ds dt &\equiv ((\sigma(\hat{u}) \cdot n, v))_{\Gamma \times I} \\ &= ((n^T \sigma(\hat{u}), v))_{\Gamma \times I} \end{aligned} \tag{2.38}$$

We rewrite the vector v in the new basis (n, τ_1, τ_2): $v = (v \cdot n)n + (v \cdot \tau_1)\tau_1 + (v \cdot \tau_2)\tau_2$, so that:

$$\begin{aligned}((n^T\sigma(\hat{u}), v))_{\Gamma\times I} &= ((n^T\sigma(\hat{u}), (v \cdot n)n))_{\Gamma\times I} + \sum_{k=1,2}((n^T\sigma(\hat{u}), (v \cdot \tau_k)\tau_k))_{\Gamma\times I} \\ &= ((n^T\sigma(\hat{u})n, v \cdot n))_{\Gamma\times I} + \sum_{k=1,2}((n^T\sigma(\hat{u})\tau_k, v \cdot \tau_k))_{\Gamma\times I}\end{aligned} \tag{2.39}$$

where:

$\sigma(\hat{u})$ *is the stress force*
n *is the normal vector*
τ_1, τ_2 *are the tangential vectors*

We mention here some models of the flow near to the wall:

1. A homogeneous Dirichlet boundary condition means that $u = 0$ on the external wall (in the Unicorn FSI model it is called *no-slip*). This model expresses the situation when the fluid sticks to the wall.

2. Skin friction boundary condition (*slip* BC) is the model when we prescribe the wall friction and the normal component of the velocity $u \cdot n$ is set to zero:

$$\begin{aligned}&u \cdot n = 0 \\ &n^T\sigma(\hat{u})\tau_k = -\beta_{friction}u \cdot \tau_k, \qquad k = 1, 2\end{aligned} \tag{2.40}$$

where $\beta_{friction} \in [0, \infty)$ is a friction coefficient. We also have that $v \cdot n = 0 \quad \forall v \in V$, so that the BC term becomes:

$$\begin{aligned}((n^T\sigma(\hat{u}), v))_{\Gamma\times I} &= \sum_{k=1,2}((n^T\sigma(\hat{u})\tau_k, v \cdot \tau_k))_{\Gamma\times I} \\ &= \sum_{k=1,2}((-\beta_{friction}u \cdot \tau_k, v \cdot \tau_k))_{\Gamma\times I}\end{aligned} \tag{2.41}$$

If we choose $\beta_{friction} = \infty$ we will have homogeneous Dirichlet BC as in the first model.

3. The *Free-slip* boundary condition describes no friction on the wall, i.e. friction is fully negligible, corresponding to $\beta_{friction} = 0$.

BC for pressure equation

We use a homogeneous Dirichlet BC for pressure at the outlet of the domain. Pressure at the inlet and at the wall is free.

2.2.6 Initial conditions

$$\begin{aligned}u(x, t = 0) &= u^0(x) \\ P(x, t = 0) &= P^0(x)\end{aligned} \tag{2.42}$$

2.3 Finite Element discretization and Solving Methods

The Finite Element Method (FEM) is a numerical method for obtaining approximate solutions for boundary value problems. This technique combines the use of variational method and piecewise approximation in small sub-domains (called finite elements) in order to solve differential equations over a large domain.

In FEM a problem domain is sub-divided into small (finite) elements. The collection of all elements is called mesh, which approximates the geometrical domain. The mesh consists of vertices (nodes), edges, faces and cells. The finer the mesh is, the better approximation of the geometry is.

The introduction to FEM below is from the lecture notes of Prof. Ulrich Rüde [10].

2.3.1 Variational form

We illustrate the concept of a variational form by considering a 1D boundary value problem.

We consider a problem with its Euler equations
$Lu^* = f$ (or $Lu^* - f = 0$) in $\Omega = (0, \pi)$ $\quad u^*(0) = u^*(\pi) = 0$,
in which u^* is the unknown, $u^* \in H_B^2$: all functions that have second derivative and satisfy the boundary conditions; Alternatively, H_B^1 is the space of all functions that have first order derivatives and satisfy the boundary conditions; L is a symmetric positive definite operator.

We convert this problem into an equivalent problem: take a so-called *test function* $v \in V \subseteq H_B^1$ to multiply with the Euler equations and take its integral over the domain:

$$\int_\Omega (Lu^* - f) \cdot v \; dV = 0 \qquad \forall v \in V \tag{2.43}$$

and the L_2 inner product is defined as $(f, v)_\Omega \equiv \int_\Omega f \cdot v \; dV$. Now we study the new problem: minimize the function $I(v)$ which is

$$I(v) := (Lv, v) - 2(f, v) \tag{2.44}$$

We call u is the solution to the new problem such that $I(u) = \min_v I(v)$. Then $I(u)$ should satisfy: for all $\epsilon \in \mathbb{R}$, $v \in V$ it holds that

$$\begin{aligned} I(u) \leq I(u + \epsilon v) &= (L(u + \epsilon v), u + \epsilon v) - (f, u + \epsilon v) \\ &= I(u) + 2\epsilon \left[(Lu, v) - (f, v) \right] + \epsilon^2 (Lv, v) \end{aligned} \tag{2.45}$$

Since ϵ can be arbitrary small and with either sign, it follows that

$$\begin{aligned} 2\epsilon \left[(Lu, v) - (f, v) \right] &= 0 \qquad \forall v \in V \\ (Lu, v) - (f, v) &= 0 \qquad \forall v \in V \end{aligned}$$

$$\boxed{(Lu - f, v) = 0 \qquad \forall v \in V; u \in V} \tag{2.46}$$

then $Lu = f$ (almost everywhere). This is the so called *variational form* (weak problem) for the initial problem $Lu^* = f$, u is the weak solution.

2.3.2 Basic idea of FEM

The basic idea of FEM is to choose a finite set of trial (basis) functions $\varphi_1, ..., \varphi_N$ so that we approximate

$$u \sim u^N := \Sigma_1^N q_i \varphi_i \qquad q_i \in \mathbb{R} \tag{2.47}$$

Consequently, to find u^N we need to find only a finite set of unknowns q_i, i.e. we have to solve a (non-)linear system of N algebraic unknowns. u^N is the "best approximation".

The goals are to have a good convergence rate, such that $u^N \to u$ as quick as possible; and to make it easy to find q_i, $i = 1..N$. These two factors depend on the choice of trial functions $\varphi_1, ..., \varphi_N$.

In the next section we illustrate a detailed discussion of solving the non-linear system of the FSI solver.

2.3.3 Standard Galerkin Finite Element method

As mentioned in (2.24), the system of equations that we will solve is:

$$\begin{aligned} \boldsymbol{\nabla} \cdot u &= 0 \qquad \textit{Continuity equation} \\ \rho(\dot{u} + u \cdot \boldsymbol{\nabla} u) &= f + \boldsymbol{\nabla} \cdot \sigma \qquad \textit{Momentum equation} \end{aligned} \tag{2.24}$$

We will now derive the finite element formulation of this system.

Denote the following:
Ω *is the space domain and* Γ *is the space domain boundary;*
I *is the time domain;*
$Q := \Omega \times I$ *- space-time domain;*
$L_2(Q)$ *- the standard space of square integrable functions over* Q*, where the space-time* L_2 *inner product is defined as* $((v, w))_{\Omega \times I} = \int_I \int_\Omega v \cdot w dx\, dt$*;*
$V_h^u(\Omega)$ *- the finite element space of piecewise linear continuous vector functions in space;*
$V_h^p(\Omega)$ *- the finite element space of piecewise linear continuous functions in space;*
$V_h^\theta(\Omega)$ *- the finite element space of piecewise constant discontinuous functions in space;*
$w = (u, p, \theta)$ *- exact solution, where* u *is the velocity,* p *is the pressure;* θ *is the phase;*
$W = (U, P, \Theta) \in V_h^u(\Omega) \times V_h^p(\Omega) \times V_h^\theta(\Omega)$ *- discrete solution;*
$\hat{v} = (v, q, v^\theta) \in V$ *- test function, where* $V = \{(v, q, v^\theta) \in V_h^u(\Omega) \times V_h^p(\Omega) \times V_h^\theta(\Omega)\}$ and the residual $R(W) = (R_u(W), R_p(W), R_\theta(W))$.

The discussion about the phase function (with residual $R_\theta(W)$) can be found in Section 2.2.4.

Space discretized momentum equation

We represent the stress term in (2.24) by $\nabla \cdot \sigma = \nabla \cdot (\Sigma^D - PI)$, where Σ^D is the constant deviatoric stress and I is the identity matrix. Then the residual equations become:

$$\begin{aligned} 0 &= R_u(W) = \rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - \boldsymbol{\nabla} \cdot (\Sigma^D - PI) - f \\ 0 &= R_p(W) = \nabla \cdot U \end{aligned} \tag{2.48}$$

Multiply the residual by the test functions and integrate by parts over the domain:

$$\begin{aligned} (R_u(W), v) &= (\rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - \boldsymbol{\nabla} \cdot (\Sigma^D - PI) - f, v) \\ &= (\rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - f, v) + (-\boldsymbol{\nabla} \cdot (\Sigma^D - PI), v) \\ &= (\rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - f, v) + (\Sigma^D - PI, \nabla v) \underbrace{- \int_\Gamma (\Sigma^D - PI) v \cdot n ds}_{\text{Boundary Condition term}} \end{aligned} \tag{2.49}$$

As for all types of boundary conditions that we discussed in Section 2.2.5 the normal velocity on boundary is zero $u \cdot n = 0$, it follows that the boundary term $\int_\Gamma (\Sigma^D - PI) v \cdot n ds$ will disappear. The momentum equation that we need to solve in weak form becomes:

$$0 = (R_u(W), v) = (\rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - f, v) + (\Sigma^D - PI, \nabla v) \tag{2.50}$$

Space discretized pressure equation

Multiply the residual continuity equation by the test function q and take integrate by part over the domain we get:

$$0 = (R_p(W), q) = (\nabla \cdot U, q) \tag{2.51}$$

2.3.4 Stabilization

In the solver, the weighted standard Galerkin/streamline diffusion method, described in [11], is implemented, which has the form

$$(R(W), \hat{v} + \delta R(\hat{v})) = 0, \forall \hat{v} \in V_h \tag{2.52}$$

with $\delta > 0$ as stabilization parameter.

For solving the pressure equation, we represent $\nabla \cdot \sigma$ as:

$$\nabla \cdot \sigma = -\nabla P + \mu_v \Delta U \tag{2.53}$$

Consequently, the residual system of equations become:

$$\begin{aligned} 0 &= R_u(W) = \rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) + \nabla P - \mu_v \Delta U - f \\ 0 &= R_p(W) = \nabla \cdot U \end{aligned} \tag{2.54}$$

Multiplying the momentum equation (2.54) by the test function $v + \delta_1(\rho U \cdot \nabla v + \nabla q)$ and the continuity equation by the test function $q + \delta_2(\nabla \cdot v)$, then combining the two equations, we get:

$$\begin{aligned} 0 = (R(W), \hat{v}) =& (\rho \dot{U}, v) + (\rho U \cdot \nabla U + \nabla P, v + \delta_1(\rho U \cdot \nabla v + \nabla q)) \\ & - (\mu_v \nabla U, \nabla v) - (f, v + \delta_1(\rho U \cdot \nabla v + \nabla q)) \\ & + (\nabla \cdot U, q) + (\nabla \cdot U, \delta_2 \nabla \cdot v) \end{aligned}$$

or

$$\begin{aligned} (\rho \dot{U}, v) + (\rho U \cdot \nabla U + \nabla P, v + \delta_1(\rho U \cdot \nabla v + \nabla q)) - (\mu_v \nabla U, \nabla v) & \\ + (\nabla \cdot U, q) + (\nabla \cdot U, \delta_2 \nabla \cdot v) & \\ = (f, v + \delta_1(\rho U \cdot \nabla v + \nabla q)) & \end{aligned} \tag{2.55}$$

Stabilized FEM for the pressure equation

Let q vary while setting $v = 0$ in (2.55), we get the stabilized finite element form for the pressure equation:

$$\begin{aligned} (\rho U \cdot \nabla U + \nabla P, \delta_1 \nabla q) + (\nabla \cdot U, q) &= (f, \delta_1 \nabla q) \\ (\rho U \cdot \nabla U, \delta_1 \nabla q) + (\nabla P, \delta_1 \nabla q) + (\nabla \cdot U, q) &= (f, \delta_1 \nabla q) \end{aligned}$$

$$\boxed{(\nabla P, \delta_1 \nabla q) = -(\rho U \cdot \nabla U, \delta_1 \nabla q) - (\nabla \cdot U, q) + (f, \delta_1 \nabla q)} \tag{2.56}$$

Stabilized FEM for the momentum equation

Let v vary while set $q = 0$ in (2.55), we get the stabilized finite element form for the momentum equation:

$$\begin{aligned} (\rho \dot{U}, v) + (\rho U \cdot \nabla U + \nabla P, v + \delta_1(\rho U \cdot \nabla v)) - (\mu_v \nabla U, \nabla v) + (\nabla \cdot U, \delta_2 \nabla \cdot v) & \\ = (f, v + \delta_1(\rho U \cdot \nabla v)) & \end{aligned}$$

$$\boxed{\begin{aligned} 0 =& (\rho \dot{U}, v) + (\rho U \cdot \nabla U, v) + (\nabla P, v) - (\mu_v \nabla U, \nabla v) - (f, v) \\ & + (\rho U \cdot \nabla U, \delta_1 \rho U \cdot \nabla v) + (\nabla P, \delta_1 \rho U \cdot \nabla v) - (f, \delta_1 \rho U \cdot \nabla v) + (\nabla \cdot U, \delta_2 \nabla \cdot v) \end{aligned}} \tag{2.57}$$

Using that $(\nabla P, v) - (\mu_v \nabla U, \nabla v) = (\Sigma^D - PI, \nabla v)$, we get another formula for momentum equation (as in (2.50) with stabilization terms):

$$\boxed{\begin{aligned} 0 =& (\rho \dot{U}, v) + (\rho U \cdot \nabla U, v) + (\Sigma^D - PI, \nabla v) - (f, v) \\ & + (\rho U \cdot \nabla U, \delta_1 \rho U \cdot \nabla v) + (\nabla P, \delta_1 \rho U \cdot \nabla v) - (f, \delta_1 \rho U \cdot \nabla v) + (\nabla \cdot U, \delta_2 \nabla \cdot v) \end{aligned}} \tag{2.58}$$

2.3.5 ALE implementation in the UC model

Applying the ALE method for the Unified Continuum model, we replace the convective velocity in the term $u \cdot \nabla u$ by the ALE convective velocity $(u - \beta)$, such that the convective term becomes $(u - \beta) \cdot \nabla u$.

Let us denote β_h as an discretized mesh velocity with mesh size h.

In the solid part ($\theta = 0$), we choose $\beta_h = U$ as a trivial case ($\Psi = I$) of ALE description. This choice will also cancel the term $(U - \beta_h) \cdot \nabla U$ in the solid part.

For fluid vertices ($\theta = 1$), mesh smoothing is used to determine β_h. Note that mesh smoothing is essential to maintain the cell quality [12].

2.3.6 Solving the non-linear algebraic system

The Newton method is used here as a standard method for solving the non-linear algebraic system that results from discretization of a time dependent PDE. This method can be found in [13] for details.

Solving the time dependent momentum equation

The momentum equation in (2.58) is the space discretization of a time dependent PDE, which is used to solve for velocity U. In order to solve it, we discretize the equation in time. In each time step, we solve the ***time independent*** equation.

$$\begin{aligned} 0 =& (\rho(\dot{U} + U \cdot \nabla U) - f, v) + (\Sigma^D - PI, \nabla v) + (\delta_1 ...) + (\delta_2 ...) \\ =& (\rho \dot{U}, v) + \underbrace{(\rho U \cdot \nabla U - f, v) + (\Sigma^D - PI, \nabla v) + (\delta_1 ...) + (\delta_2 ...)}_{f_1(U,v) :=} \end{aligned} \tag{2.59}$$

Here $f_1(U, v)$ is the time-independent part of the momentum equation. We choose to let $f_1(U, v)$ depend on updated and old values of U, in particular $f_1(U_c, v)$ where $U_c = 0.5(U^n + U^{n-1})$.

$$0 = \left(\rho \frac{U^n - U^{n-1}}{k_n}, v \right) + f_1(U_c, v) \tag{2.60}$$

where U^{n-1} is the solution of previous time-step and k_n is the time-step.

Multiplying by k_n in both sides:

$$0 = \underbrace{(\rho U^n, v) - (\rho U^{n-1}, v) + k_n f_1(U_c, v)}_{F(U^n, U^{n-1}, k_n, v) :=} \tag{2.61}$$

We introduce $F(U, U^{n-1}, k_n, v) = 0$ as the short-form of the momentum equation in (2.61),

$$0 = F(U, U^{n-1}, k_n, v) = (\rho U, v) - (\rho U^{n-1}, v) + k_n f_1(U_c, v) \tag{2.62}$$

Calculate the Jacobian matrix J of the function $F(U, U^{n-1}, k_n, v)$:

$$J = \frac{\partial F}{\partial U}$$

.

Choosing an initial guess U_p and updating it after each iterations, we have:

$$\begin{aligned} J(U - U_p) &= -F(U_p) \\ \underbrace{J}_{\text{Known matrix}} \quad \underbrace{U}_{\text{Unknowns}} &= \underbrace{JU_p - F(U_p)}_{\text{Right hand side}} \end{aligned} \tag{2.63}$$

Solving this linear system of equations (LSE) gives us the solution U^n of the momentum equation in one time step. Similarly, this is performed for all time steps after updating the previous values U^{n-1}.

Solving the continuity equation

The second equation in (2.56) is the continuity equation, which is used to solve for the pressure P.

$$\begin{aligned} 0 = (R_p(W), q) &= (\nabla \cdot U, q) + (\nabla P, \delta_1 \nabla q) + (\rho U \cdot \nabla U, \delta_1 \nabla q) - (f, \delta_1 \nabla q) \\ \underbrace{(\nabla P, \delta_1 \nabla q)}_{\text{Bilinear form } a(P,q)} &= \underbrace{-(\nabla \cdot U, q) - (\rho U \cdot \nabla U, \delta_1 \nabla q) + (f, \delta_1 \nabla q)}_{\text{Linear form } L(q)} \end{aligned} \tag{2.64}$$

These bilinear form and linear form will be assembled by Dolfin (FEniCS project [1]) to a matrix A and a vector b respectively (standard form as $Ax = b$) then this LSE is solved for ΔP.

2.3.7 Methods for solving Linear Systems of Equation (LSE)

There is multiple choice for solving the LSE $Ax = b$ in FEniCS [1] as illustrated in tables 2.1 and 2.2.

In this work, we used the combination (amg, gmres) for solving both the momentum equation and the pressure equation.

2.4 Realization of Unicorn FSI Solver

2.4.1 Converting variational forms to LSE

Form files are files written in Unified Form Language (UFL) embedded in Python. These files specify variational forms of finite element discretizations of differential equations.

For example in Unicorn, Appendix A.1 shows the form file of the discretized momentum equation (2.58). For more form files, we refer to Unicorn source code.

Usage	**Preconditioner**
"none"	No preconditioner
"ilu"	Incomplete LU factorization
"icc"	Incomplete Cholesky factorization
"jacobi"	Jacobi iteration
"bjacobi"	Block Jacobi iteration
"sor"	Successive over-relaxation
"amg"	Algebraic multigrid (BoomerAMG or ML)
"additive_schwarz"	Additive Schwarz
"hypre_amg"	Hypre algebraic multigrid (BoomerAMG)
"hypre_euclid"	Hypre parallel incomplete LU factorization
"hypre_parasails"	Hypre parallel sparse approximate inverse
"ml_amg"	ML algebraic multigrid

Table 2.1: Preconditioner parameters

Usage	**Solver**
"lu"	LU factorization
"cholesky"	Cholesky factorization
"cg"	Conjugate gradient method
"gmres"	Generalized minimal residual method
"bicgstab"	Biconjugate gradient stabilized method
"minres"	Minimal residual method
"tfqmr"	Transpose-free quasi-minimal residual method
"richardson"	Richardson method

Table 2.2: Solver parameters

The output of these files are the bilinear form and linear form of equations. For the momentum equation they are (aM, LM) and for the continuity equation (aC, LC). Similarly, (aS, LS) is the output of form file for Stress calculation, which is the realization of equation (2.26), (2.27), (2.28). These bilinear/linear forms are assembled later to matrix/vector form by the class Assembler of Dolfin.

2.4.2 Algorithm of Unicorn FSI Solver

The results of evaluating form files are saved in bilinear and linear forms, for example aC, LC for the continuity equation. Assembling these forms in each time step gives the corresponding matrices and vectors of the LSE problems. For instance, the matrix PM and the vector Pb in the LSE $PM\Delta P = Pb$ for the continuity equation, where ΔP is the unknown vector.

The general algorithm for solving the FSI system is demonstrated in algorithm 1.

Algorithm 1 Algorithm of Unicorn FSI Solver

function INITIALIZEVARIABLES
function CREATEBILINEARLINEARFORMS
while $t \leq T$ **do** ▷ T is final simulation time
  function PREPARESTEP
    $W_0 \leftarrow W$ ▷ mesh velocity
    $X_0 \leftarrow X$ ▷ node position
    $\rho_0 \leftarrow \rho$ ▷ general density of continuum
    $\sigma_0 \leftarrow \sigma$ ▷ stress tensor
    $\dot{\sigma}_0 \leftarrow \dot{\sigma}$ ▷ time derivative of stress tensor
    $U_0 \leftarrow U$ ▷ velocity - solution
    function SMOOTHMESH
  function STEP
    for $i = 1 \rightarrow maxit$ **do**
      function PREPAREITERATION
        $X_{inc} \leftarrow \frac{k(U+W_0)}{2} + X_0$
        function DEFORM_SOLID
        $X \leftarrow$ <*current position of mesh*>
        $W \leftarrow \frac{X-X_0}{k}$
        $P_0 \leftarrow P$
        function COMPUTESTABILIZATION **return** $(d1, d2)$
        function COMPUTEP
          $PM \leftarrow assemble(aC)$
          $Pb \leftarrow assemble(LC)$
          function BCS.APPLY ▷ BCs for pressure eq.
          $\Delta P \leftarrow solve(PM, \Delta P, Pb)$
          $P \leftarrow P_0 + \Delta P$
        function COMPUTES
          $\dot{\sigma} \leftarrow assemble(LS)$
          $\sigma \leftarrow \frac{1}{2}(\dot{\sigma} + \dot{\sigma}_0) * k + \sigma_0$
      function ITER
        $J \leftarrow assemble(aM)$
        $dotx \leftarrow assemble(LM)$
        function BCS.APPLY ▷ BCs for momentum eq.
        $U \leftarrow solve(J, U, dotx)$
    function SAVE(U,t) ▷ save solution
  $t \leftarrow t + k$ ▷ k is time step, t is current simulation time

Chapter 3

Optimization and Testing of the Unicorn FSI solver

3.1 Optimizations

In this work we aim to make the solver more general and robust.

In the current Unicorn FSI solver the density was set to unity ($\rho \equiv 1$), which limits the solver to cover only certain cases, but not for instance our industrial problem. Furthermore, there were no previous problems where the force term was used, meaning that the force was set to zero ($f = 0$). Moreover, the solver was not well-tested, in particular a scaling problem was identified in stabilization coefficients δ_1, δ_2.

In this chapter we outline the optimizations and tests made to the solver.

3.1.1 Modification of the momentum form file

As discussed in Section 2.3.4, we implemented the formula (2.58) in the Unicorn FSI solver. The basic improvement is the change to a general density ρ in the form file. and adding stabilization terms for consistency. For details, we refer to Appendix A.1.

The new formula reads:

$$\boxed{\begin{aligned} 0 =&(\rho\dot{U}, v) + (\rho U \cdot \nabla U, v) + (\Sigma^D - PI, \nabla v) - (f, v) \\ &+ (\rho U \cdot \nabla U, \delta_1 \rho U \cdot \nabla v) + (\nabla P, \delta_1 \rho U \cdot \nabla v) - (f, \delta_1 \rho U \cdot \nabla v) + (\nabla \cdot U, \delta_2 \nabla \cdot v) \end{aligned}} \tag{2.58}$$

to be compared to the old simplified version in the Unicorn FSI solver:

$$\boxed{\begin{aligned} 0 =&(\rho(\dot{U} + U \cdot \boldsymbol{\nabla} U) - f, v) + (\Sigma^D - PI, \nabla v) \\ &+ (\rho U \cdot \nabla U, \delta_1 \rho U \cdot \nabla v) + (\nabla \cdot U, \delta_2 \nabla \cdot v) \end{aligned}} \tag{3.1}$$

3.1.2 Modification of the continuity form file

For the continuity form file, we added the force term and the density term together with corresponding stabilization terms as in the equation (2.56). Details of the implementation are in Appendix A.2.

New formula:

$$\boxed{(\nabla P, \delta_1 \nabla q) = -(\nabla \cdot U, q) - (\rho U \cdot \nabla U, \delta_1 \nabla q) + (f, \delta_1 \nabla q)} \tag{2.56}$$

The old simplified formula in Unicorn FSI:

$$\boxed{(\nabla P, \delta_1 \nabla q) = -(\nabla \cdot U, q) - (U \cdot \nabla U, \delta_1 \nabla q)} \tag{3.2}$$

3.1.3 Calculating stabilization coefficients

To understand the scaling of the stabilization coefficients δ_1, δ_2, we here provide a dimensional argument.

Set $u = Uu^*, x = Lx^*, t = Tt^*, \rho = \Phi\rho^*, p = Pp^*, f = Ff^*$
and $\nabla = \frac{1}{L}\nabla^*$, $\frac{d}{dt} = \frac{1}{T}\frac{d}{dt^*}, T = \frac{L}{U}, P = \Phi U^2$,
where the star * denotes the non-dimensional quantities (scales) and the capital letters denote the unit.

From the UC system of equations:

$$\begin{aligned} \boldsymbol{\nabla} \cdot u &= 0 \qquad \textit{Continuity equation} \\ \rho(\dot{u} + u \cdot \boldsymbol{\nabla} u) + \nabla P - \mu_v \Delta u &= f \qquad \textit{Momentum equation} \end{aligned} \tag{3.3}$$

we rewrite the momentum equation in dimensionless form:

$$\frac{\Phi U}{T}\rho^*(\frac{du^*}{dt^*} + u^* \cdot \nabla^* u^*) + \frac{P}{L}\nabla^* P^* - \mu_v \frac{U}{L^2}\Delta^* u^* = Ff^*$$

$$\frac{\Phi U^2}{L}\left(\rho^*(\frac{du^*}{dt^*} + u^* \cdot \nabla^* u^*) + \nabla^* p^*\right) - \mu_v \frac{U}{L^2}\Delta^* u^* = Ff^* \tag{3.4}$$

The Reynold number is defined as $Re = \frac{UL}{\nu} = \frac{\Phi UL}{\mu_v}$ with $\nu = \frac{\mu_v}{\rho}$, so that:

$$\left(\rho^*(\frac{du^*}{dt^*} + u^* \cdot \nabla^* u^*) + \nabla^* p^*\right) - Re^{-1}\Delta^* u^* = f^* \tag{3.5}$$

This is so called the non-dimensional momentum-equation.

We also rewrite the continuity equation as:

$$0 = \nabla \cdot u = \frac{U}{L}(\nabla^* \cdot u^*) \tag{3.6}$$

Calculating δ_1

We add a stabilization term with parameter δ_1 to the momentum equation (3.4), which in variational form can be stated as

$$(\delta_1(\rho u \cdot \nabla u + \nabla P - f), \rho u \cdot \nabla v) \tag{3.7}$$

We study the term inside the integral:

$$\delta_1 \frac{\Phi U^2}{L} \frac{\Phi U}{L} (\rho^* u^* \cdot \nabla^* u^* + \nabla^* p^* - f^*) \cdot (\rho^* u^* \cdot \nabla^* v) \tag{3.8}$$

The terms (3.4) and (3.8) should have the same dimension (unit), such that:

$$\begin{aligned} \delta_1 \frac{\Phi U^2}{L} \frac{\Phi U}{L} &\sim \frac{\Phi U^2}{L} \\ \delta_1 &\sim \frac{L}{\Phi U} \end{aligned} \tag{3.9}$$

where we let

$$\delta_1 \equiv C_1 \frac{h}{\rho u}$$

.

In the implementation, we account also for the time-dependent part and the ALE description for the convective velocity. Therefore, δ_1 contains also information of the time step k and the ALE velocity.

$$\boxed{\delta_1 = C_1 \frac{0.5}{\rho \sqrt{\left(\frac{1}{k}\right)^2 + \left(\frac{U-W}{h}\right)^2}}} \tag{3.10}$$

where W is the mesh velocity. The factor 0.5 comes from a theoretical result of pure transport equations, which states that $\delta = \frac{1}{2}\frac{h}{U}$ is the optimal choice [14].

We also add a stabilization term with δ_1 to the pressure equation (3.6):

$$(\delta_1(\rho u \cdot u + \nabla P - f), \nabla q) \tag{3.11}$$

So that the scaling of the integral is:

$$\delta_1 \frac{\Phi U^2}{L} \frac{1}{L} (\rho^* u^* \cdot u^* + \nabla^* p^* - f^*) \cdot (\nabla^* q) \tag{3.12}$$

The terms in (3.6) and (3.12) should also have the same dimension:

$$\begin{aligned} \delta_1 \frac{\Phi U^2}{L} \frac{1}{L} &\sim \frac{U}{L} \\ \delta_1 &\sim \frac{L}{\Phi U} \end{aligned} \tag{3.13}$$

which gives the same result as in (3.9).

Calculating δ_2

The stabilization term with parameter δ_2 is

$$(\delta_2(\nabla \cdot u), \nabla \cdot v), \tag{3.14}$$

For which the scaling of the integral is:

$$\delta_2 \frac{U}{L^2}(\nabla^* \cdot u^*) \cdot (\nabla^* \cdot v) \tag{3.15}$$

Analogously, the terms (3.4) and (3.15) should have the same dimension:

$$\begin{aligned} \delta_2 \frac{U}{L^2} &\sim \frac{\Phi U^2}{L} \\ \delta_2 &\sim LU\Phi \end{aligned}$$

where we let

$$\boxed{\delta_2 = C_2 \rho U h} \tag{3.16}$$

The stabilization terms δ_1, δ_2 have been modified in Unicorn FSI solver to have the correct scaling.

3.1.4 Check-point restart

At some point of a simulation, users may want to save the state of a simulation (with all necessary data) so that he/she can use that point as initial values for another simulations or simply restart/continue from that point. This feature is implemented in FEniCS solvers, in particular in the Unicorn FSI solver. FSI checkpoint-restart is tested and modified in this project. The result after restarting from a checkpoint is exactly same as the result saved during checkpointing.

3.2 New Features

3.2.1 Tracking points

When running a FSI simulation, users can choose one or several interesting solid points (vertices) to track its/their coordinates. The coordinates of these points change during the simulation as the solid moves but the indices of these points remain the same. The idea is based on using the indices to track the points.

- In the initial mesh, we identify the index of an interesting point geometrically, within some defined tolerance. More precisely, we search for a vertex within the neighbourhood of a spatial point, which is the measured initial position of the interesting point, the radius of the neighbourhood is a user-defined tolerance.

- If a vertex is found, we store the index of it. This vertex represents the interesting point.

- Based on this index, we track and print out the changing coordinates of this vertex.

If there is no vertex found inside the neighbourhood-sphere, the index returned is -1. If there are more than one vertices found in the sphere, the index of nearest vertex to the sphere center is returned.

The implementation of the TrackingPoint class is described in Appendix B.1.

3.2.2 Applying force bases on vertices

Since the solid body moves during the FSI process, the area where the force is applied on the solid body also moves. The requirement is to move the force along with the solid part that the force is applied on.

In Unicorn, the force-area is defined by the part of the mesh that contains the solid. In form files, force is multiplied with $(1 - \theta)$ to identify the solid part. This approach does not describe the right area of the force from the beginning and can causes possible bugs when one wants to use the force apart from the form files.

We have changed the code to detect vertices in the initial force-area, so called force-vertices, to apply the force on. Those force-vertices are saved to TrackingPoint objects. All those objects are stored to the attribute $forceVertices$. When the solid moves, the force-vertices also move. Based on the force-vertices, we apply the force on them and their neighborhood.

$Array < TrackingPoint* > forceVertices$;

We refer to the source code of the ForceArea class in Appendix B.2 for details.

3.3 Testing

3.3.1 Test problem

For testing purposes, we used a simple FSI problem: a cantilever beam, shown in Fig. 3.1. We used the geometry described in Sec. 1.3. However, we removed the four pins around the top of the bar, meaning that the steel bar (density $8000kg/m^3$) now is free to move at one end and is fixed in the other end. The bar is surrounded by flowing water (density $1000kg/m^3$) or air (in this test air density is assumed to be $1.0kg/m^3$) with the inflow velocity $Ubar = 1m/s$. A force F is applied on the bar as a load in order to bend the top of the bar $1cm$ in horizontal direction. When the top of the bar reaches the desired bending position, the force is removed and the bar is released to freely vibrate.

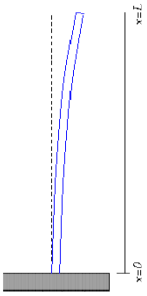


Figure 3.1: An example of Cantilever Beam

In this test, the boundary conditions (BCs) for the momentum equation 2.58 consist of:

- Inflow boundary: specified inflow velocity $Ubar$
- Fixed boundary: the boundary at the bottom of the bar is fixed (velocity equals zero)
- External walls of the pipe: no-slip BC was used
- Interface between the fluid and solid phases: no-slip BCs was applied, which means that momentum here is zero.

The BC for continuity equation 2.56 is:

- Outflow boundary: pressure on the outflow boundary is zero.

3.3.2 Force test

Up to now, FSI problems that are solved by the Unicorn FSI solver do not involve external forces. Consequently, in the continuity equation, the old solver neglects external forces (shown in equation 3.2) to avoid computational overhead. Since in our FSI problem an external force is needed to bend the bar, the force term was added according to equation (2.56) and then tests were performed for the new solver.

In the case of one free end of the bar, the formula of the bar deformation is:

$$\Delta = \frac{Fb^3}{3EI} \tag{3.17}$$

The detailed explanation can be found in Section 4.1.3.

To bend the bar top $1cm$ with the solid stiffness $\mu_s = 10^7 N/m^2$ we need a force

$$F = \frac{3EI\Delta}{b^3} = 0.000607N$$

Converting to force density, we have $f = F/V = 2.53N/m^3$. The force is applied in the y-direction. i.e. normal to the wider surface of the bar. In addition, air flow with air viscosity was used.

As can be seen in Fig. 3.2, the y-coordinate change of the bar top reached the maximum $\approx 10mm$.

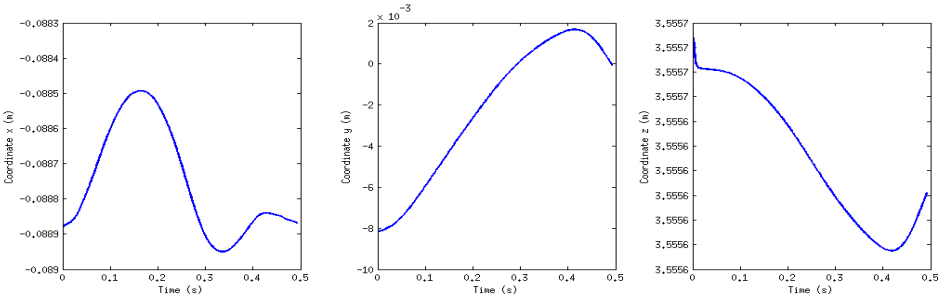
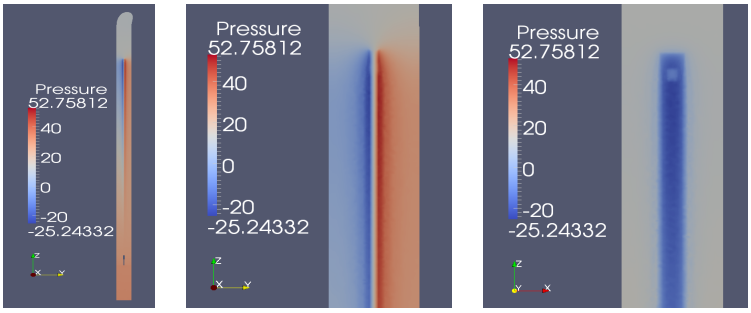


Figure 3.2: Coordinate change of the bar top during the bending process under the force density $f = 2.5N/m^3$.

The result of the pressure inside the pipe under an external force is plotted in Fig. 3.3. A difference in pressure in the direction of movement of the bar is observed, which shows that the solver works according to corresponding physics phenomena.



(a) (b) (c)

Figure 3.3: Pressure inside the air channel under a bending force by the new FSI solver; (a) Overview: a slide cut through the channel in the y-direction; (b) A zoomed-in slide cut on the bar top in the y-direction; (c) A zoomed-in slide cut on the bar top in the x-direction;

3.3.3 Velocity and Pressure comparison

Using both the old FSI solver and the new solver to run the same combination of parameters, we compare the results. In the remaining test cases of chapter 3.3, no external force was applied.

In order to have a rough approximation of the flow velocity inside the pipe, we make a calculation based on the relation of flow velocity and cross sectional-area of the pipe. The flow is accelerated inside the pipe as there is narrower flow cross section (because of the bar). We calculate the velocity of the flow in the channel where the bar is:

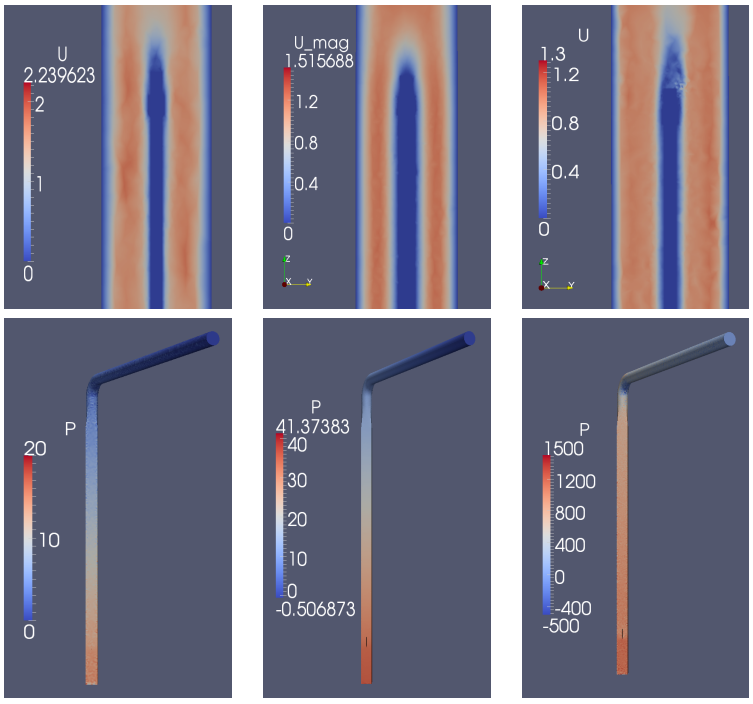
- At the inflow: the cross sectional-area of the flow is $S_1 = 80mm \times 80mm = 6400mm^2$ with the mean velocity $Ubar = 1m/s$.

- At the slide cut in the middle of the bar: The cross sectional-area of the bar: $8mm \times 20mm = 160mm^2$. The cross sectional-area of the flow is $S_2 = 6400mm^2 - 160mm^2 = 6240mm^2$, which implies that the mean velocity is $U_2 = \frac{Ubar \times S_1}{S_2} = 1.026m/s$.

Comparing the two cases (a) and (b) in Fig. 3.4 for the air flow case (which is assumed to be incompressible as discussed in Sec. 2.2), we observe that the resulting velocity U of the old solver is higher than the one in the new solver. According to the calculation above, the result from the new solver seems to be more reasonable since it is closer to the value $1.026m/s$.

The old solver does not handle the case of water flow (as well as other flow material with density different from unity), whereas the new solver can. For instance the new solver can handle the water flow (density $1000kg/m^3$), the result shown in Fig. 3.4(c). In this water flow case, the velocity field has lower values compared to the air flow case, which can be explained by the effect of higher viscosity: water viscosity is about 90 times larger than air viscosity, which lead to the fact that water resists the tendency of the flow more than air.

The pressure plots shown in Fig. 3.4 are the pressure difference between a point inside the pipe and the outflow surface (where we assume a zero pressure). The pressure of the water flow is higher than the pressure of the air flow, which can be explained by higher density of water: a larger number of molecules in a volume unit creates a higher pressure.



(a) (b) (c)

Figure 3.4: Velocity $U(m/s)$ and Pressure $P(Pa)$ for the cantilever problem with the inflow velocity $1m/s$; (a) The old FSI solver for air flow ($\rho \equiv 1kg/m^3$); (b) The new FSI solver for air flow ($\rho^f = 1kg/m^3; \rho^s = 8000kg/m^3$); (c) The new FSI solver for water flow ($\rho^f = 1000kg/m^3; \rho^s = 8000kg/m^3$).

3.3.4 Test the FSI-solver as an Incompressible Navier-Stoke solver

The idea here is that if the solid part is removed from the problem, the FSI-solver[0] should be able to handle the turbulent flow case as the pure fluid Unicorn solver called Incompressible Navier-Stoke solver (ICNS-solver) [1] does. In this test case, the bar is removed from the pipe but the bottom cross (shown in Fig. 1.1(c)) is remained; water or air is pumped into the pipe with an inflow velocity $1m/s$.

Fig. 3.5(a) illustrates the results of the FSI solver for water flow. The solver is stable for this case since it ran over all simulation time $T = 5s$. As can be seen, the flow is stable and turbulence appears in the downstream behind the cross, which indicates that the FSI solver can handle turbulent flow simulations.

Fig. 3.5(b)(c) show that for air flow case the results of the FSI-solver in (b) are different compared to the results from the ICNS-solver in (c). The FSI solver gives a lower velocity than the one from the ICNS solver. The velocity is accelerated due to the incompressibility assumption in both solvers and the cross on the bottom of

[0]From now the FSI solver is referred to as the new FSI solver.

the geometry. To identify which result is more realistic, we propose to measure also the velocity and the pressure in the real experiments.

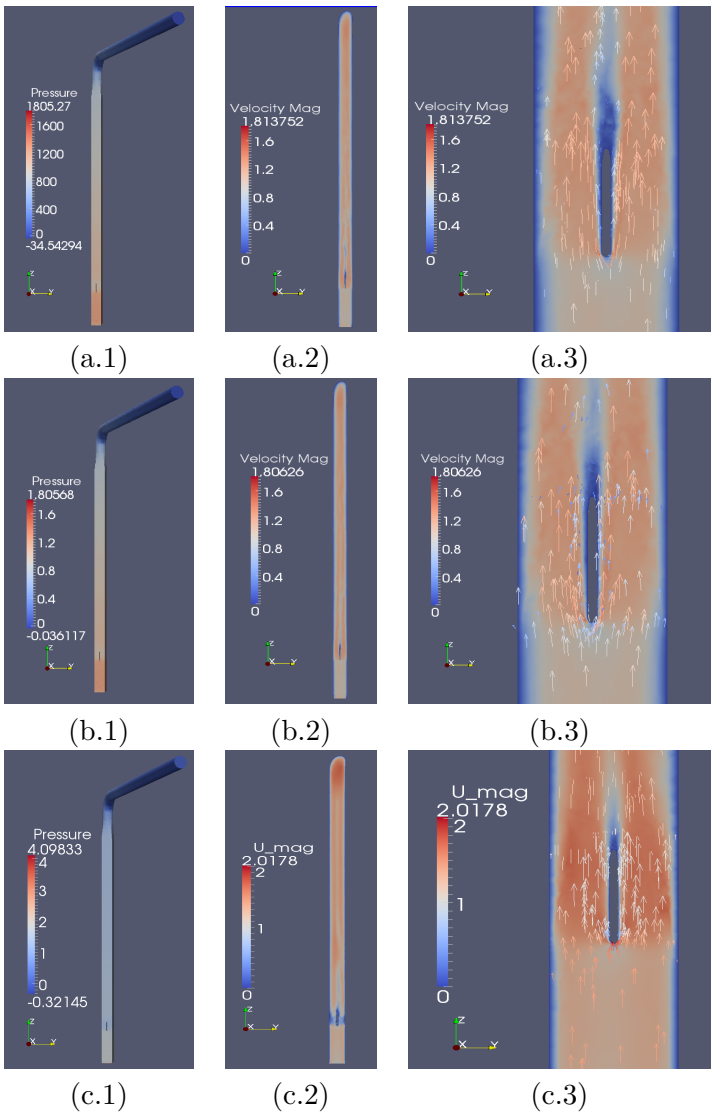


Figure 3.5: (a) Turbulent water flow by the new FSI-solver; (b) Turbulent air flow by the new FSI-solver; (c) Turbulent air flow by the ICNS-solver; (1) The pressure through the pipe; (2) Overview of a velocity slide-cut along the pipe; (3) Zoomed-in to the bottom cross part of the velocity slide-cut (2);

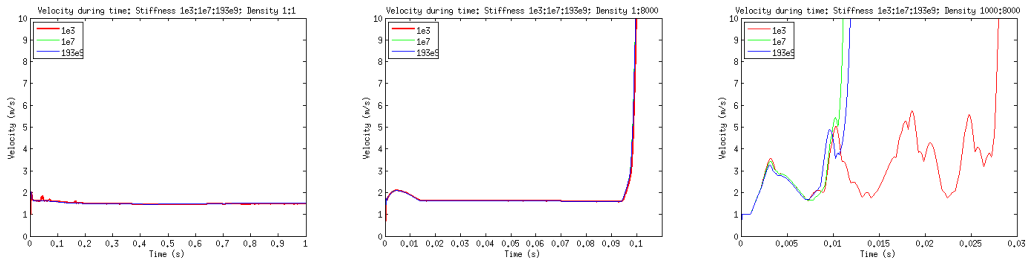
3.3.5 Sensitivity with respect to physical parameters

In each test case below only the test input parameter was changed and the rest input parameters were fixed. The main subject to be studied here is how the change of

input parameters influences the output infinity norm of the velocity vector field. If variation of the velocity magnitude becomes smaller than a threshold over time, we say that the solution converts and stability is established. In contrast, if the velocity magnitude reaches to infinity (or a large number) we call that the solution diverts and the solver is unstable.

Solid stiffness μ_s

One of the interesting parameters that may affect the sensitivity of the solver is the solid stiffness (Young's modulus). Fig. 3.6 illustrates the infinitive norm of the velocity magnitude over time.



(a) (b) (c)

Figure 3.6: Sensitivity of the FSI solver with respect to solid stiffness: (a) With fluid:solid density 1:1 (air density); (b) With fluid:solid density = 1:8000 (air/steel densities); (c) With flui:solid density = 1000:8000 (water/steel densities);

We compared three Young's modulus of the bar: the stiffness of paper $E = 1e3N/m^2$, the stiffness of rubber $E = 1e7N/m^2$ and the stiffness of steel $E = 19e9N/m^2$. These three Young's modulus were tested with three different combinations of fluid/solid densities.

As can be seen in Fig. 3.6 the results with respect to different stiffness are quite similar. However, in the case (c) the stiffness of paper appears to be easier for the solver to handle. In conclusion, in cases without the external force and low fluid density, the sensitivity of the solver appears not to depend on the stiffness of the solid material. In contrast, for high fluid density the lower the stiffness, the higher stability of the solver.

Density ρ

For the density test, two cases are studied:

1. Scaling of density: we use the same density for both fluid and solid, and scale that common density;

2. Density difference: we use different density for fluid and solid.

In these two test cases, the solid stiffness was set to $E = 1e7N/m^2$ and the fluid viscosity to $\mu_v = 0.00001983Ns/m^2$.

In the first test case we changed the density for both phases from 1 to 10, 100, 1000 and $2000kg/m^3$. The results are demonstrated in Fig. 3.7. As can be seen, scaling the density from 1 to $1000kg/m^3$ causes the velocity to be more unstable in the beginning of the simulations. The larger the scale, the broader the velocity variation and the later the velocity gets to its steady state. However, for the density $2000kg/m^3$ the velocity diverted. Consequently, larger density scale makes the solver become more sensitive.

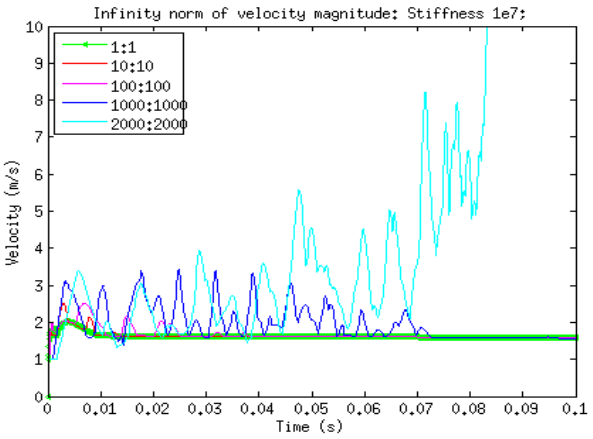


Figure 3.7: Density scaling: Fluid:solid density scaled from 1:1 to 2000:2000 (kg/m^3)

The second test case studies the sensitivity of the FSI solver concerning the difference in density of the two phases. Fig. 3.8 show the infinitive norm of the velocity magnitude during simulation time. As shown, both increasing the fluid or solid density causes the solver to be more unstable. Comparing the cases 1:8000 and 100 : 8000: the bigger difference in density is more stable. While comparing the cases 1000 : 8000 and 1000 : 2000, the bigger difference in density is more unstable. It can be concluded that the more/or less difference in density does not seem to play an important role in the solver's sensitivity.

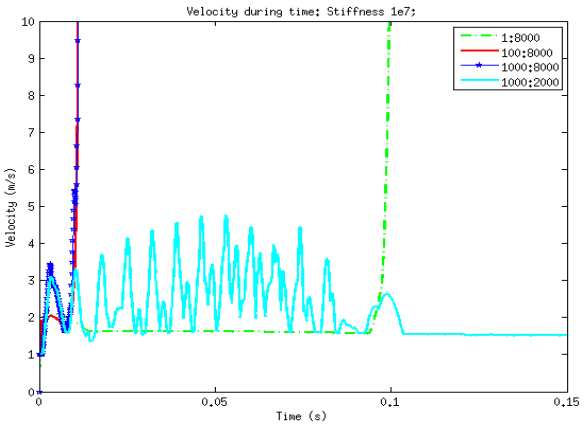


Figure 3.8: Difference in density: Fluid:solid densities changed from 1:8000 to 1000:8000, 1000:2000 (kg/m^3)

Fluid viscosity μ_v

In this section the effect of the fluid viscosity is studied.

As can be seen in Fig. 3.9 a higher viscosity of the fluid (water viscosity versus air viscosity) does not seem to improve the stability of the solver. However, an "artificial" viscosity can stabilize the solver. "Artificial" viscosity means that we set the viscosity of the fluid as a function that can stabilize the velocity, so that

$$\mu_v := \mu_v + \rho h \, ||U|| \tag{3.18}$$

where h is the local cell size and U is the local velocity in the cell.

On the other hand, as the fluid viscosity has a big influence on the movement of the solid, for accuracy it is not recommended to increase the fluid viscosity for the solver stabilization.

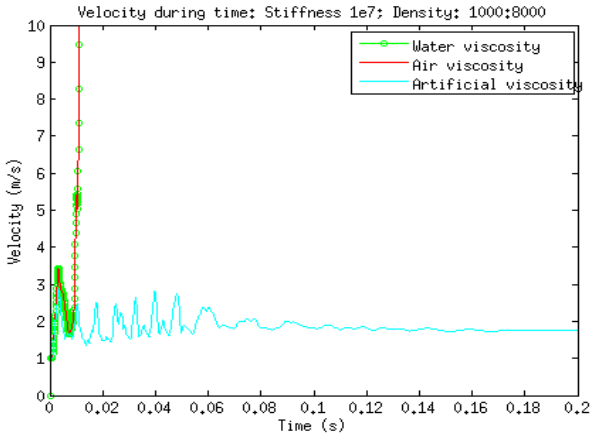


Figure 3.9: Viscosity compare: water dynamic viscosity $\mu_v = 0.00136857 Ns/m^2$; air dynamic viscosity $\mu_v = 0.00001983 Ns/m^2$; and "artificial" viscosity $\mu_v := \mu_v + \rho h \, ||U||$ with $\mu_v = 0.00136857 Ns/m^2$.

3.3.6 Sensitivity with respect to simulation parameters

Time step k

The solver stability depends strongly on time step k. Taking large time steps can save computational cost and make the simulation faster. On the other hand, it means that numerical results contain larger discretization error, or even worse - large time steps can make results unstable and whole simulation crash. Therefore, the size of time step should be inside a limit, for instance in the FSI solver the CFL condition is used. The CFL condition states that the time step must be less than the time for a wave to travel across one mesh cell ($k < h/U$), then the algorithm is CFL stable. Nevertheless, small time steps guarantee more stable run but they rise the computational cost and make the simulation more slow. In addition, too small time step can also affect other quantities, which depend on the time steps, for example the stabilization coefficient d_1 in equation (3.10).

Fig. 3.10 shows that a larger time step (which is inside a CFL limit) can be more stable than small time steps. The reason is the stabilization coefficient d_1 depends on time steps k. When k is too small, the term $1/k^2$ will dominate the term $(U - W)^2/h^2$ and defect the effect of d_1-stabilization. One suggestion is to modify the stabilization coefficient d_1 so that the terms $1/k^2$ and $(U - W)^2/h^2$ are in the same order.

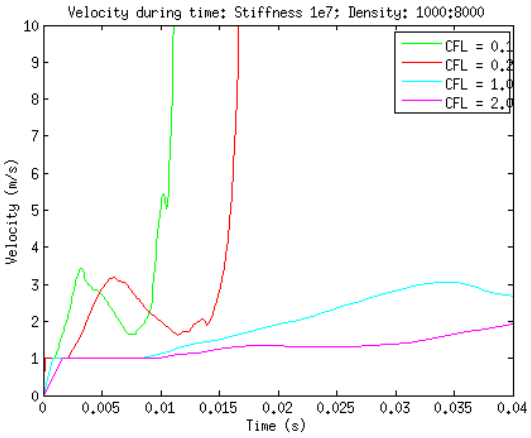


Figure 3.10: Velocity magnitude result of the solver with different time steps. The time steps depend on the minimum cell size, the inflow velocity and a CFL constant: $k = CFL * hmin/Ubar$

Stabilization coefficients C_1, C_2

We study the scaling of the stabilization coefficients C_1, C_2 mentioned in equations (3.10) and (3.16).

Scaling up the stabilization coefficients can help to stabilize the solver. The plots in Fig. 3.11 illustrate the less sensitive results by scaling up the stabilization coefficients by a factor of 2. However, increasing stabilization coefficients means that we allow a higher perturbation of the system, which leads to the fact that the output velocity is more stable but it can converge to a number that further away from the realistic velocity.

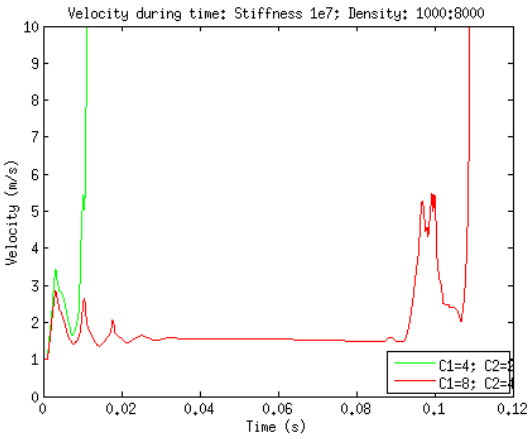


Figure 3.11: Scaling up the stabilization coefficients by a factor of 2

Chapter 4

Application of the Unicorn FSI Solver

We use the FSI solver to simulate the target problem mentioned in chapter 1. We mention here tasks that need to be performed as an user of the FSI solver.

4.1 Simulation preliminaries

4.1.1 Mesh generation

Software ANSA [15] is used in this step.

Firstly, we read CAD input data into ANSA and cleaned up the geometry. After that, surface meshing and volume meshing were done with the help of ANSA Batch Manager. Meshes from ANSA is exported to NAS format, then GMSH [16] converted .nas to .msh. After all, dolfin-convert is used to convert .msh to .xml. The final .xml mesh is the input of the solver.

We exported here two meshes: the whole domain mesh and bar mesh (solid mesh).

Properties	**Solid mesh**	**Whole mesh**
Number of vertices	18004	170999
Number of cells	71571	918062
Range of cell size	0.8 - 1.0 mm	0.8 - 5.0 mm

Table 4.1: Mesh properties

4.1.2 Parameters

Parameter	Notation	Value	Unit
solid stiffness	μ_s	193e9 (steel); 1e7 (rubber)	N/m^2
solid density	ρ^s	8000 (steel); 1100 (rubber)	kg/m^3
fluid viscosity	ν	* Table 4.3	m^2/s
fluid density	ρ^f	1000 (water); 1 (air)	kg/m^3
fluid inflow	$Ubar$	1	m/s
simulation time	T	* 5.0	s
time step	k	* $0.01hmin/Ubar$	s
external volume force	f	* 2.5	N/m^3
number of samples	N	* 200	-

** Adjustable by user*

Table 4.2: Parameter list

Temperature ^oC	Water dynamic viscosity $(kg/ms) \times 10^{-3}$	Temperature ^oC	Air dynamic viscosity $(kg/ms) \times 10^{-5}$
7.4	1.41	15	1.864
8.4	1.36857	27	1.983
10	1.30627		
11.7	1.24497		
15	1.13857		

Table 4.3: Water and air viscosity

The Reynolds number is defined by $Re = \frac{\rho u L}{\mu_v} = \frac{uL}{\nu}$. For this problem, we take $L = 30mm$, the minimum distance from bar to pipe wall; $u = 1m/s$ is the same as the inflow velocity. The Reynolds number is:

$$Re = \frac{1m/s * 30 \times 10^{-3}m}{1.41 \times 10^{-6}m^2/s} \approx 22 \times 10^4$$

4.1.3 Bending force

Force for Beam with contact problem

This problem is similar to the case Beam fixed at one end and supported (pined) at the other end in Fig. 4.1. The bending force used in this case is concentrated force

at the middle point of the bar.

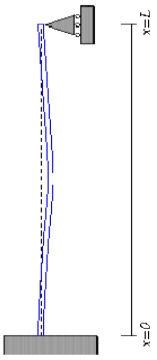


Figure 4.1: Beam with contact problem (supported pin)

Denote:
F is the concentrated force at point C
$a = 0.75m$ - distance from C to free end of the beam
$b = 0.75m$ - distance from C to fixed end of the beam
$L = 1.5m$ - the length of the beam
$B = 20mm = 0.02m$ - the width of the beam
$H = 8mm = 0.008m$ - the thickness of the beam
$\Delta = 10mm = 0.01m$ - deformation
$E = \mu_s = 193e9N/m^2$ the Young modulus/stiffness of steel
$I = \frac{BH^3}{12} = 853e - 12m^4$ Inertial momentum

According to [17] we have formula:

$$\Delta = \frac{Fa^2b^3}{12EIL^3}(3L + a) \tag{4.1}$$

To bend the bar to the maximum amplitude $\Delta = 10mm$ away from the initial position, an external force $F = 53.53N$ needed to be applied in the middle area of the bar. We measure the distance between mid-bar point and the center. When this distance reaches to $10mm$, we stop the external force.

Depending on the implementation of solver Unicorn, we convert the bending force F in N to force density $f = F/V$ in N/m^3. Where V is the volume (force-area), on which force is applied.

Physically, force should be applied in the middle of bar within a small thickness $1mm$, then $V \approx 16e-8m^3$ and $f \approx 335e6N/m^3$. Due to the instability of the solver if f is too large, we make f smaller by considering that concentrated force can be converted into load force on whole bar. Then we make force-area be the whole bar and $V = BarVolume = 1.5 \times 0.02 \times 0.008 = 24 * 10^{-5}m^3$ and consequently $f = F/V = 223100N/m^3$.

Force for cantilever beam

For a cantilever beam, the formula for deformation is:

$$\Delta = \frac{Fb^3}{3EI} \tag{4.2}$$

To bend the bar top $1cm$ with the solid stiffness $\mu_s = 10^7 N/m^2$ we need a force

$$F = \frac{3EI\Delta}{b^3} = 0.000607N$$

Converting to force density, we have

$$f = F/V = 2.53N/m^3. \tag{4.3}$$

Smoothly increasing force

In consideration of avoiding shock that can be created by the force, we introduce the force as a smooth-step function by multiplying f with a function $ramp(t)$. Consequently, the force increases from zero to f.

$$ramp(t) = \begin{cases} 0 & t \text{ in } [0, t_1] \\ 6t_t^5 - 15t_t^4 + 10t_t^3 & t \text{ in } (t_1, t_2],\ t_t = \frac{t-t_1}{t_2-t_1} \text{ in } (0, 1] \\ 1 & t \text{ in } (t_2, T] \end{cases} \tag{4.4}$$

The force should increase gradually, so that the bar will not accelerate too much. Ideally, during the bending movement, the bar should always be in the equilibrium of the bending force and its internal stress force.

4.2 Implementation preliminaries

4.2.1 Tracking points

In order to measure the frequency and the amplitude of the vibrating bar, we keep track of a point, which characterizes the vibration. For instance, in the cantilever case this point is a point on the top of the bar. In the case of a pinned bar, the interesting point lays in the middle of the bar. The tracked point has approximately the maximum amplitude in comparison to other points of the bar and can describe the interesting quantities the best. During a simulation, the coordinates of this tracked point are printed out in each time step.

4.2.2 Checking the end of the bending process

In order to check if the bar has reached to the desired bended position ($1cm$ away from the initial position), we have two checks:

- A vibrating point is tracked as described above. In each time step, we check the distance of the current position of this point to the initial coordinates of it. If this distance is larger than $1cm$, we flag that "the bar has bended".

- Another check is when the force function has reached to its maximal value. However, when the force has reached to its maximal value, the bar may still be on the way to its maximum bended position. Therefore, the force should be maintained for a while until the bar has reached the equilibrium position with this force and stops moving.

4.2.3 Boundary Conditions

Boundary conditions for momentum equation consist of:

- Inflow boundary: Specified inflow velocity $Ubar$

- Fixed boundary: Boundary on the bottom of the bar should be fixed (momentum equals zero)

- External wall of the pipe: no-slip BC

- Interface between the solid and the fluid: We applied here no-slip boundary, which means momentum is zero.

Boundary condition for continuity equation is:

- Outflow boundary: pressure in outflow boundary is zero.

4.3 Results

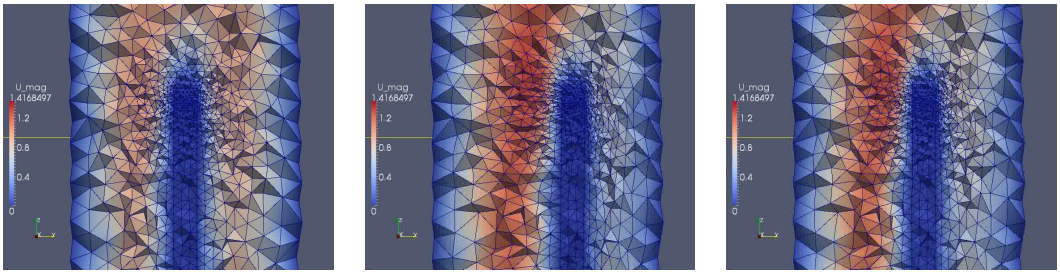
In this section we introduce test results of the geometry described in Section 3.3. The bar is made from rubber (stiffness $1e7kg/ms$) and was bended and released in an air flow (inflow velocity $1m/s$) with fluid density $1kg/m^3$. We also used the unity density for the solid part as it is a stable setting. The simulations were carried out with two different fluid viscosities.

In the simulation, first the bar is bended by an external force (the bending process) until the top of it reached nearly $1cm$ in the y-direction. Then the bar was released (the force was removed) and freely vibrated. All of these processes were simulated by the FSI-solver. The obtained results from the FSI-solver of the vibrating rubber bar give a good condition for simulating the vibration of a steel bar as in the real experiments. It also indicates that the new FSI-solver is able to simulate a real industrial FSI problem.

4.3.1 Simulation with high viscosity of the fluid

The first result that we will analyse is the case of high fluid viscosity. As known, viscosity represents the "resistance" of the fluid to movements, in particular the movement of the bar that immersed into it. A higher viscosity implies a larger resistance.

In this test case we set the "artificial" viscosity to the fluid as in (3.18). Fig. 4.2 illustrates the region of the mesh around the top of the bar. The bar region is the "blue straight" region.



(a) (b) (c)

Figure 4.2: A slide cut on the top of the bar (a) in the initial position, (b) in bended (extreme) position and (c) when it was coming back to the initial position

In Fig. 4.2(a), the bar is in its initial position. Then an external force is applied on the bar to bend it (to the right as seen in Fig. 4.2). During the movement of the bar, most of the fluid elements around the bar are deformed: they are stretched in one side of the bar and are squeezed in the other side. The deformation of the elements may result in bad quality of them (flat elements with small angels), which cause the FEM stiffness matrix to be ill-conditioned [18]. To avoid this problem, the Elastic Smoother of the FEniCS is used to smooth the mesh and improve the quality of the elements in each time step. The images in Fig. 4.2 show the smoothed elements by the Elastic smoother.

Fig. 4.2(b) shows the top of the bar when it reached to the equilibrium position, meaning that the internal stress force equals the external force (volume force $f = 2.5N/m^3$). As shown in Fig. 4.3, the bar has not reached $1cm$ in the y-direction as calculated by equation (4.3). This result may be explained by the high viscosity of the fluid.

Next, the force is removed. Due to the internal stress force, the bar is moving back toward its initial position, which is shown in Fig. 4.2(c).

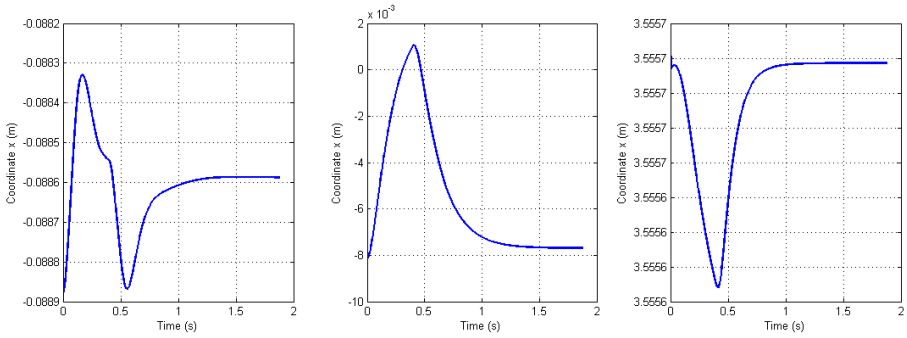
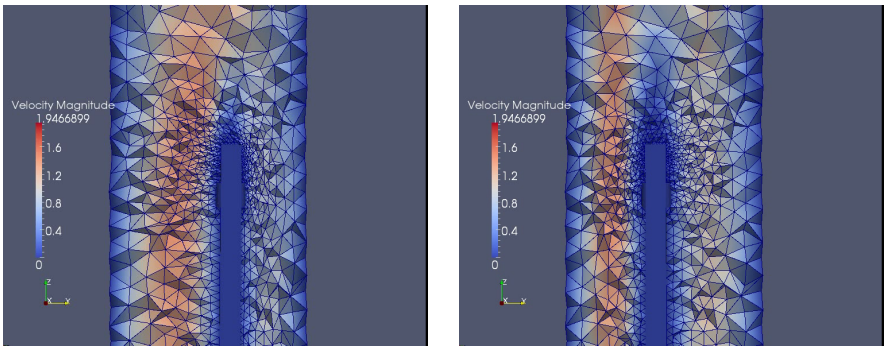


Figure 4.3: Coordinate changes of the top of a rubber bar during bending and vibration processes with fluid viscosity $\mu_v := \mu_v + \rho h U$

Fig. 4.3 illustrates the coordinates of one point on the top of the bar during the whole process described above. Note that since the force was applied in the y-direction, we are more interested in the y-coordinate of the point. As can be seen, when moving toward the initial position after being released, the bar did not cross the initial position to move further to another side (no vibration). Indeed, it was reaching to the initial position and the movement was damped quickly. The curve of the movement is similar to the one of a critical damping situation. As mentioned, this situation relates to the high viscosity of the fluid, which resists and damps out the movement of the bar.

4.3.2 Simulation with air viscosity of the fluid

In contrast to the result of the high viscosity above, in this test case we used the viscosity of air and the initial condition of the bar is in its bended position. Two snapshots of the bar movement are demonstrated in Fig. 4.4.



(a) (b)

Figure 4.4: A slide cut on the top of the bar: (a) in bended (extreme) position at $t = 0.4s$ and (b) during the vibration

In Fig. 4.4(a), the bar is in its bended position. No force is applied on the bar from this moment. Thereafter, the bar is moving toward its initial position and crosses this position. Further, the bar moves to the other side and reaches to a maximum amplitude of $4.2mm$. Fig. 4.4(b) shows the position of the bar at this moment. Then, the bar keeps vibrating with a damping effect, which will be analysed below.

The coordinates changes of one point on the top of the bar are shown in Fig. 4.5. In this figure we also show the coordinates of the bar during the bending process (from $t = 0$ to $t = 0.4s$) for a better detection of the initial position.

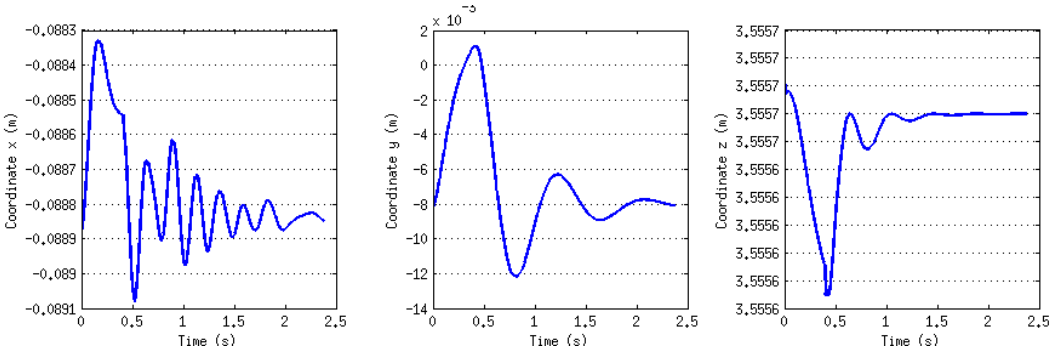


Figure 4.5: Coordinate changes of the top of a rubber bar during bending and vibration processes in an air flow

Based on the observation of the amplitude in Fig. 4.5, the period of one oscillation in the y-direction is $T = 0.8s$, the vibration frequency calculated is $f = 1/T = 1.25Hz$. Using the formula of the damping ratio from solid mechanics, for instance in [19], it holds that:

$$\ln\left(\frac{x_1}{x_2}\right) = \frac{2\pi\delta m}{\sqrt{1-\delta^2}} \tag{4.5}$$

where:
x_1 *is the amplitude of the first oscillation at time* t,
x_2 *is the amplitude of the oscillation that occurs 'm' cycles later (occurs* mT *second later),*
δ *is the damping ratio.*

It can be measured that $x_1 = 9.2mm$; $x_2 = 1.9mm$ for $m = 1$. Applying the equation (4.5) we calculated the damping ratio of this oscillation:

$$\delta = 0.2435$$

This is the under-damped oscillation ($\delta < 1$), in which the amplitude generally reaches its zero position while oscillating around this zero.

In conclusion, the external bending force can mimic the real physical force in this problem. The simulation results reflect the physical rules of viscosity. Furthermore, the vibration of the rubber bar convinces that the new FSI-solver is able to handle this industrial problem.

Chapter 5

Discussion and Future work

5.1 Summary

The problem setting, simulation preliminaries and parameters preparation are done for the target FSI problem. The user interface code is rearranged into functions and classes with comments, some new features are implemented, which provide more ease for future users to work with the FSI solver.

The generality to choose density is successfully implemented in the FSI solver, which makes both the resulting velocity and pressure closer to realistic values.

The force term is included in the mathematical model, implemented and tested for the cantilever problem. The test result is reasonable.

Parameter studies were performed to test the sensitivity of the FSI solver. Looking at the test results we can see the influence of different aspects to the stability of the solver.

Some testing results were obtained, which promise a future success for simulating the real experiments.

5.2 Future improvements of the FSI solver

The tests in Section 3.3 show that there are still numerical instability problems in the new FSI solver. The sources of this problem may come from different aspects: the applied UC mathematical model, possible bug(s) in the implementation, mesh quality issues, insufficient combination of physical and simulation parameters.

On the external wall, at the moment there is only one working choice for boundary condition (no-slip), which lower the physical reality of the simulation. The slip-boundary condition choice is implemented but not well-tested. Future work should include the testing performance of the slip BC.

5.3 Contact Modelling

5.3.1 A Simplified model

The discussed experiment in Section 1.3 has a contact problem (contact between the balls on the cylinder tips and the top of the bar). Solving this contact problem demands more analysis and workload (one solving method is mentioned in Section 5.3.2). To make the simulation easier, we first suggest to simplify this problem as described below.

We choose one point (vertex), where the intersection of the four cylinders is (see Fig. 1.1). This point is an inner point laying inside the top of the bar. The point will be fixed from sides (fix in horizontal x- and y-directions) and be free to slide up and down (free in vertical z-direction). The balls on the cylinder tips are removed to leave empty spaces between the cylinders and the bar. Therefore, we simplify the problem by neglecting the contact problem between the top of the bar and four cylinders, but we still expect the same effect as four small balls on cylinders create.

Furthermore, the four cylinders are removed to save computational cost, since the cylinders do not affect the interesting flow and the bar movement.

Mathematically, we enforce the fixed point by zero out the x- and y-component of the corresponding vertex in the right-hand-side b of the momentum LSE $Ax = b$; the z-component is untouched. Hence, during the simulation the result velocity of this fixed point in the horizontal x- and y-directions are zero; and in the vertical z-direction is kept. In the implementation, we have created a Mesh function, which is $true$ in fixed-point-vertex, and $false$ otherwise. To be coupled with this mesh function, we also pass a vector with a special value (extremely large value that is unreasonable for real physical velocity) for MeshBC class to "recognize" the fixed-point vertex. MeshBC class will enforce zero for x- and y-component of this fixed-point vertex. We refer to Appendices B.3 and B.4 for details.

This simplified model was implemented and tested in the old FSI solver. Unfortunately, the result was unstable. It could be related to several aspects that had not been fixed in the old solver. In future work, we can test this model again with the new solver after solving the instability problems.

5.3.2 Contact model

In the contact model we keep the original setting for the mesh (with four cylinders). The ball areas between the four pins and the bar are called *four interacting areas* and are set to be solid with a special material (defined by user), for instance rubber. In other word, we make the four balls become rubber.

The contact model is implemented in the new FSI solver. The preliminary results showed that in the time period under a bending force, the middle point of the bar moved much more than the top of the bar (Fig. 5.1), which promises a possible approach for our target problem. However, the question of which contact material should be used plays an important role in this model.

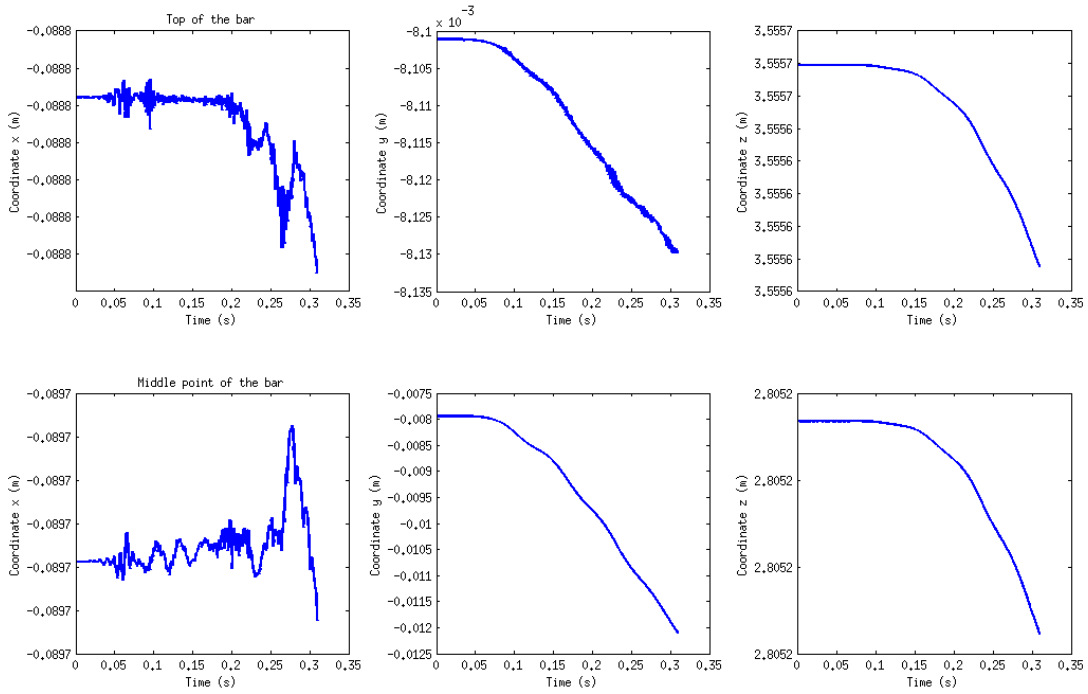


Figure 5.1: Coordinate change of the top and the middle of the bar in the contact-model test

Bibliography

[1] FEniCS team. FEniCS project. `http://fenicsproject.org`, 2003. [Online; Accessed: 2013-09-30].

[2] Stuart J. Price Michael P. Paidoussis and Emmanuel de Langre. *Fluid-Structure Interactions Cross-Flow-Induced Instabilities*. Cambridge University Press, 2010.

[3] S. Rugonyi and K. J. Bathe. On finite element analysis of fluid flows fully coupled with structural interactions. *Tech Science Press*, 2:195–212, 2001.

[4] Michael Stoeckli. A unified continuum fluid-structure interaction solver using an ALE finite element method. Master thesis, KTH, 2007.

[5] Ren de Borst Erwin Stein and Thomas J.R. Hughes. *Encyclopedia of Computational Mechanics*, volume 1: Fundamentals. John Wiley and Sons, Ltd., 2004.

[6] J. Jansson J. Hoffman and Michael Stockli. Unified continuum modelling of 3d fluid-structure interaction. *Preprint KTH-CTL-1003 Computational Technology Laboratory*, 2009.

[7] Johan Hoffman and Claes Johnson. *Computational Turbulent Incompressible Flow: Applied Mathematics Body and Soul Vol 4*. Springer, PDF version available at `www.csc.kth.se/jhoffman/pub/v4.pdf`, 2006.

[8] Richard Fitzpatrick. *Fluid Mechanics*. The University of Texas at Austin, `http://farside.ph.utexas.edu/teaching/336L/Fluidhtml/`, 2012. [Online; Accessed: 2013-04-03].

[9] T. Dunne and R. Rannacher. Adaptive finite element simulation of fluid structure interaction based on an eulerian variational formulation. *Springer*, pages 110–145, 2006.

[10] Ulrich Ruede. *Lecture notes for SIWIR-2*. FAU Erlangen-Nuremberg, 2012.

[11] P. Hansbo K. Eriksson, D. Estep and C. Johnson. *Computational Differential Equations*. Cambridge Univ. Press, 1996.

[12] E. Marchandise G. Compere and J.-F. Remacle. Transient adaptivity applied to two-phase incompressible flows. pages 1923–1942. Journal of Computational Physics 227, 2008.

[13] John Douglas Faires Richard L. Burden. *Numerical Analysis*. Brooks/Cole-Thomson Learning, seventh edition, 2001.

[14] Dmitri Kuzmin. *A Guide to Numerical Methods for Transport Equations*. FAU Erlangen-Nuremberg, 2010.

[15] BETA CAE Systems S.A. Ansa - advanced multidisciplinary cae pre-processing tool.

[16] Christophe Geuzaine and Jean-François Remacle. GMESH - A three-dimensional finite element mesh generator. `http://geuz.org/gmsh/`. [Software; Accessed 2013-04].

[17] The James F. Lincoln Arc Welding Foundation Adam Sprecace. Beam Load Equations. `http://sprecace.com/node/43`. [Accessed 2013-04].

[18] Jonathan Richard Shewchuk. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. *In 11th International Meshing Roundtable*, pages 115–126, 2002.

[19] D.J.Dunn. Solid mechanis - Dynamics - Tutorial: Damped vibrations. `http://www.freestudy.co.uk/dynamics/damped%20vibrations.pdf`. [Accessed 2013-11].

Appendix A

Form Files Implementation

Listing A.1: *NSEMomentum3D.form* script:

```
cell = "tetrahedron"

K1 = VectorElement("Lagrange", cell, 1)
# Dimension of domain
d = K1.cell_dimension()
K2 = FiniteElement("Lagrange", cell, 1)
K3 = FiniteElement("Discontinuous Lagrange", cell, 0)
K4 = VectorElement("Discontinuous Lagrange", cell, 0)
K5 = VectorElement("Discontinuous Lagrange", cell, 0, d * d)

v = TestFunction(K1)
U1 = TrialFunction(K1)
UP = Function(K1)
U0 = Function(K1)
Uc = Function(K1)
Um = Function(K4)

P = Function(K2)
Phydro = Function(K2)
nu = Function(K3)
d1 = Function(K3)
d2 = Function(K3)
ff = Function(K1)
fc = Function(K1)  # coriolis force
k = Function(K3)

rho_f = Function(K2)
theta = Function(K3)
sigma = Function(K5)

mu = Function(K3)
lmbda = Function(K3)

WP = Function(K1)
```

```
35 Wm = Function(K4)
36
37 b = Function(K3)
38
39 rho_s = Function(K3)
40
41 rho = mult(theta, rho_f) + mult(1.0 - theta, rho_s)
42
43
44 def tomatrix(q):
45     return [ [q[d * i + j] for i in range(d)] for j in range(d) ]
46
47 sigmaM = tomatrix(sigma)
48 beta   = Function(K2)     # friction parameter
49 tau1   = Function(K)
50 tau2   = Function(K)
51
52
53 def ugradu(u, v):
54     return [dot(u, grad(v[i])) for i in range(d)]
55
56 def E(e, mu, lmbda):
57     Ee = 2.0 * mult(mu, e) + mult(lmbda, mult(trace(e), Identity(d
       )))
58     return Ee
59
60 def epsilon(u):
61     return 0.5 * (grad(u) + transp(grad(u)))
62
63 beps = mult(b, epsilon(0.5*Uc))
64
65 Sf = mult(P, Identity(d)) - mult(nu, grad(Uc))
66 Ss = mult(P, Identity(d)) - mult(1.0, sigmaM) - beps
67 S = mult(theta, Sf) + mult(1.0 - theta, Ss)
68
69 Uc_ALE = Uc - WP
70 Um_ALE = Um - Wm
71
72 def f(u, v):
73     return -dot(mult(rho_f, ugradu(Uc_ALE, Uc)), v) + \
74         dot(S, grad(v)) + \
75         -mult(d1, dot(mult(rho_f, ugradu(Um_ALE, Uc)),
76                       mult(rho_f, ugradu(Um_ALE, v)))) + \
77         -mult(d1, dot(grad(P), mult(rho_f, ugradu(Um_ALE, v)))) +
           \
78         mult(d1, dot(mult(1.0 - theta, ff), mult(rho_f, ugradu(
           Um_ALE, v)) )) + \
79         -mult(d2 , dot(div(u), div(v))) + \
80         dot(mult(1.0 - theta, ff), v)
81
82 def fb(u, v):
83     return -beta*(dot(Uc,tau1)*dot(v,tau1) + dot(Uc,tau2)*dot(v,
       tau2))
84
```

```
def dfdu(u, k, v):
    return -dot(mult(rho_f, ugradu(Uc_ALE, u)), v) + \
        -mult(1 - theta, mult(k, dot(E(epsilon(u), mu, lmbda),
            grad(v)))) + \
        -mult(1 - theta, mult(b, dot(epsilon(u), grad(v)))) + \
        -dot(mult(theta*nu, grad(u)), grad(v)) + \
        -mult(d1, dot( mult(rho_f,ugradu(Um_ALE, u)),
                       mult(rho_f, ugradu(Um_ALE, v)))) + \
        -mult(d2, dot(div(u), div(v)))

def dfdub(u, k, v):
    return -beta*(dot(u,tau1)*dot(v,tau1) + dot(u,tau2)*dot(v,tau2
        ))

# cG(1)
def F(u, u0, k, v):
    uc = mult(0.5, u + u0)
    return (-dot(mult(rho, u), v) + dot(mult(rho, u0), v) + mult(k
        , f(uc, v)))

def dFdu(u, u0, k, v):
    uc = mult(0.5, u)
    return (-dot(mult(rho, u), v) + mult(k, dfdu(uc, k, v)))

def Fb(u, u0, k, v):
    uc = mult(0.5, u + u0)
    return (mult(k, fb(u, v)))

def dFdub(u, u0, k, v):
    uc = mult(0.5, u)
    return mult(1.0 * k, dfdub(uc, k, v))

a = (dFdu(U1, U0, k, v)) * dx + (dFdub(U1, U0, k, v)) * ds
L = (dFdu(UP, U0, k, v) - F(UP, U0, k, v)) * dx + (dFdub(UP, U0, k
    , v)) * ds - Fb(UP, U0, k, v) * ds
```

Listing A.2: *NSEContinuity3D.form* script:

```
cell = "tetrahedron"

K1 = VectorElement("Lagrange", cell, 1)
# Dimension of domain
d = K1.cell_dimension()
K2 = FiniteElement("Lagrange", cell, 1)
K3 = FiniteElement("Discontinuous Lagrange", cell, 0)

q  = TestFunction(K2)  # test basis function scalar
P  = TrialFunction(K2) # trial basis function
P0  = Function(K2) # trial basis function

rho = Function(K2)

uc = Function(K1)       # linearized velocity
```

```
16 Wc = Function(K1)        # mesh velocity
17
18 delta1 = Function(K3) # stabilization parameter
19
20 k = Function(K3)
21
22 theta = Function(K3)
23 f = Function(K1)
24 fc = Function(K1)
25
26 n = FacetNormal(cell)
27
28 i0 = Index() # index for tensor notation
29 i1 = Index() # index for tensor notation
30
31 def ugradu(u, v):
32     return [dot(u, grad(v[i])) for i in range(d)]
33
34 eps = 0.0
35 alpha = 2*k
36 Uc_ALE = uc - Wc
37
38 a = (eps*P*q + delta1*dot(grad(P), grad(q)) + alpha*dot(grad(P),
       grad(q)))*dx
39 L = (-q*div(uc))*dx + alpha*dot(grad(P0), grad(q))*dx - mult(
       delta1, dot( mult(rho, ugradu(Uc_ALE, uc)), grad(q) ))*dx + \
40   delta1*dot( mult(1.0 - theta, f), grad(q))*dx
```

Appendix B

New Features Implementation

Listing B.1: *TrackingPoint.h* script:

```
#ifndef __TRACKING_POINT_H
#define __TRACKING_POINT_H

#include <dolfin.h>
namespace dolfin { namespace unicorn {

class TrackingPoint : public SubDomain
/// Take coordinates of a point, find its index, fix this index
    during solving
{
public:
  // constructor
  TrackingPoint(Point fp, Mesh* inmesh, real tolerance)
  {
     this->tol = tolerance;
     localMesh = inmesh;
     // take index of tracked point
     index_fp = checkIndex(fp, localMesh, tol);

     // take coordinates of fixed_point by its index
     if (this->index_fp > -1) {
      Vertex v(*localMesh, (uint)index_fp);
      coord = v.x();
    }
    else {coord = NULL;}
  }

  // take coordinates of point and return index of nearest vertex
  int checkIndex(const Point p, Mesh* mesh, real tolerance) const
  {
    real min_dist = 100000;
    uint ver_idx;

    for (VertexIterator v(*mesh); !v.end(); ++v) {
```

```
34        Vertex& vertex = *v;
35        Point pref = vertex.point();
36
37        if (p.distance(pref) < min_dist )
38        {
39          min_dist =  p.distance(pref);
40          ver_idx = vertex.index();
41        }
42      }
43      if (min_dist < tolerance) {return (int)ver_idx;}
44      else {return -1;}
45    }
46
47    real* x() const // return current coordinates of the point
48    {
49      return coord;
50    }
51
52    int* index() const // return index of the point
53    {
54      return &index_fp;
55    }
56
57    bool inside(const real* p, bool on_boundary) const
58    {
59      bool result = false;
60      if (index_fp != -1 )
61      {
62        // detect point that lays in tracking_point region
63        if ((coord[0]-tol < p[0] && p[0] < coord[0]+tol) &&
64          (coord[1]-tol < p[1] && p[1] < coord[1]+tol) &&
65          (coord[2]-tol < p[2] && p[2] < coord[2]+tol))
66        {
67          result = true;
68        }
69      }
70        return result;
71    }
72
73    void printCoor() const
74    // print coordinates of the point
75    {
76       if (this->coord != NULL) {
77        // print the coordinates
78        std::cout.setf(std::ios::fixed);
79        std::cout.precision(16);
80        std::cout << "index: " << index_fp << " ::     " <<  coord[0]
              << "    " << coord[1] << "    " << coord[2]<< std::endl;
81      }
82      //else std::cout << "TRACKED POINT NOT FOUND" << std::endl;
83    }
84
85
86    real distance(Point p)
```

```
87    {
88      real dis;
89
90      if (this->coord != NULL) {
91        dis = sqrt( sqr(coord[0] - p[0]) + sqr(coord[1] - p[1]) +
              sqr(coord[2] - p[2]));
92      }
93      else {
94        dis = -1.0;
95      }
96
97      return dis;
98    }
99
100 private:
101    int index_fp;
102    real* coord;
103    Mesh *localMesh;
104    real tol;
105
106 };
107 }}
108 #endif
```

Listing B.2: Part of *main.cpp* script:

```
1  class ForceArea : public SubDomain
2  {
3  public:
4    ForceArea() {}
5
6    void init(Mesh& global_mesh)
7    {
8      for (VertexIterator v(global_mesh); !v.end(); ++v) {// loop
           over all vertices
9        Vertex& vertex = *v;
10       Point p = vertex.point();
11       real pver[] = {p[0], p[1], p[2]};
12
13       if (init_force_area(pver)) { // if vertex lays on force_area
14         TrackingPoint* foundVer = new TrackingPoint(p, &
               global_mesh, gmarg); // new TrackhingPoint to store the
                index
15         if (foundVer->x() != NULL) {
16           forceVertices.push_back(foundVer);
17           //foundVer->printCoor();
18         }
19         else {delete foundVer;}
20       }
21     }
22   }
23
24   /// geometrically identify force area in the mid of the bar
```

```
25    bool init_force_area(const real* p) const
26    { // inside of middle of the bar
27      return
28      ((p[0] > bar_xmin + 0.001 && p[0] < bar_xmax - 0.001) &&
29       (p[1] > bar_ymin + 0.001 && p[1] < bar_ymax - 0.001) &&
30       (p[2] > (bar_zmin + 0.45*(bar_zmax-bar_zmin))  && p[2] < (
             bar_zmin + 0.55*(bar_zmax-bar_zmin)))
31      );
32    }
33
34    /// check if one point lays on force_area, based on
          forceVertices
35    bool inside(const real* p) const
36    {
37      bool onForceArea = false;
38      real threshold = 0.001; // 1mm
39      Point pp(p[0], p[1], p[2]);
40
41      if (forceVertices.size() > 0) {
42        for (uint i=0; i < forceVertices.size(); i++) { // loop over
              all forceVertices
43          if (forceVertices[i]->distance(pp) < threshold) {
44            onForceArea = true;
45            break;
46          }
47        }
48      }
49      return onForceArea;
50    }
51
52  Private:
53    // attributes of class
54    Array <TrackingPoint*> forceVertices;
55
56  }forceArea;
```

Listing B.3: Partial fixed point implementation in *main.cpp*

```
1  Point fixed_p(-0.0888259574166, -0.00810110577457, 3.55565976544);
2
3  ...
4
5  TrackingPoint* fixpoint;
6  fixpoint = new TrackingPoint(fixed_p, &mesh, tolerance);
7
8  // Mesh function, value=true for fixed point, =0 otw
9  MeshFunction<bool>* fixed_vertices;
10 fixed_vertices = new MeshFunction<bool>(mesh, 0);
11 if (fixpoint->index_fp != -1)
12         fixed_vertices->set(fixpoint->index_fp, true);
13
14 // Helping vector to detect fixed point
```

```
15 dolfin::uint local_number = mesh.numVertices() - mesh.distdata().
       num_ghost(0);
16 Vector* fixed_p_values;
17 fixed_p_values = new Vector(3 * local_number);
18 for (uint i=0; i<fixed_p_values->size(); i++) fixed_p_values->
       setitem(i,-1.0e38);
19
20 ...
21 MeshBC p_fixedPoint(mesh, *fixpoint, *fixed_vertices,
       fixed_p_values);
```

Listing B.4: Partial fixed point implementation in *MeshBC class*

```
1  void MeshBC::applyMeshBC(Matrix& A, Matrix& As, Vector& b, Mesh&
       mesh, uint node, Array<uint>& nodes)
2  {
3  ...
4
5  if(node_values)
6  {
7    if (((*node_values)[nodes[0]] != -1.0e38) && ((*node_values)[
         nodes[1]] != -1.0e38) && ((*node_values)[nodes[2]] != -1.0e38
         )) {
8      //default case
9      bset(b, nodes[0], (*node_values)[nodes[0]]);
10     bset(b, nodes[1], (*node_values)[nodes[1]]);
11     if(nsdim == 3)
12       bset(b, nodes[2], (*node_values)[nodes[2]]);
13   }
14   else { // customized case
15     std::cout << "MESHBC:: FP enforce" << std::endl;
16     bset(b, nodes[0], 0.0);
17     bset(b, nodes[1], 0.0);
18     if(nsdim == 3) {} // not overwrite z-value
19   }
20 }
21 else
22 {
23   bset(b, nodes[0], 0.0);
24   bset(b, nodes[1], 0.0);
25   if(nsdim == 3)
26     bset(b, nodes[2], 0.0);
27 }
28
29 ...
30 }
```

TRITA-MAT-E 2014:01
ISRN-KTH/MAT/E—14/01-SE

www.kth.se