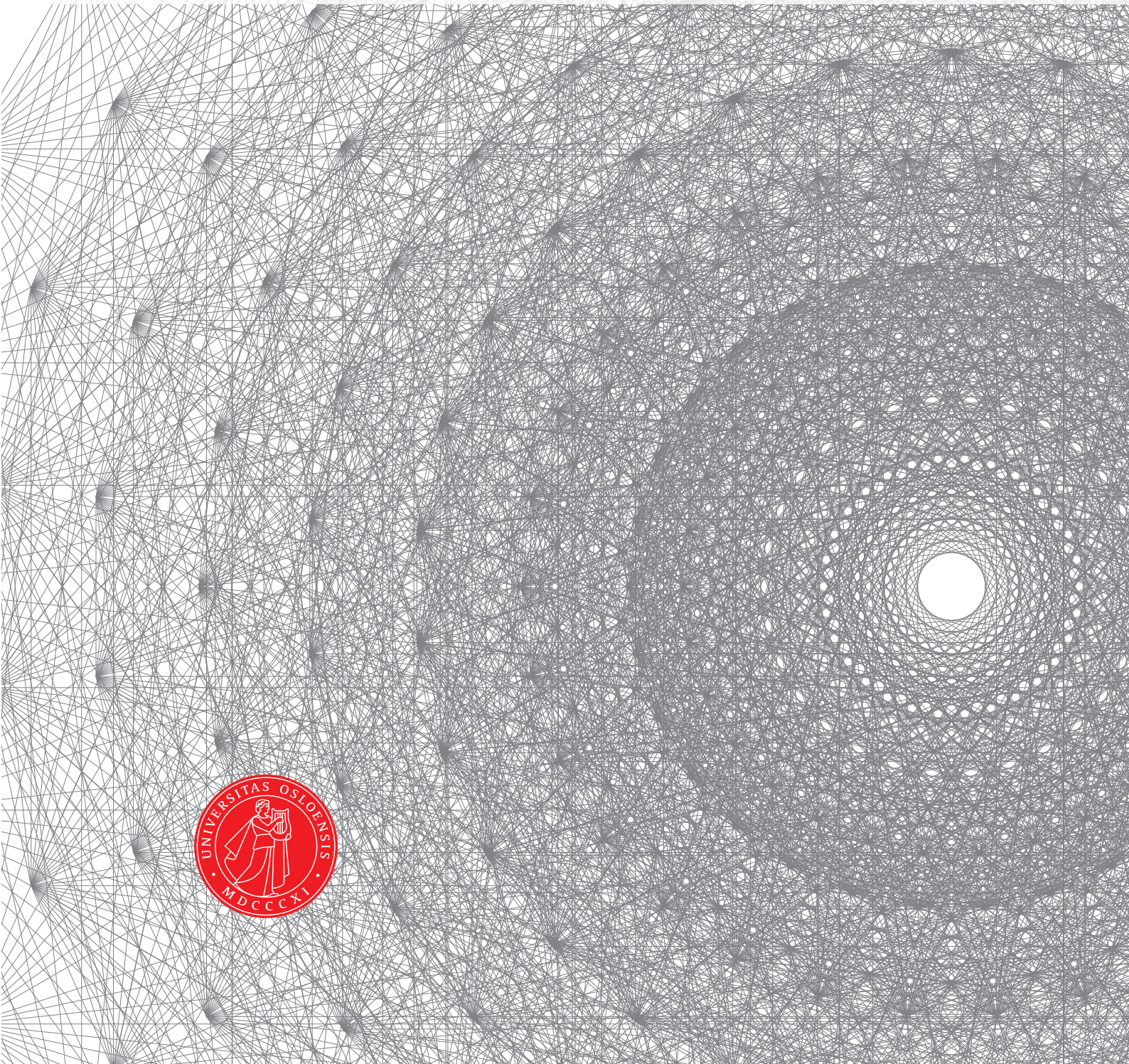
UiO : Department of Mathematics
University of Oslo

Monolithic solver for Computational Fluid Structure Interaction

Sebastian Gjertsen

Master's Thesis, Spring 2017



This master's thesis is submitted under the master's programme *Computational Science and Engineering*, with programme option *Mechanics*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Contents

Chapter 1

Introduction to Fluid-Structure Interaction

Fluid-Structure Interaction(FSI) can be observed all around us in nature. From a flag waving in the wind to a large windmill at sea. When we breath in air and our lungs expand. When our hearts fill up with blood and muscles contract to push the blood into our veins. These are all examples of fluids and structures interacting. A flag waving in the wind is mainly air(fluid) exerting force on the flag which is thin and usually made of light fabric. The fluid makes the flag flutter. The solid can also move the fluid. When we push our fingers into a water balloon or as mentioned breath air into our lungs. We deform the structure around the fluid, exerting force on the fluid.

A scary example of FSI is the collapse of the Tacoma Narrows Bridge in 1940 [1]. The bridge collapsed only two months after being opened. The collapse was due to strong winds(64 km/h) interacting with the bridge, making it flutter and ultimately collapse. No human life was lost in the collapse, but a Cocker Spaniel name Tubby left in a car was not so lucky.

Another example of FSI are inter-cranial aneurysms, which are balloon shaped geometries often occurring in a bifurcation(where a blood vessel splits into two vessels), due to weak vessel walls. Bursting of one of these aneurysms in the skull can have fatale consequences. The need to model FSI is as we can see is crucial to understand these problems.

Modelling fluid dynamics is called *Computational Fluid Dynamics* (CFD). I

Figure 1.1: Tacoma bridge still standing with large deformations

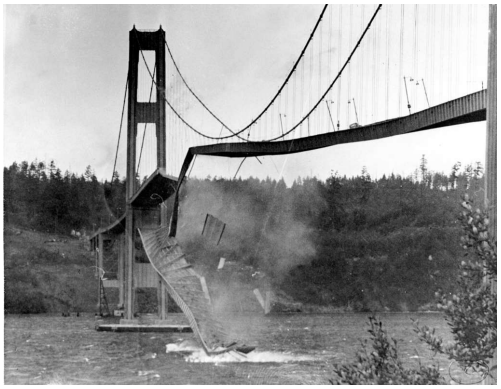


Figure 1.2: Tacoma bridge collapsed

would argue the same for FSI. That we call modeling FSI for *Computational Fluid Structure Interaction* (CFSI) Modeling FSI is used today when constructing for instance windmills. These windmills are usually rigid, hence giving a big difference in density between fluid and structure. The structure will in this case only give rise to small deformations. However modeling FSI in hemodynamics(dynamics of blood flow) deems more challenging. In these cases the densities in fluid and solid are much more similar and usually inducing large deformations. Also in both of these cases the fluid flow tends to transition to turbulence. We can see that these challenges gives the need to have rigid stabile FSI solvers. Solvers which can handle large deformations and high fluid velocity. Therefore the main goal of this master thesis is to build a framework to solve the FSI problem, investigating different approaches and schemes. The framework will be validated and verified using MMS, accompanying a wide range of benchmarks.

Chapter 2

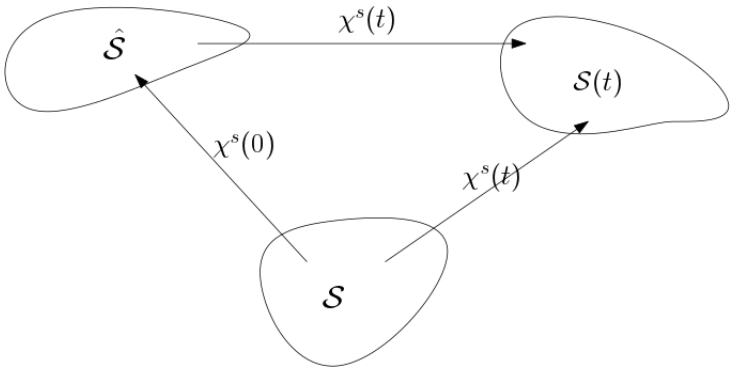
Continuum mechanics in different frames of reference

All materials are made up of atoms, and between atoms there is space. The laws that govern these atoms are complex and are very difficult to model. However materials like solids and fluids can be modeled if we assume them to exist as a continuum. This means that we assume that there exist no space inside the materials and they fill completely up the space they occupy. We can use mathematics with basic physical laws to model fluids and solids when they are assumed continuous. These laws are generally expressed in two frames of reference, Lagrangian and Eulerian. To exemplify these frameworks we can imagine a river running down a mountain. In the Eulerian framework we are the observer standing still besides the river looking at the flow. We are not interested in each fluid particle but only how the fluid acts as a whole flowing down the river. This approach fits the fluid problem as we can imagine the fluid continuously deforming along the river side.

In the Lagrangian description we have to imagine ourselves on a leaf going down the river with the flow. Looking out as the mountain moves and we stand still compared to the fluid particles. This description fits a solid problem nicely since we are generally interested in where the solid particles are in relation to each other. Modeling for instance a beam attached to a wall at one end and a weight at the other end. We can imagine the beam bending and to model this bending we need to where all the particles are compared to each other. The more the particles move in relation to each other the more stress there is in the beam.

In this chapter I will introduce both of these frameworks and the equations which are needed to model Fluid and Structure separately. I start with the Lagrangian description, by providing a short introduction to Lagrangian physics for the sake of completeness. Then introduce the solid and fluid equations. For a more detailed look on Lagrangian physics and the solid equation see [7].

2.1 Lagrangian physics



Let $\hat{\mathcal{S}}$, $\mathcal{S}$, $\mathcal{S}(t)$ be the initial stress free configuration of a given body, the reference and as the current configuration respectively. I define a smooth mapping from the reference configuration to the current configuration:

$$\chi^s(\mathbf{X}, t) : \hat{\mathcal{S}} \to \mathcal{S}(t) \tag{2.1}$$

Where $\mathbf{X}$ denotes a material point in the reference domain and χ^s denotes the mapping from the reference configuration to the current configuration. Let $d^s(\mathbf{X}, t)$ denote the displacement field which describes deformation on a body. The mapping χ^s can then be specified from its current position plus the displacement from that position:

$$\chi^s(\mathbf{X}, t) = \mathbf{X} + d^s(\mathbf{X}, t) \tag{2.2}$$

which can be written in terms of the displacement field:

$$d^s(\mathbf{X}, t) = \chi^s(\mathbf{X}, t) - \mathbf{X} \tag{2.3}$$

Let w($\mathbf{X}$,t) be the domain velocity which is the partial time derivative of the displacement:

$$w(\mathbf{X}, t) = \frac{\partial \chi^s(\mathbf{X}, t)}{\partial t} \tag{2.4}$$

2.1.1 Deformation gradient

To describe the rate at which a body undergoes deformation I will need to define a deformation gradient. Let $d(\mathbf{X}, t)$ be a differentiable deformation field in a given body, the deformation gradient is then:

$$F = \frac{\partial \chi^s(\mathbf{x}, t)}{\partial \mathbf{X}} = \frac{\partial \mathbf{X} + d^s(\mathbf{X}, t)}{\partial \mathbf{X}} = I + \nabla d(\mathbf{X}, t) \tag{2.5}$$

which denotes relative change of position under deformation in a Lagrangian frame of reference. We can observe that when there is no deformation. The deformation gradient F is simply the identity matrix.

We also need a way to change between volumes, from the reference ($\int dv$) to current ($\int dV$) configuration. This is defined with the Jacobian determinant, which is the determinant of the of the deformation gradient F:

$$J = \det(F) \tag{2.6}$$

The Jacobian determinant is used to change between volumes, assuming infinitesimal line and area elements in the current ds, dx and reference dV, dX configurations. The Jacobian determinant is therefore known as a volume ratio.

2.1.2 Strain

The relative change of location between two particles is called strain. Strain, strain rate and deformation is used to describe the relative motion of particles in a continuum. This is the fundamental quality that causes stress [13].

If we observe two neighboring points $\mathbf{X}$ and $\mathbf{Y}$. I can describe $\mathbf{Y}$ with:

$$\mathbf{Y} = \mathbf{Y} + \mathbf{X} - \mathbf{X} = \mathbf{X} + |\mathbf{Y} - \mathbf{X}| \frac{\mathbf{Y} - \mathbf{X}}{|\mathbf{Y} - \mathbf{X}|} = \mathbf{X} + d\mathbf{X} \tag{2.7}$$

Let $d\mathbf{X}$ be denoted by:

$$d\mathbf{X} = d\epsilon \mathbf{a}_0 \tag{2.8}$$

$$d\epsilon = |\mathbf{Y} - \mathbf{X}| \tag{2.9}$$

$$\mathbf{a}_0 = \frac{\mathbf{Y} - \mathbf{X}}{|\mathbf{Y} - \mathbf{X}|} \tag{2.10}$$

where $d\epsilon$ is the distance between the two points and $\mathbf{a}_0$ is a unit vector

We see now that $d\mathbf{X}$ is the distance between the two points times the unit vector or direction from $\mathbf{X}$ to $\mathbf{Y}$.

A certain motion transform the points $\mathbf{Y}$ and $\mathbf{X}$ into the displaced positions $\mathbf{x} = \chi^s(\mathbf{X}, t)$ and $\mathbf{y} = \chi^s(\mathbf{Y}, t)$. Using Taylor?s expansion $\mathbf{y}$ can be expressed in terms of the deformation gradient:

$$\begin{aligned} \mathbf{y} =& \chi^s(\mathbf{Y}, t) = \chi^s(\mathbf{X} + d\epsilon \mathbf{a}_0, t) && (2.11) \\ =& \chi^s(\mathbf{X}, t) + d\epsilon F \mathbf{a}_0 + \mathcal{O}(\mathbf{Y} - \mathbf{X}) && (2.12) \end{aligned}$$

where $\mathcal{O}(\mathbf{Y} - \mathbf{X})$ refers to the small error that tends to zero faster than $(\mathbf{X} - \mathbf{Y}) \to \mathcal{O}$.

If I set $\mathbf{x} = \chi^s(\mathbf{X}, t)$ It follows that:

$$\begin{aligned} \mathbf{y} - \mathbf{x} =& d\epsilon F \mathbf{a}_0 + \mathcal{O}(\mathbf{Y} - \mathbf{X}) && (2.13) \\ =& F(\mathbf{Y} - \mathbf{X}) + \mathcal{O}(\mathbf{Y} - \mathbf{X}) && (2.14) \end{aligned}$$

Let the **stretch vector** be $\lambda_{\mathbf{a}0}$, which goes in the direction of $\mathbf{a}_0$:

$$\lambda_{\mathbf{a}0}(\mathbf{X}, t) = F(\mathbf{X}, t)\mathbf{a}_0 \tag{2.15}$$

If we look at the square of λ:

$$\begin{aligned} \lambda^2 = \lambda_{\mathbf{a}0}\lambda_{\mathbf{a}0} &= F(\mathbf{X}, t)\mathbf{a}_0 F(\mathbf{X}, t)\mathbf{a}_0 && (2.16) \\ &= \mathbf{a}_0 F^T F \mathbf{a}_0 = \mathbf{a}_0 C \mathbf{a}_0 && (2.17) \end{aligned}$$

We have not introduced the important right Cauchy-Green tensor:

$$C = F^T F \tag{2.18}$$

Since $\mathbf{a}_0$ is just a unit vector, we see that this measures the squared length of change under deformation. We see that in order to determine the stretch one needs only the direction of $\mathbf{a}_0$ and the tensor C. C is also symmetric and positive definite $C = C^T$. I also introduce the Green-Lagrangian strain tensor E:

$$E = \frac{1}{2}(F^T F - I) \tag{2.19}$$

which is also symmetric since C and I are symmetric.

2.1.3 Stress

While strain, deformation and strain rate only describe the relative motion of particles in a given volume. Stress give us the internal forces between neighboring particles. Stress is responsible for deformation and is therefore crucial in continuum mechanics. The unit of stress is force per area.

I introduce the Cauchy stress tensor σ_s by the constitutive law of St. Venant-Kirchhoff hyperelastic material model:

$$\sigma_s = \frac{1}{J} F(\lambda_s(trE)I + 2\mu_s E)F^T \tag{2.20}$$

If we use this tensor on an area that is taking the stress tensor times the normal vector $\sigma_s \mathbf{n}$ we get the forces acting on that area.

Using the deformation gradient and the Jacobian determinant., I get the first Piola-Kirchhoff stress tensor P:

$$P = J\sigma F^{-T} \tag{2.21}$$

This is known as the *Piola Transformation* and maps the tensor into a Lagrangian formulation which will be used when stating the solid equation.

I also introduce the second Piola-Kirchhoff stress tensor S:

$$S = JF^{-1}\sigma F^{-T} = F^{-1}P = S^T \tag{2.22}$$

from this relation the first Piola-Kirchhoff tensor can be expressed by the second:

$$P = FS \tag{2.23}$$

2.2 Solid equation

The solid equation describes the motion of a solid. It is derived from the principles of conservation of mass and momentum. Stated in the Lagrangian reference system [13]:

$$\rho_s \frac{\partial d^2}{\partial t^2} = \nabla \cdot (P) + \rho_s f \tag{2.24}$$

written in terms of the deformation d, where I used the first Piola-Kirchhoff stress tensor. f is a force acting on the solid body. Considering the material law from the previous section. The stresses depend on the strain which again depends of the displacements d. The solid equation will be finalized by stating the different boundary conditions needed to solve a solid problem.

2.2.1 Solid Boundary Conditions

The solid moves within the boundary of $\partial\mathcal{S}$. On the Dirichlet boundary $\partial\mathcal{S}_D$ we impose a given value. These can be initial conditions or set to a value set to a designated spot on the domain. These initial conditions are defined for d and w:

$$d = d_0 \text{ on } \partial\mathcal{S}_D \tag{2.25}$$

$$w(\mathbf{X}, t)_0 = \frac{\partial d(t=0)}{\partial t} \text{ on } \partial\mathcal{S}_D \tag{2.26}$$

The forces on the boundaries need to equal an eventual external force $\mathbf{f}$. These are are enforced on the Neumann boundaries:

$$P \cdot \mathbf{n} = f \text{ on } \partial\mathcal{S}_N$$

2.3 Fluid equations

The fluid equation will be stated in an Eulerian framework. In this framework the domain has fixed points where the fluid passes through. The Navier-Stokes(N-S) equations are, like the solid equation, derived using principles of mass and momentum conservation. N-S describes the velocity and pressure in a given fluid continuum. They are here written in the fluid time domain $\mathcal{F}$ as an incompressible fluid:

$$\rho_f\big(\frac{\partial u}{\partial t} + u \cdot \nabla u\big) = \nabla \cdot \sigma_f + f \tag{2.27}$$

$$\nabla \cdot u = 0 \tag{2.28}$$

where u is the fluid velocity, p is the fluid pressure, ρ stands for density which, will be kept constant. f is body force and σ_f is the Cauchy stress tensor, $\sigma_f = \mu_f(\nabla u + \nabla u^T) - pI$, where I is the Identity matrix.

There does not yet exist an analytical solutions to the N-S equations, only simplified problems can be solved [20]. Actually there is a prize set out by the Clay Mathematics Institute of 1 million dollars to whomever can show the existence and smoothness of Navier-Stokes solutions [4], as apart of their millennium problems. Nonetheless this does not stop us from discretizing and solving N-S numerically. The field of modeling fluids numerically is known

as Computational Fluid Dynamics(CFD). And CFD is extensively used in for instance weather forecasts, construction of aircraft, and biomedical engineering.

One difficulty in the N-S equations is the nonlinearity appearing in the convection term on the left hand side. Non-linearity is most often handled using Newtons method or Picard iteration.

Before these equations can be solved we need to impose boundary conditions.

2.3.1 Fluid Boundary conditions

Lastly I need to impose boundary conditions. The fluid flows within the boundary noted as $\partial\mathcal{F}$. On the Dirichlet boundary $\partial\mathcal{F}_D$ we impose a given value. This can be initial conditions or set to zero as on walls with "no slip" condition. These conditions are defined for u p and d:

$$u = u_0 \text{ on } \partial\mathcal{F}_D$$

$$p = p_0 \text{ on } \partial\mathcal{F}_D$$

The forces on the boundaries need to equal an eventual external force $\mathbf{f}$. These are are enforced on the Neumann boundaries:

$$\sigma \cdot \mathbf{n} = f \text{ on } \partial\mathcal{F}_N$$

Chapter 3

Fluid Structure Interaction Problem

So far I have introduced the fluid and solid equations, and the descriptions in which they are stated. In this chapter I will introduce the full FSI problem. With all the equations, conditions and discretization needed to to build a solver.

When we compute FSI problems the computing domain is split into three parts, Fluid, Structure and Interface. Fluid and structure domains are as we know separated, and different constitutive equations are solved in each domain. The place in which fluid and structure meet is what we call the interface. The manner in which we treat the interface gives the two main methods for solving FSI problems [9]. The first method is called fully Eulerian. In fully Eulerian both the fluid and structure equations are solved in an Eulerian description. Here the interface is tracked across a still standing domain [18]. The fully Eulerian description is suited for fluid problems but is problematic for structure problems and certainly FSI where tracking of the interface across the domain is a difficult task. I therefore introduce the more commonly used method ALE. Which is the method that will be used in this thesis.

ALE stands for Arbitrary Lagrangian Eulerian. The ALE method entails formulating the fluid equations in an Eulerian framework and the solid in a Lagrangian framework. The entire domain itself moves with the structural displacements and the fluid moves through these points. In the ALE

framework we get best of both world. The structure equation will remain as previously stated (2.24), and we will need to change the fluid equations to take into account the moving domain changing the fluid velocity. Dealing with the movement of the domain is done in two ways. One way is to move the domain itself in relation to the structural displacements, and use this new domain to calculate the equations every time. Moving the domain gives advantages as we can explicitly represent the fluid-structure interface, and the equations are stated in a more familiar manner. But problems arise when there are large deformations in the solid giving large deformations into the fluid domain. Moving the mesh with large deformation can be a challenge.

In this thesis the ALE approach is used from a reference frame. Meaning we solve the equations on a initial, stress free domain, and use a series of mappings to account for the movements of the domain. The equations are mapped to the current domain, that is where the domain has moved to in the present time. It is the displacements in the solid that determines the mappings. The displacements in the solid and the interface are extrapolated into the fluid mesh. I will deal with the manner in which these extrapolations work in this chapter.

From a technical point of view, both moving the mesh and using a reference frame are equivalent [13]. Since the reference frame method does not need a function to move the mesh between each time iteration can be less time consuming. The interface is also located in the same position, making the interface easy to track.

Lastly in this chapter the equations are discretized. Discretization of the equations is tackled using either a monolithic or a partitioned approach. We will go into details later, but the overall idea is that a monolithic scheme involves computing all the equations together into one block, while the partitioned splits up into parts. That is we solve the fluid problem and structure problem separately. The advantage of this is that we can use existing solvers and techniques for each of the problems, but the difficulty is the treatment of the interface. The monolithic approach however, offers more stability but is more costly as the size of the problem increases[9].

I will start by introducing the mappings, this follows the notation and ideas from [13]:

3.1 Mapping between different frames of reference

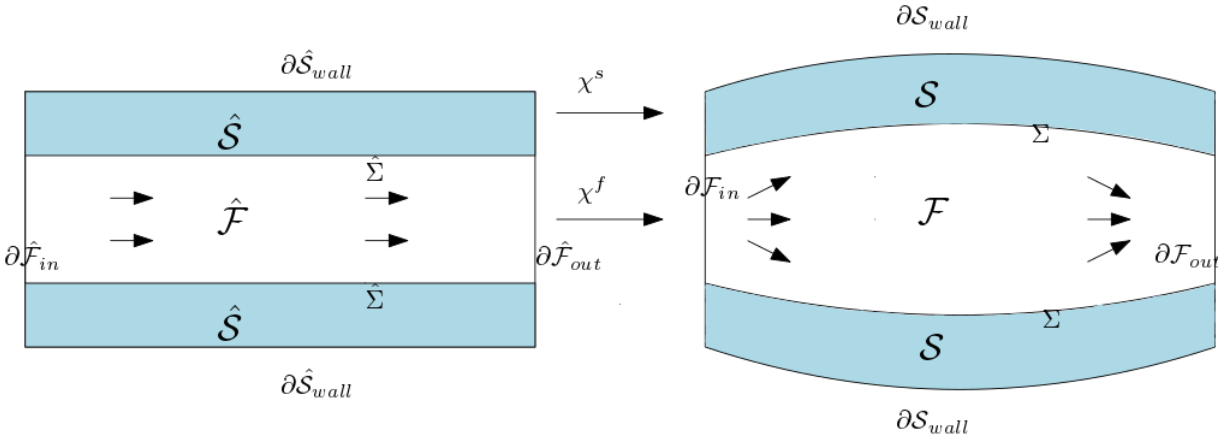


Figure 3.1: Mapping of domain from reference to current

Let $\hat{\mathcal{V}}$ be a reference volume and $\mathcal{V}(t)$ be the current time volume. Then using (2.5) and (2.6) we define a mapping between the volumes from the current to reference configurations:

$$\int_{\mathcal{V}(t)} 1dx = \int_{\hat{\mathcal{V}}} Jdx \tag{3.1}$$

The gradients acting on a vector $\mathbf{u}$ will also be mapped between current and reference configurations:

$$\int_{\mathcal{V}(t)} \nabla\mathbf{u}dx = \int_{\hat{\mathcal{V}}} J\nabla\mathbf{u}F^{-1}dx \tag{3.2}$$

Same for the divergence of a vector $\mathbf{u}$:

$$\int_{\mathcal{V}(t)} \nabla\cdot\mathbf{u}dx = \int_{\hat{\mathcal{V}}} \nabla\cdot(JF^{-1}\mathbf{u})dx \tag{3.3}$$

3.2 Governing equations for Fluid Structure Interaction

We will formulate the equations in the Eulerian, Lagrangian and the ALE description. We start by briefly talking about time derivatives in the different configurations. In the Lagrangian setting the total and partial derivatives are the same [21]:

3.2.1 Derivatives in different frameworks

$$D_t f(x,t) = \partial_t f(x,t) \tag{3.4}$$

in the Eulerian framework we have the following relation between total and partial derivatives:

$$D_t f(x,t) = \mathbf{u} + \partial_t f(x,t) \tag{3.5}$$

whilst in the ALE framework we extend this concept to take into account the motion of the domain:

$$D_t f(x,t) = \mathbf{w} + \partial_t f(x,t) \tag{3.6}$$

Here we can see that for the Lagrangian framework $\mathbf{w}$ is zero and for Eulerian $\mathbf{w} = \mathbf{u}$.

3.2.2 Solid equation

We express the solid balance laws in the Lagrangian formulation from the initial configuration

$$\rho_s \frac{\partial \mathbf{u}}{\partial t} = \nabla \cdot (P) \quad in \quad \hat{\mathcal{S}} \tag{3.7}$$

3.2.3 Fluid equations

The fluid domain is moving, and therefore we need to redefine the velocity in the convective term in (2.27) to account for the moving domain

$$\mathbf{u} \cdot \nabla \mathbf{u} \rightarrow (\mathbf{u} - \frac{\partial d_f}{\partial t}) \cdot \nabla \mathbf{u} \tag{3.8}$$

where d_f is the deformation in the fluid domain. We see that $\mathbf{w}$ is written as $\frac{\partial d}{\partial t}$. Now the actual fluid velocity will be $u - \frac{\partial d}{\partial t}$

The fluid equations are denoted from the initial configuration using the aforementioned mappings:

$$\int_{\mathcal{V}(t)} \rho_f \frac{\partial \mathbf{u}}{\partial t} dx = \int_{\hat{\mathcal{V}}} \rho_f J \frac{\partial \mathbf{u}}{\partial t} dx \tag{3.9}$$

$$\int_{\mathcal{V}(t)} \nabla \mathbf{u} (\mathbf{u} - \frac{\partial d}{\partial t}) dx = \int_{\hat{\mathcal{V}}} ((\nabla \mathbf{u}) F^{-1} (\mathbf{u} - \frac{\partial d}{\partial t}) dx \tag{3.10}$$

$$\int_{\mathcal{V}(t)} \nabla \cdot \mathbf{u} dx = \int_{\hat{\mathcal{V}}} \nabla \cdot (J F^{-1} \mathbf{u}) dx \tag{3.11}$$

$$\int_{\mathcal{V}(t)} \nabla \cdot \sigma_f dx = \int_{\hat{\mathcal{V}}} \nabla \cdot (J F^{-1} \hat{\sigma_f}) dx \tag{3.12}$$

$$\hat{\sigma_f} = -pI + \mu(\nabla \mathbf{u} F^{-1} + F^{-T} \mathbf{u}^T) \tag{3.13}$$

Assembling all these terms together with (2.27) gives the fluid equations from a reference frame:

$$\rho_f J \big(\frac{\partial \mathbf{u}}{\partial t} dx + (\nabla \mathbf{u}) F^{-1} (\mathbf{u} - \frac{\partial d}{\partial t}) \big) = \nabla \cdot (J F^{-1} \hat{\sigma_f}) + J \rho_f f \tag{3.14}$$

$$\nabla \cdot (J F^{-1} \mathbf{u}) = 0 \tag{3.15}$$

3.3 Mesh motion techniques

Using the ALE method the deformations from the structure through the interface is extrapolated to the fluid domain. The fluid domain itself acts as a structure, deforming according to the deformations from the structure domain. The choice of mesh motion technique is important for the overall

FSI problem to be calculated [22]. When large deformations occur, we need a good lifting operator to uphold the integrity of the computing domain. A poor choice will make the mesh overlap and singularities may occur. I will in this section present different mesh motion techniques, that act differently on the computational domain. In 5.2.2 the techniques will be tested and investigated.

3.3.1 Harmonic extension

For small to moderate deformations we can use a harmonic extension to lift the deformations. The harmonic extenstion uses the Laplace equation, transporting the deformations from the solid into the fluid domain. A variable $\alpha_u > 0$ can be multiplied to the Laplace equation, to control the amount of lifting of deformations to the fluid domain.

$$-\alpha_u \nabla^2 d = 0 \quad in \ \hat{\mathcal{F}} \tag{3.16}$$

$$d = 0 \quad on \ \partial\hat{\mathcal{F}}/\hat{\Sigma} \tag{3.17}$$

When using the harmonic lifting operator the variable α_u is very important when calculating moderate deformations. For small deformations a constant can be used for α_u. But for larger deformations we need to be a bit more clever. A good strategy for choosing α_u is proposed by Wick in [22], and further discussed in [17]. Which is an alpha that gets bigger when closer to the interface. This is a smart choice since this upholds the cell structure closer to the interface where most of the cell distortion appears.

3.3.2 Linear elastic extension

Since the fluid domain acts as structure we can use a linear elastic equation to lift the deformations from the solid into the fluid domain. The linear elastic equation is best known from computing solid problems [22]. The model is as follows:

$$-\nabla \cdot \sigma_{mesh} = 0 \quad in \ \hat{\mathcal{F}} \tag{3.18}$$

$$d = 0 \quad on \ \partial\hat{\mathcal{F}}/\hat{\Sigma} \tag{3.19}$$

$$\sigma_{mesh} = \alpha_\lambda (tr\epsilon) I + 2\alpha_\mu \epsilon \tag{3.20}$$

where $\epsilon = \frac{1}{2}(\nabla d + (\nabla d)^T)$. ϵ is the linearized version of the strain tensor (2.19). The parameters α_λ and α_μ are chosen to uphold domain quality. This is done by computing the parameters α_λ and α_μ from Youngs modulus and poisson ration.

3.3.3 Biharmonic extension

The last extension is the biharmonic extension. The biharmonic extension provides more freedom than the harmonic and linear elastic in choosing boundary conditions and choice of parameter $\alpha_u > 0$. This is because the biharmonic extension, extends the deformation in a manner that upholds the integrity of the cells even in large deformations, even with α_u as a constant. In its simplest form it is written as:

$$-\alpha_u \nabla^4 d = 0 \quad in \ \hat{\mathcal{F}} \tag{3.21}$$

The biharmonic extension is calculated with a mixed formulation where we introduce a new function ω (not to be confused with the deformation velocity), this function is added to the system so that we solve for 4 functions:

$$\omega = \alpha_u \nabla^2 d \quad and \quad -\alpha_u \nabla^2 \omega = 0 \ in \ \hat{\mathcal{F}} \tag{3.22}$$

with the two types of boundary conditions. The first being:

$$d = \partial_n d = 0 \quad on \ \partial\hat{\mathcal{F}} \setminus \hat{\Sigma} \tag{3.23}$$

The second imposes conditions on the function ω, and are written in terms of single component functions $d^{(1)}, d^{(2)}$ and $\omega^{(1)}, \omega^{(2)}$

$$d^{(1)} = \partial_n d^{(1)} = 0, and \quad \omega^{(1)} = \partial_n \omega^{(1)} = 0 \quad on \ \partial\hat{\mathcal{F}}_{in,out} \tag{3.24}$$

$$d^{(2)} = \partial_n d^{(2)} = 0, and \quad \omega^{(2)} = \partial_n \omega^{(2)} = 0 \quad on \ \partial\hat{\mathcal{F}}_{walls} \tag{3.25}$$

$$\tag{3.26}$$

Since the biharmonic extension is of fourth order character it will have a higher computational cost [13] than the harmonic or linear elastic.

3.4 Coupled Fluid Structure Interface conditions

In figure 3.1 we see a typical Fluid Structure Interaction. The fluid is surrounded by elastic walls, like a blood vessel. The inflow at $\partial\hat{\mathcal{F}}_{in}$, a change in pressure, or a motion of the solid determines the flow of the fluid. The fluid's stress on the walls causes deformation in the solid domain and vice versa. The interface is where these energies are transferred and we therefore need conditions on the interface.

Let $\Omega \in \hat{\mathcal{S}} \cup \hat{\mathcal{F}}$ be a global domain that is made up of the fluid, structure, and the interface. We define a global velocity function u that describes the fluid velocity in the fluid domain and the structure velocity in the structure domain. Using a global velocity makes the velocity continuous across the entire domain. Let interface be $\hat{\Sigma} \in \hat{\mathcal{S}} \cap \hat{\mathcal{F}}$.
The three interface comes from simple physical properties and consist of [13]:

- *Kinematic condition*: $\mathbf{u}_f = \mathbf{u}_s$ *on* $\hat{\Sigma}$. The fluid and structure velocities are equal on the interface. Meaning the fluid moves with the interface at all times.
  Since we use a global function for $\mathbf{u}$ in both fluid and structure domains, this condition is upheld.
  The fluid and solid velocities are usually in different coordinate systems, the solid velocity is then not available in Eulerian Coordinates. We instead link fluid velocity at the interface by using the fact that $\mathbf{u}_s = \frac{\partial d}{\partial t}$. Setting $\mathbf{u}_f = \frac{\partial d}{\partial t}$ at the interface.

- *Dynamic condition*: $\sigma_f n_f = \sigma_s n_s$ *on* $\hat{\Sigma}$.
  This relates to Newtons third law of action and reaction. The forces on the interface area, here written as the normal stresses are balanced on the interface. These will be written in a Lagrangian formulation:
  $J\sigma_f F^{-T} n_f = P n_s$ *on* $\hat{\Sigma}$.
  The dynamic condition is a Neumann condition that belongs to both subproblems.

- *Geometrical condition*: This condition says that the fluid and structure domains do not overlap, but rather that elements connect so the functions needing to transfer force are continuos across the entire domain.

3.5 Monolithic FSI Problem

As stated in the introduction there are generally two types of schemes used when simulating FSI. The first is the partitioned approach where fluid and structure are solved sequentially. This approach is appealing in that we have a wealth of knowledge and techniques on how to solve each of these kinds of problems in an efficient manner. The difficulty however is dealing with the interface. As we know there are kinematic and dynamic conditions needed in FSI, and the coupling of these conditions is where the problems arise. So called explicit coupled schemes are known to be unconditionally unstable for standard Dirichlet-Neumann strategies when there is a large amount of added-mass in the system [6], [19]. There are however, schemes which offer added-mass free stability with explicit coupling, where the interface is treated through a Robin-Neumann coupling. First for a coupling with a thin walled structure [5] and later with an extension to thick wall [6]. These schemes are rather complex and uses a number of techniques that are out of the scope of this thesis. (This may be in more detail in a later chapter (discussion and further work.))

The other approach is called monolithic. In the monolithic approach all of the equations are solved at once. This approach has the advantage of offering numerical stability for problems with strong added-mass effects [9], and are fully coupled. The disadvantage over the partitioned approach is that we loose flexibility when solving many equations simultaneously, and the problems can quickly become large and computationally costly.

3.6 Discretization of monolithic FSI equations

After introducing all the equations and boundary conditions needed to solve a FSI problem. We are ready to discretize the equations into one monolithic scheme. The equations will be discretized and solved using finite difference and finite element methods. I will introduce a so called θ-scheme which will make it easy to implement different schemes, by choosing a value for θ. I will briefly introduce the spaces needed to discretize, following the ideas and notations of [21]:

Spaces

Let $X \in \mathbb{R}^d, d \in \{1, 2\}$ be a time dependent domain we define:

$$\hat{V}_X := H^1(X), \qquad \hat{V}_X^1 := H_0^1(X) \tag{3.27}$$

H^1 indicating a Hilbert space and

$$\hat{L}_X := L^2(X), \qquad \hat{L}_X^0 := L^2(X)/\mathbb{R} \tag{3.28}$$

L^2 indicating a standard Lebesque space.

The trial and test spaces for the velocity variable in the fluid domain,

$$\hat{V}_{f,u}^0 := \{\hat{u} \in H_0^1(\mathcal{F}) : \hat{u}_f = \hat{u}_s \ on \ \hat{\Sigma}\} \tag{3.29}$$

and the same for the artificial displacement in the moving fluid domain:

$$\hat{V}_{f,d}^0 := \{\hat{d} \in H_0^1(\mathcal{F}) : \hat{d}_f = \hat{d}_s \ on \ \hat{\Sigma}\} \tag{3.30}$$

$$\hat{V}_{f,d}^0 := \{\hat{d} \in H_0^1(\mathcal{F}) : \hat{\psi}_f = \hat{\psi}_s \ on \ \hat{\Sigma}\} \tag{3.31}$$

Now that the spaces have been defined we are ready to discretize. The temporal discretization is done using finite difference schemes and the spatial is treated with the finite element method. I will employ a $\theta-$ scheme that will enables us to easily switch between schemes.

In the domain Ω and time interval [0,T]:

Find $U = \{\mathbf{u}, d, p\} \in \hat{X}_D^0$ where $\hat{X}_D^0 := \{\mathbf{u}_f^D + \hat{V}_{f,\mathbf{u}}^0\} \times \hat{L}_f \times \{d_f^D + \hat{V}_{f,\hat{f}}^0\} \times \{d_f^D + \hat{V}_{f,\hat{f}}^0\} \times \hat{L}_f^0 \times \hat{L}_s^0$ such that:

$$\int_0^T A(U)(\Psi)dt = \int_0^T \hat{F}(\Psi)dt \quad \forall \Psi \in \hat{X} \tag{3.32}$$

where $\Psi = \{\phi, \psi, \gamma\}$

$\hat{X} = \hat{V}_{f,\mathbf{u}}^0 \times \hat{L}_f \times \hat{V}_{f,d,\hat{\Sigma}}^0 \times \hat{V}_s^0 \times \hat{L}_f^0 \times \hat{L}_s^0$

I first introduce the scheme using, for simplicity, the harmonic mesh motion. Let $\mathbf{u}$ be a global function in the entire domain instead of $\mathbf{u}_f$ in the fluid and

$\mathbf{u}_s$ in the solid. Same for the test functions. This is done for ease of reading and also for the ease of implementation later.

The full monolithic FSI variational form reads:

$$
\begin{aligned}
A(U) = &(J\rho_f \partial_t \mathbf{u}, \phi) - (J(\nabla u)F^{-1}(\mathbf{u} - \partial_t d), \phi)_{\hat{\mathcal{F}}} && (3.33)\\
&+ (J\sigma_f F^{-T}, \nabla\phi)_{\hat{\mathcal{F}}} && (3.34)\\
&+ (\rho_s \partial_t \mathbf{u}, \phi)_{\hat{\mathcal{S}}} + \big(FS_s, \nabla\phi\big)_{\hat{\mathcal{S}}} && (3.35)\\
&+ (\alpha_u \nabla\mathbf{u}, \nabla\psi)_{\hat{\mathcal{F}}} + (\nabla\cdot(JF^{-1}\mathbf{u}), \gamma)_{\hat{\mathcal{F}}} && (3.36)\\
&+ \delta\big((\partial_t d, \psi)_{\hat{\mathcal{S}}} - (\mathbf{u}, \psi)_{\hat{\mathcal{S}}}\big) && (3.37)\\
&+ \big(J\sigma_{f,p}F^{-T}, \nabla\phi\big) && (3.38)
\end{aligned}
$$

The condition (3.37) , is weighted with a δ value. This is a critical detail for the program (detailed later) to run. The only two places where we use the test function ψ is on this condition and the lifting operator. The weighting says in a weak manner that this condition is important for the overall program. In [**?**] they have ρ_s in front of the condition which has no physical meaning, but will act as a weight. ρ_s is 7800 for steel and 1400 PVC [8] so these could work as weights. While in our program we set it to a large value $\delta = 10^{10}$. Whilst in [22], where the same scheme is used there is no weighting.
I will here formulate the *One step-θ scheme* from [21]. This θ scheme has the advantage of easily being changed from the backward (implicit), forward(excplicit) or Crank-Nicholson (implicit) scheme. The backward Euler scheme is of first order and is implicit in that it is using the newest time step appears on both sides of the equation. The Crank-Nicholson is of second order and both the current and previous time step is used. This scheme suffers from instabilities in its normal context, we will therefore look at a *shifted* Crank-Nicholson scheme.

We define the variational form by dividing into four categories, this may seem strenuous at first but the need for it will become evident when implementing the θ-scheme. The four divided categories consists of: a time term, implicit, pressure and the rest (stress, convection):

$$A_T(U) = (J\rho_f \partial_t \mathbf{u}, \phi) - (J(\nabla u)F^{-1}(\partial_t d), \phi)_{\hat{\mathcal{F}}} \tag{3.39}$$
$$+ (\rho_s \partial_t \mathbf{u}, \phi)_{\hat{\mathcal{S}}} + (\partial_t d, \psi)_{\hat{\mathcal{S}}} \tag{3.40}$$
$$A_I(U) = (\alpha_u \nabla \mathbf{u}, \nabla \psi)_{\hat{\mathcal{F}}} + (\nabla \cdot (JF^{-1}\mathbf{u}), \gamma)_{\hat{\mathcal{F}}} \tag{3.41}$$
$$A_E(U) = (J(\nabla u)F^{-1}\mathbf{u}, \phi)_{\hat{\mathcal{F}}} + (J\sigma_{f,u}F^{-T}, \nabla\phi)_{\hat{\mathcal{F}}} \tag{3.42}$$
$$+ \big(FS_s, \nabla\phi\big)_{\hat{\mathcal{S}}} - (\mathbf{u}, \psi)_{\hat{\mathcal{S}}} \tag{3.43}$$
$$A_P(U) = \big(J\sigma_{f,p}F^{-T}, \nabla\phi\big) \tag{3.44}$$

Notice that the stress tensors have been split into a velocity and pressure part.

$$\sigma_{f,u} = \mu(\nabla u F^{-1} + F^{-T}\nabla u) \tag{3.45}$$
$$\sigma_{f,p} = -pI \tag{3.46}$$

For the time group, discretization is done in the following way:

$$A_T(U^{n,k}) \approx \frac{1}{k}\big(\rho_f J^{n,\theta}(u^n - u^{n-1}), \phi\big)_{\hat{\mathcal{F}}} - \frac{1}{k}\big(\rho_f(\nabla u)(d^n - d^{n-1}), \phi\big)_{\hat{\mathcal{F}}} \tag{3.47}$$
$$+ \frac{1}{k}\big(\rho_s J^{n,\theta}(u^n - u^{n-1}), \phi\big)_{\hat{\mathcal{S}}} + \frac{1}{k}\big(J^{n,\theta}(d^n - d^{n-1}), \psi\big)_{\hat{\mathcal{S}}} \tag{3.48}$$

And the Jacobian is written with superscript θ as:

$$J^{n,\theta} = \theta J^n + (1-\theta)J^{n-1} \tag{3.49}$$

We can now introduce the *One step-θ scheme*: Find $U^n = \{u^n, d^n, p^n\}$

$$A_T(U^{n,k}) + \theta A_E(U^n) + A_P(U^n) + A_I(U^n) = \tag{3.50}$$
$$-(1-\theta)A_E(U^{n-1}) + \theta\hat{f}^n + (1-\theta)\hat{f}^{n-1} \tag{3.51}$$

We notice that the scheme is selected by the choice of θ. By choosing $\theta = 1$ we get the back Euler scheme, for $\theta = \frac{1}{2}$ we get the Crank-Nicholson scheme and shifted Crank-Nicholson we set $\theta = \frac{1}{2} + k$, effectively shifting the scheme towards the implicit side. $\hat{f}$ is the body forces which will be ignored in this thesis. The shifting toward the implicit side is important for long term stability for certain time step values. The shifting will be investigated in the next chapter.

3.6.1 Spaces and Elements

The velocity and pressure copling in the fluid domain must satisfy the inf-sup condition. If not stabilization has to added. We here need to define some spaces that will have these desired properties. We denote $u_h \in V_h$ and $d_h \in W_h$, here the finite element pair og pressure and velocity mush satisfy the inf-sup condition given in ALE coordinates:

$$\inf_{p_h \in L_{h,f}} \sup_{v_h \in V_{h,f}} \frac{(p_h, div(J_f F_f^{-1} u_h))_{\mathcal{F}}}{\|\|J^{\frac{1}{2}} p_h\|\|_{\mathcal{F}} \|\|J_f^{\frac{1}{2}} \nabla u_h F_f^{-T}\|\|_{\mathcal{F}}} \geq \hat{\Sigma}$$

A good choice of spaces will be P2-P2-P1 for velocity, displacement and fluid pressure respectively.

Chapter 4

FSI Implementation in FEniCS

Here we will look at the implementation of the monolithic FSI Code in FENiCS. Not every part of the code will be handled here, but hopefully enough so that someone familiar with FEniCS could in short time implement the scheme.

4.1 Mesh, mappings and stress tensors

The mesh was made with Gmesh, with a clear straight boundary splitting the fluid and solid domains. This is converted to a xml file and loaded into FEniCS.

```
mesh_file = Mesh("Mesh/fluid_new.xml")
```

The boundaries are defined using built in domain functions. We only here show the inlet where we specified where the spatial points are, and give it a mark value to be used when the Dirichlet conditions are set. This is also done using CellFunctions to define the fluid and structure domain.

```
Inlet = AutoSubDomain(lambda x: "on_boundary" and near(x[0],0))
boundaries = FacetFunction("size_t",mesh_file)
Inlet.mark(boundaries, 3)
```

All the mappings are made with python functions, this is done so we can call the mappings later in the variational form.

```
def F_(d):
        return (Identity(len(d)) + grad(d))

def J_(d):
        return det(F_(d))
```

The stress tensors are also defined as functions to be used in the variational form.

```
def E(d):
        return 0.5*(F_(d).T*F_(d) - Identity(len(d)))

def S(d,lamda_s,mu_s):
    I = Identity(len(d))
    return 2*mu_s*E(d) + lamda_s*tr(E(d))*I

def Piola1(d,lamda_s,mu_s):
        return F_(d)*S(d,lamda_s,mu_s)

def sigma_f_u(u,d,mu_f):
    return  mu_f*(grad(u)*inv(F_(d)) + inv(F_(d)).T*grad(u).T)

def sigma_f_p(p, u):
    return -p*Identity(len(u))
```

4.2 Variational form

The variational form can be written directly into FEniCS. We write all the forms and add them together to make one big form to be calculated in the upcoming timeloop. We start by looking at the fluid variational form

```
J_theta = theta*J_(d_["n"]) + (1 - theta)*J_(d_["n-1"])

F_fluid_linear = rho_f/k*inner(J_theta*(v_["n"] - v_["n-1"]), psi)*dx_f

F_fluid_nonlinear =  Constant(theta)*rho_f*\
inner(J_(d_["n"])*grad(v_["n"])*inv(F_(d_["n"]))*v_["n"], psi)*dx_f

F_fluid_nonlinear += inner(J_(d_["n"])*sigma_f_p(p_["n"], d_["n"])*\
```

```
inv(F_(d_["n"])).T, grad(psi))*dx_f

F_fluid_nonlinear += Constant(theta)*inner(J_(d_["n"])\
*sigma_f_u(v_["n"], d_["n"], mu_f)*inv(F_(d_["n"])).T, grad(psi))*dx_f

F_fluid_nonlinear += Constant(1 - theta)*inner(J_(d_["n-1"])*\
sigma_f_u(v_["n-1"], d_["n-1"], mu_f)*inv(F_(d_["n-1"])).T, grad(psi))*
    dx_f

F_fluid_nonlinear += \
inner(div(J_(d_["n"])*inv(F_(d_["n"]))*v_["n"]), gamma)*dx_f

F_fluid_nonlinear += Constant(1 - theta)*rho_f*\
inner(J_(d_["n-1"])*grad(v_["n-1"])*inv(F_(d_["n-1"]))*v_["n-1"], psi)*
    dx_f

F_fluid_nonlinear -= rho_f*inner(J_(d_["n"])*\
grad(v_["n"])*inv(F_(d_["n"]))*((d_["n"]-d_["n-1"])/k), psi)*dx_f
```

where dx_f is the fluid domain. The theta value choses the scheme we want. inv() gives the inverse of the matrix.

Next is the solid variational form:

```
delta = 1E10
F_solid_linear = rho_s/k*inner(v_["n"] - v_["n-1"], psi)*dx_s +\
delta*(1/k)*inner(d_["n"] - d_["n-1"], phi)*dx_s -\
delta*inner(Constant(theta)*v_["n"] + Constant(1-theta)*v_["n-1"], phi)*
    dx_s

F_solid_nonlinear = inner(Piola1(Constant(theta)*d_["n"] +\
Constant(1 - theta)*d_["n-1"], lamda_s, mu_s), grad(psi))*dx_s
```

The weighed delta is talked about in section 3.6

4.3 NewtonSolver

To solve a non-linear problem we need make a newton solver, taken from [Mikael kompendium]. F is derivated wrt to dvp and is assembled to a matrix. -F is assembled as b and we solve until the residual is smaller than a give

tolerance. There is also an if test which only assembles the Jacobian the first and tenth time. This reuses the Jacobian to improve speed, as we shall see later. Lastly the the mpi line is when the code is running in paralel that we only print out the values for one of the computational nodes.

```
def newtonsolver(F, J_nonlinear, A_pre, A, b, bcs, \
                 dvp_, up_sol, dvp_res, rtol, atol, max_it, T, t, **
    monolithic):
    Iter      = 0
    residual   = 1
    rel_res    = residual
    lmbda = 1

    while rel_res > rtol and residual > atol and Iter < max_it:
        if Iter % 10 == 0:
            A = assemble(J_nonlinear, tensor=A, form_compiler_parameters
     = {"quadrature_degree": 4})
            A.axpy(1.0, A_pre, True)
            A.ident_zeros()

        b = assemble(-F, tensor=b)

        [bc.apply(A, b, dvp_["n"].vector()) for bc in bcs]
        up_sol.solve(A, dvp_res.vector(), b)
        dvp_["n"].vector().axpy(lmbda, dvp_res.vector())
        [bc.apply(dvp_["n"].vector()) for bc in bcs]
        rel_res = norm(dvp_res, 'l2')
        residual = b.norm('l2')
        if isnan(rel_res) or isnan(residual):
            print "type rel_res: ",type(rel_res)
            t = T*T

        if MPI.rank(mpi_comm_world()) == 0:
            print "Newton iteration %d: r (atol) = %.3e (tol = %.3e), r
    (rel) = %.3e (tol = %.3e) " \
        % (Iter, residual, atol, rel_res, rtol)
        Iter += 1

    return dict(t=t)
```

4.4 Timeloop

In the time loop we call on the solver and update the functions v, d, p for each round. The counter value is used when we only want to take out values every certain number of time iterations.

```
1
2  while t <= T:
3      print "Time t = %.5f" % t
4      time_list.append(t)
5      if t < 2:
6          inlet.t = t;
7      if t >= 2:
8          inlet.t = 2;
9
10     frame#frame#frameResetframe framecounters
11     atol = 1e-6;rtol = 1e-6; max_it = 100; lmbda = 1.0;
12
13     dvp = Newton_manual(F, dvp, bcs, atol, rtol, max_it, lmbda,dvp_res,
       VVQ)
14
15
16     times = ["n-2", "n-1", "n"]
17     for i, t_tmp in enumerate(times[:-1]):
18         dvp_[t_tmp].vector().zero()
19         dvp_[t_tmp].vector().axpy(1, dvp_[times[i+1]].vector())
20
21     t += dt
22     counter +=1
```

Chapter 5

Verification and Validation in Computational Fluid Structure Interaction.

When we set out to solve a real world problem with numerical computing, we start by defining the mathematics, we implement the equations numerically and solve the equations on a computer. We use the solutions to extract data that will answer the questions we set out to answer. A problem then immediately arises, is the solution correct? To answer this we need to answer another question, are the equations solved correct numerically, if so is the problem defined correct mathematically, that is in accordance with the governing laws and equations? Without answering these questions, being confident that your solutions are correct is difficult [16]. The goal of this section will hence be to verify and validate the different numerical schemes outlined in the last chapter.

I start with the process of Verification where the fluid and structure parts of the code will be verified. Following will be Validation of the code where I look at a well known benchmark testing the fluid and structure parts individually and as a full FSI problem. After that I investigate the impact of using different time order schemes, and different mesh motion techniques.

5.1 Verification

Verification is the process of determining wether or not the implementation of numerical algorithms in computer code, is done correctly. [11]. In verification we get evidence that the numerical model derived from mathematics is solved correctly by the computer. The strategy will be to identify, quantify and reduce errors cause by mapping a mathematical model to a computational model. Verification does not address wether or not the computed solutions are in alignment with the real world. It only tells us that our model is computed correctly or not. To verify that we are computing correctly we can compare our computed solution to an exact solution. But the problem is that there are no known exact solution to for instance the Navier-Stokes equations, other than for very simplified problems.

In Verification there are multiple classes of test that can be performed, and the most rigorous is the *Method of manufactured solution*(MMS) [11]. Rather than looking for an exact solution we manufacture one. The idea is to make a solution *a priori*, and use this solution to generate an analytical source term for the governing PDEs and than run the PDE with the source term to get a solution hopefully matching the manufactured one. The manufactured solution does not need to have a physically realistic relation, since the solution deals only with the mathematics. The procedure is as follows [11]:

- We define a mathematical model on the form $L(u) = 0$ where $L(u)$ is a differential operator and u is a dependent variable.
- Define the analytical form of the manufactured solution $\hat{u}$
- Use the model $L(u)$ with $\hat{u}$ inserted to obtain an analytical source term $f = L(\hat{u})$
- Initial and boundary conditions are enforced from $\hat{u}$
- Then use this source term to calculate the solution u, $L(u) = f$

After the solution has been computed we perform systematic convergence tests [**?**]. The idea behind order of convergence tests is based on the behavior of the error between the manufactured exact solution and the computed solution. If we let u be the numerical solution and u_e be the exact solution, $||.||$ be the L^2 norm, we define the error as:

$$E = ||u - u_e|| \tag{5.1}$$

When we increase the number of spatial points ($\Delta x, \Delta y$ or Δz) or decrease timestep(Δt), we expect the error to get smaller. It is the rate of this error that lets us know wether the solution is converging correctly. If we assume that the number of spatial points are equal in all directions the error is expressed as

$$E = C_1 \Delta x^k + C_2 \Delta t^l \tag{5.2}$$

where $k = m+1$ and m is the polynomial degree of the spatial elements. The error is hence dependent on the number of spatial points and the timestep. If we for instance reduce Δt significantly, Δx will dominate, and Δt will be negligible. If we then look at two error where E_{n+1} has finer mesh than E_n, using (5.2):

$$\frac{E_{n+1}}{E_n} = \big(\frac{\Delta x_{n+1}}{\Delta x_n}\big)^k \tag{5.3}$$

$$k = \frac{log(\frac{E_{n+1}}{E_n})}{log(\frac{\Delta x_{n+1}}{\Delta x_n})} \tag{5.4}$$

We use k to find the observed order of convergence and match with the theoretical order of convergence for each given problem.

The manufactured solutions should be chosen to be non-trivial and analytic [14] [11]. The solutions should be so that no derivatives vanish. For this reason trigonometric and exponential functions can be a smart choice, since they are smooth and infinitely differentiable. In short a good manufactured solution is one that is complex enough so that it rigorously tests each part of the equations.

I begin with verifying the solid part of the code. Then the fluid part of the code will be tested with a given displacement, also testing the mappings between configurations. A full MMS of the entire FSI problem with all the conditions is very difficult [2]. One needs to take into account the condition of continuity of velocity on the interface [3]. The stresses need to equal on the interface and the flow needs to be divergence free. MMS of full FSI is therefore out the scope of this thesis.

5.1.1 MMS of the Solid equation

The first MMS is with the solid problem alone. Here I test the solid equation with the condition $u = \frac{\partial d}{\partial t}$.

MMS of the solid equation is done using the manufactured solutions d_e and u_e:

$$\begin{aligned} d_e =& (cos(y)sin(t), cos(x)sin(t)) \\ u_e =& (cos(y)cos(t), cos(x)cos(t)) \end{aligned}$$

To meet the requirements of MMS such as smoothness and complexity, i chose functions with sine and cosine. The derivatives does not become zero and we have time and space dependencies.

These solutions are used to make a sourceterm f_s:

$$\rho_s \frac{\partial u_e}{\partial t} - \nabla \cdot (P(d_e)) = f_s$$

The solid variational formulation is written as:

$$\left(\rho_s \frac{\partial u}{\partial t}, \phi\right)_{\hat{\mathcal{S}}} + \left(P, \nabla\phi\right)_{\hat{\mathcal{S}}} = f_s \tag{5.5}$$

$$\left(u - \frac{\partial d}{\partial t}, \psi\right)_{\hat{\mathcal{S}}} = 0 \tag{5.6}$$

These equations are solved together and we solve for d and u on a unit square domain. The number N denotes the number of spatial points in x and y direction. The functions u and d will be computed to match the source term.

Even though we have two equations we do not make a source for the second. This is because the solutions are made to uphold criteria of $u = \frac{\partial d}{\partial t}$.

In the tables below we investigate convergence in space and time. I start with checking order of convergence in space. Setting m = 1, the expected order of convergence will be 2.

N	Δt	**m**	E_u	$\mathbf{k_u}$	E_d	$\mathbf{k_d}$
4	1×10^{-7}	**1**	0.0068828	**-**	3.7855×10^{-9}	**-**
8	1×10^{-7}	**1**	0.0017204	**2.0002577**	9.4622×10^{-10}	**2.0002577**
16	1×10^{-7}	**1**	0.0004300	**2.0000622**	2.3654×10^{-10}	**2.0000622**
32	1×10^{-7}	**1**	0.0001075	**2.0000154**	5.9136×10^{-11}	**2.0000154**
64	1×10^{-7}	**1**	0.0000268	**2.0000038**	1.4783×10^{-11}	**2.0000038**

Table 5.1: Structure Method of Manufactured of Solution in space in m = 1

Lastly I check convergence in time. Here i set N = 64 and the timestep is halved for each computation.

N	$\mathbf{\Delta t}$	$E_u[\times 10^{-6}]$	$\mathbf{k_u}$	$E_u[\times 10^{-8}]$	$\mathbf{k_d}$
64	**0.1**	0.027663		0.0034221	
64	**0.05**	0.013390	**1.0467**	0.0018093	**0.9194**
64	**0.025**	0.007016	**0.9324**	0.0009246	**0.9685**
64	**0.0125**	0.003645	**0.9444**	0.0004688	**0.9798**
64	**0.00625**	0.001828	**0.9957**	0.0002414	**0.9571**

Table 5.2: Structure Method of Manufactured of Solution in time

5.1.2 MMS of Fluid equations with prescribed motion

I start by prescribing a motion to d and w and manufacturing solutions u_e and p_e. We set $u_e = w$ to start with:

$$\begin{aligned} d =& (cos(y)sin(t), cos(x)sin(t)) \\ u_e = w =& (cos(y)cos(t), cos(x)cos(t)) \\ p_e =& cos(x)cos(t) \end{aligned}$$

I make the solutions to uphold the criteria :

$$\nabla \cdot u = 0, \ \frac{\partial d}{\partial t} = w$$

I also want to test the mappings so we make the source term f_f without mappings:

$$\rho_f \frac{\partial u_e}{\partial t} + \nabla u_e (u_e - \frac{\partial d}{\partial t}) - \nabla \cdot \sigma(u_e, p_e)_f = f_f$$

Then we use f_f and map it to the reference configuration and compute:

$$\rho_f J \frac{\partial u}{\partial t} + (\nabla u) F^{-1}(u - \frac{\partial d}{\partial t}) + \nabla \cdot (J \hat{\sigma_f} F^{-T}) = J f_f$$

This is in fact the fluid equation in the ALE framework mapped ref.

The computations are done on a unitsquare domain and the computations ran with 10 timesteps and the error was calculated for each time step and then the mean of all the errors was taken and used to calculate the convergence rates.

N	Δt	**m**	E_u	k_u	E_p	k_p
64	0.1	**2**	5.1548×10^{-5}	**-**	0.008724	**-**
64	0.05	**2**	2.5369×10^{-5}	**1.0228**	0.004290	**1.0240**
64	0.025	**2**	1.2200×10^{-5}	**1.0561**	0.002058	**1.0596**
64	0.0125	**2**	0.56344×10^{-5}	**1.1145**	0.0009556	**1.1068**

Table 5.3: MMS ALE FSI u=w

N	Δt	**m**	E_u	k_u	E_p	k_p
2	1×10^{-6}	**2**	8.6955×10^{-4}	**-**	0.01943	**-**
4	1×10^{-6}	**2**	1.0844×10^{-4}	**3.0032**	0.00481	**2.0140**
8	1×10^{-6}	**2**	0.1354×10^{-4}	**3.0007**	0.00119	**2.0120**
16	1×10^{-6}	**2**	0.0169×10^{-4}	**3.0001**	0.00029	**2.0074**

Table 5.4: MMS ALE FSI u=w

5.2 Validation

After the code has been verified to see that we are indeed computing in the right fashion. We move on to Validation which is the process of determining if the model gives an accurate representation of the real world within the bounds of the intended use [16]. A model is made for a specific purpose, its only valid in respect to that purpose [10]. If the purpose is complex and trying to answer multiple question then the validity need to be determined to each question. The idea is to validate the solver *brick by brick*. We start with simple testing of each part of the model and build more complexity and eventually test the whole model.

Three issues have been identified in this process [16]: Quantifying the accuracy of the model by comparing responses with experimental responses, interpolation of the model to conditions corresponding to the intended use and determining the accuracy of the model for the conditions under which its meant to be used. Well known benchmarks will be used as validation, we will see in this chapter that these tests supply us with a problem setup, initial and boundary conditions, and lastly results that we can compare with.

The process of Validation is also, as I have experienced, a way to figure out at what size timestep and number of spatial points the model can handle to run. As we will see in the chapter all the benchmarks are run with different timesteps and number of cells to see how it reacts. The problem with using benchmarks with known data for comparison is that we do not test the model blindly. It is easier to mold the model to the data we already have. As Oberkampf and Trucano in [16] puts it “Knowing the “correct" answer beforehand is extremely seductive, even to a saint.”. Knowing the limitations of our tests will therefore strengthen our confidence in the model. It really can be an endless process of verifying and validating if one does not clearly now the bounds of sufficient accuracy.

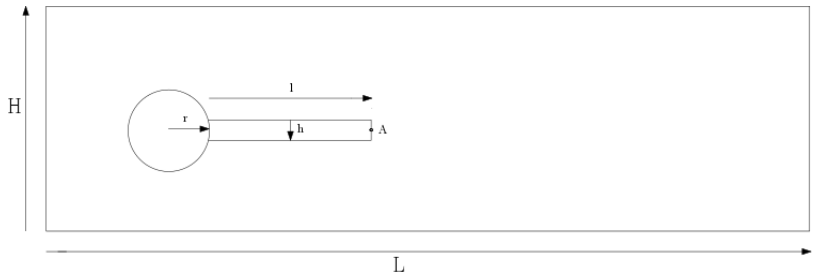
[16]
In the following we will look at tests for the fluid solvers both alone, testing laminar to turbulent flow, and with solid. We will test the solid solver, and lastly the entire coupled FSI problem.

5.2.1 Fluid-Structure Interaction between an elastic object and laminar incompressible flow

The goal of this benchmark is to test the fluid and solid solver first separately and then together as a full FSI problem [8]. This benchmark is based on the older benchmark " flow around cylinder" with fluid considered incompressible and in the laminar regime where the structure deformations are significant. The problem is setup with the solid submerged in the fluid, so that oscillations in the fluid deform the structure. We will measure the drag and lift around the circle and bar, and measure structural displacement at a given point. This benchmark will be used to test and verify different numerical methods and code implementations. Testing robustness and efficiency.

Problem Defintion

Domain



The computational domain consists of a circle with an elastic bar behind the circle. The circle is positioned at (0.2, 0.2) making it 0.05 of center from bottom to top, this is done to induce oscillations to an otherwise laminar flow. This gives a force to the elastic bar. The parameters of the domain are: L = 2.5, H = 0.41, l = 0.35, h = 0.02, A = (0.2, 0.6)

Boundary conditions

The fluid velocity has a parabolic profile on the inlet that changes over time:

$$u(0,y) = 1.5u_0 \frac{y(H-y)}{(\frac{H}{2})^2}$$
$$u(0,y,t) = u(0,y)\frac{1-cos(\frac{\pi}{2}t)}{2} \text{ for } t < 2.0$$
$$u(0,y,t) = u(0,y) \text{ for } t \leq 2.0$$

We set no slip on the "floor" and "ceiling" so to speak.

$$u(x,y,t) = 0 \text{ on } \partial\mathcal{F}_{floorandceiling}$$
$$p(x,y,t) = 0 \text{ on } \partial\mathcal{F}_{outlet}$$

Quantities for comparison

When the fluid moves around the circle and bar it exerts a force. These are split into drag and lift and calculated as follows:

$$(F_d, F_L) = \int_S \sigma_f n dS$$

where S is the part of the circle and bar in contact with the fluid.
We set a point A on the right side of the bar. This point is used to track the deformation in CSM and FSI tests.
In the unsteady time dependent problems the values are represented with a mean , amplitude and frequency:

$$mean = \frac{1}{2}(max + min) \tag{5.7}$$
$$amplitude = \frac{1}{2}(max - min) \tag{5.8}$$
$$frequency = \frac{1}{T} \tag{5.9}$$

In each test the numbers with ref are the values taken from the benchmark paper [8]

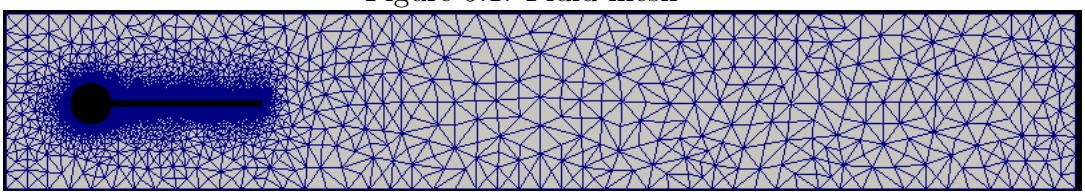
Results

CFD test

The first two CFD tests are run with Reynolds number 20 and 100 giving steady drag and lift around the circle. CFD 3 has a Reynolds number 200

which will induce oscillations behind the circle, giving fluctuations in the drag and lift. The CFD tests were run using the the bar as rigid object, that is the domain calculated is just the fluid domain. It is possible to also calculate with the bar and setting ρ_s and μ_s to a large value.

Figure 5.1: Fluid mesh



Parameters	CFD1	CFD2	CFD3
$\rho_f[10^3 \frac{kg}{m^3}]$	1	1	1
$\nu_f[10^{-3} \frac{m^2}{s}]$	1	1	1
$U[\frac{m}{s}]$	0.2	1	2
$\mathrm{Re} = \frac{Ud}{\nu_f}$	20	100	200

Table 5.5: CFD parameters

elements	**dofs**	**Drag**	**Lift**
6616	32472	14.2439	1.0869
26464	124488	14.2646	1.11085
105856	487152	14.2755	1.11795
ref		**14.29**	**1.119**

Table 5.6: CFD 1

elements	**dofs**	**Drag**	**Lift**
6616	32472	135.465	6.27158
26464	124488	136.566	9.82166
105856	487152	136.573	10.4441
ref		**136.7**	**10.53**

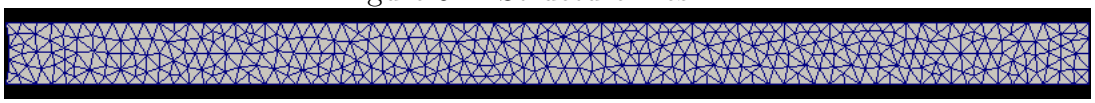
Table 5.7: CFD 2

CSM test

The CSM test are calculated using only the bar and adding a gravity term g with the same value but changing the parameters of solid. The tests CSM1 and CSM2 are steady state solutions. The difference is a more slender bar. The CSM 3 test is unsteady and even more slender causing the bar to move up and down. Since there is no resistance from any fluid the unsteady test should if energy is preserved make the bar move up and down infinitely.

Our quantity for comparing there will be the deformation at the point A.

Figure 5.2: Structure mesh



Parameters	CSM1	CSM2	CSM3
$\rho_f[10^3 \frac{kg}{m^3}]$	1	1	1
$\nu_f[10^{-3} \frac{m^2}{s}]$	1	1	1
u_0	0	0	0
$\rho_s[10^3 \frac{kg}{m^3}]$	1	1	1
ν_s	0.4	0.4	0.4
$\mu_s[10^6 \frac{m^2}{s}]$	0.5	2.0	0.5
g	2	2	2

Table 5.8: Parameters

elements	dofs	$d_x(A)[\times 10^{-3}]$	$d_y(A)[\times 10^{-3}]$
725	1756	-5.809	-59.47
2900	6408	-6.779	-64.21
11600	24412	-7.085	-65.63
46400	95220	-7.116	-65.74
ref	**ref**	**-7.187**	**-66.10**

Table 5.9: CSM 1

Elements	Dofs	$d_x(A)[\times 10^{-3}]$	$d_y(A)[\times 10^{-3}]$
725	1756	-0.375	-15.19
2900	6408	-0.441	-16.46
11600	24412	-0.462	-16.84
46400	95220	-0.464	-16.87
ref	**ref**	**-0.469**	**-16.97**

Table 5.10: CSM 2

elements	dofs	$d_x(A)[\times 10^{-3}]$	$d_y(A)[\times 10^{-3}]$
725	1756	-11.743 ± 11.744	-57.952 ± 58.940
2900	6408	-13.558 ± 13.559	-61.968 ± 63.440
11600	24412	-14.128 ± 14.127	-63.216 ± 64.744
46400	95220	-14.182 ± 14.181	-63.305 ± 64.843
ref		$\mathbf{-14.305 \pm 14.305}$	$\mathbf{-63.607 \pm 65.160}$

Table 5.11: CSM 3

Figure 5.3: Displacement of point A, CSM3

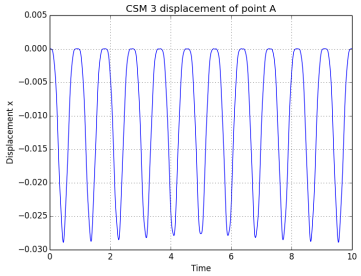


Figure 5.4: Displacement x

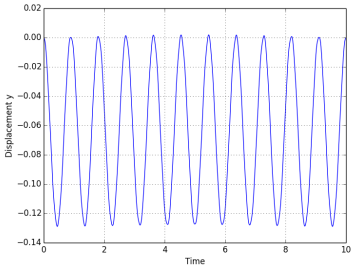


Figure 5.5: Discplacement y

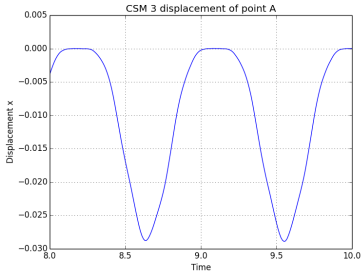


Figure 5.6: Displacement x

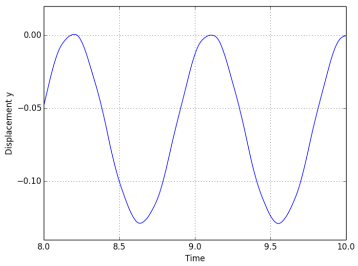


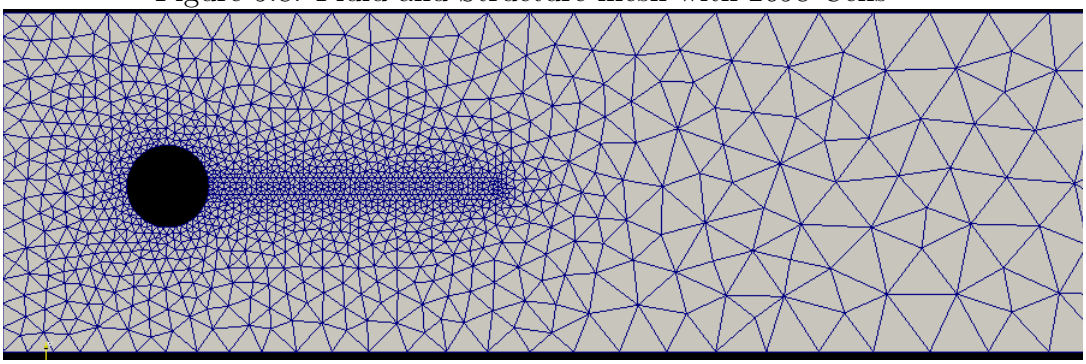
Figure 5.7: Discplacement y

The plots 5.2.1 is a plot of the CSM3 test. This was run with Crank-Nicholson, $\theta = 0.5$ and as we can see the energy has been preserved.

FSI test

Lastly we run the full FSI problem. Here we can see in 5.2.2 that now both fluid and structure has a mesh. The tests are run with 2 different inflows. FSI1 gives a steady state solution while the others are unsteady. FSI-2 gives the largest deformation is therefore considered the most difficult of the three [12]. The FSI-3 test has the highest inflow speed giving more rapid oscillations.

Figure 5.8: Fluid and Structure mesh with 2698 Cells



Parameters	FSI1	FSI2	FSI3
$\rho_f[10^3 \frac{kg}{m^3}]$	1	1	1
$\nu_f[10^{-3} \frac{m^2}{s}]$	1	1	1
u_0	0.2	1	2
$\mathrm{Re} = \frac{Ud}{\nu_f}$	20	100	200
$\rho_s[10^3 \frac{kg}{m^3}]$	1	10	1
ν_s	0.4	0.4	0.4
$\mu_s[10^6 \frac{m^2}{s}]$	0.5	0.5	2

Table 5.12: FSI Parameters

FSI1

Cells	Dofs	$d_x(A)[x10^{-3}]$	$d_y(A)[x10^{-3}]$	Drag	Lift	Spaces
2698	23563	0.0227418	0.799314	14.1735	0.761849	P2-P2-P1
10792	92992	0.0227592	0.80795	14.1853	0.775063	P2-P2-P1
43168	369448	00.227566	0.813184	14.2269	0.771071	P2-P2-P1
ref	**ref**	**0.0227**	**0.8209**	**14.295**	**0.7638**	**ref**

Table 5.13: FSI 1

FSI2

The FSI-2 results were with $\Delta t = 0.01, \theta = 0.51$, biharmonic bc 1, $\alpha_u = 0.01$

Cells	Dofs	$d_x(A)[x10^{-3}]$	$d_y(A)[x10^{-3}]$	Drag	Lift	Extrapolation
2698	23563	-15.17 ± 13.35	1.13 ± 82.5	160.12 ± 17.88	0.87 ± 259.62	Biharmonic
2698	23563	-14.92 ± 13.17	1.13 ± 81.86	160.28 ± 17.94	0.65 ± 254.07	Biharmonic
2698	23563	-15.10 ± 13.32	1.16 ± 82.46	$159.53\pm, 17.44$	0.68 ± 259.10	Harmonic
10792	92992	-14.85 ± 13.14	1.21 ± 81.72	160 ± 17.84	0.89 ± 255.10	Harmonic
ref	**ref**	$\mathbf{-14.58 \pm 12.44}$	$\mathbf{1.23 \pm 80.6}$	$\mathbf{208.83 \pm 73.75}$	$\mathbf{0.88 \pm 234.2}$	**ref**

Table 5.14: FSI-2 with $\Delta t = 0.01$

Cells	Dofs	$d_x(A)[x10^{-3}]$	$d_y(A)[x10^{-3}]$	Drag	Lift	Extrapolation
2698	23563	15.42 ± 13.10	1.14 ± 83.39	157.01 ± 15.69	-0.77 ± 174.36	Harmonic
ref	**ref**	$\mathbf{-14.58 \pm 12.44}$	$\mathbf{1.23 \pm 80.6}$	$\mathbf{208.83 \pm 73.75}$	$\mathbf{0.88 \pm 234.2}$	**ref**

Table 5.15: FSI-2 with $\Delta t = 0.001$

Figure 5.9: Deformation at $t = 9.20$sec using warp by vector in paraview

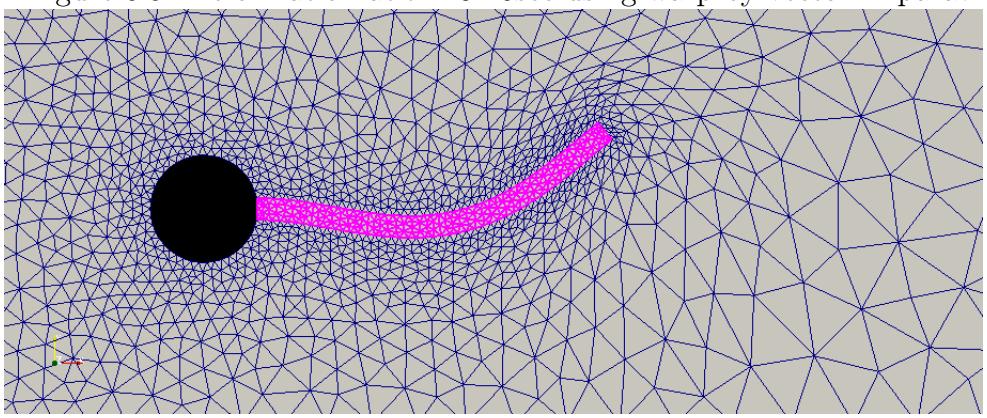
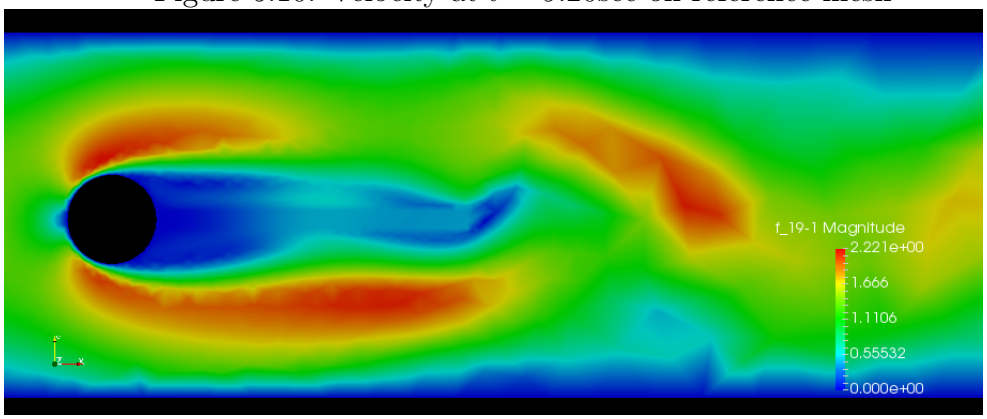


Figure 5.10: Velocity at $t = 9.20$sec on reference mesh



5.2.2 Mesh motion techniques

In this section we compare different mesh motion techniques from 3.3. The test will be run using a version of the CSM test discussed earlier. The tests will compare the different techniques by looking at the how the deformation is lifted into the fluid domain. This is done by looking at a plot of the mesh after deformation to see how much cells distort and why. This is done using Paraview.
In these test cases we have the fluid initially at rest and with no inflow on the fluid. A gravitational force is applied to the structure much like the previous CSM test. The only difference is that we now use the full domain from the 5.2.1 . The tests are run as time-dependent with a the backward Euler scheme, leading to a steady state solution. In the first test case the parameters from CSM1 are used, and in the CSM4 the gravitational force has just been increased from 2 to 4.

Boundary conditions

The upper, lower and left boundary is set as no slip, that is no velocity in the fluid. On the left boundary there is a do nothing, and zero pressure.

Quantities for comparison

The different techniques will be plotted with the minimal value of the Jacobian. The Jacobian is if we remember the determinant of the deformation rate. If the jacobian is zero anywhere in the domain it means that the cells overlap and can cause singularity in the matrices during assembly.
We will also look at the a plot of the deformation in the domain. To visualize the how the different mesh motion techniques work. It is possible to see that if get thin triangles in the computational domain then the mesh motion operator is no good.

CSM1

Figure 5.11: CSM1

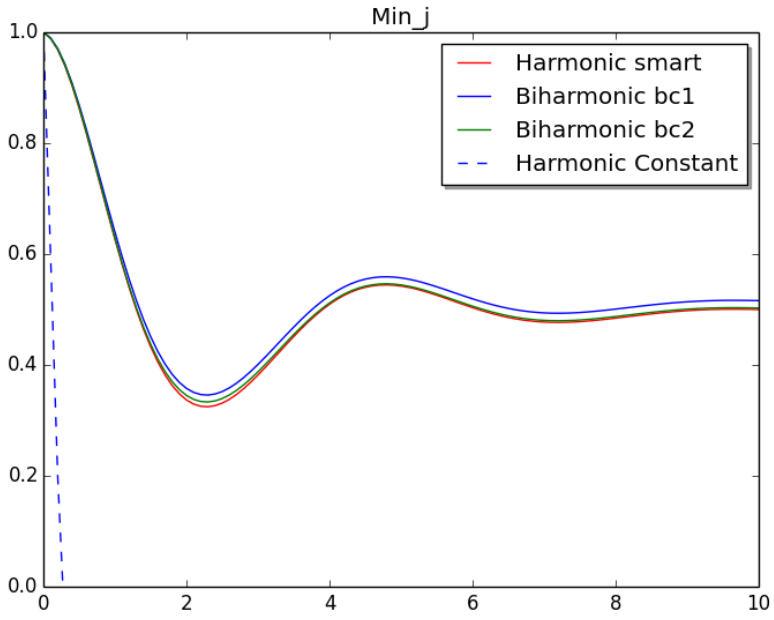


Figure 5.12: CSM1 with different techniques

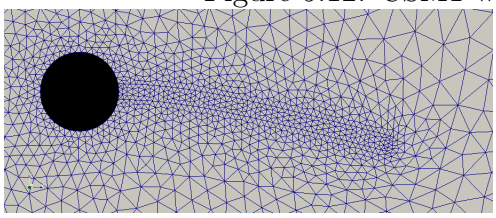


Figure 5.13: Harmonic smart

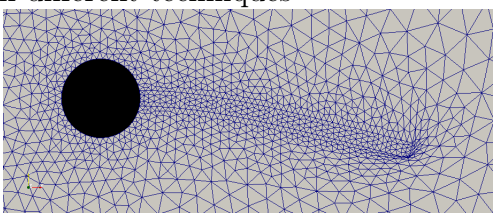


Figure 5.14: Harmonic constant

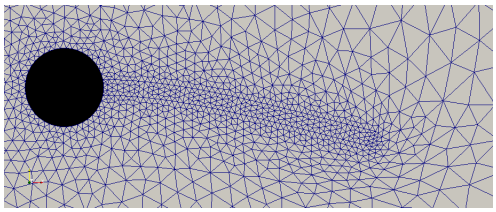


Figure 5.15: Biharmonic bc1

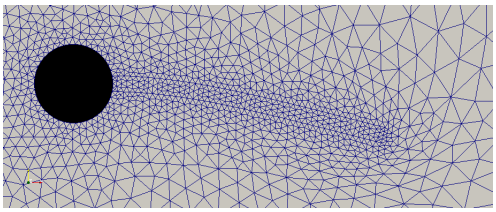


Figure 5.16: Biharmonic bc2

Technique	$d_y(A)[\times 10^{-3}]$	$d_x(A)[\times 10^{-3}]$
Harmonic	65.406	7.036
Constant	43.033	2.999
Bibc1	65.404	7.036
Bibc2	65.405	7.036

Table 5.16: Displacement of CSM1 test

5.2.3 Temporal stability

The following section will be looking at the results from choosing different θ values in our scheme. The benchmark test FSI-2 has been used to since it is known to blow up with certain values of θ and Δt. We only study the effects of Drag as the three other quantities shows the same behavior.

5.2.3 show the plots of Drag with $\Delta t = 0.01$. This will not produce very good drag values compared to the benchmark, but it shows the instability when choosing $\theta = 0.5$. The Crank-Nicholson scheme is stable until about 13 seconds where we see that it blows up and the solver diverges.

Figure 5.17: Drag for FSI2 with $\Delta t = 0.01$

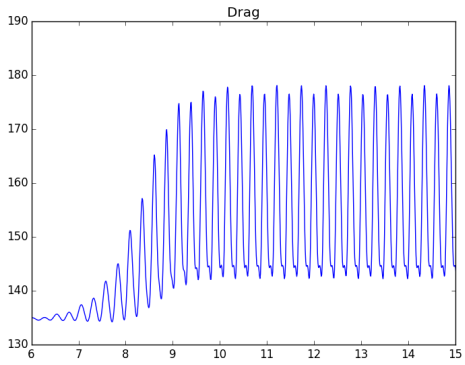


Figure 5.18: $\theta = 0.51$

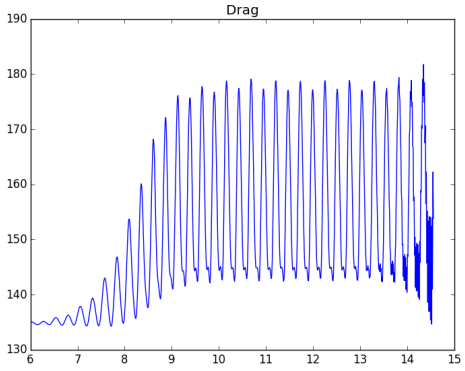


Figure 5.19: $\theta = 0.50$

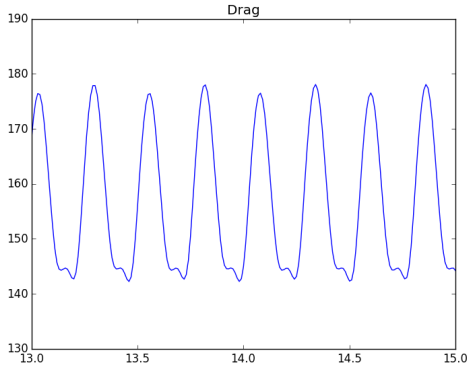


Figure 5.20: $\theta = 0.51$

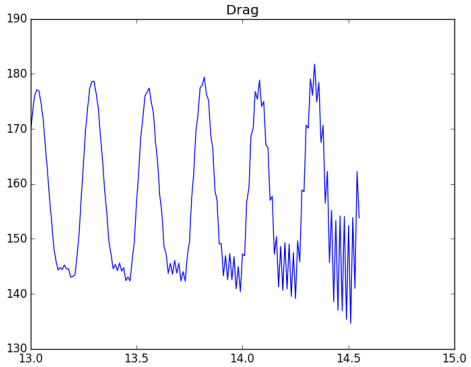


Figure 5.21: $\theta = 0.50$

Put in plot of $\theta = 0.5$ and $\Delta t = 0.001$ to see if its stable longer?

Chapter 6

Runtime improvements

Modeling Fluid Structure Interaction accurately has been difficult not only because of the lack of stable schemes. It is also difficult because of the increase in runtime when modeling both fluid and structure, especially with monolithic schemes. When accuracy is needed in CFSI the number of cells can quickly become large. The strain on computers increases giving increased runtime.

To solve non-linear FSI problems I use in this these a Newton solver. In the Newton solver we have to assemble the Jacobian of the matrix, assemble the residual and solve for each iteration, until convergence is met. Working on this thesis the issue of runtime was certainly a problem, taking days and weeks for a simulation to finish.

I have in this chapter added the runtime improvements I have made to my newton solver. The point of this chapter is not to rigorously experiment with time improvement techniques. But merely to show what helped me make the simulation time bearable. And hopefully help others when employing a newton solver. The ideas covered are not new and has been implemented in Gabriel Balaban?s master thesis in 2012 [15].

6.1 Newton runtime profile

In table 6.1 I ran the FSI1 problem with no optimizations, checking the amount of time spent on: Jacobian Assembly, residual assembly and time to solve.

Method	**Runtime [s]**	Runtime [%]	**Calls**
Assembly of Jacobian	**60.7**	94.4	**5**
Assembly or residual	**0.6**	0.9	**5**
Solve	**2.8**	4.4	**5**
Fulltime	**64.3**	100%	-

Table 6.1: Newton solver timed with no optimizations run with $\mathbf{\Delta t = 0.5}$

As we can see in table 6.1 the majority of time spent is in the assembly of the Jacobian. Therefore energy will be spent on reducing the time spent on assembling the jacobian. In the next sections I will introduce two ways of improving time spent on assembling greatly. The first is reusing of the Jacobian and the second is employing a function in FENiCS called quadrature reduce. I will compare the techniques time improvement to table 6.1.

6.2 Jacobian reuse

The majority of the time spent on in the newton solver is assembling the Jacobian. In most cases we employ a small $\delta t = 10^-2, 10^-3$, which in turn means that the Jacobian only differs by a small amount. The trick therefore is to reuse the Jacobian. This is done by telling the solver that we only assemble the Jacobian for a given number of iterations. The rest of the newton solver stays the same and keeps iterating until convergence, but the Jacobian matrix stays the same for a set number of iterations. When employed it was noticed that we needed a larger number of iterations to reach convergence, but the overall time of the Newton solver was much faster.

Method	**Runtime [s]**	Runtime [%]	**Calls**
Assembly of Jacobian	**11.5 (-80%)**	68.7	**1 (-20%)**
Assembly or residual	**0.9 (+50%)**	5.8	**9 (+46%)**
Solve	**4.2 (+50%)**	25.0	**9 (+46%)**
Fulltime	**16.8 (-74 %)**	-	-

Table 6.2: Newton solver timed with reuse of the jacobian run with $\mathbf{\Delta t = 0.5}$

In table 6.2 the same FSI1 test is done with $\Delta t = 0.5$. Even with a timestep that is fairly large, we get great improvements in runtime (-74%). The Jacobian assembles once meaning that at this timestep the Jacobian was cal-

culated once and used again 9 times. What happens when we reuse the Jacobian, we have to iterate more times (9 times in this case) and as we can see the runtime in assembling the residual has increased by 50 %. This is a much less costly process, so even when iterating more times we get a reduce in runtime.

6.3 Quadrature reduce

Assembling of the Jacobian matrix with non-linear functions, requires a high number of quadrature points. When the accuracy of the Jacobian can be reduced, we can reduce the number of quadrature degree. This improves the runtime but reduces the accuracy leading to more iterations per time step. Reducing the quadrature degree can lead to blow up of the system in some cases. But is of great help in many other cases.

The FSI1 test is run with $\Delta t = 0.5$ like the section above.

Method	**Runtime [s]**	Runtime [%]	**Calls**
Assembly of Jacobian	**4.9 (-92%)**	60.3	**5 (0%)**
Assembly or residual	**0.5 (-17%)**	6.9	**5 (0%)**
Solve	**2.6 (-7%)**	31.9	**5 (0%)**
Fulltime	**8.2 (-87%)**	-	-

Table 6.3: Newton solver timed with quadrature reduxe run with $\mathbf{\Delta t = 0.5}$

Reducing the quadrature degree as we can see gives a 92 % decrease in time spent assembling the Jacobian even with the same number of calls. The full time spent went down by 87 %.

6.4 Summary of runtime improvement techniques

Method	**Runtime [s]**	Runtime [%]	**Calls**
Assembly of Jacobian	**1.2 (-98%)**	18.1	**1 (-20%)**
Assembly or residual	**0.9 (+50%)**	14.7	**9 (+46%)**
Solve	**4.4 (+57%)**	66.2	**9 (+46%)**
Fulltime	**6.6 (-89%)**	-	-

Table 6.4: Newton solver timed with jacobian reuse and quadrature reduce run with $\mathbf{\Delta t = 0.5}$

Finally I used both the reuse of Jacobian and the reduction of the quadrature. As we can see in table 6.4 the total runtime went down 89%. In my work on this thesis I used these two techniques as often as i could. However I found that in the case of FSI3 neither reuse of the Jacobian or the reduction of quadrature degree gave convergence in the solutions. Both techniques worked with great success in the FSI2 case. So the reason for it not working in FSI3 might be because of the higher reynolds number leading to the need for a more accurate assembly of the Jacobian.

Chapter 7

Conclusions and further work

7.1 Partitioned

Bibliography

[1] K. Yusuf Billah. Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks. *American Journal of Physics*, 59(2):118, 1991.

[2] S. Étienne, A. Garon, and D. Pelletier. Some manufactured solutions for verification of fluid-structure interaction codes. *Computers and Structures*, 106-107:56–67, 2012.

[3] Stéphane Étienne, D Tremblay, and Dominique Pelletier. Code Verification and the Method of Manufactured Solutions for Fluid-Structure Interaction Problems. *36th AIAA Fluid Dynamics Conference and Exhibit*, (June):1–11, 2006.

[4] Charles L. Fefferman. Existence and smoothness of the Navier-Stokes equation. *The millennium prize problems*, (1):1–5, 2000.

[5] Miguel A. Fernández, Jimmy Mullaert, and Marina Vidrascu. Explicit robin-neumann schemes for the coupling of incompressible fluids with thin-walled structures. *Computer Methods in Applied Mechanics and Engineering*, 267:566–593, 2013.

[6] Miguel A. Fernández, Jimmy Mullaert, and Marina Vidrascu. Generalized Robin-Neumann explicit coupling schemes for incompressible fluid-structure interaction: Stability analysis and numerics. *International Journal for Numerical Methods in Engineering*, 101(3):199–229, 2015.

[7] G Holzapfel. Nonlinear solid mechanics: A continuum approach for engineering, 2000.

[8] Jaroslav Hron and Stefan Turek. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. *Fluid-Structure Interaction*, 53:371–385, 2006.

[9] Jie Liu, Rajeev K. Jaiman, and Pardha S. Gurugubelli. A stable second-order scheme for fluid-structure interaction with strong added-mass effects. *Journal of Computational Physics*, 270:687–710, 2014.

[10] Cm Macal. Proceedings of the 2005 Winter Simulation Conference ME Kuhl, NM Steiger, FB Armstrong, and JA Joines, eds. *Simulation*, pages 1643–1649, 2005.

[11] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, Cambridge, 2010.

[12] T Richter and T Wick. On time discretizations of Fluid-structure interactions. *Multiple Shooting and Time Domain Decomposition MEthods*, pages 377–400, 2013.

[13] Thomas Richter. Fluid Structure Interactions. 2016.

[14] Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1):4, 2002.

[15] Natural Sciences. A Newton ' s Method Finite Element Algorithm for Fluid-Structure Interaction. (October), 2012.

[16] Noelle Selin. Verification and Validation. (February), 2014.

[17] K. Stein, T. Tezduyar, and R. Benney. Mesh Moving Techniques for Fluid-Structure Interactions With Large Displacements. *Journal of Applied Mechanics*, 70(1):58, 2003.

[18] Boris Valkov, Chris H Rycroft, and Ken Kamrin. Eulerian method for fluid – structure interaction and submerged solid – solid contact problems.

[19] E. H. van Brummelen. Added Mass Effects of Compressible and Incompressible Flows in Fluid-Structure Interaction. *Journal of Applied Mechanics*, 76(2):021206, 2009.

[20] Frank M White. Viscous Fluid Flow Viscous. *New York*, Second:413, 2000.

[21] Thomas Wick. Adaptive Finite Element Simulation of Fluid-Structure Interaction with Application to Heart-Valve Dynamics. *Institute of Applied Mathematics, University of Heidelber*, page 157, 2011.

[22] Thomas Wick. Fluid-structure interactions using different mesh motion techniques. *Computers and Structures*, 89(13-14):1456–1467, 2011.