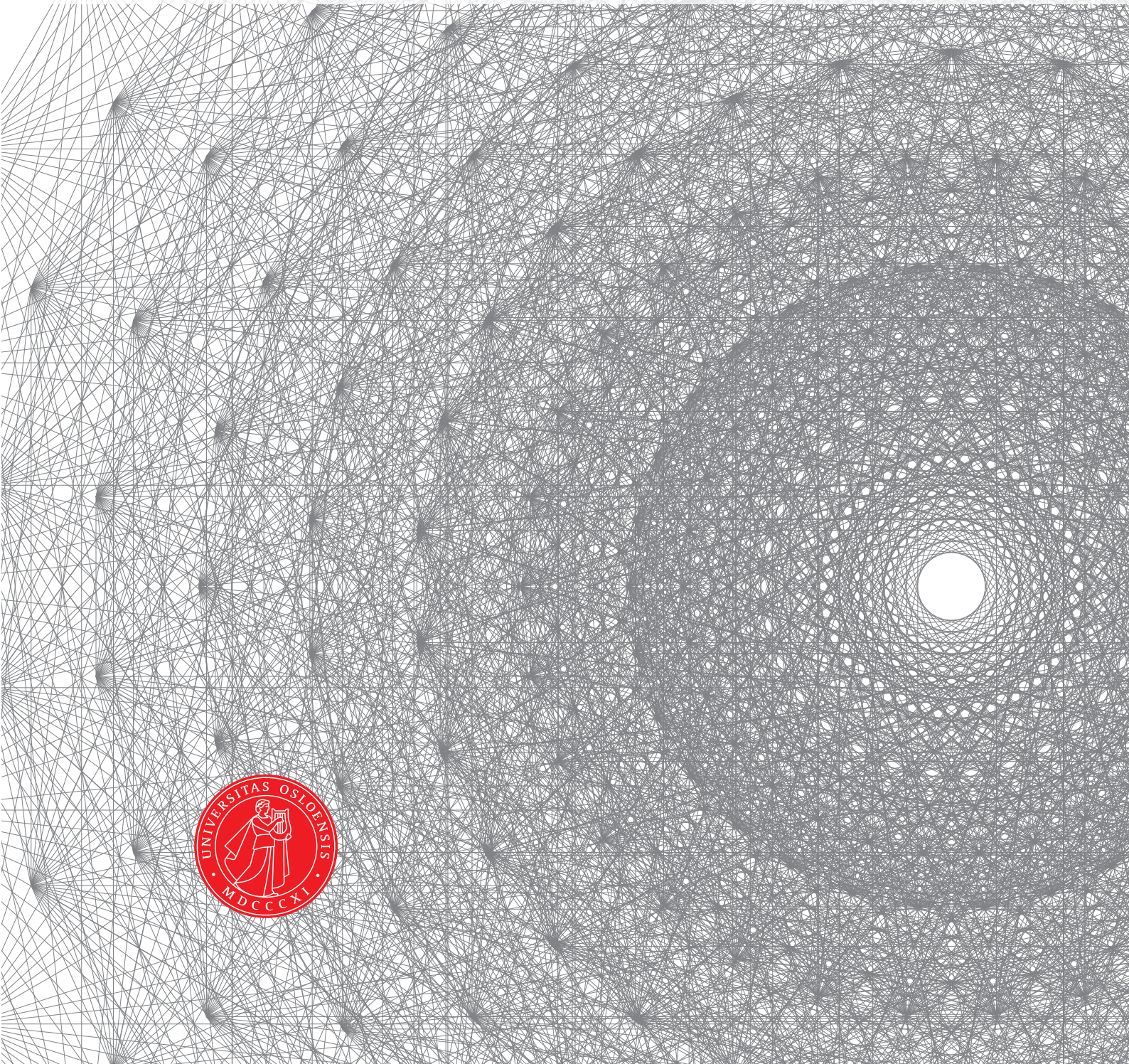
UiO : Department of Mathematics
University of Oslo

Fluid structure interaction

Andreas Strøm Slyngstad

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.

Fluid structure interaction

Andreas Strøm Slyngstad

01 01 01

Contents

Chapter 1

Continuum Mechanics

When studying the dynamics of a mediums with fluid or structure properties under the influence of forces, we need in some sense a good description of how these forces act and alter the system itself.

Any medium on a microscopic scale is built up of a structure of atoms, meaning we can observe "empty spaces " between each atom or discontinuities in the medium. Discribing any phsycial phenomen on larger scales in such a way are tedious and most often out of bounds due to the high number of particles. Instead we consider the medium to be continously distributed throughout the entire reagion it occupies. Hence we want to study some phsyical properties of the complete volume and not down on atomic scale.

We consider the medium with continuum properties. By a continuum we mean a volume $V(t) \subset \mathbb{R}^3$ consiting of particles, which we observe for some properties. One property of interest could be the veloctity $\mathbf{v}(x,t)$ for some point $x \in V(t)$ in time $t \in (0,T]$, which would mean the average velocity of the particles occupying this point x at time t The intension of this chapter is not to give a thorough introduction of continuum mechanics, but rather present key concepts needed for the evaluation of fluid-strucutre interaction.

1.1 Coordinate system

We assume that our medium is continiously distributed throughout its own volume, and we start our observation of this medium at som time t_0. As this choice is arbitary, we often choose to observe a medium in a stress free initial state. We call this state $V(t_0)$ of the medium as the *reference configuration.* We let $V(t)$ for $t \geq t_0$ denote the *current configuration.*

Central for the coordinate systems introduced in this chapter is the concept of *material* and *spatial* points. *Material* points are simply the points defining the material, moving with it as it undergoes movement. *Spatial* points on the other hand is the relative measure of movement of the *material* points. (Godt nok ??). This concept will be further explained throughout the chapter.

1.1.1 Lagrangian

As some medium is act upon by forces, one of the main properties of interest is the deformation of the medium. Hence we want to know the relative position of some particle from its initial configuration.
Let $\hat{\mathrm{x}}$ be a particle in the reference $\hat{\mathrm{x}} \in \hat{\mathrm{V}}$. Further let $\mathrm{x}(\hat{\mathrm{x}}, \mathrm{t})$ be the new location of a particle $\hat{\mathrm{x}}$ for some time t such that $x \in V(t)$. We assume that no two particles $\hat{\mathrm{x}}_a, \hat{\mathrm{x}}_b \in \hat{\mathrm{V}}$ occupy the same location for some time $V(t)$. Then the transformation $\hat{\mathrm{T}}(\hat{\mathrm{x}}, t) = x(\hat{\mathrm{x}}, t)$ maps a particle $\hat{\mathrm{x}}$ from the *reference configuration* $\hat{\mathrm{V}}$ to the *current configuration* $V(t)$ Assuming that the path for some $\hat{\mathrm{x}}$ is continuous in time, we can define the inverse mapping $\hat{\mathrm{T}}^{-1}(x, t) = \hat{\mathrm{x}}(x, t)$, which maps $x(\hat{\mathrm{x}}, t)$ back to its initial location at time $t = t_0$.
These mappings lets us track each particle from some *reference configuration* to some deformed state at time t. Such a description of tracking each particle $\hat{\mathrm{x}} \in \hat{\mathrm{V}}$ is often denoted the *Lagrangian Framework* and is a natural choice of describing structure mechanics.
We define the *deformation*

$$\hat{\mathrm{T}}(\hat{\mathrm{x}}, t) = \hat{u}(\hat{\mathrm{x}}, t) = x(\hat{\mathrm{x}}, t) - \hat{\mathrm{x}} \tag{1.1}$$

and the *deformation velocity*

$$\frac{\partial \hat{\mathrm{T}}(\hat{\mathrm{x}}, t)}{\partial t} = \hat{v}(\hat{\mathrm{x}}, t) = d_t x(\hat{\mathrm{x}}, t) = d_t \hat{u}(\hat{\mathrm{x}}, t) \tag{1.2}$$

When tracking each particle as it moves, the *material* and *spatial* points coincide

1.1.2 Eulerian

Considering a flow of fluid particles in a river, a *Lagrangian* description of the particles would be tidious as the number of particles entring and leaving the domain quickly rise to a immense number. Instead consider defining a view-point V fixed in time, and monitor every fluid particle passing the coordinate $x \in V(t)$ as time elapses. Such a description is defined as the *Eulerian framework.* Therefore the Eulerian formulation is natural for describing fluid dynamics.
We can describe the particles occupying the *current configuration* $V(t)$ for some time $t \geq t_0$

$$x = \hat{\mathrm{x}} + \hat{u}(\hat{\mathrm{x}}, t)$$

Since our domain is fixed we can define the deformation for a particle occupying position $x = x(\hat{\mathrm{x}}, t)$ as

$$\mathbf{u}(x, t) = \hat{u}(\hat{\mathrm{x}}, t) = x - \hat{\mathrm{x}}$$

and its velocity

$$\mathbf{v}(x, t) = \partial_t u(x, t) = \partial_t \hat{u}(\hat{\mathrm{x}}, t) = \hat{v}(\hat{\mathrm{x}}, t)$$

It is important to mention that the we are not interested in which particle is occupying a certain point in our domain, but only its properties. As such the *material* and *spatial* points doesn't coincide in the *Eulerian formulation*

1.2 Deformation gradients

When studying continuum mechanics we observe continious mediums as they are deformed over time. These deformations results in relative changes of positions due to external and internal forces acting.. These relative changes of postition is called *strain*, and is the primary property that causes *stress* within a medium of interest [3]. We define stress as the internal forces that particles within a continuous material exert on each other.

The equations of mechanics can be derived with respect to either a deformed or undeformend configuration of our medium of interest. The choice of refering our equations to the current or reference configuration is indifferent from a theoretical point of view. In practice however this choice can have a severe impact on our strategy of solution methods and physical of modelling. [5]. We will therefore define the strain measures for both configurations of our medium.

Definition 1.1. Deformation gradient.

$$\hat{F} = I + \hat{\nabla}\hat{u} \tag{1.3}$$

Mind that deformation gradient of $\hat{u}$ is which respect to the reference configuration. From the assumption that no two particles $\hat{\mathrm{x}}_a, \hat{\mathrm{x}}_b \in \hat{\mathrm{V}}$ occupy the same location for some time $V(t)$, the presented transformation must be linear. As a consequence from the invertible matrix theorem found in linear algebra, the linear operator $\mathbf{F}$ cannot be a singuar. We define the *determinant of the deformation gradient* as J, which denotes the local change of volume of our domain.

Definition 1.2. Determinant of the deformation gradient

$$J = det(\hat{F}) = det(I + \hat{\nabla}\hat{u}) \neq 0 \tag{1.4}$$

By the assumption that the medium can't be selfpenetrated, we must limit J to be greater than 0 [5]

1.3 Measures of Strain and Stress

The equations describing forces on our domain can be derived in accordinance with the current or reference configuration. With this in mind, different measures of strain can be derived with respect to which configuration we are interested in. We will here by [3] show the most common measures of strain. We will first introduce the right *Cauchy-Green* tensor $\mathbf{C}$, which is one of the most used strain measures [5].
Uttrykk 1.3 fra Godboka, LAG TEGNING

Let $\hat{\mathrm{x}}, \hat{\mathrm{y}} \in \hat{\mathrm{V}}$ be two points in our referemce configuration and let $\hat{\mathrm{a}} = \hat{\mathrm{y}} - \hat{\mathrm{x}}$ denote the length of the line bewtween these two points. As our domain undergoes deformation let $x = \hat{\mathrm{x}} + \hat{u}(\hat{\mathrm{x}})$ and $x = \hat{\mathrm{y}} + \hat{u}(\hat{\mathrm{y}})$ be the position of our points in the current

configuration, and let $a = y - x$ be our new line segment. By [3] we have by first order Taylor expansion

$$y - x = \hat{y} + \hat{u}(\hat{y}) - \hat{x} - \hat{u}(\hat{x}) = \hat{y} - \hat{x} + \hat{\nabla}\hat{u}(\hat{x})(\hat{y} - \hat{x}) + \mathcal{O}(|\hat{y} - \hat{x}|^2)$$
$$\frac{y - x}{|\hat{y} - \hat{x}|} = [I + \hat{\nabla}\hat{u}(\hat{x}]\frac{\hat{y} - \hat{x}}{|\hat{y} - \hat{x}|} + \mathcal{O}(|\hat{y} - \hat{x}|)$$

This detour from [3] we have that

$$a = y - x = \hat{F}(\hat{x})\hat{a} + \mathcal{O}(|\hat{a}|^2)$$
$$|a| = \sqrt{(\hat{F}\hat{a}, \hat{F}\hat{a}) + \mathcal{O}(|\hat{a}^3|)} = \sqrt{(\hat{a}^T, \hat{F}^T\hat{F}\hat{a})} + \mathcal{O}(|\hat{a}^2|)$$

We let $\hat{\mathrm{C}} = \hat{\mathrm{F}}^T\hat{\mathrm{F}}$ denote the right *Cauchy-Green tensor*. By observation the Cauchy-Green tensor is not zero at the reference configuration

$$\hat{\mathrm{C}} = \hat{\mathrm{F}}^T\hat{\mathrm{F}} = (I + \hat{\nabla}\hat{u})^T(I + \hat{\nabla}\hat{u}) = 1$$

Hence it is convenient to introduce a tensor which is zero at the reference configuration. We define the *Grenn-Lagrange strain tensor*, which arises from the squard rate of change of the linesegment $\hat{a}$ and a. By using the definition of the Cauchy-Green tensor we have the relation

$$\frac{1}{2}(|a|^2 + |\hat{a}|^2) = \frac{1}{2}(\hat{a}^T\hat{C}\hat{a} - \hat{a}^T\hat{a}) + \mathcal{O}(|\hat{a}^3| = \hat{a}^T\big(\frac{1}{2}(\hat{F}^T\hat{F} - I)\big)\hat{a} + \mathcal{O}(\hat{a}^3)$$
$$\hat{E} = \frac{1}{2}(\hat{C} - I)$$

Both the *right Cauchy-Green tensor* $\hat{C}$ and the *Green-Lagrange* $\hat{E}$ are refered to the Lagrangian coordinate system, hence the *reference configuration*.
Using similar arguments (see [3], compsda) Eulerian counterparts of the Lagrangian stress tensors can be derived.
The *left Cauchy-Green* strain tensor

$$\mathbf{b} = \hat{\mathrm{F}}\hat{\mathrm{F}}^T =$$

and the *Euler-Almansi* strain tensor

$$\mathbf{e} = \frac{1}{2}(I - \hat{F}^{-1}\hat{F}^{-T}) = \hat{F}^{-1}\hat{E}\hat{F}^T$$

It is important to note that strain itself is nothing else than the measurement of line segments under deformation. Therefore strain alone is purely an observation, and it is not dependent on the material of interest. However one expects that a material undergoing strain, will give forces within the material due to neighboring material interacting with one another. Therefore one derive materialspecific models to describe how a certain material will react to a certain amount of strain.
These strain measures are used to define models for *stress*, which is responsible for the deformation in materials (cite holzapfel). The dimention of stress is force per unit area.

1.4 Governing Equations

The fully Fluid-structure interaction problem is based on equations of balance laws, with auxiliary kinematic, dynamic and material relations. In this section, assumptions regarding these relations will be described briefly. A deeper review of the full FSI problem will be considered in the next chapter.

1.4.1 Fluid

We will throughout this thesis consider in-compressible fluids described by Navier-Stokes equations. We define the fluid density as ρ_f and fluid viscosity ν_f to be constant in time. Our phsyical unknowns fluid velocity v_f and pressure p_f both live in the time-dependent fluid domain $\hat{\Omega}_f(t)$, with an eulerian configuration. Together with the equations of momentum and continuum, the Navier-Stokes equation is defined as,

Equation 1.4.1. *Navier-Stokes equation*

$$\rho \frac{\partial \mathbf{v}_f}{\partial t} + \rho \mathbf{v}_f \cdot \nabla \mathbf{v}_f = \nabla \cdot \sigma + \rho \mathbf{f}_f \quad \text{in } \Omega_f \tag{1.5}$$

$$\nabla \cdot \mathbf{v}_f = 0 \quad \text{in } \Omega_f \tag{1.6}$$

where $\mathbf{f}_s$ is some body force. Assuming a newtonian fluid the *Cauchy stress sensor* σ takes the form
$\sigma = -p_f I + \mu_f(\nabla \mathbf{v}_f + (\nabla \mathbf{v}_f)^T$.
Additional appropriate boundary conditions are supplemented to the equation for a given problem. The first type of of boundary conditions are Dirichlet boundary conditions,

$$\mathbf{v}_f = \mathbf{v}_f^D \quad \text{on } \Gamma_f^D \subset \partial\Omega_f \tag{1.7}$$

The second type of boundary condition are Neumann boundary conditions

$$\sigma_f \cdot \mathbf{n} = \mathbf{g} \quad \text{on } \Gamma_f^N \subset \partial\Omega_f \tag{1.8}$$

1.4.2 The solid

The governing equations for the solid mechanics are given by the blalance law,

Equation 1.4.2. *Solid momentum*

$$\rho_s \frac{\partial \mathbf{v}_s}{\partial t} = \nabla \cdot \hat{\mathbf{T}} + \rho_s \mathbf{f}_s \quad \text{in } \hat{\Omega}_s \tag{1.9}$$

defined in a Lagrangian coordinate system, with respect to an initial reference configuration $\hat{\Omega}_s$. The structure configuration is given by the displacement $\hat{\mathbf{u}}_s$, with the relation $\frac{\partial \hat{\mathbf{v}}}{\partial t} = \hat{\mathbf{u}}_s$ to the solid velocity. The density of the structure is given by ρ_s, and $\hat{\mathbf{f}}_s$ express any exterior body forces acting. The tensor $\hat{\mathbf{T}}$ denotes the first Piola-Kirchhoff stress tensor, with the relation $\hat{\mathbf{T}} = \hat{\mathbf{J}}\sigma_s \hat{\mathbf{F}}^{-T}$ to the cauchy stress tensor. By definition the cauchy stress tensor is symmetric, however the first Piola-Kirchhoff tensor does not exhibit this property. As constitute equations often assumes this behaviour of symmetry, the second Piola-Kirchhoff tensor $\hat{\mathbf{S}}_s$ is convenient as it is symmetric. It is given by the relation to the firt Piola-Kirchhoff stress tensor by,

$$\hat{\mathbf{S}}_s = \hat{\mathbf{F}}^{-T}\hat{\mathbf{T}} = \hat{J}\hat{\mathbf{F}}^{-1}\sigma_s \hat{\mathbf{F}}^{-T}$$

According to the material of interest, several material models exist to model the induced stress given by material deformation. Most famous is Hooke's law, describing a linear relation between strain and stress, limited to a small-deformation regime. As we may no longer be in the range of small-deformation approximation were a linear-elastic material can be used, a consitent way to describe large deformations is needed. As such, for describing large deformation it is widley common to use a stress-strain relation based of the introduced Green Lagrangian strain tensor $\hat{\mathbf{E}}$ and the second Piola-Kirchhoff stress tensor $\hat{\mathbf{S}}$ [1]. Therefore the material is assumed to follow a hyperelastic model, specifically the Vernant-Kirchhoff(STVK) model. Though STVK can handle large deformations, it is limited of the calculation of large strain [1]. However since the deformations considered in this thesis are small, it will remain our primary choice of strain-stress relation. STVK describes materials of compressible nature, but is should be mentioned that for large deformation models describing incompressible materials can be considered. Specially the Incompressible Neo-Hooke (INH) model is considered in several publications (see [4], [2]), sharing the same hyperelastic properties as the STVK model. As both models handles large deformations, the INH is superior compared to STVK in the sense that it is valid for large strains aswell [1].

The STVK is one of the simplest hyperelastic model, as it only extend the famous Hooke's law into a non-linear regime by,

$$\sigma_s = \frac{1}{\hat{J}}\hat{\mathbf{F}}(\lambda_s(Tr(\hat{\mathbf{E}})I + 2\mu\hat{\mathbf{E}})\hat{\mathbf{F}}^{-T} \quad \hat{\mathbf{S}}_s = \lambda_s(Tr(\hat{\mathbf{E}})I + 2\mu\hat{\mathbf{E}}$$
$$\hat{\mathbf{E}} = \frac{1}{2}(\hat{\mathbf{C}} - I) \quad \hat{\mathbf{C}} = \hat{\mathbf{F}}\hat{\mathbf{F}}^{-T}$$

where $\hat{\mathbf{C}}$ is the right Cauchy-Green strain tensor mention in the last subchapter. The solid is often characterized by the Possion ratio and Young modulus. Lamè coefficients λ_s and μ_s are then given by the relation.

$$E_y = \frac{\mu_s(\lambda_s + 2\mu_s)}{(\lambda_s + \mu_s)} \quad \nu_s = \frac{\lambda_s}{2(\lambda_s + \mu_s)}$$
$$\lambda_s = \frac{\nu E_y}{(1+\nu_s)(1-2\nu_s)} \quad \mu_s = \frac{E_y}{2(1+\nu_s)}$$

Since the solid deformation is a quantity of interest a kinematic condition must be defined for the system of the form

$$\frac{\partial \mathbf{v}_s}{\partial t} = \mathbf{u_s} \quad \text{in } \Omega_s \tag{1.10}$$

One might ask the motivation of such an approach as the Lagrangian system could let us define the problem

$$\rho_s \frac{\partial^2 \mathbf{u}_s}{\partial^2 t^2} = \nabla \cdot \mathbf{T} + \rho_s \mathbf{f}_s \quad \text{in } \Omega_s \tag{1.11}$$

directly solving for the main quantity of interest namely deformation. However solving for $\mathbf{v}_s$ is more convenient, as it lets us handle constraints for the fluid-structure interaction problem easier. As for the fluid problem we define Dirichlet and Neumann boundary conditions on the form

$$\mathbf{v}_s = \mathbf{v}_s^D \quad \text{on } \Gamma_s^D \subset \partial\Omega_s$$
$$\sigma_s \cdot \mathbf{n} = \mathbf{g} \quad \text{on } \Gamma_s^N \subset \partial\Omega_s$$

Chapter 2

Implementation of Fluid Structure Interaction

For the general monolithic FSI problem, several complexities arise considering discretization. Yet divided, both the fluid and structure problem themselves impose rather difficult problems. These in combination with the cou- pling of the two subproblems and their interaction to one another, makes even the most simplest implementation surprisingly difficult. Both problem 4.1, 4.2 introduces several non-linear contributions to the governing equations. Firstly the more familiar terms from the convection term of the fluid equation, and the stress tensor of the structure model. As one can not cope and fully understand their full effect, their long-time existence within research makes them well known problems and rigorous approaches exist to.....

Second, the ALE-approach introduces the domain-velocity term to the convection in the fluid problem. Compared to the other mentioned non-linear effects, this contribution is fairly new and its effect on fluid-structure interaction schemes are not fully understood. The stability of the overall time-stepping have proven to be affected by the ALE advection term, which is difficult to control [(formaggia).]. To what extent is this unclear, but several effort by [?], based on [?] have investigated the stability of the ALE formulation with first and second-order time schemes.

This chapter will focus on the mentioned introduced problems by the monolithic ALE method. A brief description will be given for the most central components and technologies used for this thesis. »»»> 5e47855c01074dc45986e8ac56ddf338bf1f792c Chatsamtalen er avsluttet Skriv en melding ...

2.1 Implementation of a one-step θ scheme

The implementation of FSI problem requires some perception, due to the ALE- convection term. By including spatial and temporal differential operators depending non-linearly on one another, the implementation is not directly intu- itive. As mention in CITE(on time discretization of fluid-structure interaction), the discretization of terms were space and time are coupled are based on other types of equations... Les kilgen HT06a. Videre, se på diskretisering av ale termed i paper.

Problem 2.1. *ALE term*

$$\hat{\mathrm{J}}(\hat{F}_W^{-1}(\hat{\mathbf{v}} - \frac{\partial \hat{\mathrm{T}}_W}{\partial t}) \cdot \hat{\nabla})\hat{\mathbf{v}}$$

Problem 2.2. *One-step θ-scheme for laplace and elastic mesh moving model. Find $\hat{\mathbf{u}}_s, \hat{\mathbf{u}}_f, \hat{\mathbf{v}}_s, \hat{\mathbf{v}}_f, \hat{p}_f$ such that*

$$\begin{aligned}
\big(\hat{\mathrm{J}}\frac{\partial \hat{\mathbf{v}}}{\partial t},\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_f} + \big(\hat{\mathrm{J}}(\hat{F}_W^{-1}(\hat{\mathbf{v}} - \frac{\partial \hat{\mathrm{T}}_W}{\partial t}) \cdot \hat{\nabla})\hat{\mathbf{v}},\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_f} + \big(\hat{\mathrm{J}}_W \hat{\sigma} \hat{F}_W^{-T}\hat{\mathbf{n}}_f,\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Gamma}_i} \\
-\big(\hat{\mathrm{J}}_W \hat{\sigma} \hat{F}_W^{-T},\ \hat{\nabla}\hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_f} - \big(\rho_f \hat{\mathrm{J}}\mathbf{f}_f,\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_f} = 0 \\
\big(\rho_s \frac{\partial \hat{\mathbf{v}}_s}{\partial t},\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_s} + \big(\hat{\mathbf{F}}\hat{\mathbf{S}}\hat{\mathbf{n}}_f,\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Gamma}_i} - \big(\hat{\mathbf{F}}\hat{\mathbf{S}},\ \nabla\hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_s} - \big(\rho_s \hat{\mathbf{f}}_s,\ \hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_s} = 0 \\
\big(\frac{\partial \hat{\mathbf{v}}_s - \hat{\mathbf{u}}_s}{\partial t},\ \hat{\boldsymbol{\psi}}^v\big)_{\hat{\Omega}_s} = 0 \\
\big(\nabla \cdot (\hat{\mathrm{J}}\hat{F}_W^{-1}\hat{\mathbf{v}}),\ \hat{\boldsymbol{\psi}}^p\big)_{\hat{\Omega}_f} = 0 \\
\big(\hat{\sigma}_{\mathrm{mesh}},\ \hat{\nabla}\hat{\boldsymbol{\psi}}^u\big)_{\hat{\Omega}_f} = 0
\end{aligned}$$

2.2 FEniCS

The main component of this thesis is the FEniCS project, an open-source finite element environment for solving partial differential equations (https://fenicsproject.org/). Using a combination of high-level Python and C++ interfaces, mathematical models can be implemented compactly and efficiently. FEniCS consists of several submodules and we will give a brief overview of the most central components used during implementation and computation.

2.2.1 DOLFIN

DOLFIN is the computational C++ backend of the FEniCS project, and the main user interface. It unifies several FEniCs components for implementing of computational mesh, function spaces, functions and finite element assembly.

- UFL (The Unified Form Language) is a domain specific language, used for the discretization of mathematical abstractions of partial differential equations on a finite element form. Its implementation on top of Python, makes it excellent to define problems close to their mathematical notation without the use of more complex features. One uses the term *form* to define any representation of some mathematical problem defined by UFL.
- FFC (The form compiler) compiles the finite elements variation forms given by UFL, generating low-level efficient C++ code
- FIAT the finite element backend, covering a wide range of finite element basis functions used in the discretization of of the the finite-element forms. It covers a wide range of finite element basis functions for lines, triangles and tetrahedras.

DOLFIN also incorporate the necessary interfaces to external linear algebra solvers and data structures. Within FEniCS terminology these are called linear algebra backends. PETSc is the default setting in FEniCS, a powerful linear algebra library with a wide range of parallel linear and nonlinear solvers and efficient as matrix and vector operations for applications written in C, C++, Fortran and Python.

2.3 Implementation

As implementation of mathematics differ from the choices of programming languages and external libraries, a deep dive within the implementation in FEniCS will not be covered in this thesis. Only variational forms and solvers will be presented as to give the reader a general overview of the key concept and the interpretation of mathematics. Basic knowledge of coding is assumed of the reader.

2.3.1 Variational Form

Implementation of the code-blocks of the fluid variational form given in Chapter 3, and Newton solver will be presented. It is not the intention to give the reader a deep review of the total implementation, but rather briefly point out key ideas intended for efficient speedup of the calculation. These ideas have proven essential as for the reduction of computation time of the complex problem.

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

def J_(U):
        return det(F_(U))

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))

def A_E(J, v, d, rho_f, mu_f, psi, dx_f):
    return rho_f*inner(J*grad(v)*inv(F_(d))*v, psi)*dx_f \
        + inner(J*sigma_f_u(v, d, mu_f)*inv(F_(d)).T, grad(psi))*dx_f


def fluid_setup(v_, p_, d_, n, psi, gamma, dx_f, ds, mu_f, rho_f, k, dt, v_deg
    , theta, **semimp_namespace):

        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

        return dict(F_fluid_linear = F_fluid_linear, F_fluid_nonlinear = F_fluid_nonlinear)
```

Algorithm 2.1: thetaCN.py

Alorithm 1.1 presents the implementation of the fluid residue, used in the Newton iterations. Apart from the rather lengthy form of the fluid residual, the strength of Unified Form Language preserving the abstract formulation of the problem is clear. The overall representation of the problem is by now just a form, its a representation and does not yet define vectors or matrices.

```
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 % 4  == 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
```

Algorithm 2.2: newtonsolver.py

2.4 Optimization of Newtonsolver

As for any program, the procedure of optimization involves finding the bottleneck of the implementation. Within computational science, this involves finding the area of code which is the primary consumer of computer resources.
As for many other applications, within computational science one can often assume the consummation of resources follows the *The Pareto principle*. Meaning that for different types of events, roughly 80% of the effects come from 20% of the causes. An analogy to computational sciences it that 80% of the computational demanding operations comes from 20% of the code. In our case, the bottleneck is the newtonsolver. The two main reasons for this is

- **Jacobian assembly**
  The construction of the Jacobian matrix for the total residue of the system, is the most time demanding operations within the whole computation.

- **Solver**.
  As iterative solvers are limited for the solving of fluid-structure interaction problems, direct solvers was implemented for this thesis. As such, the operation of solving a linear problem at each iteration is computational demanding, leading to less computational efficient operations. Mention order of iterations?

Facing these problems, several attempts was made to speed-up the implementation. The FEniCS project consist of several nonlinear solver backends, were fully user-customization option are available. However one main problem which we met was the fact that FEniCS assembles the matrix of the different variables over the whole mesh, even though the variable is only defined in one to the sub-domains of the system.In our case the pressure is only defined within the fluid domain, and therefore the matrix for the total residual consisted of several zero columns within the structure region. FEniCS provides a solution for such problems, but therefore we were forced to construct our own solver and not make use of the built-in nonlinear solvers.

The main effort of speed-up were explored around the Jacobian assembly, as this was within our control.

Of the speed-ups methods explored in this thesis we will specify that some of them were *consistent* while others were *nonconsistent*. Consistent methods are methods that always will work, involving smarter use of properties regarding the linear system to be solved. The non-consistent method presented involves altering the equation to be solved by some simplification of the system. As these simplifications will alter the expected convergence of the solver, one must take account for additional Newton iterations against cheaper Jacobi assembly. Therefore one also risk breakdown of the solver as the Newton iterations may not converge.

2.5 Consistent methods

2.5.1 Jacobi buffering

By inspection of the Jacobi matrix, some terms of the total residue is linear terms, and remain constant within each time step. By assembling these terms only in the first Newton iteration will save some assembly time for the additional iterations needed each time step. As consequence the convergence of the Newton method should be unaffected as we do not alter the system.

2.6 Non-consisten methods

2.6.1 Reuse of Jacobian

As the assembly of the Jacobian at each iteration is costly, one approach of reusing the Jacobian for the linear system was proposed. In other words, the LU-factorization of the system is reused until the Jacobi is re-assembled. This method greatly reduced the computational time for each time step. By a user defined parameter, the number of iterations before a new assembly of the Jacobian matrix can be controlled.

2.6.2 Quadrature reduce

The assemble time of the Jacobian greatly depends on the degree of polynomials used in the discretisation of the total residual. Within FEniCS this parameter can be controlled, and as such we can specify the order of polynomials representing the Jacobian. The use of lower order polynomials reduces assemble time of the matrix at each newton-iteration, however it leads to an inexact Jacobian which may results to additional iterations.

Bibliography

[1] M Razzaq, Stefan Turek, Jaroslav Hron, J F Acker, F Weichert, I Grunwald, C Roth, M Wagner, and B Romeike. Numerical simulation and benchmarking of fluid-structure interaction with application to Hemodynamics. *Fundamental Trends in Fluid-Structure Interaction*, 1:171–199, 2010.

[2] T. Richter and T. Wick. Finite elements for fluid-structure interaction in ALE and fully Eulerian coordinates. *Computer Methods in Applied Mechanics and Engineering*, 199(41-44):2633–2642, 2010.

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

[4] Thomas Wick. Fully Eulerian fluid-structure interaction for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 255:14–26, 2013.

[5] P. Wriggers. *Computational contact mechanics, second ed., Springer*. 2006.