- Noe du gjør ofte: ..., by noe.
  Det er veldig skjelden en god setning
- Du må få fikset formateringen på avsnittene dine, har sagt det flere ganger nå.
- Du må ha fult fokus på den siste delen, for der mangler det en del, og den må på plass ASAP.

Contents

Capitalize my title. com

Alt. 2 kap.

Chapter 1

A motivation for studying fluid-structure interaction

motsatt, nature

Dette setningen har jeg lest i minst tre forskjellig kapitler.

Fluid-structure interaction(FSI) is an interdiciplinary field, appearing in many applications. In nature, FSI forms the basis of many physical phenomia. A fish swimming utpstream, generating thrust from the surrounding fluid by wave-like movements of its fin and body. Or a tree, bedning back and fourth due to strong winds of a storm passing by. Both examples are understandable, but points out two main instances of how FSI occur. When the fish swims, it deformes the fluid, altering the nearby flowfield. For the tree however, the swinging and bending is induced by the pressure of passing wind acting on the tree trunk and branches. Ultimatly, fluid-structure interaction occurs due to both initial effect of either fluid, structure or a combination.

ikke fullstendig setning

ikke de

↓

Jeg forstår ikke poenget med dette avsnittet. hvorfor er det viktig å poengtere dette. Er dette en distinksjon som du trenger senere?

Generelt: \cite{cite1, cite2}

ikke \cite{cite1}, \cite{cite2} kommentert dette flere ganger

Computational fluid-structure interaction (CFSI) has grown wast within engingeering in the recent years, and proved to be essential for design development and performance optimalizastion of many applications. Applications are, but not limited to biomedical computations such as heart valves and aneurisms([39], [41]) inflation of parachutes [38] ,Underwater explosions [19] and wind turbines [16].
Within aeronautics, CFSI have proven to be crucial for advances within flight characteristics and fuel economy. Due to a wide range of wing materials and flow profiles to be studied, CFSI have made testing of proposed models possible, while saving expences regarding small and full-scale experiments.

↳ reducing cost of...

forsiktig med ordet models, bruk det konsikvent om en matematisk modell gjennom hele oppgaven.

Winglet, a near vertical tip replacement for a conventional wingtip of an aircraft, have reduced drag induced by wingtip vortices during flight. As a result, the overall fuel consumption of long-distance flights have been reduced by $\approx 5\%$, which is why winglets can be observed within many airliners today. Another consequence of installing wingelts is the recution of wingtip vortices, which in turn reduces trailing turbulence behind the aircraft. The trailing turbulence can intervene with flight controls of aircraft passing through it, making wingles an important safety feature for flight traffic.

spor om not?

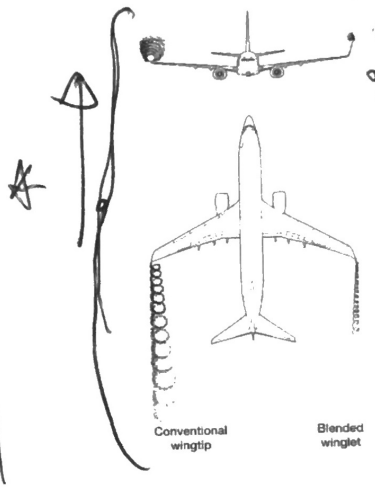

Figure 1.1: A comparison of shedding vortices from conventional wingtip, versious a winglet.

Given the multidiciplinary nature of FSI, significant developments within the field have occured within recent years. Traditionally, fluid and structure mechanics have been considered separate scientific fields, hower the complex interaction There are several causes for , coupling of equations and nonlinearity
Even though FSI play an important role within many scientific apllications,
Computational stuff, why now, refer to computational power etc
Få med beregningsorienterte, to interdiciplinary.

ostyggt

★ Is this a succses story of CFSI? Find a reference and write this explicitly.

- skriv ferdig
- få med noen klare mål som du kan komme tilbake til i løpet av oppgaven. Det vil binde den mye bedre sammen.

Ser ikke ut som du har implementert noen av mine tilbakemeldinger, så kommenterer ikke noe på dette.

Chapter 2

Governing equations

Computational fluid-structure interaction (CFSI) is a multi-physics field of science, combining two separate fields of computational mechanics, computational fluid dynamics (CFD), and computational structure dynamics (CSM). While CFD and CSM traditionally have been considered as two distinct fields of science, the goal of CFSI is to combine the separate fluid and structure problems ,and their interaction or *coupling* to one another. Therefore, the study CFSI demands understanding of each separate field. This chapter presents the governing equations of the individual fluid and structure problem. Balance laws for each separate problem, together with auxiliary kinematic, dynamic and material relations will be described briefly.

2.1 Continuum Mechanics

To interpret nature, mathematical models are needed to describe how object around us reacts to external and/or internal stimuli. The mathematical models forms a basis for establishing elementary conservation laws and partial differential equations (PDE's), making scientist and engineers not only able to understand physical phenomena, but also predict them.
Fluid and solids are both materials built up by a sequence of atoms, meaning on a microscopic level, an observer will locate discontinuties and space within the material. Evaluating each atom, or *material point*, is not impossible from a mathematical point of view. However, for mathematical moddeling and applications, the evaluation of each material point remains unpractical. In *Continuum mechanics*, first formulated by Augustin-Louis Cauchy [21], the microscopic structure of materials are ignored, assuming the material of interest is *continuously distributed* in space.

samme med dette kapitlet.

Chapter 3

Computational Fluid Structure Interaction

The multi-disciplinary nature of computational fluid-structure interaction , involves addressing issues regarding computational fluid dynamics and computational structure dynamics. In general, CFD and CSM are individually well-studied, in terms of numerical solution strategies. CFSI adds another layer of complexity to the solution process, (1) the *coupling* of the fluid and solid equations , (2) the tracking of *interface* separating the fluid and solid domains. The coupling pose two new conditions at the interface absent from the original fluid and solid conditions, which is *continuity of velocity* and *continuity of stress* at the interface.

$$\mathbf{v}_f = \mathbf{v}_s \tag{3.1}$$

$$\sigma_f \cdot \mathbf{n} = \sigma_s \cdot \mathbf{n} \tag{3.2}$$

The tracking of the interface is a issue, due to the different description of motion used in the fluid and solid problem. If the natural coordinate system are used for the fluid problem and solid problem, namely the eulerian and lagrangian description of motion, the domains doesn't match and the interface. Tracking the interface is aslo essential for fulfilling the interface boundary conditions. As such only one of the domains can be described in its natural coordinate system, while the other domain needs to be defined in some transformed coordinate system.
Fluid-structure interaction problems are formally divided into the *monolithic* and *partitioned* frameworks. In the monolithic framework, the fluid and solid equations together with interface conditions are solved simultaneously. The monolithic approach is *strongly coupled*, meaning the *kinematic* (1.1) and *dynamic*(1.2) interface conditions are met with high accuracy. However, the complexity of solving all the equations simuntainiously and the strong coupling contributes to a stronger nonlinear behaviour of the whole system [47]. The complexity also makes monolithic implementations *ad hoc* and less modular, and the nonlinearity makes solution time slow.
In the *partitioned* framework one solves the equations of fluid and structure subsequently. Sovling the fluid and solid problems individually is beneficial, in terms of the wide range of optimized solvers and solution strategies developed for each

RVS

Chapter 4

Verification and Validation

Computer simulations are in many engineering applications a cost-efficient way for conducting design and performance optimalization of physical problems. However, thrusting blindly numbers generated from a computer code can prove to be naive. It doesn't take a lot of coding experience before one realizes the many things that can brake down and produce unwanted or unexpected results. Therefore, *credability* of computational results are essential, meaning the simulation is worthy of belief or confidence [24]. *Verification and validation* (V&V) is the main approach for assessing and the reliability of computational simulations [36]. A thorough discussion of (V&V) concepts and terminology during the last century can be found in [24]. In this thesis, the definitions provided by the *American Society of Mechanical Engineers guide for Verification and Validation in Computational Solid Mechanics* [33] are followed.

Definition 4.1. Verification: The process of determining that a computational model accurately represents the underlying mathematical model and its solution.

Definition 4.2. Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

Simplified *verification* considers if one solves the equations right, while *validation* is checking if one solves the right equations for the given problem [30].
To test a computational code for all possible parameters, conditions and applications are simply too time consuming. Verification and validation are therefore ongoing processes, with no clear boundary of completeness unless additional requirements are specified [30]. The goal of this chapter is to verify our implementations using the method of manufactured solution (MMS), addressing validation in a later chapter.

4.1 Verification of Code

Within scientific computing a mathematical model is often the baseline for simulations of a particular problem of interest. For scientists exploring physical phenomena, the mathematical model is often on the form of systems of partial differential equations (PDE´s). A computer program therefore must evaluate mathematical

Chapter 5

Implementation of a Fluid-Structure Interaction solver

This chapter will focus on the implementation of a fluid-structure interaction solver, with emphasis on general implentation issues.

5.1 Black-box solvers versus a self-made implementation

A black box is any device whose workings are not understood by or accessible to its user. In terms of computational science, the term is often associated with commercially available code where the user is limited to the control of input parameters, with no intervention of the code itself. Trusting commercial software blindly is risky. Even though the software has been rigorous tested, the lack of full transparancy of the implementation ...(full understanding og math). In addition a full understanding of the software also takes time to learn.

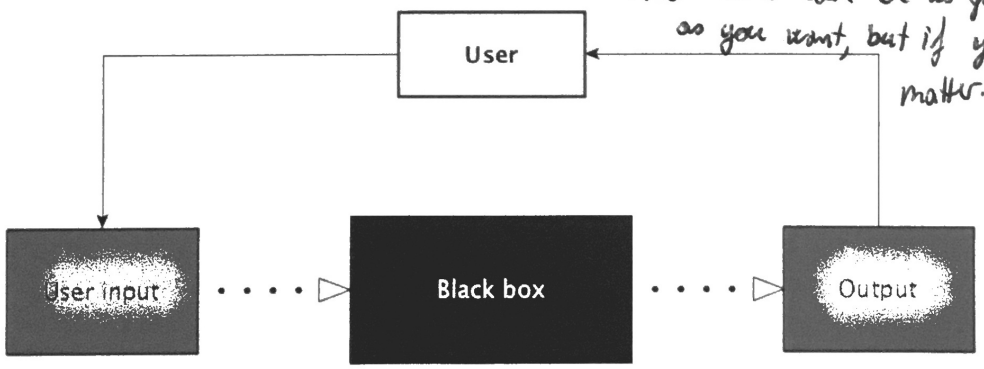

Figure 5.1: Computational domain of the validation benchmark

Several commercial and available solvers for FSI exists, both for monolithic and partitioned approaches. The commerical software ADINA makes solvers based on

both approaches available for the user [52]. The open-source OpenFOAM also incoporate both solvers, howevere enabling more flexibillity in combining different solution strategies ...

The disiplines of CFD and computational structural dynamics (CSD) have tradidionally been treated separatly. Hence, highly optimized and specialized software has been developed individually, with little or no regard to multiphysics such as fluid-structure interaction. Using a partitioned approach, the MpCCI interface is considered the "de-facto" software, for coupling separate fluid and solid solvers [[50], [12]].

hvor du introdusert dette?

In this thesis a monolithic FSI-solver is developed without the use of commercial available software. The main motivation for this choice is the full understanding of the implementation, and the ability for code intervention along the solution process. The process of implementing a self-made solver for fluid-structure interaction is not stright forward. Due to the overall size of the total system, and the assumption of a non-linear material and large deformations makes the process..
The implementation of the FSI-solver relies soley on the open-source project FEniCS, a computing platform for solving partial differential equation. The next section will present the general consepts for this..

Jeg synes dette hører hjemme i en introduksjon

5.2 FEniCS

The FEniCS project is 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 sub-modules and we will give a brief overview of the most central components used during implementation and computation.

5.2.1 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.
A comprehensive introduction FEniCS is out of the scope for this thesis, and for further insight the reader is directed to the introductional demos presented on https://fenicsproject.org/.

5.2.2 Poission Equation, hello world

When learning programming, it is common to present a "Hello world!" program, presenting the fundamental concepts of the software of interest. For solving PDE's, one of the most basic challenges is solving the Possion's equation. Let Ω be the computational domain and let $\partial\Omega$, be the boundary of Ω. The Poission equation takes the form,

$$\begin{aligned} -\nabla^2 u &= f \quad \in \Omega \\ u &= u_d \quad \in \partial\Omega \end{aligned}$$

where u is the unknown of interest and f is some known function. ∇^2 is the Laplace operator and $u = u_d$ is a prescribed Dirichlet condition on the boundary. Assuming the reader has some knowledge of the finite-element method, the unknown function u is approximated by a *trial function*. The problem is then multiplied with a *test function* v, and then we use integration by-parts over the domain Ω.

$$-\int_\Omega \nabla^2 u v dx = \int_\Omega f dx$$
$$-\int_{\partial\Omega} \frac{\partial u}{\partial n} v ds + \int_\Omega \nabla u \nabla v dx = \int_\Omega f dx$$

For simplicity, the *test function* $v = 0$ on the boundary $\partial\Omega$ due to the prescribed Dirichlet condition which leaves us with the system

$$\int_\Omega \nabla u \nabla v dx = \int_\Omega f dx$$

With the primilaries set, we focus on the implementation of the probelem in FEniCS. The poission problem can then be expressed,

```
1  from fenics import *
2
3  frame#frame frameCreateframe framemeshframe frameandframe framedefineframe framea
       frame frame1.frameorderframe framelagrangianframe framefunctionframe framespace
4  mesh = UnitSquareMesh(10, 10)
5  V = FunctionSpace(mesh, 'CG', 1)
6
7  frame#frame frameDefineframe framedirichletframe frameboundaryframe framecondition
8  u_bnd = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]', degree=2)
9
10 bcs = DirichletBC(V, u_bnd, "on_boundary")
11
12 Define variational problem
13 u = TrialFunction(V)
14 v = TestFunction(V)
15 f = Constant(-10.0)
16 a = dot(grad(u), grad(v))*dx
17 L = f*v*dx
18
19 frame#frame frameSolveframe frameproblemframe frameandframe framecomputeframe
       framesolution
20 u = Function(V)
21 solve(a == L, u, bc)
```

Algorithm 5.1: posssion.py

The possion problem expressed with FEniCS points out two important properties. Frist, the overall problem is implemented compactly while each code segement for solving the problem remains clear. Second, the abstract notation remains close to the orignal problem of interest.

5.3 implemented code

The overall codestructure of the implementation is based on dividing the full code into fragments of Python modules. The main idea is to keep key code segments functionality while maintaining modularity to each code segment.

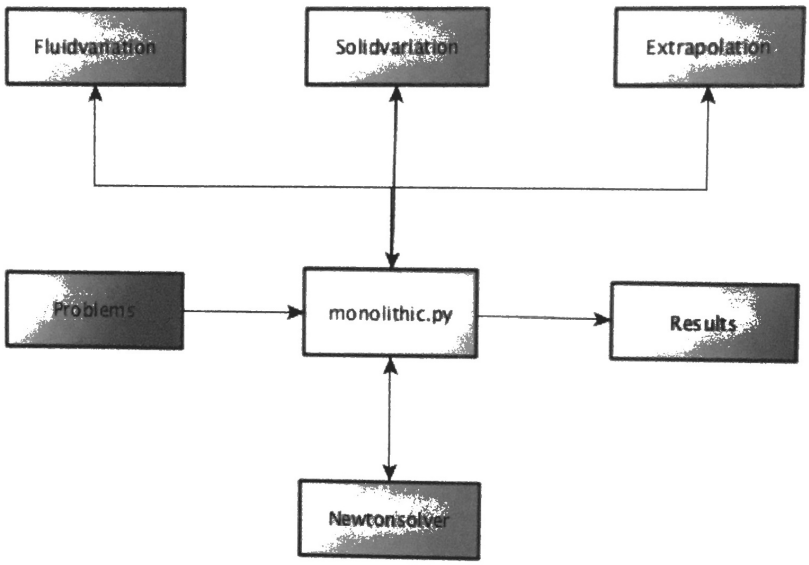

Figure 5.2: Computational domain of the validation benchmark

kjøre eksempler:

python monolithic.py -opt1 xx --dt 0.01

modul bort => - hvorfor det er enkelt å implementere et nytt problem / skjema
- checkpoint
- Lett å utforske parameterrom / bra for reproduserbar forskning

$\frac{\partial d}{\partial t}$ CN og CN long-term er ikke samme problem, ikke sant? Det er det ikke lett å forstå ut i fra teksten her. Har du noen tester knyttet til $\frac{\partial T_f}{\partial t}$ CN i det hele tatt?

Chapter 6

Numerical Experiments

6.1 Investigation of temporal stability

Ta ut?

Preliminary work regarding discretization and numerical analysis of the second order Crank-Nicholson time stepping scheme for fluid structure interaction can be found in [45]. Two main properties of interest have proven to be the stability of long-time simulation, and obtaining the expected physics for the problem of interest. One of the main challenges for constructing time-stepping schemes for ALE-methods, is the additional non-linearity introduced by the domain-velocity term in the fluid problem.

$$\hat{\mathrm{J}}(\hat{F}_f^{-1}(\hat{\mathbf{v}} - \frac{\partial \hat{\mathrm{T}}_f}{\partial t}) \cdot \hat{\nabla})\hat{\mathbf{v}} \tag{6.1}$$

The stability of first and second-order time stepping schemes for ALE-methods have proven to be affected by the ALE-convection term [9, 8], but to what extent remains unclear. Closer inspection of the convection term reviles spatial and temporal differential operators depending non-linearly on one another. These differential operators often appear separated, making discretization of a general time-stepping scheme ~~is~~ not directly intuitive, and often based on the experience of similar equations such as the Navier-Stokes equations.

The Crank-Nicolson scheme can suffer from temporal stability for long-term simulations [44]. Therefore the authors of [28], investigated temporal stability of the Crank-Nicolson scheme for the validation benchmark found in [15]. The criteria for the numerical experiments was to obtain a stable solution in the time interval [0, 10] minutes, by temporal and spatial refinement studies. The fully monolithic FSI problem discretized with second-order Crank-Nicolson, proved to give general stability problems for long-term simulation for certain time-steps k. Following the ideas of [28], a second order scheme based on the Cranck-Nicholson yields two possibilities.

and studied in this thesis.

Discretization 6.1. *Crank-Nicolson secant method*

$$\left[\frac{\hat{\mathrm{J}}(\hat{\mathbf{u}}^n)\hat{\nabla}\hat{\mathbf{v}}^n\hat{\mathbf{F}}_W^{-1}}{2} + \frac{\hat{\mathrm{J}}(\hat{\mathbf{u}}^{n-1})\hat{\nabla}v^{n-1}\hat{\mathbf{F}}_W^{-1}}{2}\right]\frac{\hat{\mathbf{u}}^n - \hat{\mathbf{u}}^{n-1}}{k}$$

6.2. *Crank-Nicolson midpoint-tangent method*

$$\left[\frac{\hat{\mathbf{J}}(\hat{\mathbf{u}}_{cn})\hat{\nabla}\hat{\mathbf{v}}_{cn}\hat{\mathbf{F}}_W^{-1}}{2}\right]\frac{\hat{\mathbf{u}}^n-\hat{\mathbf{u}}^{n-1}}{k} \quad \hat{\mathbf{u}}_{cn}=\frac{\hat{\mathbf{u}}^n+\hat{\mathbf{u}}^{n-1}}{2} \quad \hat{\mathbf{v}}_{cn}=\frac{\hat{\mathbf{v}}^n+\hat{\mathbf{v}}^{n-1}}{2}$$

The numerical experiments showed very similar performance for Discretization 6.1 and 6.2 , and significant differences of temporal accuracy was not found [28]. Two options to coupe with the presented unstabilities are the *shifted Crank-Nicolson* [28, 46, 44], and the *frac-step method*. Both of these methods are defined as A-stable time-stepping schemes meaning.. In this thesis the shifted Crank-Nocolson scheme was considered, introducing stability to the overall system by shifting the θ paramter slightly to the implicit side. If the shift is dependent of the time-step k such that $\frac{1}{2} \leq \theta \leq \frac{1}{2}+k$, the scheme will be of second order [28].

Results

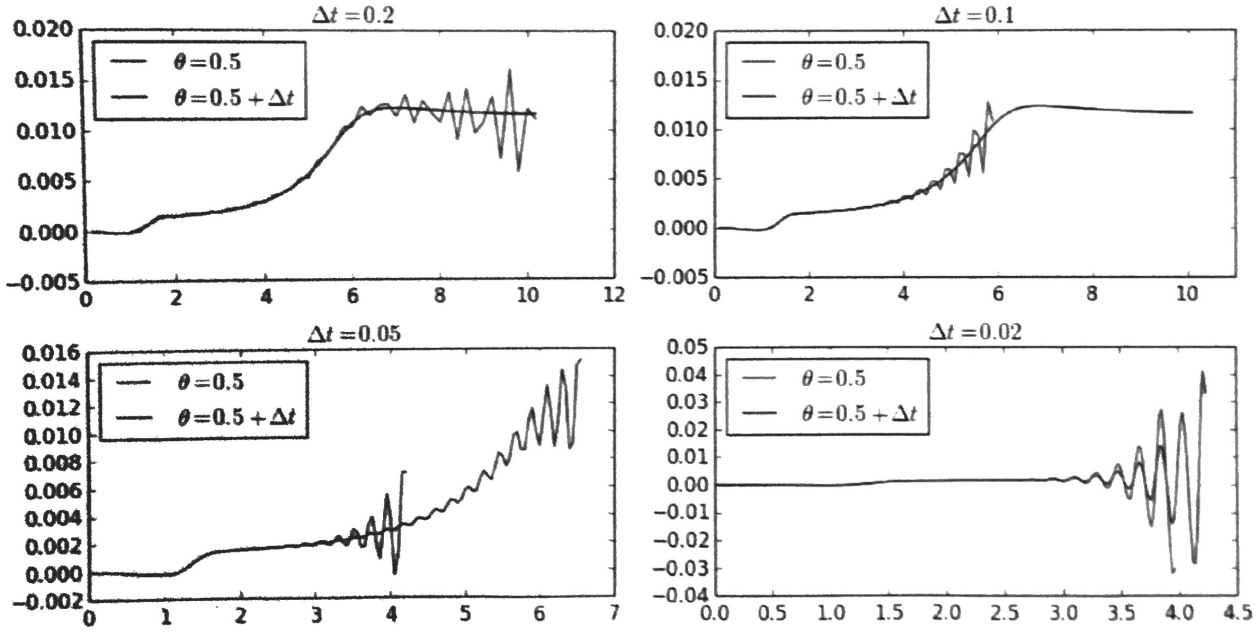

Figure 6.1: Investigation of temporal stability for the FSI3 benchbark in the time interval $t \in [0, 10]$, comparing the shifted crank nicolson to the original cranc nicolson scheme.

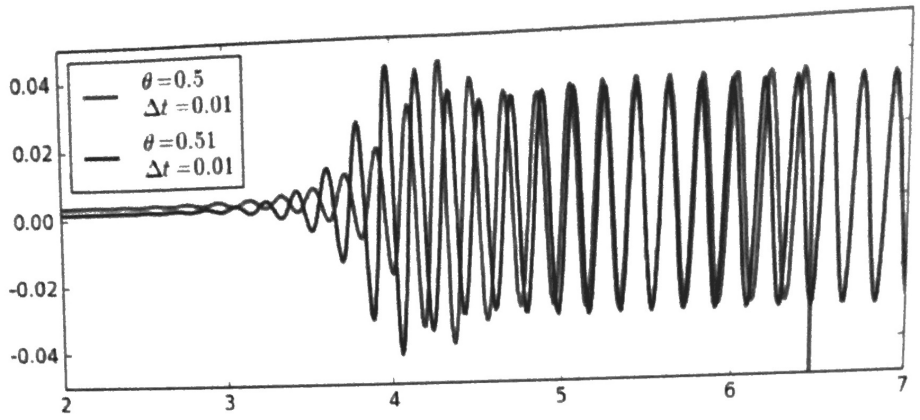

Figure 6.2: Investigation of temporal stability for the FSI3 benchbark in the time interval $t \in [0, 10]$, comparing the shifted crank nicolson to the original cranc nicolson scheme.

A numerical investigation of temporal stability in shown in Figure 6.1-2, where the shifted crank-nicolson scheme $\theta = 0.5 + k$, is compared the original crank-nicolson $\theta = 0.5$. The shifted version clearly show stability properties surpassing the original crank-nicolson scheme, for all numerical experiments. However, for $\Delta t \in [0.2, 0.1, 0.05, 0.02]$ the scheme clearly lacks the ability to capture the overall physics of the validation problem. Long-time stability and expected physical beavior is obtained for $\Delta t = 0.01$. However, numerical experiments showed that for $\Delta t \leq 0.005$ numerical stability was achieved regardless of both methods. This result is important, reducing the overall computational time needed to achieve reasonable accuracy.

Disc.

Discussion

- Importance of CN on solid. I would also want to see $\theta=1$ and $\theta=0$ as well.
- Perhaps move the $\frac{\partial d}{\partial t}$ on here
- Impact of these problems
- What is common in the litterature?

6.2 Optimization of Newtonsolver

A *bottleneck* express a phenomena where the total performance of a complete implementation is limited to small code fragments, accounting for the primary consumption 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. Of the speed-ups methods explored in this thesis, some are *consistent* while others are *nonconsistent*. Consistent methods are methods that always will work, involving smarter approaches 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.

6.2.1 Consistent methods

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.

6.2.2 Non-consisten methods

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.

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 the order of polynomials representing the Jacobian can be adjusted. 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.

6.2.3 Comparison of speedup methods

Table 6.1: Comparison of speedup techniques

Laplace					
Implementation	Naive	Buffering	Reducequad.	Reusejacobi	Combined
Mean time/timestep	123.1		31.4	61.3	11.1
Speedup %	1.0		74.46%	50.19%	90.97 %
Mean iteration	4.49		10.1	10.2	10.2
Biharmonic Type 1					
Implementation	Naive	Buffering	Reducequad.	Reusejacobi	Combined
Mean time/timestep	243.3	307.6	51.6	76.7	24.8
Speedup %	1.0	-26%	78.7%	68.4 %	89.7%
Mean iteration	4.1	6.2	4.6	7.1	6.8
Biharmonic Type 2					
Implementation	Naive	Buffering	Reducequad.	Reusejacobi	Combined
Mean time/-timestep			60.5	95.3	20.7
Speedup %	1.0	%	%	%	%
Mean iteration	4.1		6.29	6.9	6.9

Describe your results

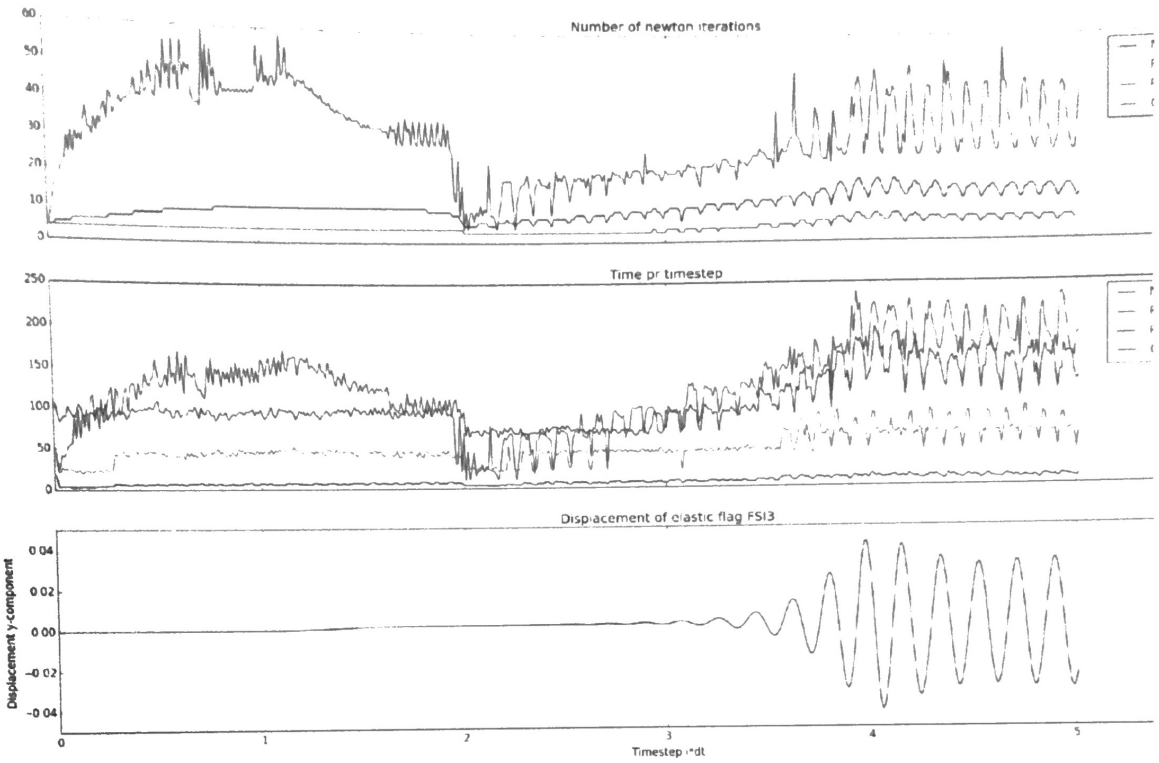

Figure 6.3: Comparison of speed-up techniques for the laplace mesh model

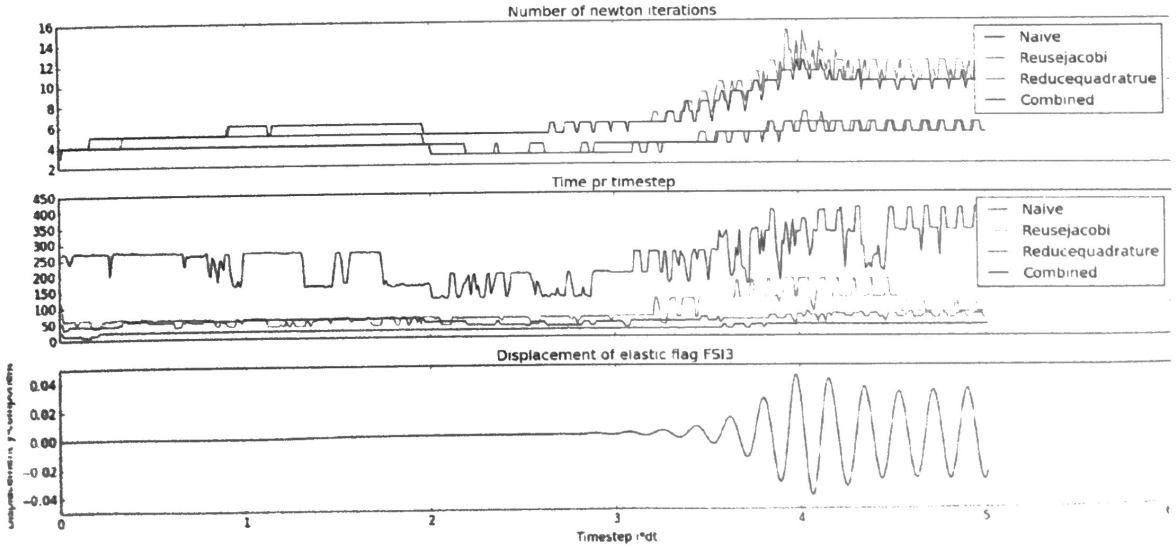

Figure 6.4: Comparison of speed-up techniques for the biharmonic type 1 mesh model