Fluid structure interaction

Andreas Strøm Slyngstad

01 01 01

Contents

1 Implementation of a Fluid-Structure Interaction solver **3**
1.1 Black-box solvers versus a self-made implementation 3
1.2 FEniCS 4
1.2.1 DOLFIN 4
1.3 Implementation 5
1.3.1 Variational Form 6

2 Time discretization and optimalization **9**
2.1 Implementation of a one-step θ scheme 10
2.1.1 Temporal stability 11
2.2 Optimization of Newtonsolver 12
2.3 Consistent methods 13
2.3.1 Jacobi buffering 13
2.4 Non-consisten methods 13
2.4.1 Reuse of Jacobian 13
2.4.2 Quadrature reduce 13

Chapter 1

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.

1.1 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.

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 [8]. 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 [[7], [1]].

In this thesis the entire 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 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..

1.2 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.

1.2.1 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.

1.3 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.

1.3.1 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 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 1.1: 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 1.2: newtonsolver.py

Chapter 2

Time discretization and optimalization

The aim of this chapter is to present some of the main challenges regarding discretization of a general monolithic fluid-structure interaction(FSI) problem, using the ALE-framework. Even separately, the discretization of fluid and structure problems impose rather difficult issues due to their non-linear nature. However, their long-time existence within research community makes them well known problems and a vast number of rigorous approaches and commercial software exist to solve them individually. When solving the fluid and structure simultaneously however, the overall problem gets more complex due to the overall dependency of the two sub-problems and their interaction to one another.

One of the main challenges is the additional non-linearty intruduced by the domain-velocity term in the fluid problem.

Problem 2.1. *ALE term*

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

Closer inspection of the convection term reviels spatial and temporal differential operators depending non-linearly on one another. Within computational science, these operators often appear separated. Therefore the 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. In this thesis, time-stepping schemes of second order will be considered. It has been reported in [] [], that the stability of first and second-order time stepping schemes are affected by the ALE-convection term, but to what extent remains unclear.

Though only the fluid problem will be discussed, it must be emphasized that the discretization of the solid problem is of great importance. Several studies exists for the individual solid problem, but a deeper analysis considering a fluid-structure interaction setting is abvient from the FSI litterature [3].

2.1 Implementation of a one-step θ scheme

For both the fluid problem and the structure problem, we will base our implementation of a θ-scheme. A θ-scheme is favourable, making implementation of classical time-steppings schemes simple. For the structure problem, θ-scheme takes the form

Problem 2.2.

$$\rho_s \frac{\partial \hat{\mathbf{v}}_s}{\partial t} - \theta \nabla \cdot \hat{\mathbf{F}}\hat{\mathbf{S}} - (1-\theta)\nabla \cdot \hat{\mathbf{F}}\hat{\mathbf{S}} - \theta\rho_s \hat{\mathbf{f}}_s - (1-\theta)\rho_s \hat{\mathbf{f}}_s = 0$$
$$\frac{\partial \hat{\mathbf{v}}_s}{\partial t} - \theta \hat{\mathbf{u}}_s - (1-\theta)\hat{\mathbf{u}}_s = 0$$

For $\theta \in [0,1]$ classical time-stepping schemes are obtained such as the first-order forward Euler scheme $\theta = 0$, backward-Euler scheme$\theta = 1$, and the second-order Crank-Nicholson scheme $\theta = \frac{1}{2}$.

Studying the fluid problem, it is initially simpler to consider the Navier-Stokes equation in an Eulerian formulation rather the ALE-formulation Following [4], a general time stepping algorithm for the coupled Navier-Stokes equation can be written as

Problem 2.3.

$$\frac{1}{\Delta}(\mathbf{u}^{n+1} - \mathbf{u}^n) + B(\mathbf{u}^*)\mathbf{u}^{n+\alpha} - \nu\nabla^2 \mathbf{u}^{n+\alpha} = -\nabla p + \mathbf{u}^{n+\alpha}$$
$$\nabla \cdot \mathbf{u}^{n+\alpha} = 0$$

Here $\mathbf{u}^{n+\alpha}$ is an "intermediate" velocity defined by,

$$\mathbf{u}^{n+\alpha} = \alpha \mathbf{u}^{n+1} + (1-\alpha)\mathbf{u}^n \quad \alpha \in [0,1]$$

while $\mathbf{u}^*$ is on the form

$$\mathbf{u}^* = \mathbf{u}^{n+\vartheta} = \begin{cases} \vartheta \mathbf{u}^{n+1} + (1-\vartheta)\mathbf{u}^n & \vartheta \geq 0 \\ \vartheta \mathbf{u}^{n-1} + (1-\vartheta)\mathbf{u}^n & \vartheta \leq 0 \end{cases}$$

At first glance, defining an additional parameter ϑ for the fluid problem seems unessecary. A general mid-point rule by $\alpha = \vartheta = \frac{1}{2}$, a second order scheme in time would easily be acchieved. However, in [4] an additional second order scheme is obtained by choosing e $\alpha = \frac{1}{2}$, $\vartheta = -1$, where $\mathbf{u}^*$ is approximated with an Adam-Bashforth linear method. Making the initial fluid problem linear while maintaining second order convergence is an important result, which have not yet been investigated thorough in litterature of fluid-structure interaction. One reason for this may be that the ALE fluid problem will remain non-linear due to the ALE-mapping.
For the structure problem, the Crank-Nicholson is of main interest due to energy preservation properties and second order convergence.

In light of By letting $\alpha = \vartheta$ $\alpha, \vartheta \in [0,1]$ for the fluid problem, and generalising the consepts in an ALE context, we derive the one-stepl θ scheme found in [6].

Problem 2.4. *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*

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

Deeper analysis in [6], specify to important properties of the one-step *theta* scheme. Firstly, it is unconditionally stable regardless of time step for the interval $\theta = [\frac{1}{2}, 1]$.

2.1.1 Temporal stability

It is known that the Crank-Nicolson scheme can suffer from temporal stability, for long-term simulations [5].
Preliminary work regarding discretization and numerical analysis of Crank-Nicholson time stepping schemes for fluid structure interaction can be found in cite WIck papers. Two main properties of interest of higher-order methods have proven to be the stability of long-time simulation, and obtaining the expected physics for the problem of interest.
The authors of [3], investigated temporal stability of the Crank-Nicolson scheme for the validation benchmark found in [2]. The critera for the numerical experiements was to obatin 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.
Following the ideas of Rick, whricter, a second order scheme based on the Cranck-Nicholson yields two possibilities.

Discretization 2.1. *Crank–Nicolson secant method*

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

Discretization 2.2. *Crank–Nicolson midpoint-tangent method*

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

The numerical experiments showed very similar performance for Discretization 1.1 and 1.2 , and significant differences of temporal accuracy was not found.
Two options to coupe with the presented unstabilities are the *shifted Crank-Nicolson* [3], [6], [5], 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 will be considered.
The shifted Crank-Nicolson scheme introduce further stability to the overall system, by shifting the θ paramter slightly to the implicit side. If the shift is dependent of the time-step size, the scheme will be of second order [3].

2.2 Optimization of Newtonsolver

The expression *bottleneck* esxpress a phenomen 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, 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.

2.3 Consistent methods

2.3.1 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.

2.4 Non-consisten methods

2.4.1 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.

2.4.2 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.

Table 2.1: Comparison of speedup techniques

Implementation	Naive	Reducequad.	Reusejacobi	Combined
Mean time/-timestep	104.5	125.5	48.3	6.8
Speedup	1.0	1.20	0.46	0.06
Mean iteration	4.49	30.59	10.29	10.29

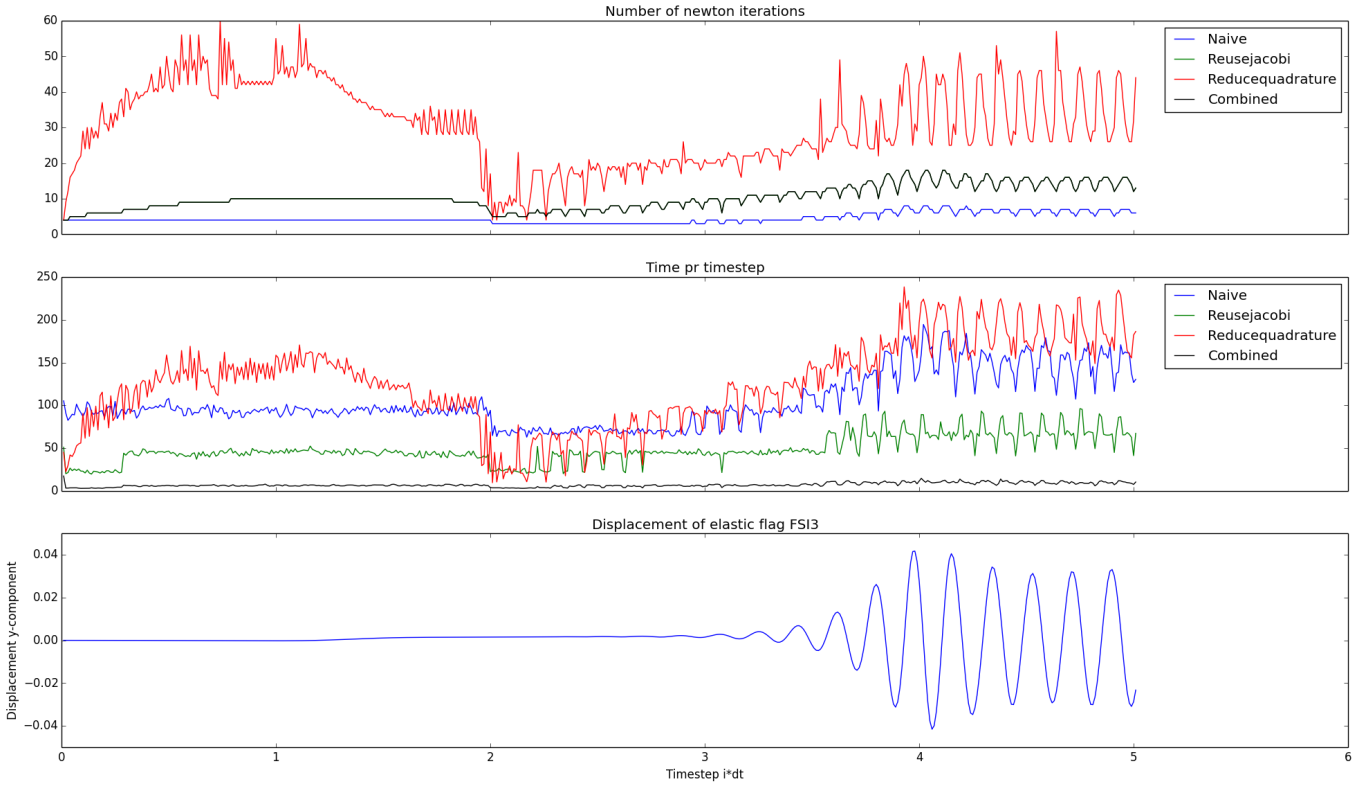

Figure 2.1: Computational domain of the validation benchmark

A brief description will be given for the most central components and technologies used for this thesis.

Bibliography

[1] Philippe Geuzaine. Numerical Simulations of Fluid-Structure Interaction Problems using MpCCI. (1):1–5.

[2] 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.

[3] Thomas Richter and Thomas Wick. On Time Discretizations of Fluid-Structure Interactions. pages 377–400. 2015.

[4] J.C. Simo and F. Armero. Unconditional stability and long-term behavior of transient algorithms for the incompressible Navier-Stokes and Euler equations. *Computer Methods in Applied Mechanics and Engineering*, 111(1-2):111–154, jan 1994.

[5] T. Wick. Stability Estimates and Numerical Comparison of Second Order Time-Stepping Schemes for Fluid-Structure Interactions. In *Numerical Mathematics and Advanced Applications 2011*, pages 625–632. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[6] Thomas Wick. *Adaptive Finite Element Simulation of Fluid-Structure Interaction with Application to Heart-Valve*. PhD thesis, Heidelberg.

[7] Klaus Wolf, Schloss Birlinghoven, Code Coupling Interface, Open Programming Interface, and Distributed Simulation. Mpcci – the Generl Code Coupling Interface. *6. LS-DYNA Anwenderforum, Frankenthal 2007 IT*, pages 1–8, 2007.

[8] Hou Zhang, Xiaoli Zhang, Shanhong Ji, Yanhu Guo, Gustavo Ledezma, Nagi Elabbasi, and Hugues DeCougny. Recent development of fluid-structure interaction capabilities in the ADINA system. *Computers and Structures*, 81(8-11):1071–1085, 2003.