JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
**Daniel Jodlbauer**

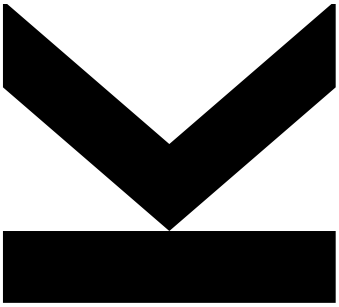
Angefertigt am
**Institut für Numerische Mathematik**

Betreuer
**o. Univ.-Prof. Dipl.-Ing. Dr. Ulrich Langer**

Mitwirkung
**Dr. Thomas Wick**

November 2016

# Robust Preconditioners for Fluid-Structure-Interaction Problems



Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

Industriemathematik

**JOHANNES KEPLER**
**UNIVERSITÄT LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

**Abstract**

Fluid-structure-interaction problems have a wide range of applications, but their efficient solution remains challenging. In this work we provide all details necessary for a monolithic ALE implementation using the finite element library deal.II. Moreover, we show different ways of incorporating the continuity conditions on the interface on the discrete level. To actually solve the arising linear systems, we develop a preconditioner based on an approximate block-wise LU-factorization, splitting the coupled system of equations into its natural fluid, solid and mesh sub-problems. Numerical results illustrate the robust convergence with respect to different material parameters and mesh-size $h$, but with an acceptable dependence on the time-step size $\Delta t$. Furthermore, we observe that this iterative approach outperforms direct solvers even for a low number of degrees of freedoms and without parallelization.

**Zusammenfassung**

Probleme welche das Zusammenspiel von Flüssigkeiten oder Gasen mit Festkörpern beschreiben treten vielfältig auf. Dennoch ist deren Lösung immer noch eine Herausforderung. Wir beschreiben die notwendigen Schritte, um ein Programm zur monolithischen Lösung eben jener Probleme zu erstellen. Dies geschieht mit Hilfe der finiten Elemente Bibliothek deal.II. Insbesondere erörtern wir verschiedene Möglichkeiten, die Interface Bedingungen in die diskrete Formulierung einzuarbeiten. Um die entstehenden Gleichungssysteme zu lösen, entwickeln wir einen Vorkonditionierer basierend auf einer näherungsweisen Block-LU-Zerlegung. Diese spaltet das gekoppelte Problem in ihre natürlichen Bestandteile, den Gleichungen für die Flüssigkeit, den Festkörper sowie für das Diskretisierungsgitter, auf. Numerische Tests zeigen die robuste Konvergenz bezüglich der Wahl der Materialparametern und Gittergröße $h$, jedoch mit einer akzeptablen Abhängigkeit von der Zeitschrittweite $\Delta t$. Auch ohne parallelisierung ist dieser iterative Ansatz deutlich schneller als ein direkter Löser, selbst bei wenigen Freiheitsgraden.

# Contents

# Chapter 1

# Introduction

In this work we will present and analyse robust preconditioners for fluid-structure-interaction (FSI) problems. As the name suggests, these kind of problems consist of two parts, fluid and solid, that influence each other on a common interface.

A typical example of an FSI problem is the air flow around the wing of an aeroplane. This flow is certainly influenced by the shape of the wing, which in turn creates forces acting on it. These further results in a deformation of the wing which again influences the surrounding flow. This example already indicates one of the difficulties of FSI problems: every change to either the fluid or the solid part automatically influences the other part.

FSI problems have also been used for computations in hemodynamics, like blood flow in vessels or the simulation of a whole human heart.

There are basically two different methods of solving such coupled problems:

A very natural attempt follows the description of the wing-flow above: first solve, e.g. the fluid sub-problem, independent of the solid part, then compute the reaction of the solid sub-problem and apply these changes again to the fluid part. If we continues to do so, we may converge to the solution of the whole FSI problem. This technique is called "partitioned" or "iterative" approach and is very efficient if we can guarantee a fast convergence. However, in some applications, it may require a lot of iterations in order to converge or even fail completely to do so. In particular, this is the case if the fluid and solid densities are similar, like for many examples in hemodynamics. More details about this so-called added-mass effect and the partitioned method in general can be found in e.g. [7, 6, 19]).

A different approach which overcomes the added-mass effect is the "monolithic" one. Here,

one attempts to solve the whole coupled FSI problem at once. While this method tends to be more robust, it comes at the prize of solving large and ill-conditioned linear systems. For a small number of degrees of freedom, a direct solver can be used to obtain a solution. However, as the size of the problem increases, direct solvers will require huge amounts of memory and additionally suffer from numerical instabilities. Therefore other solution methods have to be considered.

Since direct solvers are going to fail at some point, we have to consider iterative solvers like GMRES to solve the occurring linear systems. This class of solvers typically require less memory storage and are less susceptible to numerical breakdown. The disadvantage of iterative solvers is, that they require a good preconditioner to converge in a reasonable number of iterations. While "black-box" preconditioners often work well for simple PDEs like the Laplace equation, no such general preconditioner works well enough for the fully coupled fluid-structure-interaction problem. Therefore we are going to have a look at more sophisticated attempts to precondition the occurring linear systems.

In chapter 2, we will give an introduction to commonly used terms and notations and provide a small insight into the mathematics of continuum-mechanics. Chapter 3 summarizes these concepts resulting in the equations for the fluid-structure-interaction problem. The numerical treatment of FSI is described in chapter 4. The main emphasis of this work focuses on the chapters 5 and 6, where we will describe methods for the solution of the arising linear systems. Finally, we will present the numerical results in chapter 7.

# Chapter 2

# Prerequisites

Before we start to give a more mathematical description of fluid-structure-interaction problems, we will require a few preliminaries. Within the next sections we will summarize the notations and common definitions used throughout this work. Furthermore, we will give a short introduction to the required concepts of continuum mechanics, in particular the equations for fluids and elastic materials.

## 2.1 Notation

Although we attempt to explain all terms once they occur, we will give a short list of the notation for completeness.

| Notation | Description |
|---|---|
| $x, \hat{x}$ | variable given in Eulerian and ALE coordinates |
| $v_f, p$ | fluid velocity and pressure |
| $\hat{v}_f, \hat{p}_f$ | in ALE-coordinates |
| $\hat{u}_f$ | fluid (mesh) displacement (ALE) |
| $\hat{u}_s, \hat{v}_s$ | solid displacement and velocity (ALE) |
| $\varphi$ | test-functions |
| $n$ | outer unit-normal vector, time-step |
| $\hat{\rho}_f, \hat{\rho}_s$ | fluid and solid density (ALE) |
| $\nu_f$ | kinematic viscosity |
| $\nu_s$ | Poisson ratio |

| | |
|---|---|
| $\lambda, \mu$ | Lamé-coefficients |
| $\hat{T}$ | transformation mapping |
| $\hat{\mathcal{A}}$ | ALE-mapping |
| $\hat{F}$ | transformation / ALE gradient |
| $\hat{J}$ | transformation / ALE determinant |
| $\Omega_s, \Omega_f$ | solid and fluid domain depending on $t$ |
| $\hat{\Omega}_s, \hat{\Omega}_f$ | solid and fluid reference configuration |
| $\Gamma, \hat{\Gamma}$ | boundary (parts) of a domain |
| $\hat{\Gamma}_I$ | fluid-solid-interface |
| $\mathcal{T} \subset \Omega$ | element |
| $dim$ | spatial dimension |
| $\Delta t$ | time-step size |
| $h := \max_k diam(\mathcal{T}_k)$ | mesh-size |
| $\int dx$ | domain integral |
| $\int ds$ | boundary integral |
| $\partial_x p$ | partial derivatives |
| $dt$ | total time derivative |
| $\nabla p$ | gradient of a (scalar-valued) function $(\partial_x p, \partial_y p, \dots)$ |
| $\nabla v$ | gradient of a (vector-valued) function $(\partial_j v_i)_{i,j=1}^{dim}$ |
| $\mathrm{div}(v)$ | divergence of a vector-valued function $\sum_{i=1}^{dim} \partial_i v_i = \mathrm{tr}(\nabla v)$ |
| $\mathrm{div}(\sigma)$ | divergence of a matrix-valued function $(\mathrm{div}(\sigma_{1i}), \mathrm{div}(\sigma_{2i}), \dots)$ |
| $\triangle p$ | Laplacian of a scalar-valued function $\sum_{i=1}^{dim} \partial_{i^2} p$ |
| $\triangle v$ | Laplacian of a vector-valued function $(\triangle v_1, \triangle v_2, \dots)$ |
| $a(u, v)$ | bilinear form, linear wrt. both arguments |
| $A(u)(v, \varphi)$ | bilinear wrt. $v$ and $\varphi$, non-linear wrt. to $u$ |

## 2.2 Function-Spaces

As usual, we denote by $L^2(\Omega)$ the set of square-integrable functions with scalar-product $(f, g)_\Omega := \int_\Omega f(x)g(x)dx$. By $\langle x, y \rangle_\Gamma$, we indicate the $L^2$-scalar product over some part of the boundary. For matrix-valued functions, the corresponding integrals are interpreted as $\int_\Omega \sum_{i,j} A_{ij}B_{ij}dx$. Further, $H^1(\Omega)$ denotes the usual Sobolev space consisting of $L^2$-functions

with derivative in $L^2$ as well [1]. The corresponding vector-valued function spaces are denoted by $L^2(\Omega)^d = L^2(\Omega) \times \ldots \times L^2(\Omega)$, and analogously for all other spaces.

For the discretization, we will use continuous, piecewise polynomial functions, precisely

$$Q(k) := \left\{ \sum_{\substack{\alpha \\ \max(\alpha_i) \leq k}} c_{\alpha_i} x_i^{\alpha_i},\ c_{\alpha_i} \in \mathbb{R},\ \alpha \in \mathbb{N}_0^{dim} \right\} \tag{2.1}$$

on each element. By DGQ we denote the respective discontinuous versions, i.e. there are no common degrees of freedom between neighbouring cells (see figure 2.1).

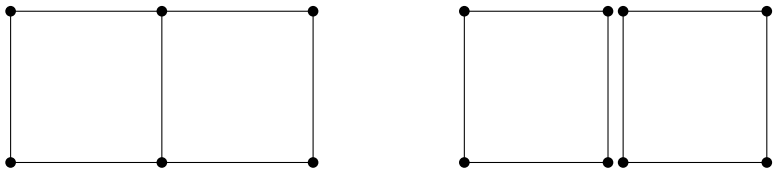


Figure 2.1: Degrees of freedom for continuous Q(1) and discontinuous DGQ(1) elements.

## 2.3 Continuum Mechanics

As of today's knowledge, all kind of objects are basically just a collection of atoms. However, such a discrete model is very unhandy from a mathematical viewpoint. Therefore, continuum mechanics seeks to approximate this discrete objects as continuous ones which occupy a whole region. For macroscopic problems, this approach turns out to be very accurate and easier to handle. In the next sections we will learn about the basic techniques used to describe the behaviour of solids and fluids. Since at the very end, we are mainly interested in solving the resulting equations, we will not focus too much on the details. These may found in e.g. [16, 17, 31, 2, 25].

First of all, we give a rough definition of continuous objects.

**Definition 2.3.1** (Object)**.** An object is an open, connected subset of $\mathbb{R}^d$ with sufficiently smooth boundary (e.g. Lipschitz-continuous).

The reference configuration denotes a state of the object that is independent of time. A usual choice is the object itself in a stress-free state, i.e. where no external forces or deformations occur, or the object at time zero.

One particular point of interest in continuum mechanics is the question how objects behave if external forces are applied. Typically, it will result in a mixture of movement and deformation, which can both be described in terms of the deformation mapping.

**Definition 2.3.2** (Deformation mapping)**.** Let $\hat{\Omega}$ be an object in reference configuration. Then the deformed object $\Omega(t)$ at time $t$ is given by the transformation

$$\begin{aligned} &\hat{T} : \mathbb{R}_+ \times \hat{\Omega} \to \Omega(t) \\ &x(t) = \hat{T}(t, \hat{x}) \\ &\Omega(t) = \hat{T}(t, \hat{\Omega}) \end{aligned} \tag{2.2}$$

Furthermore, we assume that the deformation mapping $\hat{T}(t, \hat{x})$ is

1. continuously differentiable and
2. bijective for all times $t \geq 0$.

This deformation naturally introduces two different kind of coordinates.

First, we have the coordinates belonging to the reference configuration. A point $\hat{x}$ in this configuration always refers to same material point. If we think of discrete objects again, this means that a coordinate in reference configuration always points to the same atom. This type is called reference or Lagrangian coordinates.

The second way of describing points in space are Eulerian coordinates. A point $x$ in this configuration refers to a fixed point in space, independent of which "atom" currently occupies this spot.

Eulerian coordinates are mostly used in fluid dynamics, while solids are usually described in Lagrangian coordinates. For fluid-structure-interaction problems, we need to couple these two different coordinate systems. One way of doing so is to formulate all equations in a common coordinate system, which will be done in section 3.1.

**Remark 2.3.3** (Notation)**.** In this work we will use hats to denote reference coordinates. Further, we omit the time-dependence and use $\Omega := \Omega(t)$ for simplicity.

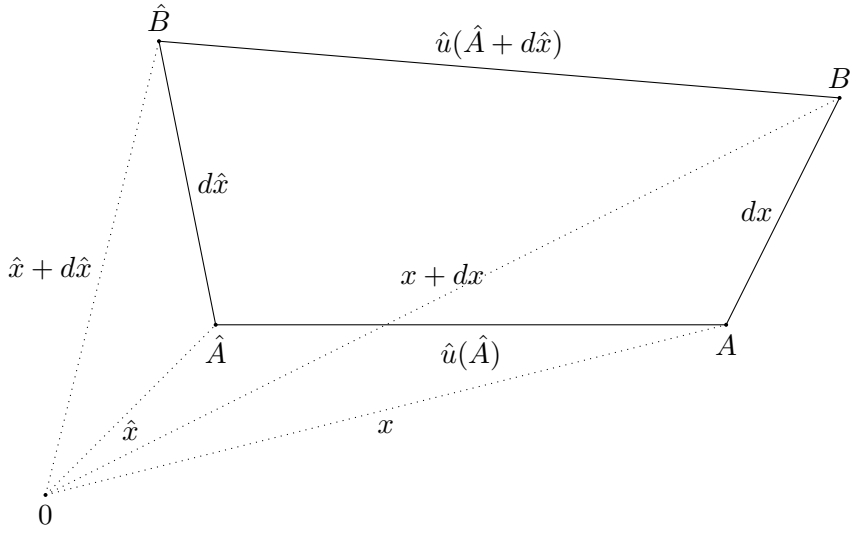



Figure 2.2: Displacements.

The deformation above may also be represented in terms of the displacement:

**Definition 2.3.4** (Displacement)**.** The displacement field associated to a deformation is given by

$$\hat{u}(t, \hat{x}) := \hat{T}(t, \hat{x}) - \hat{x} = x(t, \hat{x}) - \hat{x}$$

in Lagrangian coordinates or equivalently

$$u(x, t) = x - \hat{T}^{-1}(t, x) = x - \hat{x}(t, x)$$

in Eulerian coordinates.

Other frequently required quantities are the transformation gradient and its determinant.

**Definition 2.3.5.** The transformation gradient $\hat{F}$ is given by

$$\hat{F}(t,\hat{x}) := \hat{\nabla}\hat{T}(t,\hat{x}) = I + \hat{\nabla}\hat{u}(t,\hat{x})$$

with determinant

$$\hat{J}(t,\hat{x}) := \det(\hat{F}(t,\hat{x})).$$

In addition to the assumption given in definition 2.3.2 we further assume that the determinant is orientation preserving, i.e. $\hat{J} > 0$ for all $\hat{x} \in \hat{\Omega}$ and times $t$. This ensures the invertibility of $\hat{F}$.

**Geometrical Interpretation** The transformation gradient converts the relative positions between two "atoms" from one coordinate system into the other. This can be seen as follows: with the definition $d\hat{u} := \hat{u}(\hat{A}+d\hat{x}) - \hat{u}(\hat{A})$, we obtain from figure 2.2 that $dx = d\hat{x} + d\hat{u}$. Using a first-order Taylor-series expansion around $\hat{A}$, we further get $\hat{u}(\hat{A}+d\hat{x}) = \hat{u}(\hat{A}) + \hat{\nabla}\hat{u}(\hat{A}) \cdot d\hat{x}$ and therefore $dx = d\hat{x} + \hat{\nabla}\hat{u} \cdot d\hat{x} = \hat{F} \cdot d\hat{x}$.

The transformation determinant $\hat{J}$ denotes the change of the area (2d) or volume (3d) at a given point. A negative determinant would correspond to an object intersecting itself, which we do not allow by the above assumption.

### 2.3.1 Transformations

The following lemmas follow directly from the definition of the transformation and show how to switch between Eulerian and Lagrangian coordinates. Mostly, these are just applications of the usual chain-rule and integration by substitution, which can be found in the standard literature. Remember that we have the definition $x(t) = \hat{T}(t,\hat{x})$.

**Definition 2.3.6** (Function Values)**.** For arbitrary functions $f$ we identify function values between Eulerian and Lagrangian coordinates as follows:

$$f(t,x) = f(t,\hat{T}(t,\hat{x})) = \hat{f}(t,\hat{x}).$$

**Lemma 2.3.7** (Function Gradients)**.** Using the chain rule, we obtain for vector-valued functions

$$\hat{\nabla} f(t,x) = \hat{\nabla} f(t,\hat{T}(t,\hat{x})) = \nabla f(t,x) \cdot \hat{F}(t,\hat{x})$$

or equivalently

$$\nabla f(t,x) = \hat{\nabla} f(t,x) \cdot \hat{F}^{-1}(t,\hat{x}).$$

**Lemma 2.3.8** (Time Derivatives)**.** Similar to the lemma above, the total time derivative in Lagrangian coordinates reads

$$\hat{\mathrm{D}}_t f(t,x) = \partial_t f(t,x) + \nabla f(t,x) \cdot \partial_t \hat{T}(t,\hat{x}).$$

For the weak formulation, it is also important to transform integrals over the domain and boundary parts.

**Lemma 2.3.9** (Transformation of Integrals)**.** For the domain transformations, we have the well known substitution rule

$$\int_{\Omega} f(x)dx = \int_{\hat{\Omega}} \hat{J}(\hat{x})\hat{f}(\hat{x})d\hat{x}$$

for continuous functions $f$.

Nanson's formulae gives us the corresponding transformation of boundary integrals of tensor fields $\sigma$:

$$\int_{\partial\Omega} \sigma n ds = \int_{\partial\hat{\Omega}} \hat{J}\hat{\sigma}\hat{F}^{-T}\hat{n}d\hat{s}.$$

Closely related to Nanson's formulae is the Piola transform. It allows us to convert Eulerian stress tensors to its Lagrangian counterparts.

**Definition 2.3.10** (Piola Transform)**.** For a deformation $\hat{T}$ and a tensor field $\sigma$ defined in Eulerian coordinates, the Piola transform $\hat{\sigma}$ of $\sigma$ is given as follows:

$$\hat{\sigma}(t,\hat{x}) := \hat{J}(t,\hat{x})\sigma(t,x)\hat{F}^{-T}(t,\hat{x})$$

For the divergence of such a tensor, the following statement holds:

**Theorem 2.3.11** (Divergence of the Piola transform)**.**

$$div_R(\hat{\sigma}(t,\hat{x})) = \hat{J}(t,\hat{x})\hat{F}(t,\hat{x})\, div(\sigma(t,x)).$$

Special care has to be taken if densities are involved:

**Theorem 2.3.12** (Densities)**.** *For a mass-conserving object, we have*

$$\rho(x) = \hat{\rho}(\hat{x})\hat{J}(\hat{x})$$

*Proof.* The mass of some part $\hat{B}$ of object $\hat{\Omega}$ can be written as

$$m_B(t) := \int_B \rho(x)dx$$

and

$$\hat{m}_{\hat{B}}(t) := \int_{\hat{B}} \hat{\rho}(\hat{x})d\hat{x}$$

in reference coordinates. Since the mass of these subdomains is preserved under deformation, we have $m_B(t) = \hat{m}_{\hat{B}}(t)$.

However, by integral transformation we obtain

$$\int_{\hat{B}} \hat{\rho}(\hat{x})d\hat{x} = m_B(t) = \int_B \rho(x)dx = \int_{\hat{B}} \hat{J}\rho(\hat{x})d\hat{x}.$$

for arbitrary parts $\hat{B} \subseteq \hat{\Omega}$ and therefore $\rho(x) = \hat{\rho}(\hat{x})\hat{J}(\hat{x})$ holds. □

The following theorem allows to differentiate integrals over time-dependent regions, like those introduced by transformations.

**Theorem 2.3.13** (Reynold's Transport Theorem)**.** *Let $\hat{T}$ be a $C^2-$ transformation, $\hat{B} \subseteq \hat{\Omega}$. For a $C^1$ scalar-valued Eulerian function $f$, the following identity holds:*

$$\frac{d}{dt}\int_{\hat{T}(t,\hat{B})} f(t,x)dx = \int_{\hat{T}(t,\hat{B})} \partial_t f(t,x) + div(f(t,x)\ w(t,x))dx, \tag{2.3}$$

*with the flow-velocity $w := \partial_t \hat{T}(t,\hat{x})$.*

## 2.4 Modelling Objects

Now that we have completed most of the preliminaries, we can have a look at the equations defining the behaviour of objects. For FSI we need to know, as the name already indicates, how to model fluids and solid materials. This will be addressed in the following chapters. Note that for the moment we only pose the equations without boundary or initial conditions. These will be added later on for the full FSI problem.

### 2.4.1 Fluids

The equations for the fluid velocity $v_f$ and pressure $p$ are given by the Navier-Stokes equations, which are usually formulated in Eulerian coordinates. In this work, we restrict ourselves to incompressible fluids, which leads to the following system of equations:

$$\rho_f \partial_t v_f + \rho_f \nabla v_f \cdot v_f - \operatorname{div}(\sigma_f) = 0 \tag{2.4}$$

$$\rho_f \operatorname{div}(v_f) = 0 \tag{2.5}$$

with the fluid stress tensor given by

$$\sigma_f = -pI + \rho_f \nu_f (\nabla v_f + \nabla v_f^T). \tag{2.6}$$

Similar to the solid case, equation equation (2.4) stems from the balance of forces and momentum. The main difference here is the convection or transport term $\nabla v_f \cdot v_f$. This non-linear part describes the acceleration of fluid particles caused by the motion of the fluid itself. This may happen because of a change in the geometry, i.e. the velocity increases, if the surrounding channel narrows down. For a thorough derivation of these equations we refer to [25].

Equation (2.5) is a special case of the continuity equation. Using Reynold's transport theorem and the conservation of mass we get

$$0 = \frac{d}{dt} m(t) = \frac{d}{dt} \int_\Omega \rho_f(t,x) dx = \int_\Omega \frac{d}{dt} \rho_f(t,x) + \operatorname{div}(\rho_f \ v_f) dx.$$

Since this relation holds for all parts $B \subset \Omega$ as well, we get

$$\frac{d}{dt} \rho_f(t,x) + \operatorname{div}(\rho \ v_f) = 0.$$

For an incompressible fluid ($\frac{d}{dt}\rho_f = 0$) with constant density, this simplifies to $\operatorname{div}(v_f) = 0$.

**Assumption 2.4.1.** From now on, we will assume that the fluid is constant with respect to space and time

$$\rho_f(t,x) := \rho_f.$$

### 2.4.2 Elastic Materials

The key quantity of a solid problem is the displacement $\hat{u}_s$. Its equation is given in (2.7) and can be derived from Newton's second law (conservation of momentum). More details on the derivation and physical interpretation of the quantities involved can be found in [16].

$$\hat{\rho}_s \partial_t^2 \hat{u}_s - \operatorname{div}_R(\hat{F}\hat{\Sigma}) = 0. \tag{2.7}$$

The so-called stress tensor $\hat{\Sigma}$ is given by a constitutive law, which describes the stress-strain relation of a material. Throughout this work, we use the Saint-Venant Kirchhoff material model (STVK), which is given by

$$\hat{\Sigma}(\hat{E}) = 2\mu\hat{E} + \lambda \operatorname{tr}(\hat{E})\hat{I} \tag{2.8}$$

with constant Lamé parameters $\lambda$ and $\mu$. The non-linear Right-Cauchy-Green strain tensor is given by

$$\hat{E} = \frac{1}{2}(\hat{\nabla}\hat{u} + \hat{\nabla}\hat{u}^T + \hat{\nabla}\hat{u}\hat{\nabla}\hat{u}^T). \tag{2.9}$$

Strain measures the deformation of an object. Opposed to the deformation gradient $\hat{F}$, it is related to the "real" deformation of an object independent of translations and rotations.

Roughly speaking, stress determines the internal surface forces (pressure) caused by the interaction of neighbouring atoms.

The second-order in time equation (2.7) may also be written as a system of first order equations

$$\begin{aligned} \hat{\rho}_s \partial_t \hat{v}_s - \operatorname{div}_R(\hat{F}\hat{\Sigma}) &= 0 \\ \hat{\rho}_s \left(\partial_t \hat{u}_s - \hat{v}_f\right) &= 0 \end{aligned} \tag{2.10}$$

with the new solid velocity $\hat{v}_s = \partial_t \hat{u}_s$.

# Chapter 3

# FSI Equations

The two main ingredients, fluid and solid, have already been described in the previous section. However, we currently have the solid equations given in Lagrangian coordinates while the fluid system is described in Eulerian ones. This is problematic since we want to couple these equations on a common interface. One way of doing so is to convert both equations into a common coordinate system, the ALE - coordinates [9].

## 3.1 Arbitrary-Lagrangian-Eulerian (ALE) Coordinates

From the last sections we know, that in the Lagrangian description of the elasticity equations the computational domain remains fixed and any occurring deformation is represented through the mapping $\hat{T}(t,\hat{x}) = \hat{x} + \hat{u}(t,\hat{x})$. The idea behind ALE is to extend this mapping from the solid into the fluid domain by introducing the fluid-displacement (mesh-motion) $\hat{u}_f$ and ALE-mapping $\hat{\mathcal{A}}$:

$$\hat{u}(t,\hat{x}) := \begin{cases} \hat{u}_s(t,\hat{x}) & \hat{x} \in \hat{\Omega}_s \\ \hat{u}_f(t,\hat{x}) & \hat{x} \in \hat{\Omega}_f \end{cases} \qquad \text{and} \qquad \hat{\mathcal{A}}(t,\hat{x}) := \hat{x} + \hat{u}(t,\hat{x}) \text{ for } \hat{x} \in \hat{\Omega}. \tag{3.1}$$

We can interpret the fluid domain as an artificial structure subject to deformations given by $\hat{u}_f$. After discretization, this would correspond to moving the vertices of the mesh, which motivates the name "mesh-motion".

**Remark 3.1.1** (Clarification)**.** In this work, we are following the ALEfx approach. This means, that during the computation, the mesh is not moved. This may happen only for

visualization purposes to actually see the solid and mesh deformations. A different method would be ALEdm, where the mesh is indeed updated after every time-step by post-processing.

We still have to clarify how exactly we are going to extend the solid displacement. With the interpretation above, a natural choice is to do something like

$$-\mathrm{div}_R(\hat{\sigma}_{MM}(t,\hat{u}_f)) = 0 \qquad \text{in } \hat{\Omega} \tag{3.2}$$

$$\hat{u}_f = \hat{u}_s \qquad \text{on } \hat{\Gamma}_I \tag{3.3}$$

$$\hat{u}_f = 0 \qquad \text{on } \partial\hat{\Omega}_f \backslash \hat{\Gamma}_I \tag{3.4}$$

with an appropriate "mesh-motion stress-tensor" $\hat{\sigma}_{MM}$. The simplest choice is $\hat{\sigma}_{MM} := \hat{\nabla}\hat{u}_f$, which leads to the Laplace equation. However, if the displacements become too large, this harmonic extension will lead to degenerate meshes with $\hat{J} \approx 0$. The situation can be improved by choosing $\hat{\sigma}_{MM} := \frac{1}{\hat{J}}\hat{\nabla}\hat{u}_f$ leading to a non-linear-harmonic mesh-motion equation. For more possible extensions and a comparison of these we refer to [30].

With this uniform description of the coordinates at hand we are now able to define the whole FSI system. Note that due to the definition of the ALE mapping, we can use the equations for the elasticity problem without change - only the fluid equations have to be adjusted accordingly.

**Remark 3.1.2** (Different Strategies)**.** Several other methods than ALE exist, like fictitious domain approach, immersed boundary methods and others.

## 3.2 ALE Description of the Fluid Equations

Since ALE coordinates are just a global version of the Lagrangian ones, all the definitions and theorems from section 2.3.1 also apply for the new ALE mapping. In this section, we will use these results to obtain the fluid equations in ALE coordinates.

Summarizing the statements from section 2.3.1 we have the following transformation rules:

- $\rho_f \mapsto \hat{J}\hat{\rho}_f$
- $v_f \mapsto \hat{v}_f$
- $\nabla v_f \mapsto \hat{\nabla}\hat{v}_f\hat{F}^{-1}$

- $\partial_t v_f \mapsto \hat{\partial}_t \hat{v}_f - \hat{\nabla}\hat{v}_f \hat{F}^{-1} \cdot \partial_t \hat{\mathcal{A}}$
- $\mathrm{div}(\sigma_f) \mapsto \mathrm{div}_R(\hat{J}\hat{\sigma}_f \hat{F}^{-T})$.

For the incompressibility equation, we further use the identity $\mathrm{div}(v_f) = \mathrm{tr}(\nabla v_f)$.

After applying these rules to the fluid equations given in section 2.4.1 we get:

$$\begin{aligned} \hat{J}\hat{\rho}_f \hat{\partial}_t \hat{v}_f + \hat{J}\hat{\rho}_f \hat{\nabla}\hat{v}_f \hat{F}^{-1} \cdot (\hat{v}_f - \partial_t \hat{\mathcal{A}}) - \mathrm{div}_R(\hat{J}\hat{\sigma}_f \hat{F}^{-T}) &= 0 \\ \hat{J}\hat{\rho}\ \mathrm{tr}(\hat{\nabla}\hat{v}_f \hat{F}^{-1}) &= 0, \end{aligned} \tag{3.5}$$

with the fluid stress-tensor in ALE coordinates given as

$$\hat{\sigma}_f := -\hat{p}_f I + \hat{\rho}_f \hat{\nu}_f (\hat{\nabla}\hat{v}_f \hat{F}^{-1} + \hat{F}^{-T}\hat{\nabla}\hat{v}_f^T).$$

## 3.3 Boundary and Initial Conditions

Until now, have successfully avoided to give any boundary or initial conditions, which will be made up now. Since the precise definition of these conditions is problem-specific, we use the geometry and definition of the FSI benchmark given in section 7.1 as an example. Most FSI problems fulfil similar conditions.

**Fluid** In many FSI applications, the system is driven by the flow of some fluid. Therefore we pose some given velocity inflow profile on $\hat{\Gamma}_{in}$:

$$\hat{v}_f(t, \hat{x}) := \hat{v}_{in}(t, \hat{x}) \text{ on } \hat{\Gamma}_{in},$$

This also includes the initial condition for the velocity by setting $t := 0$. For the remaining boundary parts $\hat{\Gamma}_{top}, \hat{\Gamma}_{bottom}$ as well as $\hat{\Gamma}_c$ we use the so-called "no-slip" condition $\hat{v}_f = 0$. The physical motivation behind this condition is, that fluid particles are stuck to the atoms of the boundary and therefore do no longer move (adhesion).

Since we can not compute solutions on arbitrary large domains, we sometimes have to truncate the domain, i.e. a very long channel is represented by a small part of it. However, this leads

to question of what kind boundary conditions we should pose on the cut. One widely used setting is the so-called "do-nothing" condition

$$\hat{J}(-\hat{p}_f I + \hat{\rho}_f \hat{\nu}_f \hat{\nabla} \hat{v}_f \hat{F}^{-1}) \hat{F}^{-T} = 0 \text{ on } \hat{\Gamma}_{out}. \tag{3.6}$$

**Solid** Since the driving force in our tests will be the fluid, we use homogeneous initial conditions for the structure displacement and velocity. Furthermore, we have to attach the flag onto the cylinder, which gives rise to the boundary condition

$$\hat{u}_s = \hat{v}_s = 0 \text{ on } \hat{\Gamma}_{cf}.$$

**Mesh** As written in equation (3.2), we restrict the mesh-displacement on the whole boundary $\hat{\Gamma}_f$, which includes the boundary of the cylinder $\hat{\Gamma}_c$. However, we could allow movement of the mesh-vertices along the boundary parts for more flexibility, i.e. only $\hat{u}_f^y = 0$ on $\hat{\Gamma}_{top} \cup \hat{\Gamma}_{bottom}$.

## 3.4 Interface Conditions

To realize the interaction between fluid and solid, we have to couple them along the interface. The geometric coupling has already been discussed in the introduction of the ALE mapping:

$$\hat{u}_f = \hat{u}_s \text{ on } \hat{\Gamma}_I, \tag{3.7}$$

although, we introduced it as a boundary condition for the mesh-motion equation. If we formulate the whole FSI problem using this condition, it may also be interpreted as a condition for the solid equations. This, however, is not intended, as we only want to have coupling from the solid to the mesh equations, but not vice-versa. For monolithic approaches, such a one-sided coupling is achieved by scaling the mesh-motion part by a small factor, e.g. $\alpha_u = 10^{-5}$.

Similar to boundary conditions from before, we also have the "no-slip" condition between

the solid domain and the surrounding fluid, leading to

$$\hat{v}_f = \hat{v}_s \text{ on } \hat{\Gamma}_I. \tag{3.8}$$

The most important condition is the continuity of stresses, which propagates forces from the fluid to the solid object and vice-versa. In ALE coordinates, this conditions reads

$$\hat{J}\hat{\sigma}_f\hat{F}^{-T}\hat{n}_f + \hat{F}\hat{\Sigma}_s\hat{n}_s = 0. \tag{3.9}$$

## 3.5 Summary

Summarizing the previous results, we obtain the FSI-ALE system in strong formulation.

$$\begin{aligned}
\hat{J}\hat{\rho}_f\hat{\partial}_t\hat{v}_f + \hat{J}\hat{\rho}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1}\cdot(\hat{v}_f - \partial_t\hat{\mathcal{A}}) - \mathrm{div}_R(\hat{J}\hat{\sigma}_f\hat{F}^{-T}) &= 0 \\
\hat{J}\hat{\rho}\ \mathrm{tr}(\hat{\nabla}\hat{v}_f\hat{F}^{-1}) &= 0 \\
\hat{\rho}_s\partial_t\hat{v}_s - \mathrm{div}_R(\hat{F}\hat{\Sigma}) &= 0 \\
\hat{\rho}_s\left(\partial_t\hat{u}_s - \hat{v}_f\right) &= 0 \\
-\alpha_u\mathrm{div}_R(\hat{\sigma}_{MM}) &= 0
\end{aligned} \tag{3.10}$$

with the additional conditions

$$\hat{J}\hat{\sigma}_f\hat{F}^{-T}\hat{n}_f + \hat{F}\hat{\Sigma}_s = 0 \text{ on } \hat{\Gamma}_I \qquad \hat{v}_f = 0 \text{ on } \hat{\Gamma}_{top}\cup\hat{\Gamma}_{bottom}\cup\hat{\Gamma}_c \tag{3.11}$$

$$\hat{v}_f - \hat{v}_s = 0 \text{ on } \hat{\Gamma}_I \qquad \hat{v}_f = g \text{ on } \hat{\Gamma}_{in} \tag{3.12}$$

$$\hat{u}_f - \hat{u}_s = 0 \text{ on } \hat{\Gamma}_I \qquad \hat{u}_f = 0 \text{ on } \hat{\Gamma}_{top}\cup\hat{\Gamma}_{bottom}\cup\hat{\Gamma}_{in}\cup\hat{\Gamma}_{out}\cup\hat{\Gamma}_c \tag{3.13}$$

$$\hat{u}_s = 0 \text{ on } \hat{\Gamma}_{cf} \tag{3.14}$$

$$\hat{v}_s = 0 \text{ on } \hat{\Gamma}_{cf} \tag{3.15}$$

and

$$\hat{J}(-\hat{p}_f I + \hat{\rho}_f\hat{\nu}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1})\hat{F}^{-T} = 0 \text{ on } \hat{\Gamma}_{out}. \tag{3.16}$$

# Chapter 4

# Discretization

For the discretization of the fluid-structure-interaction equations, we have to consider several things. As usual, we require a variational formulation, but additionally, we have to take care of the time-derivatives and the occurring non-linearities. Although we we formally apply time-discretization first it is easier to present it in the opposed direction.

## 4.1 Weak Formulation

For the numerical treatment of partial differential equations, we need a variational formulation. As usual, we first multiply each equation with a corresponding test-functions, integrate over the respective reference domain and perform integration by parts. The following paragraphs show the results of this procedure for the solid, fluid and mesh-motion equations.

Special care has to be taken on the interface when defining the test and ansatz-functions. For all time-steps $n$, we have

$$(\hat{u}_s, \hat{u}_f) \in V_u := \{\hat{u}_s \in H^1(\hat{\Omega}_s)^{dim}, \hat{u}_f \in H^1(\hat{\Omega}_f)^{dim} :$$
$$\hat{u}_s = 0 \text{ on } \partial\hat{\Omega}_s \backslash \hat{\Gamma}_I,$$
$$\hat{u}_f = 0 \text{ on } \partial\hat{\Omega}_f \backslash \hat{\Gamma}_I,$$
$$\hat{u}_s = \hat{u}_f \text{ on } \hat{\Gamma}_I\},$$

$$(\hat{v}_s, \hat{v}_f) \in V_v := \{\hat{v}_s \in H^1(\hat{\Omega}_s)^{dim}, \hat{v}_f \in H^1(\hat{\Omega}_f)^{dim} :$$
$$\hat{v}_s = 0 \text{ on } \partial\hat{\Omega}_s \backslash \hat{\Gamma}_I,$$
$$\hat{v}_f = 0 \text{ on } \partial\hat{\Omega}_f \backslash (\hat{\Gamma}_I \cup \hat{\Gamma}_{in}),$$
$$\hat{v}_f = g(t) \text{ on } \hat{\Gamma}_{in},$$
$$\hat{v}_s = \hat{v}_f \text{ on } \hat{\Gamma}_I\}$$

and

$$\hat{p}_f \in V_p := L^2(\hat{\Omega}_f).$$

The test-spaces for $(\hat{\varphi}_s^u, \hat{\varphi}_f^u)$ and $\hat{\varphi}_f^p$ are the same, only for $(\hat{\varphi}_s^v, \hat{\varphi}_f^v)$ we get the homogeneous version $(\hat{\varphi}_s^v, \hat{\varphi}_f^v) \in V_v^0$, with the velocity test-space

$$V_v^0 := V_v \text{ with } g(t) \equiv 0. \tag{4.1}$$

After the usual steps, we arrive at the following weak formulation for the FSI problem:

### 4.1.1 Fluid

$$\begin{aligned}\left(\hat{J}\hat{\rho}_f\hat{\partial}_t\hat{v}_f \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\rho}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1}\cdot(\hat{v}_f - \hat{w}) \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\sigma}_f\hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f} \\ - \left\langle \hat{J}\hat{\sigma}_f\hat{F}^{-T}\cdot\hat{n}_f \ , \ \hat{\varphi}_f^v\right\rangle_{\hat{\Gamma}_I} - \left\langle \hat{\rho}_f\hat{\nu}_f\hat{J}\hat{F}^{-T}\hat{\nabla}\hat{v}_f^T\hat{F}^{-T}\cdot\hat{n}_f \ , \hat{\varphi}_f^v\right\rangle_{\hat{\Gamma}_{out}} = 0 \\ \left(\hat{J}\ \mathrm{tr}(\hat{\nabla}\hat{v}_f\hat{F}^{-1}) \ , \ \hat{\varphi}_f^p\right)_{\hat{\Omega}_f} = 0.\end{aligned} \tag{4.2}$$

### 4.1.2 Solid

$$\begin{aligned}\left(\hat{\rho}_s\hat{\partial}_t\hat{v}_s \ , \ \hat{\varphi}_s^v\right)_{\hat{\Omega}_s} + \left(\hat{F}\hat{\Sigma}_s \ , \ \hat{\nabla}\hat{\varphi}_s^v\right)_{\hat{\Omega}_s} - \left\langle \hat{F}\hat{\Sigma}_s\cdot\hat{n}_s \ , \ \hat{\varphi}_s^v\right\rangle_{\hat{\Gamma}_I} = 0 \\ \left(\hat{\partial}_t\hat{u}_s - \hat{v}_s \ , \ \hat{\varphi}_s^u\right)_{\hat{\Omega}_s} = 0\end{aligned} \tag{4.3}$$

### 4.1.3 Mesh-Motion

$$\alpha_u \left( \frac{1}{\hat{J}} \hat{\nabla} \hat{u}_f \ , \ \hat{\nabla} \hat{\varphi}_f^u \right)_{\hat{\Omega}_f} = 0. \tag{4.4}$$

### 4.1.4 Interface

Since the test-functions $\hat{\varphi}_f^v$ and $\hat{\varphi}_s^v$ are equal on the interface, we can write the continuity of stresses (3.9) as:

$$\left\langle \hat{J} \hat{\sigma}_f \hat{F}^{-T} \hat{n}_f \ , \ \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_I} + \left\langle \hat{F} \hat{\Sigma}_s \hat{n}_s \ , \ \hat{\varphi}_s^v \right\rangle_{\hat{\Gamma}_I} = 0. \tag{4.5}$$

These terms already appear in equations (4.2) and (4.3). Hence we can incorporate condition (3.9) by neglecting these interface terms in the fluid and solid equations.

**Remark 4.1.1** (Do-Nothing Condition)**.** Without the do-nothing condition, we still get contributions from the term $\left( \hat{J} \hat{\sigma}_f \hat{F}^{-T} \ , \ \hat{\nabla} \hat{\varphi}_f^v \right)_{\hat{\Omega}_f}$ on the outflow boundary. Since $\sigma_f \approx -pI + \nabla v_f + \nabla v_f^T$, this term coincides with the do-nothing condition if we remove the transposed part. This leads to

$$\left\langle \hat{\rho}_f \hat{\nu}_f \hat{J} \hat{F}^{-T} \hat{\nabla} \hat{v}_f^T \hat{F}^{-T} \cdot \hat{n}_f \ , \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_{out}}$$

within the fluid equations (4.2).

### 4.1.5 FSI

Summarizing the last sections, we finally get the weak formulation for the full FSI problem:

**Problem 4.1.2** (Weak formulation of FSI-ALE)**.** Find $((\hat{u}_s, \hat{u}_f), (\hat{v}_s, \hat{v}_f), \hat{p}_f)$ in $V_u \times V_v \times V_p$ such that for all test-functions $\left( (\hat{\varphi}_s^u, \hat{\varphi}_f^u), (\hat{\varphi}_s^v, \hat{\varphi}_f^v), \hat{\varphi}_f^p \right)$ in $V_u \times V_v^0 \times V_p$ the following equations

are satisfied for almost all times $t$:

$$\begin{aligned}
\left(\hat{\rho}_s\hat{\partial}_t\hat{v}_s\ ,\ \hat{\varphi}_s^v\right)_{\hat{\Omega}_s} + \left(\hat{F}\hat{\Sigma}_s\ ,\ \hat{\nabla}\hat{\varphi}_s^v\right)_{\hat{\Omega}_s} &= 0 \\
\left(\hat{\partial}_t\hat{u}_s - \hat{v}_s\ ,\ \hat{\varphi}_s^u\right)_{\hat{\Omega}_s} &= 0 \\
\left(\hat{J}\hat{\rho}_f\hat{\partial}_t\hat{v}_f\ ,\ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\rho}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1}\cdot(\hat{v}_f-\hat{w})\ ,\ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{F}^{-T}\hat{\sigma}_f\ ,\ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f} & \\
- \left\langle\hat{\rho}_f\hat{\nu}_f\hat{J}\hat{F}^{-T}\hat{\nabla}\hat{v}_f^T\hat{F}^{-T}\cdot\hat{n}_f\ ,\hat{\varphi}_f^v\right\rangle_{\hat{\Gamma}_{out}} &= 0 \\
\left(\hat{J}\ \mathrm{tr}(\hat{\nabla}\hat{v}_f\hat{F}^{-1})\ ,\ \hat{\varphi}_f^p\right)_{\hat{\Omega}_f} &= 0 \\
\alpha_u\left(\frac{1}{\hat{J}}\hat{\nabla}\hat{u}_f\ ,\ \hat{\nabla}\hat{\varphi}_f^u\right)_{\hat{\Omega}_f} &= 0
\end{aligned} \tag{4.6}$$

**Remark 4.1.3.** Instead of converting the strong fluid equations from the Eulerian system into ALE coordinates and then deriving the weak formulation, we could have also gone the other way: first compute the weak Eulerian formulation and then apply the ALE transformations. If we use this approach, we must not forget to transform the test-functions into ALE coordinates as well. With the simple identity $(A\cdot B):C = A:(C\cdot B^T)$ for square-matrices, we arrive at the same result.

## 4.2 Discretization in Time

In the next step, we want to get rid of the time-derivatives within the FSI equations. This is achieved using the one-step-theta scheme given below.

**Definition 4.2.1** (One-Step-Theta Scheme)**.** Given a differential equation $a(u)\partial_t u + A(u) = 0$, the one-step-theta scheme reads

$$\big(\theta\ a(u^n) + (1-\theta)\ a(u^{n-1})\ (u(t^n) - u(t^{n-1})\big) - \Delta t\ \theta\ A(u^n) - \Delta t\ (1-\theta)\ A(u^{n-1}) = 0$$

with $\theta\in[0,1]$ and $\Delta t := t^n - t^{n-1}$.

Different values of $\theta$ result in time-stepping schemes with different properties. Popular choices are (see [33]):

- $\theta = 0$, which corresponds to the first order explicit Euler scheme. This scheme requires extremely small time-step sizes for convergence. Thus, it is not feasible for FSI.

- $\theta = 0.5$ leads to the second order Crank-Nicholson scheme (CN). This scheme is A-stable, however, for long-run simulations of the FSI 2 benchmark described in section 7.1, it breaks down after some time.
- $\theta = 0.5 + \Delta t$ attempts to repair the CN scheme by shifting it to the implicit side. It is second order as well but now additionally strictly A-stable. No breakdown occurs for longer simulations (at least for FSI 2).
- $\theta = 1$ equals to the first order Implicit (Backward) Euler (BE) scheme, which is strongly A-stable. However, this time-stepping scheme introduces a lot of damping, therefore it its not applicable for the FSI 2 benchmark, as no oscillations would occur.

Applying this scheme to the fluid-structure-interaction equations (3.10) yields (modified equations only)

$$\begin{aligned}
\left(\hat{J}^\theta \hat{\rho}_f(\hat{v}_f^n - \hat{v}_f^{n-1}) \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} - \left((\hat{J}\hat{\rho}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1})^\theta \cdot (\hat{u}_f^n - \hat{u}_f^{n-1}) \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \\
\Delta t \ \left[\left(\hat{J}\hat{\rho}_f\hat{\nabla}\hat{v}_f\hat{F}^{-1}\cdot\hat{v}_f \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\sigma}_f^v\hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f}\right]^\theta \\
-\Delta t \left[\left\langle \hat{\rho}_f\hat{\nu}_f\hat{J}\hat{F}^{-T}\hat{\nabla}\hat{v}_f^T\hat{F}^{-T}\cdot\hat{n}_f \ , \hat{\varphi}_f^v\right\rangle_{\hat{\Gamma}_{out}}\right]^\theta + \\
\left(\hat{J}(-\hat{p}_f)\hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f} = 0
\end{aligned} \tag{4.7}$$

$$\begin{aligned}
\left(\hat{\rho}_s(\hat{v}_s^n - \hat{v}_s^{n-1}) \ , \ \hat{\varphi}_s^v\right)_{\hat{\Omega}_s} + \Delta t \ \left(\hat{F}\hat{\Sigma}_s \ , \ \hat{\nabla}\hat{\varphi}_s^v\right)_{\hat{\Omega}_s}^\theta = 0 \\
\left((\hat{u}_s^n - \hat{u}_s^{n-1}) - \Delta t \ \hat{v}_s^\theta \ , \ \hat{\varphi}_s^u\right)_{\hat{\Omega}_s} = 0
\end{aligned}$$

with the modification

$$\hat{\sigma}_f^v := \hat{\rho}_f\hat{\nu}_f(\hat{\nabla}\hat{v}_f\hat{F}^{-1} + \hat{F}^{-T}\hat{\nabla}\hat{v}_f^T)$$

as explained below. Further, we use the abbreviation $A^\theta := \theta A^n + (1-\theta)A^{n-1}$. The superscripts $n$ and $n-1$ denote the evaluation at times $t^n$ and $t^{n-1}$ respectively. For simplicity, we will skip the $n$ superscripts from now on.

**Remark 4.2.2** (Implicit Pressure)**.** The pressure term contained in $\hat{\sigma}_f$ is always treated fully implicitly, i.e. any appearing pressure variable within the fluid stress tensor is treated as if

$\theta = 1$ holds. Other terms are not affected by this modification. The reasoning behind this comes from the theory of differential algebraic equations (DAE), see for example [29].

## 4.3 Linearization

In order to obtain the bilinear form required for discretization, we have to get grid of the non-linearities within the FSI equations (4.6, 4.7). A common way to do so is to linearize the equations using Newton's method [11].

**Algorithm 1** Newton Linearization

Let $A(u)(\varphi)$ be a semi-linear form (linear with respect to the second argument), $F(\varphi)$ a linear form. Then the solution of $A(u)(\varphi) = F(\varphi)$ can be obtained by the following iteration:

1: Initial guess $u_0$
2: **for** $k = 0, 1, \ldots$ until convergence **do**
3: Solve $A'(u_k)(\varphi, \delta u_k) = F(\varphi) - A(u_k)(\varphi)$ for $\delta u_k$
4: Update $u_{k+1} := u_k + \delta u_k$
5: **end for**

**Remark 4.3.1** (Convergence Properties of Newton's Method)**.** Under several conditions on the differentiability and boundedness of the semi-linear form $A$, Newton's method converges quadratically if provided with a good starting value. For details on the assumptions and precise statement of the convergence property, see e.g. the Newton-Kantorovich theorem in [21].

The term $A'(u)(\varphi, \delta u)$ denotes the directional derivative of $A$ into the direction of the function $\delta u$ (similar to classical directional derivative in $\mathbb{R}^n$). This Jacobian is a bilinear form wrt. $(\varphi, \delta u)$, which is exactly what we need for discretization. For infinite dimensional function spaces, this derivative is typically called Gâteaux -differential.

**Definition 4.3.2** (Gâteaux -Derivative)**.** The directional derivative of a semi-linear form $A$ is defined as

$$A'(u)(\delta u, \varphi) := \lim_{\epsilon \to 0} \frac{1}{\epsilon} \left( A(u + \epsilon \delta u)(\varphi) - A(u)(\varphi) \right) = \frac{d}{d\epsilon} A(u + \epsilon \delta u)(\varphi)|_{\epsilon=0}$$

Let us continue with some simple examples before we attempt to linearise the whole FSI equations. We will always assume that the necessary requirements for all those operations are fulfilled.

**Example 4.3.3.** We start with the most simple example $A(u) := u$. Using the definition above, we get

$$A'(u)(\delta u) := \frac{d}{d\epsilon} A(u + \epsilon\delta u)|_{\epsilon=0} = \frac{d}{d\epsilon}(u + \epsilon\delta u)|_{\epsilon=0} = \delta u. \tag{4.8}$$

**Example 4.3.4.** Similarly we obtain for $A(u) := \nabla u$:

$$A'(u)(\delta u) = \frac{d}{d\epsilon}\nabla(u + \epsilon\delta u)|_{\epsilon=0} = \nabla\delta u. \tag{4.9}$$

**Example 4.3.5.** For non-linear expressions like $A(u) := \nabla u \cdot u$, we get:

$$\begin{aligned} A'(u)(\delta u) &= \frac{d}{d\epsilon}\left[\nabla(u + \epsilon\delta u) \cdot (u + \epsilon\delta u)\right]|_{\epsilon=0} \\ &= \frac{d}{d\epsilon}\left[\nabla \cdot u + \nabla u \cdot \epsilon\delta u + \epsilon\nabla\delta u \cdot u + \epsilon\nabla\delta u \cdot \epsilon\delta u\right]|_{\epsilon=0} \\ &= \nabla\delta u \cdot u + \nabla u \cdot \delta u. \end{aligned} \tag{4.10}$$

Note that this corresponds to the usual product-rule for derivatives.

For the differentiation of the whole FSI operator (4.12), we need the derivatives of quantities like $\hat{J}, \hat{F}, \hat{F}^{-1}$ and others. Since we will use the notation $A'$ for the full Jacobian, we denote the following derivatives by e.g. $\partial_u \hat{F}(u)$ or simply $\partial\hat{F}$, if it is clear by which variable we are differentiating.

**Theorem 4.3.6** (FSI-Related Derivatives)**.**

$$\begin{aligned} \partial\hat{F} &= \hat{\nabla}\delta\hat{u} \\ \partial\hat{J} &= \hat{J}tr(\hat{F}^{-1}\hat{\nabla}\delta\hat{u}) \\ \partial\hat{F}^{-1} &= -\hat{F}^{-1}\hat{\nabla}\delta\hat{u}\hat{F}^{-1} \\ \partial\hat{F}^{-T} &= (\partial\hat{F}^{-1})^T \\ \partial tr(E) &= tr(\partial E) \end{aligned} \tag{4.11}$$

*Proof.* unless trivial, see for example [16, 2]. □

Now we have the necessary ingredients to tackle the FSI problem. Its semi-linear form $A(U)(\Phi)$ and linear form $F(\Phi)$ are given as the sum of all equations in (4.7). Note that the derivative only applies to functions evaluated at time-step $n$. Functions from previous time-steps are constant with respect to $\delta U$ and therefore belong to the right-hand side $F$. With

this in mind, we can define the FSI operator $A$ and right-hand side $F$ after time-discretization as

$$\begin{aligned}
A(U)(\Phi) = &\left(\hat{\rho}_f \hat{J}^\theta (\hat{v}_f - \hat{v}_f^{n-1}) \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} - \left(\hat{J}\hat{\rho}_f \hat{F}^{-1}\hat{\nabla}\hat{v}_f \cdot \hat{u}_f \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} \\
&+ \Delta t \ \theta \left[\left(\hat{J}\hat{\rho}_f \hat{F}^{-1}\hat{\nabla}\hat{v}_f \cdot \hat{v}_f \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\sigma}_f^v \hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f}\right] \\
&- \Delta t \ \theta \left\langle \hat{\rho}_f \hat{\nu}_f \hat{J}\hat{F}^{-T}\hat{\nabla}\hat{v}_f^T \hat{F}^{-T} \cdot \hat{n}_f \ , \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_{out}} \\
&+ \left(\hat{J}(-\hat{p}_f)\hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f} \\
&+ (\hat{\rho}_s \hat{v}_s^n \ , \ \hat{\varphi}_s^v)_{\hat{\Omega}_s} + \Delta t \ \theta \left(\hat{F}\hat{\Sigma}_s \ , \ \hat{\nabla}\hat{\varphi}_s^v\right)_{\hat{\Omega}_s} \\
&+ (\hat{u}_s \ , \ \hat{\varphi}_s^u)_{\hat{\Omega}_s} - \Delta t \ \theta \ (\hat{v}_s \ , \ \hat{\varphi}_s^u)_{\hat{\Omega}_s} \\
&+ \alpha_u \left(\frac{1}{\hat{J}}\hat{\nabla}\hat{u}_f \ , \ \hat{\nabla}\hat{\varphi}_f^u\right)_{\hat{\Omega}_f}
\end{aligned} \tag{4.12}$$

and

$$\begin{aligned}
F(\Phi) = &\left(\hat{\rho}_s \hat{v}_s^{n-1} \ , \ \hat{\varphi}_s^v\right)_{\hat{\Omega}_s} - \Delta t \ (1-\theta) \left(\hat{F}^{n-1}\hat{\Sigma}_s^{n-1} \ , \ \hat{\nabla}\hat{\varphi}_s^v\right)_{\hat{\Omega}_s} \\
&+ \left(\hat{u}_s^{n-1} \ , \ \hat{\varphi}_s^u\right)_{\hat{\Omega}_s} + \Delta t \ (1-\theta) \ \left(\hat{v}_s^{n-1} \ , \ \hat{\varphi}_s^u\right)_{\hat{\Omega}_s} \\
&- \left(\hat{J}\hat{\rho}_f \hat{F}^{-1}\hat{\nabla}\hat{v}_f \cdot \hat{u}_f^{n-1} \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} \\
&- \Delta t \ (1-\theta) \left[\left(\hat{J}\hat{\rho}_f \hat{F}^{-1}\hat{\nabla}\hat{v}_f \cdot \hat{v}_f \ , \ \hat{\varphi}_f^v\right)_{\hat{\Omega}_f} + \left(\hat{J}\hat{\sigma}_f^v \hat{F}^{-T} \ , \ \hat{\nabla}\hat{\varphi}_f^v\right)_{\hat{\Omega}_f}\right]^{n-1} \\
&+ \Delta t \ (1-\theta) \left[\left\langle \hat{\rho}_f \hat{\nu}_f \hat{J}\hat{F}^{-T}\hat{\nabla}\hat{v}_f^T \hat{F}^{-T} \cdot \hat{n}_f \ , \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_{out}}\right]^{n-1}
\end{aligned} \tag{4.13}$$

with the test-function $\Phi := (\hat{\varphi}_f^v, \hat{\varphi}_f^u, \hat{\varphi}_f^p, \hat{\varphi}_s^v, \hat{\varphi}_s^u)$ and solution variable $U := (\hat{v}_f, \hat{u}_f, \hat{p}_f, \hat{v}_s, \hat{u}_s)$. Due to the nested non-linearities, which all require multiple applications of the product rule, the computation of all single terms of the Jacobian is quite lengthy. However, when implementing the derivatives we do not actually need to expand all terms, therefore we do not write them down explicitly here.

## 4.4 Spatial Discretization

Now we are ready to discretize the Jacobian and the Newton residual given in algorithm 1. We are going to use a quadrilateral subdivision $\hat{\Omega} = \bigcup_i \mathcal{T}_i$ of the reference domain with $Q(k)$

shape functions for displacements and velocities and discontinuous $Q(k-1)$ elements for the pressure. The subdivision matches the interface, i.e. every $\mathcal{T}_i$ is either part of the fluid or the solid domain, but not both.

Note that since $A'$ is bilinear wrt. $\delta U$ and test functions $\Phi$, we have to represent the Newton correction $\delta U$ using discrete subspaces, not the solution $U$ itself.

As indicated above we use the following discrete function spaces

$$\begin{aligned} V_h^s &:= \left\{ v \in H^1(\hat{\Omega}_s) : v|_{\mathcal{T}_i} \in Q(k) \quad \forall \mathcal{T}_i \in \hat{\Omega}_s \right\} \\ V_h^f &:= \left\{ v \in H^1(\hat{\Omega}_f) : v|_{\mathcal{T}_i} \in Q(k) \quad \forall \mathcal{T}_i \in \hat{\Omega}_f \right\} \\ L_h^f &:= \left\{ v \in L^2(\hat{\Omega}_f) : v|_{\mathcal{T}_i} \in P(k-1) \, \forall \mathcal{T}_i \in \hat{\Omega}_f \right\} \end{aligned} \tag{4.14}$$

with the nodal basis functions

$$\begin{aligned} V_h^s &= span\{\varphi_{s,h}^v[j], \ j = 1, \ldots, N_{v_s}\} = span\{\varphi_{s,h}^u[j], \ j = 1, \ldots, N_{u_s}\} \\ V_h^f &= span\{\varphi_{f,h}^v[j], \ j = 1, \ldots, N_{v_f}\} = span\{\varphi_{f,h}^u[j], \ j = 1, \ldots, N_{u_f}\} \\ L_h^f &= span\{\varphi_{f,h}^p[j], \ j = 1, \ldots, N_{p_f}\}. \end{aligned} \tag{4.15}$$

This leads to the space $X := (V_h^f)^{dim} \times (V_h^f)^{dim} \times L_h^f \times (V_h^s)^{dim} \times (V_h^s)^{dim}$ for the discrete Newton correction $\delta U$. The basis functions for the whole space $X$ are the combinations of the basis functions of the components in the following way:

$$\begin{aligned} \Phi_j &= \left(\varphi_{f,h}^v[j], 0, 0, \ldots, 0\right), \ j = 1, \ldots N_{v_f} \\ \Phi_{j+N_{v_f}} &= \left(0, \varphi_{f,h}^v[j], 0, 0, \ldots, 0\right), \ j = 1, \ldots N_{v_f} \\ \Phi_{j+2N_{v_f}} &= \left(0, 0, \varphi_{f,h}^u[j], 0, \ldots, 0\right), \ j = 1, \ldots N_{u_f} \\ &\ldots \text{ and so on.} \end{aligned} \tag{4.16}$$

Note that $\Phi_j$ has $4 \cdot dim + 1$ components since we require the basis functions for displacements and velocities $dim$-times.

Using the ansatz

$$\delta U := \sum_{j=1}^{N_{FSI}} \delta U_j \Phi_j$$

for the equation at each Newton step $k$ leads to

$$\begin{aligned} A'(U_k)(\sum_{i=1}^{N_{FSI}} \Phi_i, \sum_{j=1}^{N_{FSI}} \delta U_j \Phi_j) &= F(\sum_{i=1}^{N_{FSI}} \Phi_i) - A(U_k)(\sum_{i=1}^{N_{FSI}} \Phi_i) \\ \Leftrightarrow \sum_{i,j=1}^{N_{FSI}} A'(U_k)(\Phi_i, \Phi_j))\delta U_j &= \sum_{i=1}^{N_{FSI}} \left(F(\Phi_i) - A(U_k)(\Phi_i)\right). \\ A_h \delta U_h &= F_h \end{aligned} \tag{4.17}$$

with

$$\begin{aligned} \delta U_h &:= (\delta U_1, \ldots, \delta U_{N_{FSI}}) \in \mathbb{R}^{N_{FSI}}, \\ A_h &= \left(A'(U_k)(\Phi_i, \Phi_j)\right)_{i,j=1}^{N_{FSI}} \in \mathbb{R}^{N_{FSI} \times N_{FSI}} \text{ and} \\ F_h &= (A(U_k)(\Phi_i) - F(\Phi_i))_{j=1}^{N_{FSI}} \in \mathbb{R}^{N_{FSI}} \end{aligned} \tag{4.18}$$

Please note that we omitted the index $k$ for the Newton corrections $\delta U$ for simplicity.

Assembling $A_h$ leads to a matrix of the form

$$\begin{bmatrix} \mathcal{M} & 0 & 0 \\ 0 & \mathcal{S} & 0 \\ \mathcal{B} & 0 & \mathcal{F} \end{bmatrix}, \tag{4.19}$$

where $\mathcal{M}, \mathcal{S}$ and $\mathcal{F}$ denote the discrete versions of the mesh-motion, solid and fluid equations respectively. The coupling terms $\mathcal{B}$ simply arise because of the ALE transformation of the fluid equations.

Up to now, we have not yet included any boundary or interface conditions in our discrete formulation. This issue will be tackled in the next sections. Therein, we differentiate between degrees of freedom located on the interface (denoted by $I$) and dofs in the interior ($\Omega$). Furthermore, we split the fluid and solid variables into $\hat{v}_f, \hat{p}_f$ and $\hat{u}_s, \hat{v}_s$. Note that we do not have pressure dofs on the interface because we are using discontinuous elements. This leads

to the more detailed description of the Jacobian

$$\begin{bmatrix}
\mathcal{M}^{\Omega\Omega} & \mathcal{M}^{\Omega I} & & & & & & & \\
\mathcal{M}^{I\Omega} & \mathcal{M}^{II} & & & & & & & \\
 & & \mathcal{S}_{uu}^{\Omega\Omega} & \mathcal{S}_{uu}^{\Omega I} & \mathcal{S}_{uv}^{\Omega\Omega} & \mathcal{S}_{uv}^{\Omega I} & & & \\
 & & \mathcal{S}_{uu}^{I\Omega} & \mathcal{S}_{uu}^{II} & \mathcal{S}_{uv}^{I\Omega} & \mathcal{S}_{uv}^{II} & & & \\
 & & \mathcal{S}_{vu}^{\Omega\Omega} & \mathcal{S}_{vu}^{\Omega I} & \mathcal{S}_{vv}^{\Omega\Omega} & \mathcal{S}_{vv}^{\Omega I} & & & \\
 & & \mathcal{S}_{vu}^{I\Omega} & \mathcal{S}_{vu}^{II} & \mathcal{S}_{vv}^{I\Omega} & \mathcal{S}_{vv}^{II} & & & \\
\mathcal{B}_{fm}^{I\Omega} & \mathcal{B}_{fm}^{II} & & & & & \mathcal{F}_{vv}^{II} & \mathcal{F}_{vv}^{I\Omega} & \mathcal{F}_{vp}^{I} \\
\mathcal{B}_{fm}^{\Omega\Omega} & \mathcal{B}_{fm}^{\Omega I} & & & & & \mathcal{F}_{vv}^{\Omega I} & \mathcal{F}_{vv}^{\Omega\Omega} & \mathcal{F}_{vp}^{\Omega} \\
\mathcal{B}_{pm}^{\Omega} & \mathcal{B}_{pm}^{I} & & & & & \mathcal{F}_{pv}^{I} & \mathcal{F}_{pv}^{\Omega} & 0
\end{bmatrix}. \tag{4.20}$$

### 4.4.1 Discrete Boundary Conditions

Basically, we have to differentiate between boundary conditions subject to the solution variable $U_h$ and those that apply to the Newton corrections $\delta U_h$.

It is important to keep in mind that any constraints incorporated into the matrix $A_h$ and right-hand side $F_h$ only directly affect the Newton update and not the actual solution.

The first type of boundary conditions are the initial conditions applied to the solution itself. Typical examples include a given inflow velocity at some part of the boundary and zero values for the mesh displacement (e.g. fixed channel). These conditions are directly incorporated into the solution vector $U_h$ (more specifically into the initial guess $U_0$ of the Newton algorithm at each time-step):

$$U_h(i) := g(i), \quad \text{for all degrees of freedom } i \text{ subject to given initial conditions } g.$$

Boundary conditions applied to the Newton correction are nothing else than homogeneous versions of the previous conditions.

After some Newton iterations, we have for the solution vector

$$U_h = U_0 + \delta U_0 + \delta U_1 + \dots.$$

To retain the conditions imposed on $U_0$, we have to make sure that these are not disturbed by the following corrections. Therefore we have to apply homogeneous boundary conditions to $\delta U_k$ in every Newton step for all those degrees of freedom, that are set in $U_0$.

In the following part, we will investigate how to apply these homogeneous Dirichlet conditions.

**Notation 4.4.1.** Let $BC$ denote the set of all degrees of freedom for which such constraints should be imposed.

One way to incorporate the constraints is by the following modification of the matrix $A_h$ and right-hand-side $F_h$:

$$\begin{aligned} \forall i \in BC, j = 1, \ldots, N_{FSI} : \text{set} \\ A_h(i,i) &:= 1, \\ A_h(i,j) &:= 0, \\ A_h(j,i) &:= 0 \\ F_h(i) &:= 0. \end{aligned} \tag{4.21}$$

Trivially, the solution of $A_h x = F_h$ will have zero values for all constrained degrees of freedom. The diagonal element $A_h(i,i)$ may in fact be chosen arbitrarily non-zero, for example one could chose $A_h(i,i)$ as the average value of the eliminated values. This makes particular sense if the values around $A_h(i,i)$ are much larger or smaller than 1 as the matrix structure will not be as severely destroyed.

The obvious disadvantage of this method is that we have to modify each row and column subject to constraints, which might be costly, depending on the representation of the sparse matrix.

A simpler variant of the above method would be

$$\forall i \in BC : \text{set } A_h(i,i) := \beta, \quad \text{with } \beta \text{ sufficiently large.}$$

The penalty parameter $\beta$ has to be chosen such that it dominates the corresponding row and column of the original matrix. Then, the solution will be approximately the same as if we had eliminated these entries.

### 4.4.2 Interface Conditions

Without interface conditions, the fluid and solid equations would be completely independent of each other and no interaction would take place. Therefore we will now have a look at the most important part of fluid-structure-interaction .

As given in section 3.4 we have three interface conditions given on the continuous level:

1. continuity of stresses,
2. continuity of velocities and
3. continuity of displacements.

For the continuous problem, the continuity of stresses (1) is implicitly fulfilled, if the test functions for the velocities match on the interface. Therefore we only have to care about conditions (2) and (3), as long as the discrete velocity test-functions have the same property.

**Weak Coupling**

Again, there are several possibilities of incorporating these coupling constraints. One way to do so, is to treat the interface conditions as additional equations and test them with according functions. This approach is closely related to Nitsche's method used in DG formulations (see e.g. [28]).

**Lemma 4.4.2** (Weak Coupling)**.** Enforcing the identities $\hat{u}_f = \hat{u}_s$ and $\hat{v}_f = \hat{v}_s$ on $\hat{\Gamma}_I$ is realized by the additional equations

$$\begin{aligned} \alpha_u \left\langle \hat{u}_s - \hat{u}_f \ , \ \hat{\varphi}_s^u - \hat{\varphi}_f^u \right\rangle_{\hat{\Gamma}_I} = 0 \quad \text{and} \\ \left\langle \hat{F}\hat{\Sigma}_s \hat{n}_s - \hat{J}\hat{\sigma}_f \hat{F}^{-T} \hat{n}_f \ , \ \hat{\varphi}_s^v - \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_I} - \alpha_v \left\langle \hat{v}_s - \hat{v}_f \ , \ \hat{\varphi}_s^v - \hat{\varphi}_f^v \right\rangle_{\hat{\Gamma}_I} = 0, \end{aligned} \tag{4.22}$$

with the penalty parameters $\alpha_u$ and $\alpha_v$ sufficiently large. Note that these additional equations have to be linearized and adapted to the $\theta$-time-stepping scheme as well.

The big advantage of this method is that it can be easily implemented and no special care has to be taken if different elements for $\hat{u}_f$ and $\hat{u}_s$ or $\hat{v}_f$ and $\hat{v}_s$ are used. The disadvantage is, that we have to chose an appropriate penalty parameter. If we chose $\alpha_v$ too low, the coupling error increases, whereas a large value has a negative impact on the iterative GMRES solver. In figure 4.1 we observe the coupling error depending on different penalty parameters and figure 4.2 shows the effect on the required number of GMRES iterations.

For the displacement coupling, we could also use a slightly modified variant of this method (see e.g. [22]). Instead of using test functions $\hat{\varphi}_f^u$ for the mesh-motion part as given before, we

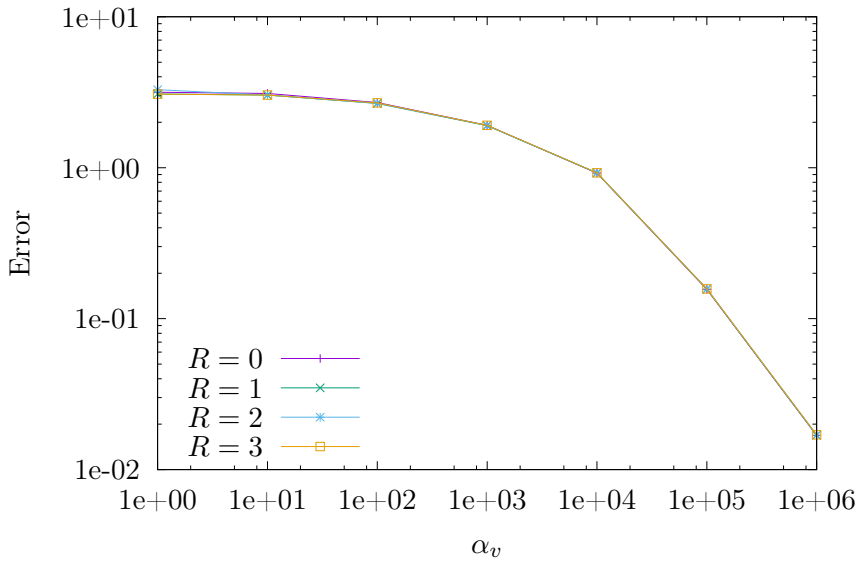



Figure 4.1: Velocity-coupling error depending on the choice of the penalty parameter $\alpha_v$ for different refinement levels R. The error was computed as $(\int_{\hat{\Gamma}_I} |\hat{v}_s - \hat{v}_f|^2 d\hat{s})^{1/2}$.

chose them to be zero on the interface $\hat{\Gamma}_I$. Doing so eliminates the second row of the matrix (4.20). This row is then populated with the displacement coupling matrices by assembling the equation

$$\left\langle \hat{u}_f - \hat{u}_s \ , \ \hat{\varphi}_f^u \right\rangle_{\hat{\Gamma}_I} = 0.$$

This leads to the modified Jacobian

$$\begin{bmatrix}
\mathcal{M}^{\Omega\Omega} & \mathcal{M}^{\Omega I} & & & & & & & \\
 & \mathcal{U}_{mm} & -\mathcal{U}_{ms} & & & & & & \\
 & & \mathcal{S}_{uu}^{\Omega\Omega} & \mathcal{S}_{uu}^{\Omega I} & \mathcal{S}_{uv}^{\Omega\Omega} & \mathcal{S}_{uv}^{\Omega I} & & & \\
 & & \mathcal{S}_{uu}^{I\Omega} & \mathcal{S}_{uu}^{II} & \mathcal{S}_{uv}^{I\Omega} & \mathcal{S}_{uv}^{II} & & & \\
 & & \mathcal{S}_{vu}^{\Omega\Omega} & \mathcal{S}_{vu}^{\Omega I} & \mathcal{S}_{vv}^{\Omega\Omega} & \mathcal{S}_{vv}^{\Omega I} & & & \\
 & & \mathcal{S}_{vu}^{I\Omega} & \mathcal{S}_{vu}^{II} + \mathcal{V}_{vu} & \mathcal{S}_{vv}^{I\Omega} & \mathcal{S}_{vv}^{II} + \mathcal{V}_{vv} & \mathcal{V}_{vv} & & \mathcal{V}_{vp} \\
\mathcal{B}_{fm}^{I\Omega} & \mathcal{B}_{fm}^{II} & & \mathcal{V}_{vu} & & \mathcal{V}_{vv} & \mathcal{V}_{vv} + \mathcal{F}_{vv}^{II} & \mathcal{F}_{vv}^{I\Omega} & \mathcal{V}_{vp} + \mathcal{F}_{vp}^{I} \\
\mathcal{B}_{fm}^{\Omega\Omega} & \mathcal{B}_{fm}^{\Omega I} & & & & & \mathcal{F}_{vv}^{\Omega I} & \mathcal{F}_{vv}^{\Omega\Omega} & \mathcal{F}_{vp}^{\Omega} \\
\mathcal{B}_{pm}^{\Omega} & \mathcal{B}_{pm}^{I} & & & & & \mathcal{F}_{pv}^{I} & \mathcal{F}_{pv}^{\Omega} & 0
\end{bmatrix} \tag{4.23}$$

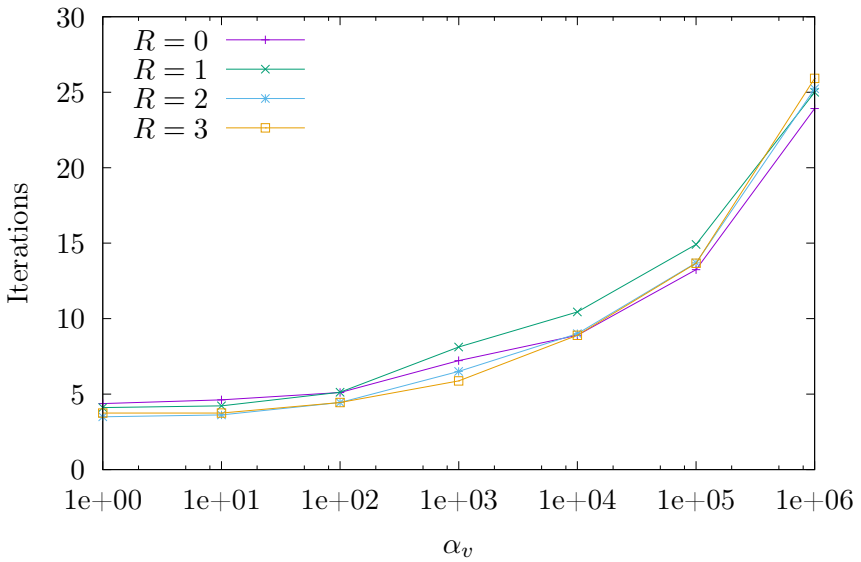



Figure 4.2: Effect of $\alpha_v$ on the convergence of the GMRES solver using the 21 preconditioner with direct solvers for the sub-problems for different refinement levels $R$. Again, the FSI 2 benchmark was considered.

Note that for simplicity of notation, the velocity coupling blocks $\mathcal{V}$ should only indicate that we get contributions at these places, but they are not meant to be equal.

The coupling mass matrices $\mathcal{U}_{mm}$ and $\mathcal{U}_{ms}$ could be replaced by "identity" matrices. However, one has to be extremely careful when modifying the Jacobian by hand, as one has to account for possibly different elements, refinement levels and orderings of the degrees of freedom. In the general case, these "identity" matrices would be some kind of interpolation and/or reordering operators.

Due to the negative effect on the required number of GMRES iterations, we use a different way of coupling the displacement and velocity, which is presented in the next section. This method is already implemented in deal.II and shows a very convenient way of implementing a wide range of constraints.

### deal.II Constraint Matrix

In this section, we describe the behaviour of the deal.II ConstraintMatrix class. The name already indicates the general usage uf this object: incorporating constraints.

**Definition 4.4.3** (Constraints)**.** Linear constraints on a degree of freedom $x_i$ can be repre-

sented as

$$x_i = \sum_{j \neq i} c_{ij} x_j$$

or equivalently $C \cdot X = 0$ with $X = (x_1, x_2, \dots)$.

Incorporating the constraints $C$ into the solution of $A \cdot X = b$ stemming from a continuous problem $a(\varphi_i, \varphi_j) = f(\varphi_j)$ leads to the following equations to be solved:

$$\begin{aligned} AX &= b \\ CX &= 0. \end{aligned} \tag{4.24}$$

However, it is not clear whether this system has a solution at all. A different and very general approach coming from hp-discretization methods is presented in [3]. The overall idea is to replace the original, unconstrained shape-functions by conforming modifications carrying the constraints. In particular, the following changes are incorporated:

$$\tilde{\varphi}_j = \varphi_j + \sum_{x_i \text{ constr}} c_{ij} \varphi_i \tag{4.25}$$

$$\tilde{A}_{ij} = \begin{cases} a(\tilde{\varphi}_i, \tilde{\varphi}_i), \text{ i, j unconstrained} \\ 1, \text{ i = j constrained} \\ 0, \text{ i or j unconstrained} \end{cases} \qquad \tilde{F}_i = \begin{cases} (f, \tilde{\varphi}_i), \text{ i unconstrained} \\ 0, \text{ i constrained} \end{cases} \tag{4.26}$$

At first glance, this method seems to be very costly as it involves a lot of manipulations of matrix entries. However, one can overcome this issue by applying these modifications when distributing the local contributions from each cell into the global matrix and vector. The only remaining thing to take care of are the additional entries in the sparsity pattern.

In the simplest case of uniform refinement (no hanging-nodes) and equal elements for the fluid and solid pairs of displacement and velocity, the interface coupling is a simple 1-to-1 coupling of degrees of freedom (see figure 4.3). Therefore, we only have constraints of the form $x_i^f = x_j^s$ for corresponding degrees of freedom on the fluid and solid side. For the general case, we have to represent one set of basis-functions by linear combinations of the others.

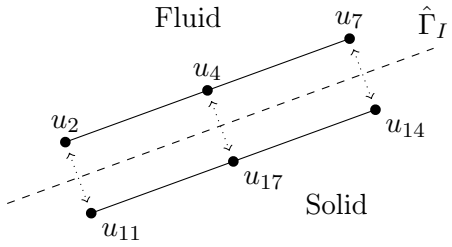



Figure 4.3: 1-to-1 interface coupling for $Q^2$-elements.

Using this approach leads to the following modified Jacobian

$$\begin{bmatrix}
\mathcal{M}^{\Omega\Omega} & 0 & & \mathcal{M}^{\Omega I} & & & & & \\
0 & I & & & & & & & \\
 & & \mathcal{S}_{uu}^{\Omega\Omega} & \mathcal{S}_{uu}^{\Omega I} & \mathcal{S}_{uv}^{\Omega\Omega} & \mathcal{S}_{uv}^{\Omega I} & & & \\
\mathcal{M}^{I\Omega} & & \mathcal{S}_{uu}^{I\Omega} & \mathcal{S}_{uu}^{II} + \mathcal{M}^{II} & \mathcal{S}_{uv}^{I\Omega} & \mathcal{S}_{uv}^{II} & & & \\
 & & \mathcal{S}_{vu}^{\Omega\Omega} & \mathcal{S}_{vu}^{\Omega I} & \mathcal{S}_{vv}^{\Omega\Omega} & \mathcal{S}_{vv}^{\Omega I} & & & \\
\mathcal{B}_{fm}^{I\Omega} & & \mathcal{S}_{vu}^{I\Omega} & \mathcal{S}_{vu}^{II} & \mathcal{S}_{vv}^{I\Omega} & \mathcal{S}_{vv}^{II} + \mathcal{F}_{vv}^{II} & & \mathcal{F}_{vv}^{I\Omega} & \mathcal{F}_{vp}^{I} \\
0 & 0 & & & & & I & 0 & 0 \\
\mathcal{B}_{fm}^{\Omega\Omega} & 0 & & \mathcal{B}_{fm}^{\Omega I} & & \mathcal{F}_{vv}^{\Omega I} & 0 & \mathcal{F}_{vv}^{\Omega\Omega} & \mathcal{F}_{vp}^{\Omega} \\
\mathcal{B}_{pm}^{\Omega} & 0 & & \mathcal{B}_{pm}^{I} & & \mathcal{F}_{pv}^{I} & 0 & \mathcal{F}_{pv}^{\Omega} & 0
\end{bmatrix} \tag{4.27}$$

Note that the moved matrix blocks are only equal if the corresponding degrees of freedom have the very same ordering. Otherwise, the ordering of the rows and columns within these blocks is different.

**Remark 4.4.4.**

- Incorporating homogeneous Dirichlet constraints as presented in equation (4.21) is a special case of this method. Therefore the deal.II Constraint-Matrix may be used for interface coupling as well as boundary conditions.
- While we focused solely on homogeneous constraints, this class can also deal with inhomogeneous ones. Also, the modification of the sparsity pattern and the distribution of local contributions is already included in deal.II.
- We will refer to the this type of interface coupling as "strong", whereas the Nitsche approach presented in section 4.4.2 will be called "weak" coupling.

### 4.4.3 Comparison With Other Formulations

For the formulation and discretization of fluid-structure-interaction problems several different variants exist. In [22, 32], the velocity and displacement variable is defined in the whole domain $\hat{\Omega}$. This makes the interface coupling a lot easier, since essentially nothing has to be done. The drawback of such a formulation is that one has to use the same finite elements for the respective fluid and solid parts, whereas the approach presented in this work allows different choices.

The final preconditioner presented in this work will be closely related to the one presented in [20, 18], adapted to the slightly different discretizations. In [20], the second order solid equations are treated by the Newmark scheme, while we split it into a first order system and tackle it by the One-Step-$\theta$ method. This further leads to different coupling strategies on the interface and moreover to a different block-structure of the Jacobian.

All of these formulations include the mesh-motion equation via ALE transformations. A different approach is utilized in [14]. Therein, the mesh is moved in every step via post-processing. Therefore, the Jacobian misses the fluid displacement variable completely.

## 4.5 Assembling Procedure

Finally, we are able to assemble the matrix and right-hand side vector. A procedure similar to algorithm 2 is typically used in all finite element codes.

**Algorithm 2** Assembling of the Jacobian (example: $a(\hat{u}, \hat{v}) = (\hat{\nabla}\hat{u}, \hat{\nabla}\hat{v})$)

```
for all cells T̂_k do
    M_local := 0
    for all quadrature points q do
        for all local degrees of freedom i do
            for all local degrees of freedom j do
                M_local(j,i) += ∇̂Φ̂_i(q) * ∇̂Φ̂_j(q) * JxW(q)
            end for
        end for
    end for
end for
```

**Remark 4.5.1.**

- The column index $i$ corresponds to the ansatz-function while the row index $j$ denotes the test-function.
- Quadrature points are chosen on a unit cell as Gauss-Legendre points of order $k+1$ for $Q^k$ elements.
- The term $JxW(q)$ denotes the weight of quadrature point $q$ taking into account the deformation of the cell compared to the unit cell (please note that this deformation is not directly related to the deformation caused by the displacements $\hat{u}_f$ or $\hat{u}_s$!).
- For multi-valued problems like FSI, it makes sense to split the innermost loop depending on the component associated to dof $j$ in the following way:

  1: **for** all local degrees of freedom $j$ **do**
  2: **if** $j \simeq \hat{u}_f$ **then**
  3: $M_{local}(j,i)$ += (mesh-motion-equation) $* \hat{\nabla}\hat{\varphi}_j^u(q) * JxW(q)$
  4: **else if** $j \simeq \hat{v}_f$ **then**
  5: $M_{local}(j,i)$ += (fluid-momentum-equation) $* \hat{\nabla}\hat{\varphi}_j^v(q) * JxW(q)$
  6: **else if** $j \simeq \hat{p}_f$ **then**
  7: $M_{local}(j,i)$ += (incompressibility-equation) $* \hat{\varphi}_j^p(q) * JxW(q)$
  8: **end if**
  9: **end for**

  This avoids the unnecessary computation of lots of zeros.

Assembling the Jacobian for the FSI 2 benchmark leads to the sparsity pattern presented in figure 4.4. Here, $Q(2)$ elements for the displacements and velocities and $DGQ(1)$ elements for the pressure were used.

## 4.6 Summary

Now we conclude this chapter by giving a rough overview of the necessary steps in a program solving the FSI equations in algorithm 3.

**Remark 4.6.1** (Initial Guess)**.** At the first time-step, we chose a zero initial guess with appropriate initial conditions for the solution $U$. For larger time-step sizes this may lead to

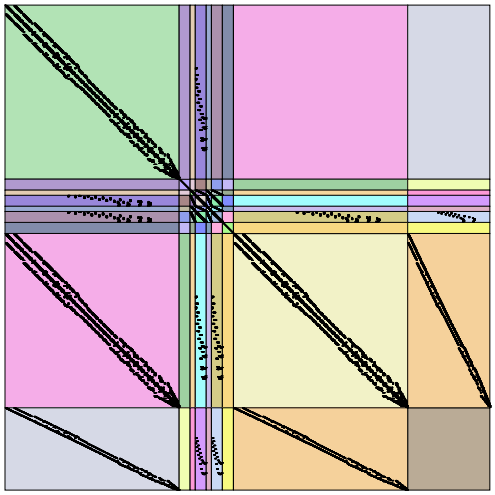


Figure 4.4: Sparsity pattern with 4148 dofs and 0.88 % (151, 530) non-zero entries. Blocks correspond to fluid-displacement (interior, interface), solid-displacement (interior, interface), solid-velocity (interior, interface), fluid-velocity (interface, interior) and pressure. Block-wise ordering using Cuthill-McKee (see section 5.1.1)

more Newton iterations at the beginning. For all subsequent time-steps, we can reuse the solution from the previous time-step with adjusted initial conditions as a starting value for the Newton iteration.

If we use an iterative solver like GMRES for the solution of the linear system, we also require an initial guess for the Newton update. In contrast to the previous situation, we have no good candidate from the time-step before, since the corrections are more or less "random". Therefore we use $\delta U_0 = 0$, which is a good choice for the last couple of Newton steps, as we would expect that $\delta U \to 0$.

**Remark 4.6.2** (Quasi-Newton-Method)**.** Within the Newton iterations, the Jacobian is not necessarily assembled in every single step. If the iteration makes good progress in reducing the residual, we can assume that we are already very close to the eventual solution. Therefore, the Jacobian from the previous Newton step might be a good enough approximation to continue with (Quasi-Newton method). This saves us some computational work for assembling the matrix and initializing the linear solver and preconditioner. In particular, if we use a direct solver, we can reuse the previous LU factorization and get the solution of the current iteration almost for free.

**Algorithm 3** FSI program.

Define problem parameters ($\Delta t, \theta$, material parameters, ...)
Create mesh
Assign degrees of freedom
Set-up constraints (hanging nodes, homogeneous boundary conditions, interface coupling)
Initialize sparsity pattern, Jacobian $A$ and vectors for the right-hand-side $rhs$, solution $U$, newton update $\delta U$ and previous solution $U_{-1}$

$t := 0$
Initial guess for current and previous solution $U := 0$, $U_{-1} := 0$

**while** $t < T$ **do**
    Apply initial conditions of time $t$ to $U$
    **while** not converged **do**
        Assemble $rhs$
        (if necessary): assemble Jacobian $A$
        Solve $A\delta U = rhs$
        Update $U := U + \delta U$
    **end while**
    Optional:
    - evaluate functionals (drag, lift, displacement, ...)
    - output results for current time-step
    $t := t + \Delta t$
    $U_{-1} := U$
**end while**

# Chapter 5

# Linear Solvers

The most challenging remaining question is: how do we solve the linear system in every Newton step? While for a low number of degrees of freedom a direct solver is a suitable choice, it will eventually become to expensive once the size of the problem increases (compare figure 5.3). This will become even worse for problems in three dimensions. As already mentioned before, iterative solvers will come to the rescue.

## 5.1 Direct Solvers

Although direct solvers are quite costly in terms of computational effort and memory usage, they also have some very big advantages (if applicable). First of all, the solution is more or less "exact" in a numerical sense. This may lead to fewer Newton-steps per time-step, depending on the convergence criterion of the iterative solver in use. More important, direct solvers are very robust. As long as the matrix is non-singular, direct methods will find the solution to the linear system independent of the choice of the parameters in the underlying PDEs.

Most available direct solvers perform a variant of the usual (sparse) LU-decomposition $A = LU$ with lower- and upper-triangular matrices $L$ and $U$. Once we have such a factorization, we can easily solve the initial system $Ax = b$ by solving $Ly = b$ and $Ux = y$ by forward and backward-substitution. The general algorithm of the LU-decomposition (without optimizations) is given in algorithm 4.

**Remark 5.1.1.** Since we know that $L_{ii} = 1$, we can merge the triangular factors $L$ and $U$

**Algorithm 4** LU decomposition of $n \times n$ matrix $A$, zero-based indexing.

**for** $j = 0, \ldots, n-1$ **do**
  $L_{jj} = 1$
  **for** $i = 0, \ldots, j$ **do**
    $U_{ij} = A_{ij} - \sum_{k=0}^{i-1} L_{ik} U_{kj}$
  **end for**
  **for** $i = j, \ldots, n-1$ **do**
    $L_{ij} = (A_{ij} - \sum_{k=0}^{j-1} L_{ik} U_{kj}) / U_{jj}$
  **end for**
**end for**

into one matrix without losing information.

Unfortunately, the LU-decomposition of a sparse matrix is in general dense. This effect, also called fill-in, depends heavily on the ordering of the rows and columns of $A$ (respectively the ordering of the underlying dofs). A somewhat extreme example is given in 5.1. There, the LU-factorization of the original matrix $A$ is dense, whereas for the inverse-ordered matrix $\tilde{A}$, the LU-factors have the same sparsity pattern as $\tilde{A}$.

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{bmatrix} \qquad \tilde{A} = \begin{bmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{bmatrix} \tag{5.1}$$

### 5.1.1 Reordering Schemes

The previous section gives an immediate starting point for optimizations of direct solvers: ordering the degrees of freedom in such a way that the fill-in is minimized. Many reordering methods have been developed, but unfortunately their analysis is mostly heuristically. An introduction and comparison of these schemes as well as implementational issues for direct solvers can be found in [13, 10]. For uncoupled problems, such reordering methods can lead to significant improvements. However, one has to be careful: while the minimal-degree ordering is very efficient for the Laplace equation, it falls far behind the Cuthill-McKee algorithm for FSI problems (see table 5.1 for a comparison). Further we have to be aware that many

| Laplace | Cuthill-McKee | Minimal | Random |
|---|---|---|---|
| nnz | 670,856 | 236,947 | 8,400,900 |
| nnz % | 3.7 | 1.3 | 47.0 |
| fill-in | 10.9 | 3.8 | 137.1 |

| FSI | Cuthill-McKee | Minimal | Random |
|---|---|---|---|
| nnz | 1,360,599 | 15,067,630 | 17,177,010 |
| nnz % | 7.9 | 87.5 | 99.8 |
| fill-in | 8.9 | 99.4 | 113.3 |

Table 5.1: Effect of reordering on the LU-decomposition of the FSI 2 system-matrix (top) using $Q(2) - P(1)$ elements and the Laplace equation on the same geometry using $Q(2)$ elements. Both tests were done with roughly 4000 degrees of freedom. The rows "nnz" and "nnz %" denote the number of non-zero entries in the L and U matrices and fill-in corresponds to the factor $\frac{\text{nnz}}{\text{nnz of original matrix}}$.

available direct solvers perform their own reordering, therefore the effect of manually applying such schemes is not immediately obvious.

**Remark 5.1.2** (Complexity)**.** Following [13], the application of direct solvers to a $2-d$ problem requires about $\mathcal{O}(n \log n)$ memory and $\mathcal{O}(n^{3/2})$ operations. For $3-d$ problems, these complexity bounds are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^{4/3})$. As we can see in figures 5.3 and 5.4, the UMFPACK solver performs slightly worse for the FSI 2 test case.

### 5.1.2 ILU

A variant of the LU-decomposition that "avoids" the fill-in, is the so-called incomplete LU factorization. This method follows the same algorithm as the LU-decomposition given in algorithm 4. The only difference is, that the computation of the LU-factors is restricted to entries available in the sparsity pattern of $A$. Clearly, such an incomplete factorization will not lead to the exact solution in general, but it can be useful as a preconditioner for iterative solvers. As we saw in the previous sections, the efficiency of ILU can be improved for simple problems by choosing a suitable ordering.

**Remark 5.1.3** (Variants)**.** Another possibility for the improvement of ILU is the use of larger

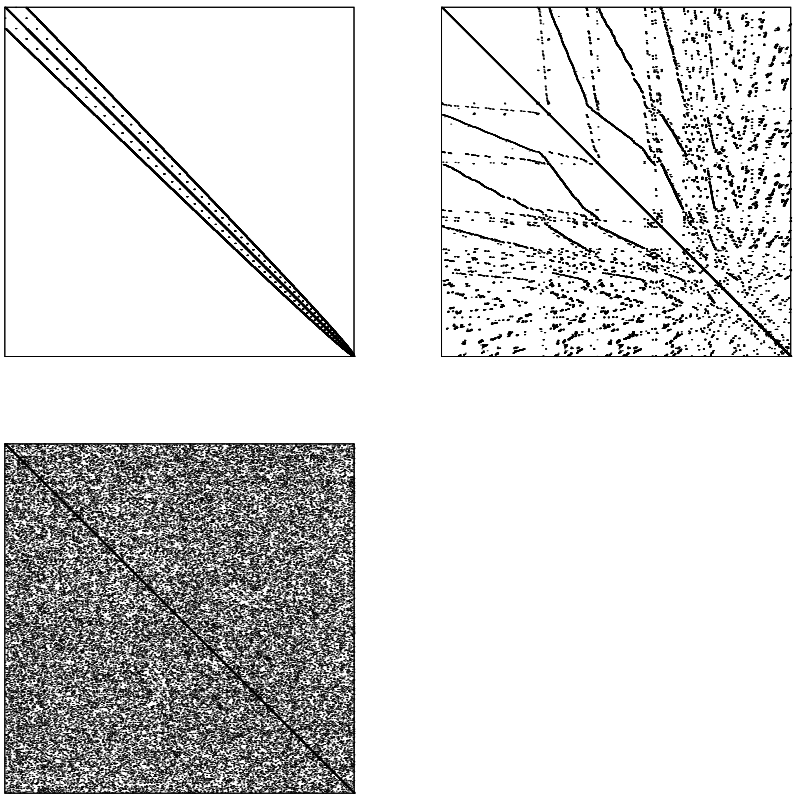


Figure 5.1: Matrix patterns for the Laplace equation using Cuthill-McKee (top-left), minimal-degree (top-right), and random ordering of the degrees of freedom.

sparsity patterns. Typically one uses a pattern generated by one of the matrices $A^2, A^3, \ldots$. These matrices are still sparse but have significantly more entries. These modifications are denoted by ILU(k) when using the pattern from $A^{1+k}$, i.e. ILU(0) denotes the "classical" ILU decomposition.

## 5.2 Schur-Complement

Closely related to the LU-decomposition is the Schur-complement. Considering a $(n+m) \times (n+m)$ block-system

$$\begin{bmatrix} A & B \\ D & C \end{bmatrix} \begin{bmatrix} v \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \tag{5.2}$$

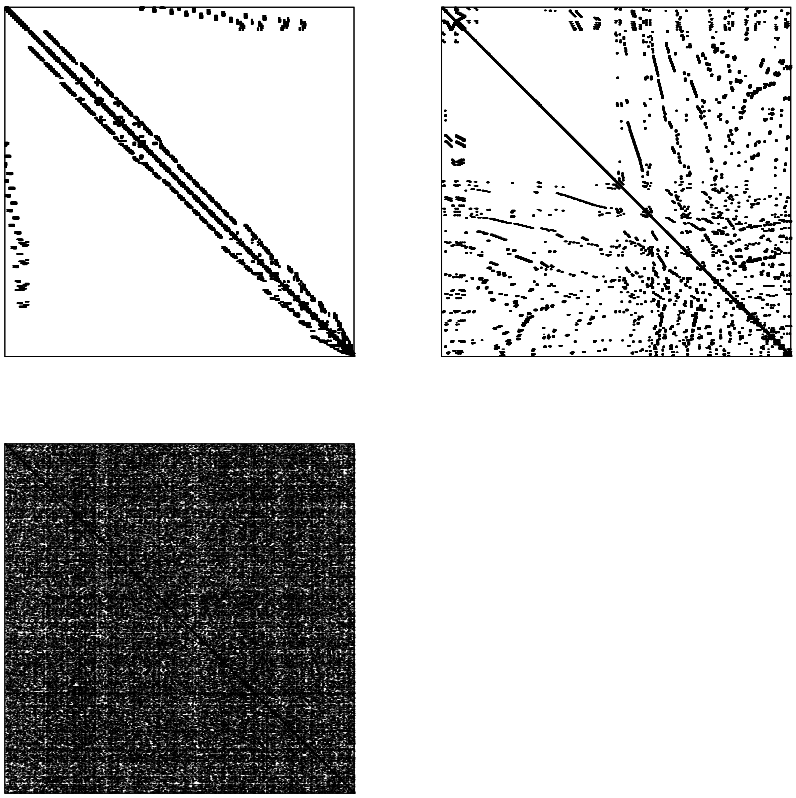


Figure 5.2: Matrix patterns for the FSI equation using Cuthill-McKee (top-left), minimal-degree (top-right), and random ordering of the degrees of freedom.

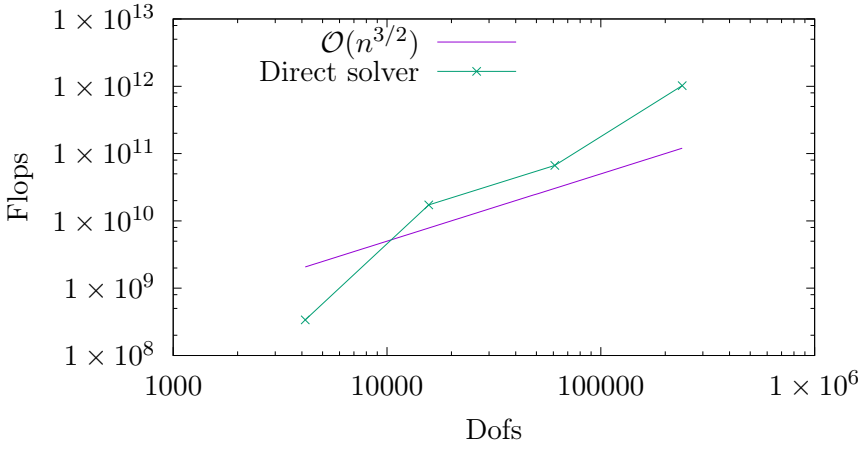



Figure 5.3: Required flops for the sparse direct solver UMFPACK.

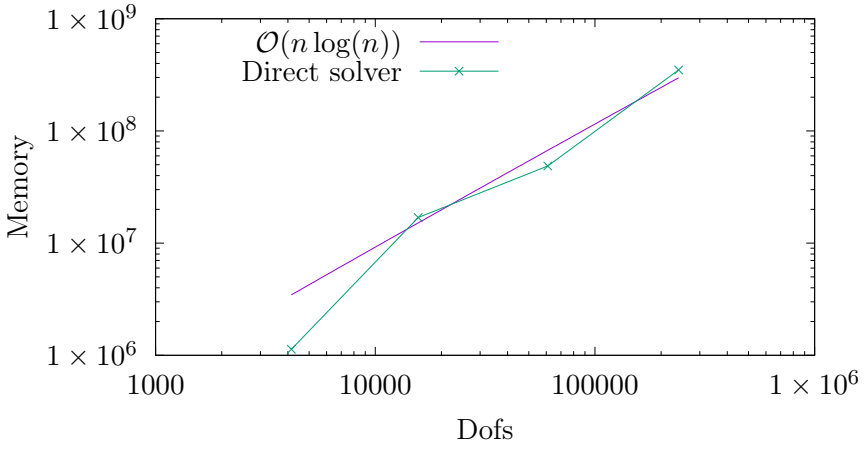



Figure 5.4: Memory usage for the sparse direct solver UMFPACK.

with a non-singular matrix $A$, we obtain by Gaussian elimination (multiplying the first row by $-DA^{-1}$ and add to the second row)

$$\begin{bmatrix} A & B \\ 0 & C - DA^{-1}B \end{bmatrix} \begin{bmatrix} v \\ p \end{bmatrix} = \begin{bmatrix} f \\ g - DA^{-1}f, \end{bmatrix} \tag{5.3}$$

if $S := C - DA^{-1}B \in \mathbb{R}^{m \times m}$ is invertible. This is matrix is usually called the Schur-complement. The modified system can now be solved by computing $Sp = g - DA^{-1}f$ and $Av = f - Bp$. Therefore, the solution of the original $(n+m) \times (n+m)$ system reduces to the solution of a $m \times m$ and $n \times n$ system, which is generally cheaper.

## 5.3 GMRES

In our application, we will use a preconditioned GMRES method [23] for solving the linear systems $Ax = b$, $A \in \mathbb{R}^{n \times n}, x, b \in \mathbb{R}^n$. In the following section, we will have a short look at the algorithm and some theoretical properties of this method.

Similar to many powerful iterative methods (like CG), the Generalized Minimal Residual Method (GMRES) is a Krylov subspace method. This means, that in every iteration the residual is minimized over an increasing subspace. More precisely, for the $m$-th step, the computed solution $x_m$ is a minimizer of $\|Ax - b\|_{l^2}$ over the space

$$x_0 + \mathcal{K}_m(A, r_0) = x_0 + \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\},$$

leading to the exact solution after at most $n$ steps when using exact arithmetic. The big advantage of GMRES compared to CG is, that it can also cope with matrices that are not necessarily symmetric or positive definite.

**Algorithm 5** GMRES without preconditioning for solving $Ax = b$

1: $x_0 =$ initial guess, $r_0 = b - Ax_0$, $\gamma_1 = \|r_0\|$, $v_1 = \dfrac{r_0}{\gamma_1}$
2: **for** $j = 1, 2, \ldots$ **do** until convergence
3: $\quad h_{i,j} = (Av_j, v_i)$, for $i = 1, \ldots, j$
4: $\quad w_j = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
5: $\quad h_{j+1,j} = \|w_j\|$
6: $\quad$ **for** $i = 1, \ldots, j-1$ **do**
7: $\quad\quad \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix} = \begin{bmatrix} c_{i+1} & s_{i+1} \\ -s_{i+1} & c_{i+1} \end{bmatrix} \cdot \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$
8: $\quad$ **end for**
9: $\quad \beta = \sqrt{h_{j,j}^2 + h_{j+1,j}^2}$
10: $\quad s_{j+1} = \dfrac{h_{j+1,j}}{\beta}$
11: $\quad c_{j+1} = \dfrac{h_{j,j}}{\beta}$
12: $\quad h_{j,j} = \beta$
13: $\quad \gamma_{j+1} = -s_{j+1}\gamma_j$
14: $\quad \gamma_j = c_{j+1}\gamma_j$
15: $\quad$ **if** $|\gamma_{j+1} > TOL|$ **then** $v_{j+1} = \dfrac{w_j}{h_{j+1,j}}$
16: $\quad$ **else**
17: $\quad\quad$ **for** $i = j, \ldots, 1$ **do**
18: $\quad\quad\quad y_i = \dfrac{1}{h_{j,j}} \left( \gamma_i - \sum_{k=i+1}^{j} h_{i,k} y_k \right)$
19: $\quad\quad\quad x = x_0 + \sum_{i=1}^{j} y_i v_i$
20: $\quad\quad$ **end for**
21: $\quad$ **end if**
22: **end for**

As we can see from algorithm 5, the required memory storage and computational effort increases in every iteration. One way to limit the amount of work is to restart the algorithm

after a fixed number of iterations $m$ (denoted by GMRES(m)). Typical values for $m$ are around $20 - 50$. However, by doing so, one loses the theoretical minimization property and convergence can no longer be guaranteed. Therefore any used preconditioner should keep the number of required iterations as low as possible such that a restart is not necessary.

**Remark 5.3.1** (Flexible GMRES)**.** The algorithm presented above assumes that a possible preconditioner is constant for all iterations. This prohibits the use of many iterative methods, like GMRES itself, within the preconditioner. The Flexible GMRES takes such non-linear preconditioners into account, which comes at the cost of higher memory requirements.

From now on, we refer to the flexible variant when talking about GMRES, unless explicitly specified.

**Remark 5.3.2** (Convergence Criterion)**.** For the GMRES solver we use a relative reduction of $\|r\|_{l^\infty} < 10^{-9}$ as stopping criterion. This is good enough for the Newton solver to converge without too many additional iterations, compared to using a direct solver. Convergence of the Newton solver is achieved once we obtain a residual with $\|r\|_{l^\infty} < 10^{-8}$.

## 5.4 Multi-Grid Methods

A different kind of solvers are the so called multi-grid methods. The main idea among them is to solve the linear system $Ax = b$ rising from the FE discretization on a sequence of grids. Within those, the results from coarser grids are used to enhance the solution on finer grids. To construct such multi-grid methods, we require a few things.

First of all, we need a hierarchy of grids. One way to obtain them is via the refinement of a given grid, which will then be used as the coarsest mesh. Since this method is based on availably geometric information, it is called "Geometric Multi-Grid" (GMG). A different approach is to construct coarser "grids" using only information provided by the matrix $A$ itself, which leads to the "Algebraic Multi-Grid" (AMG) methods. The construction of such algebraic coarsening schemes is a topic on its own, therefore we refer to [24] for an introduction on this issue.

The advantage of AMG over GMG is its flexibility. For GMG, one requires to create the necessary data structures to hold the information from all levels of refinement. Depending on the code, this may be difficult to incorporate later on. AMG methods do not require these structures, since these methods are able to compute the necessary information itself.

However, GMG methods are, if applicable, generally more efficient then AMG.

Once we have a hierarchy from either AMG or GMG, we need to be able to represent the residual of a linear system stemming from the finer grids in terms of coarser grids and vice-versa. The responsible operators for this procedure are the restriction and prolongation operators (matrices)

$$\mathcal{R}_l : \mathbb{R}^{N_l} \mapsto \mathbb{R}^{N_{l-1}} \text{ and } \mathcal{P}_l : \mathbb{R}^{N_{l-1}} \mapsto \mathbb{R}^{N_l} \tag{5.4}$$

where $N_l$ denotes the number of degrees of freedom on level $l$. For GMG methods, these operators can be naturally defined using standard FE interpolation.

However, if we just project the residual onto a coarser mesh, we lose a lot of accuracy if the residual shows a highly oscillating behaviour. The idea is to reduce these high-frequency parts first. If this is successful, we can assume that the vector can be represented on the coarser grid without too much loss of information. An illustrative example is given in figure 5.5. Damping of the oscillations within the residual is achieved via so called smoothers. Mostly, these are iterative solvers lacking a good overall convergence rate but with the property, that they reduce the error in the high-frequency parts very fast. Typical examples of such smoothers include the Jacobi method, (symmetric) Gauss-Seidel, polynomial smoothers like Chebyshev or MLS, but also methods like ILU can be used within multi-grid methods. Once we are on the coarsest mesh, we solve the problem exactly using a (sparse) LU-decomposition described in section 5.1.

A frequently used smoother (in sequential environments) is the Gauss-Seidel method, which we will also use later on. Its application to the linear system $Ax = b$ can be written as

$$x_{k+1} = (D + L)^{-1}(b - Ux_k), \tag{5.5}$$

with the decomposition $A = L + U + D$ into a lower-, upper-triangular and diagonal part and some initial guess $x_0$.

Combining all the single components introduced above, we can now state a general recursive multi-grid application in algorithm 6. Here, $\mathcal{S}(A_l, x_l, b_l)$ denotes the smoother depending on the level $l$ matrix $A_l$, an initial guess and output $x_l$ and corresponding right-hand-side $r_l$. Typical values for $\gamma$ are 1 or 2. The resulting cycles are referred to as V-cycle and W-cycle due to its characteristic shape as seen in figure 5.6.

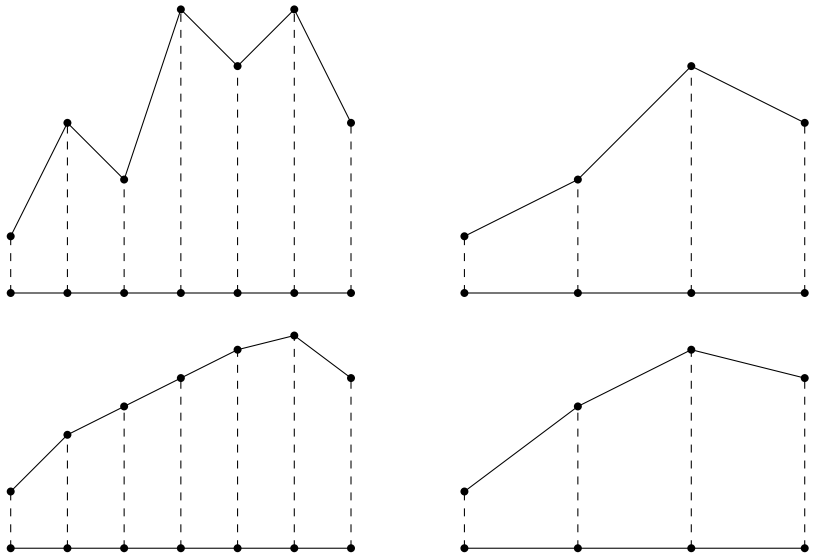


Figure 5.5: Schematic 1-d Multi-grid illustration of a non-smooth residual (top row) and smoothed version (bottom row) on the fine grid (left) and coarse grid (right). Most of the behaviour of the non-smoothed function cannot be represented on the coarser grid.

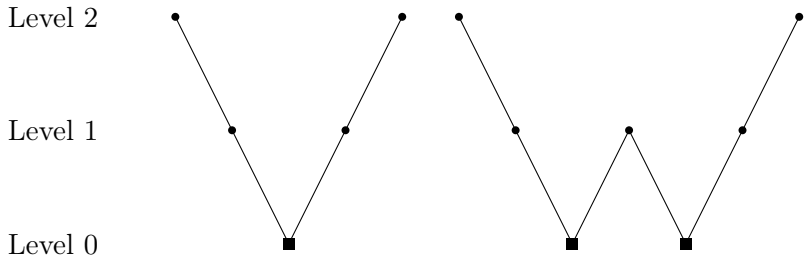



Figure 5.6: V- and W-cycle. Circles denote an application of the smoother, squares correspond to the solution on the coarsest grid.

Similar to the smoothers listed above, multi-grid methods can solve linear systems on their own, yet they are mostly used as efficient preconditioners. In our application, we use the AMG-implementation provided by the Trilinos package ML [12, 15]. Treatment of a geometric multigrid preconditioner for fluid-structure-interaction can be found in e.g. [22].

**Algorithm 6** Multi-Grid Method for Solving $Ax = b$.

**procedure** $MG_l(x_l, b_l)$
  **if** $l = 0$ **then**
    Solve coarsest level $A_l x_l = b_l$
    return
  **end if**
  Pre-smoothing: $x_l = \mathcal{S}(A_l, x_l, b_l)$, $r_l = b_l - A_l x_l$
  Restriction: $r_{l-1} = \mathcal{R}_l r_l$
  Coarse-grid approximation: $w_{l-1} = 0$
  **for** $i = 1$ to $\gamma$ **do**
    $MG_{l-1}(w_{l-1}, r_{l-1})$
  **end for**
  Prolongation: $w_l = \mathcal{P}_l w_{l-1}$
  Correction: $x_l = x_l + w_l$
  Post-smoothing: $x_l = \mathcal{S}(A_l, x_l, b_l)$
**end procedure**

# Chapter 6

# Preconditioners

Iterative solvers like GMRES become most powerful if they are combined with a good preconditioner. The development of such a preconditioner depends very much on the structure of the underlying problem. However, AMG methods turned out to be efficient "black-box" preconditioners. Unfortunately, such purely algebraic methods often do not work well for complicated coupled problems like FSI. Therefore, we can not avoid spending some time into the development of preconditioners that specifically target FSI problems.

Before we present the final results of this work, we will have a look at some general ideas that apply to many different preconditioners. Then we combine some of these ideas into a first simple working preconditioner before we finally arrive at a more sophisticated one.

## 6.1 Introduction

The principal idea of preconditioners is, that a system of linear equations $Ax = b$ can be rewritten as $P^{-1}Ax = b$ (left preconditioning) or $AP^{-1}u, x = P^{-1}u$ (right preconditioning), with an arbitrary but non-singular operator $P$. However, there is no need to compute $P$ or $P^{-1}$ explicitly. It suffices to be able to compute the action of $P^{-1}$ to a given vector.

Now we can chose $P^{-1}$ such that the preconditioned systems $P^{-1}A$ or $AP^{-1}$ are "easier" to solve, i.e. we would like to have something like $P^{-1}A \approx I$. Of course, this can be achieved by using $P^{-1} = A^{-1}$, which leads to a trivially solvable equation. However, the application of this preconditioner would correspond to solving the original equation, so we do not gain anything from that. The other extremal case would be to chose $P^{-1} = I$. This preconditioner

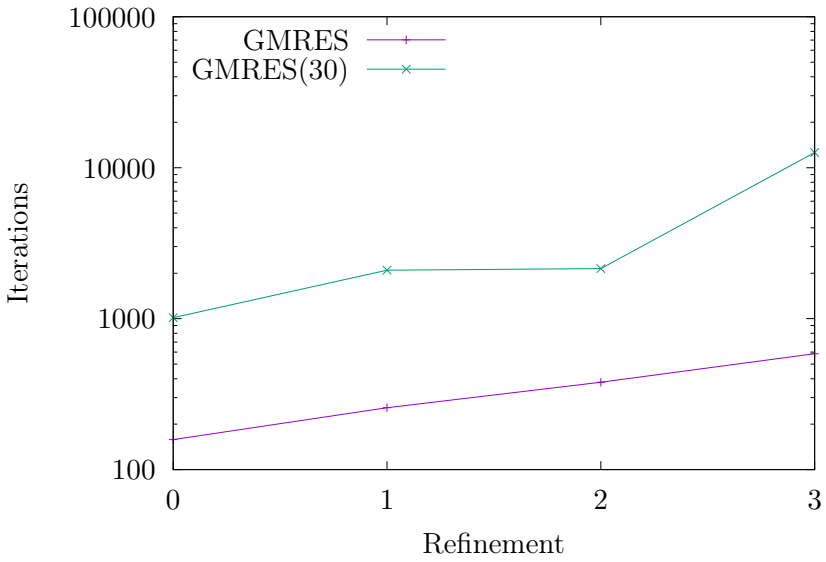



Figure 6.1: Number of iterations using GMRES and restarted GMRES(30) with ILU(0)-preconditioning.

is of course very cheap to apply, but it does not make the linear system easier to solve. While these two choices for $P^{-1}$ have no practical use, they illustrate the problem of a good choice of the preconditioner:

On the one hand, its application should be as cheap as possible while on the other hand, it should "simplify" the equations a lot.

**Remark 6.1.1.** The meaning of "simplify" depends on the actual solver used. For example a direct solver benefits from a block-diagonal structure and low bandwidth of $P^{-1}A$ (see section 5.1.1), whereas iterative solvers like GMRES work best if the eigenvalues are clustered into just a few blocks

## 6.2 First Examples

First of all, we require a handleable block representation of the Jacobian, since the structure given in 4.23 is way to detailed to work with. This step already differs depending on how the resulting preconditioner is supposed to work. Some examples of different block notations of

the same matrix are given below (see [14, 20]):

$$A \equiv \begin{bmatrix} \mathcal{M} & \mathcal{C}_{ms} & 0 \\ \mathcal{C}_{sm} & \mathcal{S} & \mathcal{C}_{sf} \\ \mathcal{C}_{fm} & \mathcal{C}_{fs} & \mathcal{F} \end{bmatrix} \tag{6.1}$$

or

$$A \equiv \begin{bmatrix} \mathcal{S} & \mathcal{C}_{sf} & \mathcal{C}_{sp} \\ \mathcal{C}_{fs} & \mathcal{F} & \mathcal{C}_{fp} \\ \mathcal{C}_{ps} & \mathcal{C}_{pf} & 0 \end{bmatrix} \tag{6.2}$$

Note that the first representation differentiates between mesh, solid and fluid (velocity and pressure) while the latter splits the degrees of freedom into solid, pressure and fluid (including mesh displacement and velocity).

A first idea shown in [14] considers replacing such a block matrix by an approximation. Several possibilities for such an approximation of (6.2) are given. All of these neglect some parts of the coupling terms to obtain the simpler systems (6.3).

$$P_{diag} := \begin{bmatrix} \mathcal{S} & 0 & 0 \\ 0 & \mathcal{F} & \mathcal{C}_{fp} \\ 0 & \mathcal{C}_{pf} & 0 \end{bmatrix} \quad P_{sup} := \begin{bmatrix} \mathcal{S} & 0 & 0 \\ \mathcal{C}_{fs} & \mathcal{F} & \mathcal{C}_{fp} \\ \mathcal{C}_{ps} & \mathcal{C}_{pf} & 0 \end{bmatrix} \quad P_{sub} := \begin{bmatrix} \mathcal{S} & \mathcal{C}_{sf} & \mathcal{C}_{sp} \\ 0 & \mathcal{F} & \mathcal{C}_{fp} \\ 0 & \mathcal{C}_{pf} & 0 \end{bmatrix} \tag{6.3}$$

These simplified block systems can be solved a lot easier than the original one. For example, the application of $P_{sup}^{-1}$ on a vector $r = (r_s, r_f, r_p)$ is given in algorithm 7.

**Algorithm 7** Evaluation of $x = P_{sup}^{-1} r$

1: Solve $\begin{bmatrix} \mathcal{F} & \mathcal{C}_{fp} \\ \mathcal{C}_{pf} & 0 \end{bmatrix} \cdot \begin{bmatrix} x_f \\ x_p \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \end{bmatrix}$

2: Update $r_s = rs - \mathcal{C}_{sf} x_f - \mathcal{C}_{sp} x_p$

3: Solve $\mathcal{S} x_s = r_s$

This approximate solver can then be used as a preconditioner for the original system (6.2) inside GMRES. However, it remains to find the solutions or some approximations of the occurring fluid and solid subsystems.

If we naively apply this technique using our ALE-formulation, this preconditioner is less useful, as we have the variables for the mesh-motion as a part of the fluid block. For problems like the FSI benchmark given in section 7.1, this is less useful as we have a very small solid system $\mathcal{S}$ (only the thin flag) but a very big fluid system.

**Remark 6.2.1.** In [14], it is also considered to replace the exact Jacobian $A$ by one of its approximations, i.e. some kind of quasi-Newton method. However, this requires a lot more Newton iterations and is hence not considered in this work.

## 6.3 LU - Approach

A generalization of the above method is based on a block-wise approximate LU-decomposition. For a first starting point, we have a look at the LU-factorization of a 3x3 matrix with a zero entry in the top-right corner given in equation (6.4). This corresponds to the same structure as our FSI-matrix (6.1).

$$
\begin{aligned}
L &= \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & -\frac{A_{12}A_{21}}{A_{11}} + A_{22} & 0 \\ A_{31} & -\frac{A_{12}A_{31}}{A_{11}} + A_{32} & -\frac{A_{23}\left(-\frac{A_{12}A_{31}}{A_{11}} + A_{32}\right)}{-\frac{A_{12}A_{21}}{A_{11}} + A_{22}} + A_{33} \end{bmatrix} \\
U &= \begin{bmatrix} 1 & \frac{A_{12}}{A_{11}} & 0 \\ 0 & 1 & \frac{A_{11}A_{23}}{-A_{12}A_{21} + A_{11}A_{22}} \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{6.4}
$$

**Remark 6.3.1** (Computation)**.** The factorization was computed using a computer algebra system. This program assumes that the entries of the matrix are scalars, therefore we have to interpret fractions as multiplications with the inverse of the denominator. Further, we have to multiply the matrices in the correct order, which is not the case in equation (6.4).

While this exact LU-decomposition is too complicated to apply to the FSI block-matrix, we can simplify it in the same manner as before by dropping one of the off-diagonal blocks.

In [20], the coupling block $\mathcal{C}_{sm}$ (corresponding to $A_{21}$) does not appear, which leads to the

factorization

$$A \approx P_{21} := \begin{bmatrix} \mathcal{M} & \mathcal{C}_{ms} & 0 \\ 0 & \mathcal{S} & \mathcal{C}_{sf} \\ \mathcal{C}_{fm} & \mathcal{C}_{fs} & \mathcal{F} \end{bmatrix} = \begin{bmatrix} \mathcal{M} & 0 & 0 \\ 0 & \mathcal{S} & 0 \\ \mathcal{C}_{fm} & \tilde{\mathcal{C}}_{fs} & \mathcal{X} \end{bmatrix} \begin{bmatrix} I & \mathcal{M}^{-1}\mathcal{C}_{ms} & 0 \\ 0 & I & \mathcal{S}^{-1}\mathcal{C}_{sf} \\ 0 & 0 & I \end{bmatrix} =: LU \tag{6.5}$$

with

$$\tilde{\mathcal{C}}_{fs} = \mathcal{C}_{fs} - \mathcal{C}_{fm}\, \mathcal{M}^{-1}\, \mathcal{C}_{ms} \tag{6.6}$$

and the fluid Schur complement

$$\mathcal{X} = \mathcal{F} - \tilde{\mathcal{C}}_{fs}\, \mathcal{S}^{-1}\, \mathcal{C}_{sf} = \mathcal{F} - (\mathcal{C}_{fs} - \mathcal{C}_{fm}\, \mathcal{M}^{-1}\, \mathcal{C}_{ms})\, \mathcal{S}^{-1}\, \mathcal{C}_{sf}. \tag{6.7}$$

Now that we have such a decomposition it is easy to compute the action of the inverse. From linear algebra we know that $P^{-1}r = U^{-1}L^{-1}r$. Consecutively solving with $L$ and $U$ finally leads to algorithm 8.

**Algorithm 8** Evaluation of $(LU)^{-1}r$.

1: Solve $x_m = \mathcal{M}^{-1}r_m$
2: Solve $x_s = \mathcal{S}^{-1}r_s$
3: Update $r_f = r_f - \mathcal{C}_{fm}x_m - \mathcal{C}_{fs}x_s$
4: Solve $x_f = \mathcal{X}^{-1}r_f$
5: Update $x_s = x_s - \mathcal{S}^{-1}\mathcal{C}_{sf}x_f$
6: Update $x_m = x_m - \mathcal{M}^{-1}\mathcal{C}_{ms}x_s$

**Remark 6.3.2** (Connection to Partitioned Methods)**.** Such preconditioners based on a LU-factorization can be seen as one (approximate) iteration of a partitioned solver. The off-diagonal coupling blocks correspond to the incorporation of the results from one sub-problem into the other. Depending on which block we neglect, we get slightly different preconditioners. In this work, we will have a look at the preconditioners resulting from dropping $\mathcal{C}_{sm}(A_{21})$, $\mathcal{C}_{sf}(A_{23})$ and $\mathcal{C}_{ms}(A_{12})$. We will denote these preconditioners by the indices $21, 23$ and $12$ respectively.

While this is basically the final preconditioner, we still have to clarify several things. First of all, we do not necessarily require the exact solution of the interior sub-problems - a "good-enough" approximation may be enough. Nevertheless, we can use a direct solver for these sub-problems in order to get a first idea of the capabilities of the preconditioner. It is reasonable to assume that the required GMRES iterations when using direct solvers are the lowest we can get from this particular preconditioner. Any approximate sub-solution will most likely increase the number of iterations. However, as observed in [14], it is possible that the approximate version performs better than the exact one. (Un)fortunately, we were not able to observe such a behaviour in any of the tests.

Although using direct solvers is not what we want at the very end due to their huge memory requirements, we still gain something. Since the computational effort of direct methods increases in a non-linear fashion, solving three smaller systems is cheaper than computing the solution of the whole system. Hence we might be able to solve larger problems than previously before we run out of memory. Further, this preconditioner with direct solvers for the sub-problems is already significantly faster than a direct solver alone (see figure 6.2).

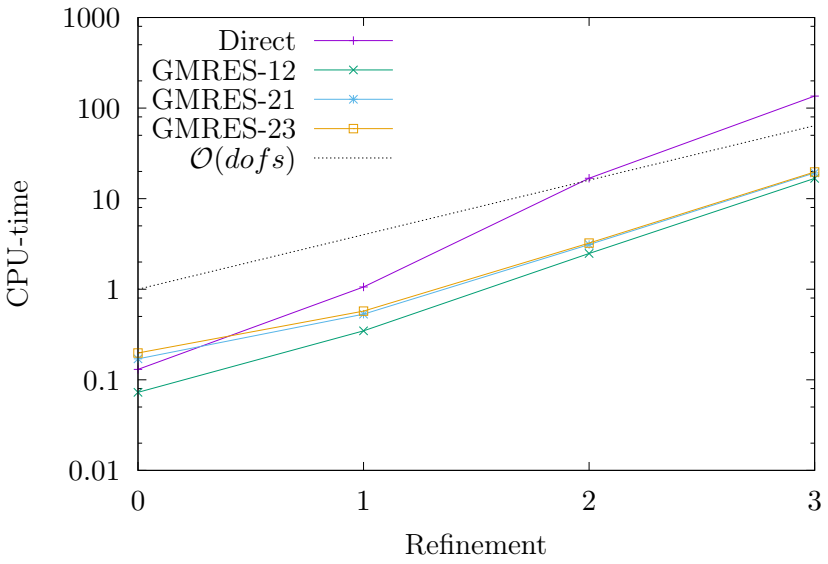



Figure 6.2: Required time using a direct solver and GMRES with direct sub-solvers.

A small benefit when using direct solvers is that we only have to factorize the sub-problems once every time the matrix changes. Thus, only the very first GMRES iteration is costly, while the effort for all subsequent iterations (and probably the next Newton steps as well) is relatively low (matrix-vector products and forward / backward substitution).

If we used iterative solvers for the approximation, every application of the preconditioner has approximately equal cost (there are still things that can be precomputed, like the coarsening strategy in AMG methods, but these do not contribute as much as a complete factorization). A closer look on the approximate solution of the sub-problems follows in section 6.4.

Another point worth a closer look is the computation of $\tilde{\mathcal{C}}_{fs}$ and the Schur complement $\mathcal{X}$. Both involve matrix-matrix multiplications and even-worse, require the inverse of $\mathcal{M}$ and $\mathcal{S}$. If we use a direct solver or AMG method for the Schur complement, we have to compute it explicitly since we need to access single entries of the matrix. Thus, we would also need $\mathcal{M}^{-1}$ and $\mathcal{S}^{-1}$ in explicit form! Of course, this is not feasible to do, since the inverse is in general not sparse any more and the computational effort is way to much. The solution to this problem is the same as in the previous sections: we have to approximate these matrices respectively its inverses. One possible way of doing so is to chose a diagonal approximation like

$$\tilde{A}(i,i) := \sum_{c} |A(i,c)| \tag{6.8}$$

for $\mathcal{M}$ and $\mathcal{S}$, which can then be easily inverted. Another choice proposed in [20] is a block-diagonal version of the original matrix with low enough block-size (e.g. 3), such that each diagonal block can still be inverted explicitly. The drawback of this approach is that the resulting approximation depends on the ordering of the degrees of freedom, which is not the case for the diagonal row-sum approximation. Also, the diagonal approach seems to perform better in most of our tests.

It turns out that we can approximate the Schur complement even more by neglecting the perturbation term $\tilde{\mathcal{C}}_{fs}\ \mathcal{S}^{-1}\ \mathcal{C}_{sf}$ completely. As shown in figure 6.3, the average number of GMRES iterations is almost the same whether we compute the Schur complement or simply take the fluid matrix $\mathcal{F}$. The computations were done for the FSI 2 benchmark with time-step-size $\Delta t = 0.01$ using direct solvers for the sub-problems. This approximation saves us the expensive matrix-matrix multiplication for the Schur complement. However, it is unclear whether this rough approximation will do well for all kind of different problems.

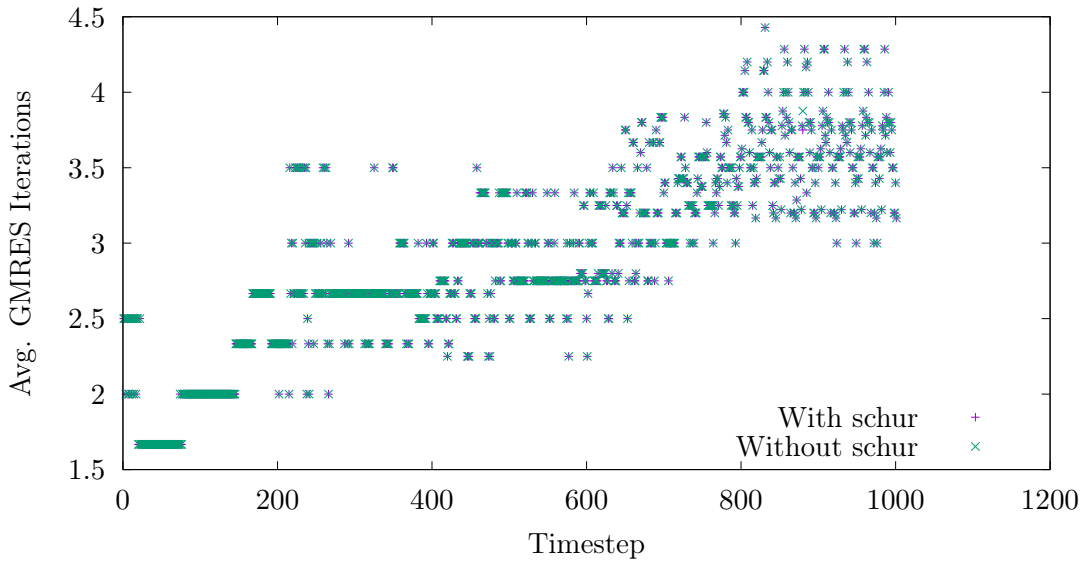



Figure 6.3: Similiar number of GMRES iterations with and without computing the Schur complement for the FSI 2 benchmark at each time-step.

## 6.4 Solving the Sub-Problems

In the following sections we will have a look at the approximate solution of the arising sub-problems $\mathcal{M}, \mathcal{S}$ and $\mathcal{X}$. Ideally, we want to obtain a good approximation with low computational effort that can eventually be run in parallel. Unfortunately, it is quite challenging to find solvers that fit all these requirements.

### 6.4.1 Mesh

For the solution of the mesh system, we apply a few AMG v-cycles with Gauss-Seidel smoothing. This is motivated by the fact that for small deformations, the Jacobian of the mesh-equation reduces to $\left(\hat{\nabla}\hat{\varphi}_f^u \, , \, \hat{\nabla}\hat{\varphi}_f^u\right)_{\hat{\Omega}_f}$. This then corresponds to the usual Laplace equation in weak form, which is known to be efficiently solved by this method. The approximation above is quite accurate, as the mesh-deformation takes place close to the interface and vanishes further away. For large displacements, we may use the AMG cycles as a preconditioner for another GMRES solver, to improve the convergence.

### 6.4.2 Solid

For the solution of the elasticity problem, we perform a Schur complement approach as described in section 5.2, by splitting the degrees of freedom into velocity in displacement parts. This yields the modified solid system

$$\begin{aligned} \mathcal{S} &= \begin{bmatrix} \mathcal{S}_{uu} & \mathcal{S}_{uv} \\ \mathcal{S}_{vu} & \mathcal{S}_{vv} \end{bmatrix} \begin{bmatrix} x_u \\ x_v \end{bmatrix} = \begin{bmatrix} r_u \\ r_v \end{bmatrix} \\ \Leftrightarrow \mathcal{P}_S &:= \begin{bmatrix} \mathcal{S}_{uu} & \mathcal{S}_{uv} \\ 0 & \mathcal{X}_S \end{bmatrix} \begin{bmatrix} x_u \\ x_v \end{bmatrix} = \begin{bmatrix} r_u \\ r_v - \mathcal{S}_{vu} r_u \end{bmatrix} \end{aligned} \tag{6.9}$$

with the Schur complement (approximation) $\mathcal{X}_S = \mathcal{S}_{vv} - \mathcal{S}_{vu}\mathcal{S}_{uu}^{-1}\mathcal{S}_{uv}$. Again, we approximate $\mathcal{S}_{uu}^{-1}$ by a diagonal version, but this time we just extract the diagonal element $\tilde{\mathcal{S}}_{uu}(i,i) := \mathcal{S}_{uu}(i,i)$. Now we can apply AMG v-cycles to the "sub-sub-problems" $\mathcal{S}_{uu}$ and $\mathcal{S}_{vv}$. To improve the approximation, we do not use this decomposition $\mathcal{P}_S$ directly, but employ it as a preconditioner inside an additional GMRES solver. Note that most of the time we only need a few iterations to significantly enhance the quality of the approximation. Contrary to the previous global Schur-complement, we can not neglect the computation of $\mathcal{X}_S$ without requiring a lot more iterations.

**Remark 6.4.1** (Nested Schur-Complements)**.** Principally, we could perform another decomposition for the matrices $\mathcal{S}_{uu}$ and $\mathcal{S}_{vv}$ by separating between interface and interior degrees of freedom (or even further, differentiating between single coordinates). However, this worsens the approximation quality and leads to a significant increase in the number of outer GMRES iterations, even when direct solvers are used for the smallest problems.

### 6.4.3 Fluid

The most challenging part is the robust solution of the fluid equations. Solving this linear system requires special care as this matrix is indefinite and includes a zero block. Precisely, we have to solve the following equation

$$\mathcal{F} := \begin{bmatrix} \mathcal{F}^{\Omega\Omega} & \mathcal{F}^{\Omega}_{vp} \\ \mathcal{F}^{\Omega}_{pv} & C \end{bmatrix} \begin{bmatrix} x_v \\ x_p \end{bmatrix} = \begin{bmatrix} r_v \\ r_p \end{bmatrix}.$$

Several methods have been developed for the (linear) Stokes problem, like Vanka-type methods, Braess-Sarazin smoother and Uzawa-like methods (see e.g. [5, 34, 26]). All of these methods are closely related to the Schur-complement, and differ mainly in the choice of arising approximations.

In our application, we will use the same approach as for the solid equations. This provided the most robust results, but there are still situations where this does not yield a good convergence.

# Chapter 7

# Numerical Results

Last but not least, we present some numerical results for the preconditioned solver. Here, we mainly focus on the well-known 2-d FSI benchmarks and some variations. These test-problems are presented in the next section. The implementation uses the deal.II FE-library [4] and the Trilinos [15, 12] packages therein for the AMG method. For the direct solution of linear systems the package UMFPACK [8] was used. The basis of the FSI implementation was provided by Thomas Wick in [32] and was adapted using a hp-element formulation for more flexibility.

## 7.1 FSI Benchmarks

Frequently used test cases for FSI solvers are the benchmarks proposed by Turek and Hron in [27]. These problems consist of an incompressible laminar channel flow around a cylinder, with an elastic beam (flag) attached to it (see figure 7.1).

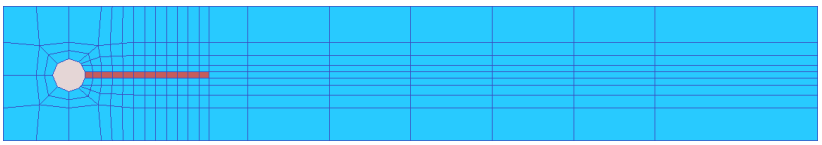


Figure 7.1: Meshed computational domain with the fluid domain (blue) and solid domain (red). Cylinder (grey) is not part of the domain.

Originally, three problems with different choices for the fluid and structure parameters were defined:

| Parameter | Symbol / Unit | FSI 1 | FSI 2 | FSI 3 |
|---|---|---|---|---|
| Structure density | $\rho_s[kg/m^3]$ | $10^3$ | $10^4$ | $10^3$ |
| Poisson ratio | $\nu_s$ | 0.4 | 0.4 | 0.4 |
| Lamé coefficient | $\mu_s[kg/m^2]$ | $0.5 \times 10^6$ | $0.5 \times 10^6$ | $2.0 \times 10^6$ |
| Fluid density | $\rho_f[kg/m^3]$ | $10^3$ | $10^3$ | $10^3$ |
| Kinematic viscosity | $\nu_f[m^2/s]$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ |
| Inflow velocity | $v_{in}[m/s]$ | 0.2 | 1.0 | 2.0 |
| Channel length | $L := 2.5m$ | | | |
| height | $H := 0.41m$ | | | |
| Cylinder center | $M := (0.2m, 0.2m)$ | | | |
| radius | $r := 0.05m$ | | | |
| Flag width | $w := 0.35m$ | | | |
| height | $h := 0.02m$ | | | |
| Measurement point | $A := (0.6m, 0.2m)$ | | | |

The point $A$ denotes the right-most point of the flag at which the displacements are measured.

The initial condition is given as the inflow profile at the left boundary

$$v(0, y) := 1.5 \; v_{in} \; \frac{4y(H - y)}{H^2},$$

with an additional smoothing of the form

$$v(t, 0, y) := \begin{cases} v(0, y) \; \frac{1}{2} \left(1 - cos(\frac{\pi}{2} t)\right) & \text{for } t < 2 \\ v(0, y) & \text{for } t \geq 2 \end{cases}$$

for FSI 2 and 3. This enhances the convergence of the Newton-solver for the first time-steps. Furthermore we have no-slip conditions $v_f = 0$ on the top and bottom boundary part of the channel, the circle boundary and on the fluid-solid interface. On the outflow boundary, we have the do-nothing condition as described in section 3.3.

The flag is not exactly centred in y-direction, which leads to differences in the forces acting on the flag from above and below. For FSI 3, the inflow velocity is large enough such that the fluid develops oscillations on its own. Within the FSI 2 benchmark, this behaviour results from the interaction between fluid and solid, i.e. the small pressure differences caused by the fluid move the flag, which in turn increases these differences even more. Figure 7.2 shows this

displacement of the flag as well as the velocity profile and figure 7.3 gives a more detailed view of the periodic behaviour.

For FSI 1, (almost) no such oscillations occur, which makes it a lot easier to solve. The solution of this benchmark converges to a stationary solution.

The most challenging benchmark is FSI 2, since the inflow velocity is quite high but the structure is still very flexible. This leads to large deformations, which can be problematic for many solution methods. For FSI 3, the Lamé-coefficient $\mu$ is larger, leading to smaller deformation despite having the highest flow velocity.

**Remark 7.1.1** (Time-Step Sizes)**.** Due to its different characteristics, the various FSI benchmarks require according time-step sizes. For FSI 1, $\Delta t = 1.0$ is good enough, since this simulation is almost stationary. FSI 2 demands $\Delta t \approx 0.01$ in order to produce the correct results, whereas step-sizes for the last benchmark should be around $\Delta t \approx 0.001$ to cope with the high velocity and fast oscillations.

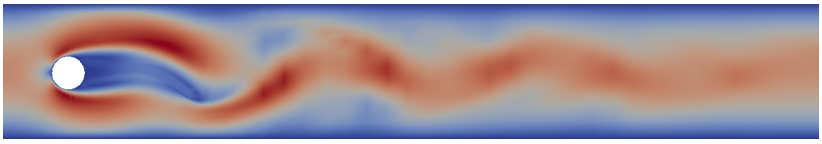


Figure 7.2: Velocity profile of FSI 2 at approx. $t = 9s$.

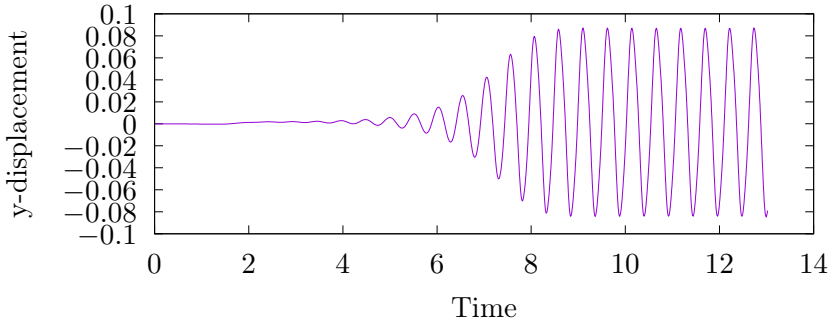



Figure 7.3: y-displacement of the flag for FSI 2.

## 7.2 Using Direct Solvers for the Sub-Problems

In our first tests, we solve the sub-problems with a direct solver to get an idea of the behaviour of the preconditioners.

### 7.2.1 Oscillations

As it can be seen in figure 7.4, the GMRES iterations tend to mimic the oscillatory behaviour of the flag for the FSI 2 benchmark.

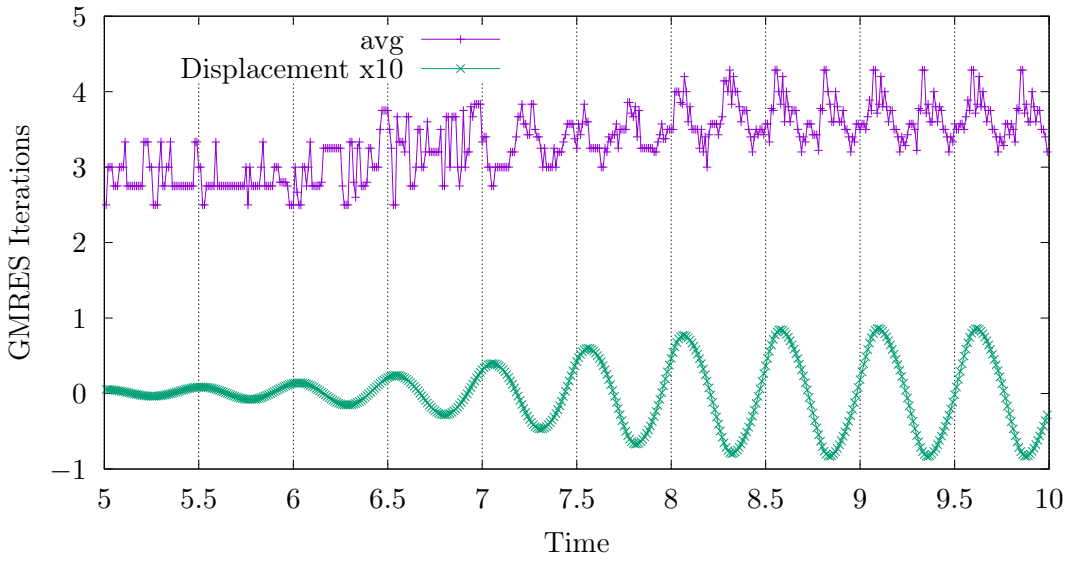



Figure 7.4: GMRES iterations for the FSI 2 benchmark and the 21-preconditioner. The number of iterations behaves similar as the y-displacement of the flag. Displacement has been scaled by a factor of 10.

### 7.2.2 Dependence on Refinement and Time-Step Size

One important property of a preconditioner is its robustness with respect to the mesh-size $h$ and time-step size $\Delta t$. As we can observe in figure 7.5, the number of GMRES iterations remains almost constant after spatial refinement, but show a small dependence on $\Delta t$. All these tests shown were done for the FSI 2 benchmark.

### 7.2.3 Dependence on Material Parameters

Furthermore, we need to be able to run tests with different kind of material configurations. Due to its relation to partitioned methods, it is also interesting to investigate the added-mass effect on the preconditioner. As we can see in figure 7.6, we are not completely independent of the choice of parameters. Nevertheless the number of required iterations remains quite small.

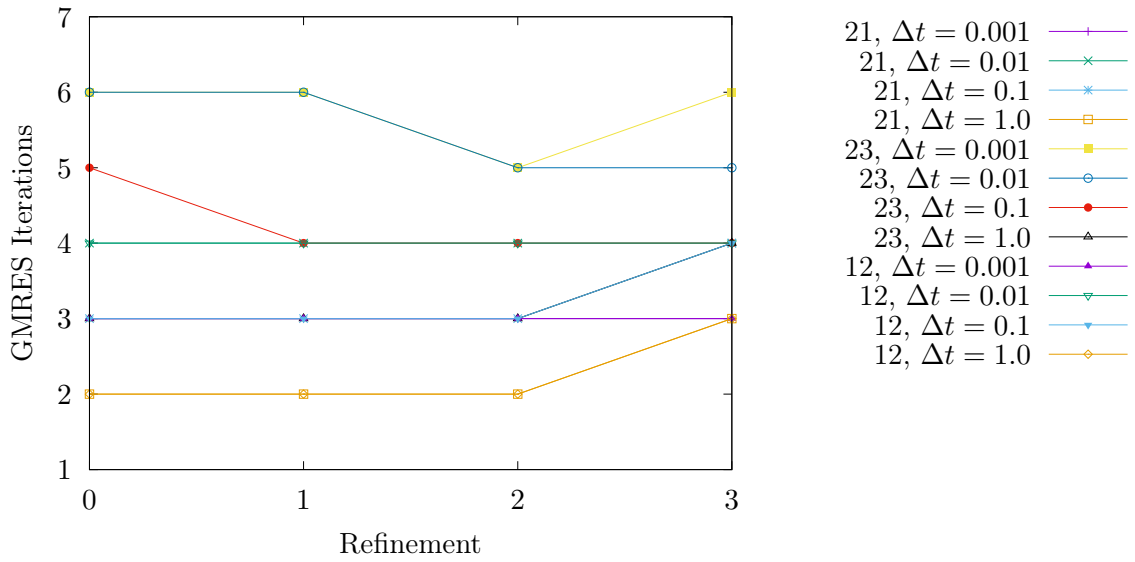



Figure 7.5: Number of GMRES iterations under $h$-refinement for different time-step values using the 21/23/21-preconditioners with direct solvers for sub-problems.

If we use direct solvers for the sub-problems, we are also very robust with respect to a wide range of material parameters (although a lot of those combinations do not make much sense). In figure 7.7 the distribution of iterations among all combinations of parameters given in table 7.1 is shown. The computation was done on the usual FSI mesh with one refinement and the CN time-stepping scheme. We can observe, that the majority of tests require less than 10 iterations. Also, the added-mass effect does not seem to have a large impact on the solver, as indicated by the green bars. All tests requiring more than 20 iterations include the almost incompressible polybutadiene material.

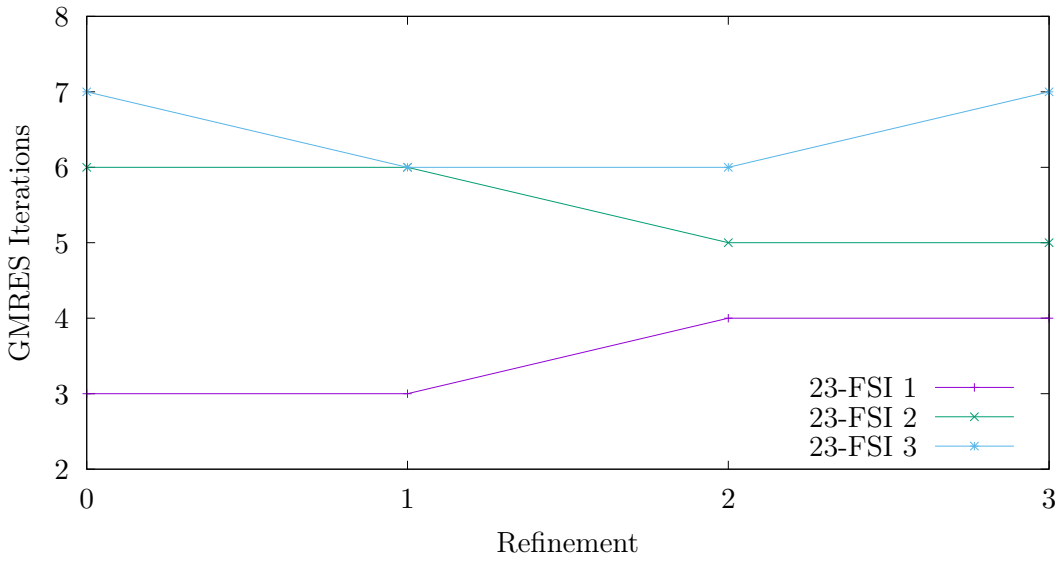



Figure 7.6: Number of GMRES iterations for the different FSI benchmarks using $\Delta t$ as proposed in remark 7.1.1 and the 23-preconditioner. The 21 and 12-preconditioners behave similar.

| Parameter | Values |
|---|---|
| Inflow velocity | 1.0 |
| Fluid | { air, water, glycerol, mercury } |
| Solid | { polybutadiene, polypropylene, cork, fsi2 } |
| Time-step size | $\{10^{-3}, 10^{-2}, 10^{-1}, 1.0\}$ |

| Name | Density | Poisson Ratio | Lamé-$\mu$ |
|---|---|---|---|
| Polybutadiene | $10^3$ | 0.499 | $10^5$ |
| Polypropylene | $10^3$ | 0.4 | $10^8$ |
| Cork | 200 | 0.25 | $10^7$ |
| FSI2 | $10^4$ | 0.4 | $0.5 \cdot 10^6$ |

Table 7.1: Test parameters and materials. The names just indicate a material that fits roughly for the given parameters.

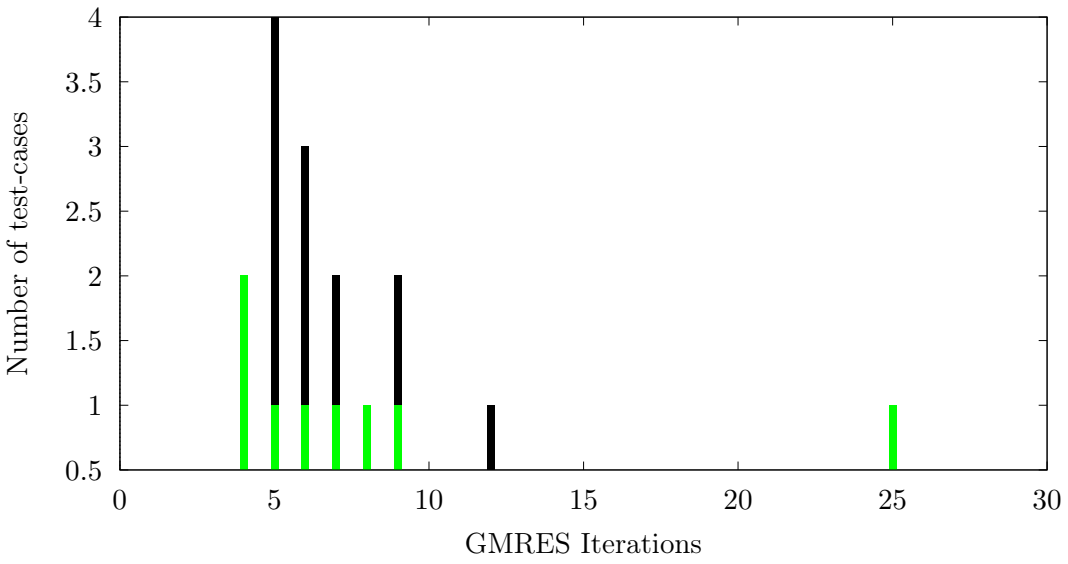



Figure 7.7: Distribution of the number of iterations for all kinds of different parameters on a single refined FSI mesh using direct solvers for the sub-problems. Green bars denote those configurations where $\rho_f = \rho_s$. The 21-preconditioner performs slightly worse.

## 7.3 Iterative Subsolvers

In this section we finally use a fully iterative solver, replacing the direct solvers for the sub-sub-systems of the Schur-complement by iterative variants as described previously.

### 7.3.1 Dependence on Step-Size and Refinement

As we can see in figure 7.8, the time-step size $\Delta t$ has a larger impact for the fully iterative solver. Nevertheless, the number of iterations is still within reasonable bounds.

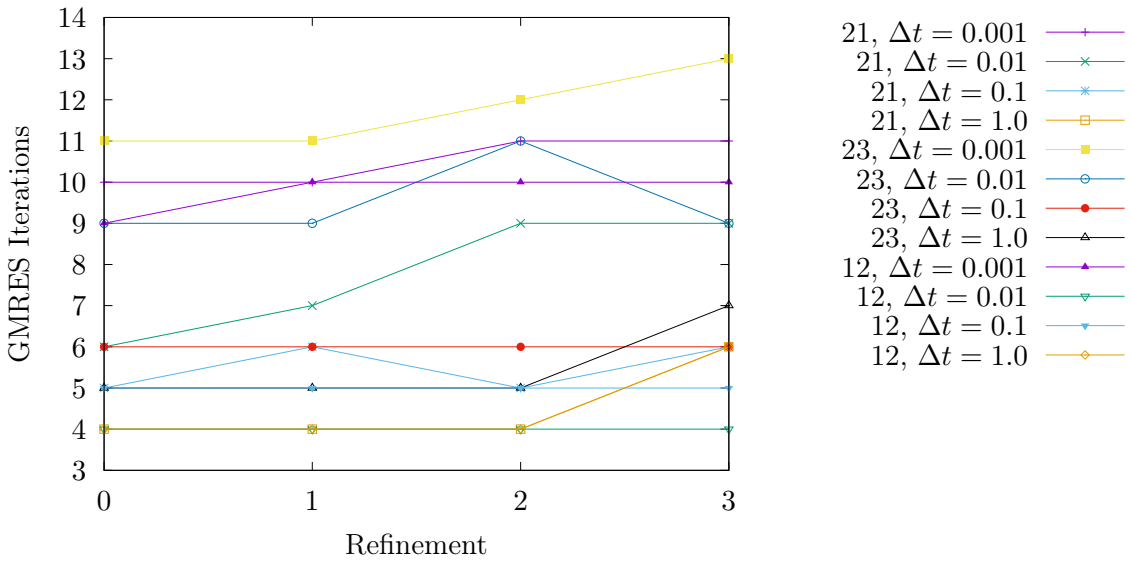



Figure 7.8: Number of GMRES iterations under $h$-refinement for different time-step values using the 21/23/21-preconditioners with iterative solvers for the sub-problems.

### 7.3.2 Material Parameters

In the next test, we vary the structural density of the FSI 2 benchmark by using $\rho_s \in \{10^3, \ldots, 10^8\}$. Figure 7.9 shows that we require more iterations for $\rho_s \approx \rho_f$, indicating that the added-mass effect has some influence on the preconditioner.

Comparing figure 7.7 and figure 7.10, we clearly see that we require more GMRES iterations than with direct solvers. In particular there are more configurations that reach the maximum number of iterations ($= 30$) at some point without convergence. Using iterative methods for

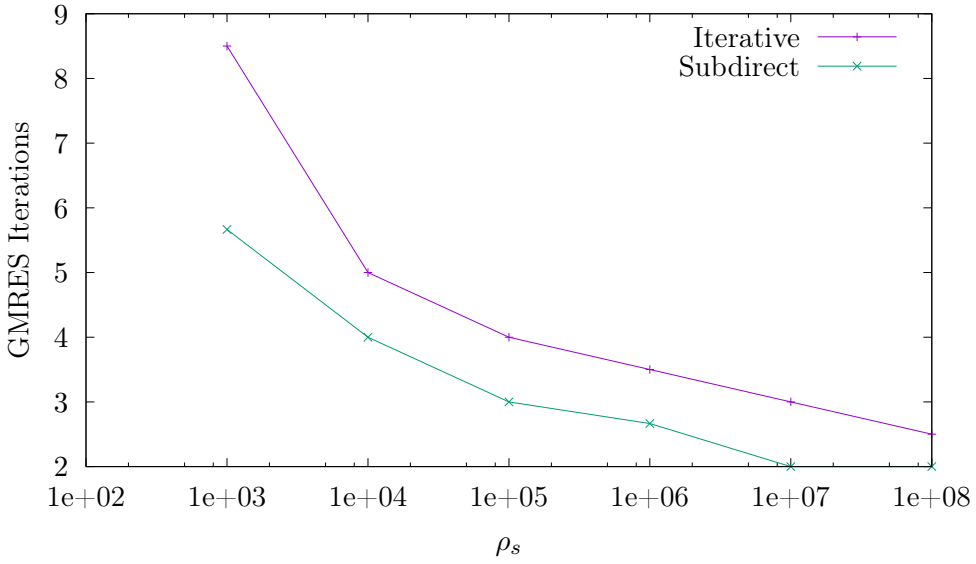



Figure 7.9: Iterations depending on the solid density $\rho_s$ ($\rho_f = 1000$).

the solution of the sub/sub-problems, the GMRES solver faces additional difficulties for several configurations, in particular those with $\Delta t = 1.0$.

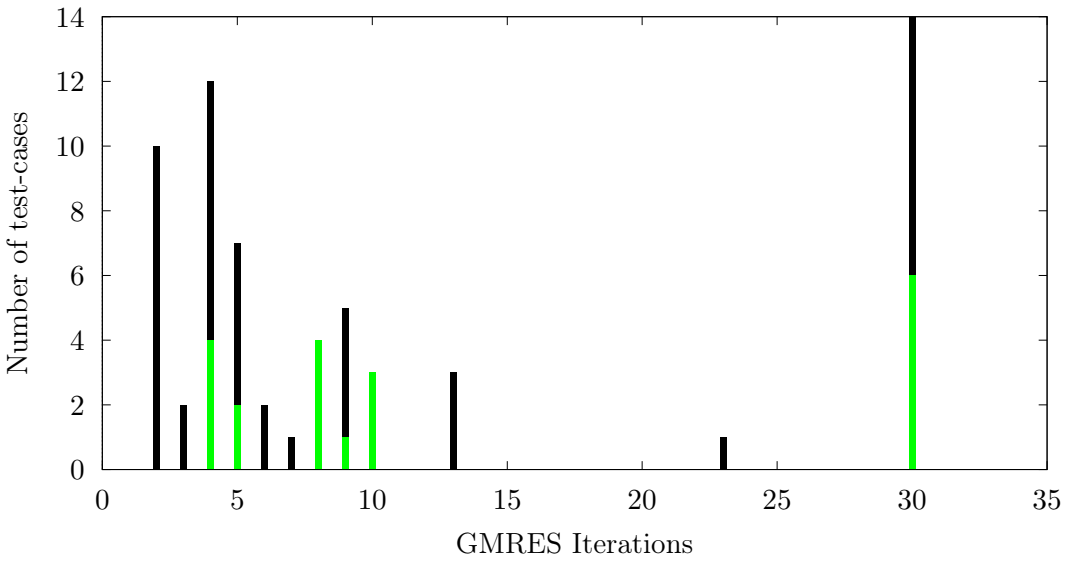



Figure 7.10: Distribution for the completely iterative solution using the 12-preconditioner.

# Chapter 8

# Conclusion

## 8.1 Summary and Comparisons

In this work we adapted the idea presented in [20] to our FSI formulation and discretization. Contrary to [20], we splitted the second order solid equations into a system of first order equations and discretize it using the One-Step-$\theta$-scheme instead of the Newmark scheme. Due to the different interface-coupling strategies used, we do not immediately arrive at a block-structure leading to a usable block-LU factorization. This issue is overcome by neglecting one of the coupling terms, similar to the idea presented in [14], yielding three slightly different preconditioners. From a variety of tests done, the 12-preconditioner seems to be the most robust one. This is particularly interesting, as this one is also the cheapest one to apply.

Furthermore, we neglect the computation of the global Schur-complement completely. This splits the coupled system into its natural sub-problems, which can then be solved by available techniques similar to the partitioned approach. However, it is unclear whether this approximation works for all kind of problem configurations.

The solution of the fluid and solid sub-problems remain challenging. In particular, the robustness of the preconditioner depends very much on the robustness of the solvers for the fluid and structural sub-problems. This makes direct solvers for the sub/sub-problems a viable option. Nevertheless, we were able to reduce the computational time significantly compared to a plain direct solver. The computations were done on a 8 GB laptop, which was just about enough for 3-times refinement ($\approx$ 250.000 dofs) when using a direct solver.

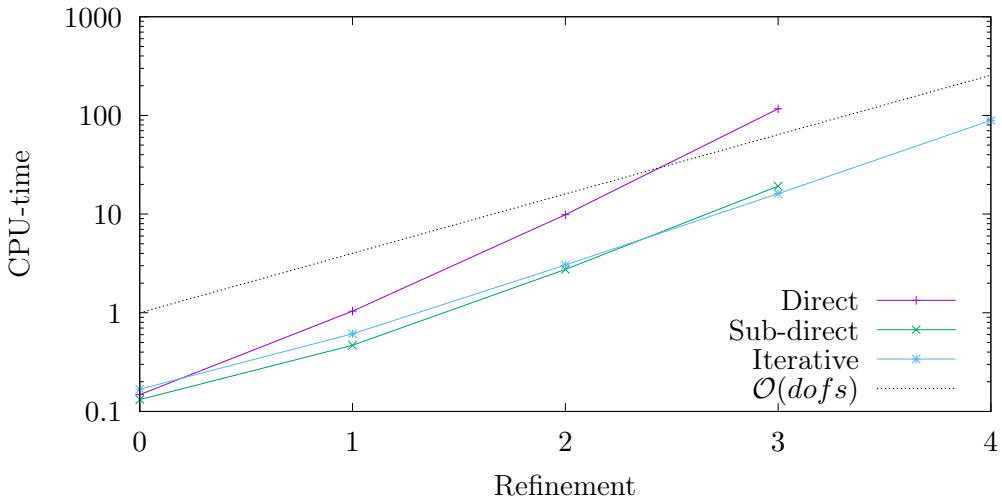



Figure 8.1: Comparison of the computational time for various solution strategies. "Direct" refers to the use of a direct solver for the whole matrix, "sub-direct" denotes the solution of the sub-problems with direct methods, whereas "iterative" solves all these systems iteratively.

## 8.2 Future Work

There are several aspects that can be improved. The most important one is to incorporate more efficient iterative solvers for the fluid and solid sub-problems, that work robustly with respect to different material parameters, $h$ and $\Delta t$. Several ideas are presented in [19, 26].Furthermore, one could adapt the preconditioner for different mesh-motion techniques like a biharmonic version. This should be straight-forward, but one will most likely have to replace the AMG mesh-solver by a different method.

Currently, only global refinements are done in our application. It would be interesting to see the effect of local refinements on the preconditioner.

Another interesting point would be to have an adaptive control of the convergence criteria for the nested iterative sub-solvers, that attempts to minimize the computational time. During testing, it was always some kind of guessing game to find a balance between speed and accuracy.

A very important issue is the parallelization on distributed memory machines. The limiting factor here is probably the scalability of the matrix-matrix multiplication required for the Schur-complements. One may overcome this by using local approximations of the Schur-complement or different solution techniques.

# Bibliography


[1] Robert A. Adams. *Sobolev spaces.* Pure and applied mathematics ; 65. New York, NY [u.a.]: Acad. Press, 1975.

[2] Stuart S. Antman. *Nonlinear problems of elasticity.* Applied mathematical sciences ; 107. New York, NY [u.a.]: Springer, 1995.

[3] Wolfgang Bangerth and Oliver Kayser-Herold. "Data Structures and Requirements for hp Finite Element Software". In: *ACM Trans. Math. Softw.* 36.1 (2009), pp. 4/1–4/31.

[4] W. Bangerth et al. "The `deal.II` Library, Version 8.4". In: *Journal of Numerical Mathematics* 24 (2016).

[5] D. Braess and R. Sarazin. "An Efficient Smoother for the Stokes Problem". In: *Appl. Numer. Math.* 23.1 (1997-02), pp. 3–19.

[6] Hans-Joachim Bungartz, Miriam Mehl, and Michael Schäfer. *Fluid Structure Interaction II: Modelling, Simulation, Optimization.* Vol. 73. Springer Science & Business Media, 2010.

[7] P. Causin, J.-F. Gerbeau, and F. Nobile. "Added-mass effect in the design of partitioned algorithms for fluid-structure problems". In: *Comput. Methods Appl. Mech. Engrg.* 194 (2005), pp. 4506–4527.

[8] Timothy A. Davis. "Algorithm 832: UMFPACK V4.3—an Unsymmetric-pattern Multifrontal Method". In: *ACM Trans. Math. Softw.* 30.2 (2004-06), pp. 196–199.

[9] J. Donea, S. Giuliani, and J.P. Halleux. "An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions". In: *Comput. Methods Appl. Mech. Engrg.* 33 (1982), pp. 689–723.

[10] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices.* Clarendon press Oxford, 1986.

[11] Miguel Ángel Fernández and Marwan Moubachir. "A Newton method using exact jacobians for solving fluid–structure coupling". In: *Computers & Structures* 83.2–3 (2005). Advances in Analysis of Fluid Structure InteractionAdvances in Analysis of Fluid Structure Interaction, pp. 127–142.

[12] M.W. Gee et al. *ML 5.0 Smoothed Aggregation User's Guide*. Tech. rep. SAND2006-2649. Sandia National Laboratories, 2006.

[13] Alan George and Joseph Liu. *Computer Solution of Sparse Linear Systems*. 1994.

[14] Matthias Heil. "An efficient solver for the fully coupled solution of large-displacement fluid-structure interaction problems". In: *Computer Methods in Applied Mechanics and Engineering* 193.1?2 (2004), pp. 1–23.

[15] Michael Heroux et al. *An Overview of Trilinos*. Tech. rep. SAND2003-2927. Sandia National Laboratories, 2003.

[16] Gerhard A. Holzapfel. *Nonlinear solid mechanics; a continuum approach for engineering*. Reprint. Chichester [u.a.]: Wiley, 2010.

[17] Stefan Kindermann. "Lecture Notes: An introduction to mathematical methods for continuum-mechanics". 2014.

[18] U. Langer and H. Yang. "Numerical Simulation of Fluid-Structure Interaction Problems with Hyperelastic Models: A Monolithic Approach". In: *Mathematics and Computers in Simulation* (2016). accepted for publication.

[19] U. Langer and H. Yang. "Numerical Simulation of Fluid-Structure Interaction Problems with Hyperelastic Models I: A Partitioned Approach". In: *Journal of Computational and Applied Mathematics* 276 (2015), pp. 47–61.

[20] U. Langer and H. Yang. "Robust and efficient monolithic fluid-structure-interaction solvers". In: *International Journal for Numerical Methods in Engineering* 108.4 (2016), pp. 303–325.

[21] J. M. Ortega. "The Newton-Kantorovich Theorem". In: *The American Mathematical Monthly* 75.6 (1968), pp. 658–660.

[22] Thomas Richter. "A monolithic geometric multigrid solver for fluid-structure interactions in ALE formulation". In: *International Journal for Numerical Methods in Engineering* 104.5 (2015). nme.4943, pp. 372–390.

[23] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[24] K. Stüben. *Algebraic Multigrid (AMG): An Introduction with Applications*. GMD-Report. GMD-Forschungszentrum Informationstechnik, 1999.

[25] Roger Temam. *Navier-Stokes equations; theory and numerical analysis*. Reprint. with corr. AMS Chelsea publishing. Providence, RI: American Math. Soc., 2001.

[26] Stefan Turek. "Efficient solvers for incompressible flow problems: An algorithmic approach". In: *Lecture Notes in Computational Science and Engineering*. 1999.

[27] Stefan Turek and Jaroslav Hron. "Proposal for Numerical Benchmarking of Fluid-Structure Interaction between an Elastic Object and Laminar Incompressible Flow". In: *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*. Ed. by Hans-Joachim Bungartz and Michael Schäfer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 371–385.

[28] Hongwu Wang and Ted Belytschko. "Fluid–structure interaction by the discontinuous-Galerkin method for large deformations". In: *International Journal for Numerical Methods in Engineering* 77.1 (2009), pp. 30–49.

[29] Jörg Weickert and Jorg Weickert. *Navier-Stokes equations as a differential-algebraic system*. 1996.

[30] Thomas Wick. "Fluid-structure interactions using different mesh motion techniques". In: *Computers & Structures* 89.13–14 (2011), pp. 1456–1467.

[31] Thomas Wick. "Lecture Notes: Modeling, Discretization, Optimization, and Simulation of Fluid-Structure Interaction". 2015-09-29.

[32] Thomas Wick. "Solving Monolithic Fluid-Structure Interaction Problems in Arbitrary Lagrangian Eulerian Coordinates with the deal.II Library". In: *Archive of Numerical Software* 1.1 (2013), pp. 1–19.

[33] W.L. Wood. *Practical Time-stepping Schemes*. Clarendon Press, 1990.

[34] Walter Zulehner. "A Class of Smoothers for Saddle Point Problems". In: *Computing* 65.3 (2000), pp. 227–246.

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, November 2016

_____

Daniel Jodlbauer