Computational Science and Engineering

Technische Universität München

Master's Thesis

# Aeroelastic simulation of slender wings for electric aircraft: a partitioned approach with DUNE and preCICE


| | |
|---|---|
| Author: | Max Firmbach |
| Examiner: | Prof. Dr.-Ing. Rainer Callies |


| | |
|---|---|
| Submission Date: | December 22nd, 2020 |

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

December 22nd, 2020 Max Firmbach

# Abstract

Distributed electric propulsion is a promising new technology for small sized aircraft. It enables new possibilities regarding the aerodynamic design, by using large slender wings. Those are prone to oscillations induced by fluid-structure interactions and thus need to be analyzed in detail. This thesis introduces a partitioned coupling approach based on the C++ libraries DUNE and preCICE to be able to simulate the occurring aeroelastic effects. In a first step, the equations of linear elastodynamics are solved based on finite elements and validated with a static and dynamic simulation scenario. Afterwards, the partitioned coupling approach is presented by introducing an adapter that connects the DUNE framework to the coupling interface provided by preCICE. The multi-physics setup and the respective adapter implementation is validated with three popular fluid-structure interaction benchmark cases.

# Contents

# 1. Introduction

Propulsion systems of modern airplanes have not changed significantly in the last 30 years. The necessary thrust to fly is mainly generated by big Turboprop- and Turbofan-engines placed under the wing. The turbines suck in air, compress it for an internal combustion process and exhaust the accelerated gases. This propulsion approach adds unnecessary weight to the aircraft and generates more noise as well as emissions. An interesting topic of research is the investigation of alternative propulsion systems especially for small size aircraft. One approach in this area is the distributed propulsion. Several small electric fans are mounted over the span of the wing. This offers new possibilities regarding airplane control and flight dynamics with thrust vectoring. In addition, the aerodynamic behavior of the wing can be changed resulting in a different geometry. Large slender wings with a short chord length similar to those of sail planes offer the necessary wingspan to mount the electric fans. Due to the high aspect ratio, the wings are prone to vibrations induced by the airflow. These aeroelastic effects play a crucial role during the design and simulation process of such wings for this propulsion concept.

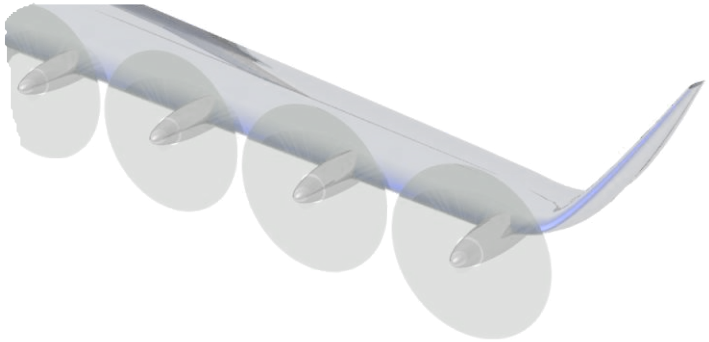


Figure 1.1.: Section of a slender wing with a distributed propulsion system (from [DLR, 2019])

Traditional, commercial programs often use a monolithic approach to calculate the underlying physical problem. The governing equations are implemented inside one framework, which solves the overall problem at once. This procedure has several drawbacks

regarding code reusability and flexibility, as the numerical solver is written for only one specific type of problem. In contrast, a partitioned approach couples independent, single physics codes and combines them to a multiphysics setup. This enables the usage of already existing programs and an efficient integration of new solvers.

This thesis presents a partitioned coupling approach for general fluid-structure interaction scenarios, which also cover aeroelastic simulation. The structural solver will be build up from scratch by using the distributed numerics environment DUNE to implement the linear equations of elastodynamics. For the fluid-structure interaction simulation the open source fluid solver OpenFOAM is coupled to the structural part by using the preCICE coupling library.

## 1.1. Review of aeroelastic modelling approaches

Fluid-structure interaction scenarios play a crucial role in multi physics simulation. A more specific scenario deals with the movement of slender, flexible structures inside a flow, called aeroelasticity. A prominent application case are wings deflecting under the load generated by a flow. In certain situations the structure starts oscillating being critial for material fatigue. The recent development drives to longer and thinner blades that make aeroelastic simulation crucial in the development process. The aeroelastic modelling of wings is nowadays mainly driven by the wind turbine development. An overview of modelling and simulation approaches are given by [Huang, 2011] and [Wang et al., 2016]. Some of the presented methods can be adapted and thus also be used for aeronautical applications.

### Aerodynamic model

The BEM (blade element momentum) model is widely used as aerodynamic participant and is implemented in several partitioned simulation programs for wind turbine analysis including AeroDyn/FAST (refere to [Moriarty and Hansen, 2005]) and HAWC2 (see [Heinz et al., 2016]). An approach more fitted to airplane wings is to neglect the forces resulting from the rotation leading to a simplified blade element model (similar to the strip theory commonly used for wing simulation). In [Manwell et al., 2009] it is stated that the blade element method is based on the local computation of aerodynamic forces on two dimensional airfoil geometries. The overall wing or blade is divided into several small elements. Each element consists of a specific two dimensional airfoil shape and a local flow condition, which sum up to the overall force acting on the structure. The aerodynamic forces are therefore calculated based on the lift and drag characteristics of the airfoil shape per element and a local velocity resulting in the lift force:

$$F_L = c_L \frac{\rho}{2} u_{rel}^2 S \tag{1.1}$$

and drag force:

$$F_D = c_D \frac{\rho}{2} u_{rel}^2 S \tag{1.2}$$

with density $\rho$, the relative velocity to the wing direction $u_{rel}$ and the reference area $S$. The forces can afterwards be transformed into a vertical and a horizontal direction component. To account for turbulence, tip loss, dynamic inflow and stall phenomena, additional empirical correction equations have to be solved. The introduced model is only a coarse approximation of the real aerodynamic system and is highly dependent on the specific airfoil parameters and empirical data stored for the calculation. This includes the lift and drag coefficients for different flow scenarios. If no reliable data is provided, the method lacks in accuracy. The simple computational approach however results in a high computational efficiency that makes the method attractive for real time simulations.

A more general approach is to resolve the exact flow around an object resulting in a CFD (computational fluid dynamics) model. The mathematical foundation for this approach are the formulas resulting from momentum, continuity and energy conservation. The fluid flows considered in this thesis are assumed to be incompressible, due to a Mach number of $Ma < 0.3$. This assumption yields a simplified version of the nonlinear hyperbolic partial differental equation system called the incompressible Navier-Stokes equations with constant density:

$$\nabla \cdot u = 0, \tag{1.3}$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\frac{1}{\rho}\nabla p + \nu \Delta u, \tag{1.4}$$

with velocity $u$, pressure $p$, density $\rho$ and kinematic viscosity $\nu$. The fluid domain is usually discretized with finite volumes in space. This results in a mesh consisting of cells, for which pressure and velocity values can be calculated. Depending on the mesh refinement the flow around objects can be resolved with high accuracy. For most applications this approach already needs significantly more computational power even for small models resulting in long solving times. With parallelization approaches this problem can be bypassed to some degree.

### Structural model

The structural domain in dedicated aeroelastic simulations is often approximated by a one dimensional approach (see [Huang, 2011] and [Wang et al., 2016]). The wing cross section properties are calculated based on special routines incorporated into the structural model. The three most relevant one dimensional discretization methods are the modal approach, multibody dynamics and finite elements.

In the modal analysis scheme, the deflection can be obtained by linear combination of the mode shapes of the structure. The scheme is based on a transformation of the physical

coordinates into the modal space. This procedure reduces the degrees of freedom of the system and decouples the equations of motion, resulting in an efficient computation. A popular approach to obtain an approximation of the different modal shapes is the usage of a three dimensional finite element preconditioner. For complex structures the calculation of the deflection might be efficient, but the initial approximation of the mode shapes can be costly and time consuming. Therefore, only the first four modes are commonly taken into consideration. Due to the linear combination approach, non-linearities like large deformations can't be modelled accurately.

In the multibody dynamics approach the structure is split up into flexible and rigid parts, which are connected by force elements (springs, dampers). The movement of the system is often described by Lagrangian's equations of motion. The accuracy of the model can be increased by adding more elements and thus the degrees of freedom, resulting in a computationally more intensive calculation.

The one dimensional FE (finite element) method splits up the structure in several nodes, which are connected by beam elements. Depending on the formulation of the elements (based on the theory of Euler-Bernoulli's or Timoshenko's description for beams) an accurate deformation can be calculated. The scheme can also be adapted for large deformations.

A more conventional approach is similar to the computational fluid dynamics procedure by discretizing the fundamental continuum mechanical equations resulting in a more flexible scheme. In this case the equations of linear elastodynamics can be discretized by two and three dimensional finite elements, resulting in a computational structure mechanics scheme.

## 1.2. Simulation approach and thesis overview

The aeroelastic modelling approaches shown in section 1.1 are mostly rough approximations of the real fluid flow and the structural deformations. In this thesis a partitioned approach is presented using the coupling library preCICE. This offers a flexible interface for the coupling of different solver types. A minimal invasive approach makes the modification of already existing solvers for a coupling straightforward and thus encourages to reuse code. The structural participant will be modeled as linear elastic material with two and three dimensional finite elements. The corresponding solver is implemented with the numerics-library DUNE. On the fluid side, a computational fluid dynamics approach is chosen, which is realized with OpenFOAM. The main focus of the thesis lies on the implementation in DUNE and the coupling approach with preCICE.

### Structure of the thesis

In chapter 2 the governing equations of the linear elastodynamic theory are discussed. In a next step, the reformulation into a variational problem and the discretization with finite elements is shown. Additionally, the necessary numerical schemes to solve such systems are presented. Chapter 3 focuses on the implementation of the presented methods inside the DUNE framework. The code structure and the simulation procedure of the structural solver is discussed. Afterwards the code is validated with a static beam bending scenario and the vibration of a cantilever plate in chapter 4. The second part of this thesis deals with the main ideas and the background theory of partitioned simulation approaches, which are given in chapter 5 presenting data mapping, coupling and time interpolation schemes. In chapter 6 the preCICE coupling library is introduced. The respective implementation into the DUNE environment is shown, resulting in the dune-precice module. The modification of a dummy solver for a partitioned coupling is presented. The coupling adapter implementation in combination with the structural solver are validated with different fluid-structure interaction scenarios in chapter 7, consisting of a flexible flap inside a channel, a lid-driven cavity with flexible bottom and the fluid-structure interaction three benchmark. Chapter 8 gives a summary of the thesis and an outlook of future topics concerning partitioned simulations related to aeroelastic modelling and the presented DUNE modules.

# 2. Background on solving linear elastodynamics

The fundamental equations describing linear elastodynamic phenomena are based on continuum mechanics. The material is assumed to be continuous ignoring the atoms it is actually made of. The mechanical model that will be discussed is assumed to be valid for small deformations using the Cauchy stress tensor $\sigma$ and the linearized strain tensor $\epsilon$. The material law describing the relation between stresses and strains is also assumed to be linear. Thus, calculations with nonlinear material behaviour or large deformations can't be accurately modeled.

In this chapter, the equations of linear elastodynamics are discussed in section 2.1. Afterwards, the formulation of the weak form and the discretization of the equations with finite elements is shown in section 2.2. Explicit timestepping methods and mass lumping approaches are presented in section 2.3. In a last step, a short introduction into direct-time integration schemes and implicit methods is given in section 2.4.

## 2.1. Equations in linear solid mechanics

The fundamental description of the movement of a body is given by a partial differential equation. It is based on an equivalence of forces given by Newton's second law. The equation of motion can thus be written as:

$$\rho \frac{\partial^2 u}{\partial t^2} + \nabla \cdot \sigma(u) = f \tag{2.1}$$

where $\rho$ describes the density, the second derivative of the displacement is resulting in the acceleration, body forces $f$ like the gravitational force are given on the right side and the Cauchy stress tensor $\sigma$. The relation between stress and strain is generally given by a so called material law. In case of the linear elasticity assumption the material can be modeled according to Hooke's law:

$$\sigma(u) = C : \epsilon(u). \tag{2.2}$$

The material stiffness tensor $C$ is based on measurable, physical parameters consisting of the Young's modulus $E$ and Poisson ratio $\nu$. The strain tensor can be obtained by a

linearization of the Green-Lagrange strain resulting in the strain-displacement relation:

$$\epsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^T) \tag{2.3}$$

with the displacement $u$ and the infinitesimal strain $\epsilon$. With the three given equations, the formulation of the overall elastodynamic problem thus yields:

$$\begin{cases} \rho\frac{\partial^2 u}{\partial t^2} + \nabla \cdot \sigma(u) = f & \text{(Newton's second law)} \\ \sigma(u) = C : \epsilon(u) & \text{(Hooke's law)} \\ \epsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^T) & \text{(Strain-displacement relation)} \end{cases} \tag{2.4}$$

describing the deformation of an object over time.

## 2.2. Finite elements in linear elastodynamics

Newton's second law given in equation 2.1, describes the motion of a continuum in every single point. To calculate an approximation of this classical solution, the problem is transformed into it's weak form obtaining a variational problem. Afterwards the equation is discretized with finite elements to obtain a system of ordinary differential equations.

### Weak formulation

To obtain the weak formulation each term of the equation of motion is multiplied with a test function $v \in V$ and integrated over the bounded domain $\Omega$ with a continuous boundary $\partial\Omega$:

$$\int_\Omega \rho\frac{\partial^2 u}{\partial t^2} \cdot v \, dV + \int_\Omega \sigma(u) : \nabla v \, dV = \int_\Omega f \cdot v \, dV + \int_{\partial\Omega} \sigma(u)n \cdot v \, dS. \tag{2.5}$$

The stress tensor $\sigma(u)$ can be rewritten in terms of the material stiffness $C$ and strain tensor $\epsilon(u)$ by using the relation of equation 2.2. With the strain-displacement relation from 2.3 the expression $\epsilon(u)$ can be replaced due to the symmetry of $\sigma(u)$:

$$\int_\Omega \sigma(u) : \nabla v \, dV = \int_\Omega C : \epsilon(u) : \nabla v \, dV = \int_\Omega \nabla u : C : \nabla v \, dV. \tag{2.6}$$

Inserting the result into equation 2.5 yields:

$$\int_\Omega \rho\frac{\partial^2 u}{\partial t^2} \cdot v \, dV + \int_\Omega \nabla u : C : \nabla v \, dV = \int_\Omega f \cdot v \, dV + \int_{\partial\Omega} \sigma(u)n \cdot v \, dS. \tag{2.7}$$

The weak formulation written in equation 2.7 can be reformulated in terms of the stiffness bilinear form $K(u, v)$, the mass bilinear form $M(\ddot{u}, v)$ and the force linear form $f(v)$ containing body and surface forces. Therefore, the variational problem of form:

$$M(\ddot{u}, v) + K(u, v) = f(v) \tag{2.8}$$

needs to be solved for $u \in V$.

### Discretization in space

By choosing appropriate global basis functions $\varphi$ the solution $u$ of the variational problem is approximated by $u_h$ and can be written as:

$$u_h = \sum_j u_j \varphi_j. \tag{2.9}$$

Choosing identical spaces for basis and test functions according to Ritz-Galerkin yields a system of ordinary differential equations, written in matrix form:

$$M \frac{\partial^2 u_h}{\partial t^2} + K u_h = f, \tag{2.10}$$

with the mass matrix $M$, stiffness matrix $K$ and load vector $f$. Setting test function $v$ to $\varphi$ and inserting equation 2.9 gives the following representations of mass and stiffness matrix:

$$M = \int_\Omega \varphi^T \rho \varphi \, dV, \tag{2.11}$$

$$K = \int_\Omega \nabla \varphi^T C \nabla \varphi \, dV. \tag{2.12}$$

The calculation of the system matrices can either be done with a global or local approach. Here, only the local approach is considered. The continuous domain is split up in discrete parts called elements. Each of these elements contributes to the global stiffness and mass matrix. This results in a local computation for the corresponding element matrices and a global assembly of the system matrices:

$$M_e = \int_{\Omega_e} \varphi_e^T \rho_e \varphi_e \, dV \quad , \quad M = \sum_{e=1}^{N} T_e^T M_e T_e, \tag{2.13}$$

$$K_e = \int_{\Omega_e} \nabla \varphi_e^T C_e \nabla \varphi_e \, dV \quad , \quad K = \sum_{e=1}^{N} T_e^T K_e T_e, \tag{2.14}$$

with transformation matrices $T$ to place the entries into the global matrix.

## 2.3. Explicit timestepping methods and mass lumping

The discretization of the equation of motion with finite elements yields a second order ordinary differential equation of the form:

$$M\frac{\partial^2 u_h}{\partial t^2} + Ku_h = f. \tag{2.15}$$

The solution to this initial value problem can be approximated by discretizing the time. This is archived with time stepping methods, which can either be implicit or explicit. In a first step, the standard explicit methods for ordinary differential equations of first order are discussed. Therefore, equation 2.15 is transformed into a system of respective order. Following [Dinkler, 2017], $u$ and $\frac{\partial u}{\partial t}$ are chosen as independent variables and the system can thus be written as:

$$\begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix} \dot{y} + \begin{bmatrix} 0 & K \\ K & 0 \end{bmatrix} y = \begin{bmatrix} 0 \\ f \end{bmatrix} \tag{2.16}$$

with

$$y = \begin{bmatrix} u \\ \frac{\partial u}{\partial t} \end{bmatrix}. \tag{2.17}$$

In general an ordinary differential equation can be formulated in explicit form:

$$\dot{y} = f(t, y). \tag{2.18}$$

Considering the transformed system from equation 2.16, the explicit form can be written as:

$$\dot{y} = \begin{bmatrix} 0 & I \\ -M^{-1}K & 0 \end{bmatrix} y + \begin{bmatrix} 0 \\ M^{-1}p \end{bmatrix}. \tag{2.19}$$

### Explicit methods

Applying explicit time stepping schemes to types of problems written in the form of equation 2.18 have the advantage of only using information that has already been calculated. According to [Hairer et al., 2008], the methods can be divided in one-step and multi-step methods. While the latter ones use several values, the one-step methods only use information from the last time step.

A family of explicit one-step methods are the Runge-Kutta schemes. An s-stage explicit Runge-Kutta method can be written as:

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_j k_j, \tag{2.20}$$

with the time step size $h$. The approximation $y_{n+1}$ at $t_{n+1}$ is calculated only using information from the last time step, thus is based on the result $y_n$ at $t_n$. The sum consists of weights $b$ and intermediate function evaluations $k$ at different times inside the time interval $t_n$ and $t_{n+1}$:

$$k_j = f(t_n + hc_j, y_n + h\sum_{l=1}^{s} a_{jl}k_l),\ j = 1...s. \tag{2.21}$$

An expansion to the method yields an approximation $y_{n+1}$ and additionally calculates a second approximation $\hat{y}_{n+1}$. The embedded Runge-Kutta methods use two schemes of different order $p$ and $\hat{p}$ (with $\hat{p} = p \mp 1$). Both algorithms use the same function evaluations, but different weighting values $b$ and $\hat{b}$. The different coefficients $a$, $b$, $\hat{b}$ and $c$ can be visualized using the butcher-tableau of form:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b \\ \hline & \hat{b}_1 & \cdots & \hat{b}_s \end{array}$$

The difference of both approximations of the solution can be used as local error estimator for an adaptive time stepping. Explicit time stepping methods are only conditionally stable, restricting the timestep size. In the scope of this thesis, an Runge-Kutta scheme with adaptive time stepping is implemented (introduced by [Dormand and Prince, 1980]). Six function evaluations are used to calculate solutions of fourth and fifth order accuracy.

### Mass lumping

Considering equation 2.19, the explicit form can only be written by inverting the mass matrix $M$. A direct inversion is often only efficiently realizable if the matrix is diagonal. Otherwise a linear system has to be solved with the mass matrix on the left side, neglecting the advantages of the explicit form. The mass matrix obtained by equation 2.11 is called consistent as the mass is distributed uniformly over the elements. A concentration of the masses on the element nodes leads to a diagonalization of the matrix, called mass lumping. The lumping procedure is in general not unique, thus several methods can be used that need to fulfill certain conditions. The mass of every element needs to be conserved, thus the overall mass should not differ from the consistent approach:

$$\sum_i M_{ii}^{(lumped)} = \int_\Omega \rho \, d\Omega. \tag{2.22}$$

Additionally, positivity of the mass matrix entries needs to be guaranteed. The most simple scheme is to sum up each row of the consistent mass matrix and place the entry on the diagonal:

$$M_{ii}^{(lumped)} = \sum_j M_{ij}. \tag{2.23}$$

This approach only fulfills the positivity condition for elements with linear shape functions and is thus not usable for higher order finite elements. Another method was introduced by [Fried and Malkus, 1975] using a so called Lobatto lumping. A special quadrature rule is constructed for the approximation of equation 2.11, where the quadrature and nodal points of the element coincide. The basis functions are thus evaluated at the nodal points of each element with a special choice of the weights, the resulting mass matrix is diagonal. A simpler approach that is also valid for higher order elements is the HRZ lumping (see [Hinton et al., 1976]), where the diagonal entries are scaled. For each element, only the diagonal entries of the mass matrix are calculated. Afterwards the entries are scaled:

$$M_{e,ii}^{(lumped)} = \frac{m_e}{tr(M_e)} M_{e,ii}, \tag{2.24}$$

with $m_e$ being the element mass. In comparison to the row summation approach, the consistent mass matrix does not need to be calculated resulting in a more efficient scheme based on the element level. An overview of customized mass lumping for specific problems is given by [Felippa et al., 2015].

## 2.4. Direct-time integration and implicit methods

For certain problems the time step size of explicit methods is very small in order to fulfill the stability condition. This results in long computation times, despite the low cost per step. In this case, implicit methods can be preferable, due to stiff parts in the equation.

Additionally, direct-time integration methods have been proposed for second order equations, which makes the transformation into a system of first order unnecessary. These schemes are thus more storage efficient as no additional variables need to be introduced as given in [Hairer et al., 2008]. In structural dynamics, such schemes are typically constructed for certain problem statements and often lack features of advanced Runge-Kutta methods.

### Direct-time integration

The direct-time integration is an alternative to the standard approach given in section 2.3, solving the second order equation directly. In [Hairer et al., 2008], such schemes are presented as Runge-Kutta Nyström methods, which can be quite efficient for problems of the form $\ddot{y} = f(t, y)$ (see [Bettis, 1973]). Considering structural dynamics, often special

schemes are constructed including the camping matrix $C$ (example given in [Lunk and Simeon, 2005]). Those directly solve the second order ordinary differential equation based on a step-by-step approach. The unknowns consisting of displacement, velocity and acceleration are updated, such that the equilibrium equation is fulfilled in each time step. Depending on how the variables vary within a time interval, different schemes can be constructed. Several methods have been proposed for the solution of the linear equation of structural dynamics (see equation 2.15). An overview of those methods is given in [Subbaraj and Dokainish, 1989] including the well known Wilson-$\theta$, Houbolt and Newmark methods. The latter one is a popular family of direct-time integration schemes for structural dynamics, which were introduced by [Newmark, 1959]. The scheme can be written in predictor/corrector form , with the predictor steps given as:

$$y^*_{n+1} = y_n + h\dot{y}_n + (0.5 - \beta)h^2\ddot{y}_n, \tag{2.25}$$

$$\dot{y}^*_{n+1} = \dot{y}_n + (1 - \gamma)h\ddot{y}_n, \tag{2.26}$$

where intermediate values for the displacement and velocity are calculated. Afterwards a linear system is solved to obtain the unknown acceleration for the next time step:

$$M_{eff}\,\ddot{y}_{n+1} = f_{n+1} - C\dot{y}^*_{n+1} - Ky^*_{n+1}, \tag{2.27}$$

with the effective mass matrix:

$$M_{eff} = M + \gamma hC + \beta h^2 K. \tag{2.28}$$

In a last step, the predictor values are corrected based on the calculated acceleration:

$$\dot{y}_{n+1} = \dot{y}^*_{n+1} + \gamma h\ddot{y}_{n+1}, \tag{2.29}$$

$$y_{n+1} = \dot{y}^*_{n+1} + \beta h^2\ddot{y}_{n+1}. \tag{2.30}$$

With the free variables $\beta$ and $\gamma$ the methods accuracy and stability can be altered. The parameters also determine the schemes type as shown in table 2.1. All methods of the Newmark family are one step methods and can archive second order accuracy for a special choice of the parameters. In addition, the Fox-Goodwin method can achieve an order four accuracy, if no damping is included. The central difference method is only fully explicit for diagonal matrices $M$ and $C$, which can be done through lumping. Considering stability, most of the methods are only conditionally stable and thus yield no major advantage over the explicit methods presented in section 2.3.

| Method | Type | $\beta$ | $\gamma$ | Stability |
|---|---|---|---|---|
| Average acceleration (midpoint rule) | Implicit | 1/4 | 1/2 | unconditional |
| Linear acceleration | Implicit | 1/6 | 1/2 | conditional |
| Fox-Goodwin | Implicit | 1/12 | 1/2 | conditional |
| Central difference | Explicit | 0 | 1/2 | conditional |

Table 2.1.: Parameters determining the type of the Newmark method considering different values for $\beta$ and $\gamma$

### Implicit methods

In some of the Newmark methods shown, results from the time $t_{n+1}$ are used for the calculation of the velocity and displacement in addition to the information at $t_n$. Such schemes are called implicit. In each timestep a linear system has to be solved to obtain the information at $t_{n+1}$, increasing the computational intensity. These schemes can however be unconditional stable independent of the size of the timestep. In regards of structural dynamics the Newmark-beta method is such a scheme, with the parameters set to $\beta = 1/4$ and $\gamma = 1/2$.

An addition to the linear acceleration given in table 2.1 was presented by [Wilson et al., 1973]. By setting $\beta = 1/6$ and $\gamma = 1/2$ as fixed values and introducing a new parameter $\theta$ the stability and numerical dissipation of the method can be changed. The basic equations of the Wilson-$\theta$ method are given by:

$$\dot{y}_{t+\theta\Delta t} = \dot{y}_t + \frac{\theta\Delta t}{2}(\ddot{y}_{t+\theta\Delta t} + \ddot{y}_t), \tag{2.31}$$

$$y_{t+\theta\Delta t} = y_t + \theta\Delta t\dot{y}_t + \frac{1}{6}(\theta\Delta t)^2(\ddot{y}_{t+\theta\Delta t} + 2\ddot{y}_t). \tag{2.32}$$

Similar to the Newmark methods, equations 2.31 and 2.32 can be split up into a corrector/ predictor scheme. The method is constructed to be unconditional stable for values of $\theta \geq 1.37$. Additionally to the stability, the numerical dissipation can be controlled with $\theta$.

# 3. Implementation: Components of the solver

This chapter gives an overview of the code structure and the most important features regarding the implementation. The single components of the program are grouped into different layers shown in figure 3.1, similar to the code presented by [Butler et al., 2020]. The whole program is based on the C++ library DUNE (see [Blatt et al., 2016]) and it's specific modules, representing the base layer. Additionally, some code examples presented by [Sander, 2020] were used as starting point of the implementation. The different components provide specific functionality to build numerical solvers, in this case for solving linear elastodynamical problems:

- `dune-istl`: The module is used for the setup of data structures like nested sparse matrices and block vectors. Additionally, methods for solving linear systems of equations are implemented. A more detailed description of the module and it's features is given in [Blatt and Bastian, 2007].

- `dune-uggrid`: Provides the implementation of unstructured grids.

- `dune-grid`: The generic grid interface (refere to [Bastian et al., 2008]) is given in this module and the connection to the unstructured grid implementation, as well as the possibility of loading gmsh files into the DUNE environment.

- `dune-functions`: The module provides different function spaces for finite elements.

- `dune-geometry`: Implements a variety of quadrature rules for the evaluation of integral equations on different reference elements.

The top layer is the so called user layer. The only way to interact with the solver is via this interface. It consists of a configuration file, where all necessary simulation files and parameters are set.

The actual solver with it's main components represents the middle layer. The code structure of the implemented solver can be split into three main parts consisting of how to use and extend the code, the assembly and last, the solving of the equations. The first section given in 3.1 focuses on the usage of the solver and the simulation setup. In a next step the different assembler implementations for global and local scopes are discussed in section 3.2. Finally, a brief description of the actual solution process of the underlying equations is given in section 3.3.

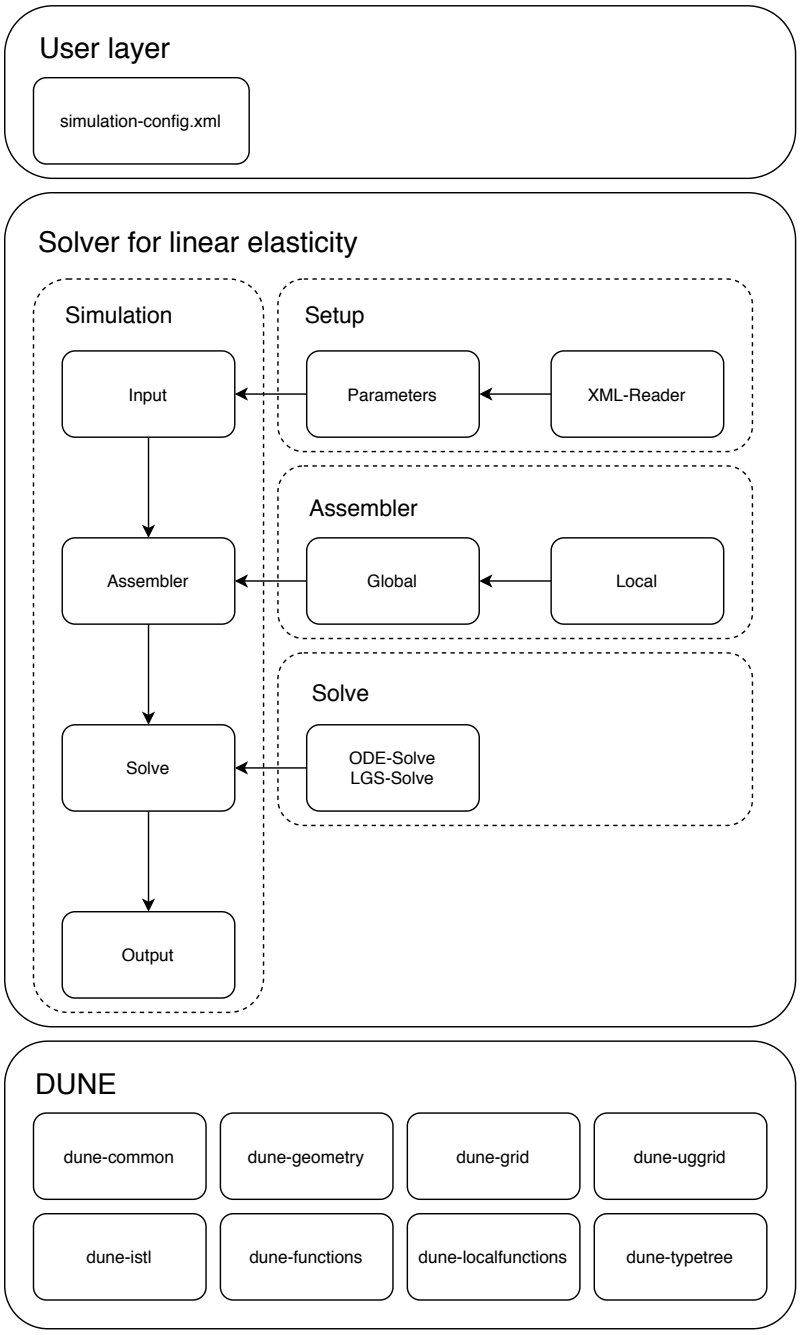



Figure 3.1.: An overview of the code structure implementing the solver for the linear elastodynamics equations

## 3.1. Using and extending the solver

Before the start of a simulation, settings regarding the solver need to be specified inside a configuration file. During runtime the parameters set in the file are parsed and stored inside a parameter class consisting of five parts:

- Solver interface: Inside the solver interface block, the dimension of the problem and the order of the basis functions is set. These two properties will be used to set up the necessary data structures and the basis for the finite element calculation. Additionally, the solver type is chosen either for a static or dynamic setup, the latter needs additional information about timestep size and simulation runtime.
- Mesh file: The meshing is based on the open source program gmsh and it's respective file format. This creates the opportunity to generate grids, which resemble complex geometries either with structured or unstructured meshes. Additionally, it is necessary to specify element and boundary tags inside the mesh file, which are unique integer numbers. Those will later be used for material and boundary assignment inside the assembler.
- Output generation: The results of the simulation are written in the VTK file format. This enables the visualization and post processing with ParaView. The ouput filename and writing interval for transient simulations can be modified.
- Coupling information: Considering multiphysics simulations, the solver is ready to be used with fluid structure interaction scenarios. The corresponding parameters are set in the coupling block. With the status setting, the interface can be turned on and off. If the parameter is set to true, the solver waits for a coupling partner to communicate with. Otherwise, conventional static and dynamic simulations are executed.
- Material properties: The material properties that need to be specified consist of the Young's modulus $E$, the Poisson ratio $\nu$, the material density $\rho$ and a name. In total three different materials can be set. They are assigned with different unique material parameter tags.

New assemblers on global and local scope can be added inside the respective files of the `/Assembler` folder. All functions are implemented as templates offering flexibility regarding the dimension of the problem and different orders of the Lagrangian basis used. The different time schemes are specified in the `/Integrator` directory and implemented as classes.

## 3.2. Assembler for global and local matrices

In this section, the implementation of the assembly of the system matrices and the computation of the element matrices is presented. First, the main aspects of the global assembler are shown and thus the procedure how to obtain the stiffness, consistent mass and lumped mass matrix. Additionally, the global incorporation procedure of the boundary conditions is shown. Afterwards, the local computation of the element contributions is discussed. The approach is based on a reference element calculation, which is mapped to the actual element geometry.

### Global Assembler

While the calculation of the element matrix is exclusively done on a single element, the respective matrices need to be placed into the global system matrix. The connection between local and global scope can be established with an index mapping. Each vertex of the finite element mesh is numbered in a global manner and additionally has a unique local index inside an element it belongs to. The boolean index mapping matrix $T$ resembles this property providing an mapping from local to global numbering by only having one entry per row that connects the scopes. The assembly of a system matrix based on the summation of the element matrices is given by:

$$A = \sum_e T_e^T A_e T_e, \tag{3.1}$$

for every element $e$ with the calculated element matrix $A_e$ and boolean element index mapping matrix $T_e$.

In the code implementation, the index set functionality of DUNE is used to provide the correct mapping. In finite element calculations the system matrices are sparse, thus only a limited number of entries of the matrix are accessed. The global assembly procedure is done with three loops. The outermost loop iterates over each element of the grid to provide the necessary information for the element matrix calculation and retrieve the results. Afterwards, the entries are added into the system matrix based on equation 3.1 by using the respective indices of the local element and map them to the corresponding global numbering. This procedure is done for row and column entries. A simple code example showing the most important features of the global assembler is given in figure 3.5.

While the main code structure shown in figure 3.5 is used for the assembly of the stiffness matrix, consistent and HRZ-lumped mass matrices, each of the global assemblers differ slightly in the initialization and setup. For the stiffness matrix, the Young's modulus and Poisson ratio of the current element is retrieved to calculate the respective material stiffness matrix. Considering the mass matrices, the density of the element is recovered. For the incorporation of the boundary conditions, an additional loop is inserted, which iterates over the element intersections to retrieve the boundary tag. Those are set during mesh generation and are based on unique integer values, which are imported while

loading the mesh. Each intersection also features an boolean value, which is true if the intersection is on the outer boundary of the mesh and is false otherwise. Based on the tag, the corresponding boundary condition is inserted into the system matrices and load vector (see figure 3.3).

```
for( const auto& element : elements(gridView)) {

  // calculate element matrix ...

  for( size_t i=0; i<elementMatrix.N(); i++) {
    auto row = localIndexSet.index(i);
    for( size_t j=0; j<elementMatrix.M(); j++) {
      auto col = localIndexSet.index(j);
      matrix[row][col] += elementMatrix[i][j];
    }
  }
}
```

Figure 3.2.: Example program showing the main features of a global assembler in DUNE

```
for( const auto& element : elements(gridView)) {
  for( const auto& isect : intersections(gridView, element)) {
    if(isect.boundary()) {
      switch(boundaryEntity[isect.boundarySegmentIndex()]) {
        case 0:
          // no boundary condition applied ...
          break;
        case 1:
          // set fixed wall boundary condition ...
          break;
        case 2:
          // set coupling boundary condition ...
          break;
      }
    }
  }
}
```

Figure 3.3.: Example program showing the main features of the global boundary condition incorporation

### Local Assembler

To obtain the element matrices, a local assembler is used. It only uses information of a single element at time and either calculates the contributions to the stiffness, consistent or HRZ-lumped mass matrix. The globally defined basis functions $\varphi$ introduced in section 2.2 are defined for every single vertex of the mesh. Additionally, the functions are chosen in a way to feature only local support, being nonzero on the respective element domain and zero otherwise. This property results in sparse system matrices in the assembly process. Concerning an appropriate basis, a continuous Lagrangian basis $S_p^0$ is chosen consisting of polynomials of order $p$ (here only first and second order basis functions are considered). Inside the code implementation, the syntax given in figure 3.4 is used to create the Lagrangian basis of order $p$.

```
const int basisOrder = // 1 or 2 ...
Functions::LagrangeBasis<GridView, basisOrder> basis(gridView);
```

Figure 3.4.: The DUNE call to create a Lagrangian basis on a given mesh of order $p$

The integral equations 2.13 and 2.14 discussed in section 2.2 need to be calculated for every element and later added into the global matrices. The integrals are solved with a Gauss-quadrature of respective order to archive an exact integration. In general the elements of a mesh each have different coordinates. Additionally, for unstructured grids the shape of each element differs. Therefore, the local quadrature is defined on a so called reference element and the evaluation of the basis is mapped to the actual element. The procedure can be done with the determinant of the Jacobian matrix of the mapping function $det(J)$. If the gradients of the basis functions have to be calculated, then the transposed inverse of the Jacobian $(J^T)^{-1}$ needs to be used in addition. Thus, the integral equations can be rewritten to:

$$M_e = \int_{\Omega_e} \varphi_e^T \rho_e \varphi_e \, dV \approx \sum_{i=1}^{n} \omega_i \hat{\varphi}_e^T(\xi_i) \rho_e \hat{\varphi}_e(\xi_i) |det(J)| \tag{3.2}$$

and

$$K_e = \int_{\Omega_e} \nabla\varphi_e^T C_e \nabla\varphi_e \, dV = \sum_{i=1}^{n} \omega_i (J^T)^{-1} \hat{\nabla} \hat{\varphi}_e^T(\xi_i) C_e (J^T)^{-1} \hat{\nabla} \hat{\varphi}_e(\xi_i) det(J)| \tag{3.3}$$

with quadrature weights $\omega$ and quadrature coordinates $\xi$.

Considering the code implementation, the structure is similar to the global assembler. Again, loops are used for the assembly process, but this time iterating over quadrature points and matrix entries. In a first step, a quadrature rule of respective order is chosen for the approximation of the integral formulation. Furthermore, the quadrature point postion,

the determinant of the Jacobi matrix as well as the invere transposed of the Jacobian are set. Afterwards, the contribution of each quadrature point is calculated and added to the element matrix. A small code example with the main features of the element assembler is given in figure 3.5.

```
auto& quadRule = QuadratureRules<double, dim>::rule(element.type(), order);

for(const auto& quadPoint : quadRule) {

    auto quadPos = quadPoint.position();
    auto determinantJacobian = geometry.integrationElement(quadPos);
    auto invTrpJacobian = geometry.jacobianInverseTransposed(quadPos);

    localFiniteElement.localBasis().evaluateFunction(quadPos, shapeFnc);
    localFiniteElement.localBasis().evaluateJacobian(quadPos, shapeFncGr);

    for( size_t i=0; i<elementmassMatrix.N(); i++) {
        for( size_t j=0; j<elementmassMatrix.M(); j++) {
            elementMatrix[i][j] += quadPoint.weight() * // ...
        }
    }
}
```

Figure 3.5.: Example program showing the main features of a element assembler in DUNE

The implementations for the stiffness, consistent mass and HRZ-lumped mass matrices use the same concept shown above, but differ heavily considering their implementation. While the calculation process of the consistent mass matrix evaluates all basis function entries, the HRZ-lumped one only uses the diagonal entries and needs an additional scaling according to equation 2.24. In contrast, the stiffness matrix validates the basis function gradients and thus needs a different calculation procedure.

## 3.3. Numerical schemes for solving the governing equations

This section gives a short overview of the implementation of numerical schemes regarding linear systems and ordinary differential equations. The first part focuses on the solution process of the linear systems of equations that need to be solved during the simulation. Afterwards the usage of the time stepping schemes presented in section 2.3 and 2.4 is presented.

### Solving linear systems

During the process of a static calculation and implicit time stepping, a linear system of equations of the form

$$Au_h = b \tag{3.4}$$

has to be solved, where $A$ is a stiffness or mass matrix, vector $b$ a load vector and $u_h$ the solution approximated by finite elements. The DUNE interface offers several numerical methods that can be applied to such problems, from which the conjugate gradient method is chosen. To speed up convergence, a precondtioner is used to reduce the condition number of matrix $A$. The implemented method uses an incomplete LU decomposition, thus the reformulated problem is given by:

$$(LU)^{-1}Au_h = (LU)^{-1}b, \tag{3.5}$$

where the inverse of $A$ is tried to be efficiently approximated. A short syntax example is given in figure 3.6, including the preconditioning, operator generation and solving steps.

```
MatrixAdapter<Matrix, Vector, Vector> operator(systemMatrix);
SeqILU0<Matrix, Vector, Vector> preconditioner(systemMatrix, 1.0);
CGSolver<Vector> cg(operator, preconditioner, tol, iter, 2);
InverseOperatorResult statistics;
cg.apply(x, RHS, statistics);
```

Figure 3.6.: Example program showing the main features of the setup of the preconditioned conjugate gradient method in DUNE

The iterative process of the method can be stopped by choosing an appropriate tolerance for the convergence criterion. Additionally, a limit for the number of iterations can be set.

### Time stepping methods

For the solution of the occurring system of ordinary differential equations, direct integration methods as well as a more traditional explicit Runge-Kutta method are chosen. For the explicit scheme, the C version of the RK5(4) method presented by [Hairer et al., 2008] is used as a simple function call. The integration of the code into the existing framework consisting of the definition of the function evaluation of the equations and providing appropriate output for visualization was only possible with modifications to the internal functions. The advantages of a pure explicit scheme are partly negated due to the usage of different data structures. The nested sparse matrices and block vectors provided by DUNE can't be used and thus the data has to be written into specific arrays provided by the code structure, which lowers performance significantly. A more convenient approach is to write the method in a syntax, which is compatible to the functionality provided by DUNE. The Newmark family is implemented as a derived class, which is inherited from a direct integrator class, such that new schemes can easily be added. The intergrator provides two main methods called `initialize()` and `step()`. The first function sets up all necessary data structures and pre-calculations, where as the step method executes one time step. A short example of the usage of the Newmark scheme is given in figure 3.7.

```
double beta = 0.25;
double gamma = 0.5;
Dune::Newmark<sparseMatrix, blockVector> integrator(mass, stiffness);
integrator.coefficients(beta, gamma);
integrator.stepsize(dt);
integrator.initialize(displacement, load);

while(t<t_end) {
    integrator.step(displacement, velocity, acceleration, load);
    t+=dt;
}
```

Figure 3.7.: Example program showing the usage of the Newmark integrator in DUNE

Providing specific member functions makes the implementation process into existing code and the modification of the time loop easier. This will especially be relevant for the simulation of fluid-structure interaction cases.

# 4. Validation cases for linear elasticity

To validate the code implementation for the equations of elastodynamics, two test cases are simulated. In a first step, a static simulation is presented in section 4.1. The scenario consists of a cantilever beam, which is clamped on one side. A force is applied on the free end resulting in a deflection of the beam under the load. The approximated solution is compared to the analytical results and thus a correct calculation and assembly of the stiffness matrix can be checked. For the dynamic validation, the vibration of a slender cantilever plate is investigated in section 4.2. The oscillation period of the discretized system is compared to the analytical calculated natural period waveform. The explicit and implicit time stepping schemes as well as the correct implementation of the mass matrix and lumping methods can therefore be checked.

## 4.1. Static analysis of a cantilever beam

For the static analysis case, the bending of a cantilever beam is analyzed. The goal hereby is to verify the correct calculation and assembly of the global stiffness matrix. Therefore, the deflection at the free end of the cantilever beam according to the theory is compared against the approximation obtained with finite elements.

The analytical solution of the vertical deflection at the free end of a Euler-Bernoulli beam is given as (corresponding to [Merkel and Oechsner, 2020]):

$$d_{analytical} = \frac{FL^3}{3EI}, \tag{4.1}$$

with the vertical force $F$ applied, beam length $L$ and the Young's modulus $E$. The second moment of area around the axis of displacement is given by:

$$I = \frac{BH^3}{12}, \tag{4.2}$$

where $B$ is the width and $H$ the height of the beam. The geometry and parameters are chosen in such a way, that the deformation due to the load is rather small compared to the overall length of the beam. Therefore, accurate results can be obtained by using the linear elasticity approach. Additionally, the difference of the analytical solution between the two most commonly used beam theories including Euler-Bernoulli and Timoschenko is negligible.

### Simulation setup

For the simulation, a two and three dimensional validation case is considered. The geometry of the beam used is similar to the one proposed in [MacNeal and Harder, 1985] with a length $L = 6m$ and a height $H = 0.2m$ (see figure 4.1). For the three dimensional setup a width $W = 0.1m$ is used, whereas in the other case the unit width is implied. Considering boundary conditions, one side of the beam is clamped, thus the displacement in all directions is restricted on that side. At the free end a unit load is applied in positive y-axis direction.


Figure 4.1.: Two dimensional simulation setup of the static beam bending scenario

For the discretization of the structure, ten $P_2$ lagragian, hexahedral elements are used in the three dimensional setup. In the respective two dimensional test scenario, the corresponding quadrilateral elements are chosen. The physical parameters consisting of the Young's modulus $E$, Poisson ratio $\nu$ and applied force of the validation case are given in table 4.1.

| | | |
|---|---|---|
| Reference force | $F$ | $1N$ |
| Young's modulus | $E$ | $10^7 N/m^2$ |
| Poisson ratio | $\nu$ | $0.3$ |

Table 4.1.: Physical parameters of the static beam bending scenario

The static system is given by $Ku = f$ and solved by applying a preconditioned conjugate gradient method. The convergence criterion is set to $10^{-8}$. As preconditioner, an incomplete LU-decomposition is utilized.

### Results

First, the overall deformation of the cantilever beam is analyzed. The comparison of the vertical deflection values at the free end and the corresponding errors are discussed afterwards. Due to the unit force applied in positive y-axis direction, the beam is bending upwards with the highest deformation occurring at the free end. The vertical displacement of the three dimensional cantilever beam is shown as cross section view in figure 4.2. The

single load induces a linear bending moment behaviour and thus a parabolic deformation profile. This corresponds to the deflection of the cantilever beam obtained by simulation. The two dimensional case yields a similar deformation result. There the overall deformation is smaller, due to the pseudo unit length taken as width.

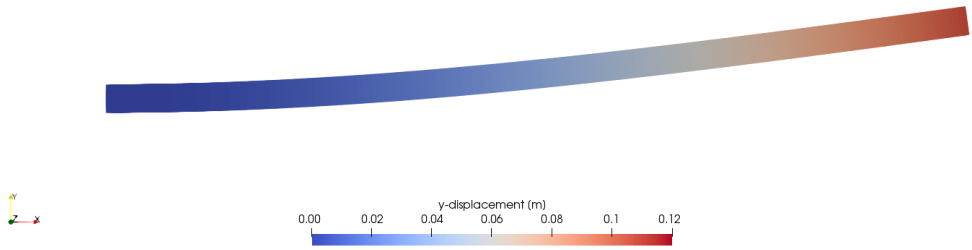



Figure 4.2.: Cross section view of the vertical displacement 3D cantilever beam (the deformation of the geometry is scaled by a factor of 5 for better visualization)

The results of the two different simulation cases, the corresponding analytical solutions and different error measures are given in table 4.2. Considering the two dimensional simulation first, the deflection at the beam tip approximated with quadratic finite elements is very close to the result corresponding to the Euler-Bernoulli beam theory. The absolute error is of the order of $10^{-6}$, whereas the relative error is at around $10^{-4}$. This results in a percentage error of $0.0185\%$. For the three dimensional case, the difference between the result given by the theory and the approximated solution is bigger. The errors are in the order of $10^{-4}$ for the absolute and $10^{-3}$ for the respective relative error. The error in percentage is given by $0.36\%$. Considering the type and number of elements used, the accuracy of the approximation in both simulation setups is considered as sufficient. Concluding the static validation case, it is assumed that the calculation of the overall stiffness matrix $K$, including element wise calculations and assembly is correct.

| | analytical | approximate | absolute error | relative error | percentage error |
|---|---|---|---|---|---|
| 2D | 0.0108 | 0.010798 | $2.0 \cdot 10^{-6}$ | $1.85 \cdot 10^{-4}$ | $0.0185\%$ |
| 3D | 0.108 | 0.108396 | $3.96 \cdot 10^{-4}$ | $3.67 \cdot 10^{-3}$ | $0.36\%$ |

Table 4.2.: Analytical and approximate solution for the two and three dimensional beam bending simulation including the different error measures

## 4.2. Oscillation of a slender plate

The second test scenario consists of a cantilever plate, which is oscillating due to an initial deflection. The overall simulation setup is similar to the one given in [MathWorks, ]. In a first step, the static deformation based on a unit load is calculated. This result is used as initial deflection for the dynamic simulation. Afterwards, the approximated period of the oscillation is compared to the analytical solution. For the transient calculation, the calculation and assembly of the mass matrix and implementation of the HRZ-lumping schemes are validated. Additionally, the time stepping methods are checked. Those consist of the explicit RK5(4) scheme and the Newmark-beta method as direct, implicit integrator.

The frequency at which a cantilever plate of length $L$, width $W$ and height $H$ is oscillating can be determined by calculating it's eigenfrequency:

$$\omega_0 = 3.516 \cdot \sqrt{\frac{EI}{L^4 h \rho}}, \tag{4.3}$$

and

$$f_0 = \frac{\omega_0}{2\pi}, \tag{4.4}$$

with the Young's modulus $E$, density $\rho$ and second moment of area $I$. The eigenfrequency or natural frequency of a system is the frequency the mechanical system tends to oscillate at in absence of damping or external loads. The oscillation period is given as:

$$T_0 = \frac{1}{f_0}. \tag{4.5}$$

The natural frequency of the discretized system can be found using a modal analysis approach. Therefore, the generalized eigenvalue problem of following form needs to be solved using the stiffness matrix $K$ and mass matrix $M$ (see [Dinkler, 2017]):

$$Kx = \omega^2 Mx, \tag{4.6}$$

with $\lambda = \omega^2$ being the distinctive eigenvalues of the system and the eigenvectors $x$. Solving the full system of equations for all eigenvalues given in equation 4.6 is very demanding in terms of computational cost. In this case only the smallest eigenvalue is of importance, due to it's correspondence to the lowest frequency (natural frequency). The analytically calculated eigenfrequency and thus it's eigenvalue are known. With algorithms like the inverse iteration, the approximated eigenvalue of the discretized system can be calulated efficiently by taking the analytical solution as inital guess.

The parameters of the simulation setup are again chosen in such a way, that the overall deformation of the plate is small compared to it's geometry size.

### Simulation setup

In contrast to the static validation case, only the three dimensional test case is discussed. For both dimensions the results obtained were very similar in terms of accuracy, thus the two dimensional setup is omitted here. Considering the discretization of the structure, 120 $P_2$ lagrangian, tetrahedral elements are used. The mesh of the plate is shown in figure 4.3.

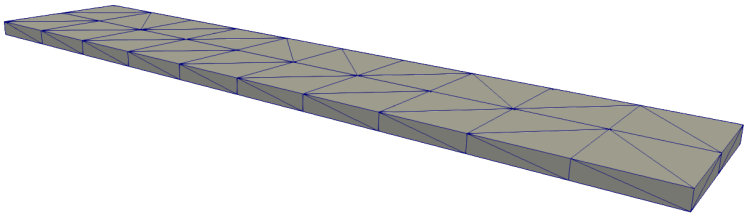


Figure 4.3.: Discretization of the plate with quadratic tetrahedral elements

The physical constants as well as the parameters used for the time stepping schemes are given in table 4.3. For the Newmark-beta method the coefficients beta and gamma are chosen in a way that the implicit method achives unconditional stability as discussed in section 2.4. The time step size is chosen rather large compared to the one calculated by the explicit RK5(4) scheme, which is around the order of $10^{-8}s$ to be numerically stable.

| | | |
|---|---|---|
| Young's modulus | $E$ | $3 \cdot 10^7\ N/m^2$ |
| Poisson ratio | $\nu$ | $0.3$ |
| Density | $\rho$ | $0.00077\ kg/m^3$ |
| RK5(4) | $\Delta t_{RK5(4)}$ | $adaptive$ |
| Newmark-beta | $\Delta t_{Newmark-beta}$ | $0.0001s$ |
| | $\beta$ | $0.25$ |
| | $\gamma$ | $0.5$ |

Table 4.3.: Physical parameters of the dynamic plate oscillation scenario

The simulation procedure is started by calculating the static deformation of the plate by applying a unit load at the free end. The boundary conditions are similar to the static validation case. The result is used as initial condition for the transient analysis. The accuracy of the static result is similar to the three dimensional case considered in section 4.1. For the dynamic simulation, the velocity and acceleration at the clamped end are restricted in addition to the displacement.

### Results

In a first step, the physical behaviour of the beam is discussed. At time $t = 0.0s$, the plate is in it's initial deflection corresponding to the static deformation. This resembles a bending in positive y-axis direction. Starting the time stepping shows the motion of the structure. It starts oscillating, where the amplitudes are given by the positive and negative value of the initial deformation. At around $t = 0.00175s$ the plate reaches the state of no deformation. Afterwards, the negative amplitude is reached at a time close to $t = 0.0035s$. The vertical displacement for different times during the oscillation are shown in figure 4.4. To ensure the correct assembly, calculation and lumping of the mass matrix, the fundamental frequency of the discretized system is compared to the analytical one. For the approximation of the eigenvalues and thus the oscillation period, an inverse iteration is used. The analytical calculated eigenvalue from equation 4.3 is used as shift and therefore as initial guess for $\lambda_{approx}$. That way a rapid convergence is achieved and a fast approximation of the discretized systems oscillation period is obtained. For the inverse iteration, equation 4.6 can be rewritten to:

$$(K - \mu M)x = (\omega^2 - \mu)Mx, \tag{4.7}$$

with the shift $\mu$. The approximated eigenvalue found is in close proximity to $\mu$, hence a good initial guess accelerates convergence. With forward and backward substitutions the iterations of the scheme are given by:

$$\begin{cases} y_{p-1} = Mz_{p-1} \\ (K - \mu M)z_p = y_{p-1} \end{cases} \tag{4.8}$$

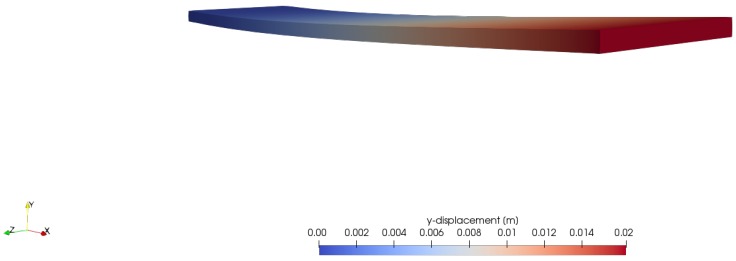
where $z_p$ is the respective eigenvector. The approximated eigenvalue can be retrieved by the Rayleigh quotient:

$$\lambda_p = \frac{z_p^T A z_p}{z_p^T z_p}. \tag{4.9}$$

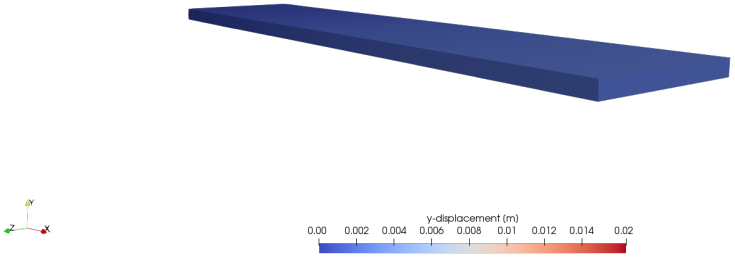
The results for the consistent and HRZ-lumped mass matrix as well as the respective error measures are shown in table 4.4. Both approaches yield similar results. The absolute errors are in the order of $10^{-5}$ and the relative ones at around $10^{-3}$. Based on the obtained approximation, the correct implementation of the mass matrix can be assumed.

| mass matrix | analytical | approximate | absolute error | relative error | percentage error |
|---|---|---|---|---|---|
| consistent | 0.007841 | 0.007885 | $4.4 \cdot 10^{-5}$ | $5.61 \cdot 10^{-3}$ | 0.56% |
| HRZ-lumped | 0.007841 | 0.007894 | $5.3 \cdot 10^{-5}$ | $6.76 \cdot 10^{-3}$ | 0.68% |

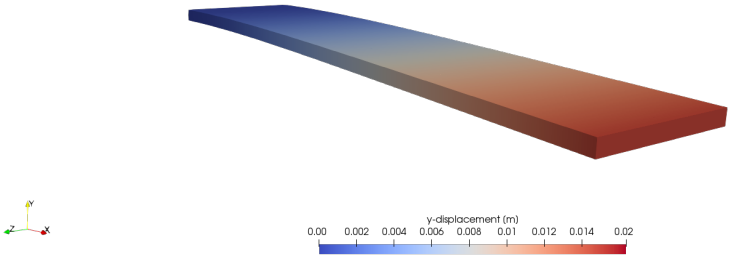
Table 4.4.: Analytical and approximate solution of the plate oscillation validation case for a consistent and lumped mass matrix, as well as the error measures




(a) t=0.0s




(b) t=0.00175s




(c) t=0.0035s

Figure 4.4.: Vertical deformation of the plate at different times (deformation of the geometry is scaled by 5 for better visualization)

The overall simulation run for $t = 0.025s$, including approximately three oscillation periods. The vertical displacement at the free end of the plate for different time stepping methods is shown in figure 4.5. The amplitude of the oscillation is given by the initial vertical displacement. A difference between both time stepping methods can mainly be seen in the amplitude resembling the maximum deflection of the plate tip. This might come from different time steps used for the schemes, as well as the different orders of accuracy of the methods. Taking into account the oscillation periods, both methods show the correct behavior.

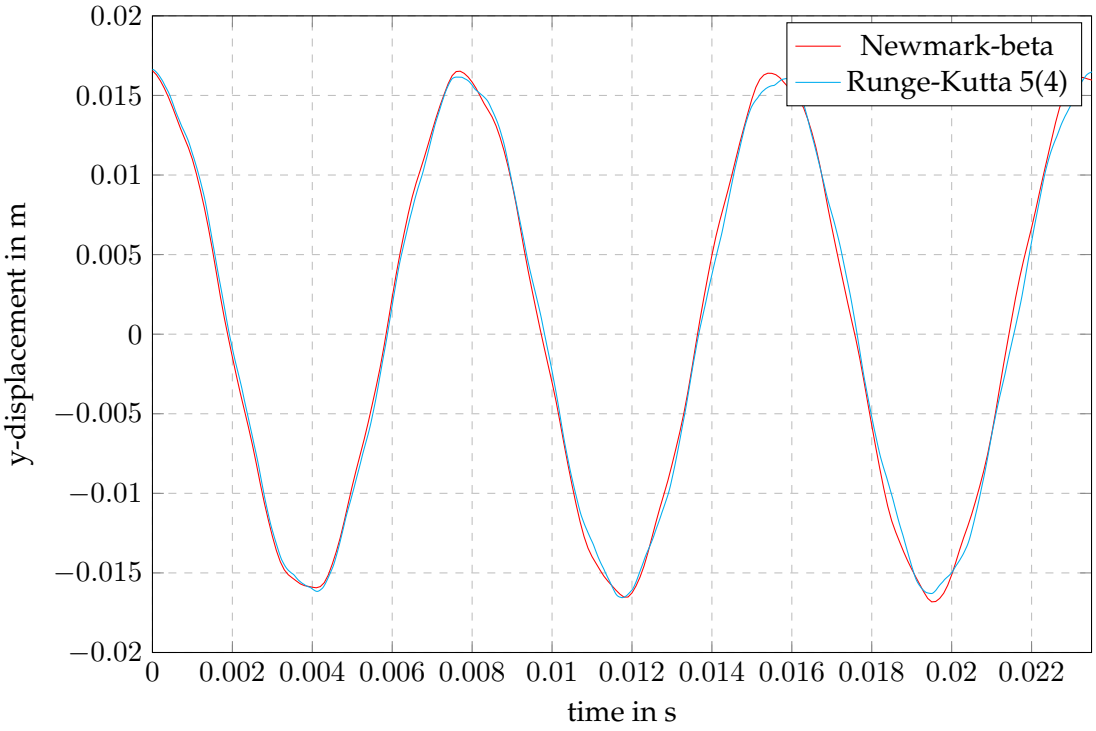



Figure 4.5.: Vertical displacement of the free end of the plate for the explicit RK5(4) and implicit Newmark-beta method

# 5. Background on partitioned simulation

Partitioned simulations are characterized by a high flexibility, coupling independent solvers with different discretizations and time stepping methods. This advantage comes with the cost of additional work that has to be done regarding communication, data mapping, time interpolation and coupling schemes.

In the following, a brief introduction is given about the most common numerical schemes used in partitioned simulation based on the related literature. First, the data mapping methods are presented in section 5.1. Afterwards, time interpolation procedures are introduced in section 5.3 and serial coupling schemes are discussed in section 5.2.

## 5.1. Data mapping methods

One of the aspects of partitioned coupling is the handling of data between meshes. The different simulation domains are often discretized differently, which makes a mapping at the coupling interface necessary. For matching meshes as described by [Gatzhammer, 2014], where the nodes of both meshes coincide at the coupling boundary, the values can just be copied from one mesh to the other. Generally, non-matching grids are used for multi-physics simulations, where the real coupling interface $\Gamma$ is only approximated and thus the grids might not coincide (see figure 5.1). A specific case are non-conforming grids, where the meshes share the same geometric boundary, but the nodes don't coincide. Commonly used mappings are based on projections (nearest-neighbor/-projection) and radial-basis function interpolation as shown in the comparison carried out by [de Boer et al., 2007].

Additionally a mapping can be conservative or consistent. For a conservative mapping the the overall sum of a quantity is conserved. For example forces at nodes add up and thus should be mapped conservative such that the overall force value on a coupling interface in conserved. For a consistent mapping, the value is just copied. This makes sense for displacements, where nodes of local proximity at the interface should have the same value.

### Nearest-neighbor mapping

The most trivial mapping form is to use information from the geometrically, closest node of the other grid. This results in a so called nearest-neighbor mapping of first-order accuracy. For each node the closest one in the other mesh is searched based on the euclidean distance and the value is copied. Related to the mapping accuracy, [Gatzhammer, 2014] considers it to be the worst choice. Additionally, this approach can lead to instabilities at the coupling

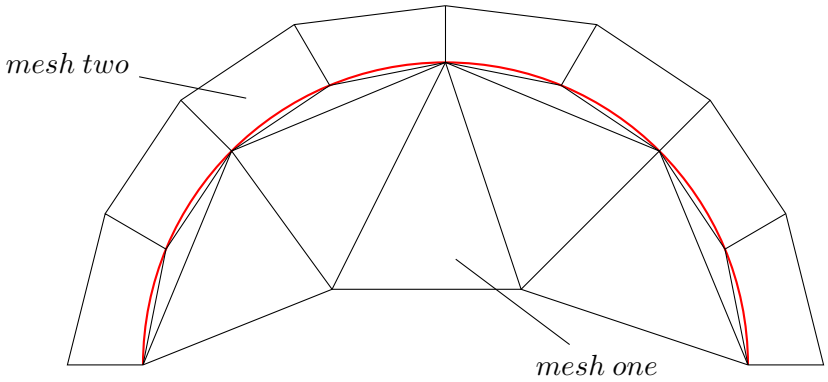



Figure 5.1.: Coupling of non-matching grids at the interface with the real boundary shown in red

interface, described by [Hertrich, 2019]. If one of the grids is substantially coarser then the other one, the meshes at the coupling interface are unphysically distorted.

### Nearest-projection mapping

A second-order accuracy method is achieved with the nearest-projection mapping. A node is orthogonally projected on the other mesh. The value there is interpolated from neighbouring nodes and used in the original point. For this kind of mapping additional information has to be provided (see [Yau, 2016]) related to the connectivity of the nodes that make up a face or edge.

### Radial-basis function interpolation

Another mapping approach is to build up an interpolant with the same dimension as the coupling interface problem. The nodes and data of one mesh is used for the creation, whereas the evaluation is done at the nodes of the other mesh. The most common type of functions used for the interpolation are radial-basis functions (RBF) with $\phi(x) = \phi(||x||)$. The RBF is build with globally or compactly supported functions. Compact supported RBF's use a radius in which the support is defined and set to zero outside:

$$\phi(||x||) = 0, \ ||x|| > r. \tag{5.1}$$

An overview of commonly used functions for radial basis interpolation is given in table 5.1 (discussed by [de Boer et al., 2007], [Gatzhammer, 2014] and [Rendall and Allen, 2008]). It was shown by [de Boer et al., 2007] that the first two RBF's shown in table 5.1 yield the most robust results. In many applications, it is easier to choose appropriate values for parameter $a$ than for the compact support radius $r$ concerning stability and accuracy of the

mapping. Due to the absence of projections, the problems connected to nearest-neighbor and nearest-projection mapping do not occure (see [Gatzhammer, 2014]).

| type | $\phi(\|\|x\|\|)$ | support |
|---|---|---|
| Thin-plate spline | $\|\|x\|\|^2 log\|\|x\|\|$ | global |
| Multi-quadric biharmonic spline | $\sqrt{a^2 + \|\|x\|\|^2}$ | global |
| Inverse multi-quadric biharmonic spline | $1/\sqrt{a^2 + \|\|x\|\|^2}$ | global |
| Gaussian | $e^{-a\|\|x\|\|^2}$ | global |
| Compact polynomial, $C^0$ | $(1 - \frac{\|\|x\|\|}{r})^2$ | local |
| Compact polynomial, $C^6$ | $(1 - \frac{\|\|x\|\|}{r})^8(32(\frac{\|\|x\|\|}{r})^3 + 25(\frac{\|\|x\|\|}{r})^2 + 8\frac{\|\|x\|\|}{r} + 1)$ | local |

Table 5.1.: Overview of different RBF's with global and local support with tuning parameter $a$ and compact support radius $r$

## 5.2. Coupling schemes

The combination and implementation of several solvers in an overall algorithm is another important task in partitioned coupling simulations. Most schemes can be grouped depending on their iteration behaviour and composition into explicit or implicit and serial or parallel procedures (see for example [Landajuela et al., 2017]).

Considering serial coupling, the solution process of the problem is staggered. One participant solves the associated subproblem and sends the solution to the other solver. The result is calculated there, send back to particpant one and the procedure starts again. Another approach is the calculation of the result by each participant at the same time. Afterwards the solution is send to the other solvers calculating the next step simultaneously. Such a procedure is called parallel coupling.

This thesis will mainly focus on serial methods of explicit and implicit nature. The notation used is similar to the one used in [Rusch, 2016] and [Wall, 1999]. The operators $F^n$ and $S^n$ resemble the calculation of a solver at a coupling step $n$. The result of an computation at a certain time is denoted by small letters, thus $f^{n+1}$ and $s^{n+1}$ would be the solutions obtained by solver $F$ and $S$ at $\tau = n + 1$.

### Serial, explicit coupling

A very simple algorithm in this category is the conventional serial staggered (CSS) scheme. Figure 5.2 shows exemplarily the procedure of the method. Solver one calculates the solution $f^{n+1}$ at the next coupling time step $\tau^{n+1}$ with the structural solution $s^n$ at $\tau^n$. The result is communicated to participant two, where the structural solution $s^{n+1}$ is calculated and send back to solver one. With this one coupling window is complete and the procedure starts again. The approach is computationally inexpensive, but lacks in accuracy. The

equilibrium of the coupling quantaties is not fullfilled, resulting in numerical instabilities for strong coupled problems.

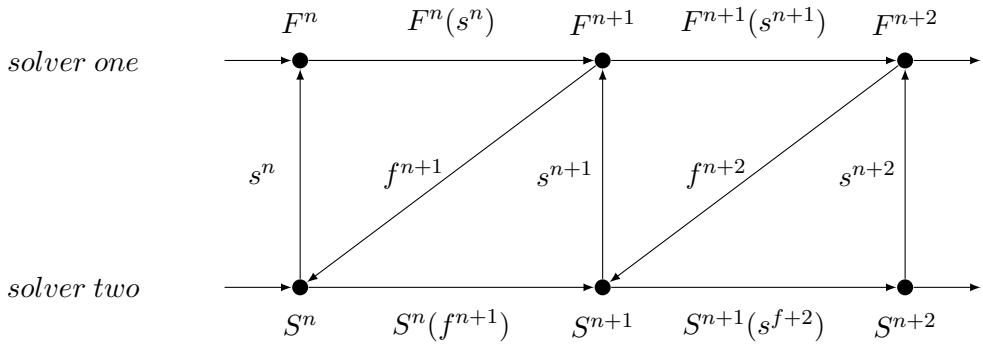



Figure 5.2.: Schematic procedure of the conventional serial staggered scheme

### Serial, implicit coupling

Serial implicit coupling is similiar to the explicit method, but additionally based on sub-iterations. This procedure is done to achieve an equilibrium between two solvers, thus the method yields a better accuracy than explicit coupling. The following notation was also used by [Rusch, 2016], [Yau, 2016] and [Gatzhammer, 2014]. The CSS scheme can be transformed into a fixed-point problem, where one iteration is written as:

$$s_{k+1} = S \circ F(s_k) = H(s_k), \quad k = 1, 2, \dots \ . \tag{5.2}$$

A more practical approach is to use the residual form of equation 5.2:

$$r_{k+1} = R(s_k) = S \circ F(s_k) - s_k = H(s_k) - s_k \tag{5.3}$$

The iteration procedure can be stopped by choosing an appropriate convergence criterion. The overall convergence of the system can be measured with an absolute criterion, taking the euclidean norm of the residual yields a scalar bound:

$$||r_{k+1}||_2 < \epsilon_{abs}. \tag{5.4}$$

The same can be done for consecutive sub-iterations, defining a relative convergence measure as:

$$\frac{||r_{k+1}||}{||\tilde{s}_{k+1}||} < \epsilon_{rel}. \tag{5.5}$$

The first iteration of this procedure is identical to the CSS approach. Afterwards, the obtained solution $s_k^{n+1}$ is not used for the next coupling window, instead the calculation

is restarted with the result as input quantity $s_k^n$ (regarding figure 5.2). This results in a new solution $f_{k+1}^{n+1}$ of participant one and thus $s_{k+1}^{n+1}$. The procedure is an iterative scheme proceeding until the convergence criterion is reached. The solution is refined by each iteration to match the equilibrium condition of the quantities at the coupling interface. The serial fixed point procedure described above is referred to as multiplicative Schwarz procedure. Due to stability reasons, a relaxation step is included called acceleration (see [Uekermann, 2016]):

$$\tilde{s}_k = S \circ F(s_k) = H(s_k) \tag{5.6}$$

and

$$s_{k+1} = \omega_k \cdot \tilde{s}_k + (1 - \omega_k) \cdot s_k, \tag{5.7}$$

where the first line describes the so called Picard-iteration similar to equation 5.2 and the second line the acceleration step included in every iteration. The most simple acceleration scheme is obtained by a constant relaxation with a fixed parameter $\omega^k$, which might yield acceptable results for some problems. A more sophisticated method was introduced by [Irons and Tuck, 1969] resulting in the dynamic Aitken choice of $\omega_k$:

$$\omega_k = -\omega_{k-1} \frac{(r_{k-1})^T (r_k - r_{k-1})}{||r_k - r_{k-1}||_2^2}, \tag{5.8}$$

with $r_k = \tilde{s}_k - s_k$. Nowadays, these group of acceleration methods is outclassed by quasi-Newton coupling schemes, which tend to be more stable for strongly coupled problems with instabilities. (discussed by [Uekermann, 2016] and [Mehl et al., 2017]). The acceleration step is changed considering the nonlinear fixed-point equation $R(s_k) = H(s_k) - s_k = 0$ from 5.3. The Picard-iteration step (equation 5.6) is included resulting in:

$$\tilde{R}(\tilde{s_k}) = \tilde{s_k} - H^{-1}(\tilde{s_k}) = 0. \tag{5.9}$$

Reformulating the residual of the Picard-iteration in terms of a Newton update yields:

$$J_{\tilde{R}}(\tilde{s}_k) \Delta \tilde{s}_k = -\tilde{R}(\tilde{s_k}), \tag{5.10}$$

with the Jacobi matrix of the residual. The linear system needs to be solved for the optimal correction $\Delta \tilde{s}_k$ and thus the final update reads as:

$$s_{k+1} = \tilde{s}_k + \Delta \tilde{s}_k, \tag{5.11}$$

where the Picard-iteration is altered by the correction term. The different quasi-Newton acceleration methods differ in the approximation of the Jacobi matrix of the residual operator. Two popular schemes are discussed in detail by [Scheufele and Mehl, 2016] and [Uekermann, 2016]: the Anderson acceleration and the generalized Broyden method. Compared to the Schwarz procedure methods, the quasi-Newton schemes are computationally

rather expensive, due to the fact that in every iteration step a linear system has to be solved. The stability and convergence behaviour is thus increased.

A further addition to the presented schemes is a manifold mapping algorithm for partitioned coupling (see [Blom et al., 2015]). The method is based on a multi-level approach, where a fine model represents the original problem, which is computationally expensive to evaluate. The coarse model is a surrogate or reduced order model based on the original problem and thus less accurate, but not as costly to compute. In a first step, an optimization problem regarding the coupling is solved on the coarse model. The solution is used to evaluate the fine model. The optimization on the coarse model is adjusted based on the solution via an affine mapping using a so called manifold mapping matrix. These steps are repeated till convergence is achived.

## 5.3. Time interpolation

Considering time-dependent, partitioned coupling problems, the involved solvers only exchange data at fixed synchronized points in time. The time interval between those points is called coupling window. Inside a coupling window, each solver uses an time integration scheme fitting to the specific sub-problem. The methods can either be explicit or implicit, low or high-order and may even consider different time scales.

### Order reduction with subcycling

A partitioned coupling with solvers using the same timestep size and thus the same amount of steps inside a coupling window is not very common. Often adaptive timesteping schemes are used in combination with fixed ones inside different solvers, resulting in a so called subcycling, as described by [Wall, 1999]. The coupling window acts as master, where solvers have to be synchronized at the end. One participant might only need one or a small amount of steps to reach the end of a coupling window, whereas the other solver utilizes a smaller step size (see figure 5.3), hence subsycles.The overall accuracy is determined by the lowest order time integration scheme used. Without a proper time interpolation, the accuracy of the overall system will degrade to order one as shown by [Birken et al., 2010], independent of the time integration methods used. This results in the need for a proper time interpolation to archive accurate high-order results (see [Rueth et al., ]).

### High-order time interpolation

To be able to achieve a high-order coupling scheme in time, a continuous representation of the solution has to be available inside a coupling window. An algorithm implementing this procedure inside the serial explicit and implicit coupling is called waveform iteration/relaxation, as discussed by [Rueth et al., ]. The relaxation is related to the acceleration step described in section 5.2, whereas the waveform regards the continuous representation of

the solution in time (see figure 5.3). Thus, in each coupling step an interpolant of at least order $\mathcal{O}(\Delta t^p)$ has to be found, where $p$ is the minimum degree of the interpolant to maintain the order. Normally, $p$ is chosen such that it has at least the degree of the time integration scheme with the lowest order. Which interpolation scheme to use is still part of ongoing research. In the waveform relaxation method presented by [Rueth et al., ], b-splines are used as interpolant. Often, the time integration in multi physics simulation is done with different methods. On a first glimpse, dense or continuous output would yield an very efficient way of construction an interpolant inside a coupling window. Due to the partitioned approach, the solvers are treated as black-boxes, where only a minimum amount of data is exchanged. Therefore, a method needs to be found that creates the interpolant efficiently without knowing which solvers are involved in the simulation.

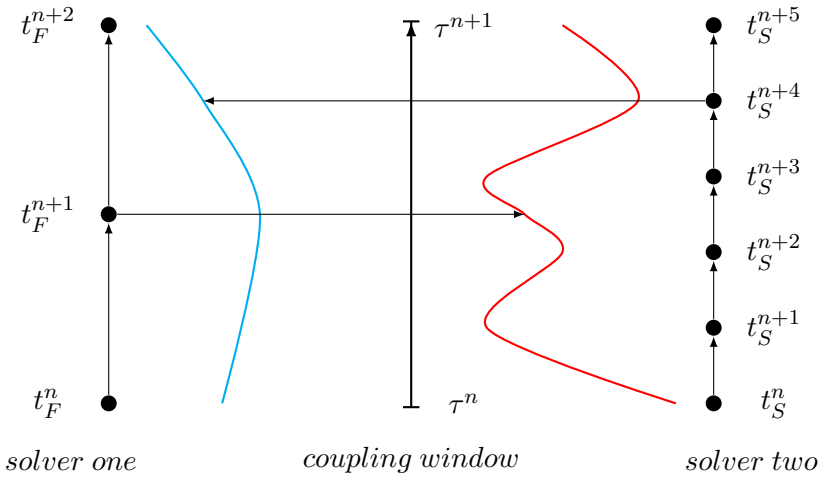



Figure 5.3.: Time interpolation approach between two coupled solvers with different time steps

# 6. A preCICE adapter for DUNE

In this chapter a new DUNE module for multi-physics simulation is introduced. In most parts it is a wrapper for the coupling-library preCICE introduced by [Bungartz et al., 2016] for partitioned simulation. Several open-source solvers were already modified by [Yau, 2016] (Calculix, Code Aster), [Chourdakis, 2017] (OpenFOAM) and [Rusch, 2016] (SU$^2$) to implement the necessary coupling functionality. For abstract numerics like a finite element library, [Hertrich, 2019] implemented a linear elasticity solver for solid mechanics in FEniCS with the preCICE coupling functionality. These implementations will be used as reference for the dune-precice module.

In a first step, a brief introduction is given about preCICE in section 6.1. The functionality and implementation of the dune-precice module is shown in chapter 6.2. The necessary steps of the implementation of the dune-precice module inside a dummy solver environment is discussed in section 6.3.

## 6.1. Coupling library preCICE

The coupling library preCICE is based on C++ and used as a tool for partitioned simulation in multiphysics. It handles the four major aspects that were mostly discussed in chapter 5 consisting of communication, data mapping, coupling schemes and time integration. Time integration is yet not fully supported in preCICE and still an object of ongoing research. The data mapping consisting of nearest-neighbor, nearest-projection and radial basis function interpolation is specified in a configuration file including the method, the parameters and the mesh where the mapping happens. The coupling scheme (explicit/implicit and serial/parallel) is also set there in addition to the coupling window size and the overall simulation time. For the implicit coupling the convergence criteria has to be stated. An acceleration method is optional as discussed before, but highly recommended. In a last step, the communication procedure is defined, consisting of the members participating in the simulation and which quantities are written or read by each participant.

To make use of the methods, preCICE offers an API to modify existing solvers for multiphysics simulation. The methods can be divided in several groups based on the functionality offered. In the following, the most important methods are presented, which are also used in the dune-precice module.

### Important methods of the preCICE API

At some point in the simulation procedure, the interface to preCICE has to be started to begin the coupling with other participants. With a call to `initialize()` the communication as well as the memory for necessary data structures is set up by the library. On the other hand, `finalize()` ends the communication, frees the memory and stops preCICE. To track the overall time of the simulation in regards to the coupling, the `advance()` method is called. It returns the time left inside a coupling window. This is mandatory for synchronization to enable the communication of data. The presented procedures are called steering methods.

```
double initialize();
double advance(double computedTimestepLength);
void finalize();
```

The most important status query is the status of the overall coupling. Depending on the state, different procedures can be triggered inside the simulation.

```
bool isCouplingOngoing();
```

Sometimes it is necessary to have conditional statements that are triggered based on a specific action. The most prominent method in preCICE using this feature is implicit coupling, where states need to be stored and reloaded based on the beginning and end of a coupling window to enable fixed-point iterations.

```
bool isActionRequired(const std::string &action);
void markActionFulfilled(const std::string &action);
```

The mesh access is necessary at the beginning of the simulation. The coupling boundary needs to be specified in order for preCICE to be able to map data values. In particular the vertex positions of the coupling boundary have to be known. For a nearest-projection mapping, the overall mesh connectivity of the coupling mesh has to be given additionally. Therefore, special methods are provided to define edges, faces and elements based on the vertices given. Additionally, the mesh ID can be retrieved, which needs to be used for the data communication.

```
int getMeshID(const std::string &meshName);
void setMeshVertices(int meshID, int size,
                     const double *positions, int *ids);
```

An important feature is the access and communication of data. The values at the coupling boundary can be written to preCICE or retrieved from it. Based on the data ID given, the communication is unique in a way that the receiver is known through the configuration file. The API provides functions for both scalar and vector valued quantities.

```
int getDataID(const std::string &dataName, int meshID);
void writeBlockScalarData (int dataID, int size,
                    const int *valueIndices, const double *values);
void writeBlockVectorData (int dataID, int size,
                    const int *valueIndices, const double *values);
void readBlockVectorData (int dataID, int size,
                    const int *valueIndices, double *values);
void readBlockScalarData (int dataID, int size,
                    const int *valueIndices, double *values);
```

Additionally to the presented methods, preCICE offers some more specific functionality. Those methods are not discussed in the scope of this thesis.

## 6.2. Functionality of the dune-precice module

The dune-precice module connects the coupling library preCICE with the numerics environment DUNE. An implementation like this will be called adapter from now on. In order to simulate multi-physics cases some basic methods and functionality need to be available, which are already provided by preCICE and discussed in section 6.1. Those include the data mapping at the coupling interface, the communication between different solvers and the coupling algorithm. Considering the coupling, both explicit and implicit schemes are available. The adapter implements the most important preCICE functions, but inside the DUNE framework some specific utility needs to be offered additionally. Thus some functions are added that handle:

- finding the coupling boundary of a DUNE grid
- converting data structures from DUNE to preCICE and the other way round
- storing and loading iteration checkpoints

Similar to the adapter written by [Rusch, 2016], some methods are just wrappers directly calling the preCICE function. Others offer a broader functionality considering the DUNE environment. All functions and internal variables for the coupling are implemented inside a templated class. The class object can be created for different dimensions and a variable parameter list. All methods are accessed via this object, allowing to implement the coupling in a minimal invasive manner, which will be shown in section 6.3.

### Setup of the coupling interface

To start the coupling with another solver, an adapter object needs to be created providing the coupling interface from dune-precice. The respective constructor takes coupling parameters stored in a class or structure, the coupling boundary tag and information about

distributed parallelization as arguments. The parameter class should contain the preCICE configuration filepath, solver- and mesh name. Additionally, it should store which data has to be written and read. At this point, the adapter is almost ready for communication, as the schemes to use for data mapping, communication and coupling algorithm are set.

In a next step the initialization method is called, which handles the coupling boundary and starts the communication. Depending on the basis and gridview, the adapter will extract the number of vertices and their coordinates at the coupling boundary and communicates them to preCICE. Afterwards the coupling with another solver is established. If the communication is set up, the initial time that is left inside the coupling window will be returned.

### Coupling interface functionality

The main coupling procedure mainly consists of reading and sending data, writing checkpoints and tracking the coupling time window. For sending and receiving data, the functions `read_blockvector_data()` and `send_blockvector_data()` are used, taking a DUNE blockvector of dimension two or three as input argument. Internally, the adapter selects the entries positioned at the coupling interface and either sends them to preCICE or overwrites them with the data received. The same functionality is also available for scalar values. For an implicit coupling, writing and reading checkpoints is necessary. Therefore, several functions triggering the storing and reloading of scalars and vectors are available. Tracking the coupling time window is done with `advance()`, which takes the time step length of the solver as input and calculates the remaining time that is left inside the actual coupling window. This is necessary to adjust the solver timestep in a way, such that the time between solvers can be synchronized for data communication. With `is_coupling_ongoing()`, the state of the coupling can be checked. The boolean value is often used to exit the solver time loop and finish the coupling with a call `finalize()`. The finalization of the coupling ends the communication and frees the memory related to preCICE.

## 6.3. Implementation into a dummy solver

To show the implementation of the dune-precice module into a minimal working example, a dummy solver is created in DUNE. To test the principal functionality the dummy is coupled to a test solver provided by the preCICE library. Thus the overall initialization and finalizing process, parameter settings and as well as data communication can be checked.

In a first step a structure is set up containing parameters necessary for preCICE (see figure 6.1). Hereby, the used methods and communication pattern is transferred with the configuration file. Additionally, the read and write data naming is specified as well as participant and mesh name. The naming can be chosen arbitrarily and only has to coincide with the configuration file for unique communication procedures.

```
// set preCICE coupling parameters
struct parameters {
    const std::string config_file      = "../precice-config.xml";
    const std::string participant_name = "SolverTwo";
    const std::string mesh_name        = "MeshTwo";
    const std::string read_data_name   = "dataOne";
    const std::string write_data_name  = "dataTwo";
};
```

Figure 6.1.: Necessary parameters for the communication of two solvers via the preCICE interface

The first step inside the main function (the major features are given in figure 6.2) is the setup of the MPI environment for parallel computation. The computation in this case is only done sequentially, but preCICE will use the rank and size variable for the interface construction. Afterwards the mesh is loaded into DUNE and a grid view is generated. Important quantaties are the coupling interface tag and the boundary entity vector, which will be used later on for the determination of the vertices on the coupling interface.

After the gridview is generated, a finite element basis can be created. In this case a first order Lagrangian basis is chosen. Now, the stiffness and mass matrices of the underlying problem can be calculated and assembled. In a next step, preCICE is started. Therefore, the structure variable coupling parameters is generated containing the members described before. Additionally the data structures for the read and write data are set up. These are block vectors with two dimensional entries. Now the preCICE object can be constructed. The dune-precice module uses templates for the most flexiblity, thus the problem dimension, the used read and write data structure as well as the parameters structure are given as template arguments. Thus the dimension of the problem can be easily changed. The same also holds for the parameters structure. To start the communication, the initialize method is called as member function of the coupling object.

With the status query of the coupling status, the time loop is controlled. A typical timeloop for the coupling consists of several steps. First is is checked if the data of the actual step should be saved for a implicit coupling procedure. Afterwards data is read from preCICE and it is processed. Then the next timestep is calculated as a minimum between the time left inside the coupling window and the actual timestep length of the solver. The modified data is written back to preCICE and the coupling is advanced. In case of implicit coupling, it is checked if the simulation is converged and if not, data might be reloaded. Otherwise, the timestep is finished with the update of time and iteration count. At the end of the program, preCICE is stopped.

```
const int dim = 2;

// set MPI environment
int commRank = mpiHelper.rank();
int commSize = mpiHelper.size();

// load mesh and setup grid view with basis
// ...

// initialize preCICE
parameters coupling_parameters;
Dune::preCICE<dim, vectortype, parameters>
coupling(coupling_parameters, dune_boundary_interface_id,
         commRank, commSize);
coupling.initialize(basis, boundaryEntity);

// start time loop
double t = 0.0, dt = 0.1;
int iteration_count = 0;

while (coupling.is_coupling_ongoing()) {
    if(coupling.is_save_required()) {
        coupling.save_current_state(write_data, t, iteration_count);
        coupling.mark_save_fullfilled();
    }
    coupling.read_blockvector_data(read_data);
    write_data += read_data;
    dt = std::min(coupling.precice_timestep_length, dt);
    coupling.send_blockvector_data(write_data);
    coupling.advance(dt);
    if(coupling.is_load_required()) {
        coupling.reload_old_state(read_data, t, iteration_count);
        coupling.mark_load_fullfilled();
    } else {
        t += dt;
        ++iteration_count;
    }
}

// finalize preCICE
coupling.finalize();
```

Figure 6.2.: A short program showing the major features of DUNE and preCICE

# 7. Validation cases for fluid structure interaction

In this chapter three fluid-structure interaction scenarios are presented for the validation of the dune-precice adapter. Therefore, the linear elasticity solver from chapter 3 is coupled to the open-source fluid solver OpenFOAM. The first simulation setup consists of a flexible flap that is wall-mounted inside a channel. This case was also used by [Rusch, 2016] and [Hertrich, 2019] for testing. For a numerically more intense problem [Wall, 1999] and [Uekermann, 2016] used a lid-driven cavity with a flexible bottom, which is used as second validation setup. The third problem consists of a flap attached to a cylinder inside a channel, called the fluid-structure interaction three benchmark (see [Turek and Hron, 2007]). The results are compared to a reference either from literature or another simulation. Additionally, problems concerning the validation cases are discussed.

## 7.1. Flexible flap in a channel

The first test case comprises a laminar flow in a channel, which is deforming a flexible flap that is mounted inside. Firstly, the overall correctness of the implemented adapter is evaluated against a reference solution calculated with a dealII-OpenFOAM coupling. This should show that the mandatory functionalities like setting the coupling boundary, data mapping and coupled time-stepping work as intended. The validation test finishes with the comparison of two different coupling methods, namely explicit and implicit coupling.

### Simulation setup

In the following, an overview of the simulation setup is given. Afterwards, the meshing and material parameters are presented. A laminar flow field inside a channel with the length $L = 6\ m$, height $H = 4\ m$ and width $W = 1\ m$ is used for the fluid domain. The boundary conditions are set for the fluid velocity and pressure. On the left side, an inflow of $10\ m/s$ is given, while the pressure gradient is set to $0$. No-slip conditions are applied to the top and bottom of the channel, resulting in a velocity of $0\ m/s$ at the wall. Again, the pressure gradient is set to $0$ in both cases. The right side contains the channel outflow, where the pressure is set to $0$ as well as the velocity gradient. A brief outline of the setup is given in figure 7.1. To be able to simulate a fluid-structure scenario, the mesh needs to be deformed. This yields the third variable, for which boundary conditions have

to be set. The mesh nodes at the inflow and outflow are not allowed to move. The respective points at the top and bottom wall are allowed to move in tangential direction. At the coupling interface to the structure, the velocity and pressure gradient is set to $0$. The mesh displacement is not constrained. Inside the channel, a flexible flap of dimensions $L = 0.1\ m$, $H = 1.0\ m$ and $W = 1.0\ m$ is mounted to the bottom wall at the middle of the overall length. To resemble the fixed placement of the flap at the wall, the displacement and velocity is set to zero at the bottom. The other boundary entities are imposed with a coupling condition that communicates to the preCICE adapter and thus allows the structure to move freely inside the fluid domain. OpenFOAM only allows to simulate three dimensional domains. For this case periodic boundary conditions are chosen for the front and back of the channel. The setup in two dimensions just sets the boundaries in the width direction to empty.

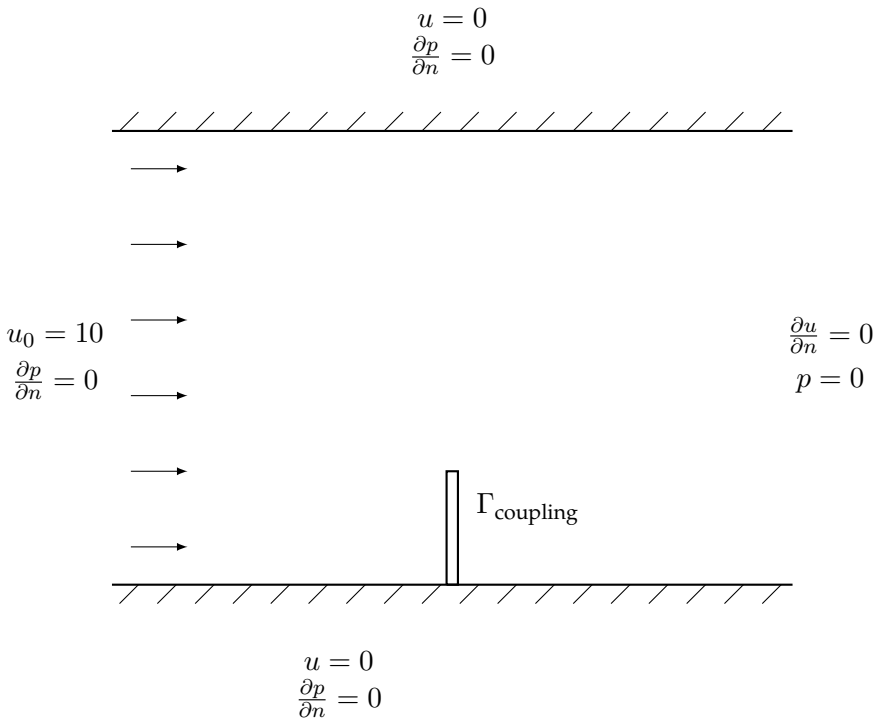
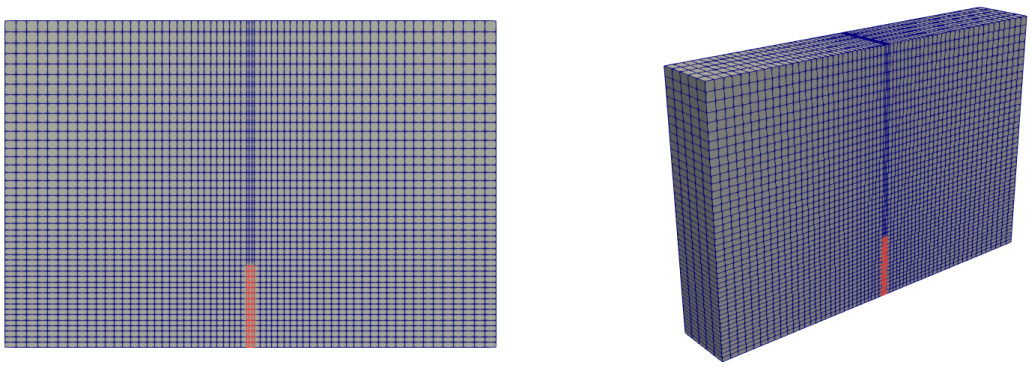



Figure 7.1.: Simulation setup of the flexible flap in channel validation testcase

For the fluid and solid domain a structured mesh is used. The solid mesh is uniform and discretized with quadrilateral, Lagrangian $P_1$ elements in 2D. In three dimensions hexahedron, Lagrangian $P_1$ elements are used. The fluid uses a non-uniform mesh that uses a finer discretization around the coupling interface and a coarser one further away from the structure. The nodes of the fluid and solid grids coincide at the coupling boundary. For the data mapping of the displacements and forces, a nearest-neighbour mapping is used. The

structured meshes for both dimensions are shown in figure 7.2. The blue mesh represents the fluid domain, where as the red grid resembles the structure.



(a) 2D domain (b) 3D domain

Figure 7.2.: Structured meshes of the flexible flap in a channel scenario

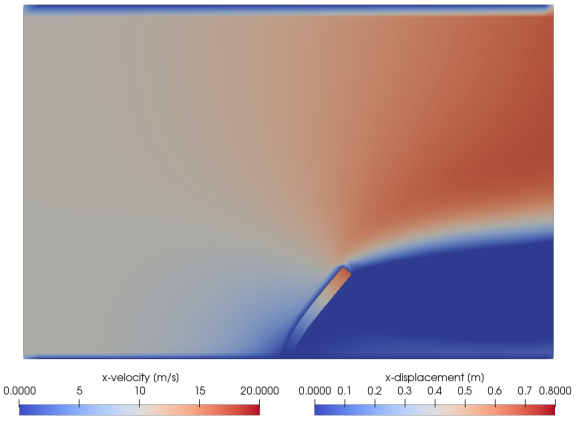
For the fluid-solver, an implicit euler scheme is chosen with a fixed time step size of $\Delta t = 0.01\ s$. On the structure side the explicit Runge-Kutta 4(5) method with adaptive time stepping is used. The coupling window, material and flow properties are given in table 7.1. Both the explicit and implicit coupling approach is tested. For the latter one, the convergence measures are set to $10^{-6}$ for both the absolute and relative one. Additionally, a maximum number of 50 iterations per coupling window is enforced. For the acceleration scheme, a dynamic Aitken under-relaxation with an initial relaxation factor of $0.1$ is chosen.

| | | |
|---|---|---|
| fluid density | $\rho_F$ | 1.0 $kg/m^3$ |
| dynamic viscosity | $\nu$ | 0.001 $kg/(ms)$ |
| inlet velocity | $u_0$ | 10.0 $m/s$ |
| structure density | $\rho_S$ | 3000 $kg/m^3$ |
| Young's modulus | $E$ | 400000 $N/m^2$ |
| Poisson ratio | $\nu$ | 0.3 |
| coupling window | $\tau$ | 0.01 s |
| simulation time | $t_{end}$ | 22 s |

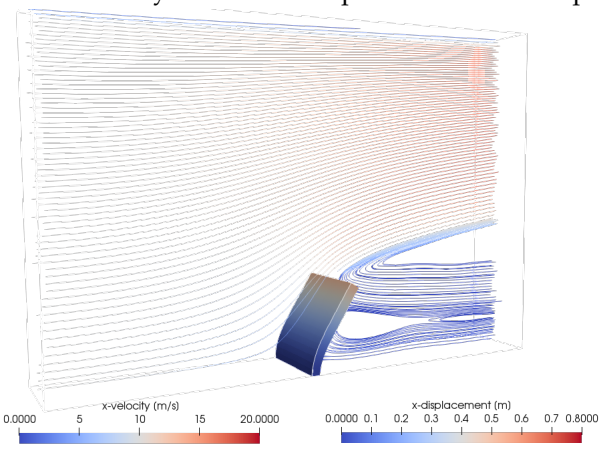
Table 7.1.: Physical parameters of the flexible flap in a channel scenario

### Results

In a first step the simulation results are analysed based on the velocity field of the fluid and the overall displacement of the structure. Due to the constant inflow, the elastic flap bends backwards and starts oscillating. The oscillation period for the given parameters is around $T = 4.5\ s$ A picture of the horizontal velocity field and the maximum x-displacement of the flap is given in figure 7.3a at $t = 2\ s$. The velocity field on the left side of the channel, near the inflow, is almost uniform. The bending flap acts like a nozzle and the fluid begins to accelerate. The horizontal velocity components behind the flap are close zero. To understand the fluid motion there, the streamlines of the flow (see figure 7.3b) are visualised. The flow detaches from the flap tip and creates several vortices. During the simulation they build up a stable oscillation behaviour that interacts with the movement of the flap. As the 3D case is just an extrusion of the 2D simulation setup, the overall displacement and flow behaviour stays the same.




(a) Horizontal velocity field and x displacement of the flap at t=2s




(b) Structured meshes of the flexible flap in a channel scenario

To verify the numerical results generated with the coupling adapter, the displacement of the flap tip is tracked over time. The results in form of the x-displacements are plotted in figure 7.4 over a time interval of 22 $s$, which resemble roughly five periods of the oscillating behaviour. The reference solution calculated with a coupling of dealII-OpenFOAM is showed as black curve. The results of both explicit and implicit coupling with the DUNE-adapter are given in red and blue. Considering the first three periods, the solution calculated with DUNE is very close to the reference result. With increasing simulation time, the results shift slightly away from the reference solution. Comparing the different coupling methods, they show the same behaviour and show only small differences in the positive peak values. A better approximation result might be obtained by changing from the low order of the elements used to a higher one for the solid mesh, which would also increase the computation time significantly. Especially the explicit coupling that is applied to this case without a relevant loss of accuracy or occurrence of numerical instabilities accelerates the computation. This is related to a weak coupling of the fluid and solid. Despite the rather large displacement of the flap, the bending only affects the flow field on a local scope around the structure.

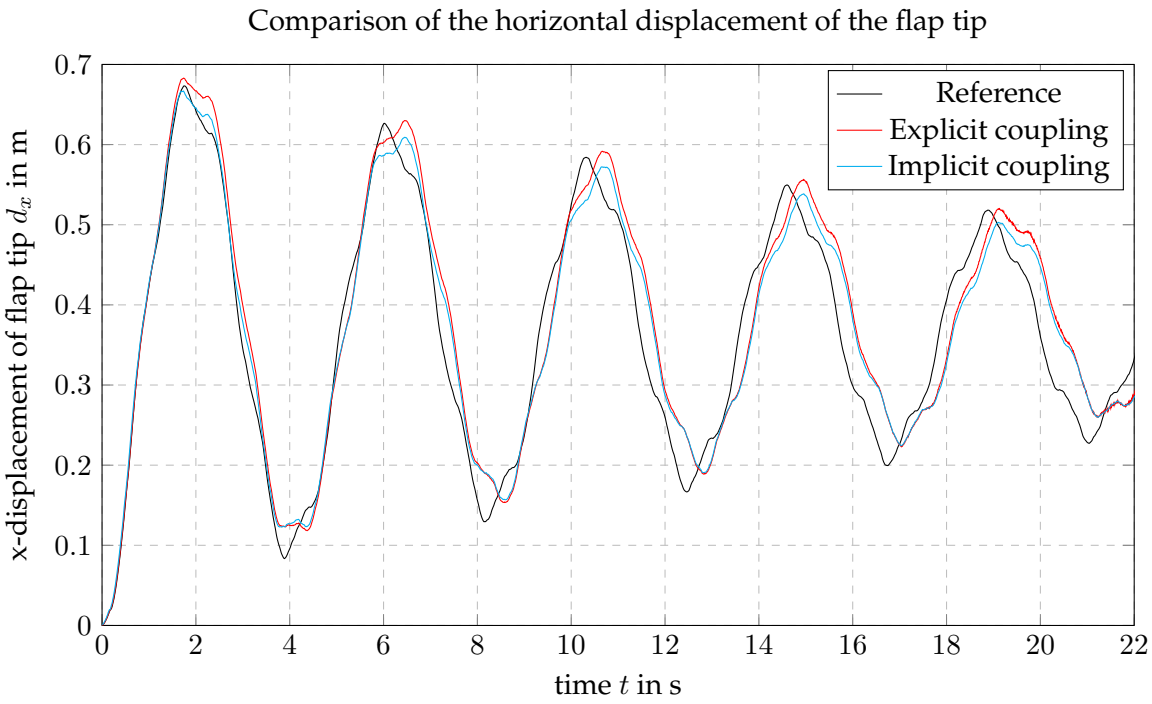



Figure 7.4.: Comparison of the coupling results with the DUNE-adapter to the reference solution calculated with the dealII-adapter

## 7.2. Lid-driven cavity with flexible bottom

The first validation case was numerically inexpensive and stable almost independent of the coupling scheme and parameters. To also simulate a more demanding case the popular lid-driven cavity with flexible bottom is chosen (see [Wall, 1999]). The simulation setup and material/flow parameters are similar to the one used by [Vzquez, 2007] and [Bathe and Zhang, 2009].

### Simulation setup

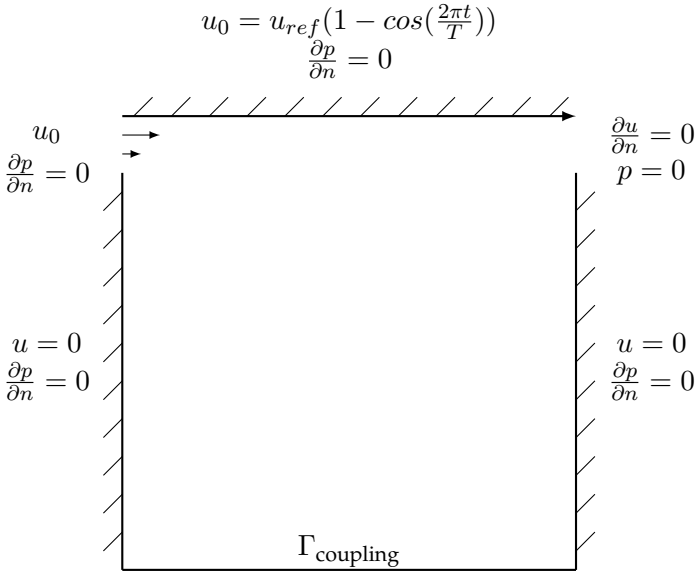



Figure 7.5.: Simulation setup of the lid-driven cavity with flexible bottom validation case

In figure 7.5 the simulation setup of the lid-driven cavity with flexible bottom is shown. The fluid domain consists of a square of length $L = 1.0\ m$. As in the original benchmark case the top wall is moving with a given velocity, in this case $1 - cos(\frac{2\pi t}{T})$. The pressure gradient at the top wall is set to zero. The left and right boundary is set to a no-slip condition, where the velocity and pressure gradient is set to 0 at the wall. A small inflow and outflow is added in order to fulfill the incompressibility condition of the fluid, thus a volume change inside the cavity due to the flexible bottom. Those are positioned at a height of $0.875\ m$ ranging to the top of the cavity. The inflow is located at the left side and consists of a linear inlet velocity profile. The velocity at the top and bottom of the inflow is linearly interpolated over it's height. At the outflow, pressure and velocity gradient are set to zero. The mesh movement at all sides is set to zero, besides the bottom wall, which is flexible and therefore the coupling boundary with the structure. The solid part resembles a membrane and has a length of $L = 1m$ and a thickness of $t = 0.002m$. The upper side of the

structure is set as the coupling boundary. On the left and right boundary, the movement is restricted.

Similar to the first validation case, structured meshes are used for both the fluid and solid domain. Again $P_1$ quadrilateral, Lagrangian elements are used for the structure. On the coupling interface, the nodes of the fluid and structure mesh coincide. Thus, the nearest-neighbour mapping is just setting the values from fluid nodes directly to the solid ones similar to the first validation case.

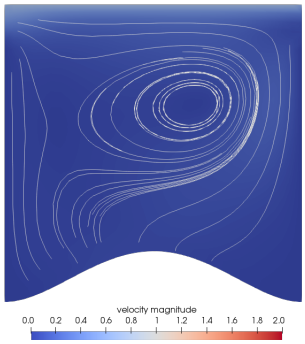
The implicit-euler scheme is set as fluid solver, as is the Runge-Kutta 4(5) for the structure side. The properties of the fluid flow and the solid can be seen in table 7.2. For the oscillation period of the moving wall $T = 5s$ is chosen. With the given $u_{ref}$, the velocity at the top boundary thus varies between $0m/s$ and $2m/s$. The most critical parameter in this case is the membrane stiffness, indirectly reflected by the Young's modulus $E$. How the parameter influences the overall simulation behaviour will be discussed later. For the implicit coupling, the error criteria are set to $10^{-4}$ and a quasi Newton scheme is used for acceleration.

| | | |
|---|---|---|
| fluid density | $\rho_F$ | 1.0 $kg/m^3$ |
| dynamic viscosity | $\nu$ | 0.01 $kg/(ms)$ |
| reference velocity | $u_{ref}$ | 1.0 $m/s$ |
| structure density | $\rho_S$ | 500 $kg/m^3$ |
| Young's modulus | $E$ | 25000 $N/m^2$ |
| Poisson ratio | $\nu$ | 0.0 |
| coupling window | $\tau$ | 0.01 s |
| simulation time | $t_{end}$ | 35 s |

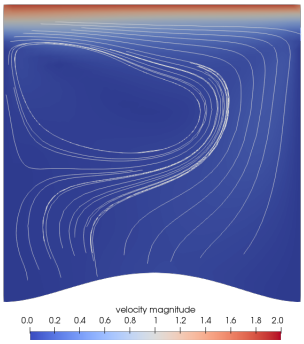
Table 7.2.: Physical parameters of the lid-driven cavity with flexible bottom scenario

### Results

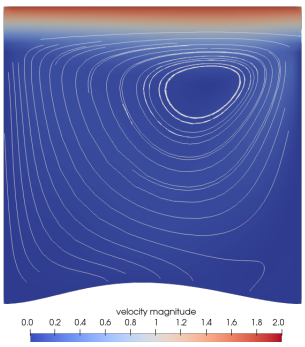
Looking at the overall physical behaviour of the fluid and membrane first, the deformation of the structure and streamlines of the fluid are tracked over a full oscillation period. Therefore, the time interval between $16s$ to $20s$ is chosen, where the simulation shows a stable oscillation of the solid. Due to the moving wall on the top and the induced velocity, the fluid inside the cavity is accelerated and forms a time-dependent vortex. The flexible membrane is affected by the flow and bends upwards and downwards, depending on the oscillating wall velocity. The behavior is shown in figure 7.6.



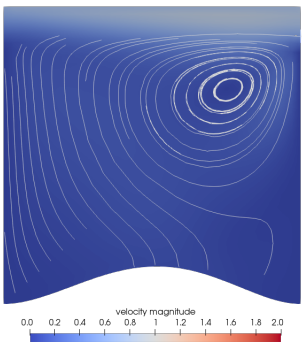

(a) t=16s



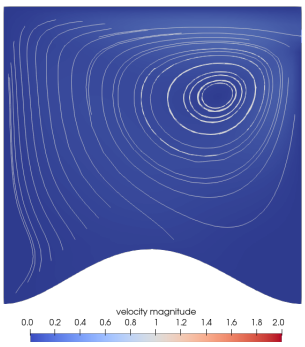

(b) t=17s



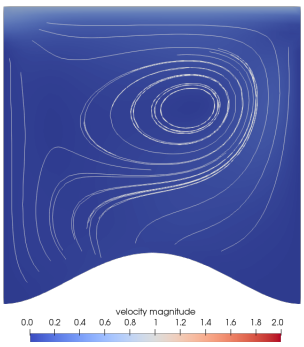

(c) t=18s




(d) t=19s




(e) t=20s




(f) t=21s

Figure 7.6.: Velocity field and flow streamlines over a full oscillation period for implicit coupling

In a next step, the vertical displacement of the membrane mid point is analysed. In figure 7.7 the results for the explicit and implicit coupling are plotted over a time span of $35s$. By adapting the stiffness parameter of the structure, the intensity of the coupling can be controlled. In this case, the stiffness value is small and thus a strong coupling occurs. The bending of the membrane influences the whole flow field of the cavity and introduces numerical difficulties, compared to the first validation case. The explicit coupling is numerically instable and the membrane starts to oscillate at a high frequency. At around $5s$ the simulation crashes due to the big mesh distortion. The implicit coupling approach shows better results, due to the subiterations in each timestep, to fulfill the equilibrium condition. After $15s$ a stable oscillation is reached, that also shows the period of $T = 5s$ from the moving wall. The traditional benchmark case is calculated with a membrane stiffness of 250, which resulted in deformations and mesh distortions that led the simulation to crash. Large deformations are one kind of nonlinearity that need special treatment inside the structure solver. The linear elasticity solver implemented with DUNE is only able to reproduce correct results for rather small deformations, due to the reference to the stiffness matrix in the undeformed state. Therefore, the stiffness of the structure has been adapted for this case similar to [Bathe and Zhang, 2009]. Compared to this validation case, the deformation of the membrane shows a different behavior for the same Young's modulus, due to a different modelling of the inlet. The displacement shown in 7.7 is closer to the results given by [Wall, 1999], despite the lower Young's modulus used there.

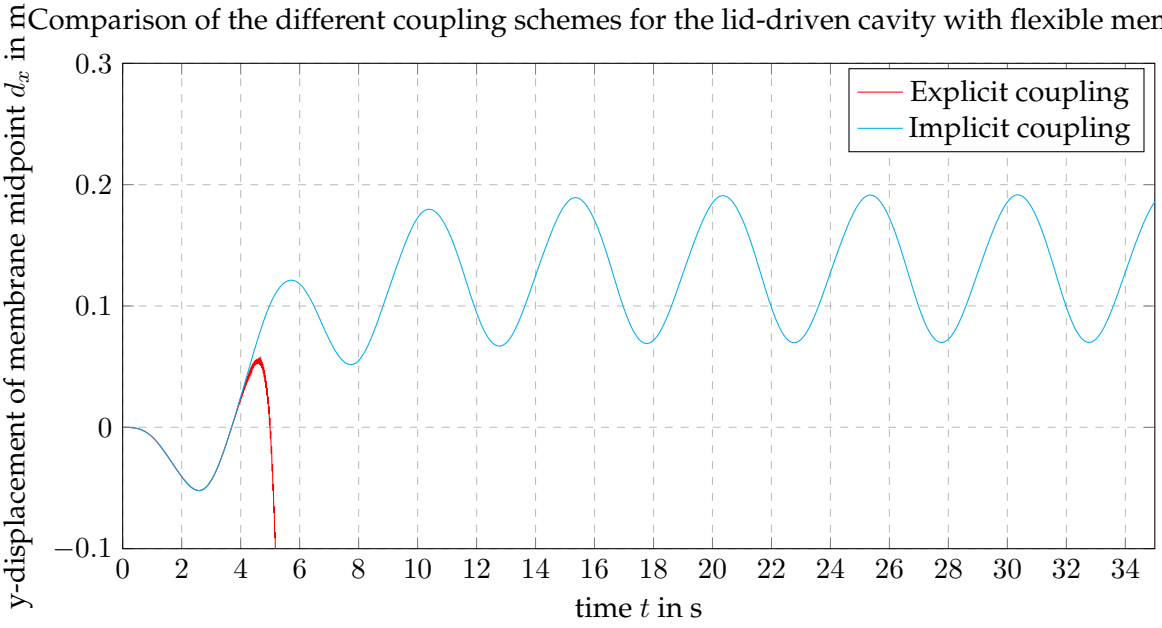



Figure 7.7.: Vertical displacement of the membrane midpoint shown for explicit and implicit coupling

## 7.3. Fluid-strucuture interaction three - benchmark

Concluding the validation, the so called fluid-structure interaction three benchmark is simulated. The test case was introduced by [Turek and Hron, 2007] and represents an even more demanding coupling than the lid-driven cavity with flexible bottom and is highly nonlinear. The benchmark was further investigated based on partitioned simulation approaches by [Breuer et al., 2012] and [Bungartz et al., 2010]. Similar to the second validation case the inflow is oscillating, resulting in an unsteady flow inducing instabilities during the simulation. Especially the so called added-mass effect (see [Causin et al., 2005] and [van Brummelen, 2010]) makes a stable simulation difficult.

### Simulation setup

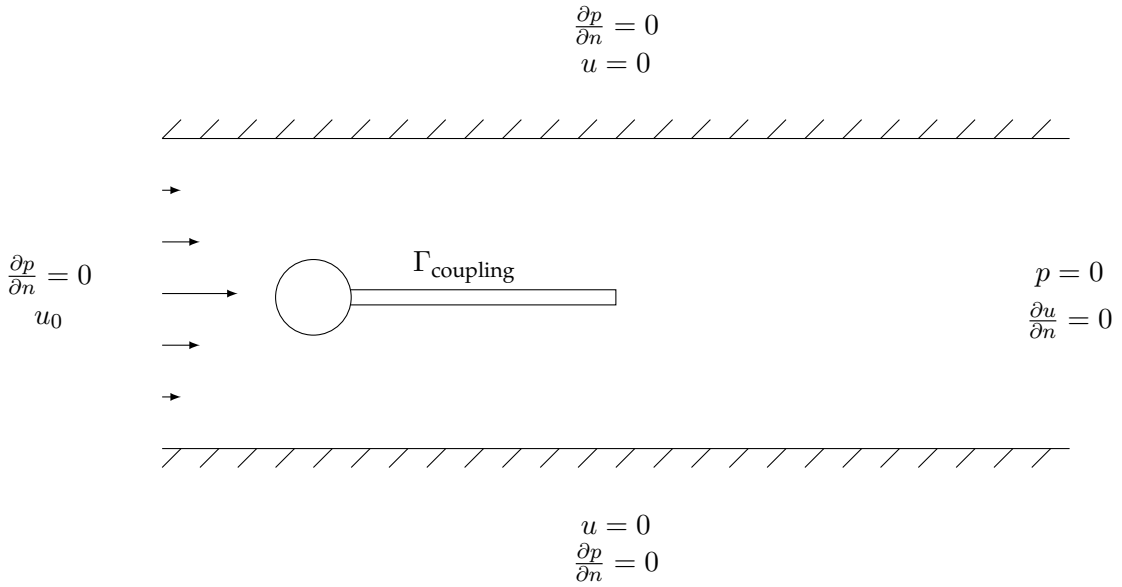



Figure 7.8.: Simulation setup of the fluid-structure interaction three benchmark

The test setup consists of laminar flow, which enters a channel with a length of $L = 2.5\,m$ and a height of $H = 0.41\,m$. Inside the channel an obstacle is installed, consisting of a cylinder with radius $r = 0.05\,m$ with a flexible flap attached to it. The cylinder is excluded from the structural simulation and only acts as fixed body inside the fluid domain. The flap geometry is given with $l = 0.35\,m$ and $h = 0.02\,m$ and interacts with the surrounding flow. The boundary conditions and the general setup are shown in figure 7.8. Similar to the first test case, the top and bottom wall of the channel is modeled as no-slip condition, with the velocity and pressure gradient set to 0. The right side consists of an outflow with pressure and velocity gradient set to 0. On the left side of the channel, the inflow is specified with

the pressure gradient set to 0 and an parabolic reference inflow velocity given by:

$$u_0(0, y) = 1.5 u_{ref} \frac{y(H - y)}{(0.5H)^2}. \tag{7.1}$$

The time dependent velocity is given by:

$$u_0(t, 0, y) = \begin{cases} u_0(0, y) \frac{1 - cos(\frac{\pi}{2} t)}{2}, & \text{if } t < 2.0 \\ u_0(0, y), & \text{otherwise} \end{cases} \tag{7.2}$$

where in the first two seconds of the simulation, the inflow velocity is set to a fixed value and afterwards the oscillation is started. The boundary of the flap with the fluid is set to the coupling boundary communicating with preCICE. The mesh movement at inflow and outflow is completely restricted, whereas a tangential movement at the walls is allowed. The flap boundary can move freely inside the fluid domain.

Similar to the validation cases before, $P_1$ Lagrangian, hexahedral elements are used for the structure and thus a structured mesh. For the fluid domain, a curvilinear grid is used to model the mesh around the cylinder appropriately. Around the obstacles the mesh is thus chosen to be very fine, getting coarser further away from it.

For the fluid solver, a second order backward method is chosen. The Runge-Kutta 5(4) is again used for the structural solver. It is assumed that an explicit coupling will induce numerical instabilities and therefore is omitted here. For the absolute measure a value of $10^{-7}$ is chosen and $5 \cdot 10^{-5}$ for the relative one. For the acceleration method a quasi Newton scheme is used. The physical parameters are listed in table 7.3.

| | | |
|---|---|---|
| fluid density | $\rho_F$ | 1000 $kg/m^3$ |
| dynamic viscosity | $\nu$ | 0.001 $kg/(ms)$ |
| structure density | $\rho_S$ | 1000 $kg/m^3$ |
| Young's modulus | $E$ | 5.6·$10^6$ $N/m^2$ |
| Poisson ratio | $\nu$ | 0.4 |
| coupling window | $\tau$ | 0.0075 s |
| simulation time | $t_{end}$ | 5 s |

Table 7.3.: Physical parameters of the fluid-structure interaction benchmark three scenario

### Results

The overall physical behavior of the test case is discussed first. The inflow velocity stays constant in the first $2s$ of the simulation, creating a stable channel flow. With the start of the inflow oscillation, the flow detaches from the cylinder and creates a vortex shedding. The intensity of the vortices depends on the inflow velocity magnitude. The created flow pattern is called Karman vortex street and is shown in figure 7.9 at $t = 3.5\ s$. Due to the pressure difference between the upper and lower surface of the elastic flap, the structure starts to bend and oscillate.

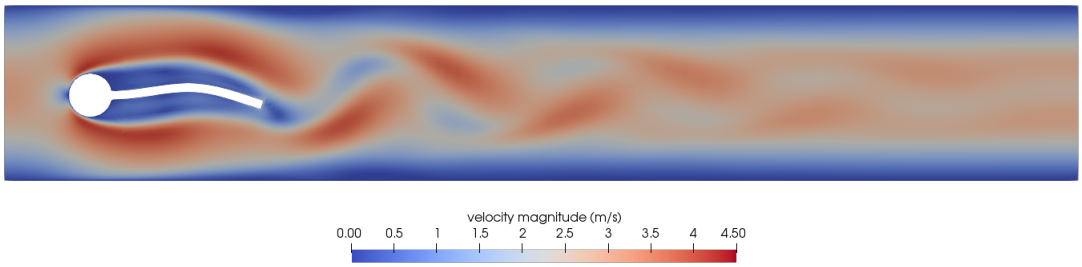



Figure 7.9.: Velocity field of the benchmark case and deformation of the mesh around the flap at t=3.5s

For a more detailed analysis of the problem, the vertical deformation of the flap tip is investigated. Figure 7.10 shows the respective displacement simulated with a dealII-OpenFOAM scenario as reference. The corresponding simulation with the dune-precice adapter and implicit-coupling is plotted as comparison. Due to the constant inflow in the first $2s$ no major deformation is occurring. Afterwards, the tip starts to oscillate irregularly. In the first $3.5\ s$ the solution obtained with the dune-precice adapter is similar to the reference. Afterwards displacement amplitude and oscillation period start to differ. In [Turek and Hron, 2007] it was shown that a stable oscillation can be obtained, which could not be reproduced here. The simulation procedure turned out to be very sensitive in terms of the coupling window size, convergence criteria and numerical schemes used.

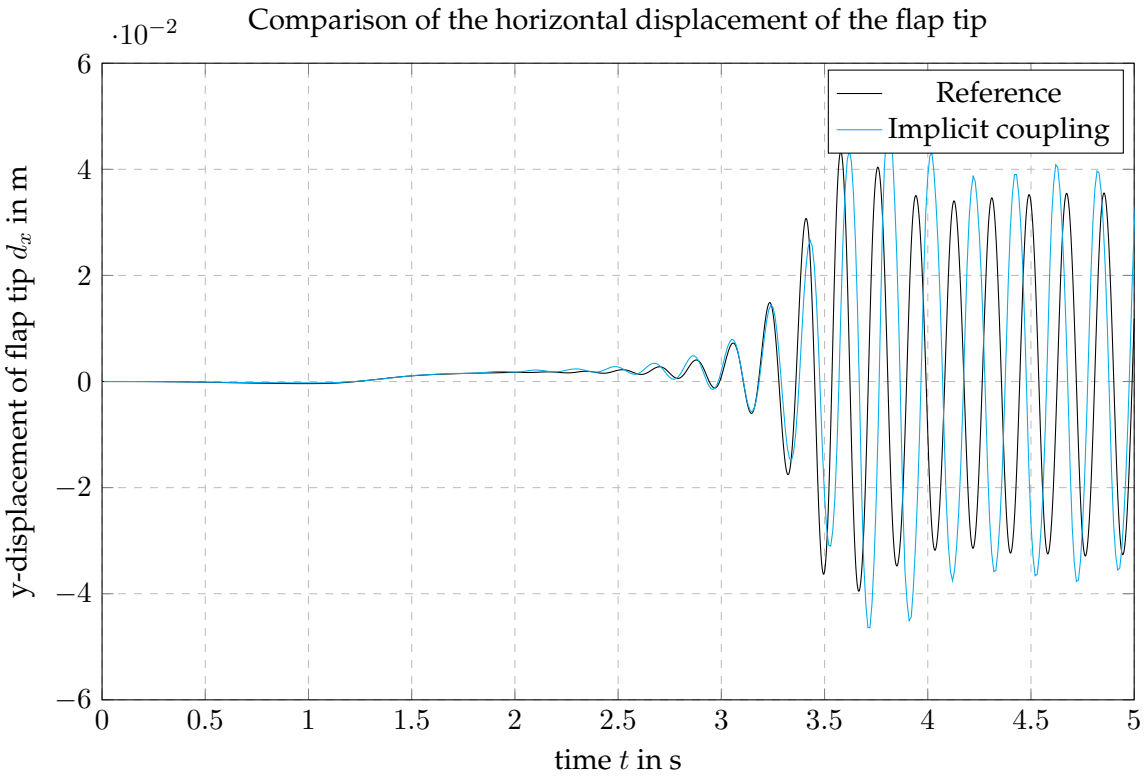



Figure 7.10.: Vertical displacement of the flap tip of an implicit coupling with the DUNE-adapter and a reference simulation with dealII-OpenFoam

# 8. Conclusion, Summary and Outlook

This chapter discusses the main challenges and results of the thesis regarding implementation and validation in section 8.1. Afterwards a short summary of each chapter is given in section 8.2. In a last step, the thesis is wrapped up by providing an outlook to future work in section 8.3.

## 8.1. Conclusion

The choice of the discretization of the governing equations with Lagrangian, continuum elements proved to be a good decision. Various literature regarding this topic is available and the functionality of DUNE enables a robust implementation. As could be seen from the validation cases, the static and dynamic simulations showed physically logic and accurate results. Initially first order elements were used, which did not yield the expected accuracy. Thus, a switch to higher order elements was done introducing difficulties regarding the mass lumping techniques. The computational intensity rapidly increased for three dimensional cases. A parallelization of the code would increase the efficiency, but was not realized in the scope of this thesis.

Considering the multiphysics setup, the coupling library preCICE offers a simple interface for partitioned coupling. To be able to handle the specific DUNE datatypes and procedures, the dune-precice module was introduced representing an adapter to the coupling library. The multiphysics-setup showed to be very sensitive in terms of coupling and material parameters chosen. Additionally, the simulations were in a first step only done with first order elements on the structural side to test the main functionality and response. The implementation of higher order elements still needs some further work. In the three valication cases the simulation showed good and fairly accurate results. Compared to a dealII-OpenFOAM reference simulation the obtained results differed slightly in the case of the flap in a channel scenario. For the fluid-structure interaction three benchmark, the solutions differed more heavily. Thus, some additional adjustments inside the dune-precice module and further work regarding it's implementation has to be done. Also, the linear elasticity assumption is violated due to the large deformations of the flexible structures.

The main goal of this thesis was to provide the basic features for the aeroelastic simulation of slender wings for electric aircraft, which can be considered as fulfilled. To be able to simulate a real three dimensional modelled wing, the efficiency of the implementation of the structural solver would have to be improved drastically as simulation times are currently very long, due to the amount of elements that are needed for an accurate

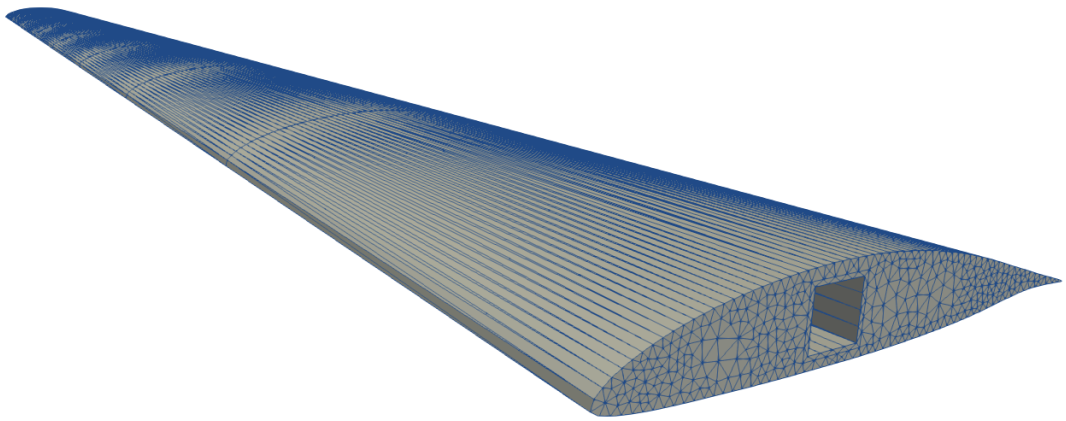


Figure 8.1.: Grid of a slender wing as might be used for electrically powered aircraft

representation of the wing (see figure 8.1). The coupling library offers the flexibility to not only connect traditional finite element and finite volume solvers, but can also provide an interface for implementation of multi-body dynamics and blade element momentum approaches to model small electric propellers along a wing.

## 8.2. Summary

To be able to simulate aeroelastic phenomena and thus fluid-structure interaction scenarios, this thesis introduced a solver for the linear elastodynamics equations implemented in DUNE. Furthermore, the coupling library preCICE and the fluid solver OpenFOAM were used for the multiphysics setup. Therefore, the prototype of the dune-precice module was presented, implementing a new adapter for the coupling interface and new module inside the DUNE environment.

In the first part of the thesis several approximation techniques for the structural and fluid domain were evaluated, which are commonly used for aeroelastic simulations. A partitioned coupling scheme was chosen were the numerical solvers are considered independently and afterwards connected for a multiphysics setup. The discretization of the equations of continuum mechanics for both participants was selected to obtain a good accuracy and high flexibility. For the linear equations of elastodynamics representing the structural part, finite elements were used as discretization scheme, whereas the Navier-Stokes equations were solved with a finite volume method provided by the black box solver OpenFOAM.

Considering the implementation of the structural part, the numerics library DUNE was used to handle the discretization with finite elements. The main steps of the program like global and local assemblers as well as the numerical schemes were shown. Additionally,

implicit and explicit time stepping methods were tested to solve the occurring system of ordinary differential equations. The structural solver was afterwards validated with a static beam bending test case and a dynamic simulation case consisting of a oscillating plate.

In a next step, the preCICE coupling library was introduced for a partitioned, multi-physics coupling. The mandatory steps considering data mapping, coupling methods and time interpolation were discussed. Afterwards, the dune-precice module and the respective implementation and usage was shown. Finally, three different validation cases of different numerical difficulty consisting of the flap in a channel, lid-driven cavity with flexible bottom and the fluid-structure interaction benchmark three were simulated.

## 8.3. Outlook

Further improvements can be made regarding the structural solver, the coupling approach and the code organization to be able to simulate large scale models with an acceptable accuracy:

- **Structural solver:** A parallel implementation would increase efficiency and thus lower the simulation time significantly. Additionally, the linear elasticity assumption could be extended to allow large deformations and nonlinear material models. This would help to predict the deflections occurring in the fluid-structure interaction scenarios more accurately.
- **Coupling approach:** The reduction of the temporal accuracy order by using subcycling is one of the drawbacks of the partitioned coupling approach. The topic is still under development by considering time interpolation inside preCICE.
- **Code organization:** The dune-precice module is a prototype only considering fluid-structure interaction scenarios, which was sufficient for this thesis. More complex scenarios with several participants still needs to be implemented. Additionally, the code implemented for the structural solver needs to be refined regarding the assemblers and time stepping schemes to provide the necessary tools to construct solvers for linear elastodynamics inside a dune-elastodynamics module. This includes more options for mass lumping and time-stepping methods, which can make use of the DUNE data structures.

# Appendix

## A.1. Gmsh-reader and boundary segments in DUNE

As described in section 3.1 unique integer values (called boundary tags) are used to identify different parts of the boundary. Those tags can be specified for different parts of the grid inside the mesh file, which are later used to specify the respective boundary condition. DUNE offers an interface to read such files and stores the corresponding integer values inside a vector. This procedure works fine for triangle and quadrilateral elements in two dimensional cases, where the boundary segments are edges. For tetrahedral elements the respective geometric entity are triangular faces. Problems occured for hexahedral elements, where the tags were set randomly inside DUNE despite the correct mesh setup. Therefore only tetrahedral elements were used in the first half of this thesis to circumvent this behavior.

With the help of Carsten Gräser and Oliver Sander from the DUNE developer team the error was found inside the gmsh-reader file. A missing method for quadrilateral boundary elements was triggering the random behavior, which was quickly fixed. The corresponding issue in GitLab can be found here describing the problem and solution in more detail: https://gitlab.dune-project.org/core/dune-grid/-/issues/115.

# List of Figures

# List of Tables

# Bibliography

[Bastian et al., 2008] Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Kornhuber, R., Ohlberger, M., and Sander, O. (2008). A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE. *Computing*, 82(2–3):121–138.

[Bathe and Zhang, 2009] Bathe, K.-J. and Zhang, H. (2009). A mesh adaptivity procedure for cfd and fluid-structure interactions. *Computers and Structures*, 87:604–617.

[Bettis, 1973] Bettis, D. G. (1973). A runge-kutta nyström algorithm. *Celestial mechanics*, 8:229–233.

[Birken et al., 2010] Birken, P., Quint, K. J., Hartmann, S., and Meister, A. (2010). A time-adaptive fluid-structure interaction method for thermal coupling. *Computing and Visualization in Science*, 13:331–340.

[Blatt and Bastian, 2007] Blatt, M. and Bastian, P. (2007). The iterative solver template library. In Kagström, B., Elmroth, E., Dongarra, J., and Wasniewski, J., editors, *Applied Parallel Computing – State of the Art in Scientific Computing*, pages 666–675, Berlin/Heidelberg. Springer.

[Blatt et al., 2016] Blatt, M., Burchardt, A., Dedner, A., Engwer, C., Fahlke, J., Flemisch, B., Gersbacher, C., Gräser, C., Gruber, F., Grüninger, C., Kempf, D., Klöfkorn, R., Malkmus, T., Mthing, S., Nolte, M., Piatkowski, M., and Sander, O. (2016). The Distributed and Unified Numerics Environment, Version 2.4. *Archive of Numerical Software*, 4(100):13–29.

[Blom et al., 2015] Blom, D. S., Uekermann, B., Mehl, M., van Zuijlen, A. H., and Bijl, H. (2015). Multi-level acceleration of parallel coupled partitioned fluid-structure interaction with manifold mapping. *Lecture notes in computational science and engineering*, 105:135–150.

[Breuer et al., 2012] Breuer, M., De Nayer, G., Münsch, T., Gallinger, T., and Wüchner, R. (2012). Fluid-structure interaction using a partitioned semi-implicit predictor-corrector coupling scheme for the application of large-eddy simulation. *Journal of Fluids and Structures*, 29:107–130.

[Bungartz et al., 2010] Bungartz, H.-J., Benk, J., Gatzhammer, B., Mehl, M., and Neckel, T. (2010). Partitioned simulation of fluid-structureinteraction on cartesian grids. *Journal of Fluids and Structures*, 29:255–284.

[Bungartz et al., 2016] Bungartz, H.-J., Lindner, F., Gatzhammer, B., Mehl, M., Scheufele, K., Shukaev, A., and Uekermann, B. (2016). preCICE – a fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250–258. Advances in Fluid-Structure Interaction.

[Butler et al., 2020] Butler, R., Dodwell, T., Reinarz, A., Sandhu, A., Scheichl, R., and Seelinger, L. (2020). High-performance dune modules for solving large-scale, strongly anisotropic elliptic problems with applications to aerospace composites. *Computer Physics Communications*, 249:106997.

[Causin et al., 2005] Causin, P., Gerbeau, J.-F., and Nobile, F. (2005). Added-mass effect in the design of partitioned algorithms for fluid-structure problems. *Computer Methods in Applied Mechanics and Engineering*, 194:4506–4527.

[Chourdakis, 2017] Chourdakis, G. (2017). A general openfoam adapter for the coupling library precice. Master's thesis, Technische Universität München.

[de Boer et al., 2007] de Boer, A., van Zuijlen, A. H., and Bijl, H. (2007). Review of coupling methods for non-matching meshes. *Computer methods in applied mechanics and engineering*, 196:1515–1525.

[Dinkler, 2017] Dinkler, D. (2017). *Einfhrung in die Strukturdynamik - Modelle und Anwendungen - 2.Auflage*. Springer Vieweg.

[DLR, 2019] DLR (2019). Nineteen-seater aircraft with distributed, electrically driven propellers. "https://www.dlr.de/content/en/articles/news/2019/04/20191126_e-aircraft-novel-configurations-open-up-new-possibilities.html".

[Dormand and Prince, 1980] Dormand, J. R. and Prince, P. J. (1980). A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26.

[Felippa et al., 2015] Felippa, C. A., Guo, Q., and Park, K. C. (2015). Mass matrix templates: General description and 1d examples. *Archives of Computational Methods in Engineering*, 22:1–65.

[Fried and Malkus, 1975] Fried, I. and Malkus, D. S. (1975). Finite element mass matrix lumping by numerical integration with no convergence rate loss. *International Journal of Solids and Structures*, 11:461–466.

[Gatzhammer, 2014] Gatzhammer, B. (2014). *Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions*. PhD thesis, Technische Universitt München.

[Hairer et al., 2008] Hairer, E., Norsett, S. P., and Wanner, G. (2008). *Solving Ordinary Differential Equations I - Nonstiff Problems*. Springer.

[Heinz et al., 2016] Heinz, J. C., Sorensen, N. N., and Zahle, F. (2016). Fluid-structure interaction computations for geometrically resolved rotor simulations using cfd. *Wind Energy*, 19:2205–221.

[Hertrich, 2019] Hertrich, R. (2019). Partitioned fluid structure interaction: Coupling fenics and openfoam via precice.

[Hinton et al., 1976] Hinton, E., Rock, T., and Zienkiewicz, O. C. (1976). A note on mass lumping and related processes in the finite element method. *Earthquake engineering and structural dynamics*, 4:245–249.

[Huang, 2011] Huang, Z. S. (2011). Review of aeroelasticity for wind turbine: Current status, research focus and future perspectives. *Frontiers in Energy Research*, 5:419–434.

[Irons and Tuck, 1969] Irons, B. M. and Tuck, R. C. (1969). A version of the aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1:175–277.

[Landajuela et al., 2017] Landajuela, M., Vidrascu, M., Chapelle, D., and Angel, F. M. (2017). Coupling schemes for the fsi forward prediction challenge: comparative study and validation. *International Journal for Numerical Methods in Biomedical Engineering*, 33.

[Lunk and Simeon, 2005] Lunk, C. and Simeon, B. (2005). Runge-kutta-nyström methods with maximized stability domain in structural dynamics. *Applied Numerical Mathematics*, 53:373–389.

[MacNeal and Harder, 1985] MacNeal, R. H. and Harder, R. L. (1985). A proposed standard set of problems to test finite element accuracy. *Finite Elements in Analysis and Design 1*, 1:3–20.

[Manwell et al., 2009] Manwell, J. F., McGowan, J. G., and Rogers, A. L. (2009). *Wind Energy Explained - Theory, Design and Application*. 2. John Wiley Sons Ltd.

[MathWorks, ] MathWorks, M. Dynamics of damped cantilever beam.

[Mehl et al., 2017] Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., and van Zuijlen, A. (2017). Parallel coupling numerics for partitioned fluid-structure interaction simulations. *SIAM Journal on Scientific Computing*, 39:404–433.

[Merkel and Oechsner, 2020] Merkel, M. and Oechsner, A. (2020). *Eindimensionale Finite Elemente - Ein Einstieg in die Methode*. Springer Vieweg.

[Moriarty and Hansen, 2005] Moriarty, P. J. and Hansen, A. C. (2005). *AeroDyn Theroy Manual*. National Renewable Energy Laboratory, 1617 Cole Boulevard, Golden, Colorado.

[Newmark, 1959] Newmark, N. M. (1959). A method of computation for structural dynamics. *Journal of the engineering mechanics division*, 85:67–94.

[Rendall and Allen, 2008] Rendall, T. C. S. and Allen, C. B. (2008). Unified fluid-structure interpolation and mesh motion using radial basis functions. *International journal for numerical methods in engineering*, 74:1519–1559.

[Rueth et al., ] Rueth, B., Uekermann, B., Mehl, M., Birken, P., Monge, A., and Bungartz, H.-J. Quasi-newton waveform iteration for partitioned fluid-structure interaction.

[Rusch, 2016] Rusch, A. (2016). Extending su to fluid-structure interaction via precice.

[Sander, 2020] Sander, O. (2020). *DUNE The Distributed and Unified Numerics Environment*. 1. Springer International Publishing.

[Scheufele and Mehl, 2016] Scheufele, K. and Mehl, M. (2016). Robust multisecant quasi-newton variants for parallel fluid-structure simulations - and other multiphysics applications. *Computers and Mathematics with Applications*, 71:869–891.

[Subbaraj and Dokainish, 1989] Subbaraj, K. and Dokainish, M. A. (1989). A survey of direct time-integration methods in computational structural dynamics-ii. implicit methods. *Computers Structures*, 32:1387–1401.

[Turek and Hron, 2007] Turek, S. and Hron, J. (2007). *Proposal for Numerical Benchmarking of FluidStructure Interaction Between an Elastic Object and Laminar Incompressible Flow*, volume 53, pages 371–385.

[Uekermann, 2016] Uekermann, B. W. (2016). *Partitioned Fluid-Structure Interaction on Massively Parallel Systems*. PhD thesis, Technische Universität Mänchen.

[van Brummelen, 2010] van Brummelen, E. H. (2010). Partitioned iterative solution methods for fluid-structure interaction. *International Journal For Numerical Methods In Fluids*, 65:3–37.

[Vzquez, 2007] Vzquez, J. G. V. (2007). *Nonlinear Analysis of Orthotropic Membrane and Shell Structures Including Fluid-Structure Interaction*. PhD thesis, Universitat Politcnica de Catalunya.

[Wall, 1999] Wall, W. A. (1999). *Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen*. PhD thesis, Universität Stuttgart, Pfaffenwaldring 7, D-70550 Stuttgart.

[Wang et al., 2016] Wang, L., Liu, X., and Kolios, A. (2016). State of the art in the aeroelasticity of wind turbine blades: Aeroelastic modelling. *Renewable and Sustainable Energy Reviews*, 64:195–210.

[Wilson et al., 1973] Wilson, E. L., Farhoomand, I., and Bathe, K. J. (1973). Nonlinear dynamic analysis of complex structures. *Earthquake engineering and structural dynamics*, 1:241–252.

[Yau, 2016] Yau, L. C. (2016). Conjugate heat transfer with the multiphysics coupling library precice. Master's thesis, Technische Universität München.