Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science – Aerospace Engineering

Fluid-Structure Interaction based on co-simulation with Multibody Dynamics

Supervisor
Prof. Pierangelo MASARATI

Co-Supervisor
Prof. Marco MORANDINI

Candidate
Claudio Giovanni CACCIA – 820091

Academic Year 2019 – 2020

Acknowledgements

First, I would like to thank my supervisors, prof. Pierangelo Masarati and prof. Marco Morandini for giving me the opportunity to work on this fascinating topic and letting me do it at my own pace. I had the chance to attend to their courses and that inspired me to improve my knowledge. I would also thank them for their support and suggestions, even if these difficult times made everything a little more complicated.

My special thanks go to the preCICE community, to Benjamin Uekermann, Gerasimos "Makis" Chourdakis and many others who were happy to exchange ideas and shared their knowledge and experience, helping me better understand the most difficult aspects of this subject.

Finally, but not least importantly, I owe my gratitude to Eleonora Colombo, who let me endlessly talk about my doubts and difficulties. She helped me put everything in perspective and gave me the motivation to pursue my goal, independently from everything else. Without her support this would not have been possible.

Abstract

The computer simulation of Fluid-Structure Interaction (FSI) phenomena allows to gain more insight on complex interactions and behaviors of solids immersed in fluid flows, helping predict their effects. Applications range from aeroelasticity, to turbomachinery, or biomechanics, just to name a few.

It is possible to perform those simulations in different ways: one of them involves a technique known as *partitioned algorithm*. A partitioned algorithm aims at solving a FSI problem basically by means of three elements, which include a fluid solver, a structural solver and a third component which performs the interaction between the other two. The advantage of this technique consists in reusing and adapting already developed and optimized solvers and connect them.

In this thesis, the MultiBody Dynamics analysis software (MBDyn) has been linked to the multiphysics coupling library precise Code Interaction Coupling Environment (preCICE), with the purpose of extending MBDyn capabilities in the field of FSI simulations.

For this reason, an *adapter*(i.e. a piece of connecting code, in this case written in C++), has been developed to implement this coupling.

Coupling MBDyn with preCICE represents and advantage and an extension of capabilities, because many other adapters for the fluid side have already been developed for this library. It is then possible and simple to choose among a considerable number of fluid solvers, including many well-validated open source and commercial codes. On the other hand, with a fully integrated *MBDyn adapter*, the library preCICE gains the opportunity to connect to a multibody dynamics software, which has not yet been completely developed.

The coupling between MBDyn and preCICE has been successfully tested in different scenarios, including some well-known FSI benchmark problems. Also some current limitations of applicability, emerged in one of those benchmarks, have been analyzed.

The current status of the adapter represents a good starting point to explore more in detail the behavior of the MBDyn-preCICE coupling even in more complex scenarios and to use it in real-world applications.

Keywords— fluid structure interaction, partitioned algorithms, multibody dynamics, MBDyn, preCICE

Sommario

La simulazione computerizzata di fenomeni di Fluid-Structure Interaction (FSI) consente di ottenere una maggiore comprensione di interazioni e comportamenti complessi di corpi solidi immersi in un fluido, aiutando a prevederne gli effetti. Le applicazioni si estendono dall'aeroelasticità, alle turbomacchine od alla biomeccanica, solo per citarne alcune.

È possibile eseguire tali simulazioni in modi differenti: uno di questi utilizza una tecnica nota come *algoritmo partizionato*. Un algoritmo partizionato tenta di risolvere un problema di FSI utilizzando tre elementi: un solutore fluido, un solutore solido ed un terzo componente che si occupa dell'interazione tra gli altri due. Il vantaggio di questa tecnica consiste nel poter riutilizzare ed adattare elementi codice già sviluppato ed ottimizzato e di connetterli.

In questa tesi, il MultiBody Dynamics analysis software (MBDyn) è stato collegato alla libreria software di simulazione multifisica precise Code Interaction Coupling Environment (preCICE), con l'obiettivo di estendere, per MBDyn, le possibilità di simulazione nell'ambito della simulazione fluido-struttura.

Per questa ragione, un *adattatore* (ovvero del codice software di connessione, in questo caso scritto in C++) è stato sviluppato per realizzare questa operazione.

La connessione di MBDyn con preCICE costituisce un vantaggio ed una estensione di potenzialità, in quanto sono già presenti molti adattatori per preCICE in ambito fluidodinamico: diventa così possibile e semplice scegliere tra un considerevole numero di solutori fluidi, tra cui molti codici *open-source* e commerciali molto noti. D'altra parte, con un *adattatore MBDyn* completamente integrato, la libreria preCICE ottiene la possibilità di connettere un solutore multiboby, un aspetto ad oggi non ancora completamente sviluppato.

L'interazione tra MBDyn e preCICE è stata sperimentata con successo in scenari diversi, tra cui una serie di problemi di riferimento in ambito FSI ben noti in letteratura. Anche alcune attuali limitazioni d'uso, emerse durante lo studio di uno di questi problemi, sono state analizzate.

Lo stato attuale di conoscenza e sviluppo dell'*adattatore* rappresenta un buon punto di partenza per analizzare più in dettaglio il comportamento dell'interazione tra MBDyn e preCICE anche in scenari più complessi e di utilizzarlo come strumento di analisi in applicazioni reali.

Parole chiave— interazione fluido struttura, algoritmi partizionati, dinamica multibody, MBDyn, preCICE

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Fluid-Structure Interaction (FSI) describes the mutual interaction between a moving or deformable object and a fluid in contact with it, surrounding or internal. It is present in various forms both in nature and in man-made systems: a leaf fluttering in the wind, water flowing underground or blood pumping in an artery are typical examples of fluid-structure interaction in nature. FSI occurs in engineered systems when modeling the behavior of turbomachinery, the flight characteristics of an aircraft, or the interaction of a building with the wind, just to name a few examples.

All the aforementioned problems go under the same category of FSI, even if the nature of the interaction between the solid and fluid is different. Specifically, the intensity of the exchanged quantities and the effect in the fluid and solid domains vary among different problems.

There can be multiple ways to classify FSI problems, based on the flow physics and on the behavior of the body. Incompressible flow assumption is always made for liquid-solid interaction, while both compressible and incompressible flow assumptions are made when a gas interacts with a solid, depending on the flow properties and the kind of simulation. The main application of air-solid interaction considers the determination of aerodynamic forces on structures such as aircraft wings, which is often referred to as *aeroelasticity*. Dynamic aeroelasticity is the topic that normally investigates the interaction between aerodynamic, elastic and inertial forces. Aerodynamic *flutter* (i.e. the dynamic instability of an elastic structure in a fluid flow) is one of the severe consequences of aerodynamic forces. It is responsible for destructive effects in structures and a significant example of FSI problems.

The subject may also be classified considering the behavior of the structure interacting with the fluid: a solid can be assumed rigid or deforming because of the fluid forces. Examples where rigid body assumption may be used include internal combustion engines, turbines, ships and offshore platforms. The rigid body–fluid interaction problem is simpler to some extent, nevertheless the dynamics of rigid body motion requires a solution that reflects the fluid forces. Within the deformable body–fluid interaction, the nature of the deforming body may vary from very simple linear elastic models in small strain to highly complex nonlinear deformations of inelastic materials. Examples of deforming body–fluid interaction include aeroelasticity, biomedical applications and poroelasticity.

The interaction between fluid (incompressible or compressible) and solid (rigid or deformable) can be *strong* or *weak*, depending on how much a change in one domain influences the other. An example of weakly coupled problem is aeroelasticity at high Reynolds number, while incompressible flow often leads to strongly coupled problems. This distinction can lead to different solution strategies, as briefly described below.

Physical models aren't the only way in which FSI problems can be classified. The solution

procedure employed plays a key role in building models and algorithms to solve this kind of problems. The two main approaches are: the *monolithic approach* in which both fluid and solid are treated as one single system and the *partitioned approach* in which fluid and solid are considered as two separated systems coupled only through an interface. This latter approach is often preferred when building new solution procedures as it allows to use solvers that have been already developed, tested and optimized for a specific domain. The solvers only need to be linked to a third component, which takes care of all the interaction aspects.

The partitioned approach can be further classified considering the coupling between the fluid and solid: the solution may be carried out using a *weakly coupled approach*, in which the two solvers advance without synchronization, or a *strongly coupled approach*, in which the solution for all the physics must be synchronized at every time step. Although the weakly coupled approach is used in some aerodynamic applications, it is seldom used in other areas due to instability issues. A strongly coupled approach is generally preferred, even though this leads to more complex coupling procedures at the interface between fluid and solid.

This work describes the implementation and the validation of what is called an *adapter*, that is the "glue-code" needed to interface a solver to a coupling software library, thus adopting a *partitioned approach* to solve FSI problems. The *adapter* presented here connects the software code *MultiBody Dynamics analysis software (MBDyn)* to the multi-physics coupling library *precise Code Interaction Coupling Environment (preCICE)*.

Interfacing MBDyn with preCICE has multiple advantages: on one side it extends the capabilities of MBDyn to be used in FSI simulations by connecting it with a software library which has been already connected to widely used CFD solvers; on the other side, it allows to describe and simulate FSI problems with a suite of lumped, 1D and 2D elements (i.e. rigid bodies, *beams*, *membranes*, *shells*, etc.) decoupling the shape of the object (i.e. the interface with the fluid) from its structural properties, which can be described by different models and constitutive laws.

The thesis is structured as follows:

- Section 2 introduces the reader to FSI problems and their complexity, with particular attention to the physical description of the fluid and solid domains and the interface.
- Section 3 focuses on numerical methods, describing how to computationally deal with these kind of problems: details regarding the different coupling approaches are given here.
- Section 4 explains the features of preCICE that the adapter needs to support and gives a short introduction to MBDyn, explaining the main functionalities of interest.
- Section 5 presents the adapter developed in this work, its most important features and how to configure a FSI simulation with it.
- Section 6 describes the validation of the adapter, the comparison of the results with some well-known benchmarks together with a comparison to a real world experiment.
- Section 7 summarizes the findings and outcomes of this work and gives an outlook to future development on this topic.
- Finally four appendices give further information concerning the structure and the configuration of the software components used in this work.

Chapter 2

Physical aspects of Fluid-Structure Interaction problems

Dynamic models of solids or fluids aim at describing the evolution of an initial configuration through time. Structural mechanics and fluid dynamics use different perspectives when describing the motion of respectively a solid or a fluid particle. When dealing with FSI problems, the two approaches need to be combined in order to obtain a suitable description of the two domains and their interface: this aspect is treated in 2.1.

As outlined in the introduction, the fluid and the solid domain of a FSI problem might be described by means of many different models: some of them are outlined in section 2.2. *Dimensional analysis* and the use of dimensionless numbers is a powerful tool used to classify fluid dynamics problems: some of the principles used there can be applied to FSI problems in order to classify them: this can help define and classify FSI problems, as described in section 2.3.

2.1 Description of motion

In a FSI model, the fluid in motion deforms the solid because of the forces exerted to the structure. The change in the shape of the solid modifies the fluid domain, causing a different flow behavior. For this reason it is necessary to describe formally the kinematics and the dynamics of the whole process. Classical continuum mechanics considers the motion of particles by means of two different perspectives [1]: the *Eulerian description*, briefly described in section 2.1.1, and the *Lagrangian description*, outlined in section 2.1.2. Those two perspectives are typically combined into the *arbitrary Lagrangian-Eulerian (ALE)* method, described in section 2.1.3.

2.1.1 Eulerian perspective

The *Eulerian perspective* observes the change of quantities of interest (e.g. density, velocity, pressure) at spatially fixed locations. In other words: the observer does not vary the point of view during different time steps. Thus, quantities can be expressed as functions of time at fixed locations. This is represented by the following notation:

$$\Theta = \tilde{\Theta}(x, y, z, t) \tag{2.1}$$

where Θ is a quantity of interest and $\tilde{\Theta}$ denotes the same quantity form an Eulerian point of view; (x, y, z) represent a fixed location in the euclidean space.

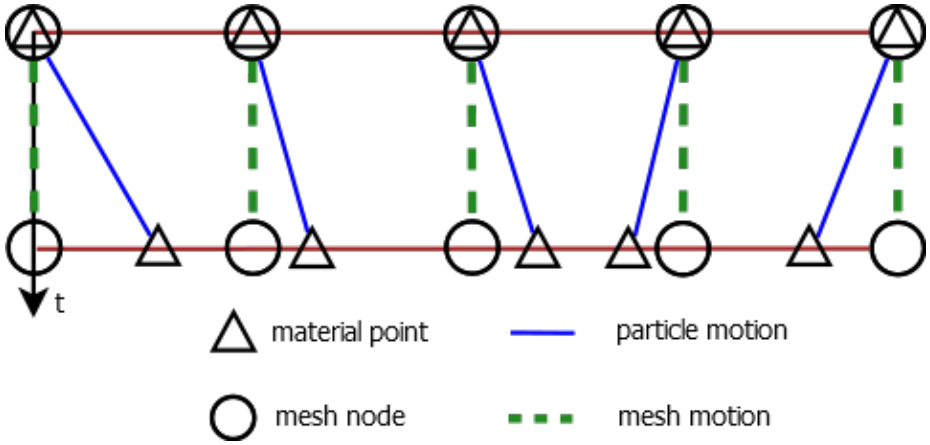


Figure 2.1. Eulerian perspective

A computational mesh can be interpreted as a number of observers distributed across the domain of interest and connected to each other in order to form a a grid with nodes. If the particles of the domain move, a purely euclidean mesh does not move and the position of the nodes remains fixed at any instance of time [8]. This behavior is represented in Figure 2.1 (adapted from [8]). The mesh is independent of particles movement, resulting in a convenient choice for Computational Fluid Dynamics (CFD) problems, where fluid flows throughout the whole computational domain. Within this approach, proper mesh refinement is crucial for computational accuracy as it defines to what extent small scale movement can be modeled and resolved [17].

2.1.2 Lagrangian perspective

A *Lagrangian observer* focuses on a single particle and follows it throughout the motion, as depicted in Figure 2.2. Changes in the quantities of interest are observed at different spatial locations.

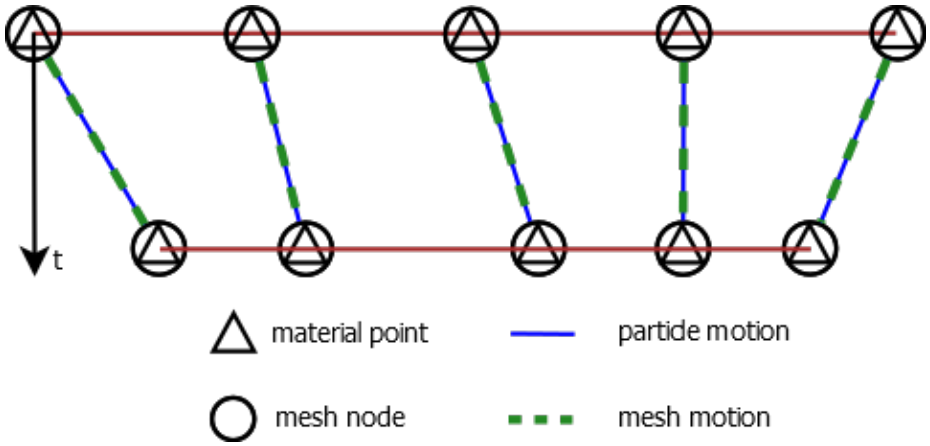


Figure 2.2. Lagrangian perspective

The motion of the particle and the other quantities of interest can be described by reference coordinates (or *material coordinates*) in Euclidean space (X, Y, Z), uniquely identifying the observed particle at a reference configuration [75]. Usually $t = 0$ is chosen as reference time

but this is not mandatory. The Lagrangian observer only registers changes concerning one specific particle as time advances. Thus, quantities of interest can be described as:

$$\Theta = \hat{\Theta}(X, Y, Z, t) \tag{2.2}$$

In contrast to the Eulerian perspective (Equation 2.1), the obtained information is strictly limited to a single material particle (implied by the usage of the capital reference coordinate variables). Information about a fixed point in space is not directly available and no convective fluxes appear in a Lagrangian description.

This perspective is again translated into computational meshes: at a reference instance of time, mesh nodes are attached to material particles. As these move, the mesh nodes move with them causing the mesh to deform. Figure 2.2 describes the situation. The mesh nodes always coincide with their respective particles.

In this situation, large-scale and irregular motions (and more importantly deformations) lead to distortions of the computational mesh, which yields smaller accuracy in simulations. This requires the application of techniques to keep the desired accuracy [48].

Lagrangian perspective is the usual method of choice for Computational Solid Mechanics (CSM) simulations.

Eulerian and Lagrangian descriptions are related [2]. A mapping between them can described by the *motion* function ϕ such that:

$$\vec{x}(t) = \phi(\vec{X}, t) \tag{2.3}$$

Equation 2.3 tells that the Eulerian, spatial position $\vec{x}$ of a particle at time t is the mapping of the particle at its reference configuration $\vec{X}$: the mapping must be bijective.

2.1.3 ALE method

As outlined above, CSM and CFD problems adopt different perspectives. The arbitrary Lagrangian-Eulerian (ALE) approach, a combination of the two points of view, is used for FSI problems. As the name implies, an ALE observer moves arbitrarily with respect to a specific material particle. Figure 2.3 depicts such a situation.

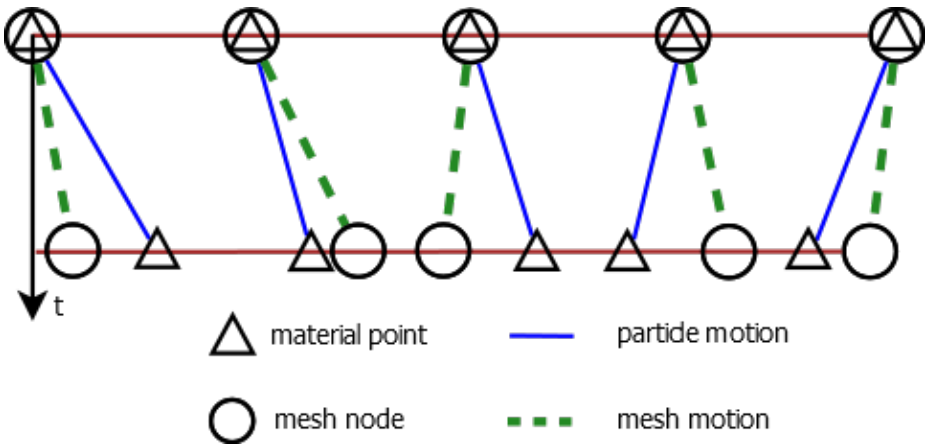


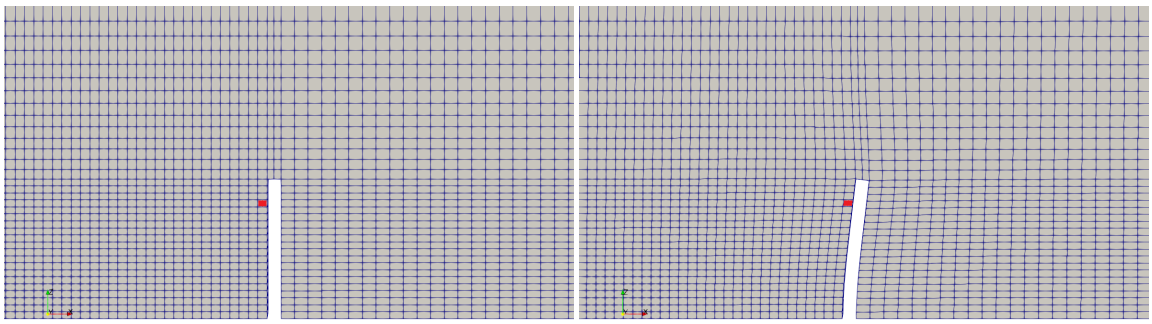
Figure 2.3. ALE perspective

When dealing with computational meshes, an ALE mesh is considered as it can move almost arbitrarily with respect to the motion of the underlying particles, as shown in Figure 2.4. The only constraint is that node movements should not distort the mesh too much

as this leads to computational inaccuracy. Many algorithms exist to implement suitable quality criteria and keep the mesh motion reasonable and to allow the nodes to follow moving particles up to a certain extent [10].

Since the mesh motion and material particle motion are not directly linked, a new unknown is introduced: the relative movement between the ALE mesh and the material domain. This approach is particularly useful in FSI problems: fluid and solid must follow the moving interface between them for physical reasons.

Since the solid domain is usually described in a Lagrangian perspective, the solid mesh is kept attached to the FSI interface. However, also the fluid domain must deform to avoid formation of gaps between the meshes. Therefore, in ALE methods the fluid mesh nodes at the interface move with it. Fluid mesh nodes follow the fluid particles sticking to the interface (for viscous flows), while the rest of the fluid mesh is allowed to move in such way that mesh distortions are kept minimal, to preserve computational accuracy [60].



(a) undistorted mesh (b) distorted mesh

Figure 2.4. ALE mesh

2.2 Domains and interface

Fluid-Structure Interaction implies that the overall model is determined by models defining the fluid behavior and the solid behavior, briefly described in sections 2.2.1 and 2.2.2. A short overview of beam models is given in section 2.2.3 as it is relevant for the model developed in this work. Finally a formal definition of the interface is given in section 2.2.4, as it is necessary to impose suitable coupling conditions at the common boundary of the solid and the fluid.

2.2.1 Fluid domain

An exhaustive description of all possible fluid models is far beyond the scope of this work. A quite general model is the viscous compressible one described by the Navier Stokes Equations (NSE).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{2.4a}$$

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v}) + \nabla p - \nabla \cdot \tau - \rho \vec{g} = 0 \tag{2.4b}$$

$$\frac{\partial}{\partial t}(\rho e_0) + \nabla \cdot (\rho e_0 \vec{v}) + \nabla \cdot (\vec{v} p + \vec{q} - \vec{v} \cdot \boldsymbol{\tau}) - \vec{v} \cdot \rho \vec{g} = 0 \tag{2.4c}$$

where:

- ρ denotes density
- $\vec{v}$ is flow velocity in all dimensions
- p denotes pressure
- $\boldsymbol{\tau}$ is the viscous stress tensor
- $\vec{g}$ represents the sum of all body forces
- e_0 is the total energy per unit mass
- $\vec{q}$ is the heat flux by conduction

They consist in the mass conservation equation (2.4a), the conservation of momentum equation (2.4b) and the energy conservation equation (2.4c). For a Newtonian fluid, the viscous stress tensor is given by:

$$\boldsymbol{\tau} = -\frac{2}{3}\mu\left(\nabla \cdot \vec{v}\right) I + 2\mu S \tag{2.5}$$

with μ being the dynamic viscosity and S the rate of deformation tensor (i.e. the symmetric part of the velocity gradient $\nabla\vec{v}$):

$$\mathbf{S} = \frac{1}{2}\left(\nabla\vec{v} + \nabla\vec{v}^T\right) \tag{2.6}$$

A detailed derivation of such equations and the theory beyond can be found for example in [59] and [58] or in [56].

The set of equations above, even with a Newtonian fluid model, lack some other information in order to form a closed set of Partial Differential Equations (PDE). A conductive heat flux model is needed (e.g. Fourier's Law), the caloric and thermodynamic equations of state have to be chosen, a proper turbulence model (if needed, see [56]) and finally, the appropriate initial and boundary conditions for the problem [22] must be defined.

Simplifications can be done to obtain less sophisticated flow models such as: adiabatic, inviscid, incompressible, and many others. Dimensional Analysis is a powerful tool to determine to what extent some reduced models are meaningful, and it is widely used in fluid dynamics, as described in section 2.3.1. Most CFD software codes allow to set up simulations with the most suitable model which can be coupled with a solid model to build a FSI problem. Some further details are given in section 3.1.

2.2.2 Solid domain

In solid mechanics, particles do not travel as much as they do in fluid dynamic problems, as described in 2.1.2. For this reason, a Lagrangian perspective is generally used.

The de-Saint Venant-Kirchhoff model [54] is very commonly used when describing the movement of a solid: it is also often used in FSI problems ad it is capable of handling large deformations. The material is considered:

- *homogeneous*: the material properties do not depend on the position of the particle
- *linear elastic*: the stress-strain relationship is linear
- *isotropic*: the stress-strain relationship is independent from the direction of the load

A general expression of the dynamic equation can be derived from the Virtual Work Principle (VWP) applied to an arbitrary control volume:

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \mathbf{T} + \rho \vec{f} \tag{2.7}$$

In equation 2.7:

- ρ: is the material density
- $\vec{u}$: is the particle displacement
- $\mathbf{T}$: is the *second Piola-Kirchhoff* stress tensor
- $\vec{f}$: is the sum of body forces

In order to close the dynamic equation, a constitutive law which must be considered to relate stress and strain:

$$\mathbf{T} = \lambda \mathbf{I} \mathrm{tr}\left[\boldsymbol{\varepsilon_G}\right] + 2\mu \boldsymbol{\varepsilon_G} \tag{2.8}$$

where $\boldsymbol{\varepsilon_G}$ is the Green-Lagrange strain tensor:

$$\boldsymbol{\varepsilon_G} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right) \tag{2.9}$$

and $\mathbf{F}$ is the deformation gradient. λ and μ are material properties and are named Lamé constants. These relate to the Young modulus E and the Poisson ratio ν which are more commonly used in practice. The relationship among the various parameters is the following:

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu} \tag{2.10}$$

$$\nu = \frac{\lambda}{2(\lambda + \mu)} \tag{2.11}$$

The set of parameters (E, ν) or (λ, μ), together with the density ρ fully define the material, under the assumptions of linear elasticity, isotropy and homogeneity.

The set of PDEs is completed when suitable initial and boundary conditions are defined.

2.2.3 Models with reduced dimensionality: beams

The equations introduced in section 2.2.2 may be a tough task to solve even in case of isotropic hyperelasticity, when considering a 3-D domain. Even with today's computers and using finite elements techniques, it is not always feasible or convenient to treat a solid as a three-dimensional continuum. Bodies with particular geometric features can be seen as lower dimension ones, with respect to the governing equations [36]. Such bodies are called *beams* (one dimension), *plates* or *shells* (two dimensions).

The *beam* model splits the description of the geometry into two sub-problems:

1. a beam is defined by its *reference line* and the movement (displacement and rotation) of the solid is completely defined by it (see Figure 2.5),

2. the beam *cross section* is considered as a whole, its movement depends on the movement of the reference line, stresses are generalized into *resultants* (axial, bending, shear, torsional) which represent the aggregate effect of all of the stresses acting on the cross section. The constitutive properties of the section (axial, shear, torsion and bending stiffness) allow to relate stresses and deformations (by means of VWP) and close the problem.

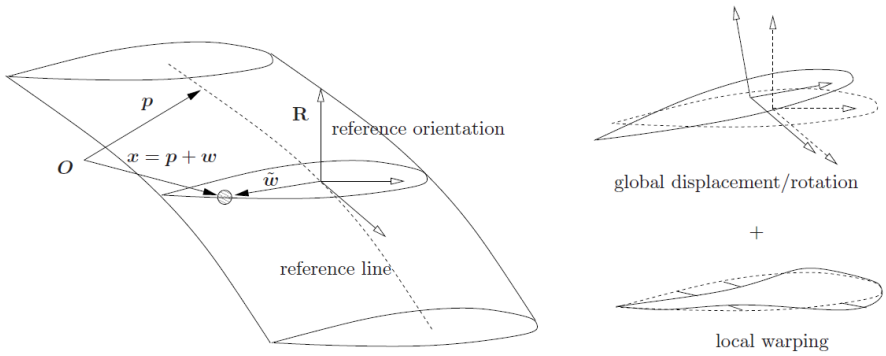


Figure 2.5. beam model (image taken from [26])

The beam model can be used to build elements of a Finite Element Method (FEM). For example, the beam element can be modeled by means of a Finite Volume approach, as described in [25], which computes the internal forces as functions of the straining of the reference line and orientation at selected points along the line itself, called evaluation points.

This approach is particularly interesting for FSI problems in which slender structures are involved. A mapping is needed between the fluid-solid interface and the reference line movement, which will be described in Sections 4.1.4 and 4.1.7.

2.2.4 Interface and interaction

Since FSI problems are centered on the interaction of the fluid and solid domain, their common interface needs to be described properly. A simple representation of the situation at the so called *wet surface* is shown in Figure 2.6. Quantities related to the solid use S subscript, while fluid domain and the interface are labeled with F and FS, respectively.

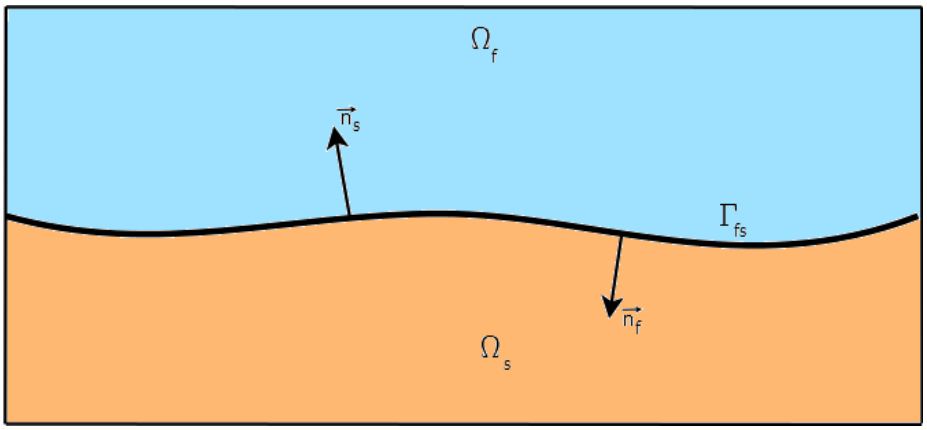


Figure 2.6. fluid solid interface

In order to have a physically correct behavior, some conditions have to be met [39]:

- solid and fluid domains should neither overlap nor separate,
- for a viscous fluid model, the flow velocity at the interface must equal the boundary velocity (*no-slip* condition),
- for an inviscid fluid model, only velocity components normal to the wet surface have to be equal to the structural velocity as the fluid may slip freely in tangential direction at any boundary,
- forces exchanged at the interface must be at equilibrium.

The first two conditions result in the *kinematical requirement* that the displacements of fluid and solid domains, as well as their respective velocities have to be equal at the wet surface (denoted by Γ_{FS}):

$$\Delta\vec{x}_F = \vec{u}_S \tag{2.12}$$

$$\vec{v}_F = \frac{\partial\vec{u}_S}{\partial t} \tag{2.13}$$

The last condition results in the equilibrium requirement. Force vectors are computed from the stresses at the interface and the outward normal vectors of fluid and solid domain, respectively. They have to be equal and opposite leading to the dynamic coupling condition:

$$\boldsymbol{\sigma_F} \cdot \vec{n}_F + \boldsymbol{\sigma_S} \cdot \vec{n}_F = 0 \tag{2.14}$$

$\boldsymbol{\sigma} \in \mathbb{R}^{3\times3}$ represents the stress tensor (note that for the fluid it comprises pressure and viscous stresses), while $\vec{n} \in \mathbb{R}$ is the outward normal unit vector.

2.3 Classification of FSI problems

In the previous chapters we have seen that there exist a lot of models that can describe fluid flow and solid mechanics. In Fluid-Structure Interaction problems we need to couple two of them: the variety of coupled problems seems to be so large that single FSI model that is applicable to every problem appears to be unfeasible. For this reason it is useful to classify FSI problems and look for specific properties in each class. The first step is to switch from dimensional quantities to dimensionless ones.

2.3.1 Dimensional analysis

We use the principle that a physical law should only relate to dimensionless quantities. There exist a rather general theorem called the Π Theorem or the *Vaschy-Buckingham Theorem* [33], which tells how many dimensionless quantities are needed to rewrite a model in dimensionless fashion. This theorem states that the number of dimensionless quantities, P, is equal to that of the dimensional ones describing the problem, N, minus R, which is the rank of the matrix of dimension exponents. This matrix is formed by the columns of the dimension exponents of all variables [34]. An example is given in the following Section 2.3.2, Table 2.1.

2.3.2 Dimensional analysis in fluid domain

Dimensional analysis is widely used in fluid dynamics. In order to keep the analysis simple, we consider the adimensionalization of the incompressible Navier-Stokes momentum equation equation for a Newtonian fluid [20]:

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\, \vec{v} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \vec{v} + \vec{g} \tag{2.15}$$

The variables involved in equation 2.15 are the following:

- t: time
- $\vec{x}$: coordinates
- $\vec{v}$: velocity field
- p: pressure field
- ρ: fluid density
- ν: fluid kinematic viscosity
- $\vec{g}$: gravity (or body forces)
- L: reference dimension
- V_0: reference velocity

	t	$\vec{x}$	$\vec{v}$	p	ρ	ν	$\vec{g}$	L	V_0
L	0	1	1	-1	-3	2	1	1	1
M	0	0	0	1	1	0	0	0	0
T	1	0	-1	-2	0	-1	-2	0	-1

Table 2.1. fluid matrix of dimension exponents

The rank of the above matrix is 3 so 6 dimensionless parameters are needed to rewrite the equation 2.15:

- *length*: $\vec{x}^* = \frac{\vec{x}}{L}$
- *velocity*: $\vec{v}^* = \frac{\vec{v}}{V_0}$
- *time*: $t^* = \frac{V_0 t}{L} = \frac{t}{T_{fluid}}$
- *pressure*: possible choices: $p^* = \frac{p}{\rho V_0^2}$ or, if viscous forces are dominant, $p^* = \frac{pL}{\rho \nu V_0}$
- *Reynolds number*: $Re = \frac{V_0 L}{\nu}$. It defines the ratio between inertia and viscous forces
- *Froude number*: $Fr = \frac{V_0}{\sqrt{gL}}$. It defines the ratio between the flow inertia to the body field forces

The adimensionalized momentum equation becomes:

$$\frac{\partial \vec{v}^*}{\partial t^*} + (\vec{v}^* \cdot \nabla)\, \vec{v}^* = -\nabla p^* + \frac{1}{Re} \nabla^2 \vec{v}^2 + \frac{1}{Fr^2} \vec{g} \tag{2.16}$$

From Equation 2.16 a lot of models might be derived, from *Stokes regime* when viscosity is dominant, to *Euler regime* when viscosity is negligible with respect to inertia forces.

2.3.3 Dimensional analysis in solid domain

Even if it is seldom used, Dimensional Analysis can be made also for the solid domain [49].

The variables involved in a solid dynamics equation are:

- t: time
- $\vec{X}$: coordinates
- $\vec{u}$: displacement field
- ρ_S: solid density
- E: elastic modulus
- $\vec{g}$: gravity
- L: reference dimension
- U_0: reference displacement

From the variables above the following parameters can be derived:

- *length*: $\frac{\vec{X}}{L}$: dimensionless coordinate
- *displacement*: $\frac{\vec{u}}{L}$: dimensionless displacement
- *time*: $\frac{t\sqrt{\frac{E}{\rho_S}}}{L} = \frac{t}{T_{solid}}$ dimensionless time
- entity of displacements: $\frac{U_0}{L} = \delta$: *displacement number*
- gravity: $\frac{\rho_S g L}{E}$: *elastogravity number*

T_{solid} can be seen as $\frac{L}{c}$ with $c = \sqrt{\frac{E}{\rho_S}}$ which is the scale of elastic wave velocity. The displacement number δ tells how big the structure displacement are related to the overall dimension, and defines the *large displacements* region. Finally, the *elastogravity number* combines gravity (or body forces in general), density and stiffness: when large the deformation induced by body forces in the solid are large.

2.3.4 Dimensional analysis of coupled problems

It is now possible to undertake the dimensional analysis of a fully coupled fluid and solid interaction problem. Some of the parameters are only defined in the fluid side or in the solid side (e.g. viscosity or stiffness). Some parameters are common to both domains (e.g. length scale or gravity). The variables of interest are now the velocity in the fluid and the displacements in the solid. Each of them can be related to all the parameters without separation. For example, the fluid velocity relationship is of the kind:

$$g(\vec{v}; \vec{x}, t; \rho, \mu, V_0; \rho_S, E; \vec{g}, L) = 0 \tag{2.17}$$

Equation 2.17 is composed of 11 dimensional parameters. Applying π theorem, the total number of independent dimensionless parameters is expected to be 8. Starting from the ones derived in the previous sections:

1. $\vec{x}^* = \frac{\vec{x}}{L}$: dimensionless coordinates
2. $\vec{v}^* = \frac{\vec{v}}{V_0}$: dimensionless fluid velocity
3. $t_f^* = \frac{V_0 t}{L}$: dimensionless time
4. $Re = \frac{V_0 L}{\nu}$: Reynolds number
5. $Fr = \frac{V_0}{\sqrt{gL}}$: Froude number
6. $\delta = \frac{U_0}{L}$: displacement number
7. $\frac{\rho_S g L}{E}$: elastogravity number

The 7 quantities above derive from the separated problems. The last one necessarily mixes things from the fluid and the solid side otherwise it would have been found in one uncoupled case. There is no unique choice for this parameter, the following are the most common ones.

Mass number

The simplest, but arguably most important parameter is the ratio of the two densities: the *Mass Number* M.

$$M = \frac{\rho}{\rho_S} \tag{2.18}$$

This can range from $\mathcal{O}\left(10^{-4}\right)$ in air-steel interaction to $\mathcal{O}\left(1\right)$ when both media have about the same density. This parameter is particularly significant for the so called *added mass* stability problem, described in Section 3.5.

Reduced velocity

Another possible choice is the reduced velocity:

$$U_R = \frac{V_0}{\sqrt{\frac{E}{\rho_S}}} \tag{2.19}$$

It is the ratio between the free fluid velocity and the velocity of elastic waves in a solid, c. It contains information on the way the two dynamics are related and it can range different orders of magnitude.

Cauchy number

Another possible parameter combines stresses or stiffness. It is known as the Cauchy number, as defined in [11]:

$$C_Y = \frac{\rho V_0^2}{E} \tag{2.20}$$

It is the ratio between the fluid inertial forces, quantified by the dynamic pressure, and the stiffness of the solid E. The higher it is, the more the solid is elastically deformed by the flow.

These are actually the most important parameters involving Fluid-Structure Interaction problems. Among them, there is no universally better choice but there are efficient choices that would be more helpful in solving or in analyzing a given problem.

Chapter 3

Computational aspects of Fluid-Structure Interaction problems

This section deals with the computational aspects of FSI problems. The first possible categorization of solution techniques distinguishes between monolithic and partitioned approach, as discussed in Section 3.1. This work is based on the latter approach, so its two different coupling strategies, namely strong and weak, are discussed in Section 3.2. As strong coupling is generally needed for accurate solution, an overview of strong coupling algorithms is given in Section 3.3. Section 3.4 focuses on aspects concerning the interface mesh, and how the solid and the fluid exchange data between them. Finally Section 3.5 briefly describes a common issue arising in strongly coupled problems: the added mass effect (AME).

3.1 Monolithic and Partitioned Approach

Analytical solutions are impossible to obtain for the large majority of Fluid-Structure Interaction problems; on the other hand, laboratory experiments may be costly, unfeasible or limited. For those reasons, numerical simulations may be employed to analyze the physics involved in the interaction between fluids and solids. With the current capabilities of computer technology, simulations of scientific and engineering models have become increasingly detailed and sophisticated.

The numerical methods used to solve FSI problems may be roughly classified into two classes: the *monolithic approach* and the *partitioned approach*. There is no exact distinction between the two approaches, as they might be seen differently among fields of applications. The idea here is to consider how many solvers are used to find a solution.

In the *monolithic approach*, the whole problem is treated as a unique entity and solved simultaneously with a specialized ad hoc solver (see Figure 3.1). The fluid and structure dynamics form a single system of equations for the entire problem, which is solved simultaneously by a unified algorithm. The interface conditions are implicit in the solution procedure [40], [64].

This approach can potentially achieve better accuracy, as it solves the system of equations exactly and the interface conditions are implicit in the model [62], but it may require more resources and expertise to develop from scratch a specialized code (it solves a very specific model) that can be cumbersome to maintain.

On the other hand, in the *partitioned approach*, the fluid and the solid domains are treated as two distinct computational fields, with their respective meshes, that have to be solved separately (see Figure 3.2: how data are passed between solvers is detailed in Section 3.2).

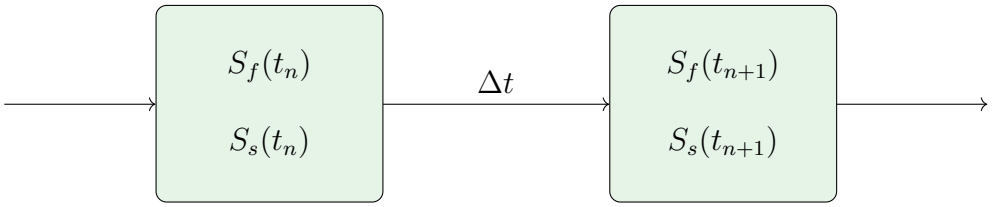


Figure 3.1. monolithic approach: S_f, S_s denote the fluid and the structure solutions

The interface conditions are used explicitly to communicate information between the fluid and structure solutions. This implies that the flow does not change while the solution of the structural equations is calculated and vice versa [13]. The partitioned approach thus requires a third software module (i.e. a coupling algorithm) to incorporate the interaction aspects. It communicates the boundary conditions described in Section 2.2.4: i.e. forces or stresses (dynamic data), calculated by the fluid solver at the wet surface, are passed to the solid component and displacements or velocities (kinematic data), computed by the solid solver at the interface, are sent to the fluid component in return. Finally, fluid and structural solutions together yield the FSI solution.

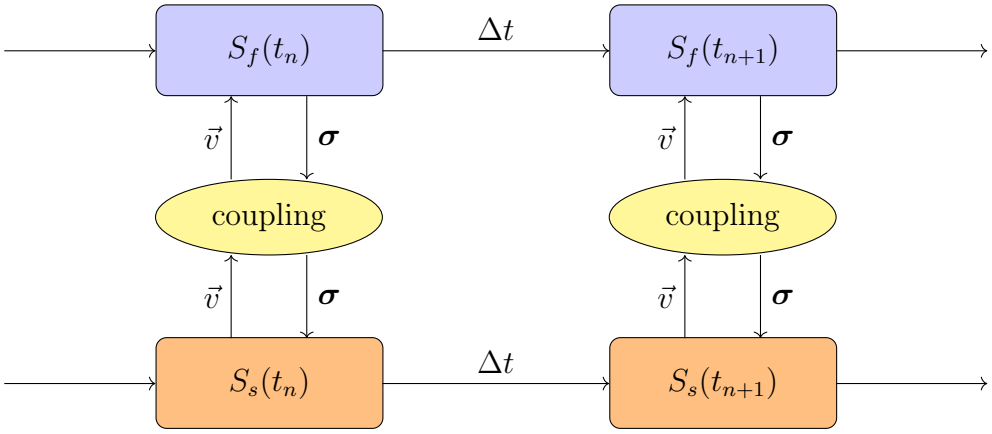


Figure 3.2. partitioned approach: S_f, S_s denote the fluid and the structure solutions, while $\boldsymbol{\sigma}$ and $\vec{v}$ represent coupling data

A big advantage of this approach is that software modularity is preserved: different and efficient solution techniques can be used for the flow equations and structural equations. Provided that they can exchange data, existing solvers for the fluid and solid problem can be reused, ranging from commercial to academic and open-source codes. Those solvers are usually well-validated. Besides, compared to monolithic procedures, the programming efforts are lower for partitioned approaches, as only the coupling of the existing solvers has to be implemented rather than the solvers themselves. The challenge of this approach is, however, to define and implement algorithms to achieve accurate and efficient fluid-structure interaction solution with minimal code modification. Particularly, the interface location that divides the fluid and the structure domains changes in time. The partitioned approach requires that the fluid solver has ALE capabilities, as introduced in Section 2.1.3. More detailed and practical explanations about the coupling component used in this work are given in Section 4.2.

3.2 Coupling Strategies

Because of the modularity, the partitioned approach has gained much attention in research. The structure sketched in Figure 3.2 needs to be detailed and specialized in function of the coupling strategies.

In an interface multi-physics coupling like FSI, the boundary surface is in common between the two sides of the simulation. The results make sense and are numerically stable only if the two sides of the interface are in agreement, since the output values of the one simulation become input values for the other (and vice-versa). The solution strategies can be roughly divided into weakly and strongly coupled approaches. They are often referred to as *explicit* and *implicit* methods in the literature. When the fluid and solid solutions are computed iteratively until some convergence criteria within the same time step, the scheme is called *implicit coupling*. The faster, simpler but less precise *explicit coupling* consists in executing a fixed number of iterations (typically one per time step) and exchange coupling values without convergence checks.

3.2.1 Explicit coupling schemes

As in the previous Section, S_f represents the fluid solver, which computes the stresses (named d_f here) at the deformable boundary. S_s is the structure solver, which uses these forces to compute the displacement and velocity of the boundary (named d_s). In a *serial-explicit* (or *conventionally staggered*) coupling scheme, the solver S_f uses the old time step boundary values $d_s^{(n)}$ to compute the values of $d_f^{(n+1)}$ for the next time step:

$$d_f^{(n+1)} = S_f\left(d_s^{(n)}\right) \tag{3.1}$$

When the fluid solver completes the time step, data are passed to the structural solver:

$$d_s^{(n+1)} = S_s\left(d_f^{(n+1)}\right) \tag{3.2}$$

Note that Equation 3.1 uses values computed at t^n, while Equation 3.2 uses values computed at $t^{(n+1)}$. The order of execution might be inverted.

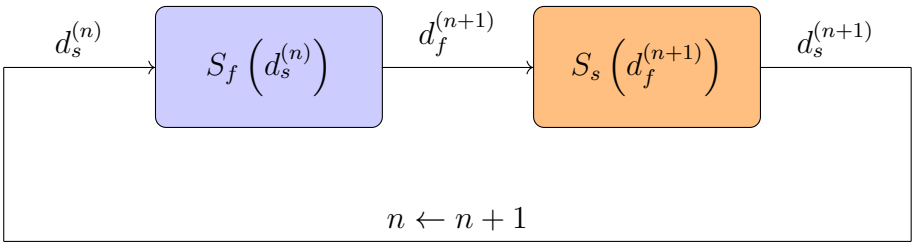
In order to reduce execution time, the solvers might run in parallel, using data from the same time step (*parallel-explicit coupling*):

$$d_f^{(n+1)} = S_f\left(d_s^{(n)}\right) \tag{3.3a}$$

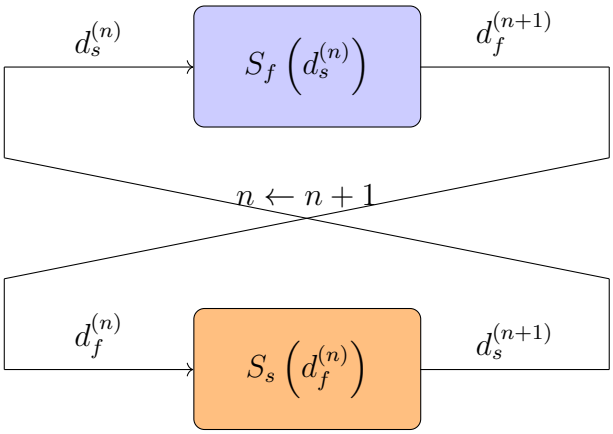
$$d_s^{(n+1)} = S_s\left(d_f^{(n)}\right) \tag{3.3b}$$

The two explicit approaches are shown schematically in Figures 3.3a and 3.3b.

In general, an explicit coupling is not enough to regain the exact (as in the monolithic approach) solution of the problem as the matching of coupling conditions between the solvers is not enforced within each time step: no balance between fluid and structural domain with respect to forces and displacements at the interface can be guaranteed ([39], [13]). Nevertheless, explicit coupling yields good results if the interaction between fluid and solid is weak, as in aeroelastic simulations, where in general the simulations show small displacements of the structure within a single time step and the flow field isn't much influenced by the structural displacements [18].



(a) serial explicit coupling



(b) parallel explicit coupling

Figure 3.3. Explicit coupling schemes

3.2.2 Implicit coupling schemes

On the other hand, strongly (implicit) coupling techniques require an iterative method to solve the fixed-point equation that derives from enforcing the agreement of the interface variables. The coupling conditions at the wet surface are enforced in each time step up to a convergence criterion. If the criterion is not met, another subiteration within the same time instance is computed. Therefore, the solution can approximate the monolithic solution to an arbitrary accuracy.

As in the explicit case, solvers may run in a sequential mode: the coupling is then named *serial* (or staggered) and the solvers wait for each other.

$$d_f^{(n+1),i+1} = S_f\left(d_s^{(n+1),i}\right) \tag{3.4a}$$

$$d_s^{(n+1),i+1} = S_s\left(d_f^{(n+1),i+1}\right) \tag{3.4b}$$

Equations 3.4 show that, in contrast with explicit coupling, both solvers use interface values at time step $n+1$, but one of them uses data from previous iteration. If run in parallel mode [53], the system becomes:

$$d_f^{(n+1),i+1} = S_f\left(d_s^{(n+1),i}\right) \tag{3.5a}$$

$$d_s^{(n+1),i+1} = S_s\left(d_f^{(n+1),i}\right) \tag{3.5b}$$

At convergence, the following relation holds of serial (or *Gauss-Seidel*) coupling:

$$d_s^{(n+1)} = S_s\left(S_f\left(d_s^{(n+1)}\right)\right) \tag{3.6a}$$

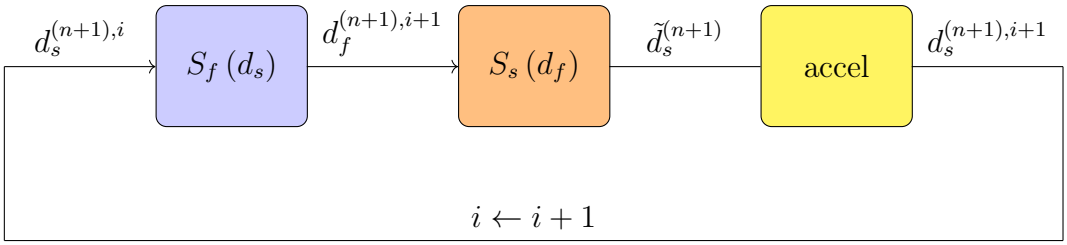
$$d_s^{(n+1)} = S_s \circ S_f\left(d_s^{(n+1)}\right) \tag{3.6b}$$

and the following relation holds for parallel (or *Jacobi*) coupling:

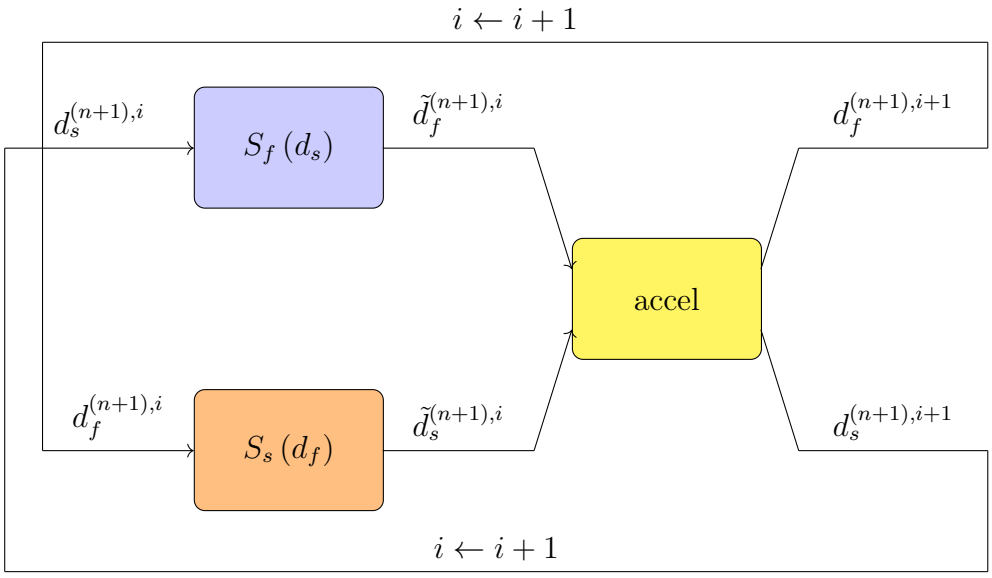
$$\begin{pmatrix} d_s^{(n+1)} \\ d_f^{(n+1)} \end{pmatrix} = \begin{pmatrix} 0 & S_f \\ S_s & 0 \end{pmatrix} \begin{pmatrix} d_s^{(n+1)} \\ d_f^{(n+1)} \end{pmatrix} \tag{3.7}$$

Acceleration techniques are necessary to bring fixed point equation 3.6b or 3.7 to convergence. Those techniques are described in Section 3.3.

The two implicit schemes are shown schematically in Figures 3.4a and 3.4b: *accel* refers to the post-processing step implemented to speedup convergence. After every non-converged iteration, the latest stored state of the solver (*checkpoint*) is reloaded and coupling iteration i for the current time step is incremented. When the solution converges, the time step n is incremented.



(a) serial implicit coupling



(b) parallel implicit coupling

Figure 3.4. Implicit coupling schemes

Implicit methods are generally applicable to any kind of FSI problems, in contrast with explicit methods. When fluid and structure are strongly coupled, explicit coupling can be

subject to numerical instabilities, a problem that cannot always be solved by reducing the coupling time step size [71]. These instabilities can be overcome by implicit methods, even if several coupling iterations may be executed every time step, until the values on both sides of the interface converge.

3.3 Strong coupling algorithms

As mentioned in the previous section, implicit methods require some post-processing (generally called *acceleration*) techniques to to make the solution of the single time step of the coupled partitioned FSI problem converge. This requires to solve a a *fixed-point equation*, in fact:

$$H(d_s) \coloneqq S_s \circ S_f(d_s) \tag{3.8a}$$
$$d_s = H(d_s) \tag{3.8b}$$
$$R(d) \coloneqq H(d) - d = 0 \tag{3.8c}$$

Equation 3.8a represents the composition of the solid and the fluid solution, while Equation 3.8b represents the resulting fixed point equation. As the order of execution can be switched, in Equation 3.8c, where the *residual* is defined, the input data d_s is generically substituted with d.

The basic approach to solve the fixed point equation is to perform the corresponding fixed point iteration (FPI):

$$x_{i+1} = H(x_i) \quad i = 1, 2, \dots \tag{3.9}$$

which is known to converge if the mapping H is a contraction, but this is not the general case in FSI computations [53].

3.3.1 under-relaxation

The way to stabilize the iterations is to perform a FPI with *under-relaxation* as illustrated in the following algorithm:

Algorithm 1: FPI with relaxation

Result: d_k
initialization of d_0;
k = 0;
$\tilde{d}_1 = S_s \circ S_f(d_0)$;
$r_0 = \tilde{d}_1 - d_0$;
while $\|r_k\| > \varepsilon$ **do**
    compute d_k by relaxation;
    k = k +1;
end

The under-relaxation is defined by:

$$d_{k+1} = d_k + \omega \left(H(d_k) - d_k \right) \tag{3.10}$$

Where ω in Equation 3.10 is the *relaxation factor*. The relaxation parameter has to be small enough to keep the iteration from diverging, but as large as possible in order to use as

much of the new solution as possible [45]. The optimal ω value is problem specific and not known a priori. A suitable dynamic relaxation parameter is a better choice, like the *Aitken under-relaxation* [41] which adapts the factor at each iteration with the following relation:

$$\omega_i = -\omega_{i-1} \frac{r_{i-1}^T (r_i - r_{i-1})}{\|r_i - r_{i-1}\|^2} \tag{3.11}$$

Aitken under-relaxation can be a good choice for strong interaction with a fluid solvers that does not fully converge in every iteration or for compressible fluid solvers.

3.3.2 Quasi-Newton Least Squares schemes

Under-relaxation is a good choice for easy stable problems, but is outperformed by more sophisticated quasi-Newton coupling schemes. Equation 3.8c could be solved iteratively with a Newton method [70]:

$$R(d_k) := r_k \tag{3.12a}$$

$$R(d_k) + \left.\frac{\partial R}{\partial d}\right|_{d_k} (d_{k+1} - d_k) = 0 \tag{3.12b}$$

$$d_k + \left(\left.\frac{\partial R}{\partial d}\right|_{d_k}\right)^{-1} (-r_k) = d_{k+1} \tag{3.12c}$$

The *residual* at iteration k is defined in Equation 3.12a. If the Jacobian matrix of the equation is known, a Newton iteration can be performed as in Equation 3.12b. The updated values can be computed using Equation 3.12c.

In situations where:

- *black-box* systems are considered (i.e. the Jacobian is unknown),
- the cost of a function evaluation is sufficiently high that numerical estimation of the Jacobian is prohibitive,

there exist a number of matrix-free methods that use only information derived from the consecutive iterations and that build an approximation based on those values. This approach is known as *quasi-Newton method* [32]. Input and output data of H and R are used to approximate the solution of 3.12c. Algorithm 2 (taken from [69]) shows the basics

steps to estimate data at next step using the *Quasi-Newton Least Squares Method*:

Algorithm 2: Quasi-Newton Least Squares method

Result: d_{k+1}

1 initial value d_0;
2 $\tilde{d}_0 = H(d_0)$ and $R_0 = \tilde{d}_0 - d_0$;
3 $d_1 = d_0 + \omega r_0$;
4 **for** $k = 1 \ldots$ **do**
5 $\quad \tilde{d}_k = H(d_k)$ and $r^k = \tilde{d}_k - d_k$;
6 $\quad V^k = \left[\Delta r_0^k, \ldots, \Delta r_{k-1}^k\right]$ with $\Delta r_i^k = r^i - r^k$;
7 $\quad W^k = \left[\Delta \tilde{d}_0^k, \ldots, \Delta \tilde{d}_{k-1}^k\right]$ with $\Delta \tilde{d}_i^k = \tilde{d}_i - d_k$;
8 $\quad$ decompose $V^k = Q^k U^k$;
9 $\quad$ solve the first k lines of $U^k \alpha = -{Q^k}^T R^k$;
10 $\quad \Delta \tilde{d}^k = W^k \alpha$;
11 $\quad d_{k+1} = \tilde{d}_k + \Delta \tilde{d}_k$;
12 **end**

In algorithm 2 the matrices V^k and W^k are constructed from the previous iterations and the known values of $d_0, \ldots d_k$ and $\tilde{d}_0, \ldots, \tilde{d}_k$. $\Delta \tilde{d}^k$ is constructed in the column space of W^k (line 10). For this reason a least squares problem is solved:

$$\alpha = \underset{\beta \in \mathbb{R}^k}{\operatorname{argmin}} \|V^k \beta + R(d_k)\| \tag{3.13}$$

The least squares problem is solved computing the decomposition of V^k into an orthogonal matrix $Q^k \in \mathbb{R}^{k \times k}$ and an upper triangular matrix $U^k \in \mathbb{R}^{n \times k}$ (line 8). Then α is computed in line 9.

When building matrices V^k and W^k (lines 6-7), it is possible to use information from previous time steps.

Finally, to ensure linear independence of columns in the multi-secant system for Jacobian estimation, a filter can be used [31], in order to drop nearly dependent columns of Q^k and avoid singularity of the approximated Jacobian.

The above algorithm is usually denominated in FSI interface quasi Newton with inverse Jacobian from a least squares model (IQN-ILS) (or Anderson acceleration). There exist other algorithms, like generalized Broyden (IQN-IMVJ) or manifold mapping to solve the problem. A complete description of those methods goes beyond the scope of this work: a presentation of the most common algorithms can be found in [3], while a comparison of the performances can be found in [46].

3.3.3 Convergence criteria

At each time step, the coupling algorithm enforces matching conditions at the wet surface up to a convergence criterion. If not sufficiently met, another iteration within the same time step is performed. The fixed point formulation itself induces a criterion based on the current residual r_{k+1}.

A scalar *absolute convergence criterion* can be defined as in Equation 3.14: it is useful for close to zero values of the coupling quantities, when rounding errors become important:

$$\|r_{k+1}\| \leq \epsilon_{abs} \tag{3.14}$$

The more common *relative convergence criterion*, defined in Equation 3.15, is particularly useful when different quantities (e.g. forces and displacements) are compared together to evaluate convergence:

$$\frac{\|r_{k+1}\|}{\|\tilde{d}_{k+1}\|} \leq \epsilon_{rel} \tag{3.15}$$

3.4 Interface Mesh and Data Mapping

FSI methods can also be classified considering how the fluid and solid meshes are treated. The *conforming mesh methods* consider the interface as a physical boundary condition (see Section 3.4.2), while *non-conforming mesh methods* treat the boundary location as a constraint imposed on the model equations (see Section 3.4.1) [39].

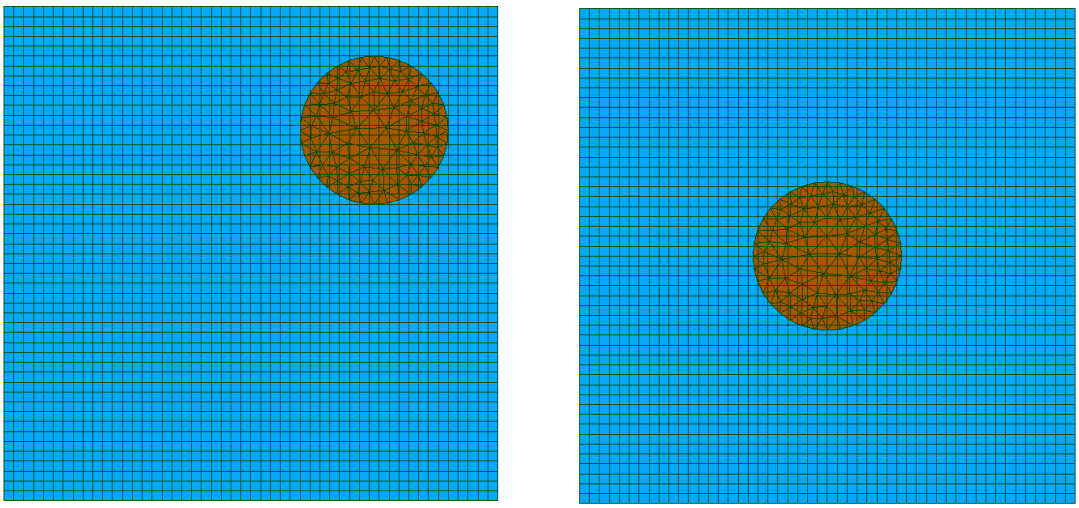
3.4.1 Non-conforming Mesh methods

In non-conforming mesh strategies all interface conditions are imposed as constraints on the flow and structural governing equations. It is possible to use non-conforming meshes for fluid and solid domains as they remain geometrically independent from each other (see Figure 3.5).

This approach is mostly used in *immersed boundary* methods [42]. Coupling is imposed by means of additional force terms appearing in the model equations of the fluid, which impose the kinematic and dynamic conditions. The forces represent the effects of a boundary or body being immersed in the fluid domain. A purely Eulerian mesh (see Section 2.1.1) can be used for the whole computational domain, since the force terms are dynamically added at specific locations to represent the structure.

The fluid forces applied on the solid at the wet surface are computed and used as input for the structural solver, which employs a standard Lagrangian mesh (see Section 2.1.2).

Immersed boundary methods are particularly innovative and are useful to overcome some issues in CFD computations, on the other hand most of the current implementations of FSI problem implement a conforming mesh strategy.



(a) mesh at $t = t_1$ (b) mesh at $t = t_2$

Figure 3.5. non conforming mesh example

3.4.2 Conforming Mesh methods

Conforming mesh methods adapt very well to the partitioned approach described in Section 3.1, as they usually consists in the computational steps described above, namely: computation in the fluid, computation in the solid, enforcing of interface conditions and mesh movement (see Figure 3.6).

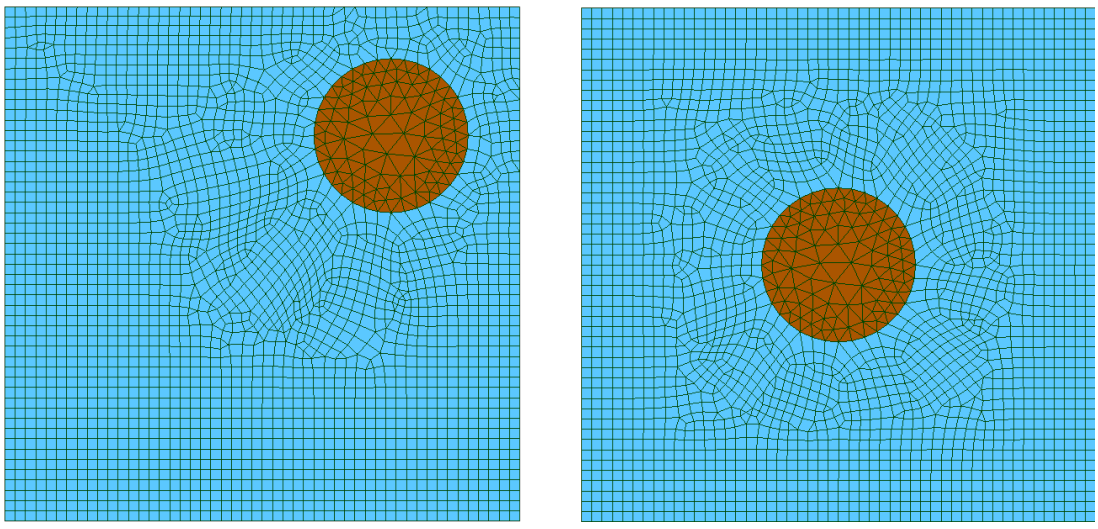
Fluid and structure meshes need to share the boundary of the wet surface, as the coupling conditions are enforced by applying kinematic or dynamic conditions to those boundaries. Node-to-node matching of fluid and structure meshes at the interface is not required, as long as a suitable mapping between the interface nodes is performed (see Section 3.4.3).

The match between the interfaces must hold at each time step: this implies that both solid and fluid domains need to deform. Deformation is easily expressed in the solid domain as the structural mesh is usually represented in Lagrangian perspective (see Section 2.1.2). ALE perspective (Section 2.1.3) for the fluid domain becomes necessary in this case.

Mesh deformation can turn out to be a complicated task as in general the fluid mesh is deformed during motion (see Figure 2.4). Mesh smoothing techniques need to be applied in order to keep a good mesh quality in terms of distorted elements which can lead to accuracy loss in simulations. (the following video shows highly distorted fluid elements during FSI motion: video).

Mesh smoothing is generally applied to keep the fluid mesh as uniform and undistorted as possible during movement. There is a wide variety of mesh updating procedures [72]. The *torsional spring analogy* [12] is a fairly simple technique that computes mesh movement considering mesh edges as springs and solving the subsequent Laplace equation that derives from the mesh movement.

Some other references about mesh motion alternatives can be found in [29].



(a) mesh at $t = t_1$ (b) mesh at $t = t_2$

Figure 3.6. conforming mesh example

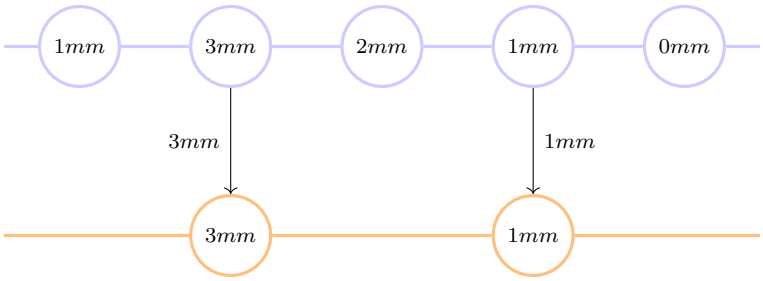
3.4.3 Data Mapping

When partitioned coupling is involved and the meshes are conforming but not node-to-node coincident, the challenge is to correctly map the data between the solid and the fluid sides.

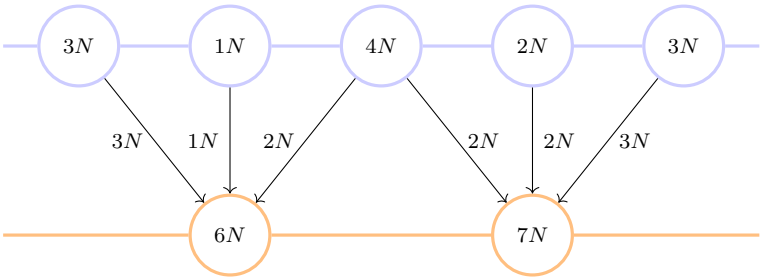
This is a common situation as fluid and solid require a different mesh refinement at the interface.

The mapping procedure needs not only to find the closest available mesh point (or points) on the opposite mesh, but also to preserve mass and energy balance. Variables are basically mapped in two ways: *consistent* and *conservative* forms.

In the *consistent mapping* a value on a node of one grid has the same value of the corresponding node on another grid: that is, it reproduces the values on both meshes. In the *conservative form*, integral values are preserved between meshes. In an FSI problem, nodal forces are mapped in conservative form, while velocities or displacements are mapped in consistent form. An example is shown in Figure 3.7.



(a) displacements: consistent mapping (nearest neighbor)



(b) forces: conservative mapping

Figure 3.7. Examples of mapping data between non-coincident meshes: consistent (a) and conservative (b) schemes.

different mapping strategies can be implemented [5]:

- *Nearest Neighbor*: finds the closest point on the source mesh and uses its value for the target mesh. It does not require any topological information and is first-order accurate. It is the computationally easiest implementation and it is useful when interface meshes are coincident.
- *Nearest Projection*: projects the points of the target mesh on the source mesh, interpolates the data linearly and assigns the values to the target mesh. It requires topology information for the source mesh. The interpolation on the mesh elements is second order accurate.
- *Radial Basis Function (RBF)*: this method does not requires topological information and works well on general meshes. The mapping uses radial basis functions centered at the grid points of the source mesh [47].

3.5 Stability: Added Mass Effect

When a solid moves or vibrates in a fluid domain, the interaction changes the way in which the structure behaves. There exists a vast variety of literature (e.g.: [7], [9], [24], [61]) describing the effects of Fluid-Structure Interaction in terms of *added mass*, *added damping* or even *added stiffness* on a vibrating structure in function of fluid properties (e.g. density or viscosity), or flow properties (e.g. velocity), or geometry.

Besides the physical aspects of the interaction, some numerical issues arise when trying to simulate this kind of problems.

The numerical issue named added mass effect (AME) is introduced here, as it is relevant for both strongly and weakly coupled partitioned approaches in the solution of FSI problems.

Weakly coupled algorithms give good results in aeroelasticity studies, but they are known to become unstable under certain conditions, in particular when fluid and structure densities are comparable ($M \approx 1$, see Section 2.3.4) and when the structure is particularly slender [6]. Under the same conditions, strongly coupled algorithms exhibit convergence problems.

For this reason, the mass ratio M is a suitable indicator to determine the kind of interaction between solid and fluid: when $M \ll 1$ (i.e. when the solid is much denser than the fluid), the interaction is weak, while when densities are comparable the interaction is strong and imposes some limits on the partitioned solution techniques.

The problem has been analyzed in literature by means of "toy FSI models" (in [6], [14] or [53]). Even though the number of parameters affecting stability is large and not completely understood in complex scenarios, all of the studies point out the mass ratio M as the most relevant parameter.

A simplistic explanation of the phenomenon stems from the idea that at the interface, fluid and structure have no gaps (Section 2.2.4). For this reason, if the structure moves, also the fluid particles around it have to move: the acceleration of the surrounding fluid results in greater inertial forces and the structure appears more inert.

Added mass effect appears both in incompressible and in compressible fluid models, but with slightly different effects and implications.

3.5.1 AME in compressible regime

In compressible regime, the use of a weakly coupled algorithm, which does not enforce mass and energy balance at the interface (see Section 3.2.1), imposes a limit to the mass ratio above which the simulation becomes unstable and the algorithm fails to find a solution [4].

A strongly coupled algorithm (Section 3.2.2) does not become unstable, but converges slowly: many sub-iterations are needed at each time step in order to reach the required convergence criteria (Section 3.3.3).

It can be shown that, in the compressible case, changing the time step of the partitioned simulation can be beneficial to the solution. The time step reduction to an arbitrarily small value cannot stabilize a weakly coupled algorithms when the stability criterion on the mass ratio is not met, but has an effect on strongly coupled ones [71].

A step size reduction can compensate a higher mass ratio value [19]: the convergence of a strongly coupled algorithm improves proportionally to the time step reduction. At the theoretical limit of vanishing time step, the monolithic solution could be found.

3.5.2 AME in incompressible regime

AME is a much greater issue in incompressible regime than in compressible flows. A simple physical explanation of this fact could be as follows: a deformation of the structure results

in a perturbation of the fluid domain close to the structure, which then propagates through the rest of the fluid domain. In compressible models, the speed at which a perturbation propagates (speed of sound) has a finite value. For this reason the effect of a perturbation is spatially limited during a time step. In contrast, in an incompressible model, the speed at which the aforementioned perturbation propagates through the domain is infinitely large. A change in the geometry affects the whole domain without delay and impacts the whole domain at each time step [6].

It has been observed that loose coupling of fluid and structural part in the context of incompressible flow and slender structures frequently yields unstable computations [19]. However, strict stability limits exist also for strongly coupled algorithms. Those limits have a different relation to the time step with respect to compressible models.

Simulations show that reducing the time step may result in increased instability. The AME is inherent in the coupling itself: in sequentially staggered schemes the fluid forces depend upon predicted structural interface displacements rather than the correct ones and thus contain a portion of incorrect coupling forces. This contribution yields the instability [14].

Provided that the stability limit is not exceeded, implicit methods behave differently in incompressible regime: the number or sub-iterations required to reach convergence during a single time step increases when the time step decreases. Besides, achieving the monolithic solution limit is not guaranteed ([19], [71]).

These observations are consistent with the aforementioned physical explanation.

Chapter 4

Software Packages used in this work

This chapter illustrates the main software tools used in this work. First, the MultiBody Dynamics analysis software (MBDyn) is shortly presented in Section 4.1. Then, the coupling library precise Code Interaction Coupling Environment (preCICE) is introduced in Section 4.2.

4.1 MBDyn

MBDyn is free and open-source[1] general purpose Multibody Dynamics analysis software developed at the *Dipartimento di Scienze e Tecnologie Aerospaziali* of the University Politecnico di Milano.

Most of the information concerning MBDyn is taken from the official documentation given in the software website[2] and from the input manual[3].

MBDyn allows to build a system to simulate multi-body dynamics, which[4] is the study of the behavior of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacements.

MBDyn can simulate linear and non-linear dynamics of rigid and flexible bodies (including geometrically exact and composite-ready beam and shell finite elements, component mode synthesis elements, lumped elements) subjected to kinematic constraints, external forces and control subsystems [51]. MBDyn has been developed to serve as an analysis tool for rotorcraft research and can simulate essential fixed-wing and rotorcraft aerodynamics.

As explained more in detail later, MBDyn is open to be connected to other software components to perform multi-physics simulations. In particular, it is possible to pass externally computed forces and to steer the multi-body simulation from an external API: this feature has been particularly useful in building and *adapter* to couple MBDyn with the library preCICE (see Chapter 5).

In this section we give a short introduction to some of the relevant features of MBDyn, starting form basic information on the input file syntax (Section 4.1.1), and the output files (Section 4.1.8). Then some of the elements relevant for the description of the models used in the this work are presented: nodes (Section 4.1.2), beam elements (Section 4.1.4), bodies (Section 4.1.5), joints (Section 4.1.6) and forces (Section 4.1.7).

[1] the software is available through a public git repository gitlab.polimi.it/Pub/mbdyn

[2] MBDyn website: mbdyn.org

[3] a copy can be found at the following link [input manual] or in the code repository

[4] see Wikipedia entry: Multibody system

4.1.1 Basic syntax

MBDyn is a command line tool and can be generally started from a terminal passing an input file containing all the information required to perform a simulation.

The input files is structured in blocks and each block has a syntax described in Backus Naur form in the input manual.

```
begin : data ;
   # select a problem
   problem : initial value ;
end : data ;

begin : initial value ;
   # problem-specific data
end : initial value ;

begin : control data ;
   # model control data
end : control data ;

begin : nodes ;
   # nodes data
end : nodes ;

begin : elements ;
   # elements data
end : elements ;
```

Listing 4.1. MBDyn input file structure

As illustrated in listing 4.1, statements are logically divided in blocks. Each block is opened by a `begin` statement and it is closed by an `end` statement.

The sequence of relevant valid blocks is:

- `data`: it defines the kind of problem to be solved by the analysis. The most significant one is `initial value` and is already defined in the example.
- type of problem: it takes the name of the problem defined in the data block (`initial value` in this case) and contains all the information required by the integration method to perform the desired simulation (e.g. simulation type, time step, number of iterations, tolerance ...).
- `control data`: it mostly contains the required information to ensure that a consistent model will be generated (i.e. the number of nodes, elements, forces...).
- `nodes`: this block contains all the nodes required by the simulation. They are defined as the entities that make degrees of freedom available to the simulation, so they must exist before any element is generated.
- `elements`: it contains all the elements. They are defined as the entities that generate equations using the degrees of freedom provided by the nodes.

We focus now on the main entities of a MBDyn simulation, namely nodes and elements. A detailed example of the input file used in most simulations in this work can be found in Appendix D.1.

4.1.2 Nodes

Nodes are the basic blocks of a model: they instantiate kinematic degrees of freedom and the corresponding equilibrium equations. There can be different type of nodes in MBDyn, we focus here on `structural nodes`.

Structural nodes can have 3 degrees of freedom, thus describing the kinematics of point mass motion in space (position) or 6 DoFs (position and orientation), and thus describing the kinematics of rigid-body motion in space.

Each node has a unique label and can be specified in different ways:

- `static`: this keyword instantiates only equilibrium equations (force for 3 DoF nodes or force and moment for 6 DoF nodes)
- `dynamic`: this also instantiates momentum (3 and 6 DoFs) and momenta moment (6 DoFs)

Also `modal` and `dummy` nodes exist, but are beyond the scope of this work. Nodes are the starting point to define the elements of a simulation.

4.1.3 Elements

Elements constitute the components of the multi-body model. Each has a unique numerical label and is connected to one or more nodes. They write contributions to nodes equations and represent *connectivity* and *constitutive properties*.

There exist many types of elements in MBDyn, we focus here on the ones used in the FSI simulation: beams, bodies, joints and forces.

4.1.4 Beam elements

As briefly introduced in Section 2.2.3, when simulations involve slender bodies, it is particularly interesting to use a 1D finite element model together with a form of mapping between the interface (wet surface) and the model to exchange kinematics and dynamics information. This can be performed in MBDyn using `beam` elements (described in this section) and the `external structural mapping` element (described in Section 4.1.7).

MBDyn models slender deformable components by means of finite volume `beam` elements with a high level of flexibility.

The beam element is defined by its nodes and a reference line; although 2 and 3 nodes beam elements are implemented, only `beam3` are considered here (Figure 4.1). Each node of the beam is related to a `structural node` (node 1 to 3 in Figure 4.1) by an offset (o_1 to o_3) and a relative orientation.

The Finite Volume approach described in [25] is used to model the beam element. It computes the internal forces as functions of the reference line strain and as functions of the orientation at the *evaluation points* (i.e. integration points, point I and II in Figure 4.1) that are between nodes 1 and 2, and between nodes 2 and 3 (at $\xi = -1/\sqrt{3}$ and $\xi = 1/\sqrt{3}$ of a non-dimensional abscissa $-1 \leq \xi \leq 1$ ranging from node 1 to node 3).

A 6D constitutive law is defined at each evaluation point: it relates the strains and the curvatures of the beam (and their time derivatives) to the internal forces and moments at

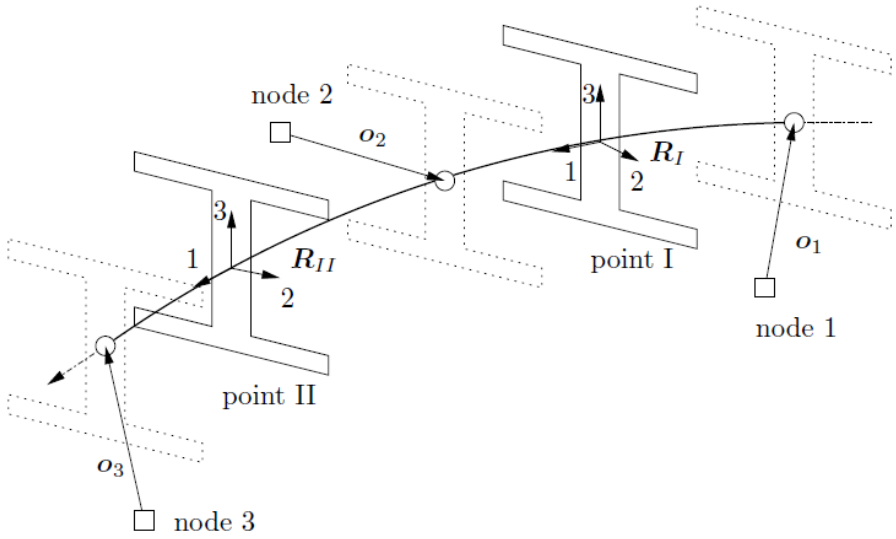


Figure 4.1. MBDyn beam model, taken from the input manual

the evaluation points in the form:

$$\begin{Bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{Bmatrix} = f\left(\begin{Bmatrix} \epsilon_x \\ \gamma_y \\ \gamma_z \\ \kappa_x \\ \kappa_y \\ \kappa_z \end{Bmatrix}, \begin{Bmatrix} \dot{\epsilon_x} \\ \dot{\gamma_y} \\ \dot{\gamma_z} \\ \dot{\kappa_x} \\ \dot{\kappa_y} \\ \dot{\kappa_z} \end{Bmatrix} \right) \tag{4.1}$$

Using the convention of *x-axis* as beam axis we have:

- F_x: axial force component,
- F_y and F_z: shear force components,
- M_x: torsional moment component,
- M_y and M_z: bending moment components,
- ϵ_x: axial strain component,
- γ_y and γ_z: shear strain components,
- κ_x: torsional curvature component,
- κ_y and κ_z: bending curvature components,
- f: constitutive law.

Beam section Constitutive Law

In dynamic simulations, linear elastic or viscoelastic laws are generally used, even though nonlinear laws can be used. Focusing on linear laws, MBDyn allows the user to define every kind of constitutive laws, going from an isotropic beam section to a fully anisotropic one:

in fact the entire 6×6 constitutive matrix can be provided. It is up to the user to define a valid law as the matrix must satisfy some constraints, e.g. it must be symmetric. The simplest case of linear elastic constitutive law is represented in Equation 4.2, in which a diagonal matrix relates internal forces to strain and curvature of the beam.

$$f = \begin{bmatrix} EA & 0 & & \dots & & 0 \\ & \chi GA & & & & \\ & & \chi GA & & & \\ & & & GJ_p & & \vdots \\ & & sym. & & EJ_y & 0 \\ & & & & & EJ_z \end{bmatrix} \cdot \begin{Bmatrix} \epsilon_x \\ \gamma_y \\ \gamma_z \\ \kappa_x \\ \kappa_y \\ \kappa_z \end{Bmatrix} \tag{4.2}$$

The simplest case of linear viscoelastic law uses a proportional factor to be applied to the stiffness matrix of Equation 4.2 such that: $viscosity = factor \cdot stiffness$.

In the context of FSI simulations, this feature allows the user to define a section constitutive law that is independent from the shape of the beam itself. Thus the aerodynamic aspects and the structural aspects are handled by two distinct elements of the model: i.e. the interface mesh defines the aerodynamic forces and the beam constitutive law defines the structural properties.

Some studies about the definition of general beam section constitutive properties (*composite beam section characterization*) are available in the literature: an early work can be found in [27], a review in [37] or an application to wind turbine blades in [44].

4.1.5 Bodies

The `body` element describes a lumped rigid body when connected to a regular, 6 DoF structural node, or a point mass when connected to a rotationless, 3 DoF structural node. It can be used in connection with a structural element to give inertial properties: for example, in a `beam` element (see Section 4.1.4), 2 bodies are added to the evaluation points of the beam to account for lumped inertia of each portion in which the beam is divided.

4.1.6 Joints

`structural` nodes can be constrained by means of `joint` elements. Many different joints are available. In the FSI model the following types of joints are used:

- `clamp`: grounds all 6 DoFs of a node in an arbitrary position and orientation.
- `total joint`: allows to arbitrarily constrain specific components of the relative position and orientation of two nodes [50].

4.1.7 Forces

The `force` element is a general means to introduce a right-hand side term to the equations. Structural forces are specific to structural nodes and have three components that may depend on arbitrary parameters and a location in space.

MBDyn allows to communicate with an external software that computes forces based on information regarding the kinematics of the model. This feature is at the basis of the development of the *adapter*. The following elements can be used.

External Structural

The `External Structural` element allows to communicate with an external software that computes forces applied to a pool of `nodes` and may depend on the kinematics of those nodes. In this case forces are applied directly to the nodes. In a FSI model, this would require that each interface mesh node has a correspondent MBDyn structural node.

External structural mapping

This element is similar to the previous one, but the nodes where forces are applied and the kinematics is computed depend on structural nodes through a linear mapping. This element has been used in building the adapter. In order to use the `external structural mapping` elements, the following steps have to be performed:

1. a set of points is defined for each `structural node` according to a specified offset. Those points are used to compute the kinematics of the interface points, originating from the rigid-body motion of the structural nodes,
2. before the simulation, a linear mapping matrix H is generated starting from the position of the above points and the interface mesh points (this is performed by means of an `Octave` script which is part of MBDyn). The matrix is stored in sparse form,
3. the mapping matrix is used during the simulation to map forces and kinematics between the interface nodes and the structural nodes.

The constant matrix mapping allows to compute the position and the velocity of the *interface* points as function of the points rigidly offset from structural nodes:

$$x_{interf} = H x_{mbdyn} \tag{4.3a}$$
$$\dot{x}_{interf} = H \dot{x}_{mbdyn} \tag{4.3b}$$

The same matrix is used to map back the forces onto structural nodes based on the preservation of the work done in the two domains:

$$\delta x^T_{mbdyn} \cdot f_{mbdyn} = \delta x^T_{interf} \cdot f_{interf} = \delta x^T_{mbdyn} \cdot H^T \cdot f_{interf} \tag{4.4}$$

which implies

$$f_{mbdyn} = H^T f_{interf} \tag{4.5}$$

When performing an FSI simulation with strong coupling (see Section 3.3), MBDyn may need to compute multiple iterations of the same time step in order to reach global convergence. This is performed by using the keyword `tight` in the coupling of the `external structural mapping`. The computation and the communication pattern is the following:

1. MBDyn sends the predicted kinematics for time step k,
2. MBDyn receives a set of forces sent by the external peer; those forces are computed based on the kinematics at iteration j,
3. MBDyn continues iterating until convergence using the last set of forces until, while reading the forces, it is informed that the external peer converged. this implies that MBDyn solves the kinematics for time step k at iteration j using the forces evaluated by the external solver for iteration $j-1$.

The communication with the external software, in our case the adapter itself, is performed by means of a local unix socket.

4.1.8 Simulation output

There are different output files regarding a simulation performed with MBDyn. the name of those files is specified with the option `-o` (otherwise the name is the same as the input file) and the extensions are:

- `.out`: for miscellaneous output
- `.mov`: for kinematic output of the nodes
- `.ine`: for the dynamic output of the nodes
- `.frc`: for the output of force elements
- `.act`: for the output of beam elements
- `.jnt`: for the output of joint elements

The `.out` file contains information regarding the simulation iterations residuals while other files are described in more detail in Appendix D.2.

4.2 preCICE

The main information concerning preCICE is taken from the official documentation: i.e. [23] and [5]. The preCICE website is also a source of documentation[5].

The open-source[6] software library preCICE provides the components to connect traditional single-physics solvers and create a partitioned multi-physics simulation (e.g. fluid-structure interaction, conjugated heat transfer, solid-solid interaction, etc.). It aims at coupling existing solvers in a partitioned black-box manner (see Section 3.2): only minimal information about the solver is available and connection involves just the interface nodes.

In order to be flexible and easily implemented, the impact on the solvers should be as minimal as possible: for this reason, preCICE offers a high level application programming interface (API) (Section 4.2.5) in different languages, such as C/C++, Fortran and Python. The ability to switch among different solvers is advantageous as it provides a lot of flexibility in developing and testing new coupled components.

In a nutshell, preCICE simply affects the input and observes the output of the solvers (called *participants*). The required data and control elements are accessed using an *adapter*, i.e. a "glue code" that is attached to the corresponding solver and communicates the information with the library.

preCICE makes all the actions required to perform a coupled simulation:

- it implements the coupling strategy (Section 4.2.1),
- it verifies convergence criteria (Section 4.2.1),
- it instantiates the communication between the participants (Section 4.2.2) ,

[5] www.precice.org

[6] The code can be accessed via Github: github.com/precice/precice

- it computes the mapping of data between meshes (Section 4.2.3).

preCICE is configured by means of an extensible markup language (XML) file (Section 4.2.4).

4.2.1 Implemented coupling strategies

The *partitioned approach* (Section 3.2) is obviously the coupling strategy adopted by preCICE. It allows both *explicit* (Section 3.2.1) and *implicit* coupling (Section 3.2.2). The possible variants are four:

- `serial-explicit`: a serial, weakly coupled algorithm (Figure 3.3a). The first solver uses the second solver solution at the last time step to compute its current solution. In contrast, the second solver needs the current first solution to compute its solution at the same time instance. The order of execution is user defined.
- `parallel-explicit`: both solvers advance in parallel (Figure 3.3b) and exchange data at the end of each time step, resulting in a less stable procedure. The bottleneck of this procedure is related to the most time consuming solver.
- `serial-implicit`: a serial strongly coupled algorithm (Figure 3.4a). The user can define the order of execution, the coupling algorithm (basically all the algorithms described in Section 3.3) and its parameters in a section of the configuration file named `acceleration`.
- `parallel-implicit`: again both solvers execute in parallel (Figure 3.4b). An implicit scheme modifies the result of the fixed-point iteration on both data [53].

4.2.2 Communication strategies

All the participants need to communicate with each other, in order to share coupling data. Each solver might be executed in multiple processes or on different nodes of a cluster (*intrafield parallelism*). This form of parallelization requires efficient forms of communication between the solver in order to avoid that data transfer becomes a bottleneck during a simulation.

preCICE implements a fully parallel process-to-process communication approach [65] using:

- *message passing interface (MPI)*: available on most scientific computers, it may be necessary to adapt/change the MPI versions of the respective single-physics solvers or of preCICE.
- *Transmission Control Protocol/Internet Protocol (TCP/IP)*: popular means of network communication and free of incompatibilities between versions.

As to performances, MPI is the best technique especially when a high numbers of nodes is present. Anyway, socket communication is quite as fast, such that both techniques are very well-suited for larger-scale simulations [23].

In each solver, executed in parallel, one "master" process is defined to manage the progress of the simulation. No central node is required. The participating processes use asynchronous point-to-point (M:N) communication. The channels are static and defined in the beginning of the simulation. This sets a limit in using preCICE with dynamically adaptive meshes or immersed boundaries.

4.2.3 Data mapping

Even if volume coupling is possible, preCICE is mainly designed to couple simulations that share a common surface boundary (namely *conforming meshes*, in the terminology used in Section 3.4.2). The meshes do not need to be node-to-node coincident, so it is necessary to map variables at the interface, preserving the geometry (i.e. no gaps or superpositions at the interface) and the mass and energy balances.

The user defines which data are shared by each *participant* (i.e. solver) and the way data is shared in the configuration section named `mapping`. As described in Section 3.4.3, two kinds of mapping are available:

- `consistent`: the value of a node at the one grid is the same as the value of the corresponding node (or nodes) at the other grid. In general the number of fluid nodes is at least the same or, more often, exceeds the nodes of the structure, so a single structural node is associated to several fluid nodes. The mapping of displacements is consistent: in the simplest case, all fluid nodes experience the same displacement of a single solid node, otherwise an interpolation is performed (see Section 3.4.3 for a detailed explanation and Figure 3.7a for an example).
- `conservative`: in the same conditions as before, forces are mapped from multiple fluid nodes to a single solid node in an additive manner (see Section 3.4.3 and Figure 3.7b for an example).

Along with the mapping strategy, a method must be defined: `nearest-neighbor`, `nearest-projection`, `rbf` (see Section 3.4.3). For the latest method, preCICE implements a wide variety of basis functions, with Gaussian and thin plate splines being the most widely used.

4.2.4 Configuration

In order to run a multi-physics simulation with preCICE, all the participating, *adapted* solvers have to be started (the order is irrelevant). Some configuration files are needed:

- each adapter generally needs its own configuration file. It normally contains information about the boundaries (wet-surface) used for the coupling, the names of exchanged data, mesh and the name of the common preCICE configuration file, together with other parameters, specific to the adapter. The one for the MBDyn adapter will be described in more detail in Chapter 5 and in Appendix B.
- preCICE configuration file: This is an XML file and each participant points at it. It defines all the information relevant to the simulation:
  - type and name of exchanged data and meshes over which those data are passed,
  - which solvers participate in the simulation, which data produce or consume, and how the mapping is performed,
  - how solvers communicate among each other,
  - the coupling scheme and all the concerning necessary information.

  The structure of a preCICE configuration file is illustrated in Appendix A.

4.2.5 Application Program Interface

A solver, in order to be coupled to preCICE, must either provide a way to access its core functions (e.g. initialize, set input data, read output, advance...) from outside the code (via API, socket, etc...) or it has to be slightly modified in order to perform all the operations required by the preCICE library.

The result is an *adapter*, which can be the modified and recompiled original solver, or a standalone piece of code that communicates with the original unmodified solver on one side and with the preCICE library on the other. The adapter groups together all the calls to the preCICE methods from its API (a list of the API calls, taken from the official documentation, can be found in Appendix C.1).

While preCICE is written in C++, there exist APIs also for other languages, so that the adapter can be written also in C, Fortran or Python.

A coupling consists of a configuration and an initialization phase, multiple coupling advancements and a finalization phase: the general structure of an adapter can be found in Appendix C.2.

4.2.6 Official Adapters

This work introduces an adapter to preCICE for MBDyn. It is based on previous MBDyn adapters and on the examples given on the preCICE website[7]. Official adapters are currently available for several free solvers, e.g. CalculiX, Code-Aster and SU2. Also some closed-source software packages are supported. A (maybe outdated) list of official adapters can be found in [68], while the current status of coupled codes can be found at the following link: preCICE adapters.

[7] github.com/precice/precice/wiki/Adapter-Example

Chapter 5

MBDyn Adapter and its integration

To prepare an existing simulation code for coupling, preCICE has to be integrated with the solver, using an API described in Section 4.2.5 and in Appendix C.1. The "glue-code" required for this operation is called *adapter*, as depicted in Figure 5.1.

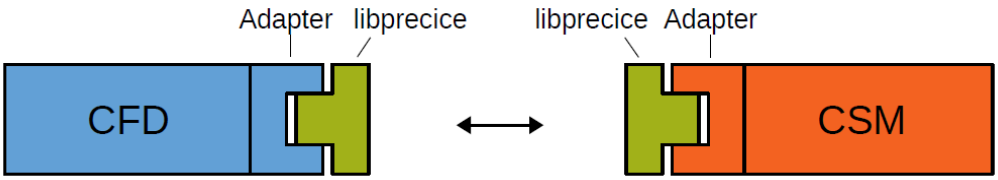


Figure 5.1. Coupling CFD to CSM via preCICE.The existing solver code, the adapter and the linked library are highlighted (image taken from [68]).

5.1 Design of the adapter structure

In order to couple MBDyn with preCICE, a C++ adapter has been implemented within the scope of this work. The *adapter* needs to be integrated with both the MBDyn solver and the coupling library. The two connections are distinct but strictly interconnected. The adapter has the advantage of being completely independent from both the preCICE library and MBDyn. The first connection is achieved via the API given by the library `libprecice.so`, the second connection exploits the API given by MBDyn through its library `libmbc.so`.

5.2 Structure of the code

The code for the adapter is available through a public git repository[1]. The code is conceptually divided in two classes, as illustrated in Figure 5.2.

The main class is `MBDynAdapter`, which implements the functions given by the preCICE interface. It has access to the class `MBDynConnector` which takes care of all the aspects regarding MBDyn. Attributes, methods and operations of each class are briefly described in the following sections.

[1] mbdyn-beam-adapter

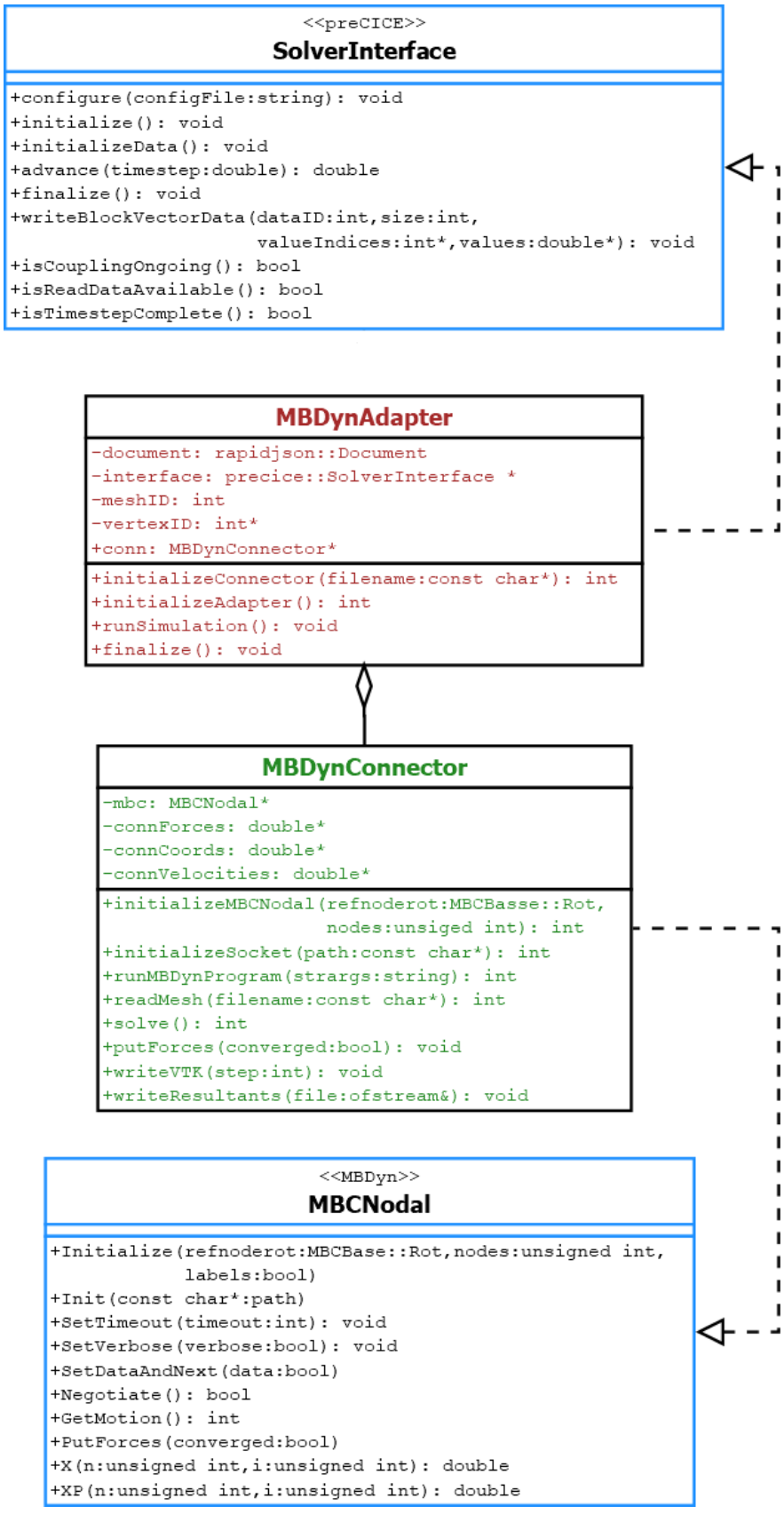


Figure 5.2. MBDyn adapter class structure

5.2.1 Class MBDynAdapter

The file `MBDynAdapter.h` and its source file `MBDynAdapter.cpp` implement all the methods required to perform a FSI simulation with MBDyn as the solid solver. The basic steps are:

1. prepare the MBDyn solver,
2. prepare the interface,
3. provide access to the mesh and initialize the coupling data,
4. steer the coupled simulation,
5. finalize the simulation.

Initialization

In the initialization phase, the instance of `MBDynAdapter` gets a JavaScript Object Notation (JSON) file (see Section 5.3) that contains all the parameters useful for the simulation. Then it instantiates the `MBDynConnector` (see Section 5.2.2) which takes care of all the operations concerning MBDyn: in particular starting the simulation and creating an instance of `MBCNodal` in order to have access to the simulation.

In the next step an instance of `precice::SolverInterface` is initialized and configured with all the relevant information data:

- preCICE configuration file (see Section 4.2.4 and Appendix A).
- *participant* (i.e. solver) name
- information regarding the data to be read and written

The next initialization step is the definition of the interface mesh. The data concerning the vertices are stored in the `MBDynConnector` to be used to plot the output and are passed to the `SolverInterface` to define the wet surface nodes. The mesh nodes are stored in the same text file that is used by MBDyn to build the `external structural mapping` information (see Section 4.1.7). This means that the MBDyn mapped points coincide with the interface mesh on the structural side (note that it doesn't have to be the same mesh of the fluid side, as preCICE can map non identical meshes, as described in 4.2.3). The suitable size of memory is then initialized to contain the coupling information: mainly *displacements*, to be written on the preCICE interface, and *forces*, to be read from the interface.

Execution

The simulation phase of the adapter follows the steps briefly described in Section 4.2.5 and in Appendix C. The most important elements of the code are illustrated in listing 5.1.

The main execution phases include:

1. read initial checkpoint (i.e. reload state if previous iteration did not converge) [line 6]
2. read *forces* from interface (i.e. get the current available fluid solution) [line 10]
3. copy forces to MBDyn: this step is performed as forces can be scaled by a user specified coefficient during the initial part of the simulation, see Section 5.3 [line 14]
4. make MBDyn connector solve the current simulation time step [lines 16-18]

5. make MBDyn connector compute force and moment resultants [line 22]
6. write the *displacements* computed at the current iteration of the current time step to preCICE in order to perform coupling [line 25]
7. get the interface time step (this would be mandatory in case of different time steps between fluid and solid) [line 28]
8. check if the current time step converged [line 31]: notify MBDyn [lines 35 and 38]. If converged, update time, iterations and write output data (see Section 5.4).

```
1  void MBDynAdapter::runSimulation(){
2    // Start coupling:
3    while (interface->isCouplingOngoing()){
4
5      if(interface->isActionRequired(cowic)){
6        interface->fulfilledAction(cowic);
7      }
8
9      // read forces from interface
10     interface->readBlockVectorData(forceID, vertexSize, vertexIDs, forces);
11
12     // copy forces to connector with coefficient
13     conn->computeForces(forces,coeff);
14
15         // MBDyn solves the current iteration
16     if(conn->solve()){
17       conn->writeVTK(iteration);
18       break;
19     }
20
21         // compute force and moment resultants on the structure
22     conn->computeResultants(true, root);
23
24     // write data to interface
25     interface->writeBlockVectorData(displID, vertexSize, vertexIDs,
           displacements);
26
27     // advance time step
28     precice_dt = interface->advance(precice_dt);
29
30     // Checkpoint
31     if(interface->isActionRequired(coric)){
32       // timestep not converged
33       interface->fulfilledAction(coric);
34       // tell MBDyn that time step not converged
35       conn->putForces(false);
36     }else{
37       // timestep converged
38       conn->putForces(true);
39       iteration++;
```

```
40        t += precice_dt;
41        conn->writeVTK(iteration);
42      }
43    }
44  }
```

Listing 5.1. MBDynAdapter simulation execution

Finalization

In the finalization phase all the objects used during the simulation are closed and memory is released.

5.2.2 Class MBDynConnector

The class `MBDynConnector` takes care of all the operations related to MBDyn and the structural side of the simulation.

Initialization

The instance of `MBDynConnector` is initialized within the `MBDynAdapter` and some different tasks are performed in this phase.

The first task is to read the mesh of interface points. The mesh file location is passed through the input file: to avoid too much duplication, it is the same file used by MBDyn to build the mapping matrix before the simulation (see Section 4.1.7). The mesh file contains information concerning the interface points and the connectivity. The (x, y, z) coordinates are passed to preCICE to build the structural part of the interface mesh. The same points, together with the connectivity, are stored in the `MBDynConnector` and are used to write the output in *VTK* format during the simulation (see Section 5.4).

The following task consists in starting MBDyn. The MBDyn input file location is passed through the JSON file and a separated process is spawned with the parameters required to run an MBDyn program. Upon a correct initialization, the MBDyn simulations hangs waiting for an external connection (i.e. the `external structural mapping`).

Finally, an instance of `MBCNodal` is created to take care of the communication with MBDyn: a socket is opened and the communication is initialized.

Execution

During the execution phase, `MBDynConnector` steers the simulation on the MBDyn side. Some of the actions have been already introduced in listing 5.1:

- send *forces* to the `external structural mapping` points,
- perform the simulation step,
- retrieve *displacements*,
- inform MBDyn that the time step has converged or not.

If the time step has converged (at a user defined time interval), the simulation data are saved.

5.3 Input parameters

Some input data are needed in order to perform a simulation. Such data are stored in a JSON file[2]. The file is given as input in the form:

```
mbdyn-beam-adapter -f config.json
```

An example of input file is given in Appendix B. The type of data can be related to different aspects of the simulation:

1. preCICE:
   - `precice-config`: the interface must be initialized with the *preCICE configuration file* (see Section 4.2.4 and Appendix A),
   - `readDataName`: label of the data to be read from the interface (typically forces),
   - `writeDataName`: label of the data to be written (typically displacements),
   - `participantName`: label the participant in the FSI simulation,
   - `meshName`: label of the interface mesh in the FSI simulation.
2. MBDyn:
   - `mesh`: location of the file containing the points mapped in MBDyn,
   - `mbdyn-input`: input file name to be passed to MBDyn,
   - `node-socket`: name of the socket to exchange data between the adapter and MBDyn (the name has been made configurable to allow different simulations to run in parallel using different sockets: the user must take care to use the same name in the MBDyn input file),
   - `mbdyn-output`: location and prefix of all MBDyn related output files (see Section 4.1.8 and Appendix D.2).
3. Simulation:
   - `displacement-delta`: Boolean value that tells the adapter whether to pass the relative displacement between two consecutive iterations or the displacement from the initial configuration. This is mainly due to compatibility with different fluid adapters (e.g. the current version of the *SU2 adapter* requires delta displacements),
   - `iterstart`: the following four parameters are used to give progression to the forces applied to the structure (Figure 5.3 gives an example of the behavior). The purpose is to ease the beginning of the simulation, letting the flow settle and avoiding initial spikes in the application of the forces that could result in a diverging simulation, The current parameter tells `MBDynConnector` how many iterations to perform with a reduced starting coefficient (200 in the example),
   - `coeff0`: this value is the initial coefficient (0.1 in the example),
   - `period`: this value is the number of iterations to reach 1 (600 in the example),

[2] see for example json.org for some information about the syntax

- `ramp-type`: label defining the type of ramp. A linear law or a $1/2 \cdot (1 - \cos)$ law can be used.

4. Output (see Section 5.4 for some more details):
   - `write-interval`: value that defines how often the `MBDynConnector` writes output data,
   - `resultant-file`: location of the file containing the resultant and moment of the fluid forces applied to the structure,
   - `root-coords`: vector containing the coordinates of the point about which the resultant moment has to be computed,
   - `vtk-output`: location and prefix of the output file.
5. Some parameters are used for debug purposes and might be removed in future releases:
   - `pre-iteration`: number of simulation iterations performed by MBDyn before coupling with the fluid solver. This can ease the initial coupling in case the definition of the structural model isn't in equilibrium,
   - `read-coords`: Boolean value used to read the mapped points coordinates back in the adapter from MBDyn instead of reading the mesh file,
   - `every-iteration`: Boolean value used to force the adapter to write output data at each coupling iteration.

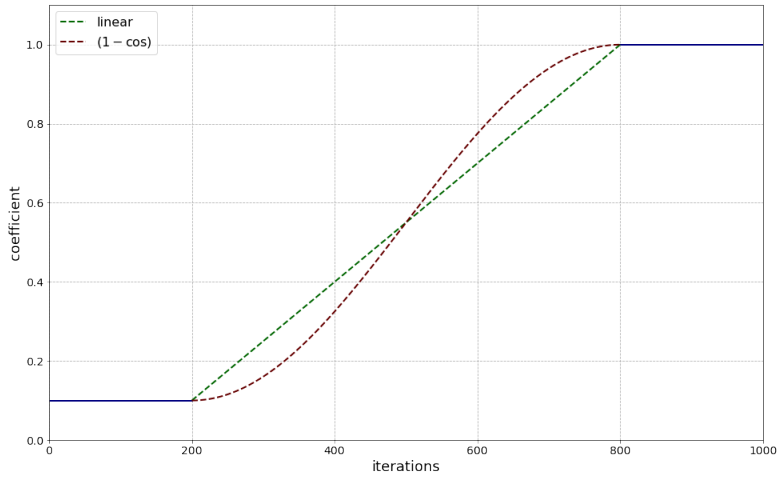


Figure 5.3. Coefficient applied to nodal forces at beginning of simulation

5.4 Output results

Besides the output strictly related to preCICE, concerning information about convergence and number of iterations, or the output strictly related to the MBDyn solution (see Appendix D.2), the adapter gives two types of output.

The information concerning the interface mesh points is saved in `vtu` file format, which is a binary XML VTK file format used to store data on unstructured grids of points. A sequential number corresponding to the current iteration is appended to the name of the file so that any visualization software (e.g. *ParaView*[3]) can load the whole dataset as a time series. Each file contains the following vector data:

- displacement,
- displacement delta,
- velocity,
- nodal force,

for each node. An example of visualization is given in Figure 5.4. For debug purposes also the area and the normal of each cell can be saved.

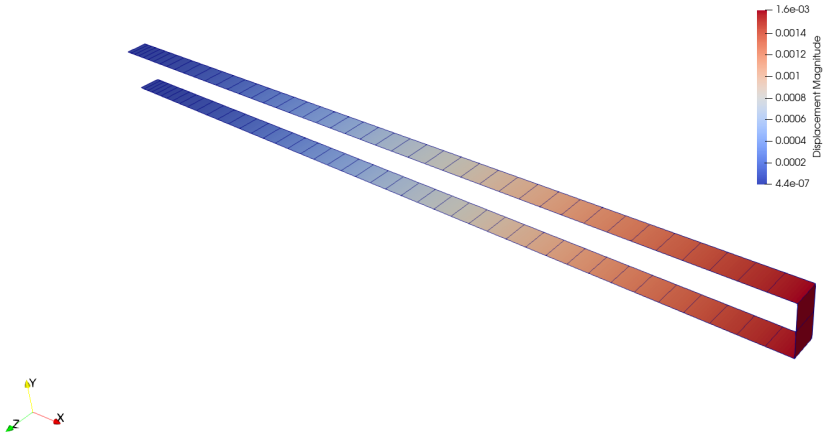


Figure 5.4. Output data in VTU format

The second output file contains the resultant and the moment applied to the structure by the fluid forces.

Each line of the file has the form:

`t␣Fx␣Fy␣Fz␣Mx␣My␣Mz␣c`

where t is the current time step, F_x, F_y, F_z are the components of the resultant, M_x, M_y, M_z are the components of the moment (the point is defined in the input file) and finally c is the coefficient applied to the nodal forces (see Figure 5.3).

[3] paraview.org

Chapter 6

Validation Test Cases

6.1 Introduction

In order to validate the coupling adapter developed in this work, some test cases have been simulated, which are supposed to qualitatively and quantitatively confirm the physical correctness of the implementation.

The validation has been carried out in different steps: the first task (Section 6.2) consisted in validating the MBDyn model that has been used throughout the simulations and in verifying that the implementation of the coupling works as expected.

The second step aimed at comparing the results obtained performing a FSI simulation with MBDyn with another structural solver, both in compressible and in incompressible regime (Sections 6.3 and 6.4).

Then, some results obtained from FSI simulations, coupling MBDyn with OpenFOAM, have been compared with some benchmarks present in literature. At first, the problem described in [60] has been considered (see Section 6.5): it is composed of a square bluff body with a trailing flap. It is characterized by a mass ratio (see Section 2.3.4) of $1.18 \cdot 10^{-3}$: this turned out to be a fundamental parameter for the convergence of simulations, and has been extensively analyzed in the subsequent sections.

One set of well-known benchmarks in FSI literature are the three Turek-Hron FSI test cases described in [66]. Those cases are characterized by the same domain (a round cylinder with a trailing flap) and the same fluid properties. Changes impact only fluid velocity and structural properties (in particular ρ, E).

At first, the so-called FSI2 benchmark (characterized by a mass number of 0.1) has been considered (Section 6.6). This benchmark, together with the previous one, shows the validity of the adapter and its potential use in FSI simulations.

Finally, the FSI3 benchmark has been extensively analyzed (Section 6.7). It has a mass number of 1 and, up to now, it has not been possible to find a suitable set of coupling parameters that can make this case converge when coupled with MBDyn.

It appears that the added mass effect (AME) (see Section 3.5), plays a dominant role in the convergence of a FSI problems with MBDyn. For this reason, a sensitivity analysis has been performed on the FSI3 problem setup (Section 6.8). In particular, flow velocity, fluid density and solid stiffness have been varied in order to gain some insight about the range of parameters and adimensional numbers that must be considered to understand how critical a simulation can be.

The last Section of this chapter (6.9) introduces a a real case scenario, comparing some of the results obtained in the experiments described in [35], concerning a flapping wing, with a simulation reproducing the same physical model.

Each section starts with a short description of the test case, including the most relevant parameters: e.g. the fluid domain geometry and discretization, structural and coupling parameters. Finally some results are presented, together with the coupling performances.

The meshes for all test cases have generated with the free software *Salome*[1]. Files for MBDyn interface points have been generated by a Python script running within the *Salome* environment. The script is part of the software made for this work.

6.2 Dummy fluid solver

The first task implemented to validate the adapter consisted in developing a simple *dummy fluid solver*: i.e. a software component connected to the coupling library preCICE with the simple task to apply forces to user-defined nodes on the structure and read back the displacements of the interface nodes.

This approach allowed first to validate the MBDyn model with the `external structural mapping` component (see Section 4.1.7) and the correct data exchange between preCICE and the adapter.

The *dummy solver* has been written in Python and it is part of the software package in this work.

A MBDyn cantilever beam model composed of 5 `beam3` elements (see Section 4.1.4) has been written to perform this test, as depicted in Figure 6.1. This requires a total of 11 `node` elements to define the structure.

The beam section is uniform and rectangular ($w \times h$) and the physical properties (e.g. ρ, E, ν) are constant throughout the beam length.

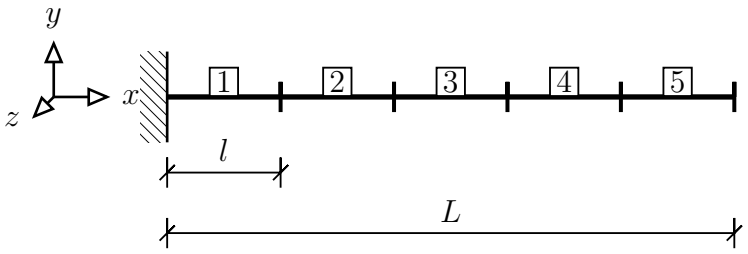


Figure 6.1. cantilever made of 5 beam elements

The inertia of the structure is provided by 2 `body` elements (see Section 4.1.5) attached to the second and third nodes of each beam (see Figure 4.1). The center of gravity of each body placed at the corresponding node. Each body has the following inertial properties (see Figure 6.2):

$$m = \rho w h \frac{l}{2} \quad I = \frac{m}{12} \begin{bmatrix} h^2 + w^2 & 0 & 0 \\ 0 & \frac{l^2}{16} + w^2 & 0 \\ 0 & 0 & \frac{l^2}{16} + h^2 \end{bmatrix} \tag{6.1}$$

The first node of the structure is clamped (to implement the cantilever constraint), while all other nodes are constrained (with `total joint` elements) to move in the $x - y$ plane and rotate only around the $z - axis$ so that the structure can move only in the $x - y$ plane. The interface mesh is represented in Figure 6.3. Only the connectivity elements belonging to the $x - z$ and $y - z$ plane are represented, but it does not affect the behavior if the structure.

[1] salome-platform.org/

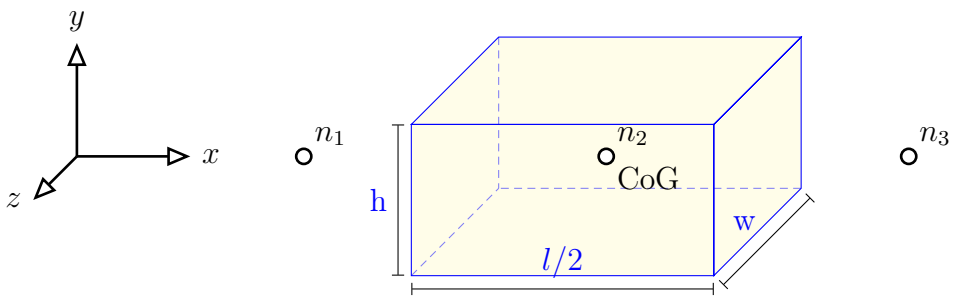


Figure 6.2. `body` element attached to node 2 of the `beam` element

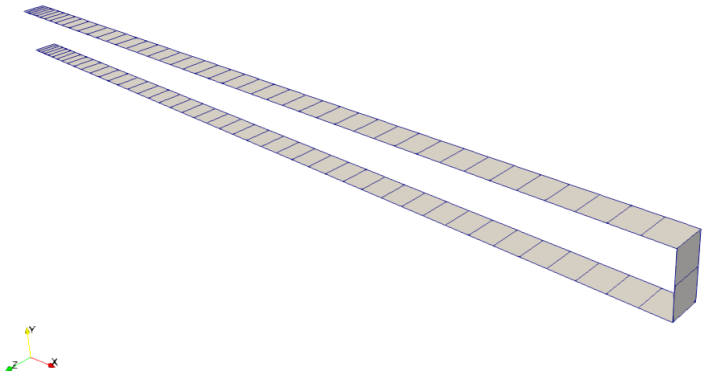
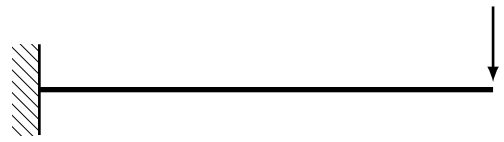
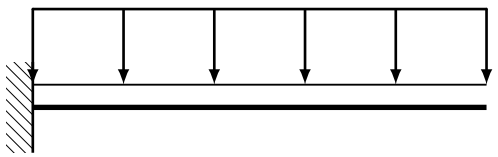


Figure 6.3. interface points mesh

The model has been loaded by a concentrated load on two tip nodes (Figure 6.4a) and with an equivalent distributed load on the nodes belonging to the upper surface (Figure 6.4b).



(a) cantilever with tip load



(b) cantilever with distributed load

Figure 6.4. model and data exchange test-cases

The results have been compared to the expected theoretical values in terms of tip displacement at steady state and in terms of frequency of the tip movement when the system has been loaded with a step load.

When both the MBDyn model and the data exchanged through the preCICE interface have been validated, the model has been coupled to a CFD solver.

6.3 Vertical flap: incompressible regime

The first validation step consisted in comparing the results obtained with MBDyn with the ones given by another structural solver with the same fluid model.

For this purpose a simple case of a vertical flap immersed in a flow in incompressible regime has been considered, borrowing an example given in the preCICE website.

6.3.1 Fluid domain

The fluid domain is represented in Figure 6.5. The inlet is on the left with uniform flow velocity, the outlet is on the right, while all other boundaries are *no-slip* walls.

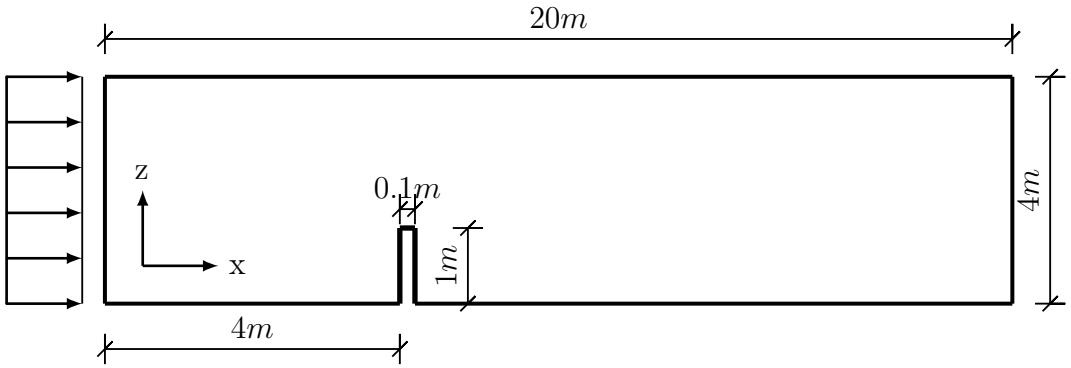


Figure 6.5. vertical flap: fluid domain

The fluid domain is discretized in an structured hexaedral mesh as depicted in Figure 6.6. The main fluid and mesh values are given in Table 6.1 and 6.2.

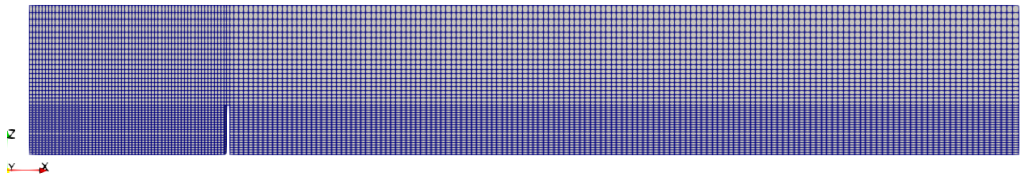


Figure 6.6. vertical flap: fluid mesh

parameter			value
fluid density	ρ	$[\mathrm{kg\,m^{-3}}]$	1
kinematic viscosity	ν	$[\mathrm{m^2\,s^{-1}}]$	10^{-3}
flow velocity	$\vec{v}$	$[\mathrm{m\,s^{-1}}]$	10
flow type			laminar

Table 6.1. Vertical flap: fluid properties

parameter		value
number of mesh points	n_{dof}	17344
number of cells	n_c	8400
number of interface cells	n_{int}	42

Table 6.2. Vertical flap: mesh properties

6.3.2 Simulation and Coupling parameters

The coupling between the fluid solver and the structural solver is the same for the MBDyn and the CalculiX simulation. The main data are given in Table 6.3:

parameter			value
simulation time	t	[s]	50
step size	Δt	[s]	10^{-2}
coupling scheme			serial implicit $S \rightarrow F$
coupling algorithm			IQN-ILS
displacement rel. convergence limit			10^{-4}
force rel. convergence limit			10^{-3}
interface mesh mapping			RBF

Table 6.3. Vertical flap: coupling parameters

6.3.3 Structural Solver: CalculiX

The structural model built in CalculiX is composed of 40 **C3D8**[2] as represented in Figure 6.7.

The properties of the solid are reported in Table 6.4.

parameter			value
solid density	ρ	[kg m^{-3}]	3000
Elastic modulus	E	[Pa]	$4 \cdot 10^6$
Poisson coefficient	ν		0.3

Table 6.4. Vertical flap: solid properties

6.3.4 Implementation with MBDyn

The MBDyn model uses the same solid properties of Table 6.4 and it is composed of 10 `beam3` elements. Apart from the different orientation, the setup is the same as the one briefly described in Section 6.2. The only relevant parameter that can be explicitly set up in MBDyn, compared to CalculiX, consists in the structural damping of the beam elements (see Section 4.1.4), which is set to be proportional to stiffness matrix of the element with a coefficient of $2 \cdot 10^{-3}$.

[2] general purpose linear brick element with 8 nodes

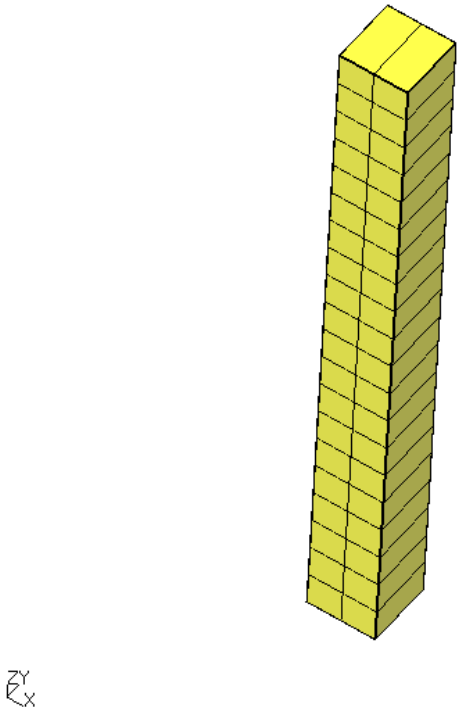


Figure 6.7. vertical flap: CalculiX mesh

The interface mesh is divided into 20 faces for the front and back surfaces of the flap. The upper surface is divided into 2, so that the interface is identical to the one obtained in CalculiX.

6.3.5 Results

The problem considered is characterized by the dimensionless parameters given in Table 6.5 and its solution is represented in Figure 6.8.

parameter		value
mass number	M	$3.3 \cdot 10^{-4}$
reduced velocity	U_R	0.274
Cauchy number	C_Y	$2.5 \cdot 10^{-5}$

Table 6.5. Vertical flap: dimensionless numbers

The solutions between CalculiX and MBDyn are first compared in terms of resultants applied to the structure during the simulation (Figures 6.9 and 6.10), then the tip displacement in x direction is considered in Figure 6.11. The forces and moments applied to the structure in the two cases are very close. The tip displacement shows that both structures exhibit the same damping and the oscillating frequency is very close: CalculiX shows a slightly higher frequency and a more visible second order frequency. The MBDyn structure looks a little more flexible: after 50 seconds of simulation tends to $58mm$ of tip displacement in x direction, while the same structure in CalculiX tends to $53mm$ of displacement.

The convergence and the number of iterations required by the two solvers are shown in Figures 6.12a and 6.12b. They show that, in general, MBdyn and CalculiX require 2

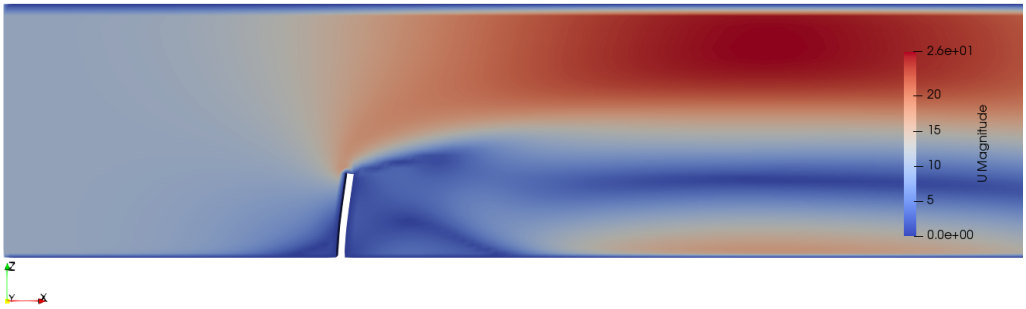


Figure 6.8. vertical flap: velocity field

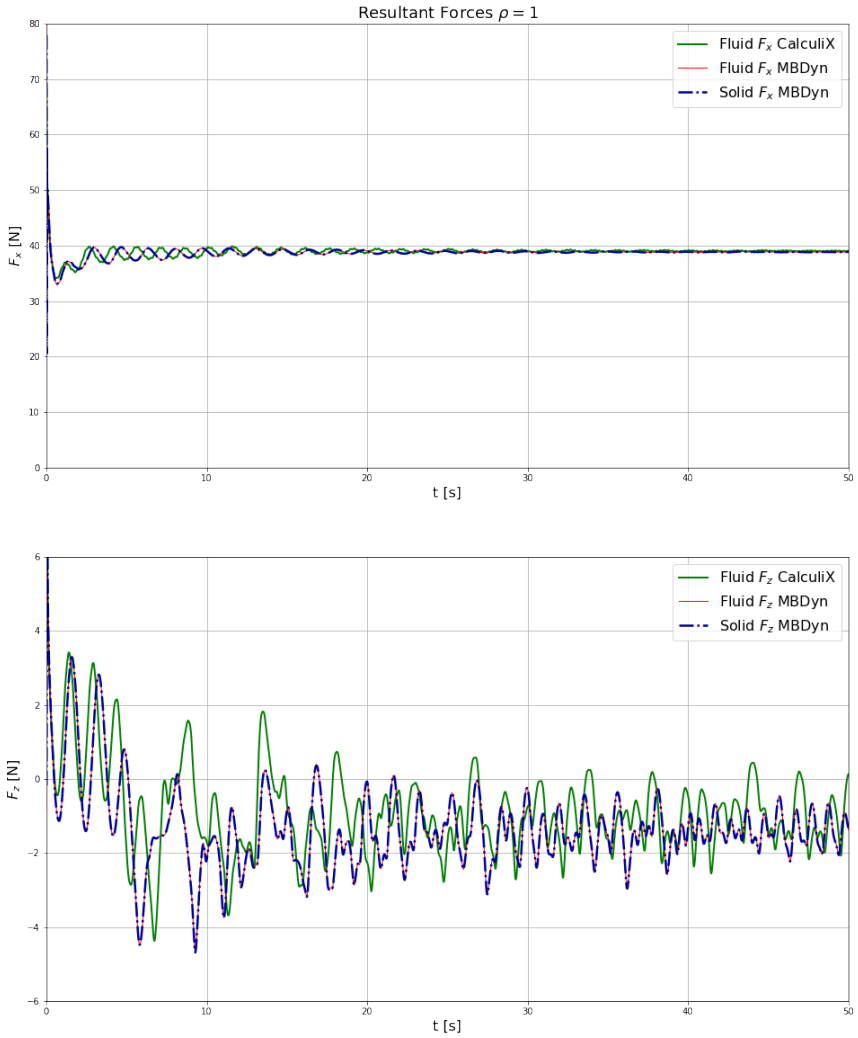


Figure 6.9. vertical flap: resultant forces

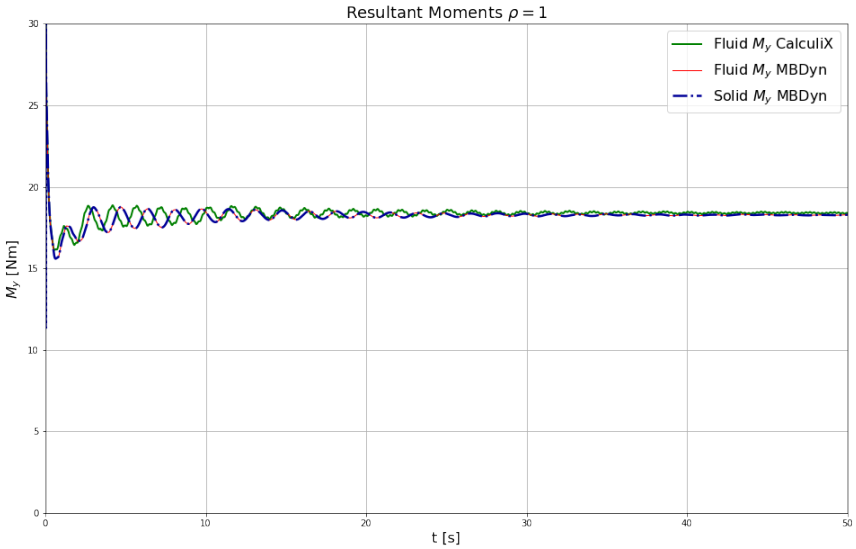


Figure 6.10. vertical flap: moment applied at root

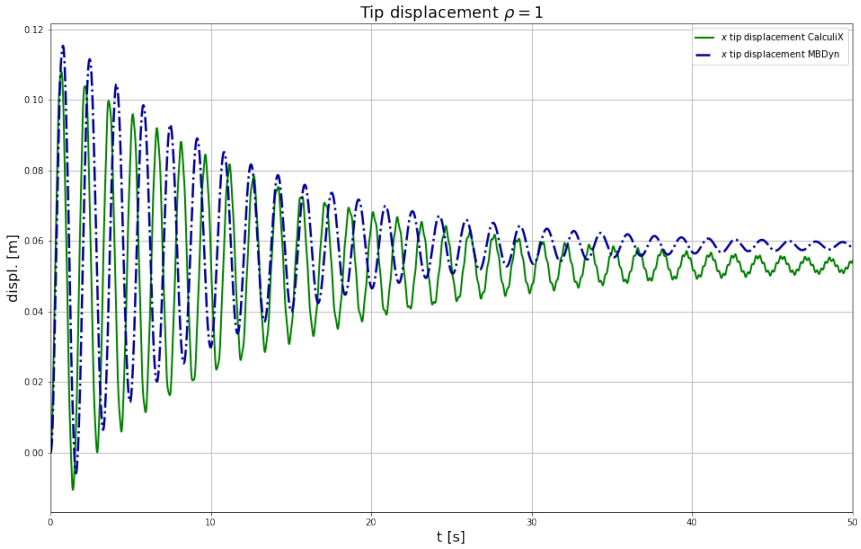
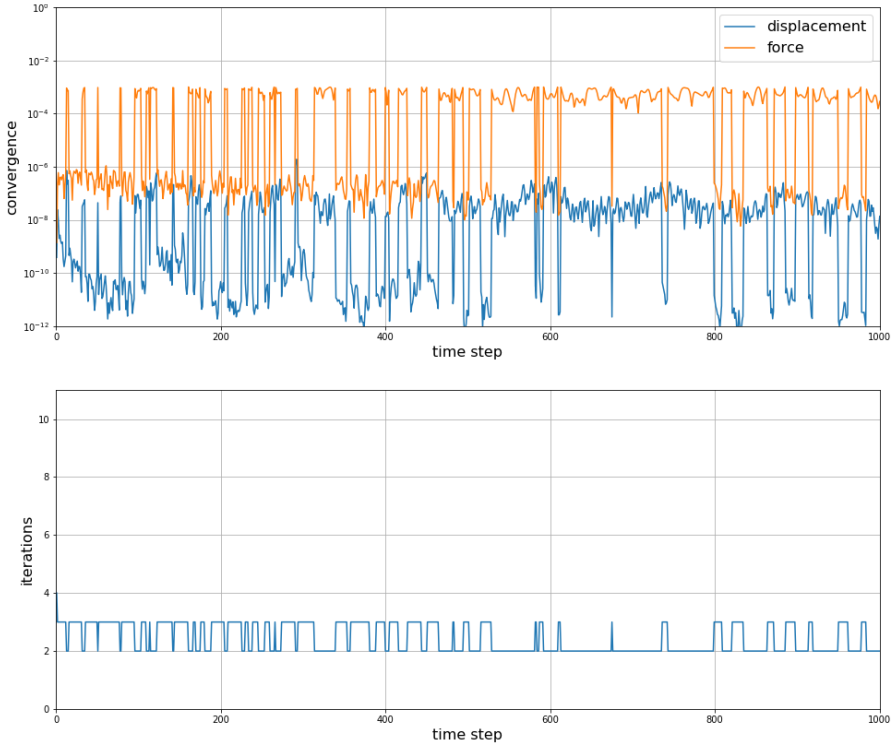


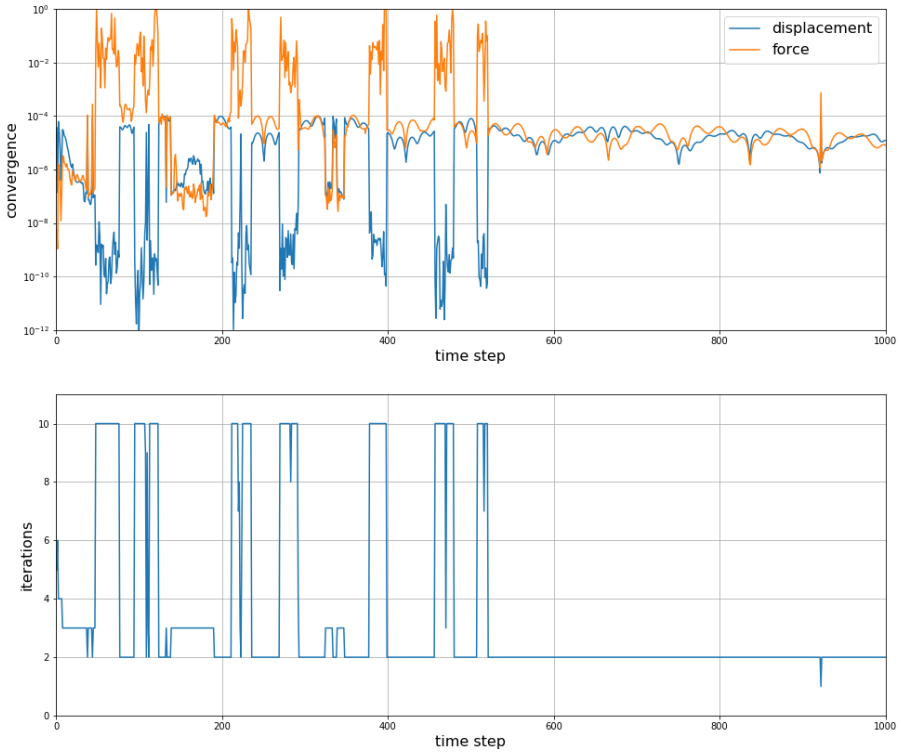
Figure 6.11. vertical flap: tip displacement x direction

iterations to converge. This can be explained by the small mass number of this case. At some time steps MBDyn requires more iterations: this is due to the fact that, for some reasons due to the coupling not completely understood at the moment, a force unbalance arises in the y-direction (out of the plane of the model) which produces a moment around x-axis. This moment is absorbed by the rotation constraints at the nodes and it does not affect the behavior of the resultant in x and z directions, but it affects the convergence.

Finally, MBDyn allows to analyze the internal forces of each beam, as represented in Figure 6.13.



(a) CalculiX



(b) MBDyn

Figure 6.12. vertical flap: convergence and iterations

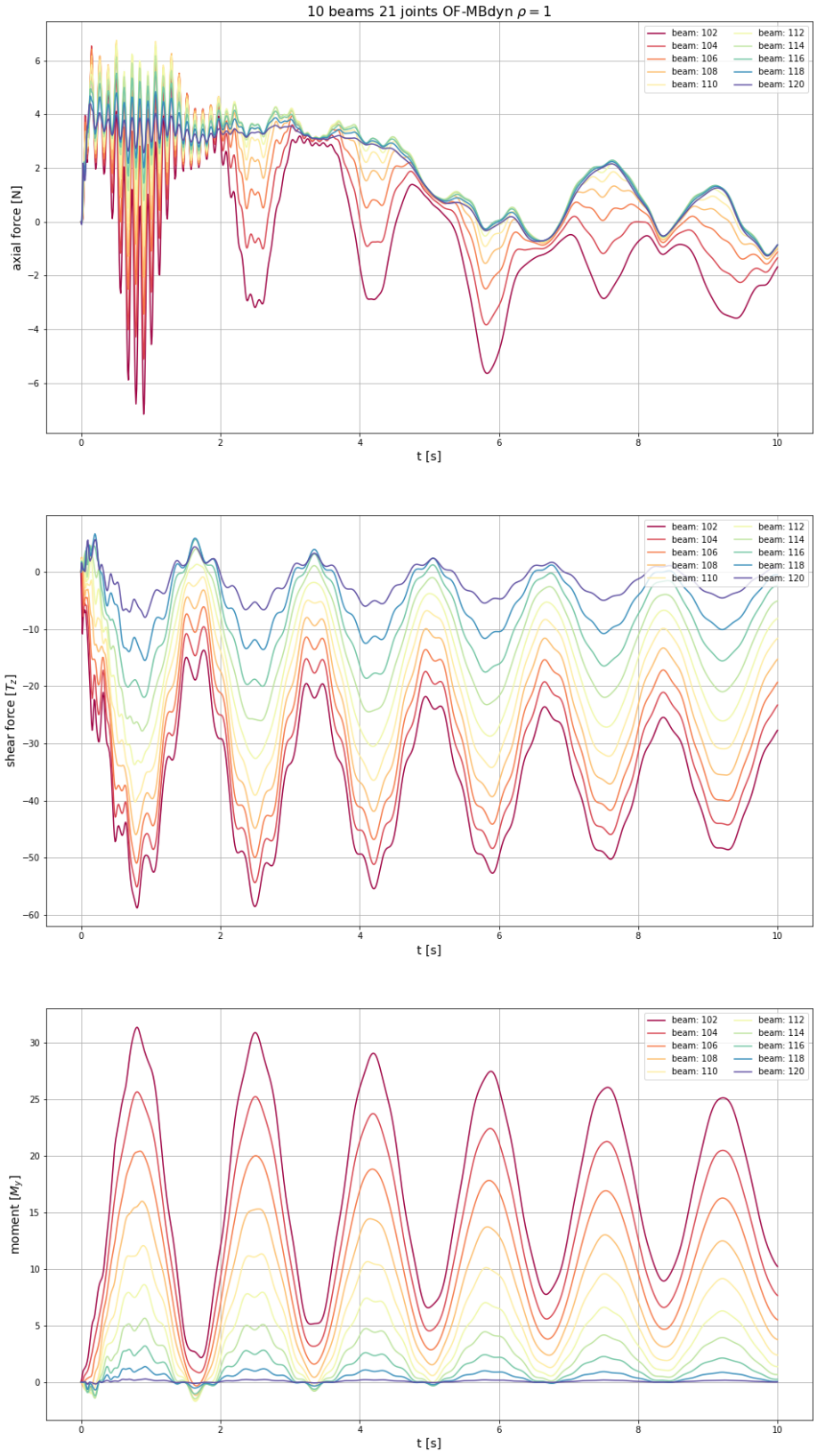


Figure 6.13. vertical flap: MBDyn beam internal forces

6.4 Vertical flap: compressible regime

A model similar to the one described in the previous section has been used to test the coupling capabilities of the adapter with a different fluid solver: SU2[3]. The current coupling adapter between SU2 and preCICE allows to build FSI simulations only in compressible regime. Besides, SU2 allows to define bidimensional domains, while OpenFOAM and MBDyn are strictly tridimensional. Bidimensionality is enforced in OenFOAM using *empty* boundary conditions, while in MBDyn is enforced through constraints on nodes. For those reasons a similar model has been built.

6.4.1 Fluid domain

The fluid domain is represented in Figure 6.14. The inlet is on the left with uniform flow velocity, the outlet is on the right, while all other boundaries are *slip* walls.

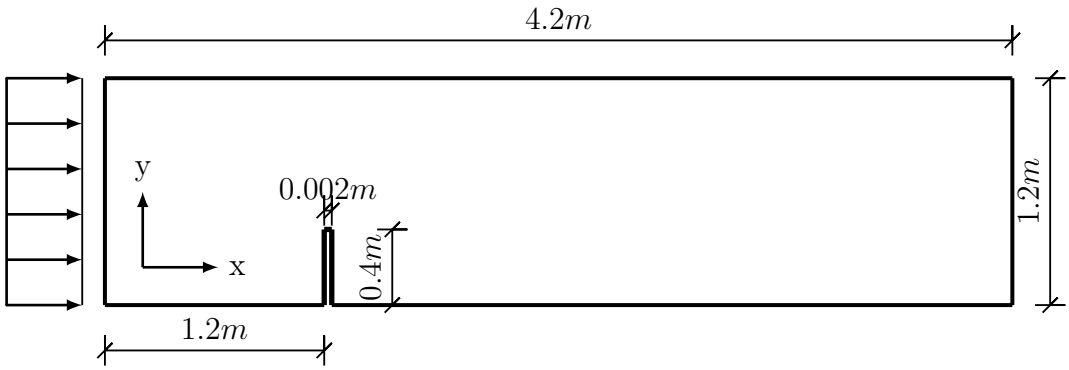


Figure 6.14. vertical flap (compressible): fluid domain

The fluid domain is discretized in an unstructured triangular mesh as depicted in Figure 6.15. The main fluid and mesh values are given in Table 6.6 and 6.7.

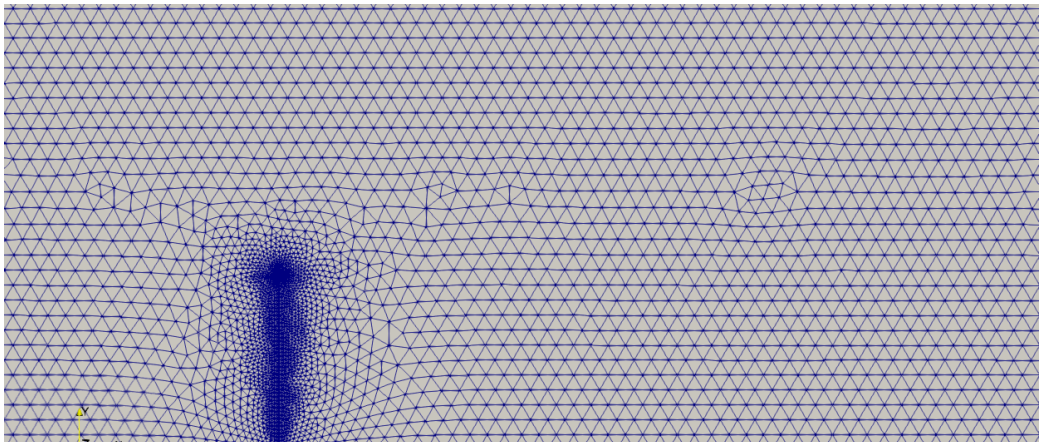


Figure 6.15. vertical flap (compressible): fluid mesh

[3] su2code.github.io

parameter			value
fluid properties			standard air
outlet pressure	p	[Pa]	101300
inlet temperature	T	[K]	288
inlet Mach number	Ma		0.1
flow type			euler

Table 6.6. Vertical flap (compressible): fluid properties

parameter		value
number of mesh points	n_{dof}	5580
number of cells	n_c	10709
number of interface points	n_{int}	162

Table 6.7. Vertical flap (compressible): mesh properties

6.4.2 MBDyn model

The MBDyn model uses solid properties defined in Table 6.8 and is composed of 5 `beam3` elements.

parameter			value
solid density	ρ	[kg m^{-3}]	1000
Elastic modulus	E	[Pa]	$5.6 \cdot 10^9$
Poisson coefficient	ν		0.4
structural damping			$1 \cdot 10^{-3}$

Table 6.8. Vertical flap: solid properties

6.4.3 Coupling parameters

The main data concerning the coupling between MBDyn and SU2 are given in Table 6.9:

parameter			value
simulation time	t	[s]	1
step size	Δt	[s]	10^{-3}
coupling scheme			serial implicit $S \rightarrow F$
coupling algorithm			IQN-ILS
displacement rel. convergence limit			10^{-5}
force rel. convergence limit			10^{-3}
interface mesh mapping			RBF

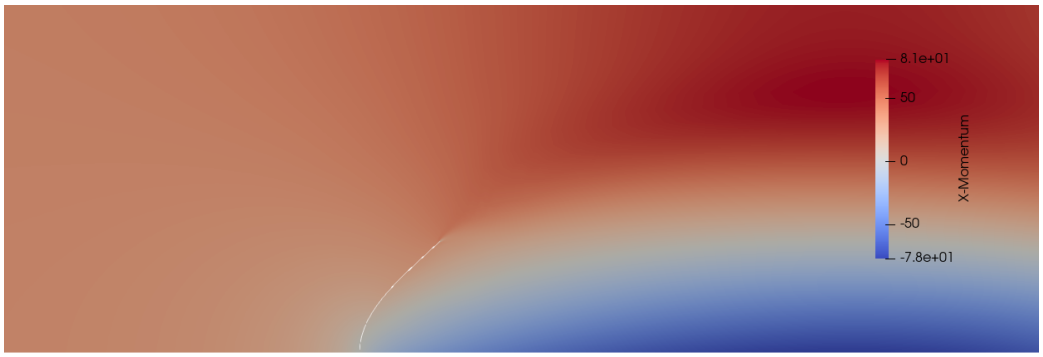
Table 6.9. Vertical flap (compressible): coupling parameters

6.4.4 Results

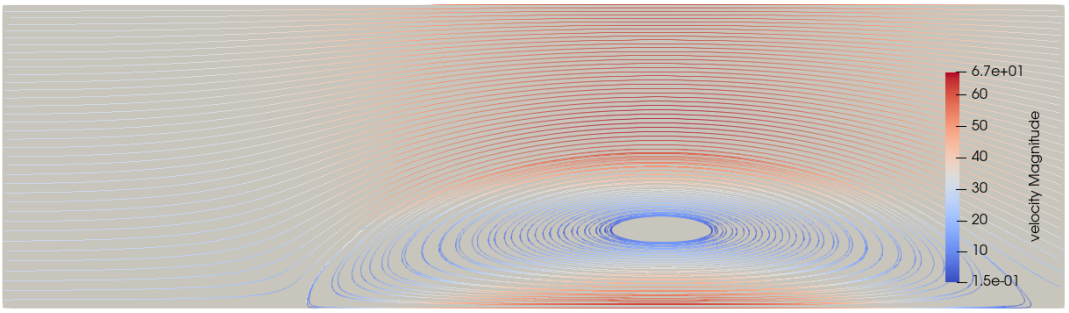
The problem considered is characterized by the dimensionless parameters given in Table 6.10: in particular the mass number remains in the order of $\mathcal{O}\left(10^{-3}\right)$. A sketch of the flow solution is represented in Figure 6.16.

parameter		value
mass number	M	$\approx 1.2 \cdot 10^{-3}$
reduced velocity	U_R	$\approx 1.44 \cdot 10^{-2}$
Cauchy number	C_Y	$\approx 2.53 \cdot 10^{-7}$

Table 6.10. Vertical flap (compressible): dimensionless numbers



(a) vertical flap (compressible): x-momentum



(b) vertical flap (compressible): streamlines

Figure 6.16. vertical flap (compressible): flow solution

The combination of parameters considered in this test case aimed at considering a very thin flexible element surrounded by a low-Mach flow, so that large displacements are involved. The structure reaches a steady deformed shape after around 1 second as shown in Figure 6.18, concerning tip displacement.

The fluid flow has been initialized with a steady solution obtained considering a rigid structure, then the FSI problem has been simulated progressively applying the aerodynamic load to the flap: a linear ramp starting at 1% of the load with a duration of 0.2 seconds (see Section 5.3 for the adapter input parameters).

Resultant forces (R_x, R_y) and moment (M_z) computed at the root of the flap are shown in Figure 6.17.

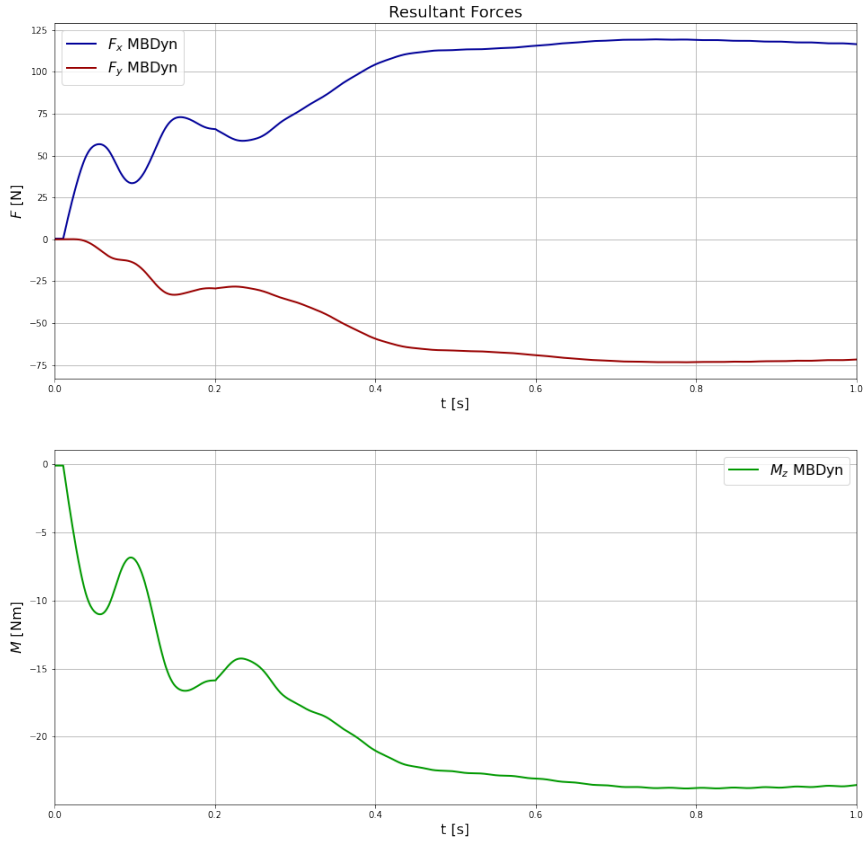


Figure 6.17. vertical flap (compressible): resultant forces

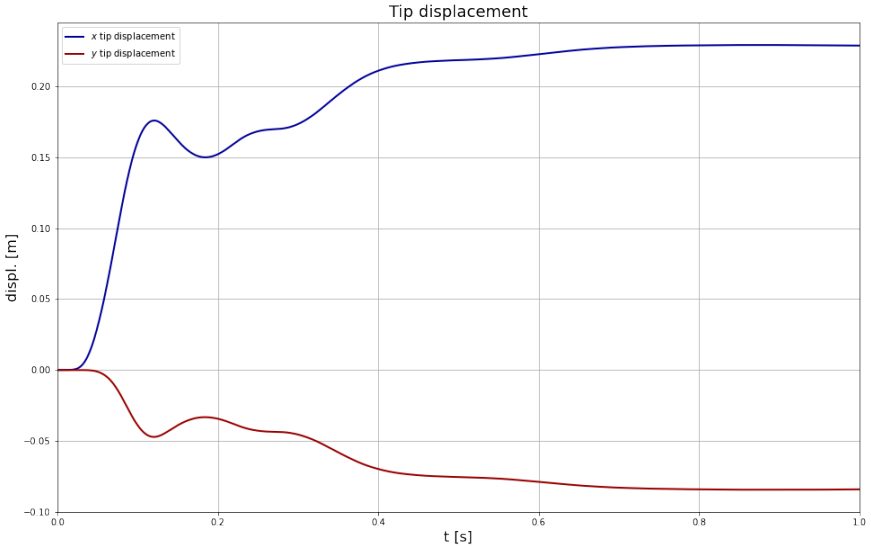


Figure 6.18. vertical flap (compressible): tip displacement

Each time step of the problem converges quite rapidly, requiring 5 iterations in most cases (see Figure 6.19). This outcome agrees with the fact that a low mass number and a high fluid velocity make the problem loosely coupled (see section 3.3 for some details). Nevertheless an explicit simulation would diverge immediately.

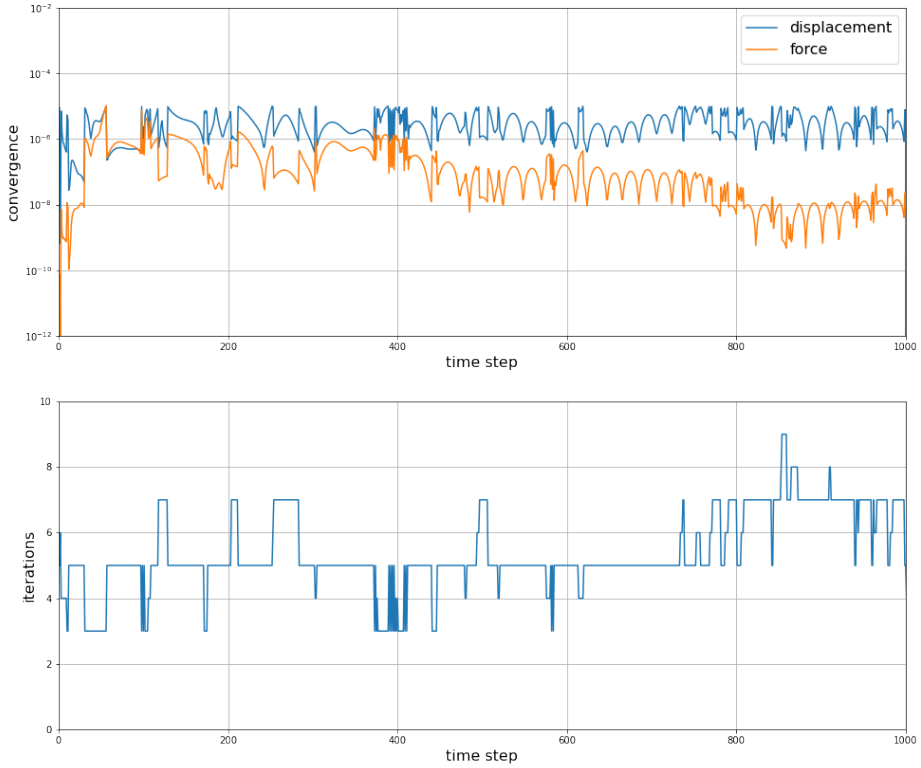


Figure 6.19. vertical flap (compressible): convergence and iterations

Axial and shear forces and bending moment in each of the MBDyn elements are plotted in Figure 6.20.

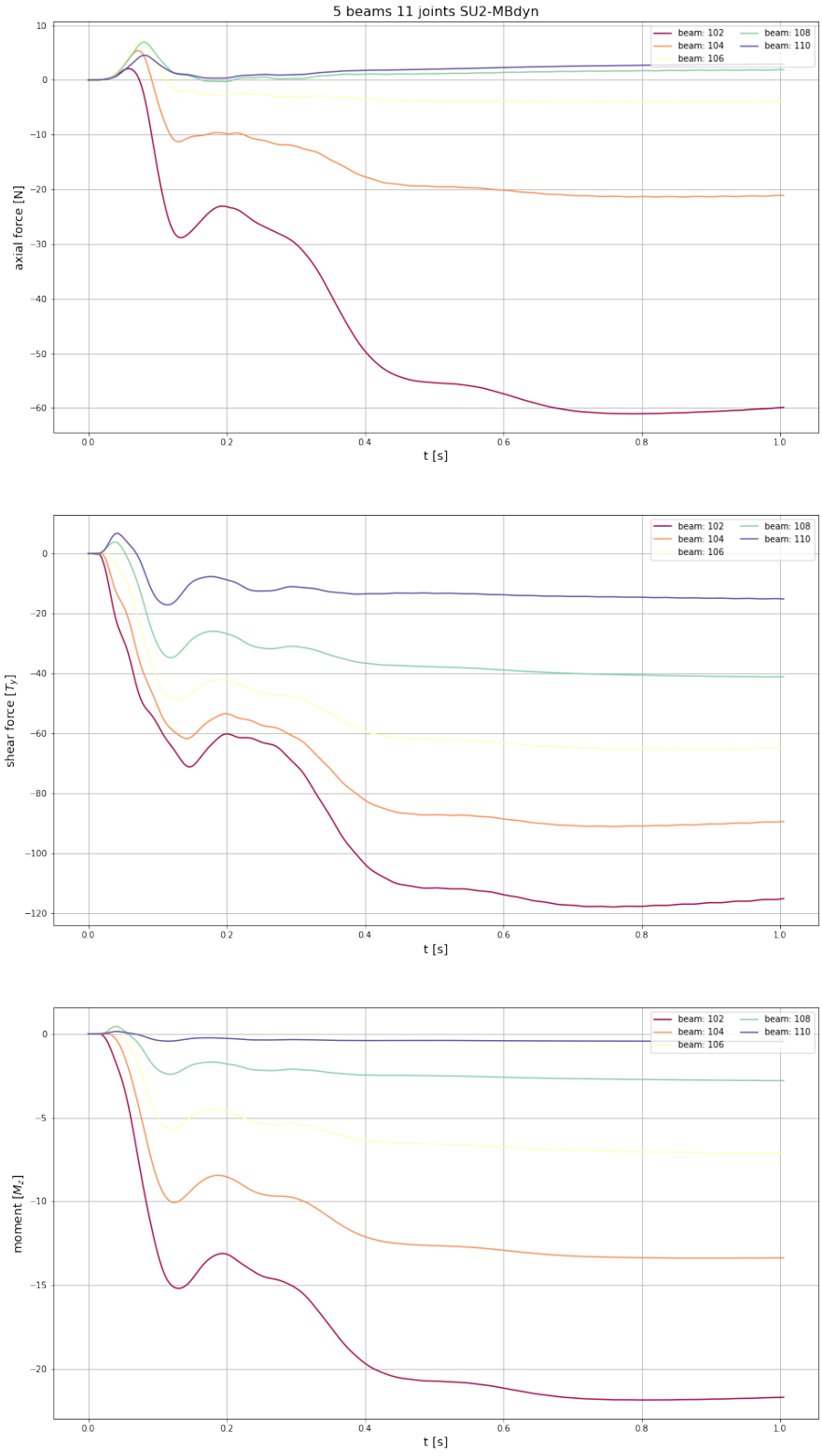


Figure 6.20. vertical flap (compressible): MBDyn internal forces

6.5 Square bluff body Benchmark

The first simple problems aimed at verifying that the complete FSI problem works and gives correct results: i.e. the MBDyn model, the mapping with the interface mesh, the coupling adapter and the fluid model (i.e. fluid flow and mesh movement performed with different solvers).

After that, it is important to verify the performance of the system considering well-known benchmarks, so that the results can be compared with a large number of algorithms and techniques.

6.5.1 Problem description

The first benchmark considered here has been described in [60]. The structural part is composed of a square bluff body with a trailing thin flap, as shown in Figure 6.21. The geometrical data are given in Table 6.11.

parameter		value
H	[mm]	10
L	[mm]	40
t	[mm]	0.6

Table 6.11. square bluff body: geometry

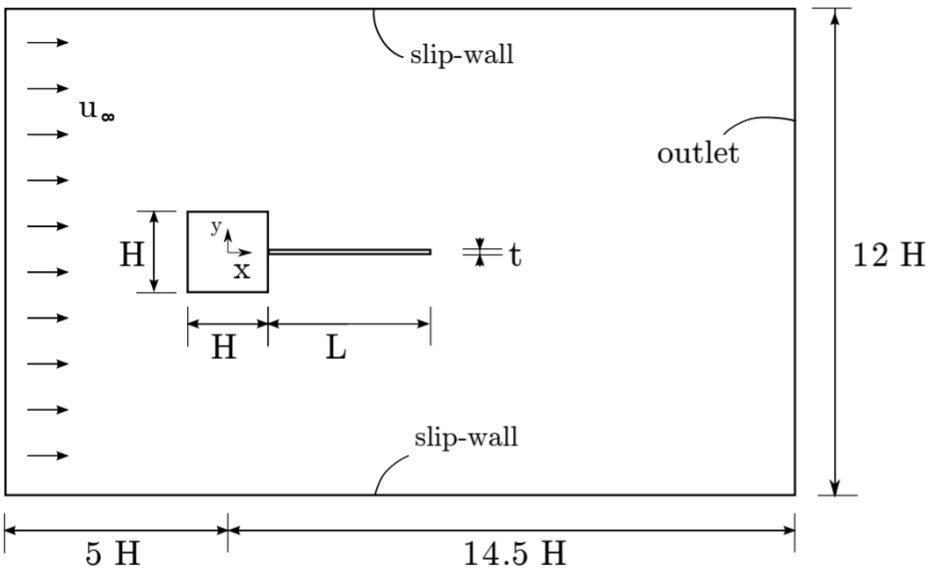


Figure 6.21. square bluff body benchmark: domain

6.5.2 Fluid domain

The fluid domain is represented in Figure 6.21 and has been simulated in OpenFOAM. The inlet is on the left with uniform flow velocity, the outlet is on the right, the external boundaries (top and bottom) are *slip-walls* while all other boundaries (body and flap) are *no-slip* walls.

The fluid domain is discretized with an structured hexaedral mesh as depicted in Figure 6.22. The main fluid and mesh values are given in Table 6.12 and 6.13.

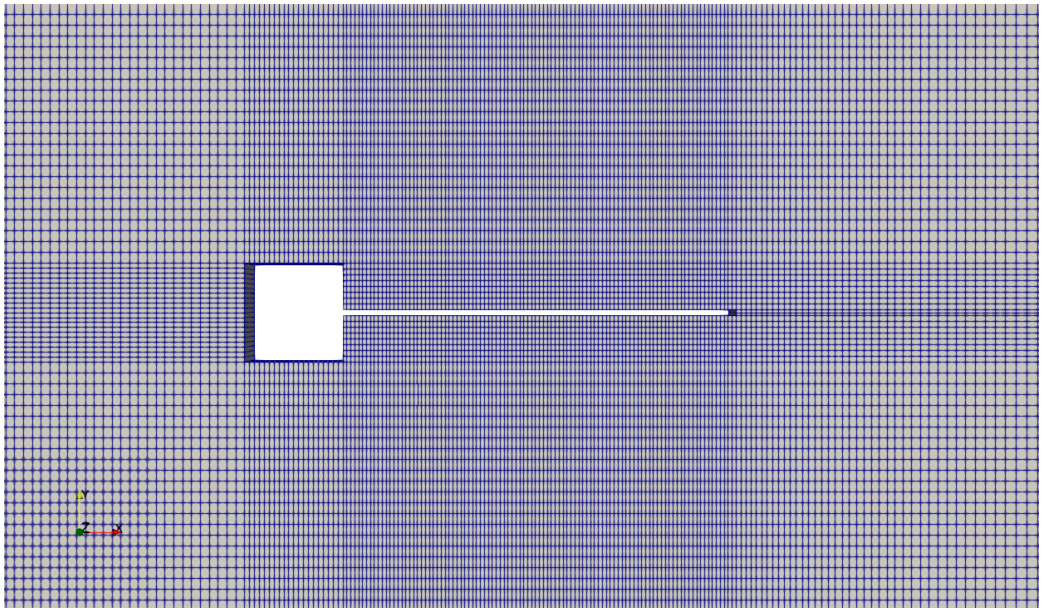


Figure 6.22. square bluff body: fluid mesh

parameter			value
fluid density	ρ	$[\mathrm{kg\,m^{-3}}]$	1.18
dynamic viscosity	μ	$[\mathrm{kg\,m^{-1}\,s^{-1}}]$	$1.82 \cdot 10^{-5}$
Reynolds number	Re		332
flow velocity	$\vec{u}_\infty$	$[\mathrm{m\,s^{-1}}]$	0.513
flow type			laminar

Table 6.12. square bluff body: fluid properties

parameter		value
number of mesh points	n_{dof}	59096
number of cells	n_c	29040
number of interface cells	n_{int}	202

Table 6.13. square bluff body: mesh properties

6.5.3 Solid domain

The properties of the solid are reported in Table 6.14. The MBDyn model is composed of 10 `beam3` elements. The setup resembles the one briefly described in Section 6.2. The structural damping of the beam elements (see Section 4.1.4) has been set to be proportional to the stiffness matrix of the element with a coefficient of $1 \cdot 10^{-2}$. This value appeared to be high enough to give a smooth structural solution without being excessively damping.

The interface mesh is divided into 202 faces and it is shown in Figure 6.23.

parameter			value
solid density	ρ	$[\mathrm{kg\,m^{-3}}]$	100
Elastic modulus	E	[Pa]	$2.5 \cdot 10^5$
Poisson coefficient	ν		0.35

Table 6.14. square bluff body: solid properties

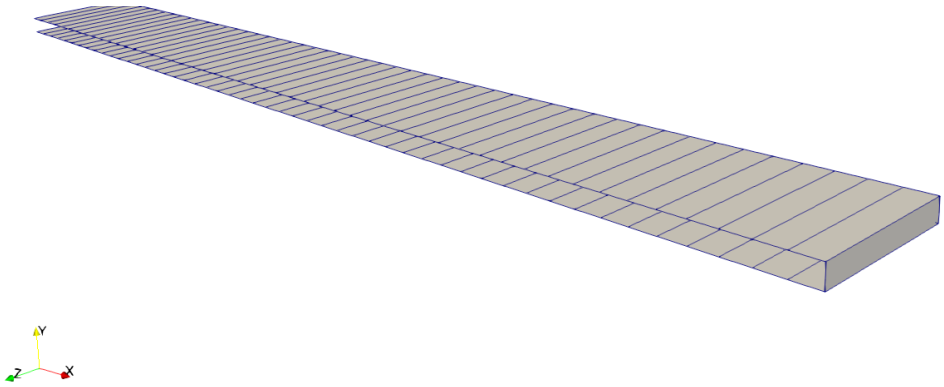


Figure 6.23. square bluff body: structural interface mesh

6.5.4 Coupling parameters

The main coupling data are given in Table 6.15. The particularly short time step is required by fluid mesh movement: as it will be apparent in the results, the thin flexible flap moves quickly and with large displacements. The diffusion algorithm used by OpenFOAM to deform the fluid mesh produced inconsistent cell volumes when using higher time steps.

parameter			value
simulation time	t	[s]	6
step size	Δt	[s]	$5 \cdot 10^{-4}$
coupling scheme			serial implicit $S \rightarrow F$
coupling algorithm			IQN-ILS
displacement rel. convergence limit			10^{-4}
force rel. convergence limit			$2 \cdot 10^{-4}$
interface mesh mapping			RBF

Table 6.15. square bluff body: coupling parameters

Most parameters are similar to the ones used in other examples as they turned out to fit well with most of the experiments.

6.5.5 Results

The problem presented here is characterized by the dimensionless parameters given in Table 6.16 and its solution is represented in Figure 6.26.

As in other simulations, fluid forces are applied with a ramp to ease convergence at the beginning of the simulation: during the first 100 ms they are scaled to 10%, to reach 100%

parameter		value
mass number	M	$1.18 \cdot 10^{-2}$
reduced velocity	U_R	$\approx 1 \cdot 10^{-2}$
Cauchy number	C_Y	$1.24 \cdot 10^{-6}$

Table 6.16. square bluff body: dimensionless numbers

after another 100 ms.

Alternating vortices begin developing quite rapidly and the structure starts oscillating. After around 2 seconds it reaches a vortex lock-in regime (e.g. [38]), as shown in Figure 6.24 (in particular the tip displacement in y direction).

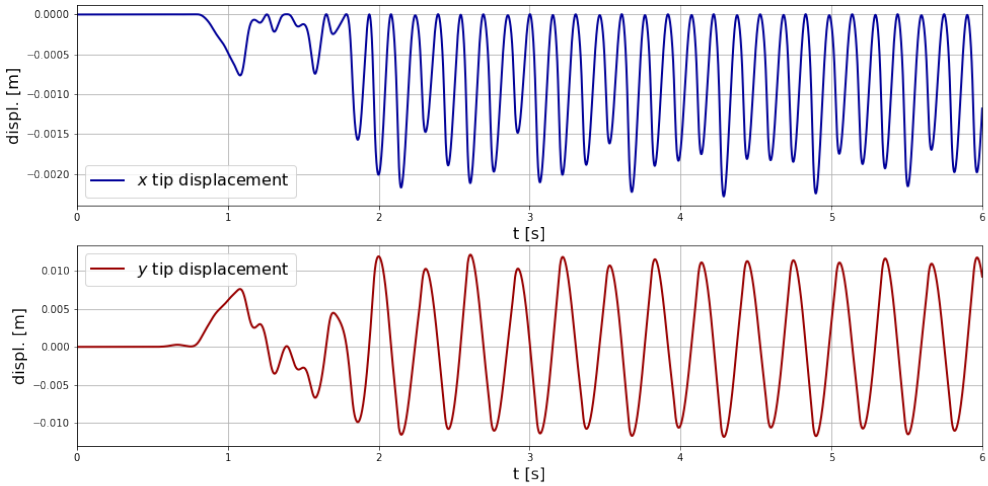


Figure 6.24. square bluff body: tip displacement

Forces are measured on the whole structure on the fluid side, while they are measured only on the flap in the solid domain. Figure 6.25 shows a detail of 0.5s of simulation. The difference in x-direction represents the drag force on the square body. Differences in lift and moment are almost zero.

Each time step converges with an average of 8 iterations, except for some conditions in which the coupling algorithm fails in making coupling forces converge. Those situations show a quite distorted fluid mesh that make the overall FSI problem harder to solve.

An higher average number of iterations agrees with the fact that a higher mass number makes the problem strongly coupled. Here M is in the order of $\mathcal{O}\left(10^{-2}\right)$.

As in the previous examples, the axial and shear forces and bending moment in each of the MBDyn elements are plotted in Figure 6.28: in this case a temporal slice of 0.5s has been considered.

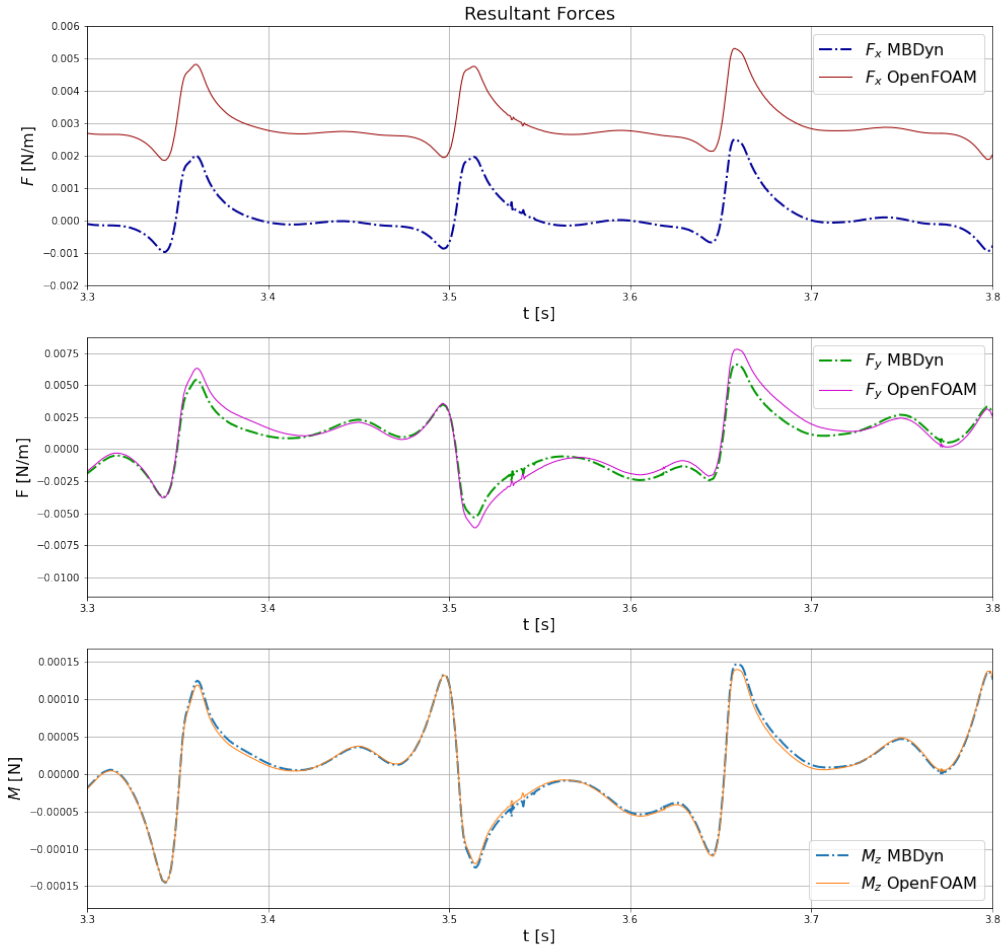
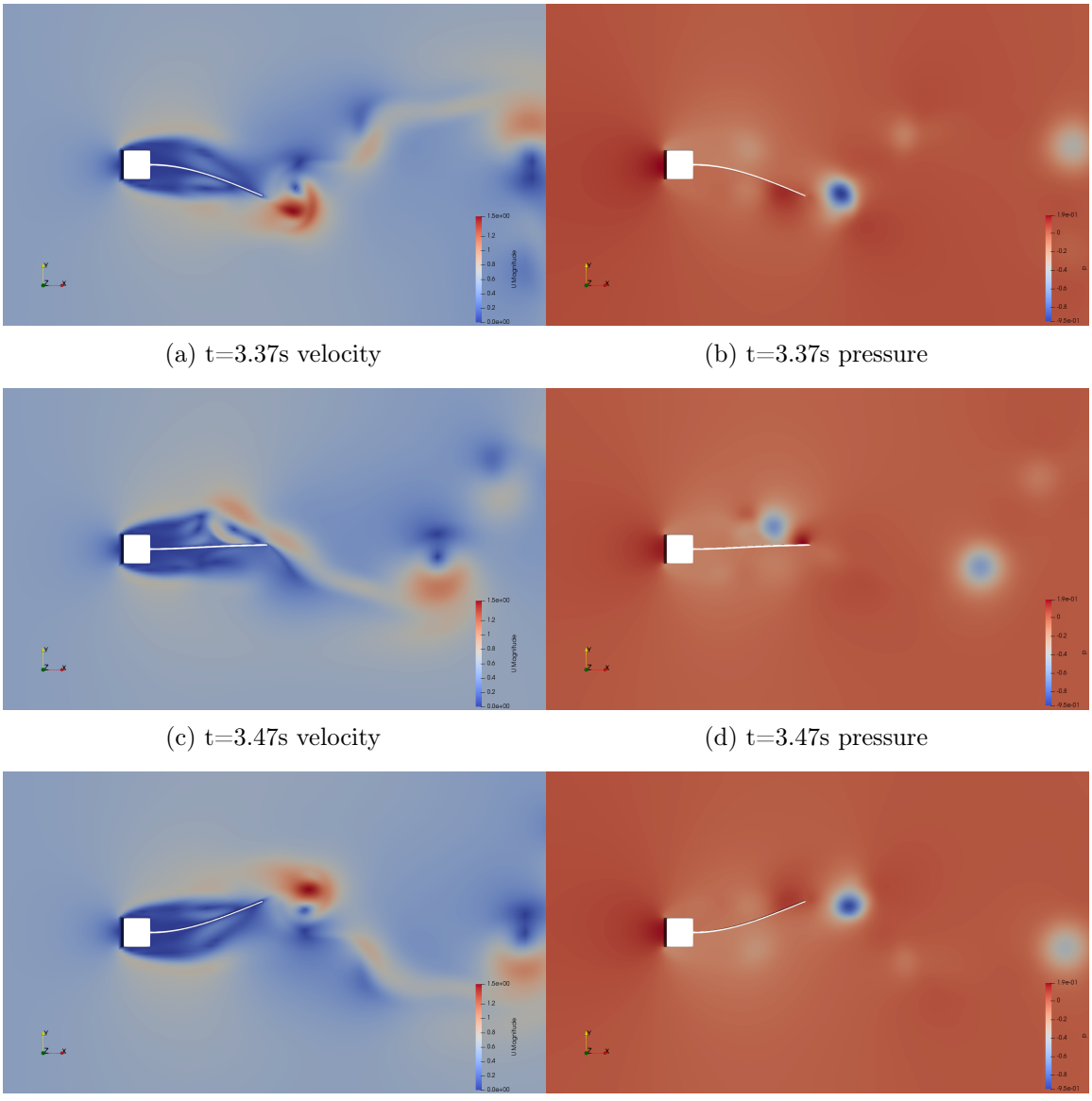


Figure 6.25. square bluff body: resultant forces (detail)



(a) t=3.37s velocity (b) t=3.37s pressure

(c) t=3.47s velocity (d) t=3.47s pressure

(e) t=3.53s velocity (f) t=3.53s pressure

Figure 6.26. square bluff body: fluid solution

6.5.6 Validation

This problem has been used as a benchmark in a great number of studies in literature. The comparison between the simulations is generally made on the tip displacement, in terms of amplitude and frequency of oscillation. In Table 6.17 some results are given and compared to the present study.

The comparison shows a very good agreement among the studies: considering all the previous simulations, the average tip displacement is 11.2 mm with a standard deviation of 1.06 mm, while the average frequency of oscillation is 3.1 Hz with a standard deviation of 0.09 Hz.

This study, with a tip displacement of 11.2 mm at a frequency of 3.067 Hz, shows that

[4]Fluid Structure Interaction Problems using SU2. First SU2 Annual Developers Meeting. TU Delft, 6 September 2016

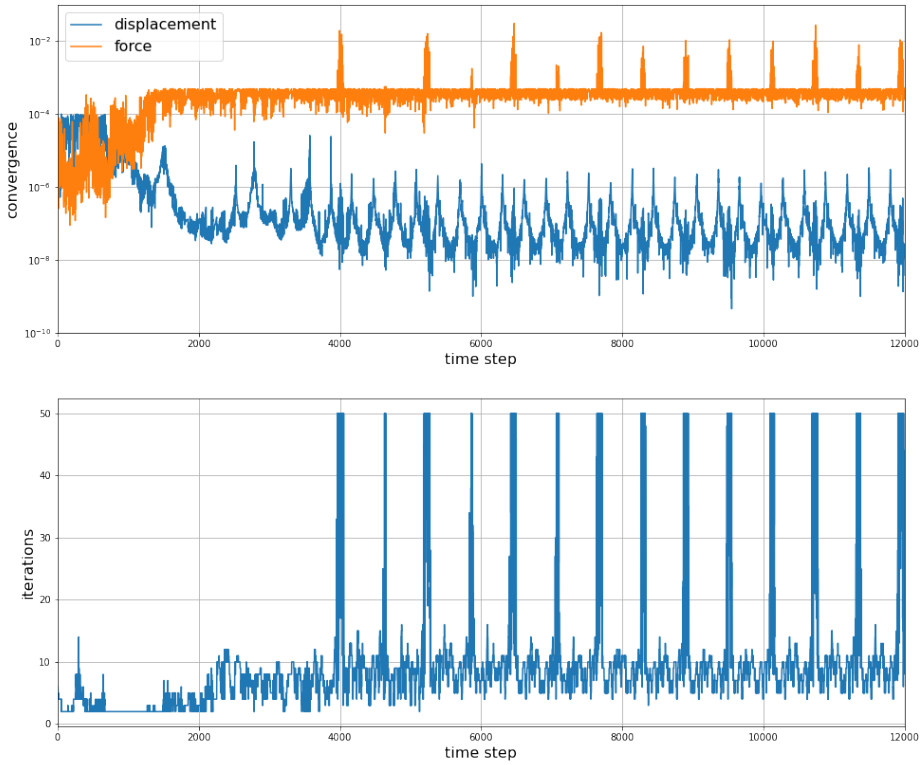


Figure 6.27. square bluff body: convergence and iterations

Study	f [s^{-1}]	$d_{y\,tip}$ [mm]
Wall and Ramm [60]	3.08	13.1
Kassiotis et al. [43]	2.98	10.5
Wood et al. [74]	2.94	11.5
Olivier et al. [55]	3.17	9.5
Walhorn et al. [73]	3.14	10.2
Matthies and Steindorf [52]	3.13	11.8
Dettmer and Peric [16]	3.03	12.5
Habchi et al. [30]	3.25	10.2
Froehle and Persson [21]	3.18	11.2
Sanchez et al.[4]	3.15	11.5
Present study	3.067	11.2

Table 6.17. square bluff body: results

the implementation of the adapter can give good results.

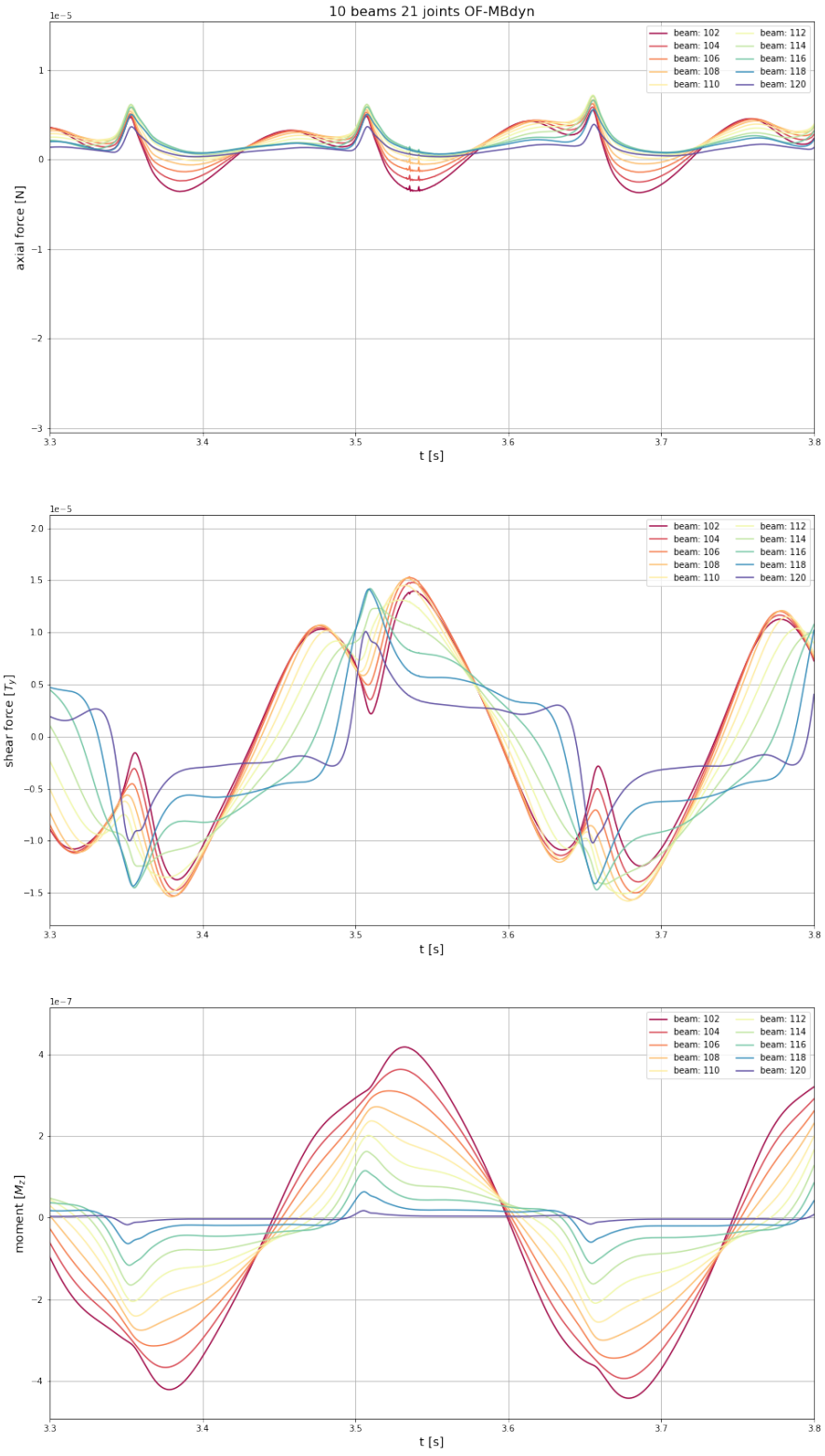


Figure 6.28. square bluff body: MBDyn internal forces (detail)

6.6 Turek-Hron FSI2 Benchmark

Another well known benchmark, concerning FSI simulations performed in incompressible flow regime, is proposed by Turek and Hron [66]. A very important thing to notice about this study is that it is a "Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow". So the paper gives a specific problem setup to which others can contribute.

The fluid domain is similar to the one described in the previous section. The domain is again composed of a bluff body (a cylinder) and a flap, but different flow velocities and structural properties are considered, so that 3 cases arise, commonly known as FSI1, FSI2 and FSI3.

For reasons that will be apparent in the next section, FSI2 is considered first.

6.6.1 Problem Description

The structural part is composed of a round cylinder with a trailing thin flap, as described in Figure 6.29. The geometrical data are given in Table 6.18.

parameter		value
H	[m]	0.41
L	[m]	2.5
l	[m]	0.35
h	[m]	0.02
C	[m]	$(0.2, 0.2)$
r	[m]	0.05

Table 6.18. FSI2: geometry

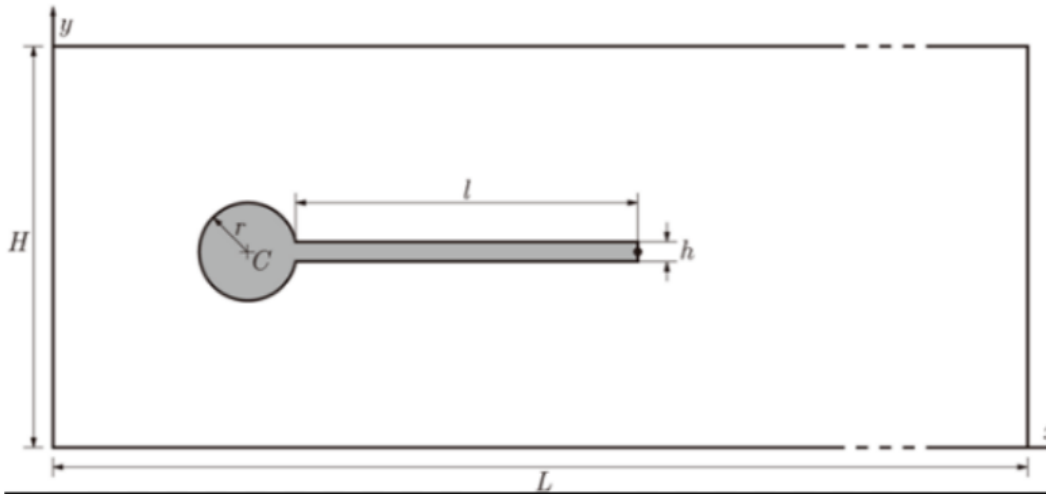


Figure 6.29. FSI2: domain

6.6.2 Fluid domain

The fluid domain is represented in Figure 6.29 and has been simulated in OpenFOAM. The inlet is on the left with *parabolic* flow velocity, the outlet is on the right, while all other boundaries are *no-slip* walls.

The fluid domain is discretized in a structured hexaedral mesh as depicted in Figure 6.30. The main fluid and mesh values are given in Table 6.19 and 6.20.

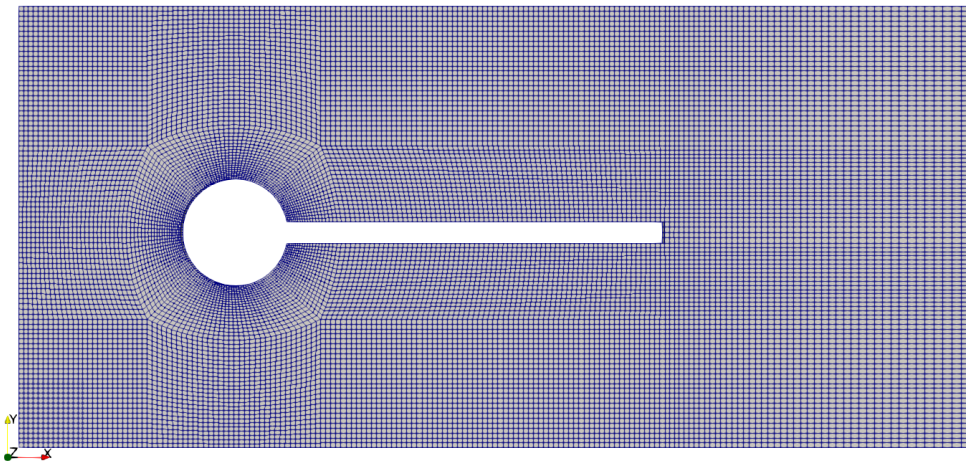


Figure 6.30. FSI2: fluid mesh

parameter			value
fluid density	ρ	$[\mathrm{kg\,m^{-3}}]$	1000
kinematic viscosity	ν	$[\mathrm{m^2\,s^{-1}}]$	$1 \cdot 10^{-3}$
Reynolds number	Re		100
max flow velocity	$\vec{u}_{max}$	$[\mathrm{m\,s^{-1}}]$	1.5
mean flow velocity	$\vec{u}$	$[\mathrm{m\,s^{-1}}]$	1
flow type			laminar

Table 6.19. FSI2: fluid properties

parameter		value
number of mesh points	n_{dof}	51464
number of cells	n_c	25224
number of interface cells	n_{int}	180

Table 6.20. FSI2: mesh properties

6.6.3 Solid domain

The properties of the solid are reported in Table 6.21.

parameter			value
solid density	ρ	[kg m^{-3}]	10000
Elastic modulus	E	[Pa]	$1.4 \cdot 10^6$
Poisson coefficient	ν		0.4

Table 6.21. FSI2: solid properties

The MBDyn model is the same as the one used in 6.5 and is composed of 10 `beam3` elements, with a structural damping of $1 \cdot 10^{-2}$.

The interface mesh is divided into 90 faces and is shown in Figure 6.31: it has been built so that there is a solid interface cell every two fluid interface cells.

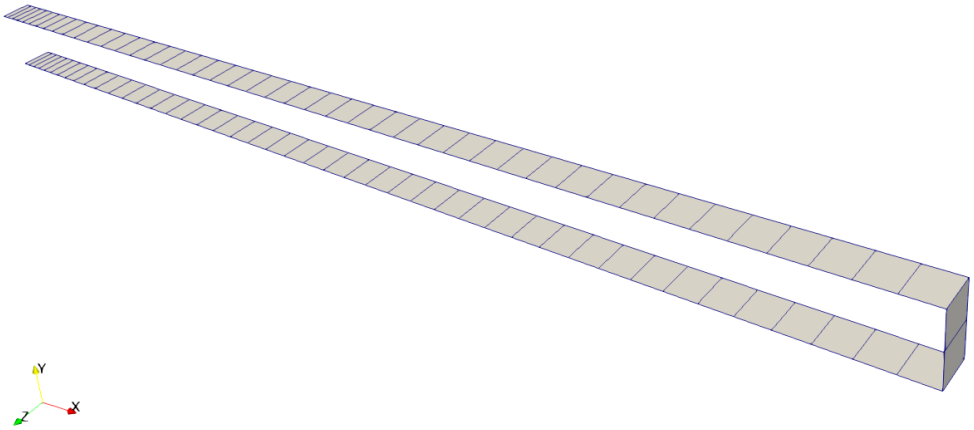


Figure 6.31. FSI2: structural interface mesh

6.6.4 Coupling parameters

The main coupling data are given in Table 6.22. In this case a time step of 1 ms is enough for the simulation.

parameter			value
simulation time	t	[s]	15
step size	Δt	[s]	$1 \cdot 10^{-3}$
coupling scheme			serial implicit $S \rightarrow F$
coupling algorithm			IQN-ILS
displacement rel. convergence limit			10^{-4}
force rel. convergence limit			$2 \cdot 10^{-4}$
interface mesh mapping			RBF

Table 6.22. FSI2: coupling parameters

6.6.5 Results

The problem presented here is characterized by the dimensionless parameters given in Table 6.23 and its solution is represented in Figure 6.34.

parameter		value
mass number	M	0.1
reduced velocity	U_R	$8.45 \cdot 10^{-2}$
Cauchy number	C_Y	$7.14 \cdot 10^{-4}$

Table 6.23. FSI2: dimensionless numbers

In this case, the ramp applied to fluid forces at the beginning of the simulation is longer, in order to make the fluid flow and the structure settle: during the first 500 ms forces are scaled to 10% and reach 100% after another 500 ms.

Alternating vortices begin developing and the flap begins oscillating with an increasing amplitude. After around 4 seconds the structure reaches a vortex lock-in regime, as shown in Figure 6.32.

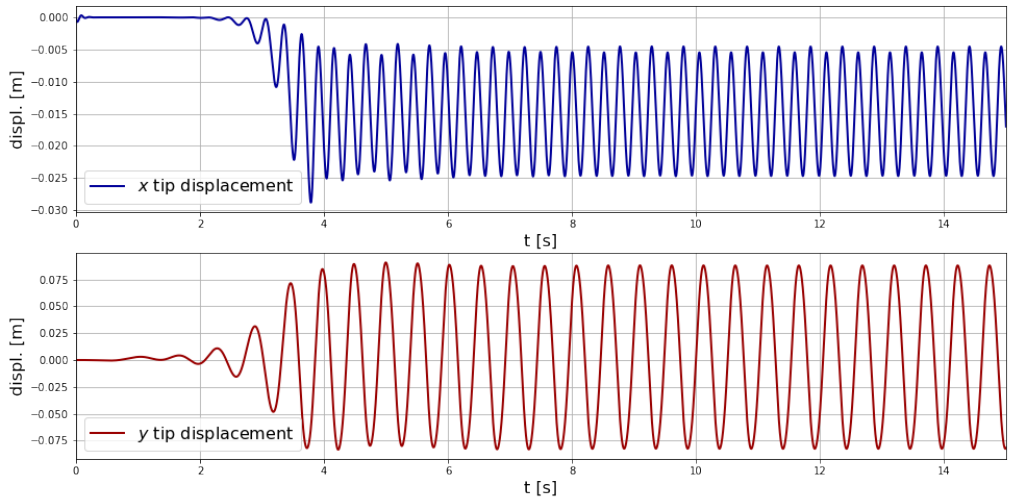


Figure 6.32. FSI2: tip displacement

Forces are measured on the whole structure on the fluid side of the simulation, while they are measured on the flap alone in the solid domain. Figure 6.33 shows a detail of 2 s of simulation. The difference in x-direction represents the drag force on the cylinder.

Each time step converges with an average of 14.5 iterations, which again confirms the trend of more coupling iterations as the mass number increases.

As in the previous examples, the axial and shear forces and bending moment in each of the MBDyn elements are plotted in Figure 6.28: in this case a temporal slice of 2 s has been considered. The values plotted here consider a beam with 2 mm width: i.e. the thickness of the solid and fluid domains considered in this case.

6.6.6 Validation

Turek and Hron benchmarks have been used as a reference in many studies considering strongly coupled FSI simulations.

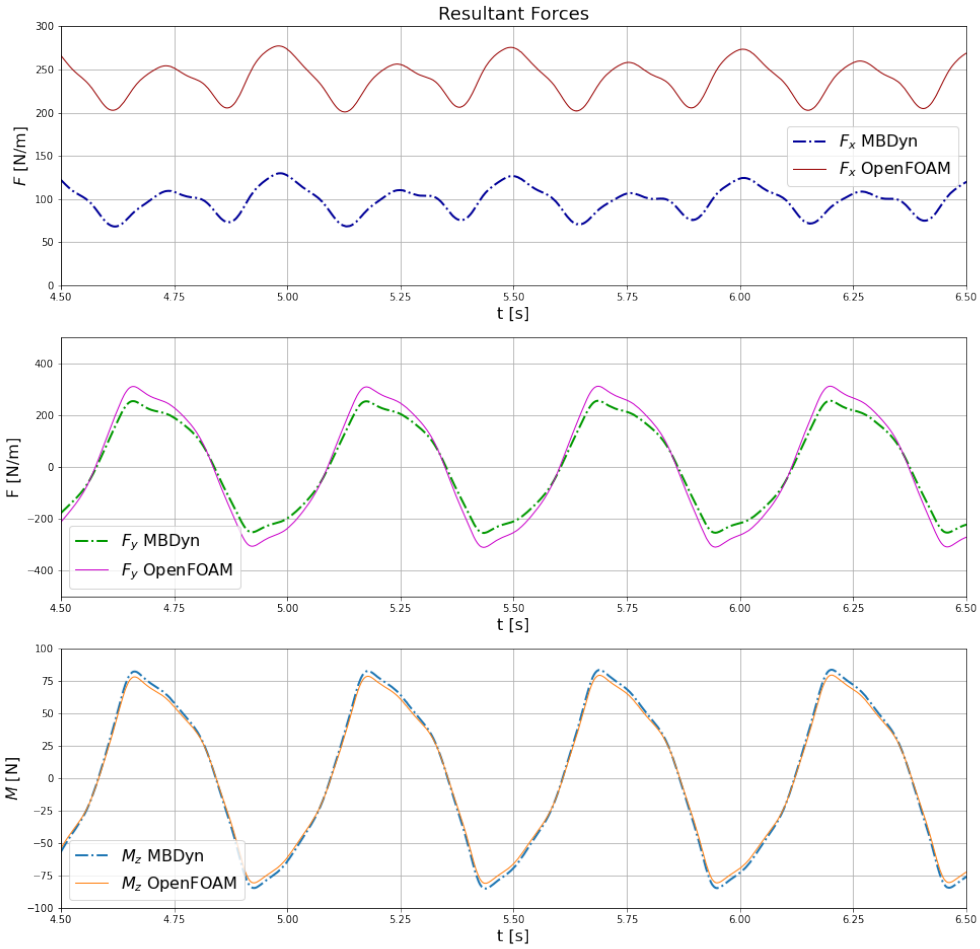
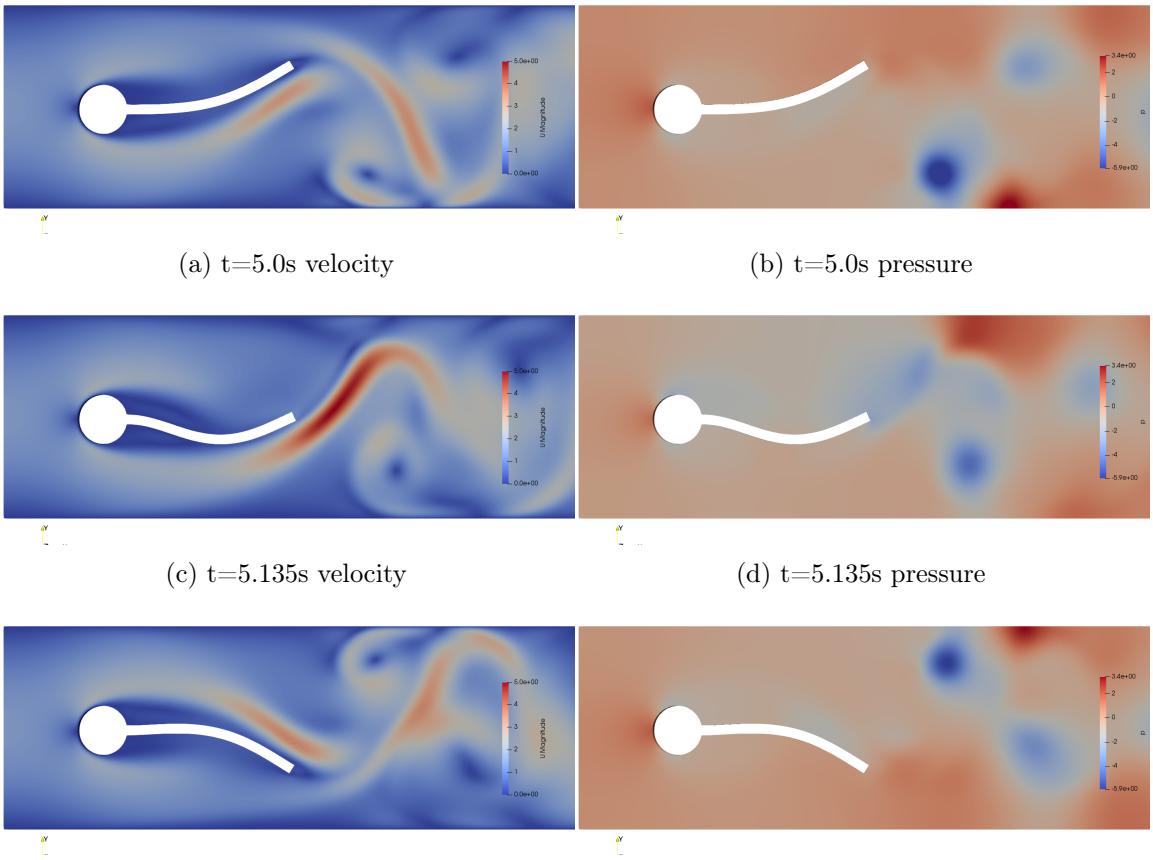


Figure 6.33. FSI2: resultant forces (detail)



(a) t=5.0s velocity

(b) t=5.0s pressure

(c) t=5.135s velocity

(d) t=5.135s pressure

(e) t=5.27s velocity

(f) t=5.27s pressure

Figure 6.34. FSI2: fluid solution

FSI2 is fully oscillating while the same problem, considering the structure rigid (named CFD2 in [66]) is steady: for this reason it is considered an excellent check for interaction mechanisms [67].

Besides, FSI2 gives the largest deformation and in some cases it is considered the most difficult of the three benchmarks [63], as it gives deformations four times greater than the flap height.

The comparison of the results can be done in terms of tip displacement and in terms of lift and drag force applied to the whole structure. The results given in the original paper and in a review made by the same authors, together with other studies found in literature, are compared to the present study in Table 6.24 for the data concerning the tip displacement, and in Table 6.25 for the forces applied to the structure.

The study in [67] proposes 7 different simulation methods and results for FSI1 and FSI3 test cases. In the numerical results it is stated that "clear differences between the different approaches with regard to accuracy are visible. Particularly for the drag and lift values, which lead to differences of up to order 50%, and also for the displacement values which are in the range of 10% errors".

For the FSI2 test case only the results from the initial Turek and Hron paper [66] and a few others are available.

Comparing the results of this simulation with previous studies (Table 6.24), the average displacement in x direction is very close to all other data (mainly due to mean pressure applied to the tip face), while mean oscillation is lower of about $3mm$. The displacement in

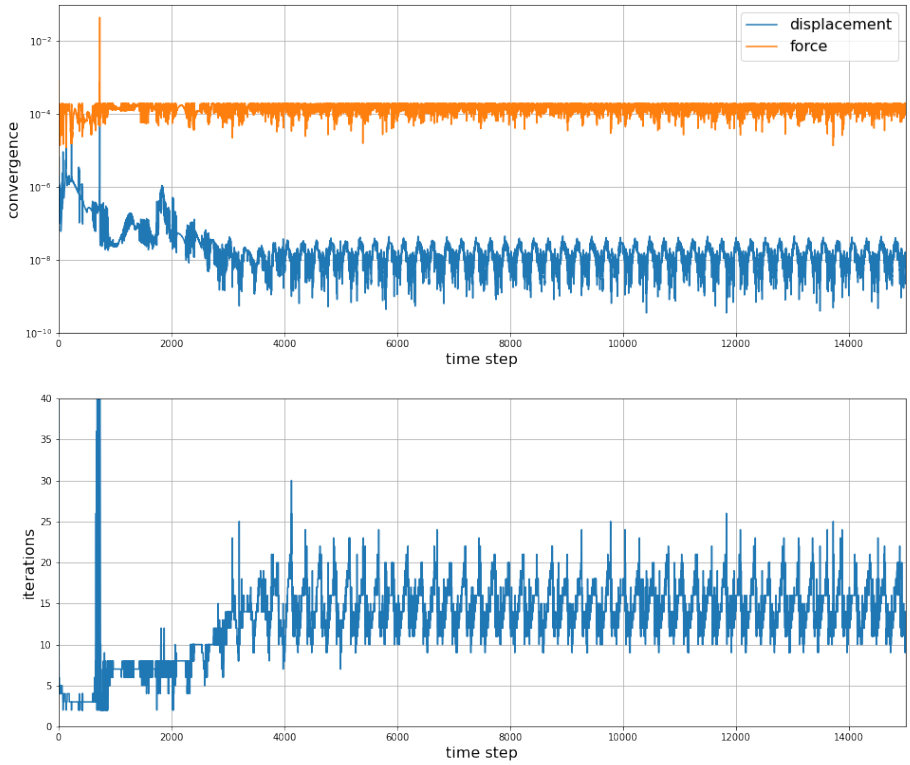


Figure 6.35. FSI2: convergence and iterations

Study	$d_{x\,tip}$ [mm]	f [Hz]	$d_{y\,tip}$ [mm]	f [Hz]
Benchmark [66]	-14.58 ± 12.44	3.8	1.23 ± 80.6	2.0
Turek et al. (2010) [67]	-14.85 ± 12.70	3.86	1.30 ± 81.7	1.93
Gjertsen [28]	-14.83 ± 13.11		1.24 ± 81.6	
Degroote [15]	-14.07 ± 12.37	3.7	1.18 ± 76.5	1.9
Present study	-14.95 ± 9.85	3.87	2.78 ± 83.39	1.93

Table 6.24. FSI2: comparison of results (displacements)

Study	drag [N m^{-1}]	f [Hz]	lift [N m^{-1}]	f [Hz]
Benchmark [66]	208.83 ± 73.75	3.8	0.88 ± 234.2	2.0
Turek et al. (2010) [67]	215.06 ± 77.63	3.86	0.61 ± 237.8	1.93
Gjertsen [28]	161.50 ± 73.75		0.88 ± 234.2	
Degroote [15]	217.52 ± 84.65	3.7	-0.74 ± 267.6	1.9
Present study	239.13 ± 31.93	3.87	3.43 ± 308.57	1.93

Table 6.25. FSI2: comparison of results (forces)

y direction is about 2 mm higher. It looks like the MBDyn structure is a bit more flexible (as also seen in Section 6.3 for the vertical flap). This also reflects the fact that average drag and lift are higher, as shown in Table 6.25.

The greatest difference can be seen in the oscillation of drag, which is much smaller than

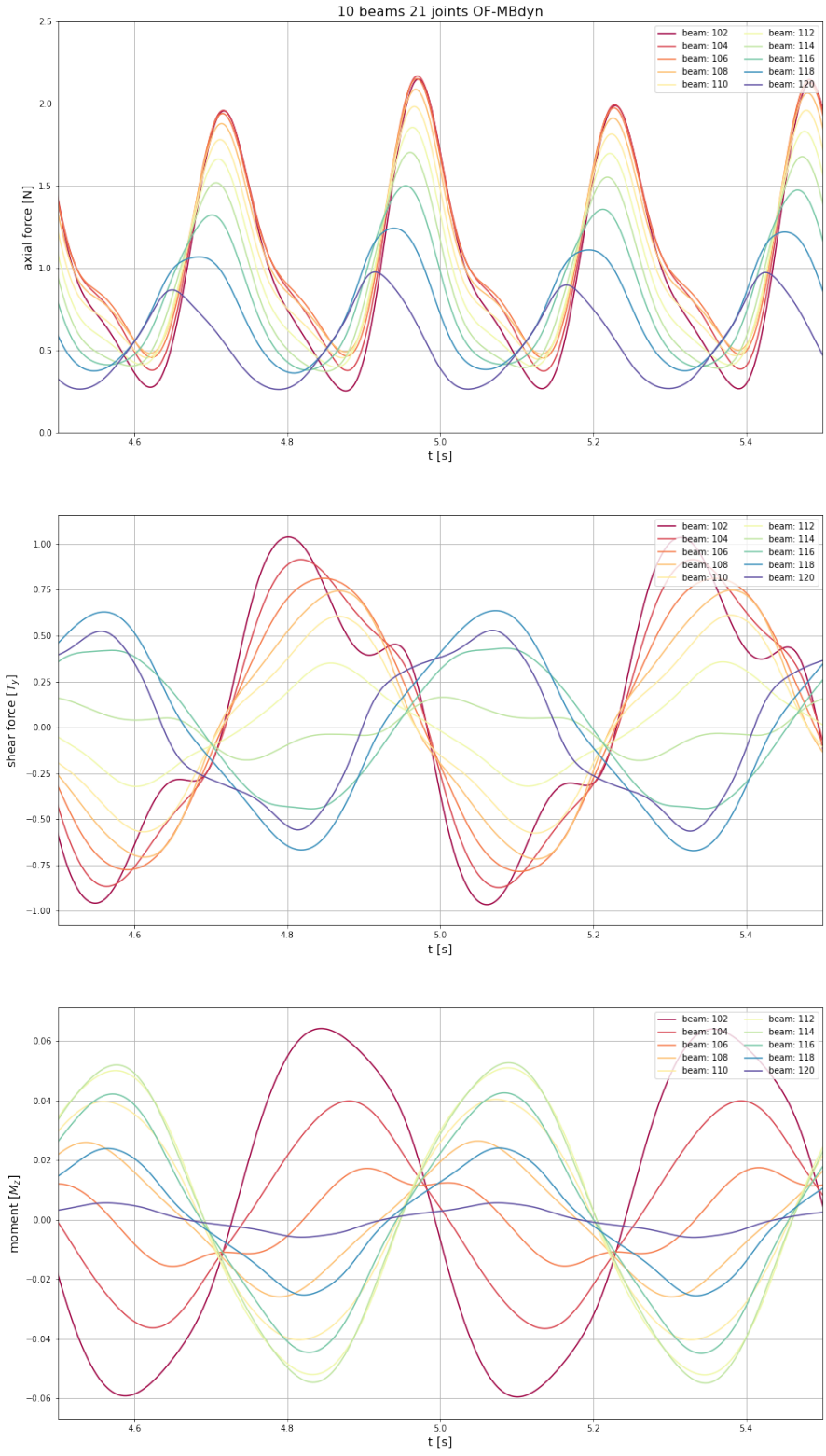


Figure 6.36. FSI2: MBDyn internal forces (detail)

other studies. The data collected during the simulation allow to separate the contribution of the cylinder from the one of the flap (see first graph of Figure 6.33). The cylinder has an average drag of 140 $\mathrm{N\,m^{-1}}$ with a standard deviation of ≈ 7 $\mathrm{N\,m^{-1}}$. Considering that the average C_D of a cylinder at $Re = 100$ is around 1.35 [57], and an average fluid velocity around the cylinder of about 1.4 $\mathrm{m\,s^{-1}}$, the contribution is as expected. The contribution of the flap seems to be much lower than the values given by other studies.

As previously stated, the results for the FSI3 case differ by in some cases 50% for drag and lift. With this in mind, we could expect similar behavior also in the FSI2 results.

6.7 Turek-Hron FSI3 Benchmark

Omitting for a while the test case named FSI1 as it is not oscillating, we considered the other benchmark proposed in [66], named FSI3. Its parameters are very similar to FSI2, with the only different properties shown in Table 6.26. The corresponding dimensionless numbers are given in Table 6.27.

parameter			FSI2	FSI3
solid density	ρ	$[\mathrm{kg\,m^{-3}}]$	10000	1000
Elastic modulus	E	[Pa]	$1.4 \cdot 10^6$	$5.6 \cdot 10^6$
max flow velocity	$\vec{u}_{max}$	$[\mathrm{m\,s^{-1}}]$	1.5	3
mean flow velocity	$\vec{u}$	$[\mathrm{m\,s^{-1}}]$	1	2

Table 6.26. FSI2-FSI3: different parameters

parameter		FSI2	FSI3
mass number	M	0.1	1
reduced velocity	U_R	$8.45 \cdot 10^{-2}$	$2.67 \cdot 10^{-2}$
Cauchy number	C_Y	$7.14 \cdot 10^{-4}$	$7.14 \cdot 10^{-4}$
Reynolds number	Re	100	200

Table 6.27. FSI2-FSI3: dimensionless numbers

The simulation of the FSI3 benchmark using MBDyn and OpenFOAM connected with preCICE proved to be an unfeasible task, at least at the time of writing. This same test case is known to work, giving correct results, coupling OpenFOAM and CalculiX as described in the preCICE website[5]. Nevertheless, substituting CalculiX with MBDyn, all other parameters being equal, makes the simulation diverge.

Different approaches have been experimented, acting on each part of the problem. For example:

- in the fluid domain:
  1. using more strict convergence criteria for the fluid solver
  2. using a linear or parabolic profile of the inlet velocity (tests on FSI2 showed very little difference)
  3. using a ramp also in the fluid velocity
  4. defining a finer (2× or 3×) fluid mesh
  5. considering a different thickness of the fluid and solid domain (which showed that having thicker domain produces a negative impact on the convergence)
- in the solid domain:
  1. considering more or fewer MBDyn nodes (from 5 to 20 nodes)
  2. changing solver (`naive`, `umfpack`, `klu`, ...)

[5] Tutorial-for-FSI-with-OpenFOAM-and-CalculiX

3. using different damping coefficients

- in the adapter:
  1. using different ramp profiles and times
- in the coupling parameters:
  1. changing coupling strategies (serial, parallel)
  2. using more strict convergence criteria
  3. turning the *extrapolation* parameter on or off (*on* gets a better initial guess for the next time step)
  4. changing acceleration algorithm and parameters

None of the simulations allowed to simulate the system correctly. In general every simulation diverged at some point of the ramp phase.

Thus a sensitivity analysis has been carried out, in order to better understand the limits of the current coupling, as described in the following section.

6.8 Sensitivity analysis of FSI1-FSI3 Benchmarks

It can be interesting to understand what combination of simulation parameters has a stronger impact on the convergence or divergence of the model. We decided to focus on the FSI3 benchmark and perturb some of the input parameters.

Table 6.27 shows that the two test cases present the same Cauchy Number C_Y. It looks that, at first, it might not be a significant parameter to understand the limits of application of this setup. For that reason we decided to focus on the variation of the *mass number* M and the *reduced velocity* U_R and observe in which cases the simulation diverges.

We started from the original FSI3 test case with the same configuration parameters given in Section 6.6, and modified the following parameters:

1. ρ_f (fluid density): $100 \rightarrow 1000\ \mathrm{kg\,m^{-3}}$, so that the mass number is in the range $0.1 \rightarrow 1$. A mass number of 0.1 has shown to converge (i.e. FSI2 benchmark) while previous tests, characterized by smaller mass number, did not present convergence issues. Solid density ρ_s is kept fixed at $1000\ \mathrm{kg\,m^{-3}}$.
2. E (structure elastic modulus): $5.6 \cdot 10^6 \rightarrow 2 \cdot 10^{11}$ Pa, so ranging from the elastic modulus of a soft rubber (used in the FSI3 benchmark) to the one of steel. The corresponding reduced velocity U_R, at $\vec{u} = 2\mathrm{m\,s^{-1}}$, is in the range $\approx 7 \cdot 10^{-4} \rightarrow 0.13$
3. $\vec{u}$ (mean flow velocity): $0.2 \rightarrow 10\ \mathrm{m\,s^{-1}}$ so that the Reynolds number Re ranges from 20 (as in FSI1) to 1000.

We were interested in keeping the simulation time short, so we considered 1 s of simulation: the first 0.5 s has been used to ramp up forces on the structure, from $10% up to 100%, then the simulation evolves for the remaining 0.5s. An experiment is considered successfully completed if it reaches the final time without non-physical results due to lack of convergence at some time steps.

The results are presented in the following subsections.

6.8.1 FSI3 sensitivity Analysis

The first scenario consists in a perturbation of the FSI3 test case, i.e. keeping $\vec{u} = 2\text{m}\,\text{s}^{-1}$ and $Re = 200$. The results are illustrated in Table 6.28:

		mass ratio				
U_R	E[MPa]	0.1	0.2	0.25	0.5	1
$1.41 \cdot 10^{-4}$	$2 \cdot 10^5$	2.204		2.618	3.126	4.418
$6.32 \cdot 10^{-4}$	$1 \cdot 10^4$		15.024	43.1		
$2 \cdot 10^{-3}$	$1 \cdot 10^3$	23.46		10.78		
$6.32 \cdot 10^{-3}$	$1 \cdot 10^2$		7.592	11.788		
$1.41 \cdot 10^{-2}$	20	11.46		87.675		
$2.67 \cdot 10^{-2}$	5.6	7.216	36.426			**FSI3**

Table 6.28. FSI3 sensitivity analysis: *green cells* simulation completed with average number of iterations, *red cells* simulation diverged

Table 6.28 shows that there is a limit in the ability of the coupled system to produce results. It is situated in the range between 0.25 and 0.5 of mass ratio. This condition can be improved if the material is much stiffer (which corresponds to a smaller U_R) and can become worse when the material is more flexible. The lower right corner of the table, which represents the FSI3 setup, appears to be quite far from the actual range of convergence of the system. The table also shows, for completed simulations, the average number of iterations required for each time step to converge. A trend is visible (even if it does not hold for every case): the higher *mass ratio*, the more iterations are needed. The similar trend is visible for U_R.

The empty green cells have not been simulated. It is supposed that, if a simulation completes successfully for a given *mass ratio*, it will complete also for a lower one. Conversely, if a simulation does not work for a given U_R, it will not work for a higher U_R (i.e. for a more flexible material).

6.8.2 FSI1 sensitivity Analysis

The same kind of analysis has been conducted considering a fluid velocity of $0.2\text{m}\,\text{s}^{-1}$ ($Re = 20$), thus replicating FSI1 test case in [66].

		mass ratio				
U_R	E[MPa]	0.1	0.2	0.25	0.5	1
$1.41 \cdot 10^{-5}$	$2 \cdot 10^5$		2.138		2.146	3.064
$6.32 \cdot 10^{-5}$	$1 \cdot 10^4$			2.337		
$2 \cdot 10^{-4}$	$1 \cdot 10^3$		9.83			
$6.32 \cdot 10^{-4}$	$1 \cdot 10^2$	8.688		24.45		
$1.41 \cdot 10^{-3}$	20		18.238	38.398		
$2.67 \cdot 10^{-3}$	5.6	6.314				**FSI1**

Table 6.29. FSI1 sensitivity analysis

Table 6.29 illustrates the results: as in the previous section, *green cells* indicate that the simulation completed (the number represents the average number of iterations), while *red cells* indicate that the simulation diverged. In this case *yellow cells* indicate that the simulation finished, but at some point it reached the maximum number of iterations, meaning that the progressive loading of the structure is difficult. It can be possible that there exist a better set of coupling parameters that makes a specific test work better, nevertheless it looks to be a symptom of the fact that the simulation is close to the limits of applicability.

Comparing the results with Table 6.28, it can be seen that the range of feasible parameters becomes smaller. This seems to be in accordance with the idea that a lower fluid velocity makes the interaction stronger and the convergence more difficult.

Similarly, the lower right cell represents test case FSI1 and it seems to be not possible to simulate it with the current setup.

6.8.3 Sensitivity Analysis at higher velocity

In order to have more insight on the influence of the physical parameters on the simulation, also a higher fluid velocity of $10\,\mathrm{m\,s^{-1}}$ ($Re = 1000$) has been considered.

		mass ratio				
U_R	E[MPa]	0.1	0.2	0.25	0.5	1
$7.07 \cdot 10^{-4}$	$2 \cdot 10^5$		5.552		6.706	8.647
$3.16 \cdot 10^{-3}$	$1 \cdot 10^4$	10.184				9.713
$1 \cdot 10^{-2}$	$1 \cdot 10^3$		14.101	50.731		36.566
$3.16 \cdot 10^{-2}$	$1 \cdot 10^2$	10.640		17.122		
$7.07 \cdot 10^{-2}$	20		42.998	35.046		
$1.34 \cdot 10^{-1}$	5.6	11.725				

Table 6.30. Re 1000 sensitivity analysis

Table 6.30 shows a pattern similar to previous experiments. In this case, a larger number of parameter combinations appears to be feasible.

6.8.4 Analysis of the results

The current knowledge regarding the overall setup of the test case does not allow to correctly simulate the benchmark experiments FSI1 and FSI3, characterized by a mass ratio of 1. Perturbing some of the parameters, thus affecting the mass ratio M or the reduced velocity U_R, allows to reach feasible configurations.

Tables 6.28 to 6.30 show all a similar path: the upper left part of the table contains feasible configurations, while the lower right part the unfeasible ones. If the fluid velocity is lower, more configurations fail.

Those aspects appear to agree with the explanations given in Section 3.5 and in the cited literature. The model considered here shows a a clear relationship between mass ratio, structure stiffness and simulation outcome at a prescribed fluid velocity. The same clear path would not hold if we considered reduced velocity U_R only, thus putting all experiments together. For example, row 1 of Table 6.30 tells that $U_R \approx 7 \cdot 10^{-4}$ would allow to reach $M = 1$, while row 4 of Table 6.29 tells that $U_R \approx 6.3 \cdot 10^{-4}$ would allow to reach only $M \approx 0.2$.

6.9 Flapping wing simulation

An application example of preCICE-MBDyn coupling in the study of FSI problems can be found in [35], in which the effect of spanwise wing flexibility on thrust, lift and propulsive efficiency of a rectangular wing, oscillating in pure heave, is analyzed by means of water tunnel experiments.

The study shows that, for some oscillating frequencies, a degree of spanwise flexibility yields a small increase in the thrust coefficient and a small decrease in power-input requirement, resulting in higher overall efficiency.

6.9.1 Experimental setup

Before describing the FSI model and the simulation, it is necessary to briefly introduce the experimental setup.

The study considers three types of rectangular wings, with profile *NACA 0012* and with different section properties, as shown in Figure 6.37. The section labeled (i) is considered *inflexible*, the one labeled (ii) is considered *flexible* and the last one *highly flexible*.

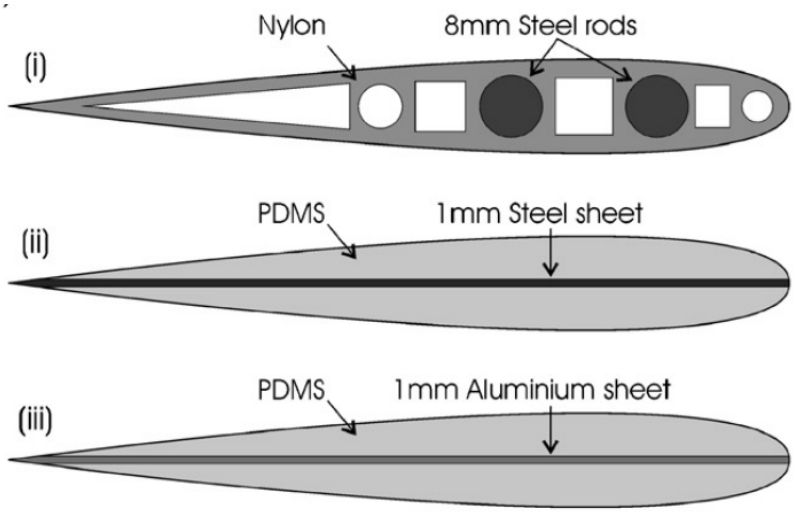


Figure 6.37. wing section properties (image taken from [35])

Each wing has the following dimensions: chord $c = 100$ mm and span $b = 300$ mm.

The experimental setup is shown in Figure 6.38. The displacement of the root section is given by $s = a_{ROOT} \sin(\omega t)$, where $a_{ROOT} = 0.175 \cdot c$. The flow velocity U_0 is in the range $1 \div 3\ \mathrm{m\,s^{-1}}$. The following dimensionless parameters are considered:

- $Re = \frac{\rho U_0 c}{\mu}$: Reynolds number
- $k_G = \frac{\pi f c}{U_0}$: Garrick reduced frequency
- $S_r = \frac{2 f a_{MID}}{U_0}$: Strouhal number at mid-span

Experiments are carried out for the three types of wings in the following ranges: $Re = 1 \cdot 10^4 \div 3 \cdot 10^4$ and $k_G = 0 \div 7$.

The results give information concerning the average thrust coefficient $C_T = \frac{T}{\frac{1}{2}\rho U_0^2 c}$ over a finite number of cycles and the mean power input coefficient $\bar{C}_P = \frac{\bar{F_y} v}{\frac{1}{2}\rho U_0^3 c}$.

Besides, information concerning the ratio $\frac{a_{TIP}}{a_{ROOT}}$ and tip phase lag ϕ are given.

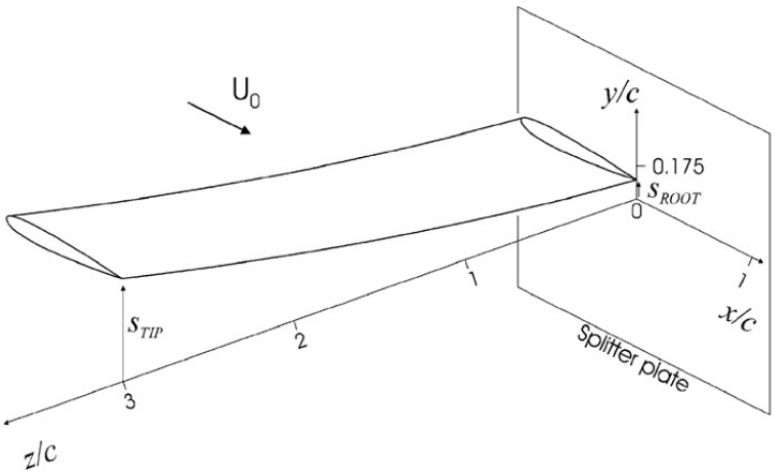


Figure 6.38. experimental setup (image taken from [35])

6.9.2 Simulation setup

The experimental setup described in [35] has been replicated in a FSI simulation using MBDyn as structural solver and OpenFOAM as CFD solver.

Fluid domain

The fluid domain is represented by a box of size $1.5 \times 0.6 \times 0.5$m, as represented in Figure 6.39. The boundary conditions are:

- constant inlet velocity for the face at $x = -0.25$m,
- constant pressure for the face at $x = 1.25$m,
- symmetry plane at the root of the wing ($z = 0$),
- slip walls for the other external surfaces,
- no-slip wall for the wing surface.

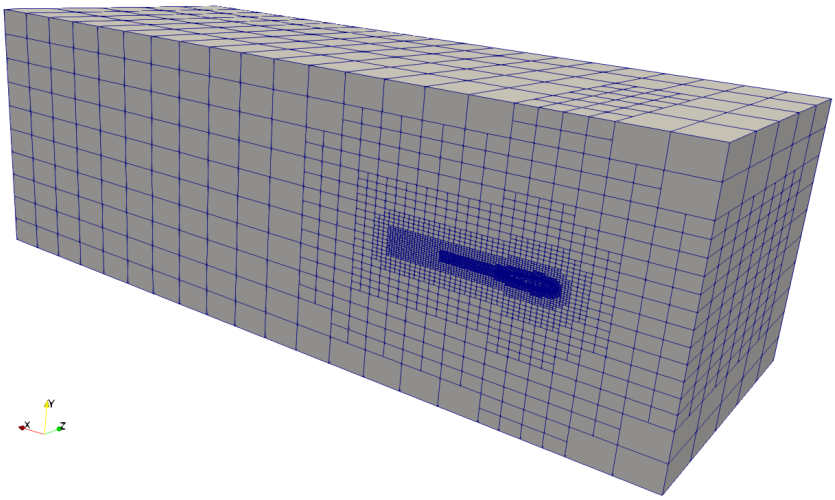
The NACA 0012 wing has been drawn in *Salome* and exported in OpenFOAM as *.stl* file. The mesh has been built with the tool *snappyHexMesh* and it is composed of 218451 cells.

Interface

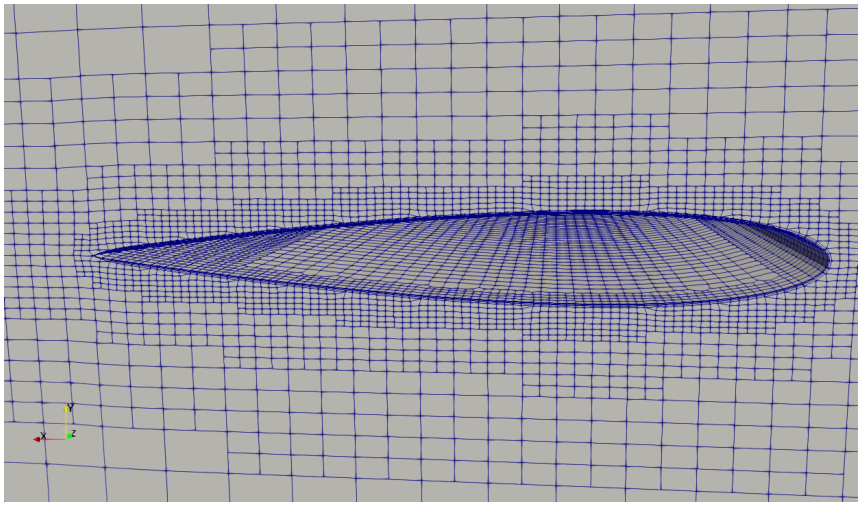
The same NACA 0012 profile drawn in Salome has been used to generate the interface mesh for the `external structural mapping` of MBdyn. The interface mesh is composed of 1286 cells (1200 quadrangles and 86 triangles), as shown in Figure 6.40.

Structural domain

The structural model is composed of 10 MBDyn `beam` elements. Considering the *flexible* (or the *highly flexible*) structure, the stiffness of the wing can be considered completely given by the 1 mm metal plate, as the PDMS, with a Young modulus of $360 \div 870$ kPa would contribute with only a small fraction of the overall stiffness.



(a) mesh bounding box



(b) wing detail

Figure 6.39. NACA 0012 fluid mesh

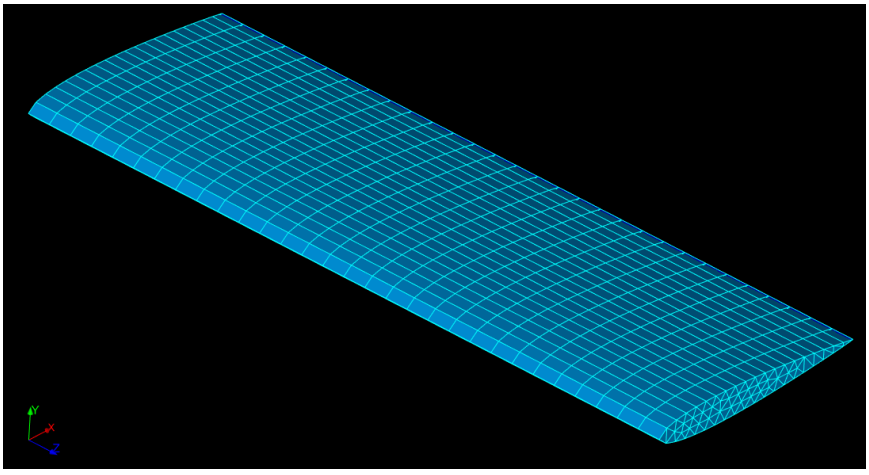


Figure 6.40. NACA 0012 interface mesh

The stiffness matrix of each beam element is given in Equation 6.2, in which w represents the chord and h the metal thickness.

$$\begin{bmatrix} Ewh & & 0 & \dots & & 0 \\ & Gwh & & & & \\ & & Gwh & & & \\ & & & G\frac{1}{3}wh^3 & & \vdots \\ & & sym. & & E\frac{1}{12}wh^3 & \\ & & & & & E\frac{1}{12}hw^3 \end{bmatrix} \tag{6.2}$$

At each of the 21 nodes of the structure there is a `body` element attached, carrying the mass of the corresponding chunk of beam (both metal and PDMS).

The wing motion is achieved by means of a `total pin joint` element attached to the root of the wing, which moves the root `node` with a $\sin(\omega t)$ law along the y direction. The motion starts with a ramp of the kind $\frac{1}{2}\left(1-\cos\left(\frac{t}{\tau}\right)\right)$ in order to ease convergence.

Coupling

The main data concerning the coupling between the fluid solver and the structural solver are given in Table 6.31:

parameter			value
simulation time	t	[s]	2
step size	Δt	[s]	10^{-3}
coupling scheme			serial implicit $S \rightarrow F$
coupling algorithm			IQN-ILS
displacement rel. convergence limit			10^{-3}
force rel. convergence limit			10^{-3}
interface mesh mapping			Nearest neigh. and RBF

Table 6.31. NACA 0012: coupling parameters

Different experiments have been carried out considering different parameters. For example, a shorter time step is beneficial for a better mesh displacement, at the expense of a longer simulation time. At the same time, convergence limits have an obvious impact on the simulation time, as more iterations are required in order to reach a lower convergence limit.

Finally, also mesh mapping strategy (see Section 3.4.3) influences the simulation quality and time: *nearest-neighbor* is faster but can be detrimental for mesh movement, on the other side, the Radial Basis Function (RBF) mapping gives better results, but for larger grids (≈ 4000 points) the computational cost of the mapping becomes relevant. Moreover, it needs to be tuned.

6.9.3 Results

Due to resource constraints, only some preliminary results have been carried out fort the *flexible* wing. In particular, the results obtained with a root displacement of ± 2.5 mm at 2.5 Hz with a fluid velocity of 1 $\mathrm{m\,s^{-1}}$ are presented here. This corresponds to:

$$Re = 10000, \quad k_G = 0.785, \quad S_r \approx 1.5 \cdot 10^{-2} \tag{6.3}$$

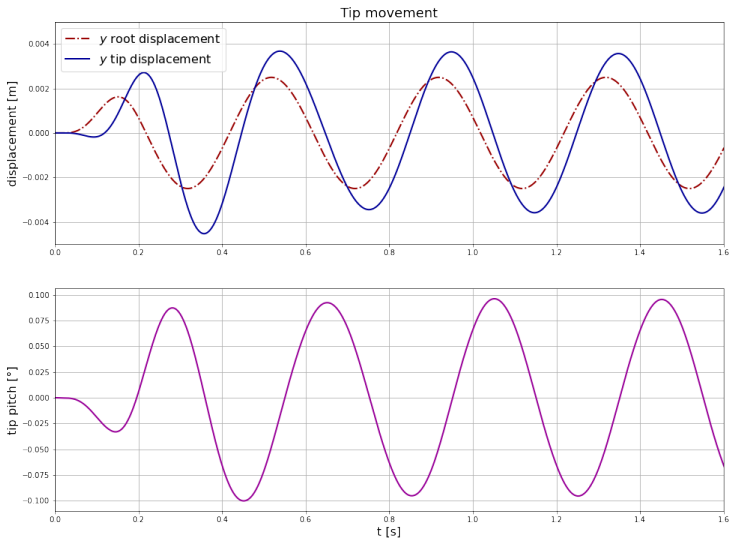


Figure 6.41. NACA 0012 displacement

The tip displacement is shown in Figure 6.41. After the first transient oscillation, the peak tip displacement settles to around 3.62mm. compared to the root displacement, the ratio is around $\frac{a_{TIP}}{a_{ROOT}} \approx 1.45$ with a lag of around 27^o. These results are much closer to the the ones of the*highly-flexible* wing presented in [35].

The first eigenfrequency of the model, computed in MBDyn, is ≈ 6.8 Hz. The first eigenfrequency of the metal plate alone, considered as a cantilevered beam, would be around 9 Hz. The mass of the PDMS lowers the value, as expected. The amplification and the lag found in the simulation seem to be compatible with those theoretical data, while the experimental model seems to be stiffer. Citing the article: "the airfoil comprises a tear-drop solid aluminium leading edge followed by a flexible steel plate". The aluminium leading edge might contribute to the overall stiffness of the model.

The root displacement considered in the simulation is much smaller than the one used in the article in order to keep the mesh deformation small. This aspect might also be a cause of the discrepancy in the results. Nevertheless the difference between the experimental and the computer model needs to be further analyzed.

On the other hand, the pitch angle of the wing tip, in the second plot of Figure 6.41, shows that the wing is torsionally rigid, as expected.

The forces and moment exerted on the wing are shown in Figure 6.42. Focusing on the force on the x-direction, it is possible to compute the average C_D:

$$\bar{C}_D = \frac{\bar{F}_x}{\frac{1}{2}\rho U_0^2 c \cdot b} \approx 0.02615 \tag{6.4}$$

which is close to the values $0.0245 \div 0.028$ presented in the sources of the article.

The number of required iterations is, in this model, very stable: 15 iterations are required at each time step to converge, as shown in Figure 6.43. As in the previous examples, nodal forces converge slower than displacements.

Finally some images taken from the fluid and the solid side of the simulation are shown in Figure 6.44 and in Figure 6.45.

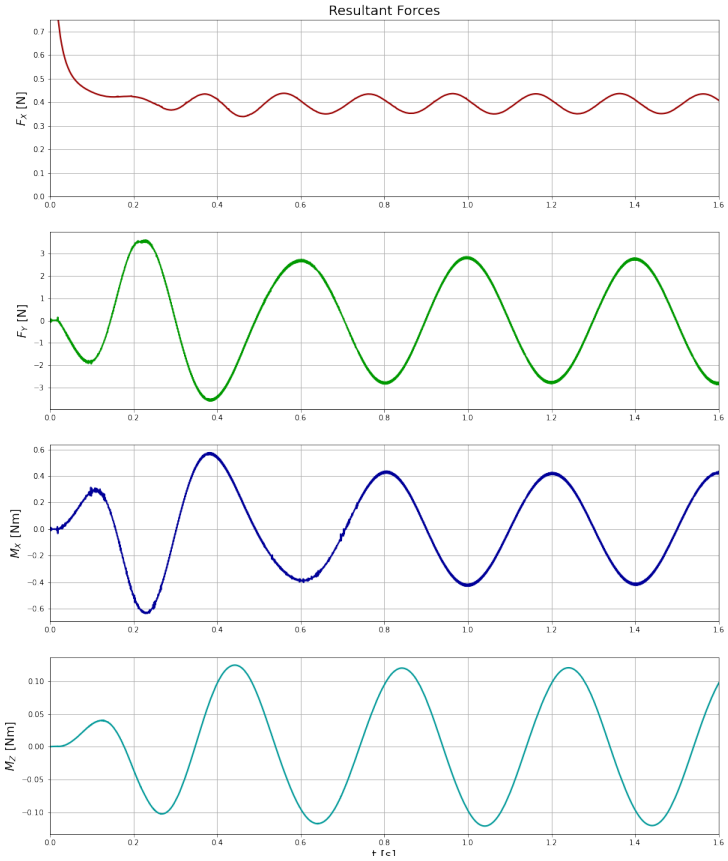


Figure 6.42. NACA 0012 forces

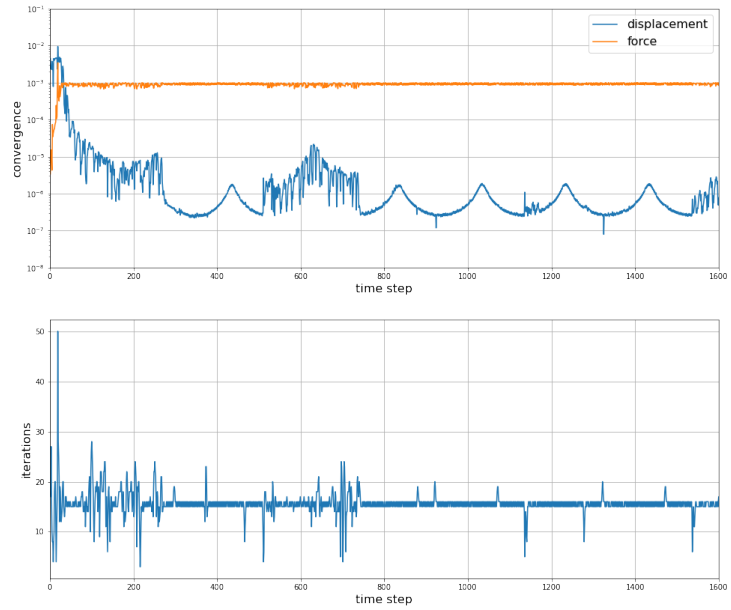
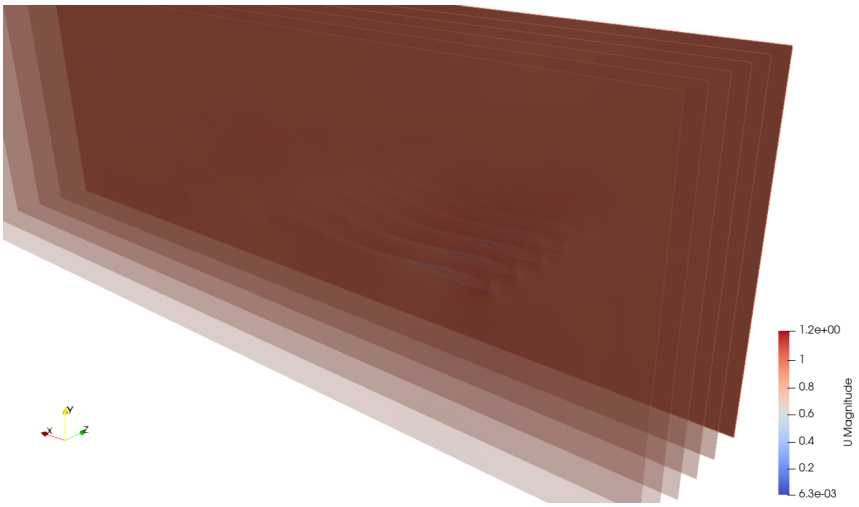
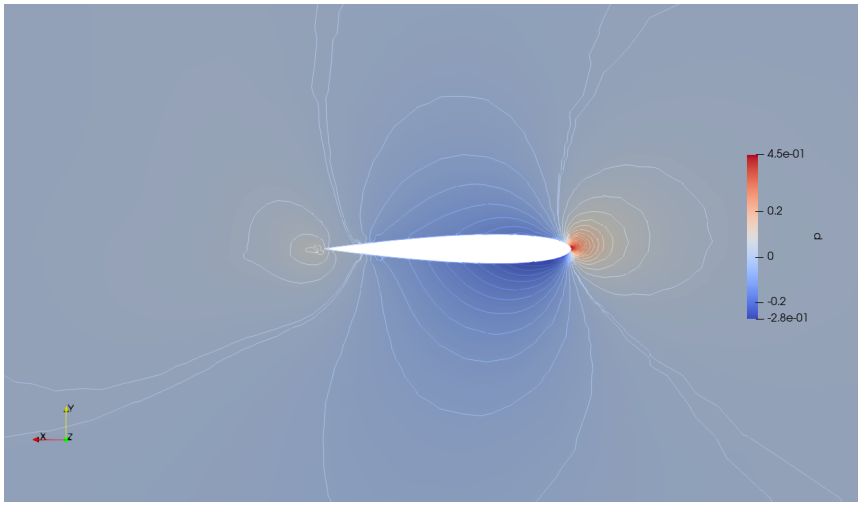


Figure 6.43. NACA 0012 iterations

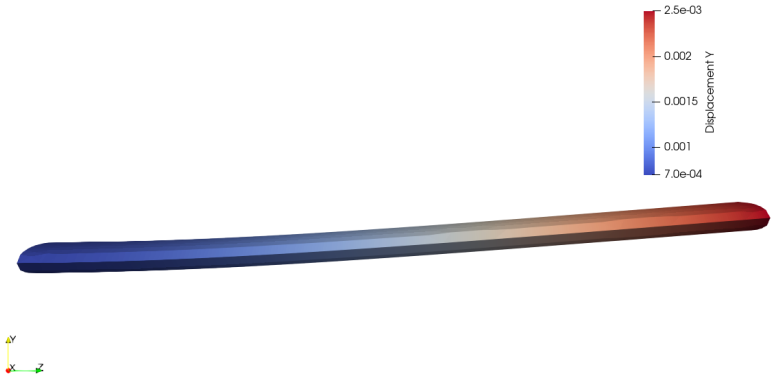


(a) NACA 0012 velocity t=1.2s

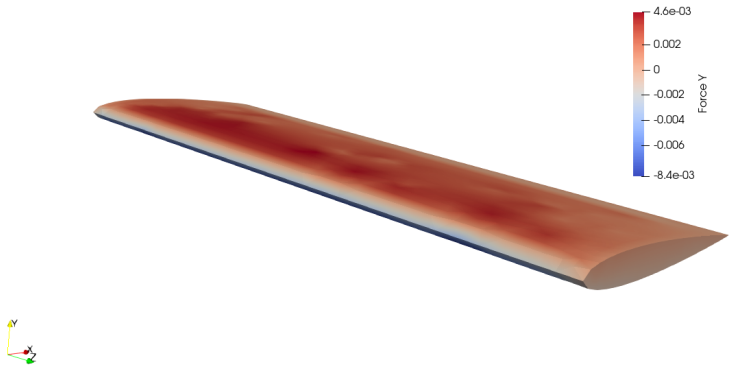


(b) NACA 0012 pressure profile t=1.2s

Figure 6.44. NACA 0012 fluid domain



(a) NACA 0012 deformed shape (10x)



(b) NACA 0012 forces on surface

Figure 6.45. NACA 0012 solid domain

Chapter 7

Conclusions

In this thesis, we developed an adapter for MBDyn to enable Fluid-Structure Interaction (FSI) simulations with the coupling library preCICE. The adapter has been validated both qualitatively and quantitatively with several test cases.

This work allows MBDyn to extend its capabilities in FSI simulations, as it can be coupled with a wide set of CFD solvers which are already connected to preCICE.

Among other possible simulations, this connection allows the FSI simulation of slender bodies immersed in fluid flow and gives the possibility of prescribing different structural properties independently from the external geometry: shape and structural properties of the structure can be decoupled, thus allowing specific and independent tuning of parameters.

The adapter introduced here can also be used with other MBDyn elements (e.g. shells), simply changing the MBDyn configuration file. Besides, it has been developed independently from MBDyn itself as it makes use of external APIs. So it does not impact directly the development of MBDyn.

During the development phase, a great attention has been given to the configuration process of a simulation. For this reason, most of the adapter parameters are read at runtime from a JSON configuration file.

Given the development and the tests carried out in this work, some directions of further analysis and development appear to be particularly interesting. First of all it is necessary to better understand the limits of application for the whole FSI setup as it is now: for example if there exists a set of parameters that allows to simulate models similar to the FSI3 benchmark, as it is known to work by simply using a different structural solver.

It will also be important to analyze the behavior and the performances of a simulation in more complex, 3D real world scenarios, where more computational resources are needed.

There can also be future developments in the software code of the adapter itself. Some of them concern the whole simulation: for example it would be useful to make the configuration easier and less redundant, as some of the parameters are duplicated among the components.

Other extensions are relative to the possibility of applying a variable time step to the MBDyn simulation: in principle, the fluid and the solid simulation can advance at different time steps, as they might have different specific needs. But in this case, at some point in the simulation, preCICE will need to synchronize the two simulations by forcing a prescribed time step to the solvers. For our simulations we kept the time step coincident among the solvers: the fluid solver generally requires a shorter time step and the MBDyn simulation is not costly. But there can be situations in which this feature might be useful. It has already been partly developed and it does not require much work to be finalized.

Another extension concerning the adapter consists in giving preCICE information about the topology of the mesh (e.g. triangles or quadrangles defining the interface). This would

allow a better use of some of the mapping strategies. This information is already available in the adapter as it is used to draw output information and should be easy to send it also to preCICE.

Anther, more complex, development would consist in possibly extending MBDyn capabilities to simulate Solid-Solid Interaction (SSI) applications. This would allow to couple MBDyn, though preCICE, with different FEM solvers (or even with itself). This kind of coupling would allow to simulate multibody models, in which some relevant sub-models are simulated by means of a dynamic FEM model. This kind of development would require some modifications also in the MBDyn code.

Some of the preliminary results have been already presented to the *second preCICE community hour*, but for the continuation and the future development of this work we aim at including the code, together with some usage examples, into the MBDyn and preCICE repositories and websites, with the goal to find other benchmarks, users, comparisons and use scenarios.

Appendix A

preCICE configuration file

The following represents a rather standard preCICE configuration file. Its main parts are:

- definition of the interface dimension (line 3)
- definition of names of exchanged data (lines 5-6)
- definition of the mesh names and which data are read or written (lines 8-15)
- definition of the solvers (participants) together with information concerning the meshes and exchanged data (lines 17-27)
- definition of the communication protocol (line 29)
- definition of the coupling strategy, together with simulation time, time step, convergence criteria and acceleration method (lines 31-40)

```
1  <?xml version="1.0"?>
2  <precice-configuration>
3  <solver-interface dimensions="3">
4
5    <data:vector name="Forces" />
6    <data:vector name="Displacements" />
7
8    <mesh name="FluidMesh">
9      <use-data name="Forces" />
10     <use-data name="Displacements" />
11   </mesh>
12   <mesh name="StructureMesh">
13     <use-data name="Forces" />
14     <use-data name="Displacements" />
15   </mesh>
16
17   <participant name="FluidSolver">
18     <use-mesh name="FluidMesh" provide="yes" />
19     <use-mesh name="StructureMesh" from="StructureSolver" />
20     <write-data name="Forces" mesh="FluidMesh" />
21     <read-data name="Displacements" mesh="FluidMesh" />
22   </participant>
```

```
23   <participant name="StructureSolver">
24     <use-mesh name="StructureMesh" provide="yes"/>
25     <write-data name="Displacements" mesh="StructureMesh" />
26     <read-data name="Forces" mesh="StructureMesh" />
27   </participant>
28
29   <m2n:sockets from="FluidSolver" to="StructureSolver" />
30
31   <coupling-scheme:serial-implicit>
32     <participants first="FluidSolver" second="StructureSolver" />
33     <max-time-windows value="10" />
34     <time-window-size value="1.0" />
35     <max-iterations value="15" />
36     <relative-convergence-measure limit="1e-3" data="Displacements" mesh="
         StructureSolver"/>
37     <exchange data="Forces" mesh="StructureMesh" from="FluidSolver" to="
         StructureSolver" />
38     <exchange data="Displacements" mesh="StructureMesh" from="
         StructureSolver" to="FluidSolver"/>
39   </coupling-scheme:serial-implicit>
40 </solver-interface>
41 </precice-configuration>
```

Listing A.1. preCICE configuration file example

Appendix B

MBDyn adapter configuration file

The data required by the MBDyn *adapter* are described in Section 5.3. The order within the file is irrelevant, but for clarity reasons similar parameters have been grouped together:

- Data concerning the preCICE coupling (lines 2-6)
- Data necessary for MBdyn coupling (lines 7-11)
- Data used for the simulation (lines 12-15)
- Information for output (lines 16-19)
- Debug parameters (lines 20-22)

```
1  {
2    "precice-config": "../precice-config.xml",
3    "readDataName": "Displacements0",
4    "writeDataName" :"Forces0",
5    "participantName" :"Solid",
6    "meshName": "Solid-Mesh",
7    "mesh": "./mesh/root0f.dat",
8    "mbdyn-input": "./map_10n_3x_21j.mbd",
9    "node-socket": "/tmp/mbdyn.node.sock",
10   "mbdyn-output": "./out_mbd",
11   "displacement-delta": false,
12   "iterstart": 500,
13   "coeff0": 0.05,
14   "period": 2000,
15   "ramp-type": "linear",
16   "write-interval": 10,
17   "resultant-file": "./output/resultant.txt",
18   "root-coords": [0.0, 0.0, 0.0],
19   "vtk-output": "./output/MBDyn-OF_",
20   "pre-iteration": 5,
21   "read-coords": true,
22   "every-iteration": false
23 }
```

Listing B.1. MBDyn adapter configuration file example

Appendix C

preCICE API

C.1 preCICE API calls

Each participating solver needs to be linked to the preCICE library and call methods from its application programming interface (API). The preCICE adapter groups together the calls to the API. While preCICE is written in C++, it provides an API also for C, Fortran and Python. An excerpt from the C++ API is shown in Listing C.1 as drawn from the preCICE documentation.

A coupled simulation is first configured and initialized phase, then multiple coupling advancements occur up to a finalization phase.

First a `SolverInterface` object needs to be created: its parameters are the rank of the process and the size of the communicator if the execution is parallel.

The `configure()` method defines the preCICE configuration file, reads it and sets up the communication inside the solver. The following methods that follow in Listing C.1 are called "steering methods". The `initialize()` method sets up the data structures and communication channels to other participants.At the first communication the participants exchange meshes. It returns the maximum time step size that the solver is allowed to execute next.

`initializeData()` optionally transfers any initial non-zero coupling data values among participants. The `advance()` method take care of equation coupling, data mapping and communication.

`finalize()` destroys the data structures and closes the communication channels.

Each solver defines its interface mesh. Each mesh and each node on the mesh are assigned an integer ID. `getMeshID()` gets the ID of the mesh over which the coupling is performed. `setMeshVertex()` creates a vertex on the specified mesh position. Additionally, topological information, such as edges or triangles can be passed to preCICE with additional methods.

Data values are assigned to meshes by means of methods with names of the kind `write*Data()`, which fill the buffers with data from the solver's mesh or with methods like `read*Data()`, which read data from the buffers into the solver's mesh.

Some other methods allow to access useful information for the coupling, in particular if the simulation is still running, if there are new data do be read or written or if the current coupling time step has converged or not.

The solver asks if it has to write or read a checkpoint and it informs preCICE that it fulfilled these tasks [70].

```
class SolverInterface
{
public:
  SolverInterface(
    const std::string& solverName,
    int solverProcessIndex,
    int solverProcessSize);

  void configure(const std::string& configurationFileName);

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

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

  void writeScalarData(int dataID, int valueIndex, double value);
  void writeVectorData(int dataID, int valueIndex, const double* value);
  void writeBlockScalarData(
    int dataID,
    int size,
    int* valueIndices,
    double* values);

  bool isCouplingOngoing();
  bool isReadDataAvailable();
  bool isWriteDataRequired(double computedTimestepLength);
  bool isTimestepComplete();

  bool isActionRequired(const std::string& action);
  void fulfilledAction(const std::string& action);
// ...
};
```

Listing C.1. preCICE API methods

C.2 preCICE adapter structure

An example of an adapted solver is shown in Listing C.2, as published in the presentation article of preCICE [5] and also in the preCICE website[1]

```
turnOnSolver(); //e.g. setup and partition mesh

precice::SolverInterface precice("FluidSolver","precice-config.xml",rank,
    size); // constructor

int dim = precice.getDimension();
int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at wet surface

// determine vertexSize
double* coords = new double[vertexSize*dim]; // coords of vertices at wet
    surface

// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);
delete[] coords;

int displID = precice.getDataID("Displacements", meshID);
int forceID = precice.getDataID("Forces", meshID);
double* forces = new double[vertexSize*dim];
double* displacements = new double[vertexSize*dim];

double dt; // solver timestep size
double precice_dt; // maximum precice timestep size

precice_dt = precice.initialize();
while (precice.isCouplingOngoing()){

  if(precice.isActionRequired()){
    saveOldState(); // save checkpoint
    precice.markActionFulfilled();
  }

  precice.readBlockVectorData(displID, vertexSize, vertexIDs, displacments);
  setDisplacements(displacements);
  dt = beginTimeStep(); // e.g. compute adaptive dt
  dt = min(precice_dt, dt);
  computeTimeStep(dt);
  computeForces(forces);
  precice.writeBlockVectorData(forceID, vertexSize, vertexIDs, forces);
  precice_dt = precice.advance(dt);
  if(precice.isActionRequired()){ // timestep not converged
    reloadOldState(); // set variables back to checkpoint
```

[1] precice/wiki/Adapter-Example

```
43      precice.markActionFulfilled();
44    }
45    else{ // timestep converged
46      endTimeStep(); // e.g. update variables, increment time
47    }
48  }
49  precice.finalize(); // frees data structures and closes communication
      channels
50  delete[] vertexIDs, forces, displacements;
51  turnOffSolver();
```

Listing C.2. preCICE adapter structure

Appendix D

MBDyn input/output file example

D.1 MBDyn input file

Main File

The main file used for the FSI simulations follows the structure described in Section 4.1.1.

```
begin: data;
  problem: initial value;
end: data;

set: real DT = 5e-4;
set: real INITIAL_TIME = 0.0;
set: real FINAL_TIME = 10.0;

begin: initial value;

  initial time: INITIAL_TIME;
  final time: FINAL_TIME;
  time step: DT;

  method: ms, .6;
  nonlinear solver: newton raphson, modified, 5;
  linear solver: umfpack, colamd, mt, 1;

  tolerance: 1e-6;
  max iterations: 1000;

  derivatives coefficient: 1e-9;
  derivatives tolerance: 1e-6;
  derivatives max iterations: 100;

  output: iterations;
  output: residual;
end: initial value;

# number of beam3 elements
set: integer N = 10;
```

```
32
33 begin: control data;
34   structural nodes:
35     +1 # clamped node
36     +2*N # other nodes
37   ;
38   rigid bodies:
39     +2*N # mass of other nodes
40   ;
41   joints:
42     +1 # clamp
43     +2*N # other total joints on nodes to force 2D
44   ;
45   beams:
46     +N # the whole beam
47   ;
48   forces:
49     +1 # loads the beam
50     +1 # load on last node
51   ;
52 end: control data;
53
54 # root reference
55 set: const integer ROOT = 1;
56 set: const real Xroot = 0.0;
57 set: const real Yroot = 0.0;
58 set: const real Zroot = 0.0;
59 reference: ROOT, Xroot, Yroot, Zroot, eye, null, null;
60
61
62 # material density [kg/m^3]
63 set: real rho = 100.;
64 # beam width (z direction)
65 set: real w = 4.e-3;
66 # beam height (y direction)
67 set: real h = 6.e-4;
68 # beam lenght (x direction)
69 set: real L = 0.04;
70 # lenght of half beam3 element
71 set: real dL = L/(2*N);
72
73 # mass of single chunck
74 set: real m = rho*w*h*dL;
75
76 # elastic properties
77 set: real E = 2.5e5;
78 set: real nu = 0.35;
79 set: real G = E/(2*(1+nu));
80 set: real A = w*h;
81 set: real Jz = 1./12.*w*h^3;
```

```
82  set: real Jy = 1./12.*h*w^3;
83  set: real Jp = Jy + Jz;
84  set: real damp = .01;
85
86  set: integer curr_node;
87
88  begin: nodes;
89    structural: ROOT, static,
90      reference, ROOT, null,  # position
91      reference, ROOT, eye,  # orientation
92      reference, ROOT, null,  # initial velocity
93      reference, ROOT, null;  # angular velocity
94
95    set: curr_node = 2;
96    include: "beam.nod";
97    set: curr_node = 4;
98    include: "beam.nod";
99    set: curr_node = 6;
100   include: "beam.nod";
101   set: curr_node = 8;
102   include: "beam.nod";
103   set: curr_node = 10;
104   include: "beam.nod";
105   set: curr_node = 12;
106   include: "beam.nod";
107   set: curr_node = 14;
108   include: "beam.nod";
109   set: curr_node = 16;
110   include: "beam.nod";
111   set: curr_node = 18;
112   include: "beam.nod";
113   set: curr_node = 20;
114   include: "beam.nod";
115 end: nodes;
116
117
118 set: const integer BEAM_NNODES = 2*N+1;
119 set: real da = .005;
120 set: integer CURR_BEAM = 1;
121
122 begin: elements;
123   joint: 500+ROOT, clamp, ROOT, node, node;
124
125   set: curr_node = 2;
126   include: "beam.elm";
127   include: "joint.elm";
128   set: curr_node = 4;
129   include: "beam.elm";
130   include: "joint.elm";
131   set: curr_node = 6;
```

```
132   include: "beam.elm";
133   include: "joint.elm";
134   set: curr_node = 8;
135   include: "beam.elm";
136   include: "joint.elm";
137   set: curr_node = 10;
138   include: "beam.elm";
139   include: "joint.elm";
140   set: curr_node = 12;
141   include: "beam.elm";
142   include: "joint.elm";
143   set: curr_node = 14;
144   include: "beam.elm";
145   include: "joint.elm";
146   set: curr_node = 16;
147   include: "beam.elm";
148   include: "joint.elm";
149   set: curr_node = 18;
150   include: "beam.elm";
151   include: "joint.elm";
152   set: curr_node = 20;
153   include: "beam.elm";
154   include: "joint.elm";
155
156     force: CURR_BEAM, external structural mapping,
157     socket,
158     create, yes,
159         path, "/tmp/mbdyn.node.sock",
160         no signal,
161         coupling,
162           # loose,
163           tight,
164     #reference node, 1,
165     #orientation, euler 123,
166     #use reference node forces, yes,
167     points number, 3* BEAM_NNODES,
168       CURR_BEAM,
169         offset, null,
170         offset, 0,da, 0.,
171         offset, 0., 0., da,
172       CURR_BEAM + 1,
173         offset, null,
174         offset, 0,da, 0.,
175         offset, 0., 0., da,
176       CURR_BEAM + 2,
177         offset, null,
178         offset, 0,da, 0.,
179         offset, 0., 0., da,
180       CURR_BEAM + 3,
181         offset, null,
```

```
182         offset, 0,da, 0.,
183         offset, 0., 0., da,
184       CURR_BEAM + 4,
185         offset, null,
186         offset, 0,da, 0.,
187         offset, 0., 0., da,
188       CURR_BEAM + 5,
189         offset, null,
190         offset, 0,da, 0.,
191         offset, 0., 0., da,
192       CURR_BEAM + 6,
193         offset, null,
194         offset, 0,da, 0.,
195         offset, 0., 0., da,
196       CURR_BEAM + 7,
197         offset, null,
198         offset, 0,da, 0.,
199         offset, 0., 0., da,
200       CURR_BEAM + 8,
201         offset, null,
202         offset, 0,da, 0.,
203         offset, 0., 0., da,
204       CURR_BEAM + 9,
205         offset, null,
206         offset, 0,da, 0.,
207         offset, 0., 0., da,
208       CURR_BEAM + 10,
209         offset, null,
210         offset, 0,da, 0.,
211         offset, 0., 0., da,
212       CURR_BEAM + 11,
213         offset, null,
214         offset, 0,da, 0.,
215         offset, 0., 0., da,
216       CURR_BEAM + 12,
217         offset, null,
218         offset, 0,da, 0.,
219         offset, 0., 0., da,
220       CURR_BEAM + 13,
221         offset, null,
222         offset, 0,da, 0.,
223         offset, 0., 0., da,
224       CURR_BEAM + 14,
225         offset, null,
226         offset, 0,da, 0.,
227         offset, 0., 0., da,
228       CURR_BEAM + 15,
229         offset, null,
230         offset, 0,da, 0.,
231         offset, 0., 0., da,
```

```
232       CURR_BEAM + 16,
233         offset, null,
234         offset, 0,da, 0.,
235         offset, 0., 0., da,
236       CURR_BEAM + 17,
237         offset, null,
238         offset, 0,da, 0.,
239         offset, 0., 0., da,
240       CURR_BEAM + 18,
241         offset, null,
242         offset, 0,da, 0.,
243         offset, 0., 0., da,
244       CURR_BEAM + 19,
245         offset, null,
246         offset, 0,da, 0.,
247         offset, 0., 0., da,
248       CURR_BEAM + 20,
249         offset, null,
250         offset, 0,da, 0.,
251         offset, 0., 0., da,
252     # echo, "flap_points.dat", surface, "flap.dat", output, "flap_H.dat",
          order, 2, basenode, 12, weight, 2, stop;
253     mapped points number, 204,
254     sparse mapping file, "flap_H.dat";
255
256 # constant absolute force in node 11
257   force: 2,absolute,
258     2*N + 1,
259     position, null,
260     0., 1., 0.,
261     # slope, initial time, final time / forever, initial value
262     ramp, .0, 0., 1., 0.;
263 end: elements;
```

Listing D.1. MBDyn input file example

Beam nodes

The position of the nodes (see Section 4.1.2) is defined with the following included file.

```
1  structural: curr_node, dynamic,
2    reference, ROOT, (curr_node - 1) * dL, 0.0, 0.0,
3    reference, ROOT, eye,
4    reference, ROOT, null,
5    reference, ROOT, null;
6
7  structural: curr_node + 1,dynamic,
8    reference, ROOT, curr_node * dL, 0.0, 0.0,
9    reference, ROOT, eye,
10   reference, ROOT, null,
```

```
11   reference, ROOT, null;
```

Listing D.2. MBDyn beam nodes

Beam elements and Bodies

beam elements (see Section 4.1.4) and `body` elements (see Section 4.1.5) are defined in the following input file.

```
1  # body: BODY_LABEL, NODE_LABEL,
2  # mass
3  # reference node offset
4  # inertia tensor
5
6  body: 1000+ curr_node, curr_node,
7    m,
8    null,
9    diag, 1./12.*(h^2+w^2)*m, 1./12.*(dL^2+w^2)*m, 1./12.*(dL^2+h^2)*m;
10
11 body: 1000+ curr_node + 1,curr_node + 1,
12   m,
13   null,
14   diag, 1./12.*(h^2+w^2)*m, 1./12.*(dL^2+w^2)*m, 1./12.*(dL^2+h^2)*m;
15
16 # beam 3 nodes
17 # NODE1_LABEL, offset
18 # NODE2_LABEL, offset
19 # NODE3_LABEL, offset
20 # orientation from node
21 # CONSTUTIVE LAW: EA, GAy, GAz, GJ, EJy, EJz
22 #linear elastic generic, diag,
23 #EA, GAy, GAz, GJ, EJy, EJz,
24
25 beam3: 100+curr_node,
26   curr_node - 1,null,
27   curr_node, null,
28   curr_node + 1,null,
29   eye,
30   linear viscoelastic generic,
31     diag, E*A, G*A*5./6., G*A*5./6., G*Jp, E*Jy, E*Jz,
32     proportional, damp,
33   same,
34   same;
```

Listing D.3. MBDyn beam elements

Joints

In order to keep a simulation bidimensional, some `joint` elements (see Section 4.1.6) are applied to nodes as a constraint.

```
joint: 500+curr_node, total joint,
  ROOT,
    position, null,
    position orientation, eye,
    rotation orientation, eye,
  curr_node,
    position, null,
    position orientation, eye,
    rotation orientation, eye,
  position constraint, 0,0,1,null,
  orientation constraint, 1,1,0,null;

joint: 500+curr_node+1, total joint,
  ROOT,
    position, null,
    position orientation, eye,
    rotation orientation, eye,
  curr_node+1,
    position, null,
    position orientation, eye,
    rotation orientation, eye,
  position constraint, 0,0,1,null,
  orientation constraint, 1,1,0,null;
```

Listing D.4. MBDyn joints

D.2 MBDyn output file structure

`.mov` file

For each time step the file contains one row for each node whose output is required. The rows contain the following columns:

- 1: the label of the node
- 2–4: the three components of the position of the node
- 5–7: the three Euler angles that define the orientation of the node
- 8–10: the three components of the velocity of the node
- 11–13: the three components of the angular velocity of the node
- 14–16: the three components of the linear acceleration of the dynamic and modal nodes (optional)
- 17–19: the three components of the angular acceleration of the dynamic and modal nodes (optional)

All the quantities are expressed in the global frame, except for the relative frame type of dummy node, whose quantities are, by definition, in the relative frame.

`.ine` file

This output file refers only to dynamic nodes, and contains their inertia. For each time step, it contains information about the inertia of all the nodes whose output is required. Notice that more than one inertia body can be attached to one node; the information in this file refers to the sum of all the inertia related to the node. The rows contain the following columns:

- 1: the label of the node
- 2–4: item the three components of the momentum in the absolute reference frame
- 5–7: item the three components of the momenta moment in the absolute reference frame, with respect to the coordinates of the node, thus to a moving frame
- 8–10: the three components of the derivative of the momentum
- 11–13: the three components of the derivative of the momentum moment

`.frc` file

An external structural element writes one line for each connected node at each time step in this file. Each line contains the following columns:

- 1: the label of the element and that of the corresponding node; the format of this field is `element_label@node_label`
- 2–4: the three components of the force
- 5–7: the three components of the moment

If a reference node is defined, a special line is output for the reference node, containing the following columns:

- 1: the label of the element and that of the corresponding node; the format of this field is `element_label#node_label`
- 2–4: the three components of the force applied to the reference node, as received from the peer
- 5–7: the three components of the moment applied to the reference node, as received from the peer
- 8–10: the three components of the force that are actually applied to the reference node, in the global reference frame
- 11–13: the three components of the moment with respect to the reference node that are actually applied to the reference node, in the global reference frame
- 14–16: the three components of the force resulting from the combination of all nodal forces, in the global reference frame
- 17–19: the three components of the moment with respect to the reference node resulting from the combination of all nodal forces and moments, in the global reference frame

`.act` file

This type of file contains the output related to beam elements. The internal forces and couples are computed from the interpolated strains along the beam by means of the constitutive law, at the two evaluation points. For each time step and for each element, the format of the columns is:

- 1: the label of the beam
- 2–4: the three components of the force at the first evaluation point
- 5–7: the three components of the couple at the first evaluation point
- 8–10: the three components of the force at the second evaluation point
- 11–13: the three components of the couple at the second evaluation point

`.jnt` file

The output concerning joint elements is generally made of a standard part, plus some extra information depending on the type of joint, which, when available, is described along with the joint description. Here the standard part is described:

- 1: the label of the joint
- 2–4: the three components of the reaction force in a local reference
- 5–7: the three components of the reaction couple in a local frame
- 8–10: the three components of the reaction force in the global frame
- 11–13: the three components of the reaction couple, rotated into the global frame

for total joints:

- 14–16: the three components of the relative displacement in the reference frame of the element
- 17–19: the three components of the relative rotation vector in the reference frame of the element
- 20–22: the three components of the relative velocity in the reference frame of the element
- 23–25: the three components of the relative angular velocity in the reference frame of the element

Acronyms

ALE arbitrary Lagrangian-Eulerian. 3, 5, 6, 16, 24

AME added mass effect. 15, 26, 47

API application programming interface. 29, 35, 39, 93, 99

CFD Computational Fluid Dynamics. xiii, 2, 4, 5, 7, 23, 39, 49, 86, 93

CSM Computational Solid Mechanics. xiii, 5, 39

FEM Finite Element Method. 9, 94

FPI fixed point iteration. 20

FSI Fluid-Structure Interaction. v, vii, 1–3, 5–7, 9, 10, 13, 15–17, 19, 23, 24, 26, 31, 34, 41, 47, 64, 67, 72, 75, 85, 86, 93, 103

IQN-ILS interface quasi Newton with inverse Jacobian from a least squares model. 22, 51, 59, 66, 74, 88

JSON JavaScript Object Notation. 41, 43, 44, 93

MBDyn MultiBody Dynamics analysis software. v, vii, 2, 29, 39, 45, 47, 85, 86, 93, 97

MPI message passing interface. 36

NSE Navier Stokes Equations. 6

PDE Partial Differential Equations. 7

preCICE precise Code Interaction Coupling Environment. v, vii, 2, 29, 35, 39, 45, 48, 85, 93, 95, 97, 99

RBF Radial Basis Function. 25, 51, 59, 66, 74, 88

SSI Solid-Solid Interaction. 94

TCP/IP Transmission Control Protocol/Internet Protocol. 36

VWP Virtual Work Principle. 8, 9

XML extensible markup language. 36, 37, 46

Bibliography

[1] Batra, R. C. *Elements of continuum mechanics*. Aiaa, 2006.

[2] Bertram, A. *Elasticity and plasticity of large deformations*. Springer, 2012.

[3] Blom, D., Lindner, F., Mehl, M., Scheufele, K., Uekermann, B., and van Zuijlen, A. A review on fast quasi-newton and accelerated fixed-point iterations for partitioned fluid–structure interaction simulation. In *Advances in Computational Fluid-Structure Interaction and Flow Simulation*. Springer, 2016, pp. 257–269.

[4] Bodnár, T., Galdi, G. P., and Nečasová, Š. *Fluid-structure interaction and biomedical applications*. Springer, 2014, ch. 13.

[5] Bungartz, H.-J., Lindner, F., Gatzhammer, B., Mehl, M., Scheufele, K., Shukaev, A., and Uekermann, B. precice–a fully parallel library for multi-physics surface coupling. *Computers & Fluids 141* (2016), 250–258.

[6] Causin, P., Gerbeau, J.-F., and Nobile, F. Added-mass effect in the design of partitioned algorithms for fluid–structure problems. *Computer methods in applied mechanics and engineering 194*, 42-44 (2005), 4506–4527.

[7] Chen, S., Wambsganss, M. t., and Jendrzejczyk, J. Added mass and damping of a vibrating rod in confined viscous fluids. In *asme* (1976).

[8] Cheng, J., Zhao, G., Jia, Z., Chen, Y., Wang, S., and Wen, W. Sliding free lagrangian-eulerian finite element method. In *International Conference on Computational Science* (2006).

[9] Conca, C., Osses, A., and Planchard, J. Added mass and damping in fluid-structure interaction. *Computer methods in applied mechanics and engineering 146*, 3-4 (1997), 387–405.

[10] De Boer, A., Van der Schoot, M., and Bijl, H. Mesh deformation based on radial basis function interpolation. *Computers & structures 85*, 11-14 (2007), 784–795.

[11] De Langre, E. *Fluides et solides*. Editions Ecole Polytechnique, 2001.

[12] Degand, C., and Farhat, C. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers & structures 80*, 3-4 (2002), 305–316.

[13] Degroote, J., Bathe, K.-J., and Vierendeels, J. Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. *Computers & Structures 87*, 11-12 (2009), 793–801.

[14] Degroote, J., Bruggeman, P., Haelterman, R., and Vierendeels, J. Stability of a coupling technique for partitioned solvers in fsi applications. *Computers & Structures 86*, 23-24 (2008), 2224–2234.

[15] Degroote, J., Haelterman, R., Annerel, S., Swillens, A., Segers, P., and Vierendeels, J. An interface quasi-newton algorithm for partitioned simulation of fluid-structure interaction. In *International Workshop on Fluid-Structure Interaction. Theory, Numerics and Applications* (2009), kassel university press GmbH, p. 55.

[16] Dettmer, W., and Perić, D. A computational framework for fluid–rigid body interaction: finite element formulation and applications. *Computer Methods in Applied Mechanics and Engineering 195*, 13-16 (2006), 1633–1666.

[17] Donea, J., Huerta, A., Ponthot, J.-P., and Rodríguez-Ferran, A. Arbitrary l agrangian–e ulerian methods. *Encyclopedia of Computational Mechanics Second Edition* (2017), 1–23.

[18] Farhat, C., Van der Zee, K. G., and Geuzaine, P. Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. *Computer methods in applied mechanics and engineering 195*, 17-18 (2006), 1973–2001.

[19] Förster, C., Wall, W. A., and Ramm, E. The artificial added mass effect in sequential staggered fluid-structure interaction algorithms. In *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006* (2006), Delft University of Technology; European Community on Computational Methods ....

[20] Fox, R. W., McDonald, A. T., and Mitchell, J. W. *Fox and McDonald's introduction to fluid mechanics*. John Wiley & Sons, 2011.

[21] Froehle, B., and Persson, P.-O. A high-order discontinuous galerkin method for fluid–structure interaction with efficient implicit–explicit time stepping. *Journal of Computational Physics 272* (2014), 455–470.

[22] Galdi, G. *An introduction to the mathematical theory of the Navier-Stokes equations: Steady-state problems*. Springer Science & Business Media, 2011.

[23] Gatzhammer, B. *Efficient and flexible partitioned simulation of fluid-structure interactions*. PhD thesis, Technische Universität München, 2014.

[24] Gauthier, J., Giroux, A., Etienne, S., and Gosselin, F. A numerical method for the determination of flow-induced damping in hydroelectric turbines. *Journal of Fluids and Structures 69* (2017), 341–354.

[25] Ghiringhelli, G. L., Masarati, P., and Mantegazza, P. Multibody implementation of finite volume c beams. *AIAA journal 38*, 1 (2000), 131–138.

[26] Ghiringhelli, G. L., Masarati, P., Morandini, M., and Muffo, D. Integrated aeroservoelastic analysis of induced strain rotor blades. *Mechanics of Advanced Materials and Structures 15*, 3-4 (2008), 291–306.

[27] Giavotto, V., Borri, M., Mantegazza, P., Ghiringhelli, G., Carmaschi, V., Maffioli, G., and Mussi, F. Anisotropic beam theory and applications. *Computers & Structures 16*, 1-4 (1983), 403–413.

[28] Gjertsen, S. Development of a verified and validated computational framework for fluid-structure interaction: Investigating lifting operators and numerical stability. Master's thesis, 2017.

[29] González, A. O., Vallier, A., and Nilsson, H. Mesh motion alternatives in openfoam. *PhD course in CFD with OpenSource software* (2009).

[30] Habchi, C., Russeil, S., Bougeard, D., Harion, J.-L., Lemenand, T., Ghanem, A., Della Valle, D., and Peerhossaini, H. Partitioned solver for strongly coupled fluid–structure interaction. *Computers & Fluids 71* (2013), 306–319.

[31] Haelterman, R., Bogaers, A. E., Scheufele, K., Uekermann, B., and Mehl, M. Improving the performance of the partitioned qn-ils procedure for fluid–structure interaction problems: Filtering. *Computers & Structures 171* (2016), 9–17.

[32] Haelterman, R., Degroote, J., Van Heule, D., and Vierendeels, J. The quasi-newton least squares method: a new and fast secant method analyzed for linear systems. *SIAM Journal on numerical analysis 47*, 3 (2009), 2347–2368.

[33] Hanche-Olsen, H. Buckingham's pi-theorem. *NTNU: http://www. math. ntnu. no/˜ hanche/notes/buckingham/buckingham-a4. pdf* (2004).

[34] Hardtke, J.-D. On buckingham's π-theorem. *arXiv preprint arXiv:1912.08744* (2019).

[35] Heathcote, S., Wang, Z., and Gursul, I. Effect of spanwise flexibility on flapping wing propulsion. *Journal of Fluids and Structures 24*, 2 (2008), 183–199.

[36] Hjelmstad, K. D. *Fundamentals of structural mechanics*. Springer Science & Business Media, 2007.

[37] Hodges, D. H. Review of composite rotor blade modeling. *AIAA journal 28*, 3 (1990), 561–565.

[38] Hong, J.-H., and Arai, N. Fluid-structure interaction under the vortex lock-in regime. In *Computational Fluid Dynamics 2000*. Springer, 2001, pp. 601–606.

[39] Hou, G., Wang, J., and Layton, A. Numerical methods for fluid-structure interaction—a review. *Communications in Computational Physics 12*, 2 (2012), 337–377.

[40] Hübner, B., Walhorn, E., and Dinkler, D. A monolithic approach to fluid–structure interaction using space–time finite elements. *Computer methods in applied mechanics and engineering 193*, 23-26 (2004), 2087–2104.

[41] Irons, B. M., and Tuck, R. C. A version of the aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering 1*, 3 (1969), 275–277.

[42] Kajishima, T., and Taira, K. Immersed boundary methods. In *Computational Fluid Dynamics*. Springer, 2017, pp. 179–205.

[43] Kassiotis, C., Ibrahimbegovic, A., Niekamp, R., and Matthies, H. G. Nonlinear fluid–structure interaction problem. part i: implicit partitioned algorithm, nonlinear stability proof and validation examples. *Computational Mechanics 47*, 3 (2011), 305–323.

[44] Kim, T., Hansen, A. M., and Branner, K. Development of an anisotropic beam finite element for composite wind turbine blades in multibody system. *Renewable Energy 59* (2013), 172–183.

[45] Küttler, U., and Wall, W. A. Fixed-point fluid–structure interaction solvers with dynamic relaxation. *Computational mechanics 43*, 1 (2008), 61–72.

[46] Lindner, F., Mehl, M., Scheufele, K., and Uekermann, B. A comparison of various quasi-newton schemes for partitioned fluid-structure interaction. In *Proceedings of 6th International Conference on Computational Methods for Coupled Problems in Science and Engineering, Venice* (2015), pp. 1–12.

[47] Lindner, F., Mehl, M., and Uekermann, B. Radial basis function interpolation for black-box multi-physics simulations.

[48] Lipton, S., Evans, J. A., Bazilevs, Y., Elguedj, T., and Hughes, T. J. Robustness of isogeometric structural discretizations under severe mesh distortion. *Computer Methods in Applied Mechanics and Engineering 199*, 5-8 (2010), 357–373.

[49] Longo, S. *Analisi Dimensionale e Modellistica Fisica: Principi e applicazioni alle scienze ingegneristiche*. Springer Science & Business Media, 2011.

[50] Masarati, P. A formulation of kinematic constraints imposed by kinematic pairs using relative pose in vector form. *Multibody System Dynamics 29*, 2 (2013), 119–137.

[51] Masarati, P., Morandini, M., and Mantegazza, P. An efficient formulation for general-purpose multibody/multiphysics analysis. *Journal of Computational and Nonlinear Dynamics 9*, 4 (2014).

[52] Matthies, H. G., and Steindorf, J. Partitioned strong coupling algorithms for fluid–structure interaction. *Computers & structures 81*, 8-11 (2003), 805–812.

[53] Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., and Van Zuijlen, A. Parallel coupling numerics for partitioned fluid–structure interaction simulations. *Computers & Mathematics with Applications 71*, 4 (2016), 869–891.

[54] Ogden, R. W. *Non-linear elastic deformations*. Courier Corporation, 1997.

[55] Olivier, M., Morissette, J.-F., and Dumas, G. A fluid-structure interaction solver for nano-air-vehicle flapping wings. In *19th AIAA Computational Fluid Dynamics*. 2009, p. 3676.

[56] Pope, S. B. Turbulent flows, 2001.

[57] Qin, Z., and Suckale, J. Direct numerical simulations of gas–solid–liquid interactions in dilute fluids. *International Journal of Multiphase Flow 96* (2017), 34–47.

[58] Quartapelle, L., and Auteri, F. *Fluidodinamica comprimibile*. Casa editrice ambrosiana, 2013.

[59] Quartapelle, L., and Auteri, F. *Fluidodinamica incomprimibile*. Casa editrice ambrosiana, 2013.

[60] Ramm, E., and Wall, W. Fluid-structure interaction based upon a stabilized (ale) finite element method. In *4th World Congress on Computational Mechanics: New Trends and Applications, CIMNE, Barcelona* (1998), pp. 1–20.

[61] Ricciardi, G., and Boccaccio, E. Modelling of the flow induced stiffness of a pwr fuel assembly. *Nuclear Engineering and Design 282* (2015), 8–14.

[62] Richter, T. *Fluid-structure interactions: models, analysis and finite elements*, vol. 118. Springer, 2017.

[63] Richter, T., and Wick, T. On time discretizations of fluid-structure interactions. In *Multiple Shooting and Time Domain Decomposition Methods*. Springer, 2015, pp. 377–400.

[64] Ryzhakov, P., Rossi, R., Idelsohn, S., and Oñate, E. A monolithic lagrangian approach for fluid–structure interaction problems. *Computational mechanics 46*, 6 (2010), 883–899.

[65] Shukaev, A. K. A fully parallel process-to-process intercommunication technique for precice. Master's thesis, Technische Universitt München, June 2015.

[66] Turek, S., and Hron, J. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In *Fluid-structure interaction*. Springer, 2006, pp. 371–385.

[67] Turek, S., Hron, J., and Razzaq, M. Numerical benchmarking of fluid-structure interaction between elastic object and laminar incompressible *Universitätsbibliothek Dortmund* (2010).

[68] Uekermann, B., Bungartz, H.-J., Cheung Yau, L., Chourdakis, G., and Rusch, A. Official precice adapters for standard open-source solvers. In *Proceedings of the 7th GACM Colloquium on Computational Mechanics for Young Scientists from Academia* (2017).

[69] Uekermann, B., Bungartz, H.-J., Gatzhammer, B., and Mehl, M. A parallel, black-box coupling algorithm for fluid-structure interaction. In *Proceedings of 5th International Conference on Computational Methods for Coupled Problems in Science and Engineering* (2013), pp. 1–12.

[70] Uekermann, B. W. *Partitioned fluid-structure interaction on massively parallel systems*. PhD thesis, Technische Universität München, 2016.

[71] van Brummelen, E. H. Added mass effects of compressible and incompressible flows in fluid-structure interaction. *Journal of Applied mechanics 76*, 2 (2009).

[72] van Loon, R., Anderson, P. D., van de Vosse, F. N., and Sherwin, S. J. Comparison of various fluid–structure interaction methods for deformable bodies. *Computers & structures 85*, 11-14 (2007), 833–843.

[73] Walhorn, E., Hübner, B., and Dinkler, D. Space-time finite elements for fluid-structure interaction. In *PAMM: Proceedings in Applied Mathematics and Mechanics* (2002), vol. 1, Wiley Online Library, pp. 81–82.

[74] Wood, C., Gil, A., Hassan, O., and Bonet, J. Partitioned block-gauss–seidel coupling for dynamic fluid–structure interaction. *Computers & structures 88*, 23-24 (2010), 1367–1382.

[75] Xing, J. T. Chapter 3 - fundamentals of continuum mechanics. In *Fluid-Solid Interaction Dynamics*, J. T. Xing, Ed. Academic Press, 2019, pp. 57 – 101.