LIÈGE université

Efficient and flexible implementation of an interfacing Python-based tool for numerical simulations of fluid-structure interaction problems

PhD Thesis
submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Engineering Sciences

by David Thomas
Supervisor : Professor V.E. Terrapon

2020

My thesis is written in
ow!
Blood
Sweat
what am i doing with my life??
Tears
JORGE CHAM © 2013
and COFFEE.
WWW.PHDCOMICS.COM

Abstract

Applications involving a solid structure actively interacting with a surrounding fluid, referred to as fluid-structure interaction (FSI), are constantly gaining interest in fundamental or industrial research and development. Such a multi-physics system is characterized by a complex dynamics which requires technically advanced methods to be studied. An accurate understanding of FSI systems becomes a key factor in the current trend of producing lighter and more flexible industrial designs. Nowadays, numerical simulations are extensively used to produce high-fidelity FSI models for various engineering applications. The typical approach considered in this thesis performs the coupling between two independent solvers, each of them being specifically designed to solve the governing equations for one particular physics. In this context, this thesis presents the development of an original coupling environment called CUPyDO. It is designed for the coupling of independent single-physics solvers within a unified architecture based on a Python wrapping methodology.

The development of CUPyDO has been conducted with the aim of producing a tool that leverages computing efficiency and accuracy, coupling flexibility and user-friendliness while circumventing some limitations and improving some other aspects of the existing solutions. The tool gathers state-of-the-art coupling methods which are deeply reviewed in this work. This includes major capabilities such as under-relaxed coupling algorithms, non-matching interface meshes and multi-core parallelization. The modern Python wrapping technology used in CUPyDO provides easy access to all the features through a high-level API which can be further customized and extended with minimal effort. Furthermore, the same wrapping technology provides a flexible black-box coupling with various existing standalone solvers without any architectural or algorithmic adaptation to be performed in the coupling environment.

The implementation and the capabilities of CUPyDO are demonstrated and verified by simulating several FSI test cases for transonic aeroelastic flutter, vortex-induced vibrations and conjugate heat transfer. The test cases are also used to assess the coupling tool in terms of robustness, efficiency and accuracy. The results are successfully validated against experimental and numerical reference data found in the literature, and some usability guidelines are drawn. Furthermore, the coupling flexibility is highlighted by using various structural solvers through the different test cases.

The coupling tool is used for the aeroelastic study of a very flexible flat plate wing with various geometrical configurations. In a first step, the linear flutter velocity and frequency are sought and compared with experimental data with good agreement. In a second step, the post-flutter aeroelastic response is simulated for some configurations of the plate and it is found that the system is characterized by a supercritical Hopf bifurcation. For one particular plate configuration, two distinct post-flutter limit cycle oscillations are observed depending on the initial structural perturbation and on the occurrence of aerodynamic nonlinearities. This further highlights the importance of using high-fidelity coupled models for representing nonlinear aeroelastic solution that reduced-order linear models are not able to predict.

Acknowledgements

It would have not been possible to push this thesis to its end without the help and the support of many kind people. This work owes a part of itself to each of them.

I would first like to express my gratitude to my advisor Prof. Vincent E. Terrapon and my co-advisor Prof. Grigorios Dimitriadis for giving me the opportunity to perform my PhD in the fields of computational aeroelasticity. I sincerely thank them for their valuable support and advice, and for helping me exceeding my own limits. I would also like to thank the other members of the jury, Prof. Christophe Geuzaine, Prof. Rafael Palacios, Prof. Joris Degroote, Dr. Romain Boman and Dr. Thomas Andrianne, for their time spent reading and evaluating this work.

I gratefully acknowledge Dr. Romain Boman for teaching me the basis of code wrapping, which is the founding element of this thesis, and for his work on the Metafor interface. I thank him and Dr. Luc Papeleux for their incredibly efficient technical advice. I also acknowledge Dr. Marco Lucio Cerquaglia for his valuable collaboration on improving and extending the capabilities of CUPyDO, and for the memorable time we spent at the Coupled Problems 2017 conference. I thank Prof. Christophe Geuzaine for his valuable contribution to the GetDP interface.

I would like to sincerely acknowledge Prof. Juan J. Alonso, Dr. Thomas Economon and all the Aerospace Design Lab at Stanford University for giving me the opportunity to spent a research stay in their team. The work achieved and the people I met there are now parts of my best memories. I would also like to thank the SU2team for its technical support and for the given opportunity to contribute to the development of the solver.

I am very thankful to all my co-workers at MTFC research group, especially Adrien Crovato, Dr. Hüseyin Güner, Samir Sid and Dr. Amandine Guissart, for the pleasant time we spent together and their helpful support when things were not going as expected. I also thank my new colleagues at NUMECA International for their advice and their interest in this work.

Last but not least, I address my heartfelt thanks to my family and my friends. Thank you for enlightening my path when it was too dark for me to keep going. To my parents and my brother, thank you for always being so supportive and kind. To my beloved Linh, thank you for sharing my life, my joy, my grief, and for giving me endless love, care and support. You raised my head up when everything was about to collapse. I owe you so much. To my baby girl Emi, who is enlightening my life with new colours and flavours. You gave me the willpower to finish this work.

Contents

List of abbreviations

AD Algorithmic Differenciation

ADT Alternating Digital Tree

ALE Arbitrary Lagrangian-Eulerian

AoA Angle of Attack

API Application Programming Interface

BEM Boundary Elements Method

BGS Block-Gauss-Seidel

CFD Computational Fluid Dynamics

CFL Courant Friedrichs-Lewy (number)

CHT Conjugate Heat Transfer

CSD Computational Solid Dynamics

DN Dirichlet-Neumann

FEM Finite Elements Method

FFTB Flux Forward Temperature Back

FIV Fluid-Induced Vibrations

FSI Fluid-Structure Interaction

FVM Finite Volume Method

HPC High Performance Computing

IB Immersed Boundary

IBQN Interface Block Quasi-Newton

ILS Inverse Least Square

IP Internet Protocol

IQN Interface Quasi-Newton

MPI Message Passing Interface

ND Neumann-Dirichlet

NN Nearest Neighbor

RBF	Radial Basis Function
TCP	Transmission Control Protocol
TFFB	Temperature Forward Flux Back
(U)RANS	(Unsteady) Reynolds-Averaged Navier-Stokes equations
VIV	Vortex-Induced Vibrations

Mathematical notations

General notation

Scalar notation uses both lower or uppercase symbols with no other significant font style than the default mathematical emphasis:

$$a, A, \omega, \Omega.$$

Vector and first-order tensor notation uses bold symbols with mathematical emphasis:

$$\boldsymbol{v}, \boldsymbol{V}, \boldsymbol{\gamma}.$$

Matrix and second-order tensor notation uses bold symbols with straight font:

$$\mathbf{v}, \mathbf{V}, \mathbf{\Gamma}.$$

Any other specific and non-general notations shall be detailed at occurrence.

Indices and other operators

Indicial notation may be used for vectors, v_i and matrices, V_{ij}. The Einstein summation convention is used when applicable:

$$a_i b_i = \sum_k a_k b_k \,,$$
$$A_{ij} b_j = \sum_k A_{ik} b_k \,.$$

Total and partial derivatives shall be expressed by the following notation:

$$\frac{da_i}{dx} = d_x a_i \,,$$
$$\frac{\partial a_i}{\partial x} = \partial_x a_i \,.$$

and the Einstein convention may be applicable:

$$\partial_j A_{ij} = \sum_k \partial_k A_{ik} \,.$$

The gradient of a scalar in cartesian coordinates is expressed as

$$\nabla a = \begin{bmatrix} \partial_x a \\ \partial_y a \\ \partial_z a \end{bmatrix} .$$

The divergence of a tensor is expressed as

$$\nabla \cdot \boldsymbol{a} = \partial_i a_i$$

and

$$\nabla \cdot \mathbf{A} = \begin{bmatrix} \partial_j A_{xj} \\ \partial_j A_{yj} \\ \partial_j A_{zj} \end{bmatrix}$$

for vectors and matrices, respectively. Finally, the Laplacian operator is expressed as

$$\nabla^2 a = \partial_i^2 a$$

and

$$\nabla^2 \boldsymbol{v} = \begin{bmatrix} \nabla^2 v_x \\ \nabla^2 v_y \\ \nabla^2 v_z \end{bmatrix}$$

for scalars and vectors, respectively.

List of symbols

Latin symbols

a	m s^{-1}	Speed of sound
Bi	[-]	Biot number
c	N s m^{-1} | m	Damping coefficient | Airfoil chord
c_p	J kg^{-1} K^{-1}	Specific heat at constant pressure
c_v	J kg^{-1} K^{-1}	Specific heat at constant volume
C_D	N	Drag coefficient
C_L	N	Lift coefficient
$\boldsymbol{d}$	m	Displacement
D	m | N	Diameter | Drag force
e	J kg^{-1}	Internal energy per unit mass
E	J kg^{-1} | Pa	Total energy per unit mass | Young's modulus
$\mathbf{E}$	[-]	Strain tensor
f	s^{-1}	Frequency
$\tilde{\boldsymbol{f}}$	N kg^{-1}	Force per unit mass
$\hat{\boldsymbol{f}}$	N m^{-3}	Force per unit volume
F	N	Force
$\mathbf{F}$	[-]	Deformation gradient tensor
h	W m^{-2}K^{-1} | m	Convective heat transfer coefficient | Plunge displacement
I	kg m^2	Moment of inertia
$\mathbf{I}$	[-]	Unity tensor
k	N m^{-1} | m^2 s^{-2}	Stiffness | Turbulent kinetic energy
L	m | N	Characteristic length | Lift force
m	kg	Mass
M	[-]	Mach number
Ma	[-]	Mass ratio
p	Pa	Pressure
P	Pa	Characteristic pressure
Pr	[-]	Prandtl number
q	J s m | m	Heat generation per unit volume | Generalized displacement
$\boldsymbol{r}$	[-]	Residual
R	J kg^{-1} K^{-1}	Specific gas constant
Re	[-]	Reynolds number
s	m	Span
Str	[-]	Strouhal number
t	s | m	Time | Thickness
$\boldsymbol{t}$	N	Surface load
T	K	Temperature
$\boldsymbol{v}$	m s^{-1}	Velocity
U	m s^{-1}	Characteristic velocity
$\boldsymbol{x}$	m	Cartesian coordinates

Greek symbols

α	m/s^2 | rad	Thermal diffusivity | Pitch angle
β	K^{-1}	Volumetric thermal expansion coefficient
γ	[-]	Specific heats ratio
δ	[-]	Kronecker delta
Γ	[-]	Fluid-structure interface
Δ	[-]	Difference operator
ζ	[-]	Damping ratio
λ	W m^{-1} K^{-1}	Thermal conductivity
Λ	.$^\circ$	Sweep angle
μ	Pa s	Dynamic viscosity
ν	m^2 s^{-1} | [-]	Kinematic viscosity | Poisson coefficient
ρ	kg m^{-3}	Density
$\boldsymbol{\tau}$	Pa	Shear stress tensor
$\boldsymbol{\sigma}$	Pa	Cauchy stress tensor
ω	rad/s | s^{-1} | [-]	Frequency | Specific energy dissipation rate | Relaxation parameter
Ω	[-]	Spatial domain
∇	m^{-1}	Nabla operator

Introduction

Context

Fluid-structure interaction (FSI) refers to phenomena that involve a structure interacting with a surrounding fluid. A wide range of fields and applications, such as aerospace, civil engineering or biomedical are concerned by FSI problems. FSI systems have gained increasing interest from engineers and scientists because they usually involve very complex dynamics that are still not fully understood, making efficient designs even more challenging. In this particular context, history has taught us that FSI can lead to disastrous consequences if not properly integrated in the design process. One of the most cited disaster directly linked to FSI problem is the collapse of the Tacoma Narrow bridge in 1940. In the aerospace industry, we observe a clear trend towards larger (see Fig. 1), lighter and more flexible (e.g., B787 and A350 in Fig. 1 are mostly based on composites) designs over the years. For such designs, FSI dynamics of the wings may strongly impact the global performance of an aircraft.

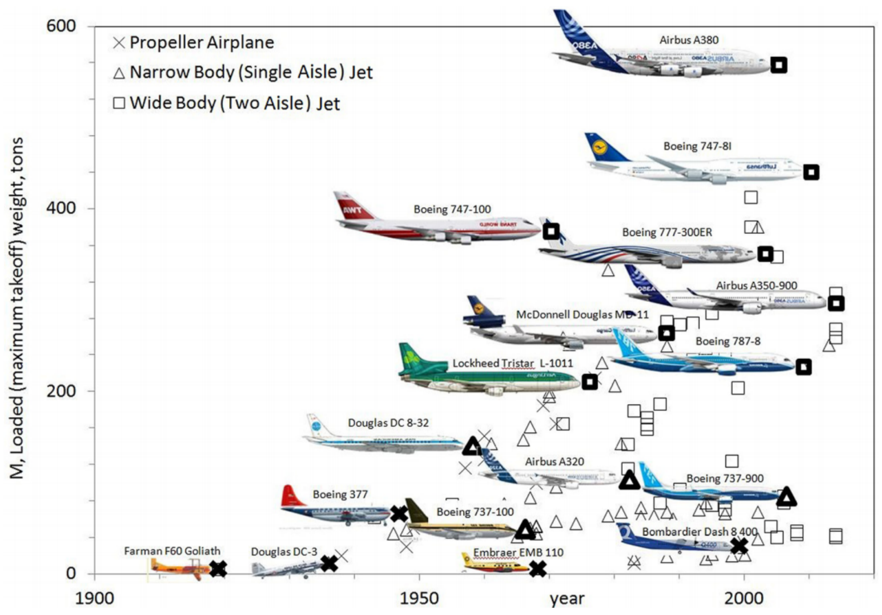


Figure 1: Illustration of the continuous increase in aircraft take-off weight over the century. Taken from Bejan *et al.* [1]. On the right extremity of the graph, we note the presence of the B787 and A350 models which are known to integrate a significant part of lighter and more flexible composite materials.

Research study or design of FSI systems usually leverages two main approaches. On the one hand, valuable real-life data can be acquired by experimental investigations. Although their use tends to be minimized because of their relatively high cost, experimental models and prototyping remain truly necessary in some phases of the design process (e.g. during certification). On the other hand, the use of computational models has intensified over the past decades, following the continuous increase in the average computer power. Nowadays, computational models or solvers are used on a daily basis as they usually represent a smaller cost compared to experiments. A wide range of high-fidelity solvers have been developed by industries and research centers, either on a commercial or open-source basis. Most of the time, these models are able to solve the governing equations, both spatially and temporally discretized, for one single physics. For instance, a structural code is used to compute the deformations and the stresses within a solid subjected to specified loads but has no capabilities to compute a fluid flow. Conversely, fluid solvers are specifically designed to compute a fluid flow but cannot be used to solve structural dynamics. For FSI applications, a segregated use of these solvers without considering the effects of the coupling (or the interaction between the two physics) cannot accurately represent the complexity of FSI systems, and coupled computational models must then be developed and considered for reliable designs. This however comes with additional challenges that are related to the coupling technology and algorithm, the numerical stability of that algorithm, solvers inter-communication, or the difference in the scaling of the physical characteristics between the two coupled fields.

In this particular context, this thesis presents the development of an original computational tool, CUPyDO, designed as a coupling environment of single-physics solvers (fluid and solid) for FSI applications. A partitioned strategy is selected, where the coupled framework takes direct benefit from the existing technologies that are already implemented in each coupled solver. CUPyDO is designed based on modern programming techniques, such the Object-Oriented formalism, that combines two standard and efficient programming languages that are C++ and Python. In the development of CUPyDO emphasis is given to computing efficiency, flexibility and user-friendliness.

Motivation and objectives

The development of CUPyDO is seen as a result of a detailed analysis of what can be found in the current state-of-the-art to computationally solve coupled problems. While the development of a new coupling tool cannot be considered as a technological breakthrough, this thesis proposes a true modern alternative that can circumvent some limitations and improve some other aspects of the existing solutions. The technological specificity of CUPyDO, based on a Python wrapping of C++ core codes, significantly improves the interfacing flexibility and the usability without altering the efficiency.

The main objective followed in this work is to develop a coupling environment that provides a minimal amount of basic capabilities such as standard state-of-the-art coupling algorithms and interpolation schemes for non-matching interface meshes. Access to these capabilities must be given through a high-level user-friendly API while keeping the code as open as possible for customization, which is truly powered by an efficient wrapping technology. When coupling new existing codes to a third-party environment, the question of code adaptation also comes rapidly into play. In this work, emphasis is given to minimizing development effort to couple a given solver by defining a flexible interfacing procedure, again powered by Python wrapping, that limits source code invasiveness and increases coupling flexibility and modularity.

The coupling environment is intended to be used by both academic (students and researchers) and industrial practitioners. The tool must thus allow user customization while hiding the technical and technological aspects as much as possible for students and scientists conducting fundamental research. As the average size and complexity of simulated problems is always increasing, parallelization and HPC-compliance is also a *must have* for any viable computational tool.

The development of CUPyDO must be considered with a long-term perspective that surely goes beyond the time window of this thesis. The use of modern formalisms such as Object-

Oriented Programming (OOP), which maximizes code re-usability, and Pyhon wrapping is motivated by the need for maintainability and expandability of the code. An open-source license is also considered as an opportunity to grow the code via contributions of an entire, world-wide, community.

In addition to the development of the coupling environment, this thesis is dedicated to its validation and verification. By using different open-source fluid and solid solvers to be coupled, the different features and capabilities that resulted from the aforementioned strategy must be demonstrated on representative and documented computational cases involving FSI problems.

Innovation and contributions

Although CUPyDO is based on current state-of-the-art numerical algorithms, the innovative aspect lies in its implementation and coupling technology. In particular, the Python-wrapping methodology is leveraged in order to develop a unified coupling framework in which data are transferred via the memory space and that does not need to rely on TCP/IP. This allows the user to launch a coupled simulation as it would be a single executable. Furthermore, data and functionalities are exposed from the coupled solvers to the coupling tool, and not the opposite as usually encountered in existing coupling software. Hence, there is no specific CUPyDO routines to be introduced in the coupled code, which truly highlights its minimally intrusive nature. This also relieves some design constraints regarding the coupled Python wrappers because they do not need to be FSI-oriented and can be reused for many other applications than only achieving the coupling with CUPyDO. The original C++/Python dual nature of CUPyDO confines the set-up and the customization of the coupled simulation at a user-friendly and intuitive high level (Python) while the computationally intensive tasks are hidden and efficiently executed at the lower-level (C++). This is particularly leveraged by the OOP architecture of CUPyDO in which each component is designed for one specific coupling task, such as solver synchronisation, time marching, interface interpolation and parallel data structure management, to cite the most important ones. Such framework architecture also contributes to maintain a high level of code re-usability, maintainability and expandability.

The source code of CUPyDO can be found in the following Github repository: `https://github.com/ulgltas/CUPyDO`, under the Apache 2.0 open-source license. By means of its intuitive architecture and flexible coupling methodology, the list of the current available compatible solvers is already larger than the few solvers mentioned in this work. In addition, because CUPyDO is implemented with state-of-the-art numerical models, it has already been successfully used (and still is) in many other research projects:

- Newton-based FSI algorithms have been implemented for FSI applications with strong added-mass effects. This involves the coupling with a particle finite element (PFEM) code for the fluid part. The targeted applications were simulations of bird impact and FSI with free-surface flows [2, 3], as illustrated in Fig. 2(a).

- The FLOW code, a FEM-based steady solver for the full compressible potential flow equation, has been coupled with CUPyDO for steady aeroelastic analysis of composite wings in transonic regime [4, 5], as depicted in Fig. 2(b).

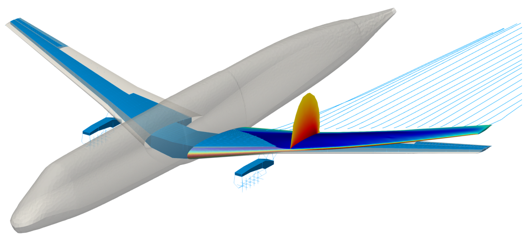
- The ability of representing the solid structure through a modal decomposition instead of a full FEM model has been enabled by coupling a modal solver (Modali) with CUPyDO. The coupling between Modali and SU2 has been used to validate lower-fidelity methods for flutter calculations of composite wings in transonic regime [6, 7].

- The coupling capabilities of CUPyDO have recently been extended towards adjoint formulation for calculation of steady FSI cases. For this purpose, the adjoint structural solver of the SU2 package has been independently coupled with the adjoint SU2 fluid solver through

CUPyDO. This extension of CUPyDO to FSI adjoint calculation is being developed in close collaboration with the SU2 development team.

- A Vortex Latice Model (VLM) code has also been coupled to CUPyDO for fast, low-fidelity, FSI computations.

(a) Dam break against an elastic–plastic obstacle. Free-surface flow simulated with the PFEM code and coupled with Metafor [2].



(b) Steady aeroelastic simulation of the Embrear Benchmark Wing. The flow is computed with FLOW and coupled with Modali [5].

Figure 2: Examples of FSI applications involving the coupling with CUPyDO in other research projects.

CUPyDO is now regularly used for both steady and unsteady FSI in several research projects and also in collaboration with a major industrial partner (Embraer). Ongoing research works mainly focus on extending the FLOW code to perform unsteady FSI (possibly steady adjoint as well), and extending the steady adjoint calculation capability to unsteady adjoint based on a harmonic balance approach. In a longer term, the fluid code OpenFOAM and the structural solver TACS [8] are envisaged for further enlarging the list of coupled solvers, together with extending the coupling capabilities to true unsteady adjoint.

Associated publications

D. Thomas, M.L. Cerquaglia, R. Boman, T.D. Economon, J.J Alonso, G. Dimitriadis, V.E. Terrapon. CUPyDO- An integrated Python environment for coupled fluid-structure interaction. *Advances in Engineering Software*, 128:69-85, 2019. doi:10.1016/j.advengsoft.2018.05.007

M.L. Cerquaglia, D.Thomas, R.Boman, V.Terrapon, J.-P. Ponthot. A fully partitioned Lagrangian framework for FSI problems characterized by free surfaces, large solid deformations and displacements, and strong added-mass effects. *Computer Methods in Applied Mechanics and Engineering*, 348:409-442, 2019. doi:10.1016/j.cma.2019.01.021

H. Güner, D. Thomas, G. Dimitriadis, V.E. Terrapon. Unsteady aerodynamic modeling methodology based on dynamic mode interpolation for transonic flutter calculations. *Journal of Fluids and Structures*, 84:218-232, 2019. doi:10.1016/j.jfluidstructs.2018.11.002

D. Thomas, A. Variyar, R. Boman, T.D. Economon, J.J. Alonso, G. Dimitriadis, V.E. Terrapon. Staggered strong coupling between existing fluid and solid solvers through a Python interface for fluid-structure interaction problems. *Proceedings of the VII International Conference on Computational Methods for Coupled problems in science and engineering, Coupled Problems 2017*, 12-14 June 2017, Rhodes Island, Greece. Proceedings.

R. Sanchez, H.L. Kline, D. Thomas, A. Variyar, M. Righi, T.D. Economon, R. Sanchez, H.L. Kline, D. Thomas, A. Variyar, M. Righi, T.D. Economon, J.J. Alonso, R. Palacios, G. Dimitriadis, and V. Terrapon. Assessment of the fluid-structure interaction capabilities for aeronautical applications of the open-source solver SU2. *Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2016*, June 2016 Crete Island, Greece. doi:10.7712/100016.

Outline

This thesis is divided into three parts that are composed of three, two and two chapters, respectively. In the first part, Chapter 1 gives an overview of the FSI problem and introduces the main phenomenological aspects. Chapter 2 focuses on general mathematical models for FSI, by describing the governing equations for each coupled physics and then presenting the interface conditions. Chapter 3 is dedicated to all the numerical aspects of FSI. It presents the different methodologies for building a FSI model, the coupling algorithms, the specific numerical treatment of the fluid-structure interface, and the different solvers that are used in this work. In the second part, existing code coupling technologies are reviewed in Chapter 4 and eligibility for the development of a new coupling framework is established. The detailed implementation of CUPyDO is given in Chapter 5 by covering all the aspects introduced in the previous chapters. The last part of this thesis is used to validate the coupling tool, in Chapter 6, by using it to simulate standard FSI cases referenced in the scientific literature. The last Chapter 7 proposes an application of CUPyDO for the aeroelastic study of a very flexible thin plate wing. Finally, the conclusion summarizes the main findings of this work and proposes several avenues for future work.

Part I

The Fluid-Structure Interaction problem

Chapter 1

An overview of the fluid-structure interaction problem

This first chapter introduces the fundamental basis for the physical understanding of the fluid-structure interaction problem, with a focus on phenomenological aspects. A fluid-structure interaction problem (FSI) refers to the system formed by a solid body in contact with a fluid where the dynamics of each type of physics is mutually coupled through a common interface. Studying FSI systems is of major interest for engineers because such systems are present in many fields of applications such as aerospace, civil engineering, biomedical/biological, electronics, energy or food processing. One can imagine many situations in which a structure in contact with a fluid will be subjected to fluid-induced loads or fluid-induced thermal constraints.

Hydro- or aeroelasticity studies the action of a fluid (liquid or gas) on a flexible solid structure that could move or deform under the actions of the fluid loads. The motion or deformation of the solid has a feedback influence on the fluid, resulting in a change of the loading pattern so that the dynamics of the coupled system is ruled by this mutual interaction. Aeroelastic systems are the coupling of the elastic and inertial forces in the solid with the fluid loads, as illustrated in Fig. 1.1.

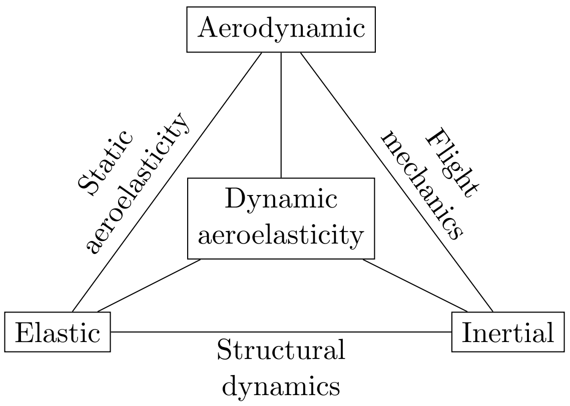


Figure 1.1: Collar's aeroelastic triangle: force interactions in an aeroelastic system.

Conjugate Heat Transfer (CHT) describes the heat transfer occurring between a solid and a surrounding fluid usually having a significant temperature difference. The heat conduction in the solid is coupled to the heat convection (convection = combination of conduction by molecular diffusion and advection by macroscopic transport) in the fluid. Forced convection, as shown in Fig. 1.2, occurs when the fluid motion is caused by externals means, such as a pump or the relative motion of the solid. On the other hand, free or natural convection takes place when the fluid motion is caused by buoyancy forces induced by a density gradient submitted to gravity, i.e., induced by the heat transfer itself.

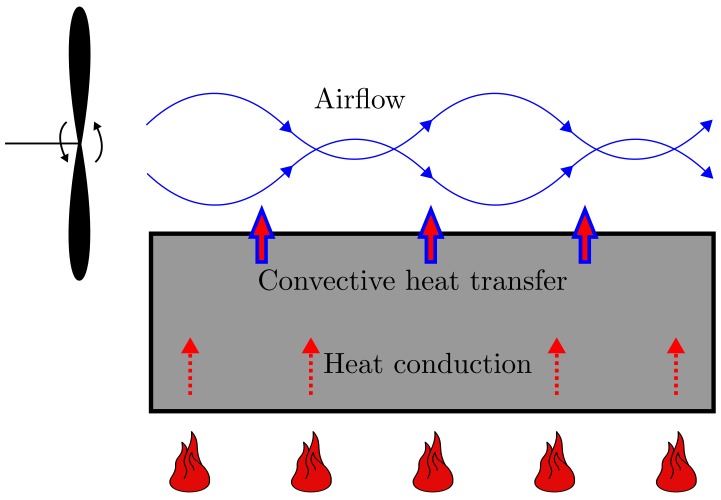
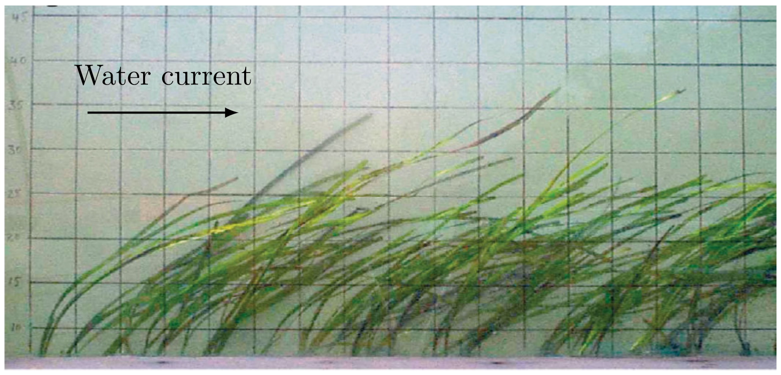


Figure 1.2: Fan-generated airflow for hot plate cooling by forced convection.

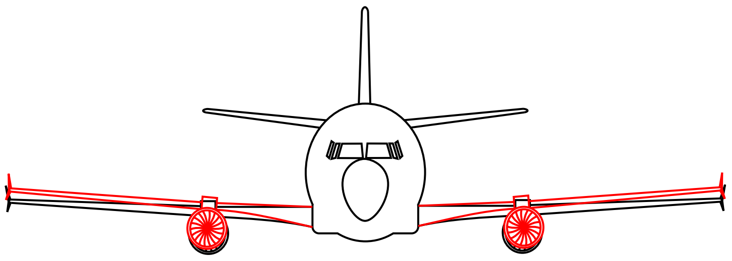
Remark
In this thesis, the *FSI* terminology includes both mechanical and thermal (CHT) interactions, although in the literature this terminology usually refers only to the mechanical interaction while CHT applies to thermal interaction.

1.1 Classification of fluid-structure interaction phenomena

There are many ways to classify FSI problems according to the particular dynamics and phenomenology involved in the process. As a start, we could distinguish static from dynamic FSI problems. Fluid-solid systems whose flow and structural variables (flow velocity field, structural displacement, etc) do not evolve in time are qualified as static FSI problems. The Collar's triangle in Fig. 1.1 may represent a static system after removing the inertial forces in the structure. A typical example can be observed in nature where aquatic river plants are bent by the action of a stabilized water current, as illustrated in Fig. 1.3(a). If the flow is maintained at a constant speed and does not feature significant unsteady mechanisms such as vortex shedding, the deflection of the plants, known as vegetation reconfiguration, will remain constant in time. A change in the flow rate will bring the system to another configuration through a transition phase but a new static equilibrium will be reached as long as the flow rate is maintained constant at its new value. Another example of static FSI is the in-flight deflection of aircraft wings under the action of the lift force. Recent designs use lighter and more flexible materials for the aircraft structures. The resulting significant flexibility of large span wings causes larger in-flight lift-induced deflection, as illustrated in Fig. 1.3(b), whose impact on the flight performance has to be taken into account during the design process. The design of wind turbine blades may also be strongly impacted by static aeroelasticity. Note that unsteady effects might also be important to consider but the initial phase of the design generally focuses on steady loads. Engineers at the design stage are concerned by the maximal admissible aeroelastic deflection that guarantees structural integrity, i.e. avoid rupture of the structure, as well as proper aerodynamic performance. However, the stability of the system must also be of interest. Indeed, for a given wing design under given flow conditions, the magnitude of the destabilizing static aerodynamic loads can exceed that of



(a) Reconfiguration of aquatics plants subjected to a steady water flow. Experimental setup taken from Abdelrhman [9].



(b) Schematic illustration of lift-induced wing deflection on commercial airliner; on-ground configuration is in black, in-flight configuration is in red.

Figure 1.3: Example of static fluid-structure interaction problems.

the structural restoring forces. This can result in unbounded wing deflection and failure. This instability is called static divergence and has been encountered since the beginning of aviation by pilots and engineers [10].

Contrary to the static case, dynamic FSI features time-dependent flow and structural behavior. When the structural response is of vibrating nature, which is quite common, the interaction is referred to as Flow-Induced Vibrations (FIV). In this category of phenomena, the range of vibration mechanisms is wide. Kaneko *et al.* [11] proposed an interesting classification accounting for both single-phase and two-phase flows. In order to restrict the analysis to the scope of this thesis, Fig. 1.4 illustrates the classification of FIV only for single-phase flows. The distinction is made between steady and unsteady oncoming flows. The link between a vibratory structural response and unsteady flows is intuitive as we can easily understand that non-stationary flow phenomena, such as pressure or flow rate pulsations, will induce non-stationary structural deformations or displacements. Typical examples can be found in pipes conveying fluids where the flow rate is maintained by the action of compressors or pumps that may induce flow pulsations at a specific frequency. If the pulsating flow frequency becomes comparable to the piping system's natural frequency, significant vibrations may appear, hence leading to strong noise generation or even structural failure. Water hammer and valve vibrations are other examples of pulsation-induced structural responses of piping systems. Pulsating flows in conveying pipes have been extensively studied for more than 20 years and recent studies may be found in Gorman [12], Liu [13] and Zhou [14]. Arterial blood flows may also fall in the category of FSI systems with pulsating flows. For instance, the considerable flexibility of arterial walls coupled with hemodynamics and heart-induced pulsations is of critical matter for aneurysms. Turbulence is another source of excitation for flexible structures. Turbulence may be seen as chaotic flow fluctuations superposed to its mean component. Aircraft wings or civil engineering structures are endlessly subjected to atmospheric turbulence, resulting in buffeting that usually exhibits small amplitudes and a stochastic character depending on the turbulent intensity level of the wind itself.

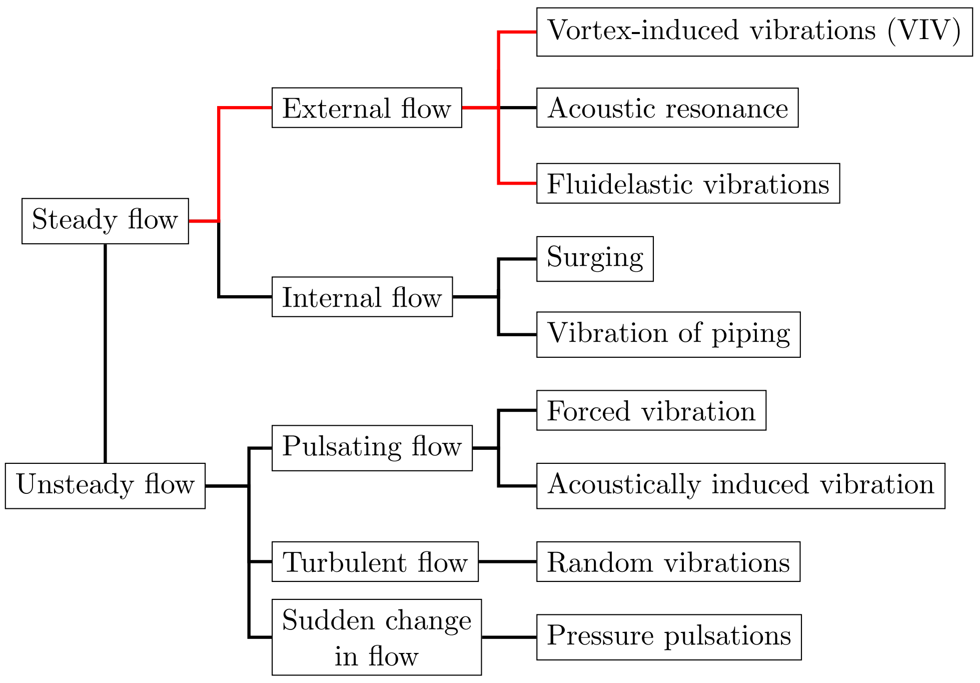


Figure 1.4: FIV classification for single-phase flows. The red path highlights the mechanisms that are treated in details in this thesis. Charts taken from [11].

Dynamic FSI involving oncoming steady flows is much less intuitive. The flow is now qualified as steady because it does not feature any unsteadiness such as those generated by external means (e.g. a pumping system) before interacting with the flexible structure. Conveying pipes may undergo vibrations when the flow rate is set beyond a critical value. Vibrations are sustained for constant values of the fluid velocity above the critical value and are not linked to any harmonic excitation such as pressure pulsations. Details about conveying pipe dynamics and instabilities can be found in Païdoussis [15]. One of the typical examples of pipe instability is the garden-hose instability. This mechanism was already studied by Bourriéres in 1939 and is still under investigation today [16]. This instability appears for flexible conveying pipes whose one extremity is free to move. When the fluid velocity reaches or is set beyond a critical value, the pipe experiences large and sustained vibratory motions. The garden-hose instability is schematically depicted in Fig. 1.5. Flexible structures submitted to an external steady flow may experience large vibrations and unstable modes as well. This particular type of FSI will be of major interest in this thesis, focusing on three distinct phenomena, namely Vortex-Induced Vibrations (VIV), galloping and flutter. These phenomena will be discussed in more detail in the next three sections.

In the case of conjugate heat transfer, the large difference in the time constant between the fluid and the solid does not usually permit rapid variations of the thermal field within the solid, even for unsteady flows, so that the thermal coupling is considered as steady or at least monotonically transient most of the time. Examples of CHT systems involving forced convection are engines or computer processor cooling. For instance, the Cessna 150 engine is cooled by the relative airflow generated by the aircraft's forward motion and the propeller flux, as illustrated in Fig. 1.6. Cooling fins are usually added on the surface of the cylinder blocks in order to increase the heat exchange efficiency by increasing the exchange surface area. In turbojet engines, the turbine blades are in contact with the hot gases exiting from the combustion chamber and thus

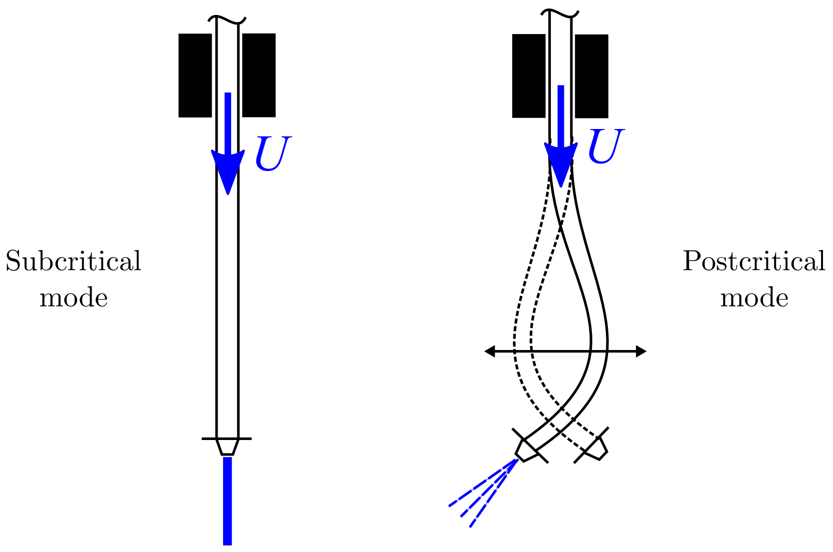


Figure 1.5: Illustration of the garden-hose instability. On the left, stable behavior when the fluid velocity U is under the critical value. On the right, unstable configuration when U is beyond the critical limit.

submitted to temperatures that can be higher than their melting point [17]. Cooling air from high- and low-pressure compressors is thus pumped inside internal blade channels and then blown through small holes drilled at the surface in contact with the hot gas flow. This creates a thin cooling film that controls the thermal transfer between the hot flow and the blade, as depicted in Fig. 1.7. Power plant heat exchangers and heat shields used for re-entry vehicles are other examples of applications for which CHT phenomena significantly impact the design process.

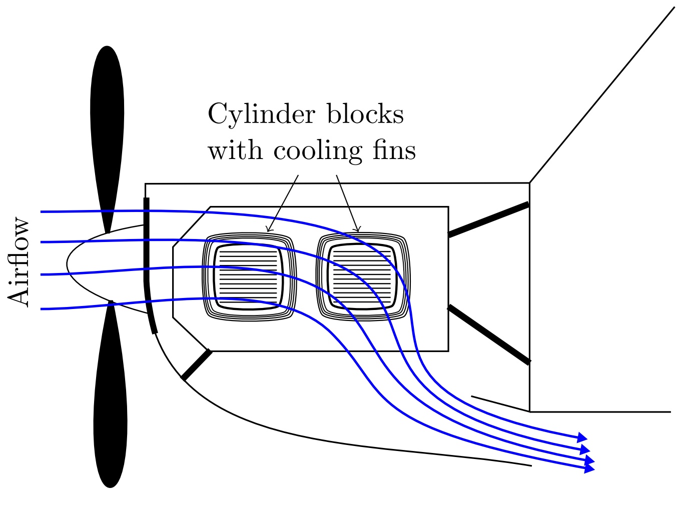


Figure 1.6: Cessna engine cooled by external airflow.

Mechanical and thermal fluid-structure interactions were introduced separately for clarity and because usually one of the two interactions is significantly stronger than the other, so that the latter can be neglected. However, there are also systems for which the two mechanisms have to be taken into account as they have mutual influence. For example in supersonic/hypersonic aircraft design, thermal effects induced by high-speed aerodynamic heating may have a significant

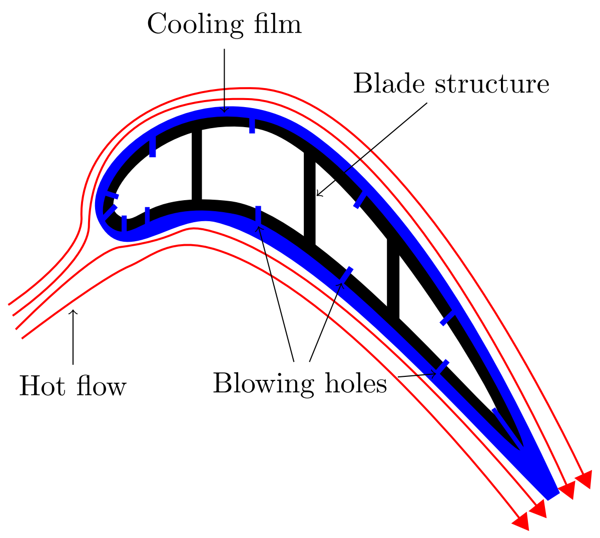


Figure 1.7: Turbine blade submitted to hot gas flow and protected by a fresh air cooling film.

influence on the aeroelastic response of flexible aircraft structures. This phenomenon is referred to as aero-thermoelasticity.

1.2 Vortex-induced vibrations

Vortex-induced vibrations can affect bluff flexible structures submitted to a steady cross-flow. This phenomenon has been extensively studied during the past few decades [18–20] and is still under investigation today [21–23] due to its significant impact on the design of many civil engineering structures: bridges, towers, towed cables, transmission lines, marine cables and offshore structures are typical examples of situations where VIV matters. VIV can be illustrated by the canonical case of a circular cylinder subjected to cross-flow. It is well-known that above a certain flow regime threshold, convective effects in the flow dominate viscous mechanisms and the vorticity produced at the solid boundary is not dissipated in the vicinity of the body. Two naturally unstable shear layers of opposite vorticity are generated and the wake behind the cylinder becomes unstable, turning into the so-called Von Kármán vortex street [20]. The resulting wake consists in a periodic vortex shedding that is characterized by a particular frequency over a large range of supercritical flow regimes. Fig. 1.8 illustrates the Von Kármán street in the wake of a circular cylinder in laminar flow.

The vortical pattern of the wake results in unsteady aerodynamic loads that may cause vibrations of flexible or flexibly mounted structures leading to stresses or fatigue damage [25]. The structural motion will, in turn, influence the vortex generation mechanism through a feedback process that may lead to unstable dynamics and large vibratory amplitudes when the shedding frequency of the vortices is close to the natural frequency of the structure. In practice, the flow regimes that correspond to this specific condition fall in the so-called lock-in region, in which the dynamics of the system is locked on a particular behavior involving large, but self-limited, motion. Structural instabilities or failure are the ultimate consequences of an uncontrolled VIV mechanism. Maritime structures such as drilling risers and pipelines are particularly concerned by VIV caused by the water current [25, 26]. Particular attention is also paid during the design process of slender structures submitted to wind such as bridge suspenders [27], towers and industrial chimneys [28]. When the design does not permit a satisfactory safety margin for the natural frequency and in order to prevent the full development of a Von Kármán street, VIV suppression devices such as helical strakes (Fig. 1.9 (e)) or foil-shaped fairings (Fig. 1.9 (h)) [29]

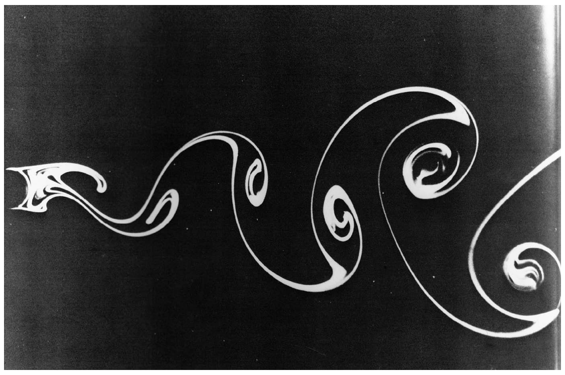


Figure 1.8: Laminar Von Kármán vortex street in the wake of a circular cylinder. Taken from [24].

can be used.

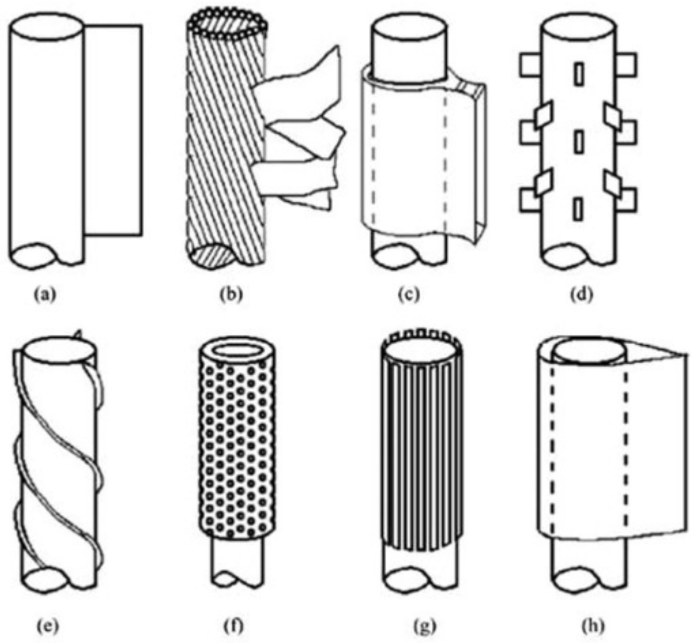
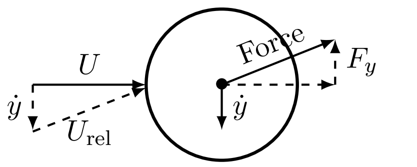


Figure 1.9: Illustration of typical VIV suppression devices. Taken from Holland *et al.* [29].

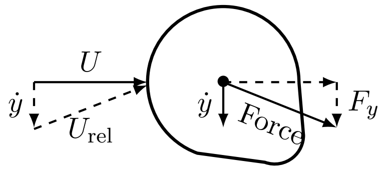
1.3 Galloping

At first sight galloping strongly resembles VIV because both phenomena result in high amplitude structural vibrations beyond a critical flow regime. However, and in contrast to VIV, galloping occurs in the absence of vortex formation and does not involve significant wake interaction. In contrast to VIV, which can be considered as a resonant phenomenon where the vortex shedding acts as an external excitation for the structure, galloping results from a physical instability of the coupled fluid-structure system. Considering a flexibly mounted one-degree-of-freedom bluff body, galloping should be seen as a negatively damped response of the system [30]. As for VIV, galloping is flow regime dependent, meaning that the instability can only develop above a certain threshold (e.g. above a critical flow velocity). A significant difference with VIV is that the range of flow regimes for which unstable dynamics occurs is unbounded above the critical conditions.

Although a perfectly circular cylinder is immune to galloping, comparing this example with a non-circular cylinder provides a good illustration of the phenomenon. Fig. 1.10 illustrates how the shape of a solid body could determine the occurrence of galloping. Fig. 1.10 (a) shows that for a perfectly circular body free to move vertically in a cross-flow of velocity U, the vertical component of the force caused by the relative motion $\dot{y}$ (for simplicity, the oscillatory lift component due to vortex shedding is purposely ignored in this case) will always oppose the displacement itself, thus preventing any possibility for the system to reach a sustained vibratory state. However, for a non-circular cylinder as shown in Fig. 1.10 (b) the force might be oriented in the same direction as the displacement, making an energy transfer from the flow to the structure possible, as discussed in Paidoussis *et al.* [30] and previously in Den Hartog [31]. This would result in an amplified motion of the structure over time. A very practical example of



(a) Perfect circular shape: the restoring force always opposes the structural motion.



(b) Arbitrary non circular shape: the force can be oriented in the same direction as the structural motion that can in turn be amplified.

Figure 1.10: Body shape dependence for galloping to occur.

the occurrence of galloping induced by a shape modification is the accretion of ice on electric transmission lines due to freezing meteorological conditions. When the wind is sufficiently strong, iced shape power lines can gallop and may cause conductor clashing, resulting in an interruption of service [32].

Another famous occurrence of galloping was that resulting in the spectacular oscillation and collapse of the Tacoma Narrows Bridge (Washington, USA) on November 7th 1940. The bridge experienced large torsional vibrations (twisting angle up to 35° !) at a low frequency of 0.2 Hz until it collapsed (Fig. 1.11).

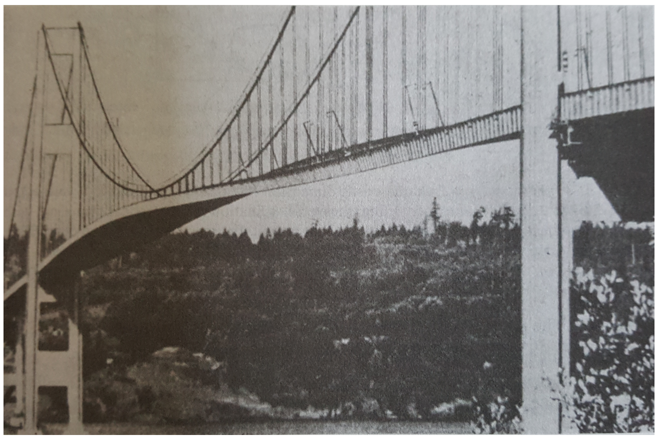


Figure 1.11: Instantaneous photograph of the torsional galloping of the Tacoma Narrows Bridge before its collapse. Taken from Païdoussis [30], originally taken from Scruton [33].

Depending on the circumstances, the dynamics of galloping can become quite complex. Parkinson [34] has already identified the two main classes of galloping which are the transverse galloping, as in the case of the iced power lines, and the torsional galloping of bridge decks. The complexity of the phenomenon can be further increased in case of interaction with the reattachment of the separated shear layer and/or vortex shedding. In particular, VIV and galloping can occur for the same structure at very different flow velocities or can overlap depending on the conditions. For the interested reader, many details can be found in Païdoussis *et al.* [30]

1.4 Flutter

Aeroelastic flutter is an instability similar to galloping in the sense that it occurs at flow regimes for which the total damping of the fluid-structure system becomes negative and where energy from the airstream is absorbed by the structure. The basic type of flutter is usually referred to as coupled-mode flutter, because it involves two structural modes interacting with each other due to the aerodynamics (only one mode is involved in galloping). The most relevant example of coupled-mode flutter occurrence is on aircraft wings and stabilizers which usually are the most flexible parts. Beyond a critical airspeed, vibratory mixed bending-torsion of the wing occurs in the form of self-feeding, potentially violent and destructive oscillations.

In order to schematically represent coupled-mode flutter, one can describe the resulting motion of the airfoil located at the wing tip. The motion, as depicted in Fig. 1.12, is the combination of a vertical translation (wing bending) and a rotation (wing twist) of the airfoil. For any airspeed beyond the critical threshold, any small perturbation such as turbulent buffeting, gust or control input, can cause the wing to enter flutter. The amplified vibration of the wing may lead to a drastic reduction of controls effectiveness (ailerons, elevator, rudder) or a catastrophic structural failure. It is important to note the significant contribution of the two components to

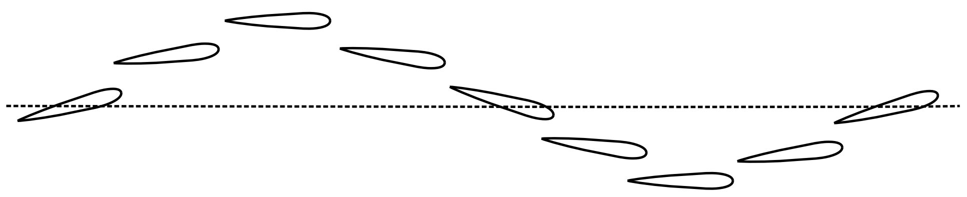


Figure 1.12: Example of typical motion of wing tip undergoing flutter.

the motion. Flutter occurs due to the dependence on the airspeed of the amplitude ratio and the phase shift between the two components, defining some conditions under which the structure can extract energy from the flow [35].

Aviation and aircraft designers have been concerned with flutter phenomena since the early ages of aviation [10]. Pilots and engineers noticed very early the self-excited behavior of flutter and the effect of wing rigidity, position of the flexural axis and mass distribution (specially for flight controls). In modern aircraft design, as the wing flexibility has dramatically increased with the introduction of composite materials and larger span, aeroelastic analysis is performed at the early stage of the design process. Aircraft certification also requires a flight flutter test, during which the boundary of the flight envelop is explored and confirmed to be flutter-free. Fig. 1.13 illustrates an occurrence of tail flutter during a flight flutter test performed by NASA on a Piper PA-30 Twin Comanche in 1966. Fig. 1.13 (a) and 1.13 (b) shows the significant deformation of the tail at two different times (separated by barely a few tenths of a second).

Similarly to galloping, flutter is a quite complex phenomenon that can involve a variety of mechanisms. Stall flutter refers to a particular aspect of flutter that falls under the study of nonlinear aeroelasticity. While classical flutter occurs in the absence of flow separation, such nonlinear effects dominate in the case of stall flutter [36]. This type of flutter may appear for rotorcraft or in rotating machineries where the blades operate at angles of attack that are close to the static stall limit. Shock dynamics and boundary-layer shock interaction in transonic

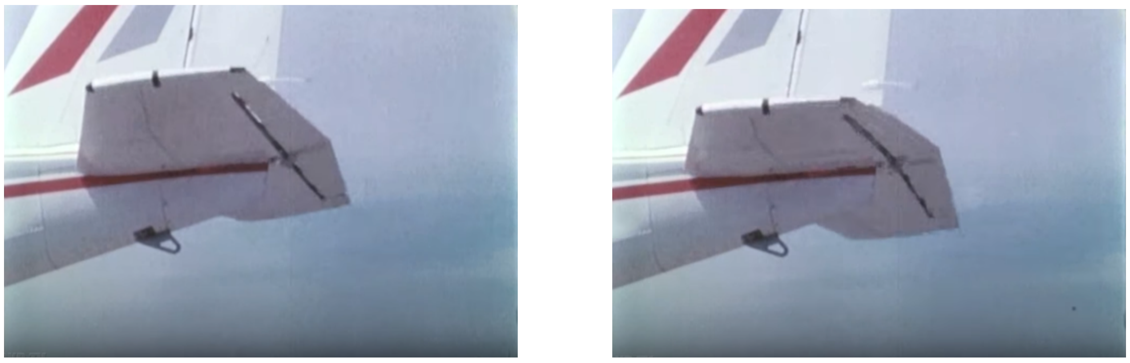

(a) Deformation of the tail at time t (b) Deformatin of the tail at time $t + \Delta t$

Figure 1.13: Tail flutter occurring during a flight test on a Piper PA-30 Twin Comanche (NASA, 1966).

flight or the presence of fuel tanks or missile for military aircraft are also examples of factors impacting the flutter characteristics. The flutter dynamics of a wing might also interact with the engine mount stiffness and with the combination of the gyroscopic torque of the engine and the propeller. This leads to a precession-type instability of the engine-propeller called whirl flutter.

1.5 The fluid-structure interaction problem in the engineering design process

Fluid-structure interaction plays a major role in many engineering applications and understanding the phenomena at play is key to elaborate safe designs. Ignoring the important role of FSI may lead to a drastic reduction in efficiency (e.g. bad aerodynamic performances or engine cooling), or may even cause structural failure (e.g. bridge collapse, aircraft crash or melting of materials). A simple solution to avoid FSI instabilities is to increase the structural weight. While this approach has been successfully applied in the past, it goes against the current trend towards lighter, larger and more flexible structures. Debrabandere [37] cited in his thesis the typical examples of the Millau viaduct in France, that has a length of 2460 m for a maximum height of 343 m, the Siemens 6 MW wind turbine with a rotor diameter of 154 meter or the Airbus A380 having a wingspan of 79.75 m. Such spectacular designs would not have been possible without efficient estimation of FSI dynamics. In research, scientists are also increasingly interested in FSI behaviour of micro air vehicles [38,39], bird or insect flapping flight [40], aeroelastic energy harvesting systems [41–44], or cardiovascular systems [45,46], to cite a few. It is therefore of major importance to develop engineering models that represent coupled fluid-solid systems with the best accuracy, efficiency and robustness while being sufficiently versatile to be applied to a wide range of physical configurations.

Experimental investigations, both in situ or in a wind tunnel, allow us to acquire accurate real-life data and measure phenomena that simplified mathematical models cannot represent. This makes experimental models truly necessary for the design or the certification process (e.g. aircraft flight flutter test). Experimental tests can also be used to validate numerical simulations. However, their use in industrial research and development tends to be reduced due to the significant cost associated with different aspects such as prototype manufacturing, experiment design and set up, or the need for destructive testing. Difficulties to match testing conditions with the design operating conditions may also impose the use of similarity laws based on non-dimensional quantities. These laws and the use of geometrically simplified scaled models could also introduce uncertainties in the results.

The work presented in this thesis falls into the category of numerical modeling of fluid structure interaction problems. Nowadays computational models and numerical simulations have become a tool of choice that is used on a daily basis to help design and predict system response. This increasing use of numerical simulations in research and industrial R&D has mostly been driven by the continuous increase in computer power over the past decades. Development of accurate numerical models with increasingly fewer physical simplifications improves the expected accuracy of the results and broadens the range of complex phenomena that can be represented. These models are also reproducible and less costly when compared to experimental models. This allows the industry to reduce its effort invested in experimental investigations or to postpone them to later in the design process, hence further reducing the development cost. Furthermore, the amount of numerical data available from simulations is much larger than the amount of measured data coming from experiments as the capabilities and the number of measuring devices are usually limited in the experimental set-up. However, numerical models are most of the time based on discretized physical equations that are solved by means of numerical algorithms. The discretization process and the implementation of the solvers are significant sources of uncertainties and errors in the results so that experimental validation is still required at some point in order to gain confidence in the ability of the models to generate reliable solutions. However, numerical models may require long computational times (from days to months) before providing exploitable results. Accurate simulations of real-life system configurations usually lead to large data sets to be computed after spatial dicretization. Modern computing architectures such as computational clusters allow us to distribute the data set over several computing units to parallelize in space the calculations and thus limit the computing time. Nevertheless, parallelization in time is not possible and, consequently, time-dependent problems featuring a large range of time scales still require long simulated time.

Summary of chapter 1

In this first chapter, an overview of the FSI problem has been given for both mechanical and thermal applications. The FSI problem has been defined as the mutual interaction between two distinct physics that are, on one hand, the fluid flow and on the other hand the deformation and heat conduction in the solid structure. FSI problems have been classified into several subcategories that mainly depend on the characteristics of the fluid flow (for instance pulsating flow vs turbulence buffeting vs steady flow, etc.). In this thesis the focus is on FSI phenomena involving steady external flows.

In that context, three mechanical FSI phenomena have been introduced. Vortex-Induced Vibrations (VIV) occur for a slender body when the shedding frequency of the wake is close to a natural frequency of that body. This results in a resonant, high-amplitude, response of the structure that can lead to fatigue or even failure. Bridges, towers and cables have been cited as typical examples of structures concerned by VIV. Galloping resembles VIV as it also involves large amplitude structural vibrations but does not result from resonant vortex shedding. Galloping is usually seen as a physical instability of the coupled system that is triggered beyond a specific flow regime. The Tacoma Narrows Bridge has been cited as the most famous example of torsional galloping that led to a catastrophic collapse. Flutter has been introduced as the typical FSI phenomenon that must be considered for aircraft design. Similarly to galloping, flutter is a physical instability of the coupled system that is triggered beyond a specific flow regime, but usually involves the interaction of several structural modes interacting with each other due to the aerodynamics. For that reason, it is also commonly referred to as coupled-mode flutter. In aircraft engineering, this phenomenon is accounted for from the early design phase and is a part of the certification process (flight flutter tests).

Finally, the importance of FSI in the engineering design process has been discussed. As FSI can be observed in a wide range of applications and can cause failure, its understanding is key to the development of safe and efficient designs. Two main approaches

have been reviewed for the study of FSI problems: the experimental approach and the computational approach. The former gives valuable real-life data and remains necessary at some stages of the design process. However, the use of experimental tests tends to be minimized due to their high operational cost, and because they can be partly replaced by computational models. Nowadays, efforts are invested by researchers and scientists in developing numerical models with always higher accuracy, following the increase of the general computing power. The work presented in this thesis falls into the category of numerical and computational modelling of FSI problems.

Chapter 2

Mathematical model of FSI

Despite the remarkable diversity of fluid-structure systems (many fields of applications and many physical phenomena involved), most of them can be represented in a very general way by the same set of basic equations. Mirroring the fundamental nature of FSI, which couples two distinct physics, the mathematical models will be described by first focusing on the fluid and solid dynamics separately, and then by introducing the coupling conditions.

2.1 The monolithic and the partitioned approaches

The mathematical formulation of FSI problems is usually based on one of two possible strategies: the monolithic or the partitioned approach [47–49]. In the monolithic approach, the structural and fluid problems are represented within the same mathematical framework where the sub-domains are treated simultaneously by a single set of governing equations. A unified algorithm is used to solve the entire system where interfacial conditions are implicit to the procedure. This can lead to improved stability, convergence and accuracy of the solution. However, the disadvantage of monolithic methods is that they lead to a large system of equations that is poorly conditioned due to different orders of magnitude of the physical variables between each sub-domain. Conversely, the partitioned approach describes the fluid-structure interaction problem as two distinct mathematical problems that retain separate domains for the fluid and solid. Distinct sets of equations are developed for each physics, leading to a better-conditioned system to be solved. Nevertheless, the coupling conditions at the interface between the domains need to be treated explicitly and are thus a source of error and reduced robustness. The major advantage of the partitioned approach is that it permits the use of specific algorithms that are tailored to each sub-system in order to guarantee the accuracy and robustness of each solution procedure separately. In this thesis, the partitioned approach is chosen as the coupling strategy for its modularity in the description of each physics.

The physical quantities used to describe each physics are defined on continuous fluid and solid domains, $\Omega_\mathrm{f} \subset \mathbb{R}^{n_d}$ and $\Omega_\mathrm{s} \subset \mathbb{R}^{n_d}$, respectively, where n_d is the number of physical dimensions (usually two or three). The subscripts f and s are used when a clear distinction between fluid and solid quantities is required. As illustrated in Fig. 2.1 for the generic case of a clamped solid beam immersed in a fluid, it is assumed that the two domains do not overlap but share a common boundary $\Gamma = \bigcup_i \Gamma_i$ which is defined as the fluid-structure interface. The remaining boundaries of the fluid and solid domains are denoted as $\partial\Omega_\mathrm{f} = \bigcup_i \partial\Omega_\mathrm{f}^i$ and $\partial\Omega_\mathrm{s} = \bigcup_i \partial\Omega_\mathrm{s}^i$, respectively. In those expressions, Γ_i and $\partial\Omega^i$ are piecewise decomposition of the fluid-structure interface subsets and domain boundaries, respectively.

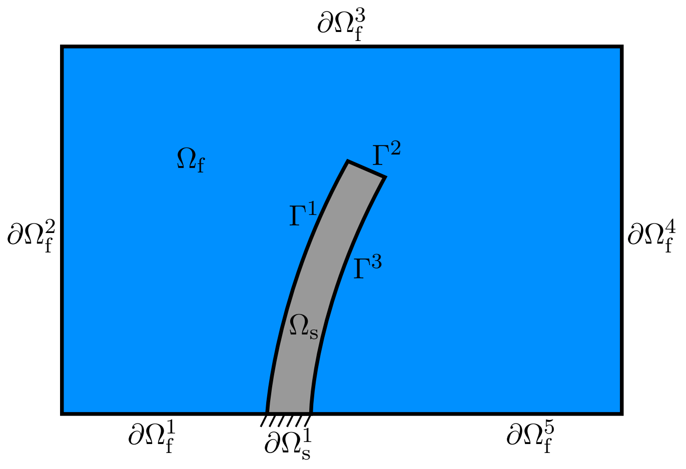


Figure 2.1: Domain separation for a generic fluid-solid system with interface and intrinsic boundaries.

2.2 General description of fluid and solid motion

The study of continuum mechanics is usually based on two main approaches for the description of motion: the Lagrangian and Eulerian approaches. In the Lagrangian formalism, the spatial points follow the motion of the material particles. A Lagrangian description is usually chosen for the motion of solids for which two neighbor material points remain neighbor over time. Engineers are also mostly interested in the loads resulting from the motion. In the solid, the loads depend on the displacements. That makes the Lagrangian description, in which the relative position of points matters, a natural choice. On the contrary, this approach is usually inappropriate for the description of fluid motion for which the relative position of material points is not relevant due to their significant relative displacements (fluids do no conserve their shape). The Eulerian approach is then used to describe fluids motion. In this case, the velocity of material particles is tracked at specific locations of the domain and the loads depend on this velocity. However, and for some cases such as flows in a closed domain (i.e. without inflow and outflow) or free-surface flows, the position of particles can be meaningful and the Lagrangian approach might be preferred.

The coupling of two distinct fluid and solid domains through a common boundary raises the problem of interface tracking. For the solid part, the displacement of the interface is directly obtained from the Lagrangian description. This is not the case for the fluid part in which the standard Eulerian approach does not provide an explicit description of the boundary motion. Consequently, a representation that combines the advantages of each description is developed in the fluid domain. Such a representation is known as the arbitrary Lagrangian-Eulerian (ALE) method [50, 51]. It is a generalization of the Lagrangian (reference moving with the material point) and Eulerian (reference fixed) descriptions which allows the reference domain to move in some arbitrarily specified way and independently of the motion of the material particles. The description of boundary motion in the fluid domain Ω_f is thus enabled by the ALE description. However, the ALE also introduces additional complexity. First, the arbitrary motion of the fluid spatial domain has to be specified or inferred as part of the solution procedure. Moreover, the governing equations describing the fluid dynamics have to be adapted to take into account the effect of the domain motion (see next Section).

2.3 Fluid dynamics

The basic equations governing the motion of a compressible fluid invoke the three fundamental physical principles, namely the conservation of mass, momentum and energy. These principles are expressed in the form of a system of partial differential equations (PDE) on Ω_f, following an ALE description of the domain in Cartesian coordinates:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho(\boldsymbol{v} - \boldsymbol{v}_\Omega)) &= 0, \\
\frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}(\boldsymbol{v} - \boldsymbol{v}_\Omega)) &= \rho \boldsymbol{f} - \nabla p + \nabla \cdot \boldsymbol{\tau}, \\
\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E(\boldsymbol{v} - \boldsymbol{v}_\Omega)) &= \nabla \cdot (\lambda \nabla T) - \nabla \cdot (p \boldsymbol{v}) + \nabla \cdot (\boldsymbol{\tau} \boldsymbol{v}) + \rho \boldsymbol{f} \cdot \boldsymbol{v} + q.
\end{aligned} \tag{2.1}$$

In these equations ρ is the fluid density, $\boldsymbol{v}$ the velocity, $\boldsymbol{f}$ some volume force per unit mass (e.g. gravity), $\boldsymbol{\tau}$ the shear stress tensor, E the total energy per unit mass, λ the thermal conductivity, T the temperature, q some volume heat source and $\boldsymbol{v}_\Omega$ the velocity field of the ALE fluid domain. Note that for clarity the subscript "f" for all fluid quantities has been discarded. The arbitrary motion of the fluid domain, with respect to the distinct motion of the fluid particles, will be generally expressed by the fact that $\boldsymbol{v}_\Omega \neq \boldsymbol{v} \neq \boldsymbol{0}$. From the general ALE description, a purely Lagrangian description is recovered when $\boldsymbol{v}_\Omega = \boldsymbol{v}$, which is the case at the fluid-structure interface in presence of a viscous flow and where the velocity is imposed by the solid motion. A purely Eulerian description is recovered when $\boldsymbol{v}_\Omega = \boldsymbol{0}$, i.e. at locations where the fluid domain is kept fixed.

Equations (2.1) define the dynamics of a compressible fluid but additional equations are required in order to close the system. A constitutive equation is used to express the shear stress tensor in terms of the velocity field. For a Newtonian fluid, the viscous stress is given by

$$\boldsymbol{\tau} = \mu \left(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^\mathrm{T} - \frac{2}{3} \mathbf{I} \left(\nabla \cdot \boldsymbol{v} \right) \right) , \tag{2.2}$$

where μ is the dynamic viscosity of the fluid. Combining this expression with the above conservation laws leads to the compressible Navier-Stokes equations. Only Newtonian fluids will be considered in this thesis. The incompressible form of the Navier-Stokes equations is obtained by treating the density as a constant in space and time.

On the other hand, if the fluid is considered inviscid, i.e., if molecular processes such as viscous and thermal diffusion can be neglected ($\mu = \lambda = 0$), the compressible Navier-Stokes equations reduce to the Euler equations.

Liquids and low Mach gas flows are usually treated as incompressible. For compressible gas flows, additional state equations are used to further close the system of conservation equations. In the following only a calorically perfect gas assumption will be considered, but more complex fluid models could be considered. More specifically, the equation-of-state for a calorically perfect gas reads

$$p = \rho R T, \tag{2.3}$$

where R is the specific gas constant. Additionally, the specific internal energy $e = E - ||\boldsymbol{v}||^2/2$ is expressed as

$$e = c_v T \, , \tag{2.4}$$

where the specific heat at constant volume c_v is considered constant. Similarly, the specific enthalpy $h = e + RT$ is given by

$$h = c_p T \, , \tag{2.5}$$

where the specific heat at constant pressure c_p is also constant. The two specific heats are related by

$$c_p - c_v = R \, , \tag{2.6}$$

and their ratio is defined as

$$\gamma = \frac{c_p}{c_v} \, . \tag{2.7}$$

The temperature dependence of the viscosity can be modeled through Sutherland's law

$$\mu = \mu_0 \left(\frac{T}{T_0} \right)^{3/2} \frac{T_0 + S}{T + S} \, , \tag{2.8}$$

where μ_0 is the viscosity at temperature T_0 and S is a constant. Additionally, the elementary kinetic theory of perfect gases [52] shows that $\lambda \propto \mu c_p$, so that the Prandtl number

$$\mathrm{Pr} = \frac{\mu c_p}{\lambda} \tag{2.9}$$

can be considered constant. Tab. 2.1 summarizes key material properties for dry air (at standard conditions), that is most of the time well approximated by the calorically perfect gas assumption.

R [J kg^{-1} K^{-1}]	287.058
μ_0 [Pa s]	$1.72 \cdot 10^{-5}$
T_0 [K]	273.15
Pr [-]	0.7
c_v [J kg^{-1} K^{-1}]	717.6
c_p [J kg^{-1} K^{-1}]	1004.7
γ [-]	1.4

Table 2.1: Typical material properties of dry air at standard conditions under the calorically perfect gas assumption.

Finally, note that the ALE form of the conservative equations requires an expression for the domain velocity $\boldsymbol{v}_\Omega$ in order to obtain a fully closed system. The arbitrary motion of the fluid domain is usually computed by independent additional equations, as discussed further below.

2.3.1 Turbulence modeling

Turbulence is a consequence of the nonlinear character of the advection term in the Navier-Stokes equations (Eqs. (2.1)). A complete representation of turbulent flows based on a direct solution of those equations would require a very fine discretization of the fluid domain in order to capture the smallest flow scales. From a computational point of view, such an approach called Direct Numerical Simulation (DNS) is most of the time unfeasible because of its excessive computational cost (memory and CPU time requirements). DNS are thus usually limited to simple geometries such as channel flows for instance and low to moderate Reynolds number. Consequently, turbulence models have been developed in order to obtain an affordable computational representation of a mean or filtered component of the flow by modeling a part of the turbulent spectrum. The Large Eddy Simulation (LES) method decomposes the flow field into large and small turbulent scales by applying a spatial filter [53, 54]. In LES, the largest flow scales are directly resolved while the smallest are taken into account through the use of turbulence, or subgrid-scale, models. Even if a LES is computationally less expensive than DNS, the resolution of the dynamically relevant scales is usually still very demanding, especially in the near-wall region.

The alternative approach used in this work, which is more affordable, is based on the Reynolds-Averaged Navier-Stokes (RANS) equations [55]. In this approach, the flow variables are decomposed into a mean component and turbulent fluctuations, such that

$$\boldsymbol{v} = \bar{\boldsymbol{v}} + \boldsymbol{v}' \tag{2.10}$$

where $\bar{\boldsymbol{v}}$ is the mean component and $\boldsymbol{v}'$ is the turbulent component. The mean flow quantities can be obtained by solving the RANS equations. In this approach, the entire turbulence spectrum is modeled. Hence a statistical representation of the averaged flow is obtained at reduced computational cost. For statistically unsteady flows, unsteady-RANS (URANS) equations can be written to account for time dependence that are characterized by the separation of time scales between the unsteadiness of the mean flow and the unsteadiness of turbulent fluctuations. Introducing the decomposition 2.10 into the Navier-Stokes equations and averaging these equations[1] leads to the same set of equations for the mean flow up to an additional unclosed stress-like term referred to as the Reynolds stress $R_{ij} = \rho\overline{\boldsymbol{v}' \otimes \boldsymbol{v}'}$. Following the Boussinesq approximation [55], the Reynolds stress is very often modeled in a similar fashion to the viscous stress by introducing an eddy, or turbulent, viscosity μ_t. In this case, the RANS equations are also given by Eqs. (2.1) but with the flow variables representing averaged quantities. Furthermore, Eq. (2.2) is replaced by

$$\boldsymbol{\tau} = \mu^* \left(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^\mathrm{T} - \frac{2}{3} \boldsymbol{I} \left(\nabla \cdot \boldsymbol{v} \right) \right) , \tag{2.11}$$

where the viscosity is the sum of the dynamic molecular viscosity μ and the eddy (turbulent) viscosity μ_t:

$$\mu^* = \mu + \mu_\mathrm{t} \, . \tag{2.12}$$

The thermal conductivity in the energy transport equation is also decomposed into an intrinsic molecular part λ and a turbulent contribution λ_t such that the resulting conductivity is given by

$$\lambda^* = \lambda + \lambda_\mathrm{t} \, . \tag{2.13}$$

The eddy conductivity is generally expressed as a function of the eddy viscosity μ_t by means of a turbulent Prandtl number Pr_t whose value is usually imposed as an external parameter:

$$\lambda_\mathrm{t} = \frac{\mu_\mathrm{t} c_p}{\mathrm{Pr}_\mathrm{t}} \, . \tag{2.14}$$

The eddy viscosity itself is calculated by suitable turbulence models. A large range of turbulence models have been developed, each having slightly different properties, advantages and shortcomings and targeted applications. A complete description of each turbulence model is beyond the scope of this thesis but the Spalart-Allmaras [56] and the Shear Stress Transport $k - \omega$ [57] models can be cited as two of the most common and widely used models for engineering fluid (and by extension FSI) applications. The descriptions of these two turbulence models can be found in Appendix A. The models compute the eddy viscosity by means of additional transport equations for turbulent-related variables such as the turbulent kinetic energy and the specific turbulent dissipation rate.

2.4 Solid dynamics

The dynamic behavior of a deformable solid results from Newton's second law of motion and the balance between inertial, internal and external forces. This equilibrium is usually developed on a Lagrangian deformable solid domain. Under this assumption, it is common to describe the motion of the solid in time by the mapping between a reference configuration, where the positions of the material particles are denoted by $\boldsymbol{X}$, and the current configuration where the positions of the same material particles are denoted by $\boldsymbol{x}$. Note that the reference configuration is not necessarily the initial configuration of the solid. At any time, the mapping between the reference and the current configuration can be defined as

$$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{X}, t) \, . \tag{2.15}$$

[1]Note that for the compressible equations, Favre averaging is usually used.

The displacement of the material points from the reference to the current configuration is given by

$$\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{X}\,, \tag{2.16}$$

and the deformation gradient is defined by the tensor

$$\mathbf{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}\,. \tag{2.17}$$

The equilibrium equation within a solid domain Ω_s considered in its current configuration reads

$$\rho \frac{\partial^2 \boldsymbol{d}}{\partial t^2} - \nabla \cdot \boldsymbol{\sigma} = \boldsymbol{b}\,, \tag{2.18}$$

where ρ, $\boldsymbol{\sigma}$ and $\boldsymbol{b}$ are the solid density, the Cauchy stress tensor and the external body forces, respectively. Kinematic and constitutive laws are used to close the system. A kinematic equation is used to define the strain tensor $\mathbf{E}$ as a function of the displacement field. In a general representation of solid dynamics, the kinematic relation is a nonlinear equation for representing finite strains and potentially large displacements. The usual Green strain tensor is defined by the relation

$$\mathbf{E}^G = \frac{1}{2}\left(\mathbf{F}^\mathrm{T}\mathbf{F} - \mathbf{I}\right) = \frac{1}{2}\left(\nabla\boldsymbol{d} + \nabla\boldsymbol{d}^\mathrm{T} + \nabla\boldsymbol{d}^\mathrm{T}\nabla\boldsymbol{d}\right)\,, \tag{2.19}$$

which is invariant under rigid-body motion. Another common choice is to measure deformation with the natural strain tensor [58]:

$$\mathbf{E}^N = \frac{1}{2}\ln\left(\mathbf{F}^\mathrm{T}\mathbf{F}\right)\,, \tag{2.20}$$

which is also invariant under rigid-body motion. In the case of the linear theory for infinitesimal strain, the Cauchy strain tensor

$$\mathbf{E}^C = \frac{1}{2}\left(\nabla\boldsymbol{d} + \nabla\boldsymbol{d}^\mathrm{T}\right)\,, \tag{2.21}$$

is usually used as a linearization of the Green tensor, but its linear formulation breaks the invariance of the strain with respect to rigid-body motion and is thus only valid under the assumption of small displacements.

The constitutive equation expresses the stress tensor as a function of the strain tensor. For an elastic material, the current stress state depends only on the current strain:

$$\boldsymbol{\sigma} = \mathbb{C}\mathbf{E}\,, \tag{2.22}$$

where $\mathbb{C}$ is the fourth-order stiffness tensor, generally nonlinear. For an isotropic linear elastic material, this tensor becomes the well-known Hooke's tensor, written in indicial notation as

$$\mathbb{C}_{ijkl} = \mathbb{H}_{ijkl} = \frac{E}{1+\nu}\left[\frac{1}{2}\left(\delta_{ik}\delta_{jl} + \delta_{jk}\delta_{il}\right) + \frac{\nu}{1-2\nu}\delta_{ij}\delta_{kl}\right]\,, \tag{2.23}$$

where E and ν are the Young's modulus and the poisson coefficient of the material, respectively. For a more general representation of the material behavior, for instance involving plasticity, the constitutive law may be written in a differential form (time derivative) where the current stress state also depends on the history of the deformation:

$$\dot{\boldsymbol{\sigma}} = \mathbb{M}\dot{\mathbf{E}}\,.$$

In this expression $\mathbb{M}$ is a general nonlinear fourth-order elasto-plastic tensor. A detailed description of elasto-plasticity models is beyond the scope of this thesis that only deals with purely elastic materials, but interested readers may find additional background in Bertram and Glüge [59], Ibrahimbegovic [60] or Ponthot [58].

The thermo-mechanical behavior of a deformable solid is obtained when the temperature field is considered in addition to the deformation. The temperature field obeys the heat equation in the current configuration of Ω_s,

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) = q \,, \tag{2.24}$$

where c is the specific heat capacity and q is some possible volume heat source, that may come from external and independent thermal loads or may be coupled to the solid motion for which irreversible (plastic) deformations or friction forces due to contacts will produce additional heat sources. Thermal effects on the dynamics of the solid can be expressed by the multiplicative decomposition of the thermomechanics deformation gradient $\tilde{\mathbf{F}}$ into its mechanical part $\mathbf{F}$ and a thermal part $\mathbf{F}_T$ related to the thermal expansion as

$$\tilde{\mathbf{F}} = \mathbf{F}_T \mathbf{F} \,, \tag{2.25}$$

with

$$\mathbf{F}_T = (1 + \beta(T - T_0))\, \mathbf{I} \,, \tag{2.26}$$

where β is the thermal expansion coefficient of the material, and T and T_0 the current and reference temperatures, respectively.

2.5 Boundary and coupling conditions

The set of partial differential equations representing the dynamics of both fluid and solid can be solved as a well-posed problem only if they are complemented by a set of boundary conditions that must be satisfied at any time. Boundary conditions are expressed as constraints imposed on the physical variables at the boundary of the finite domains $\partial\Omega_\mathrm{f}$ and $\partial\Omega_\mathrm{s}$. Dirichlet type boundary conditions explicitly impose the value of a physical quantity. Typical examples of Dirichlet boundary conditions are freestream flow conditions, clamped structural surfaces or isothermal walls. On the other hand, Neumann type boundary conditions impose the value of the spatial derivative of a physical quantity. Typical examples of Neumann boundary conditions are boundary layer pressure gradients, structural external loads or heat fluxes at heat exchange boundaries. For a given problem, Dirichlet and Neumann boundary conditions can be combined on non-intersecting subsets of $\partial\Omega_\mathrm{f}$ and $\partial\Omega_\mathrm{s}$. A special weighted combination of Neumann and Dirichlet boundary conditions on the same subset of $\partial\Omega$, called Robin type boundary condition, can also be used under some circumstances. For time dependent problems, the governing equations can be integrated in time provided that initial conditions, describing the initial state of the problem, are also specified.

Boundary and initial conditions play a major role in the definition of the physical problems and are directly related to the final solution. In the case of fluid-structure interaction problems, the distinct fluid and solid domains are connected through a common subset, Γ, of $\partial\Omega_\mathrm{f}$ and $\partial\Omega_\mathrm{s}$, in which fluid and solid boundary conditions will depend on each other. The resulting coupling conditions express the continuity of certain physical quantities through Γ. In particular, the mechanical coupling condition expresses the continuity of the displacement $\boldsymbol{d}^\Gamma$ and load $\boldsymbol{t}^\Gamma$ on Γ,

$$\begin{aligned} \boldsymbol{d}_\mathrm{f}^\Gamma &= \boldsymbol{d}_\mathrm{s}^\Gamma \,, \\ \boldsymbol{t}_\mathrm{f}^\Gamma &= \boldsymbol{t}_\mathrm{s}^\Gamma \,, \end{aligned} \tag{2.27}$$

where the load on the fluid side is given by $\boldsymbol{t}_\mathrm{f} = -p\boldsymbol{n} + \boldsymbol{\tau}\boldsymbol{n}$ and the load on the solid side by $\boldsymbol{t}_\mathrm{s} = \boldsymbol{\sigma}\boldsymbol{n}$, $\boldsymbol{n}$ being the unit normal to the interface pointing outwards of the solid domain. The continuity of the velocity can be naturally derived by taking the first derivative of the continuity for the displacement,

$$\boldsymbol{v}_\mathrm{f}^\Gamma = \dot{\boldsymbol{d}}_\mathrm{s}^\Gamma \,, \tag{2.28}$$

that is a natural expression of the no-slip condition for viscous flows. In the case of inviscid flows based on Euler equations, only the normal component of the velocity with respect to the interface is continuous:

$$(\boldsymbol{v} \cdot \boldsymbol{n})_\mathrm{f}^\Gamma = \left(\dot{\boldsymbol{d}} \cdot \boldsymbol{n}\right)_\mathrm{s}^\Gamma \,, \tag{2.29}$$

and expresses the impermeability condition. On the other hand, the continuity of the temperature and heat flux is expressed by the thermal coupling condition as

$$\begin{aligned} T_\mathrm{f}^\Gamma &= T_\mathrm{s}^\Gamma \,, \\ (\lambda \nabla T \cdot \boldsymbol{n})_\mathrm{f}^\Gamma &= (\lambda \nabla T \cdot \boldsymbol{n})_\mathrm{s}^\Gamma \,. \end{aligned} \tag{2.30}$$

The coupling conditions presented above are crucial in the definition of the coupled fluid-solid problem because they mathematically represent the mutual influence that each physics imposes on the other through constraints on their respective boundary conditions. Hence, the solution within one domain is directly dependent on the solution within the other domain.

2.6 Non-dimensional parameters

Non-dimensional parameters are defined by the combination of different physical quantities. They are widely used to quantify the relative importance of the underlying mechanisms involved in the physical system. This section briefly introduces the relevant non-dimensional numbers related to fluid-structure interaction problems that are frequently used throughout this work.

Mach number The Mach number M is defined as the ratio between a reference fluid flow velocity U and the speed of sound a:

$$\mathrm{M} = \frac{U}{a} \,. \tag{2.31}$$

The speed of sound is defined as the speed of a sound wave as it propagates, by successive molecular collisions, through a fluid. The formal definition of the speed of sound is

$$a = \left.\frac{\partial p}{\partial \rho}\right|_s \,, \tag{2.32}$$

where s is the entropy. Under the calorically perfect gas assumption, the speed of sound reduces to

$$a = \sqrt{\gamma R T} \,, \tag{2.33}$$

a function of temperature only. The Mach number expresses the importance of compressibility effects, such as the occurrence of shock waves, in a fluid flow and separates the flow regimes between subsonic ($\mathrm{M} < 1$) and supersonic ($\mathrm{M} > 1$). For gas flows at low Mach number, conventionally $\mathrm{M} < 0.3$, density variations can be neglected and the flow can be considered as incompressible.

Reynolds number The Reynolds number Re expresses the ratio between inertial (advective) and viscous effects in a fluid flow:

$$\mathrm{Re} = \frac{UL}{\nu} \,, \tag{2.34}$$

where L is a characteristic length and $\nu = \mu/\rho$ the kinematic fluid viscosity. The Reynolds number is used to characterize the turbulence level, from smooth laminar flows (low Re) to highly turbulent flows (large Re), and the size of the viscous boundary layer.

Strouhal number The Strouhal number Str is a normalization of the dominant frequency that may appear for unsteady flows:

$$\mathrm{Str} = \frac{fL}{U}\,, \tag{2.35}$$

where f is the main flow frequency. For instance, in the case of bluff body aerodynamics, experimental measurements have shown that the Strouhal number based on the vortex shedding frequency can be expressed as a function of the Reynolds number.

Prandtl number The Prandtl number Pr expresses the ratio between the momentum diffusivity and thermal diffusivity of a fluid. It was already defined in Eq. (2.9) but can also be expressed as

$$\mathrm{Pr} = \frac{\nu}{\alpha}\,, \tag{2.36}$$

where $\alpha = \lambda/\rho c_p$ is the thermal diffusivity. Small values of the Prandtl number ($\mathrm{Pr} << 1$) indicate that thermal diffusivity dominates, or that the heat conduction is relatively more important compared to advection. The thickness of the thermal boundary layer is thus larger than the thickness of the momentum boundary layer. Conversely, large values of the Prandtl number ($\mathrm{Pr} >> 1$) indicate the dominance of momentum diffusivity and advective heat transfer.

Biot number The Biot number represents a comparative measure between conduction within a solid and convection at its surface:

$$\mathrm{Bi} = \frac{hL}{\lambda}\,, \tag{2.37}$$

where h is a convective heat transfer coefficient depending on the flow conditions. Low values of the Biot number ($\mathrm{Bi} << 1$) indicate rapid adaptation of the temperature field inside the solid against external thermal constraints.

Mass number The mass number Ma simply expresses the relative importance of the ratio between the fluid and solid densities:

$$\mathrm{Ma} = \frac{\rho_\mathrm{s}}{\rho_\mathrm{f}}\,. \tag{2.38}$$

This number is usually of great importance in fluid-structure interaction problems, especially because it is related to the added-mass effect. Added-mass can be seen as the inertia added to the structure as a consequence of the acceleration given to a certain volume of surrounding fluid during the motion. The fluid forces resulting from the motion of the solid are associated with an inertia force, where the coefficient of inertia is called the added mass. Significant added-mass effects are known to appear when $\mathrm{Ma} \leq 1$, and are usually a cause of numerical instability in computational models.

Reduced velocity The reduced velocity U_r is the ratio between the reference flow velocity and a characteristic propagation speed of elastic waves through the solid, c:

$$U_\mathrm{r} = \frac{U}{c}\,, \tag{2.39}$$

where $c = \sqrt{E_\mathrm{s}/\rho_\mathrm{s}}$ depends on the Young modulus and the solid density.

Cauchy number The Cauchy number Cy expresses the ratio between the flow dynamic pressure and the elasticity of the solid:

$$\mathrm{Cy} = \frac{\rho_\mathrm{f} U^2}{E_\mathrm{s}}\,. \tag{2.40}$$

The Cauchy number usually measures the amount of elastic deformation of a solid submitted to a fluid flow. For high values of Cy, large structural deformations are usually expected.

Summary of chapter 2

The mathematical modelling of FSI problems has been introduced in this chapter. The two common strategies for the mathematical formulation have been reviewed. The monolithic approach embeds the coupled system into the same algorithmic framework which brings accuracy, stability and implicit treatment of the interface conditions. However, it leads to a large, complex and poorly scaled system of equations. Conversely, the partitioned approach, which is followed in this work, considers the coupled problem as an assembly of sub-modules (one per coupled physics). Dedicated and optimized algorithms can thus be used in each domain, but the interface conditions have to be treated explicitly.

The general formalism for the description of fluid and solid motion has been reviewed. Usually the Lagrangian formalism is used for solids while the Eulerian formalism is most of the time better suited for fluids. However, the problem of interface tracking raised by the coupling of a fluid and solid domain cannot be naturally tackled by using a pure Eulerian approach within the fluid domain. Consequently an arbitrary Lagrangian-Eulerian (ALE) formalism is usually developed. By combining advantages from both formalisms, the ALE approach allows an Eulerian description of the fluid motion within an arbitrary moving fluid domain. This adds complexity to the model, as the motion of the fluid domain has to be described and the fluid governing equations have to be extended.

The governing equations for the fluid and solid dynamics have been presented. The ALE form of the compressible Navier-Stokes equations are considered in the fluid domain (accounting for both mechanical and thermal aspects). They reduce to the Euler equations in the inviscid limit. The state of the fluid is supposed to follow the perfect gas law. Turbulence is treated by introducing Reynolds decomposition of the fluid variables and averaging the transport equations. This leads to the (unsteady) Reynolds-Averaged form of the governing equations that are complemented with a turbulence model such as the one-equation Spalart-Allmaras or the two-equation SST $k - \omega$. In the solid domain, the mechanical part is governed by the classical equilibrium equation in a Lagragian formalism. A nonlinear description of the deformations is considered to account for large displacements, but only pure elastic materials are considered. The solid heat equation has been introduced separately to cover the thermo-mechanical aspects.

In addition to the intrinsic boundary conditions for each physics, the interface or coupling conditions have been introduced. These conditions simply express the continuity of the displacements (or velocity), loads, temperature and heat fluxes across the fluid-solid interface. They are the central point of the coupling process as they explicitly state that the solution within one domain is directly dependent on the solution within the other domain.

Finally, the most common non-dimensional parameters governing the coupled physics have been recalled. In a coupled FSI problem, the mass ratio, Ma, and the Biot number, Bi, are the most important ones as they combine physical quantities from both physics.

Chapter 3

Numerical model of FSI

In the previous chapter, the mathematical formulation of FSI systems was introduced. Unfortunately, no analytical solution is available for such a complex model. Hence, computational models are required to numerically solve the coupled problem. This chapter deals with such numerical models.

The problem is thus discretized in space and integrated in time. The fluid and solid domains are discretized into computational meshes whose nodes contain the variable solutions of the physics. The resulting numerical problem can be treated and solved on a computing unit with the help of specialized solvers. This chapter covers the major aspects of the numerical treatment of FSI problems. In Section 3.1, general aspects of the numerical methods are introduced. Specifically, a comparison between monolithic and partitioned approaches is carried out. The latter, in which each domain is solved by dedicated solvers, is then developed in more detail. Sections 3.2 and 3.3 introduce the mechanical and thermal coupling algorithms, respectively, that are designed to ensure continuity of the boundary conditions at the interface. This procedure requires information transfer at the interface, but meshes might not feature matching discretization and interpolation is consequently required. This is addressed in Section 3.4. Each domain is discretized independently and, as the domains deform, mesh deformation is also required, particularly for the fluid part. Mesh dynamics are detailed in Section 3.5. Finally, Sections 3.6, 3.7 and 3.8 introduce the numerical models used for each physics and implemented in the fluid and solid solvers, respectively, that will be used later as black-box tools.

3.1 Introduction to the different methodologies for computing FSI problems

This section introduces general aspects of numerical models for FSI systems. Firstly, one may distinguish low-fidelity models from high-fidelity models. The former usually give fast results but are limited in the range of physics they can represent, as opposed to the latter. Secondly, the difference between a one-way coupling, in which the motion of the solid is directly imposed, and a two-way coupling is detailed. Finally, the concepts of monolithic and partitioned approaches are extended from the mathematical formulation, as described in the previous chapter, to a numerical formulation.

3.1.1 Low-fidelity versus high-fidelity models

Simple fluid-structure models can be established for specific configurations. For instance, VIV of circular cylinders and two-mode flutter of airfoils can be represented by simplified numerical models involving rigid body motion described by Lagrange's equations (Appendix B) into which the aerodynamic forces have been introduced. The determination of these aerodynamic forces is then the main difficulty. The simplest solutions estimate these forces based on empirical aerodynamic coefficients. Low-fidelity aerodynamic models, for instance based on linearized potential flows and panels methods, are suitable for preliminary parametric studies because they provide fast results. They are extensively used in linear aeroelastic design. However, they are still limited in the representation of the complete physics of the problem. High-fidelity models are established based on nonlinear fluid/solid governing equations such as those introduced in the previous chapter. Such models are more costly but required when complex, usually nonlinear, phenomena play a significant role. The governing equations are discretized on computational meshes, which are the discrete version of the support domain Ω, by means of Finite Difference (FD), Finite Volume (FV) or Finite Element (FE) methods, to cite the most common ones. Sophisticated integration schemes and algorithms are used to solve the dynamics of the discretized system. Such high-fidelity methods, falling in the field of Computational Fluid Dynamics (CFD) and Computational Solid Mechanics (CSM), are used in this thesis. A mixed approach using both low- and high-fidelity models can also be envisaged. A typical example, also commonly used in linear aeroelasticity, is the coupling between a CFD solver and a linear structural model based on a modal approach [37]. In this context, the motion of the solid is decomposed into the weighted sum of the natural mode shapes $\boldsymbol{\phi}$. In practice, only a restricted number, M, of modes is retained for the computation, which gives

$$\boldsymbol{d}_{\mathrm{s}} = \sum_{m=1}^{M} q_m \boldsymbol{\phi}_m. \tag{3.1}$$

The mode shapes are then computed only once, with any CSD solver, and the generalized displacements q_m become the new unknowns which are obtained by solving an oscillator equation where the forcing term corresponds to the fluid loads projected on the mode shapes.

3.1.2 One-way versus two-way coupling

The most basic coupling approach is obtained when the fluid response to a prescribed motion of the structural part is calculated (note that it could also be the opposite, where the fluid loads are applied to a structure). The flow field then depends on the motion of the solid while the structural dynamics is not affected by the fluid loads. This approach may be called a one-way coupling. Most of the time, the prescribed motion of the solid is generated by the combination of several modes. In this case, the real-time coupling with a solid solver is not required, since only a modal analysis has to be performed prior to the FSI simulation. This approach has been applied for a long time due to its low computational cost. It is particularly suitable for the assessment of flutter stability in turbo-machinery applications, for which only small displacements around a (linear or nonlinear) fixed point are considered and for which the structural motion is usually well defined (shape, frequency and amplitude). The flow response is then analysed in order to estimate a flutter stability criterion in these particular conditions. The major inconvenience of the one-way coupling method is the assumption to be made for the vibration response of the structure which can be a significant source of errors in case of complex motion. Furthermore, flutter stability analysis through one-way coupling neglects the counteracting effect of the structure, which results in a too conservative prediction.

In the presence of strongly interacting (nonlinear) physics a two-way coupling is generally required. In this case, the calculated flow field is influenced by the response of the structure to the fluid flow, and conversely.

3.1.3 Monolithic versus partitioned coupling

The distinction established in the previous chapter between the partitioned and monolithic approaches for describing FSI problems can be extended to their numerical implementation. In the monolithic approach, the equations governing each coupled physics are solved simultaneously, possibly by using a unified mesh for the entire fluid-solid domain $\Omega_\mathrm{f} \cup \Omega_\mathrm{s}$. This brings the challenge of generating a high quality mesh for both the fluid and solid parts at the same time. The resulting unified framework, as depicted in Fig. 3.1, leads to the development of a specific solver with significant complexity [61]. Such a specialized code is most of the time developed to accommodate a particular case of interest, thereby often suffering some lack of generality and versatility. Also, the resulting system of equations to be solved is usually ill-conditioned due to the different orders of magnitude of the physical quantities in the coupled domains. However, the main advantage of the monolithic approach is its low sensitivity to possible numerical instabilities related to the coupling.

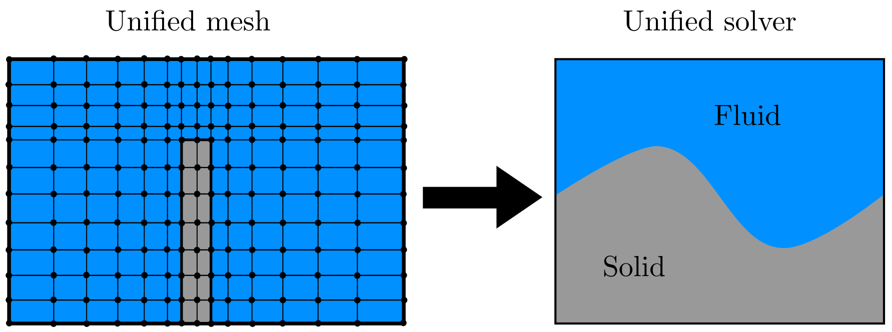


Figure 3.1: Illustration of the monolithic coupling architecture.

On the contrary, the partitioned approach, that is investigated in this thesis, allows us to solve each physics by using specialized and dedicated solvers on separate meshes with optimized and independent refinement. Existing codes that are already validated can be used as black-box tools, which significantly eases the access to all their capabilities and increases the generality of the approach. Furthermore, separate solvers are usually developed and maintained independently by different teams of experts. The partitioned approach becomes even more appealing when it allows the coupling of a large set of compatible codes that are based on well distinct physical and/or numerical models. However, in this approach the coupling conditions at the interface have to be explicitly treated by communicating information between the fluid and structure solvers. The development of interfacing tools and synchronization algorithms, as illustrated in Fig. 3.2, is still a challenging task. This difficulty is further compounded when, from a software design point of view, the two solvers were not developed to support this kind of communication with the external environment.

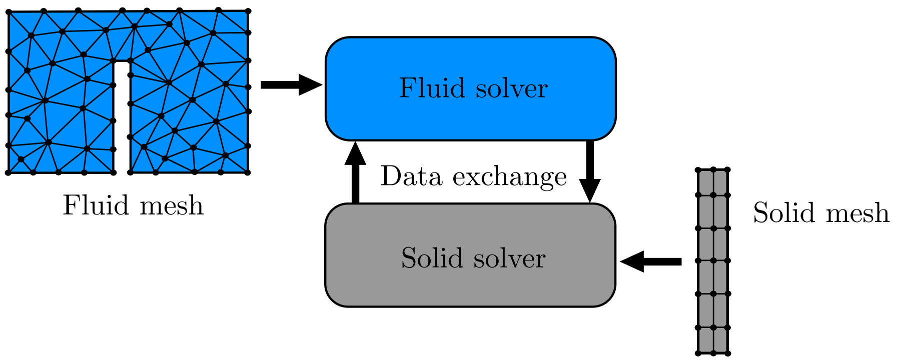


Figure 3.2: Illustration of the partitioned coupling architecture.

3.2 Mechanical coupling algorithm

The following section introduces the coupling algorithms that are developed for a partitioned two-way approach, assuming a body-fitted fluid mesh and a fluid solution computed with the ALE formalism[1]. Mechanical and thermal coupling are described separately for the sake of clarity. This separation allows us to develop all the relevant methodologies that are specific to each physics. The coupling algorithms for mechanical FSI are first introduced as a common basis and then are extended, in Section 3.3, to methodologies used for CHT.

3.2.1 Fixed-point formulation of the coupled problem

In the most general case, the formulation of the numerical model for mechanical fluid-structure interaction problems can be described by a three-field problem [62]: the fluid, the structure and the dynamical mesh resulting from the ALE description of the fluid problem. This formulation is written in the following general form,

$$\begin{cases} \mathcal{F}(\boldsymbol{u}, \boldsymbol{z}) = \boldsymbol{0} \\ \mathcal{S}(\boldsymbol{u}, \boldsymbol{d}) = \boldsymbol{0}\,, \\ \mathcal{M}(\boldsymbol{d}, \boldsymbol{z}) = \boldsymbol{0} \end{cases} \tag{3.2}$$

where $\mathcal{F}$, $\mathcal{S}$ and $\mathcal{M}$ represent the set of governing equations for the fluid, solid and mesh dynamics, respectively, or the associated solvers at the discrete level. In this system of equation, $\boldsymbol{u} = [\rho, \rho\boldsymbol{v}, \rho E]^{\mathrm{T}}$ is the vector of the fluid conservative variables, $\boldsymbol{d}$ the displacement vector of the structure and $\boldsymbol{z}$ the displacement vector of the fluid mesh. An iterative procedure to solve system (3.2) can be developed by writing the linearized form of the system:

$$\begin{bmatrix} \mathcal{F} \\ \\ \mathcal{S} \\ \\ \mathcal{M} \end{bmatrix} + \begin{bmatrix} \dfrac{\partial \mathcal{F}}{\partial \boldsymbol{u}} & \boldsymbol{0} & \dfrac{\partial \mathcal{F}}{\partial \boldsymbol{z}} \\ \dfrac{\partial \mathcal{S}}{\partial \boldsymbol{u}} & \dfrac{\partial \mathcal{S}}{\partial \boldsymbol{d}} & \boldsymbol{0} \\ \boldsymbol{0} & \dfrac{\partial \mathcal{M}}{\partial \boldsymbol{d}} & \dfrac{\partial \mathcal{M}}{\partial \boldsymbol{z}} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{u} \\ \\ \Delta \boldsymbol{d} \\ \\ \Delta \boldsymbol{d}_{\Omega} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \\ \boldsymbol{0} \\ \\ \boldsymbol{0} \end{bmatrix}, \tag{3.3}$$

that can be solved by a Newton-Raphson procedure, for instance. However, in practice, an exact calculation of the tangent matrix in Eq. (3.3) would require the computation and the access to all Jacobians, which is highly cumbersome and almost impossible in the case of a partitioned framework with independent solvers. Consequently, the coupling is reformulated as a Dirichlet-Neumann partitioning. In order to represent the fluid side of the coupling, it is usual to define a nonlinear Dirichlet operator $\mathcal{D}_{\mathrm{f}}$ that computes the fluid loads at the interface Γ from a given fluid interface displacement,

$$\boldsymbol{t}_{\mathrm{f}}^{\Gamma} = \mathcal{D}_{\mathrm{f}}(\boldsymbol{d}_{\mathrm{f}}^{\Gamma})\,. \tag{3.4}$$

This nonlinear operator may be seen as a black-box abstraction of a generic fluid solver. Analogously, a nonlinear Neumann operator $\mathcal{D}_{\mathrm{s}}$ that computes the solid interface displacement as a function of the solid loads is defined by

$$\boldsymbol{d}_{\mathrm{s}}^{\Gamma} = \mathcal{D}_{\mathrm{s}}(\boldsymbol{t}_{\mathrm{s}}^{\Gamma})\,, \tag{3.5}$$

as a black-box abstraction of a generic solid solver. Combining the definition of these two operators with the continuity conditions defined in Eqs. (2.27), the coupling can be reformulated as the fixed-point problem

$$\boldsymbol{d}^{\Gamma} = \mathcal{D}_{\mathrm{s}} \circ \mathcal{D}_{\mathrm{f}}\left(\boldsymbol{d}^{\Gamma}\right)\,, \tag{3.6}$$

[1]The use of non-conforming fluid mesh such as in the Immersed Boundary method is discussed further in this chapter.

where $\boldsymbol{d}^\Gamma$ is the displacement of the fluid-solid interface. It is important to note that the dynamic fluid mesh operator does not appear explicitly in the fixed-point equation because it is implicitly included in the definition of $\mathcal{D}_\mathrm{f}$ as an intrinsic task of the fluid computation. The fixed-point equation leads to a more intuitive way of representing the partitioned coupling. It is also well suited for the description of partitioned problems since it focuses only on interface variables by embedding the treatment of the domain (fluid or solid) variables into the black-box operators $\mathcal{D}_\mathrm{f}$ or $\mathcal{D}_\mathrm{s}$. The fixed-point equation is solved by a sequential iterative procedure that can be described as follows: for a given displacement of the fluid-structure interface, a fluid computation is first performed to obtain the fluid loads. These loads are then transferred as inputs to the structural solver for the computation of a new interface position, that is then used as new fluid boundary conditions for the next coupling iteration. One can distinguish between two main procedures to solve the resulting coupled problem: the weak, or loose, coupling and the strong coupling. Explicit weak coupling, as described in Section 3.2.2, is obtained when only one coupling iteration is performed at each time step. In other words, the fluid and solid solvers compute their solutions and exchange their boundary conditions only once per time step, with no corrective feedback to ensure the coupling conditions. A strongly-coupled implicit scheme is obtained when the coupling conditions on the fluid-structure interface at each time step are met by iterating between the fluid ($\mathcal{D}_\mathrm{f}$) and solid ($\mathcal{D}_\mathrm{s}$) computations and by exchanging their boundary conditions until the interfacial conditions are converged. Strongly coupled procedures are detailed in Sections 3.2.3 and 3.2.4.

3.2.2 Loosely-coupled procedure

The loosely-coupled approach is illustrated in Fig. 3.3 for a time-marched computation and formally described by Alg. 1:

Algorithm 1 Loosely-coupled algorithm

```
1: Enter time step $n$ and advance the fluid-mesh position based on the last calculated solid displacement, $\mathcal{M}\left(\boldsymbol{z}^{n-1}, \boldsymbol{d}^{n-1}, \boldsymbol{z}^n\right) = 0$ and compute grid velocity $\dot{\boldsymbol{z}}^n = \dot{\boldsymbol{z}}^n\left(\boldsymbol{z}^n, \boldsymbol{z}^{n-1}\right)$
2: Set the grid velocity in the fluid solver, $\mathcal{M} \xrightarrow{\dot{\boldsymbol{z}}^n} \mathcal{F}$
3: Advance the fluid solution with the fluid solver, $\mathcal{F}\left(\boldsymbol{u}^{n-1}, \dot{\boldsymbol{z}}^n, \boldsymbol{z}^n, \boldsymbol{u}^n\right) = 0$
4: Communicate the fluid interface loads to the solid solver, $\mathcal{F} \xrightarrow{\boldsymbol{t}^{\Gamma,n}} \mathcal{S}$
5: Advance the solid solution with the solid solver, $\mathcal{S}\left(\boldsymbol{d}^{n-1}, \boldsymbol{u}^n, \boldsymbol{d}^n\right) = 0$
6: Communicate the solid interface displacement to the mesh dynamic module, $\mathcal{S} \xrightarrow{\boldsymbol{d}^{\Gamma,n}} \mathcal{M}$
7: Go to the next time step with $n := n + 1$
```

In this approach, the fluid and solid solvers exchange their interface data only once per time step with no guarantee that these data satisfy the coupling conditions. The absence of iterative correction minimizes the amount of fluid, mesh and solid computations that have to be performed at each time step, thus reducing the computation time. However, since the solutions are advanced in a time-lagged fashion, the forces induced by the fluid on the structure are computed from displacements and velocities obtained at the previous time step. This results in an energy non-conserving coupling that is more prone to numerical instabilities, as detailed further in this chapter. Nevertheless, weak coupling can be efficiently applied when only a steady state coupled solution is sought.

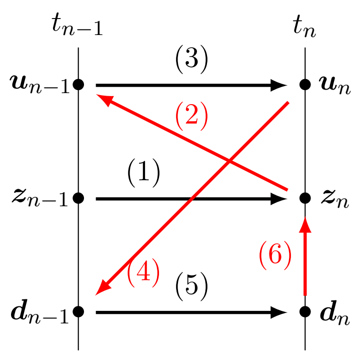


Figure 3.3: Time step advancement procedure for a loosely-coupled scheme. Black: time advancement steps, red: communication steps. The order of the different operations is indicated by the number in parenthesis next to each arrow.

Remark

A link between the general operators $(\mathcal{F}, \mathcal{M}, \mathcal{S})$ and the restricted Dirichlet-Neumann operators $(\mathcal{D}_\mathrm{f}, \mathcal{D}_\mathrm{s})$ can be established. More specifically, the sequence that computes the fluid loads from a given displacement of the interface in Alg. 1 corresponds to (1)→(3) and thus represents $\mathcal{D}_\mathrm{f}$, in which the fluid mesh dynamic is implicitly embedded. The operator $\mathcal{D}_\mathrm{s}$ is then simply associated with the sequence (5) of Alg. 1.

3.2.3 Strongly-coupled procedure

The strongly-coupled scheme is illustrated in Fig. 3.4 for a time-marched computation. The time advancement of the three-field system is described by Alg. 2.

Algorithm 2 Strongly-coupled algorithm

```
1: Enter time step $n$ with $j = 0$
2: Set $\boldsymbol{d}_0^n = \boldsymbol{d}^{n-1}$
3: repeat
4:     Set $j := j + 1$
5:     Advance the fluid-mesh position based on the last calculated solid displacement, $\mathcal{M}\left(\boldsymbol{z}^{n-1}, \boldsymbol{d}_{j-1}^n, \boldsymbol{z}_j^n\right) = 0$
6:     Compute grid velocity $\dot{\boldsymbol{z}}_j^n = \dot{\boldsymbol{z}}_j^n\left(\boldsymbol{z}_j^n, \boldsymbol{z}^{n-1}\right)$
7:     Set grid velocity in the fluid solver, $\mathcal{M} \xrightarrow{\dot{\boldsymbol{z}}_j^n} \mathcal{F}$
8:     Advance the fluid solution with the fluid solver, $\mathcal{F}\left(\boldsymbol{u}^{n-1}, \dot{\boldsymbol{z}}_j^n, \boldsymbol{u}_j^n\right) = 0$
9:     Communicate the fluid interface loads to the solid solver, $\mathcal{F} \xrightarrow{\boldsymbol{t}_j^{\Gamma,n}} \mathcal{S}$
10:    Advance the solid solution with the solid solver, $\mathcal{S}\left(\boldsymbol{d}^{n-1}, \boldsymbol{u}_j^n, \boldsymbol{d}_j^n\right) = 0$
11:    Communicate the solid interface displacement to the mesh dynamic solver, $\mathcal{S} \xrightarrow{\boldsymbol{d}_j^{\Gamma,n}} \mathcal{M}$
12:    Compute coupling residual $\boldsymbol{r}_j^\Gamma = \boldsymbol{d}_j^{\Gamma,n} - \boldsymbol{d}_{j-1}^{\Gamma,n}$
13: until $||\boldsymbol{r}_j|| < \varepsilon$
14: Set $\boldsymbol{d}^n = \boldsymbol{d}_j^n$
15: Go to the next time step with $n := n + 1$
```

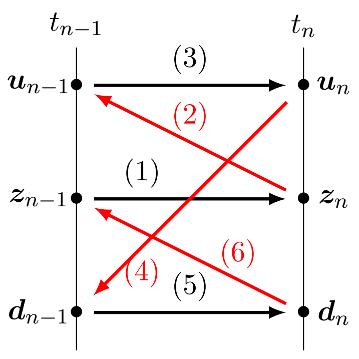


Figure 3.4: Time step advancement procedure for a strongly coupled scheme. Black: time advancement steps, red: communication steps. The order of the different operations is indicated by the number in parenthesis next to each arrow.

In the strongly-coupled approach, the fluid and solid solvers exchange their interface data several times per time step within a iterative coupling procedure, represented here by the iterator j, in order to guarantee the convergence of the interfacial conditions. The iterative procedure is stopped as soon as the convergence criterion is met. In this case, convergence is assessed based on a residual vector that is defined as the difference between the computed solid displacement at the current iteration and the last computed displacement from the previous iteration:

$$\boldsymbol{r}_j^\Gamma = \boldsymbol{d}_j^{\Gamma,n} - \boldsymbol{d}_{j-1}^{\Gamma,n} \,. \tag{3.7}$$

At the first iteration of each time step ($j = 0$), the residual is evaluated based on the last converged displacement from the previous time step:

$$\boldsymbol{r}_0^\Gamma = \boldsymbol{d}_0^{\Gamma,n} - \boldsymbol{d}^{\Gamma,n-1} \,. \tag{3.8}$$

The convergence criterion is satisfied if

$$||\boldsymbol{r}_j^\Gamma|| < \varepsilon \,, \tag{3.9}$$

where ε is a case-dependent dimensional tolerance that is set for a given problem. The described iterative procedure corresponds to a strongly-coupled block-Gauss-Seidel (BGS) method [63], where the fluid, fluid-mesh and structural problems are solved sequentially, which is equivalent to neglecting the off-diagonal terms in the tangent matrix of Eq. (3.3). Since the solution in each domain has be to computed several times per time step, the computational cost of strong coupling is higher compared to weak coupling. However the strongly-coupled approach is more robust and provides the same level of accuracy as monolithic coupling [47].

3.2.4 Strongly coupled procedure with time step prediction

A variant of the strongly-coupled BGS method is introduced in order to improve the convergence of the iterative coupling at each time step and thus reduce the number of coupling iterations required to reach a given tolerance. This is illustrated in Fig. 3.5. At the end of a converged time step, the solid interface position at the next time step is first predicted and used as a guess for the first fluid computation. The efficiency of the predictor step in reducing the number of coupling iterations usually depends on the quality of the prediction. The predicted displacement $\boldsymbol{d}^\Gamma_{\mathrm{pred},n+1}$ at time step $n{+}1$ is computed based on converged interface displacements and velocities obtained at previous time steps:

$$\boldsymbol{d}^\Gamma_{\mathrm{pred},n+1} = \boldsymbol{d}^\Gamma_n + \alpha_0\,\Delta t\,\dot{\boldsymbol{d}}^\Gamma_n + \alpha_1\,\Delta t\left(\dot{\boldsymbol{d}}^\Gamma_n - \dot{\boldsymbol{d}}^\Gamma_{n-1}\right)\,, \tag{3.10}$$

in which $\alpha_0 = 1$ and Δt is the time step of the temporal discretization. A second- or first-order prediction is obtained by setting $\alpha_1 = 0.5$ or $\alpha_1 = 0$, respectively. Eq. (3.10) is obtained by a simple Taylor expansion truncated at order two and where the second derivative of the interface displacement (or interface acceleration) is approximated by finite differences. The time advancement procedure taking into account the prediction step is described by Alg. 3.

Algorithm 3 Strongly-coupled algorithm with prediction

```
 1: Enter time step $n$ with $j = 0$
 2: Set $\boldsymbol{d}_0^n = \boldsymbol{d}_{\text{pred}}^n$
 3: repeat
 4:     Set $j := j + 1$
 5:     Advance the fluid-mesh position based on the last calculated solid displacement, $\mathcal{M}\left(\boldsymbol{z}^{n-1}, \boldsymbol{d}_{j-1}^n, \boldsymbol{z}_j^n\right) = 0$
 6:     Compute grid velocity $\dot{\boldsymbol{z}}_j^n = \dot{\boldsymbol{z}}_j^n\left(\boldsymbol{z}_j^n, \boldsymbol{z}^{n-1}\right)$
 7:     Set grid velocity in the fluid solver, $\mathcal{M} \xrightarrow{\dot{\boldsymbol{z}}_j^n} \mathcal{F}$
 8:     Advance the fluid solution with the fluid solver, $\mathcal{F}\left(\boldsymbol{u}^{n-1}, \dot{\boldsymbol{z}}_j^n, \boldsymbol{u}_j^n\right) = 0$
 9:     Communicate the fluid interface loads to the solid solver, $\mathcal{F} \xrightarrow{\boldsymbol{t}_j^{\Gamma,n}} \mathcal{S}$
10:     Advance the solid solution with the solid solver, $\mathcal{S}\left(\boldsymbol{d}^{n-1}, \boldsymbol{u}_j^n, \boldsymbol{d}_j^n\right) = 0$
11:     Communicate the solid interface displacement to the mesh dynamic solver, $\mathcal{S} \xrightarrow{\boldsymbol{d}_j^{\Gamma,n}} \mathcal{M}$
12:     Compute coupling residual $\boldsymbol{r}_j^\Gamma = \boldsymbol{d}_j^{\Gamma,n} - \boldsymbol{d}_{j-1}^{\Gamma,n}$
13: until $||\boldsymbol{r}_j|| < \varepsilon$
14: Set $\boldsymbol{d}^n = \boldsymbol{d}_j^n$
15: Predict interface displacement $\boldsymbol{d}_{\text{pred}}^n$ using Eq. (3.10).
16: Go to the next time step with $n := n + 1$
```

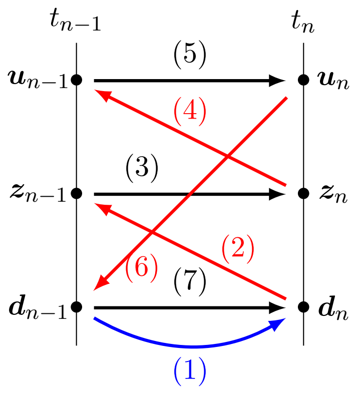


Figure 3.5: Time step advancement procedure for a predictive strongly coupled scheme. Black: time advancement steps, red: communication steps, blue: prediction step.

3.2.5 Stability of the coupling procedure and Aitken relaxation for the strongly coupled scheme

One of the major drawbacks of partitioned methods is their relative sensitivity to numerical instabilities induced by the coupling. These instabilities are a consequence of the iterative coupling procedure and are encountered when the interaction between the fluid and the solid features a significant feedback effect (i.e. a change in the solution of one physics has a strong impact on the solution of the other physics). For loosely coupled solutions, instabilities will cause an unbounded increase of the structural displacement in time while strongly coupled solutions will require a significantly higher number of coupling iterations or even diverge in the most severe cases. If not properly identified, these numerical instabilities might be erroneously taken for physical instabilities such as flutter, hence potentially altering the engineering design. Historically, partitioned coupling instabilities have usually been referred to as added-mass effect [47]. The added-mass effect can be understood as additional structural inertia resulting from the displacement of a surrounding fluid when the solid is moving. In other words, the added mass represents the inertia of the fluid that has to be displaced by the solid. In order to illustrate the added-mass effect, let us consider a simplified fluid-structure model. A rigid body of mass m attached to a spring of stiffness k is immersed in a still fluid, as illustrated in Fig. 3.6, where the fluid domain is supposed to extend to infinity. It is assumed that the solid rigid displacement η takes the form $\eta = q(t)\boldsymbol{\phi}$, where q is an unknown function of time only and $\boldsymbol{\phi} = \boldsymbol{e}_y$ a well defined modal shape corresponding here to a vertical rigid displacement. A simple application of Lagrange's equation [64] leads to a standard oscillator equation,

$$m\ddot{q} + kq = \zeta \tag{3.11}$$

where ζ is the resultant aerodynamic force applied by the fluid projected along the solid motion. Assuming small amplitudes of the solid motion a linearization of this fluid-structure system (details can be found in Appendix C) leads to a modified structural equation under the form,

$$(m + m_a)\,\ddot{q} + kq = 0\,, \tag{3.12}$$

where m_a is the added mass that mainly depends on the fluid pressure field. In other words, Equation (3.12) demonstrates that the motion of the solid in the fluid corresponds to its free motion in vacuum (i.e. without a surrounding medium) but with an increased inertia term that accounts for fluid-solid interaction, as illustrated in Fig. 3.6. In this simple case, a direct consequence of the presence of the fluid on the free response of the solid is a modification of the natural frequency of the oscillator.

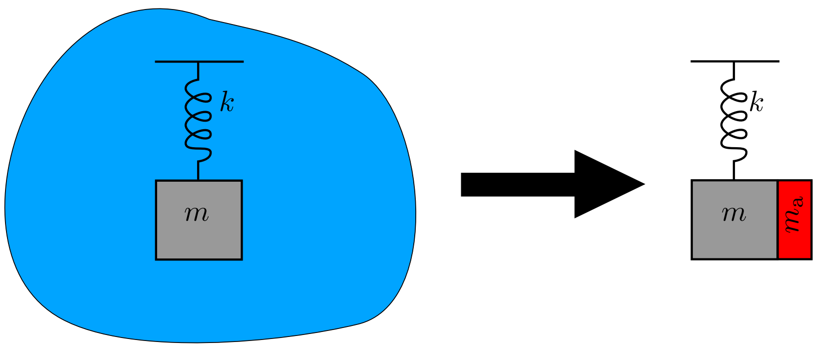


Figure 3.6: Illustration of the added-mass effect for the simplified fluid-solid coupled problem.

A characterization of the added-mass effect is much less straightforward to obtain for more realistic fluid-structure systems due to the geometrical complexity and nonlinear physics involved, but simplified approaches still provide a means to highlight the main features of the added-mass effect. Causin *et al.* [65] describe a simplified fluid-structure problem involving inviscid incompressible flows in order to show that, similarly to what was done earlier, the resulting structure equation is modified by the introduction of an added-mass operator representing the effect of the fluid. After a spectrum analysis of the added-mass operator and a stability analysis of the chosen time discretization scheme, they show that a loosely coupled algorithm is unconditionally unstable if

$$\frac{\rho_{\mathrm{s}}\, L_{\mathrm{s}}}{\rho_{\mathrm{f}}\, \hat{\mu}_{\max}} < 1\,, \tag{3.13}$$

where L_{s} is a parameter related to the geometry of the solid and $\hat{\mu}_{\max}$ is the largest eigenvalue of the added-mass operator. The stability criterion shows an explicit dependence on the mass number Ma. When the solid density is much higher than the fluid density (Ma $\gg 1$), a stable explicit coupling scheme can be performed. This is for instance typically the case for aeronautical applications. For biological flows, such as arterial blood flows, where the fluid density approaches the solid density, coupling instabilities are more likely to appear. Furthermore, it should be noted that the stability criterion (3.13) of the coupled procedure does not depend on the time step size: reducing the time step does not stabilize the solution as it is usually the case for common explicit procedures. On the contrary, numerical observations [65, 66] show that the instability appears sooner in the simulated time as the time step decreases. Analogously, increasing the temporal accuracy by using predictors or higher order time schemes (e.g. for the fluid solution) [66, 67] has also a destabilizing effect. Numerical observations also show that fluid viscosity has a destabilizing effect whereas the structural stiffness has a stabilizing influence, as indicated by Förster *et al.* [66].

The implicit strongly coupled BGS procedure is used to guarantee accuracy in the solution of the coupled system and to stabilize computations where weak coupling suffers from numerical instabilities. However, it has been found that the BGS procedure does not always ensure unconditional stability. In the most severe cases, the coupling residual increases at each iteration and the solution diverges. A standard method to stabilize the iterative procedure is to introduce numerical relaxation when updating the interface displacement $\boldsymbol{d}^{\Gamma}$ after the solid computation and the evaluation of the coupling residual (3.7). Specifically, the relaxed displacement at a coupling iteration j is calculated as:

$$\boldsymbol{d}_j^{\Gamma} = \boldsymbol{d}_{j-1}^{\Gamma} + \omega_j \boldsymbol{r}_j^{\Gamma}\,, \tag{3.14}$$

with ω being the relaxation parameter such that $0 < \omega \leq 1$. This includes the particular case $\omega = 1$ where no relaxation is applied. Note that the expression for the residual is slightly modified such that

$$\boldsymbol{r}_j^{\Gamma} = \tilde{\boldsymbol{d}}_j^{\Gamma} - \boldsymbol{d}_{j-1}^{\Gamma}\,, \tag{3.15}$$

where $\tilde{\boldsymbol{d}}^{\Gamma}$ represents the displacement actually computed by the solid solver and $\boldsymbol{d}^{\Gamma}$ the relaxed displacement. Causin *et al.* [65] also performed a stability analysis of the strong Dirichlet-Neumann coupling scheme for a simplified system and developed the following stability criterion for the relaxation parameter:

$$0 < \omega < \frac{2\left(\rho_{\mathrm{s}}\, L_{\mathrm{s}} + C\Delta t^2\right)}{\rho_{\mathrm{s}}\, L_{\mathrm{s}} + \rho_{\mathrm{f}}\, \mu_{\max} + C\Delta t^2}\,, \tag{3.16}$$

where C is a constant related to the solid stiffness. This criterion highlights the effect of the time step, the mass ratio and the stiffness on the stability. An increase of the mass ratio ($\rho_{\mathrm{s}} \gg \rho_{\mathrm{f}}$) or an increase of the solid stiffness and/or time step are all stabilizing effects in a sense that they increase the upper limit of the relaxation parameter. Conversely, the stability range for ω shortens when the fluid density becomes comparable to or larger than the solid density or when the solid stiffness and the time step decreases. As stated in Causin *et al.* [65], it is interesting to note that for a time step approaching zero ($\Delta t \to 0$),

$$0 < \omega < \frac{2}{1 + \frac{\rho_{\mathrm{f}}\, \mu_{\max}}{\rho_{\mathrm{s}}\, L_{\mathrm{s}}}}\,, \tag{3.17}$$

and thus whenever the explicit scheme diverges ($\rho_\mathrm{f}\,\mu_\mathrm{max} > \rho_\mathrm{s}\,L_\mathrm{s}$), the implicit relaxed BGS will only converge if ω is below a threshold value that is strictly smaller than one.

The stability criteria (3.13) for explicit and (3.16) for implicit schemes provide good insight into the stability problem induced by added-mass effects. However, they were derived for a strongly simplified case and their generalization to more complex FSI problems involving highly nonlinear viscous flows and complex geometries is not feasible. Whereas this simplified analysis suggests that explicit schemes should not be used when strong added-mass effects are expected, it does not provide the optimal value of the relaxation parameter ω, which is clearly problem-specific. If the chosen value of ω falls beyond the stability limit, the iterative scheme diverges. Conversely, if the chosen value is much below the limit, the iterative procedure remains stable but converges very slowly, which is computationally very inefficient. A good trade-off between stability and convergence rate can be achieved by adapting dynamically the value of the relaxation parameter at each iteration. The most common and efficient method to compute ω dynamically is Aitken's Δ^2 method [63, 68]. The central idea of the method is to use values from two previous FSI iterations in order to improve the solution of the coupled procedure. Assuming a scalar interface displacemnt d^Γ, the iterative procedure can be expressed as finding the root d_j^Γ of $f(d_j^\Gamma) = \tilde{d}_{j+1}^\Gamma - d_j^\Gamma = 0$. Dropping momentarily the superscript Γ for simplicity, the application of the recursive secant method leads to

$$\begin{aligned} d_j &= \frac{d_{j-2}\,f(d_{j-1}) - d_{j-1}\,f(d_{j-2})}{f(d_{j-1}) - f(d_{j-2})} \\ &= \frac{d_{j-2}\,(\tilde{d}_j - d_{j-1}) - d_{j-1}\,(\tilde{d}_{j-1} - d_{j-2})}{(\tilde{d}_j - d_{j-1}) - (\tilde{d}_{j-1} - d_{j-2})} \\ &= \frac{d_{j-2}\,\tilde{d}_j - d_{j-1}\,\tilde{d}_{j-1}}{r_j - r_{j-1}} \end{aligned} \tag{3.18}$$

where the scalar residual has been defined as $r_j = \tilde{d}_j - d_{j-1}$. From Eq. (3.14) the relaxation parameter can be written as

$$\omega_j = \frac{d_j - d_{j-1}}{r_j}\,, \tag{3.19}$$

and replacing d_j by Eq. (3.18), we obtain, after few simplifications,

$$\omega_j = \frac{d_{j-2} - d_{j-1}}{r_j - r_{j-1}}\,. \tag{3.20}$$

The next step consists in replacing d_{j-1} in this last expression using again Eq. (3.14) with $j := j-1$. After a few more simplifications, one finally obtains the recursive formula

$$\omega_j = -\omega_{j-1}\,\frac{r_{j-1}}{r_j - r_{j-1}}\,. \tag{3.21}$$

For the more general vector case, a simple division cannot be defined. It is thus suggested by Irons and Tuck [68] to project $\boldsymbol{r}_{j-1}$ on the normalized vector $\boldsymbol{r}_j - \boldsymbol{r}_{j-1}$, leading to the final and general expression:

$$\omega_j = -\omega_{j-1}\,\frac{(\boldsymbol{r}_{j-1}^\Gamma)^\mathrm{T}\,(\boldsymbol{r}_j^\Gamma - \boldsymbol{r}_{j-1}^\Gamma)}{||\boldsymbol{r}_j^\Gamma - \boldsymbol{r}_{j-1}^\Gamma||^2}\,, \tag{3.22}$$

in which the superscript Γ has been reintroduced for sake of completeness in the notation. Since the recursive formula (3.22) requires the value of the relaxation parameter ω_{j-1} at the previous iteration, it cannot be applied to the first iteration of a time step. The first relaxation step is thus computed with a fixed value ω_0. The last ω_n obtained at the previous time step can be used, but it is usually better to constrain it with an upper or lower bound such that

$$\omega_0^i = \min(\omega_n^{i-1}, \bar{\omega}) \quad \text{or} \quad \max(\omega_n^{i-1}, \bar{\omega})\,, \tag{3.23}$$

where $\bar{\omega}$ is a problem-specific fixed parameter. Typically, the choice of an upper bound (by using the min criterion) is more conservative from a stability point of view but can lead to a higher number of coupling iterations and thus slower convergence of the iterative procedure. The coupling algorithm using Aitken's relaxation in combination with a prediction step is described by Alg. 4 for one time step. In order to highlight the relaxation procedure, Alg. 4 uses the simplified description of the fluid and solid operators based on the fixed-point problem (3.6).

Algorithm 4 Aitken's Δ^2 relaxation algorithm

```
 1: Enter time step with predicted value $\boldsymbol{d}_{\text{pred},0}$
 2: Call fluid solver to compute $\tilde{\boldsymbol{t}}_0 = \mathcal{D}_{\text{f}}(\boldsymbol{d}_{\text{pred},0})$
 3: Call solid solver to compute $\tilde{\boldsymbol{d}}_0 = \mathcal{D}_{\text{s}}(\tilde{\boldsymbol{t}}_0)$
 4: Compute residual $\boldsymbol{r}_0 = \tilde{\boldsymbol{d}}_0 - \boldsymbol{d}_{\text{pred},0}$
 5: Choose $\omega_0$ with Eq. (3.23)
 6: Perform static relaxation $\boldsymbol{d}_0 = \boldsymbol{d}_{\text{pred},0} + \omega_0\, \boldsymbol{r}_0$
 7: $j = 1$
 8: while $||\boldsymbol{r}_{j-1}|| > \varepsilon$ do
 9:     Call fluid solver $\tilde{\boldsymbol{t}}_j = \mathcal{D}_{\text{f}}(\boldsymbol{d}_{j-1})$
10:     Call solid solver $\tilde{\boldsymbol{d}}_j = \mathcal{D}_{\text{s}}(\tilde{\boldsymbol{t}}_j)$
11:     Compute residual $\boldsymbol{r}_j = \tilde{\boldsymbol{d}}_j - \boldsymbol{d}_{j-1}$
12:     Compute $\omega_j$ using Eq. (3.22)
13:     Perform relaxation $\boldsymbol{d}_j = \boldsymbol{d}_{j-1} + \omega_j \boldsymbol{r}_j$
14:     $j = j + 1$
15: end while
16: Predict the value of the displacement and go to next time step
```

The use of Aitken's Δ^2 accelerator is appealing because it is relatively simple to implement. The update of the relaxation parameter only requires the evaluation of residuals based on data that are easily available from the solid solver.

Another interesting accelerator is described by Küttler *et al.* [63] and called the steepest descent relaxation. A minimization problem is posed in order to finally approximate the relaxation parameter with

$$\omega_j = -\frac{\left(\boldsymbol{r}_j^\Gamma\right)^{\text{T}} \boldsymbol{r}_j^\Gamma}{\left(\boldsymbol{r}_j^\Gamma\right)^{\text{T}} \mathbf{J}^\Gamma \boldsymbol{r}_j^\Gamma}\,, \tag{3.24}$$

with the interface Jacobian given by

$$\mathbf{J}^\Gamma = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{d}}\,. \tag{3.25}$$

The calculation of the relaxation parameter by the steepest descent requires now a matrix vector product $\mathbf{J}^\Gamma \boldsymbol{r}_i^\Gamma$ and the evaluation of the interface Jacobian which is not explicitly available in the procedure. Küttler *et a.* [63] propose two calculation methods for the matrix vector product, either via finite difference or via approximated fluid derivative, but in both cases the complexity of the procedure increases.

3.2.6 Newton-based techniques for the strongly coupled problem

The relaxation with Aitken's Δ^2 acceleration is the most widely used method because it is fast and cheap to implement and because it provides good performance for otherwise unstable cases. However, a Newton-based family of coupling techniques has been developed over the past few years in order to further improve the convergence of the iterative procedure. For the standard solution of nonlinear systems, Newton techniques have been shown to provide better convergence rate than the Gauss-Seidel approach. Their application to FSI systems has been driven by the recent gain in popularity of coupled simulations in the biomedical/biological field where added-mass effects are prominent due to the presence of denser fluids and/or softer solids. Assuming a constant cost associated with one call of each of the coupled solvers, one can easily understand that reducing the number of coupling iterations per time step leads to a considerable decrease of the overall simulation cost.

An alternative to the fixed-point equation (3.6) is first obtained by combining the fluid and solid operators, defined by Eqs. (3.4) and (3.5) respectively, with the continuity conditions (2.27) into the problem of finding the root of the system

$$\begin{aligned} \boldsymbol{r}_\mathrm{f} &= \mathcal{D}_\mathrm{f}(\boldsymbol{d}) - \boldsymbol{t} = \mathbf{0}\,, \\ \boldsymbol{r}_\mathrm{s} &= \mathcal{D}_\mathrm{s}(\boldsymbol{t}) - \boldsymbol{d} = \mathbf{0}\,. \end{aligned} \tag{3.26}$$

This system can be solved by a standard Newton-Raphson iterative procedure,

$$\begin{bmatrix} \dfrac{\partial \boldsymbol{r}_\mathrm{f}}{\partial \boldsymbol{d}} & \dfrac{\partial \boldsymbol{r}_\mathrm{f}}{\partial \boldsymbol{t}} \\ \dfrac{\partial \boldsymbol{r}_\mathrm{s}}{\partial \boldsymbol{d}} & \dfrac{\partial \boldsymbol{r}_\mathrm{s}}{\partial \boldsymbol{t}} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{d} \\ \Delta \boldsymbol{t} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_\mathrm{f} \\ \boldsymbol{r}_\mathrm{s} \end{bmatrix}, \tag{3.27}$$

where the updated value of the displacements and loads at iteration $k+1$ is obtained from their previous value at iteration k and the solution of the above system:

$$\begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{t} \end{bmatrix}_{k+1} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{t} \end{bmatrix}_{k} + \begin{bmatrix} \Delta \boldsymbol{d} \\ \Delta \boldsymbol{t} \end{bmatrix}_{k}. \tag{3.28}$$

A full description of the Jacobian in Eq. (3.27) is given by Bogaers *et al.* [69]:

$$\mathbf{J}_\mathbf{r} = \begin{bmatrix} \dfrac{\partial \mathcal{D}_\mathrm{f}}{\partial \boldsymbol{d}} & \dfrac{\partial \mathcal{D}_\mathrm{f}}{\partial \boldsymbol{d}} \dfrac{\partial \boldsymbol{d}}{\partial \boldsymbol{t}} - \mathbf{I} \\ \dfrac{\partial \mathcal{D}_\mathrm{s}}{\partial \boldsymbol{t}} \dfrac{\partial \boldsymbol{t}}{\partial \boldsymbol{d}} - \mathbf{I} & \dfrac{\partial \mathcal{D}_\mathrm{s}}{\partial \boldsymbol{t}} \end{bmatrix}. \tag{3.29}$$

Recalling Eq. (3.26) and introducing the Jacobians of the fluid and solid solvers that satisfy $\mathbf{J}_\mathrm{f} = \frac{\partial \mathcal{D}_\mathrm{f}}{\partial \boldsymbol{d}} = \frac{\partial \boldsymbol{t}}{\partial \boldsymbol{d}}$ and $\mathbf{J}_\mathrm{s} = \frac{\partial \mathcal{D}_\mathrm{s}}{\partial \boldsymbol{t}} = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{t}}$ respectively, the final expression for the overall Jacobian is

$$\mathbf{J}_\mathbf{r} = \begin{bmatrix} \mathbf{J}_\mathrm{f} & \mathbf{J}_\mathrm{f}\mathbf{J}_\mathrm{s} - \mathbf{I} \\ \mathbf{J}_\mathrm{s}\mathbf{J}_\mathrm{f} - \mathbf{I} & \mathbf{J}_\mathrm{s} \end{bmatrix}. \tag{3.30}$$

In a partitioned framework where the fluid and solid solvers are executed in a staggered fashion, the system of linear equations is solved in a block-Newton way [70]. The full Newton method can be applied only if the Jacobians of the fluid/solid solvers $\mathbf{J}_\mathrm{f}$ and $\mathbf{J}_\mathrm{s}$ are known. In a partitioned approach using black-box solvers, these Jacobians are almost impossible to determine or would need specifically enhanced differentiated solvers. Instead, Vierendeels *et al.* [71] adopted a quasi-Newton approach where these Jacobians, and more exactly the product between Jacobians and vectors, are only approximated by reduced order models based on the resolution of a least square problem that combines input/output variables of each coupled solver. This leads to the so-called interface block quasi-Newton with approximation of the Jacobian from least square (IBQN-LS)[2] method. The full description of the approximation of the Jacobians can be found in Vierendeels *et al.* [71] and Degroote *et al.* [70]. The central idea behind reduced order modeling is

[2]In the denomination IBQN (interface block quasi-Newton), the term *interface* means that the Newton procedure is applied only on variables defined at the fluid-structure interface, the term *block* means that the Jacobian is made of distinct blocks related to the coupled solvers and finally the term *quasi* is applied when the Jacobian is approximated instead of calculated exactly.

to accumulate and store information coming from the inputs/outputs of the solvers over coupling iterations. This information is used to build an approximation of the Jacobians that increases the convergence of the iterative procedure when compared to the standard Aitken's Δ^2 method. However, and because the IBQN-LS method needs at least two coupling iterations to create the reduced order models, an Aitken relaxation, usually performed with ω small enough to avoid rapid divergence, is used at the first iteration. Degroote *et al.* [67] compared IBQN-LS and the Aitken relaxation and showed that, unlike Aitken's method, IBQN provides a monotonic convergence once the reduced order models are available. They also found that, for cases with poor stability, the number of coupling iterations increases when the stiffness of the structure or the time step are reduced, irrespective of the method used. However, IBQN performs significantly better than Aitken in limiting this increase of iterations.

The IBQN-LS, although improving the convergence of the coupling procedure, has two major drawbacks. First, the block root finding form of the coupled problem is still solved in a staggered way where (reduced order) models have to be developed for each block. Then, two linear systems have to be solved at each coupling iteration. Even if these systems have a size that is directly proportional to the number of points at the fluid-structure interface, which is always much smaller than the total number of points of the fluid and solid domains, they are usually dense [67, 72] and are thus an inevitable source of complexity and computing cost. One can easily get around the first drawback by rewriting the same fixed-point equation (3.6) under a root-finding equation for the interface only:

$$\mathcal{D}_{\mathrm{s}} \circ \mathcal{D}_{\mathrm{f}}(\boldsymbol{d}^{\Gamma}) - \boldsymbol{d} = \boldsymbol{r}(\boldsymbol{d}^{\Gamma}) = \boldsymbol{0}\,. \tag{3.31}$$

Unlike the root-finding system (3.26), this equation only involves the interface displacements so that an iterative Newton procedure can be developed for the update $\Delta\boldsymbol{d}$:

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{d}}\right]_k \Delta\boldsymbol{d}_k = -\boldsymbol{r}_k\,, \tag{3.32}$$

with

$$\boldsymbol{d}_{k+1} = \boldsymbol{d}_k + \Delta\boldsymbol{d}_k\,. \tag{3.33}$$

Nevertheless, a linear system has still to be solved at each coupling iteration. The Jacobian matrix $\partial\boldsymbol{r}/\partial\boldsymbol{d}$ has to be known explicitly if a direct solver is used for this linear system or it has to be possible to calculate the product of the Jacobian matrix with a vector if this linear system is solved iteratively [73]. As already mentioned, an explicit calculation of the Jacobian is not feasible for a black-box partitioned coupling since it would require the access to the Jacobian of each coupled solver. On the other hand, the matrix-vector product can be approximated using finite-differences but this requires an evaluation of the residual operator at every iteration of the iterative solver [73]. Following the same methodology as the IBQN-LS approach, Degroote *et al.* [70, 73] suggested a technique to approximate the Jacobian of a function based on inputs and outputs of that function. They show that, with a special choice of the inputs and outputs, an approximation of the inverse of the Jacobian $\partial\boldsymbol{r}/\boldsymbol{d}$ can be obtained. The second drawback of IBQN-LS is thus also removed since no linear system has to be solved anymore for the Newton procedure. The approximation of the Jacobian (for quasi-Newton approaches) is motivated by the fact that only certain components of the error on the interface position become unstable or are badly damped during Gauss–Seidel iterations [67, 74]. Consequently, the approximate Jacobian only has to describe the reaction to those unstable or badly damped components, the other components being damped anyhow. The full Jacobian is thus not required for fast convergence of the coupled problem and an approximation can be used instead [70].

The approximation of the Jacobian is constructed as follows [73]. During each coupling iteration, the two vectors

$$\begin{aligned}\Delta\boldsymbol{r}_l &= \boldsymbol{r}_l - \boldsymbol{r}_j\,,\\ \Delta\tilde{\boldsymbol{d}}_l &= \tilde{\boldsymbol{d}}_l - \tilde{\boldsymbol{d}}_j\,,\end{aligned} \tag{3.34}$$

with $l = 0, \ldots, j-1$, are built based on the previous values (subscript l) and the last value (subscript j). These vectors are accumulated and stored as columns of two matrices

$$\begin{aligned} \mathbf{V}_j &= \left[\Delta \boldsymbol{r}_{j-1} \quad \Delta \boldsymbol{r}_{j-2} \quad \ldots \quad \Delta \boldsymbol{r}_1 \quad \Delta \boldsymbol{r}_0\right] , \\ \mathbf{W}_j &= \left[\Delta \tilde{\boldsymbol{d}}_{j-1} \quad \Delta \tilde{\boldsymbol{d}}_{j-2} \quad \ldots \quad \Delta \tilde{\boldsymbol{d}}_1 \quad \Delta \tilde{\boldsymbol{d}}_0\right] , \end{aligned} \tag{3.35}$$

of size $n_s \times q$ where n_s is the number of points at the solid interface and q is related to the number of coupling iterations at the current time step. Usually, in practice, it is found that $q \ll n_s$. If the number of coupling iterations exceeded the number of points at the solid interface, the number of columns q could be limited to n_s by simply discarding the rightmost columns. At convergence the residual $\boldsymbol{r}$ should by definition vanish. The current deviation from this target value, $\Delta \boldsymbol{r} = \mathbf{0} - \boldsymbol{r}_j$, is then expressed as a linear combination of all known $\Delta \boldsymbol{r}_l$,

$$\Delta \boldsymbol{r} \approx \sum_{l=0}^{j-1} a_l \, \Delta \boldsymbol{r}_l = \mathbf{V}_j \boldsymbol{a}_j , \tag{3.36}$$

where the vector $\boldsymbol{a}$ of unknown coefficients is a column vector of size q. This problem of determining $\boldsymbol{a}_j$ is typically over-determined since $q \ll n_s$ and must therefore be solved in a least-square sense. A QR-decomposition of $\mathbf{V}$ is performed

$$\mathbf{V}_j = \mathbf{Q}_j \, \mathbf{R}_j , \tag{3.37}$$

where $\mathbf{Q}$ is an orthogonal matrix of size $n_s \times q$ and $\mathbf{R}$ is an upper triangular matrix of size $q \times q$. The coefficient vector is then obtained by solving the relatively small triangular system

$$\mathbf{R}_j \, \boldsymbol{a}_j = \mathbf{Q}_j^{\mathrm{T}} \, \Delta \boldsymbol{r}_j \tag{3.38}$$

through back substitution. Although a linear system must also be solved here, the cost associated with the solution of a triangular system of size $q << n_s$ is mush smaller than that of solving a dense system of size n_s. Finally, the corresponding vector $\Delta \tilde{\boldsymbol{d}}$ is written using the same linear combination as Eq. (3.36),

$$\Delta \tilde{\boldsymbol{d}} \approx \sum_{l=0}^{j-1} a_l \, \Delta \boldsymbol{d}_l = \mathbf{W}_j \boldsymbol{a}_j , \tag{3.39}$$

and, since $\Delta \boldsymbol{r} = \Delta \tilde{\boldsymbol{d}} - \Delta \boldsymbol{d}$ by definition of the residual, one has

$$\Delta \boldsymbol{d} = \mathbf{W}_j \, \boldsymbol{a}_j - \Delta \boldsymbol{r} = \mathbf{W}_j \, \boldsymbol{a}_j + \boldsymbol{r}_j \, . \tag{3.40}$$

The approximate inverse of the Jacobian is consequently never explicitly calculated and must not be stored. Instead the product between the inverse Jacobian and a vector is approximated in a matrix-free form such that the update $\Delta \boldsymbol{d}_j$ is given by

$$\Delta \boldsymbol{d}_j = \mathbf{W}_j \, \boldsymbol{a}_j + \boldsymbol{r}_j \, . \tag{3.41}$$

The resulting coupling procedure is referred to as interface quasi-Newton with approximation of the inverse Jacobian from least-square (IQN-ILS). Similarly to IBQN-LS, the Newton procedure cannot be applied at the first iteration, so that a static relaxation is used. The algorithm is illustrated in Alg. 5 for one time step.

Algorithm 5 IQN-ILS algorithm as described by Degroote *et al.* [73]

```
 1: Enter time step with predicted value $\boldsymbol{d}_{\text{pred},0}$
 2: Call fluid solver to compute $\tilde{\boldsymbol{t}}_0 = \mathcal{D}_\text{f}(\boldsymbol{d}_{\text{pred},0})$
 3: Call solid solver to compute $\tilde{\boldsymbol{d}}_0 = \mathcal{D}_\text{s}(\tilde{\boldsymbol{t}}_0)$
 4: Compute residual $\boldsymbol{r}_0 = \tilde{\boldsymbol{d}}_0 - \boldsymbol{d}_{\text{pred},0}$
 5: Perform static relaxation $\boldsymbol{d}_0 = \boldsymbol{d}_{\text{pred},0} + \omega\,\boldsymbol{r}_0$
 6: $j = 1$
 7: while $||\boldsymbol{r}_{j-1}|| > \varepsilon$ do
 8:     Call fluid solver $\tilde{\boldsymbol{t}}_j = \mathcal{D}_\text{f}(\boldsymbol{d}_{j-1})$
 9:     Call solid solver $\tilde{\boldsymbol{d}}_j = \mathcal{D}_\text{s}(\tilde{\boldsymbol{t}}_j)$
10:     Compute residual $\boldsymbol{r}_j = \tilde{\boldsymbol{d}}_j - \boldsymbol{d}_{j-1}$
11:     Update $\mathbf{V}_j = \begin{bmatrix}\Delta\boldsymbol{r}_{j-1} & \dots & \Delta\boldsymbol{r}_0\end{bmatrix}$ with $\Delta\boldsymbol{r}_l = \boldsymbol{r}_l - \boldsymbol{r}_j$
12:     Update $\mathbf{W}_j = \begin{bmatrix}\Delta\tilde{\boldsymbol{d}}_{j-1} & \dots & \Delta\tilde{\boldsymbol{d}}_0\end{bmatrix}$ with $\Delta\tilde{\boldsymbol{d}}_l = \tilde{\boldsymbol{d}}_l - \tilde{\boldsymbol{d}}_j$
13:     Perform QR-decomposition $\mathbf{V}_j = \mathbf{Q}_j\,\mathbf{R}_j$
14:     Solve triangular system $\mathbf{R}_j\,\boldsymbol{a}_j = \mathbf{Q}_j^\text{T}\,(-\boldsymbol{r}_j)$
15:     Compute the update $\Delta\boldsymbol{d}_j = \mathbf{W}_j\,\boldsymbol{a}_j + \boldsymbol{r}_j$
16:     Update the solution $\boldsymbol{d}_j = \boldsymbol{d}_{j-1} + \Delta\boldsymbol{d}_j$
17:     $j = j + 1$
18: end while
19: Predict the value of the displacement and go to next time step
```

Remark

If a residual vector $\Delta\boldsymbol{r}_l$ is identical to another one or to a linear combination of other residual vectors, one of the diagonal elements of $\mathbf{R}_j$ will be zero. Consequently, the equation related to that row of $\mathbf{R}_j$ cannot be solved during the back substitution and the corresponding element of $\boldsymbol{a}_j$ is set to zero [73].

Taking advantage of the low computational cost associated with the quasi-Newton method, it is possible to further improve the convergence of the procedure by including the contribution of previous time steps. This is performed by combining the matrices $\mathbf{V}_j$ and $\mathbf{W}_j$ with those coming from the r previous time steps, as suggested by Degroote *et al.* [73],

$$\begin{aligned}\mathbf{V}'_j &= \begin{bmatrix}\mathbf{V}_j^n & \mathbf{V}_j^{n-1} & \dots & \mathbf{V}_j^{n-r}\end{bmatrix}\,, \\ \mathbf{W}'_j &= \begin{bmatrix}\mathbf{W}_j^n & \mathbf{W}_j^{n-1} & \dots & \mathbf{W}_j^{n-r}\end{bmatrix}\,,\end{aligned} \tag{3.42}$$

where the current time step is denoted by superscript n. When information from the history of the computation is reused, the relaxation step at the beginning of each time step becomes unnecessary and the Newton procedure can directly be applied; only the first time step of the computation requires a preliminary relaxation.

The IQN-ILS method can be seen as a reduction of the IBQN-LS in the sense that a block description of the root-finding problem is reduced to a single formulation. In IBQN-LS, the same matrices $\mathbf{V}$ and $\mathbf{W}$ are constructed to accumulate information along the coupling iterations, but the block procedure requires models for both the fluid and the solid part and thus four matrices, $(\mathbf{V},\,\mathbf{W})_\text{f}$ and $(\mathbf{V},\,\mathbf{W})_\text{s}$, have to be built. The use of information from previous time steps when building the matrices is also possible. Degroote *et al.* [70] compare the performance (mainly the mean number of coupling iterations per time step) of IBQN-LS and IQN-ILS with the performance of the standard Aitken relaxation. They show that IBQN-LS performs very similarly to IQN-ILS, and that both perform much better than Aitken by typically reducing the number of iterations by a factor 4 for cases with poor stability. It is interesting to note that the Aitken relaxation could also be seen as an IQN technique. If the inverse Jacobian in Eq. (3.32) is approximated by $-\omega\,\mathbf{I}$, the Aitken formulation is recovered. This clearly illustrates the fact that the central improvement of I(B)QN-(I)LS is the possibility to store information from previous iterations, and potentially time steps, in order to enhance the convergence. However, when using information from previous time steps, it is not a priori clear how many past instances should be retained, i.e. how long old data is representative of the problem at the current time level [75]. Moreover, as the number r of retained time steps increases, the probability of columns in the matrices being nearly linearly dependent also increases. This may cause numerical issues in the computation of the (inverse) Jacobian so that convergence not only stalls, but the iterative procedure might even diverge. Haelterman *et al.* [75] have proposed filtering techniques that identify and remove columns that are (nearly) linearly dependent, thus further improving the performance of IQN-ILS. Another method was proposed by Bogaers *et al.* [72] as the multi-vector interface quasi-Newton technique (MV-IQN). The main difference with respect to IQN-ILS is that MV-IQN is based on an iteratively updated approximation of the inverse Jacobian that removes the need to explicitly define user parameters to control the amount of computation history that is retained. The methodology is still based on information accumulation through $\mathbf{V}$ and $\mathbf{W}$ matrices for a current time step but the time history is automatically taken into account by updating the inverse Jacobian from the last time iteration only with the current $\mathbf{V}$ and $\mathbf{W}$, i.e. without combining with those from the past. However the principal limitation of this method is that the full approximated inverse Jacobian has to be stored in order to apply the iteratively updated formulation [72, 76].

3.3 Thermal coupling schemes

In general, thermally coupled problems can be solved with the exact same approach as mechanically coupled problems. By analogy to Eq. (3.2), the thermally coupled problem can be written in the general form

$$\begin{cases} \mathcal{F}\left(\boldsymbol{u}, \boldsymbol{w}\right) = \mathbf{0}\,, \\ \mathcal{S}\left(\boldsymbol{u}, \boldsymbol{w}\right) = \mathbf{0}\,, \end{cases} \tag{3.43}$$

where the mesh operator is not required here because there is no fluid mesh motion. In this system, $\boldsymbol{w}$ represents the structural temperature field. The problem can be expressed as a fixed-point problem similar to Eq. (3.6) for thermal interface quantities, but instead of exchanging structural displacements and fluid loads at the interface, the coupled procedure exchanges heat fluxes, through Neumann boundary conditions, and temperatures, through Dirichlet boundary conditions. In mechanical coupling, a Dirichlet-Neumann (DN) coupling was introduced: the displacement of the interface is imposed to the fluid solver (Dirichlet BC) and, in return, the interface fluid loads are imposed to the solid solver (Neumann BC). The DN procedure is, from both the physical and numerical points of view the most intuitive way to exchange interface mechanical data. Neumann-Dirichlet (ND) or even Neumann-Neumann are rarely used and have enjoyed little success in terms of coupling stability, especially for high added-mass cases [77]. In the case of thermal coupling, both DN and ND coupling seem to be intuitively acceptable and have been successfully applied to numerous CHT applications [78]. In thermal coupling, DN and ND schemes are usually referred to as Flux Forward Temperature Back (FFTB) and Temperature Forward Flux Back (TFFB), respectively [79–81]. These two schemes are depicted in Fig. 3.7. In FFTB, the fluid interface heat flux is transferred to the solid as a Neumann BC, and the solid interface temperature is transferred to the fluid as a Dirichlet BC. Conversely, in TFFB the Dirichlet fluid temperature is applied to the solid and the Neumann solid heat flux is applied to the fluid, as discussed below. The choice for a particular scheme is mainly dictated by the stability of the coupling procedure.

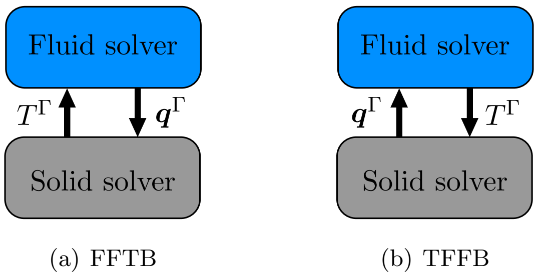


Figure 3.7: Standard thermal coupling schemes.

3.3.1 Stability of the thermal coupling schemes

A pioneering study on the stability of CHT coupling was carried out by Giles [82]. He applied the stability theory of Godunov and Ryabenkii [83] to a discretized simplified 1D model. Several simplifications were made, such as a uniform grid on both sides of the interface and the omission of the convection terms in the fluid domain. Giles concluded that numerical stability is achieved by using Neuman boundary conditions (heat flux) for the structural calculation and Dirichlet boundary conditions (temperature) for the fluid calculations, i.e. a FFTB scheme. However, this might not irrevocably correspond to the stability behavior found in practice on more complex CHT cases [80, 81]. Verstraete *et al.* [81] introduced and developed stability criteria based on physical considerations, similarly to the added-mass effect based on the mass ratio for the mechanical coupling. Again a simplified 1D CHT problem was considered for the stability analysis: a steady-state boundary layer flow at temperature T_f^∞ over a bottom-heated flat plate at temperature T_s, as illustrated in Fig. 3.8. At this stage, it is interesting to note that

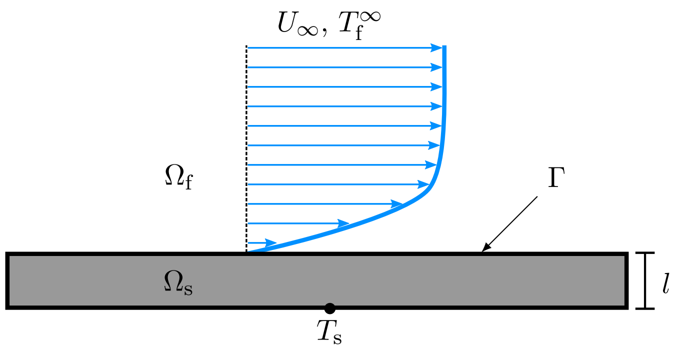


Figure 3.8: Flat plate flow for 1D CHT problem. Analysis is performed in the normal direction to the plate.

while the unstable behavior of the mechanical coupling is related to inertia involving dynamic (time-dependent) quantities, the instabilities arising in the thermal coupling can appear even for steady-state coupling. Under these considerations, Verstraete *et al.* [81] demonstrated that the FFTB scheme is unconditionally stable if and only if the Biot number Bi < 1. On the contrary, the TFFB scheme is unconditionally stable if and only if Bi > 1. This shows that the stability of the thermal coupling is directly related to the physics of the thermal exchange between the fluid and the solid, through the Biot number,

$$\mathrm{Bi} = \frac{h\,l}{\lambda_\mathrm{s}}\,. \tag{3.44}$$

For low Biot number (Bi $\ll 1$), the temperature at the fluid-solid interface T^Γ will be close to T_s and the convergence of the FFTB scheme will be as fast as Bi is low. For high Biot number

(Bi $\gg$ 1), T^{Γ} will be close to the fluid temperature and this time the TFFB will converge as fast as Bi is high.

Convergence with the FFTB or TFFB schemes can still be achieved outside of their respective stability limits by using a relaxation procedure such as Aitken's relaxation that was introduced with, but was not restricted to, mechanical coupling in the previous section. However, for the thermal coupling, another stabilization method is usually used that is based on Newton's law of cooling at the fluid-structure interface Γ,

$$\boldsymbol{q}^{\Gamma} \cdot \boldsymbol{n} = h\left(T^{\Gamma} - T_{\mathrm{f}}^{\infty}\right) , \tag{3.45}$$

where $\boldsymbol{q}$ is the heat flux and h a convective heat transfer coefficient that depends on the flow conditions. At the discrete level of the coupled problem, a convective heat transfer relation can be imposed through a Robin type boundary condition that combines heat flux and temperature:

$$\boldsymbol{q}^{\Gamma} \cdot \boldsymbol{n} - hT^{\Gamma} = cst\,. \tag{3.46}$$

In practice, the condition imposed to the solid side (i.e. the solid solver input at each coupling step) is given by

$$\boldsymbol{q}^{\Gamma} \cdot \boldsymbol{n} = \tilde{h}\left(T^{\Gamma} - \hat{T}\right) , \tag{3.47}$$

where $\tilde{h}$ is a numerical heat transfer coefficient and $\hat{T}$ is an equivalent temperature. At each coupling step, these two parameters need to be calculated on one side of the interface, typically the fluid side, and then are transferred to the solid side in order to compute the resulting heat flux. The most intuitive way to define $\tilde{h}$ and $\hat{T}$ is by using the fluid results in the first cells normal to the interface. However, this again requires deeper access to data in the fluid solver which is not always possible and therefore leads to a loss in the general applicability of the approach. A simpler and more stable method imposes a constant positive value of $\tilde{h}$. Although the heat flux is not directly exchanged at the interface, the method results in a continuous temperature and heat flux field across the domains. The value of $\tilde{h}$ only influences the convergence rate and does not affect the final result [81]. The Robin boundary condition in Eq. (3.47) is thus used not only to stabilize the coupling procedure but also to increase the rate of convergence. The equivalent temperature is then calculated by expressing Eq. (3.47) on the fluid side of the interface, where the fluid interface heat flux is the direct output of the fluid solver:

$$\hat{T} = T_{\mathrm{f}}^{\Gamma} - \frac{\boldsymbol{q}_{\mathrm{f}}^{\Gamma} \cdot \boldsymbol{n}}{\tilde{h}}\,. \tag{3.48}$$

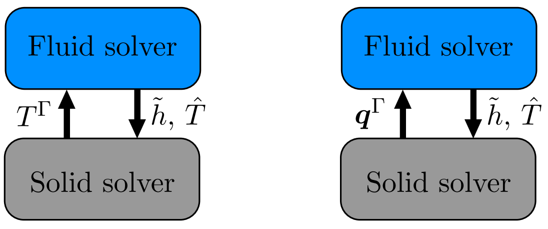
The same relation expressed on the structural side of the interface will give a new heat flux as an input to the solid solver:

$$\boldsymbol{q}_{\mathrm{s}}^{\Gamma} \cdot \boldsymbol{n} = \tilde{h}\left(T_{\mathrm{s}}^{\Gamma} - \hat{T}\right)\,. \tag{3.49}$$

The resulting new coupling scheme is either called Heat Transfer Coefficient Forward Flux Back (hFTB) or Heat Transfer Coefficient Forward Temperature Back (hFTB) depending on the quantity returned by the solid solver and given as input to the fluid solver. These two schemes are illustrated in Fig. 3.9. The stabilizing effect of the Robin transmission (fluid to solid) by the use of a numerical heat transfer coefficient comes from the fact that it introduces a certain amount of interface stiffness that forces the boundary condition on one domain to behave in the same way as the boundary of the opposite domain [84, 85].

Similarly to any kind of relaxation parameter, the value for the numerical heat transfer coefficient that leads to stable and fast convergence can be established by a stability analysis. For the hFTB scheme, it can be demonstrated that convergence is ensured if the amplification factor $\tilde{G}$ satisfies the following condition [81, 86]:

$$\left|\tilde{G}\right| = \left|\frac{\tilde{\mathrm{Bi}} - \mathrm{Bi}}{\tilde{\mathrm{Bi}} + 1}\right| < 1\,, \tag{3.50}$$



(a) hFTB (b) hFFB

Figure 3.9: Thermal coupling schemes with stabilizing Robin boundary condition and numerical heat transfer coefficient.

where $\tilde{\mathrm{Bi}}$ is the numerical Biot number associated with $\tilde{h}$. When $\tilde{\mathrm{Bi}} \geq \mathrm{Bi}$, the condition (3.50) is always satisfied and thus a sufficiently high value of $\tilde{h}$ will always lead to convergence. Besides, the convergence rate will be as fast as $\tilde{\mathrm{Bi}}$ is close to Bi. When $\tilde{\mathrm{Bi}} < \mathrm{Bi}$ the condition (3.50) becomes

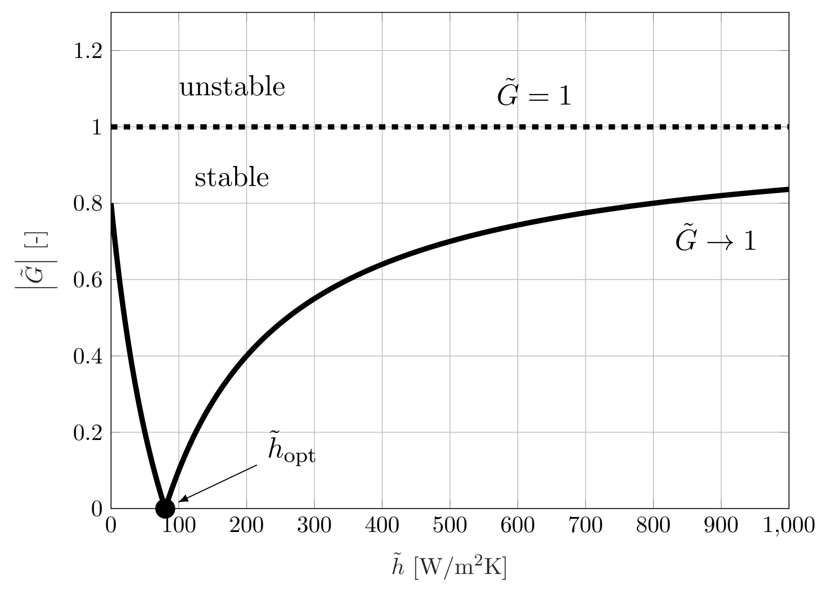
$$\tilde{\mathrm{Bi}} > \frac{\mathrm{Bi}-1}{2} \tag{3.51}$$

which is always satisfied for $\mathrm{Bi} < 1$. Thus, the hFTB method remains unconditionally stable for a physical Biot number below one. The dependence of the amplification factor $|\tilde{G}|$ on $\tilde{h}$ is illustrated in Fig. 3.10 for two different Biot numbers. Two distinct branches can be identified on either side of the optimal value of the numerical heat transfer coefficient $\tilde{h}_{\mathrm{opt}}$, for which $\tilde{G} = 0$ (i.e. convergence is theoretically reached in 0 iteration). The left branch corresponds to cases where $\tilde{\mathrm{Bi}} < \mathrm{Bi}$ and stability is limited by condition (3.51). The left branch of Fig. 3.10(a) shows that this condition is always met when $\mathrm{Bi} < 1$. The right branch is obtained for $\tilde{\mathrm{Bi}} > \mathrm{Bi}$ and is always stable but the rate of convergence decreases as $\tilde{h}$ increases. The stability analysis of the hFTB scheme shows that a stable and optimal (i.e. leading to the highest rate of convergence) numerical heat transfer coefficient exists and is such that $\tilde{\mathrm{Bi}} = \mathrm{Bi}$. Similar conclusions can be found in the works of Errera *et al.* [85,87,88] in which a more detailed and complex stability analysis is performed on a discretized 1D model where a time-marching procedure is used to converge a steady-state solution. Their analysis also shows that the optimal heat transfer coefficient depends on the fluid time step and the fluid mesh discretization normal to the interface. Although they provide a complete local and dynamic expression for the optimal heat transfer coefficient, the approach cannot be readily applied without deep access to the fluid and solid solvers, which goes against the black-box approach followed here. Corral *et al.* [89] have proposed a simplified method to approximate the optimal value for $\tilde{h}$. The optimal condition is obtained when $\tilde{\mathrm{Bi}} = \mathrm{Bi}$, which simply implies $\tilde{h} = h$. Using this relation, the numerical heat transfer coefficient is locally approximated by

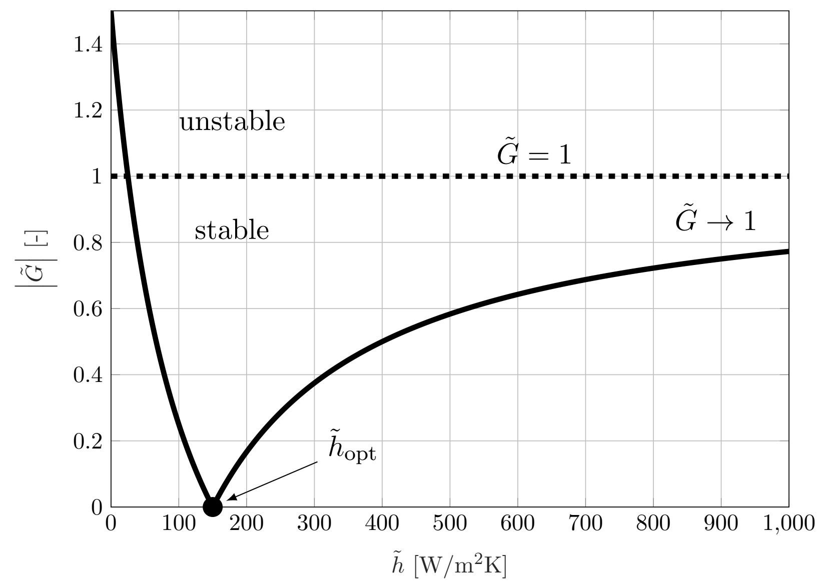
$$\tilde{h}_{\mathrm{opt},l} = h_l = \left(\frac{\partial(\boldsymbol{q}\cdot\boldsymbol{n})^\Gamma}{\partial T^\Gamma}\right)_l = \left|\frac{(\boldsymbol{q}\cdot\boldsymbol{n})^\Gamma_j - (\boldsymbol{q}\cdot\boldsymbol{n})^\Gamma_{j-1}}{T^\Gamma_j - T^\Gamma_{j-1}}\right|_l, \tag{3.52}$$

at each fluid grid point l and between two coupling iteration j. This expression only requires interface data that are already available in the coupling procedure. The use of such a local numerical heat transfer coefficient is advantageous as it takes into account the non-homogeneous distribution of the physical Biot number along the fluid-structure interface and, thus, provides better stability and convergence. A simpler but widely used approach is to consider a constant coupling coefficient over the interface. Because the stability and convergence rate are in this case determined by the worst local conditions, other parts of the interface experience stable but very slow convergence.

The same analysis can be applied to the hFFB method. For this scheme, the amplification factor and the conditions that have to be fulfilled to guarantee stability are given by the following



(a) Bi = 0.8



(b) Bi = 1.5

Figure 3.10: Absolute value of the amplification factor of the hFTB scheme as a function of the numerical convective heat transfer coefficient $\tilde{h}$ for a Biot number smaller (a) and larger (b) than one.

expression ([81,86]):

$$\left|\tilde{G}\right| = \left|\frac{1}{\mathrm{Bi}}\frac{\tilde{\mathrm{Bi}}-\mathrm{Bi}}{\tilde{\mathrm{Bi}}+1}\right| < 1\,. \tag{3.53}$$

When $\tilde{\mathrm{Bi}} \leq \mathrm{Bi}$ this condition is always satisfied and a sufficiently low value of $\tilde{h}$ will always lead to convergence. Again, the convergence rate will be as fast as $\tilde{\mathrm{Bi}}$ is close to Bi (or $\tilde{h}$ is close to h). When $\tilde{\mathrm{Bi}} > \mathrm{Bi}$, the condition (3.53) becomes

$$\tilde{\mathrm{Bi}} < \frac{2\mathrm{Bi}}{1-\mathrm{Bi}}\,. \tag{3.54}$$

Note that for cases where $\mathrm{Bi} > 1$, the hFFB method remains unconditionally stable. Fig. 3.11 illustrates the typical dependence of $|\tilde{G}|$ on $\tilde{h}$ for two different Biot numbers. Again two branches can be identified on either side of the optimal heat transfer coefficient, where the left branch ($\tilde{\mathrm{Bi}} < \mathrm{Bi}$) is this time always stable and the stability of the right branch is limited by condition (3.54) if $\mathrm{Bi} < 1$ (unconditionnally stable otherwise). However, it is important to note that the optimal heat transfer coefficient does not depend on the scheme (hFTB or hFFB) since it corresponds to $\tilde{\mathrm{Bi}} = \mathrm{Bi}$ in both cases. Hence, the relation (3.52) could be used for both thermal coupling schemes.

Tab. 3.1 summarizes the most important trends in terms of stability for the two heat transfer coupling schemes. For each of them, the optimal (stable) rate of convergence is obtained for $\tilde{\mathrm{Bi}} = \mathrm{Bi}$, i.e. $\tilde{h}_{\mathrm{opt}} = h$. It is also worth noting that the FFTB scheme is only a particular case of hFTB for which $\tilde{h} \to 0$. Analogously, the TFFB scheme is the particular case of hFFB when $\tilde{h} \to \infty$. It is also important to keep in mind that these trends have been established for a

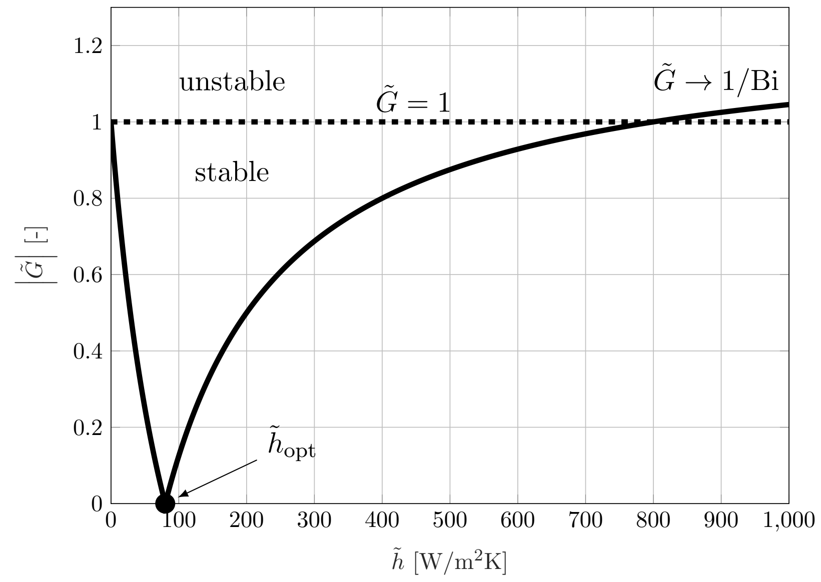
		$\mathrm{Bi} < 1$	$\mathrm{Bi} > 1$
hFTB	$\tilde{\mathrm{Bi}} \geq \mathrm{Bi}$	stable	stable
	$\tilde{\mathrm{Bi}} < \mathrm{Bi}$		stable if $\tilde{\mathrm{Bi}} > \frac{\mathrm{Bi}-1}{2}$
hFFB	$\tilde{\mathrm{Bi}} > \mathrm{Bi}$	stable if $\tilde{\mathrm{Bi}} < \frac{2\mathrm{Bi}}{1-\mathrm{Bi}}$	stable
	$\tilde{\mathrm{Bi}} \leq \mathrm{Bi}$	stable	

Table 3.1: Stability conditions on the numerical heat transfer coefficient (through the numerical Biot number) for different numerical schemes as a function of the physical Biot number. The particular value of $\mathrm{Bi} = 1$ leads to a stable coupling, whichever the scheme is applied.

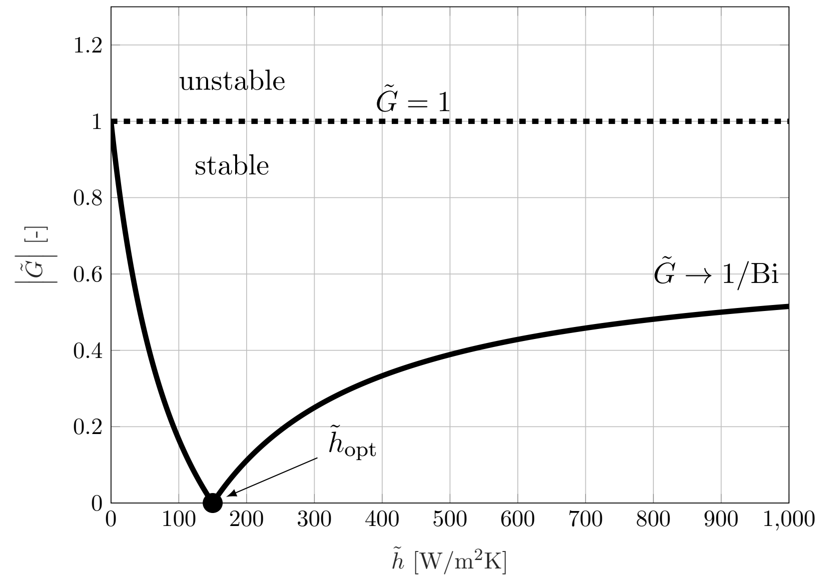
simple analytic 1D case. More details about the effect of the discretization procedure on the stability and the optimal value of $\tilde{h}$ can be found in the work of Errera *et al.* [87]. However, several authors have observed that these trends are also valid and that an optimal numerical heat transfer coefficient exists for more complex 2D/3D cases [89–91]. Furthermore, Gimenez *et al.* [86] have concluded that the steady stability theory based on the Biot number can be extended at least to weakly transient CHT problems using a quasi-dynamic coupling approach, which is described in the next section.

Remark

A similar approach to the numerical heat transfer coefficient method (hFTB and hFFB) has also been proposed for mechanical coupling and consists in using an Interface Artificial Compressibility (IAC). Details about this method can be found in Degroote *et al.* [92].



(a) Bi = 0.8

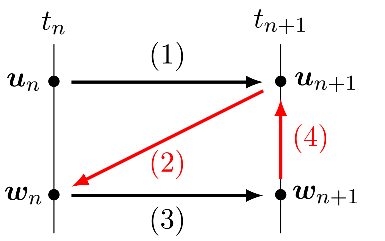


(b) Bi = 1.5

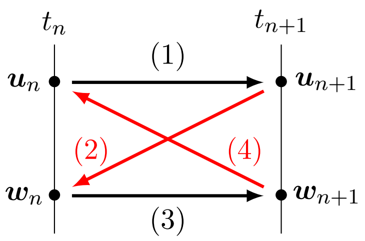
Figure 3.11: Absolute value of the amplification factor of the hFFB scheme as a function of the numerical convective heat transfer coefficient $\tilde{h}$ for a Biot number smaller (a) and larger (b) then one.

3.3.2 Time advancement of the thermal coupling procedure

The time advancement procedure of problems involving thermal coupling can be built in the same way as that developed for mechanical coupling: the two solvers are advanced in physical time with the same time step using either a weakly-coupled or a strongly-coupled approach such as illustrated in Fig. 3.12. The thermal fluid ($\boldsymbol{u}$) and solid ($\boldsymbol{w}$) states are communicated according to the different schemes previously described (TFFB, hFTB, ...). However, the generally significant



(a) Weak coupling, one (1)-(4) sequence per time step.



(b) Strong coupling, repeated (1)-(4) sequences until convergence at current time step.

Figure 3.12: Time step advancement for a loosely- or strongly-coupled scheme for thermal interaction. Black: time advancement steps, red: communication steps.

difference in time scales controling the thermal physics of the fluid and solid domains has led to the development of other strategies. The fluid (convective) time scale can be approximated as

$$\tau_{\mathrm{f}} = \frac{L}{U}\,, \tag{3.55}$$

where L and U are a characteristic length and velocity, respectively, while the solid (conduction) time scale is approximated as

$$\tau_{\mathrm{s}} = \frac{L^2}{\alpha_{\mathrm{s}}}\,, \tag{3.56}$$

where α_s is the solid thermal diffusivity. In engineering CHT problems, for instance in turbomachinery design, the ratio $\tau_{\mathrm{s}}/\tau_{\mathrm{f}}$ is typically very large so that it usually takes a considerable amount of time for the solid to adapt its thermal state to a change in the fluid domain. The thermal penetration depth within the solid in response to a fluid harmonic excitation at frequency f is proportional to $\sqrt{\alpha_{\mathrm{s}}/f}$ [93]. Consequently, the higher the frequency of the thermal perturbation caused by the fluid, the lower the impact on the thermal state of the solid. In these conditions, a fully synchronized coupling scheme may lead to an unnecessarily high computing cost due to the long integration time required. The absence of inner iterations in the loosely-coupled approach already leads to a reduction in computation time when compared to a strongly-coupled approach. However, the two solvers are still synchronized in physical time and they exchange their interface data at each time step $\Delta t = \Delta t_{\mathrm{f}} = \Delta t_{\mathrm{s}}$. Better efficiency can be achieved by considering the difference in physical time scales, as illustrated in Fig. 3.13, and by reducing the frequency of exchange between the two solvers so that the interface data are exchanged only at every coupling time step Δt_c with $\Delta t_c = N\Delta t$ [91, 94, 95]. Although this contributes to lowering the time associated to the interface data treatment, a sufficiently high coupling frequency is still necessary to guarantee an acceptable accuracy of the coupled solution [93] as well as a stable procedure [91]. Finally, the difference in the time scales may lead to different stability requirements in the fluid and solid domains. Therefore, different time step sizes can even be used for each coupled solver to advance its own solution between two coupling steps. Usually the time step size in the solid domain will be larger than the time step size in the fluid ($\Delta t_{\mathrm{s}} > \Delta t_{\mathrm{f}}$), but since the solvers are kept synchronized in time, the coupling time step

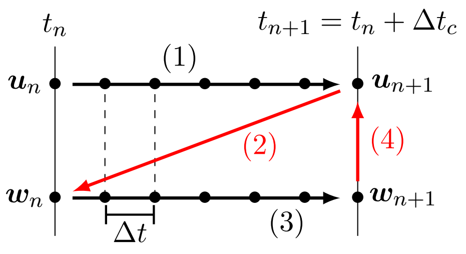


Figure 3.13: Time step advancement of a very loose coupled scheme for CHT, the data exchange is performed after several physical time steps. Black: time advancement steps, red: communication steps.

must satisfy $\Delta t_c = N_\mathrm{f} \Delta t_\mathrm{f} = N_\mathrm{s} \Delta t_\mathrm{s}$, where N_f and N_s are integers. This coupling procedure is depicted in Fig. 3.14.

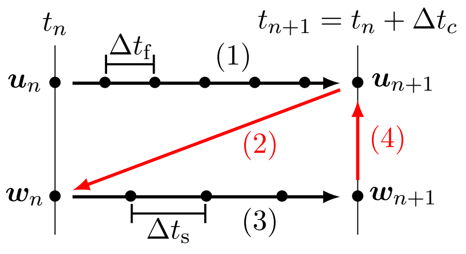


Figure 3.14: Time step advancement of a very loose coupled scheme for CHT, the data exchange is performed after several physical time steps which are not the same between the fluid and solid domains. Black: time advancement steps, red: communication steps.

For weakly transient CHT coupling, if the influence of unsteadiness in the fluid domain remains negligible, the flow field may be considered as a sequence of steady-states. In this case, it seems legitimate to couple steady fluid calculations with relatively large-time-step transient solid calculations at a coupling period Δt_c that corresponds to several solid time steps Δt_s [85,86,96]. This approach, referred to as quasi-dynamic, can be applied either with a loose coupling, where the solvers exchange their interface data after a certain number of solid time steps, or strong coupling, where the same transient solid calculation is repeated between two exchanges until convergence, as illustrated in Fig. 3.15. This last choice usually depends on the required level of accuracy on the time evolution of the thermal state. Note that for the strongly coupled approach, the boundary conditions applied on the solid interface at intermediate steps can be interpolated between two known fluid computations [85]. Unlike the fully transient coupling, the quasi-dynamic procedure reduces the time associated with the computation of the fluid flow, since only a steady-state solution is required instead of a full time-dependent response with potentially low time step size. The steady flow solution is usually obtained by solving the Navier-Stokes equations with a time-marching procedure involving a numerical, i.e. non physical, time step.

Finally, a coupled steady-state solution is also often sought in practice, where the Laplace or Poisson equation (i.e. Eq. (2.24) without the time derivative term) is solved in the solid domain. A steady flow solution is still usually obtained by a time-marching procedure [81, 89, 90]. As any physical time-dependence is removed, physical time scales have no effect on the steady-state coupling procedure and this approach is obviously the fastest way to obtain a coupled stationary solution.

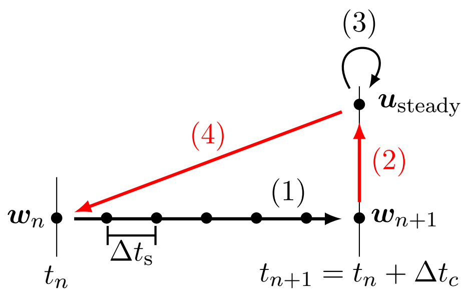


Figure 3.15: Time step advancement of the quasi-dynamic coupling scheme for CHT. Black: time advancement steps, red: communication steps.

3.4 Treatment of the fluid-structure interface

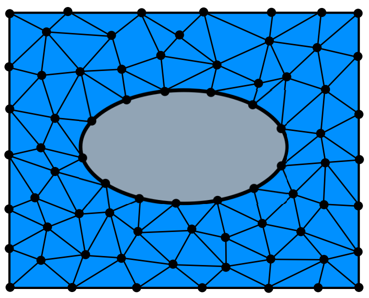
The solid-fluid interface is critical in FSI problems as the coupling between the two physics takes place across it. As such, the communication of interface data between the solvers is one of the key steps in all coupling algorithms, as described above. By definition the interface belongs simultaneously to both domains. Because in a partitioned coupling approach each domain is treated with its own numerical approach, the numerical description of the interface might differ on the solid and fluid side. These differences are directly related to the respective domain meshes.

This section focuses thus on the characterization of the interface from a numerical point of view. It first discusses the two main approaches in mesh treatment, differentiating between conforming and non-conforming meshes. Then, in the context of conforming meshes that are used in the present work, it describes several interpolation techniques that are typically used when the interface nodes on the fluid and solid sides do not coincide.

3.4.1 Conforming vs non-conforming meshes

FSI procedures can be divided into two main approaches depending on whether conforming or non-conforming meshes are used [49], as depicted in Fig. 3.16. In conforming mesh methods, the meshes conform with the interface boundary Γ, as illustrated in Fig. 3.16(a), with no overlap between fluid and solid meshes. Boundary and coupling conditions are thus explicitly enforced as physical conditions. Because the interface generally moves and/or deforms as part of the solution, the domain mesh itself must also adapt dynamically to the interface changes. This is typically achieved on the fluid side through a mesh deformation algorithm in conjunction with an ALE approach. The main drawbacks of the conforming approach are the cost overhead brought by the dynamic fluid mesh treatment and the difficulty of keeping a high mesh quality when deformations are large. When the deformation algorithm cannot generate a valid mesh, remeshing (full or partial) must be applied, thus further increasing the complexity and the cost of the procedure.

On the other hand, the non-conforming mesh method treats the solid boundary and the interface as constraints imposed on the governing equations so that very distinct meshes that do not conform at the interface can be used. Non-conforming mesh methods are commonly developed with fluid solvers based on the immersed boundary (IB) method. A detailed description of this method is beyond the scope of this thesis but interested readers may find many references on this subject, e.g. [40, 49, 97–102]. The principal advantage of the IB method is its simplified fluid mesh treatment. The fluid dynamics is solved using a pure Eulerian formalism (no need for ALE in this case), on a Cartesian mesh in which the structural mesh is embedded, as illustrated in Fig. 3.16(b), while the structural motion is still tracked in a Lagrangian way. Due to the non-conformity of the fluid mesh, boundary conditions such as those required to enforce


(a) Conforming mesh method.

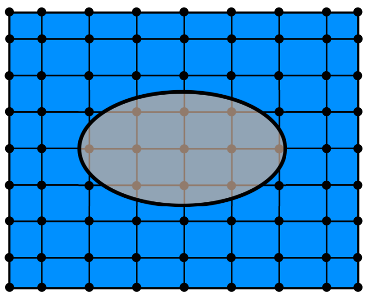

(b) Non-conforming mesh method.

Figure 3.16: Mesh treatment methods for the characterization of the fluid-structure interface.

interface continuity cannot be imposed directly on Γ but must be imposed indirectly. Several methods have been developed in order to link the fluid Eulerian to the solid Lagrangian variables at the interface [99, 103, 104]. The continuous or discrete forcing approach spreads the influence of the immersed solid boundary through forcing terms in the Navier-Stokes equations (2.1). The spreading is performed by means of a smooth distribution function that leads to a diffuse representation of the boundary. Other discrete approaches retain the immersed boundary as sharp interface with no diffuse spreading. The computational stencils are locally modified in the vicinity of the immersed boundary by ghost-cell or cut-cell approaches in order to enforce the boundary conditions.

The main advantage of IB methods is the simplified Cartesian mesh generation procedure, in contrast to the complex unstructured grids usually needed to accommodate complex 3D geometries. The use of Cartesian meshes is also particularly well suited for acceleration techniques such as geometric multi-grids or for parallel partitioning. However another significant drawback, in addition to the complex enforcement of the interface boundary conditions, is the difficulty to locally control mesh refinement in the vicinity of solid boundaries, especially in boundary layer regions for large Reynolds number flows. In particular, a Reynolds number increase leads to a much faster increase of the mesh size (number of points) in the case of IB methods when compared to standard conforming methods. Nonetheless, fluid IB methods have recently gained popularity for FSI [40, 49, 101, 102] due to their ability to treat very large solid motions without the need to deform or regenerate the computational mesh. Consequently, there is no additional cost associated with a mesh dynamics component in the coupled problem, and the severe mesh distorsion that may arise after deformation of body-conforming meshes are circumvented.

In this thesis, the conforming mesh approach is retained. As the interface conditions are naturally enforced as boundary conditions, data communication between the coupled solvers is much easier. However, an overview of immersed boundary methods has been provided for the sake of completeness. Moreover, the partitioned coupling algorithms in section 3.2 do not actually depend on the choice of the interface treatment approach, except in that the mesh dynamic solver is not required anymore for IB (i.e. the operator $\mathcal{M}$ and the fluid mesh displacement $\boldsymbol{z}$ can be discarded from Eq. (3.2)). This means that a coupling architecture based on black-box solvers could be used with either an ALE solver or an IB solver without any loss of generality. This actually highlights one of the significant advantages of the partitioned coupling approach.

3.4.2 Interface mesh interpolation

In a partitioned coupling approach, no matter the interface mesh treatment method chosen, fluid and solid meshes are likely to be generated independently of each other by distinct engineering teams [105]. This way each mesh can be optimized for the respective physics to model and corresponding constraints. More specifically, mesh resolution, element types and topology can widely differ between the different domains [106]. Consequently, the meshes at the interface are usually non-matching, i.e., the nodes of the solid and fluid meshes on the interface boundary do not coincide, as illustrated in Fig. 3.17. An interpolation of the data transferred from the other domain is thus required. For non-conforming mesh methods, particularly IB methods, the interpolation is implicit to the interface capturing procedure. However, for conforming mesh methods, an explicit interpolation procedure has to be added to the coupling algorithm.

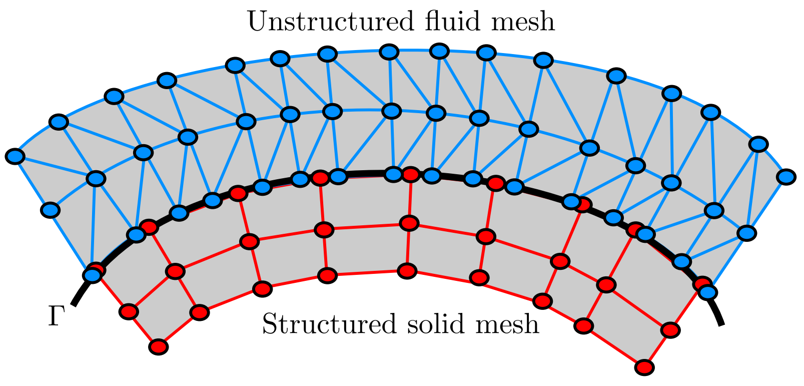


Figure 3.17: Illustration of a non-matching mesh discretization at the fluid-structure interface.

Mathematically, the solid-to-fluid mesh interpolation procedure can be represented by an operator $\mathcal{I}_{\mathrm{s}}^{\mathrm{f}}$ that maps the displacement of the solid interface onto the fluid interface:

$$\boldsymbol{d}_{\mathrm{f}}^{\Gamma} = \mathcal{I}_{\mathrm{s}}^{\mathrm{f}}(\boldsymbol{d}_{\mathrm{s}}^{\Gamma})\,. \tag{3.57}$$

Analogously, the fluid-to-solid mesh interpolation procedure can be represented by an operator $\mathcal{I}_{\mathrm{f}}^{\mathrm{s}}$ that maps the fluid interface loads onto the solid interface:

$$\boldsymbol{t}_{\mathrm{s}}^{\Gamma} = \mathcal{I}_{\mathrm{f}}^{\mathrm{s}}(\boldsymbol{t}_{\mathrm{f}}^{\Gamma})\,. \tag{3.58}$$

The same operators can be defined for the interpolation of temperature and heat flux in case of thermal coupling. These interpolation operators can be considered as the discrete version of the coupling conditions, Eq. (2.27) (mechanical) and Eq. (2.30) (thermal). Note that for the sake of clarity and conciseness, the coupling algorithms described in section 3.2 were developed assuming perfect equality in the coupling conditions at the discrete level, i.e., assuming perfectly matching fluid and solid meshes at the interface. For a non-matching interface, the operators (3.57) and (3.58) can simply be added when solid-fluid and fluid-solid communications, respectively, have to be performed, i.e. typically between two solver calls.

The central aspect of mesh interpolation is then the numerical description of the interpolation operator. Many methods have been developed to map non-matching interface nodes for the transfer of structural displacement to the fluid and fluid loads to the solid. In the following, the direct pairing method introduced by Farhat *et al.* [61] is first described. Then, a more generic mapping method involving an interpolation matrix is presented, where several approaches to compute this matrix, such as the finite element interpolation, the Mortar method or the use of radial basis functions, are presented.

Direct pairing method

In the context of a finite element discretization for the solid problem, Farhat *et al.* [61] developed a methodology based on a direct pairing between interface fluid cells and Gauss points of interface structural elements, as illustrated in Fig. 3.18. Every Gauss point is directly associated with a fluid cell, in which pressure and shear stress are computed by the fluid discretization method, e.g. finite volume, so that structural nodal loads can be computed by finite element quadrature rules using the appropriate solid shape functions. However, this approach suffers from two

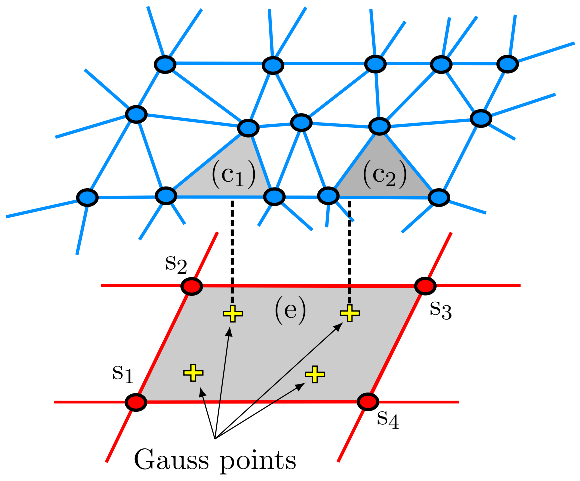


Figure 3.18: Pairing between fluid cells (c_l) and Gauss points of structural elements (e) for the evaluation of fluid loads at nodes s_m.

significant disadvantages. First, it does not guarantee energy conservation at the interface. Energy conservation implies that the energy released or absorbed by the structure is equal to the energy gained or released by the fluid. Energy conservation thus requires that the sum of the discrete fluid loads,

$$\boldsymbol{t}_{\mathrm{f}}^{\mathrm{tot}} = \sum_{i=1}^{n_{\mathrm{f}}} \boldsymbol{t}_{\mathrm{f}}^{i}, \tag{3.59}$$

n_{f} being the number of grid points on Γ_{f}, is equal to the sum of the discrete solid loads $\boldsymbol{t}_{\mathrm{s}}^{\mathrm{tot}}$ on Γ_{s}.

Remark

Although Farhat *et al.* [61] have demonstrated that a strictly speaking non-conservative mapping method can be reliable and accurate, one can still expect numerical energy production or dissipation to have consequences for particular problems such as aeroelastic systems near the flutter point, where spurious energy transfer could alter the prediction of the flutter threshold.

The second drawback of the direct pairing method arises from more practical considerations. Although the fluid and solid domains have been described as sharing a common geometric support for their interface Γ, this is not always true in practice. For instance, in an aeroelastic analysis (see Fig. 3.19), a wing might be represented by its exact airfoil geometry in the fluid domain so that the loads are correctly predicted, but this same wing could be represented as an equivalent structural box in the solid domain. Consequently, the geometric discrepancies between Γ_{f} and Γ_{s} are rather large and the mapping method may fail to properly represent the aerodynamic loads on the structural model [61].

Conservative and consistent interpolation

Farhat *et al.* [61] have suggested another approach that can ensure the conservation of energy at the interface and the exact equilibrium between the fluid tractions computed on Γ_{f} and the

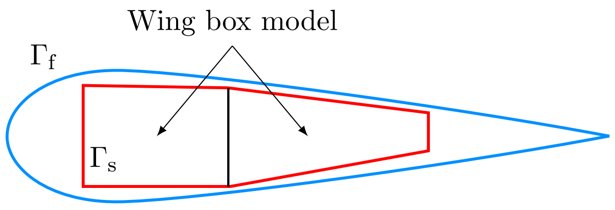


Figure 3.19: Large geometric discrepancies between the fluid interface Γ_f and the solid interface Γ_s due to a simple wing-box geometry description of the solid problem.

solid tractions computed on Γ_s, independently of any discretization issue. The loads on both sides of the interface are computed using the discretization methods and mesh of the same field. For example, if the fluid grid is finer than the structural grid, the forces induced by the fluid on the structure are computed by using the discretization method of the fluid on the geometrical support Γ_f.

The idea consists then in applying the interface continuity conditions on the virtual displacement of the interface. Whichever approximation method is chosen for enforcing compatibility [61, 105–107], this leads to the discrete version

$$\delta \boldsymbol{d}_\mathrm{f}^\Gamma = \mathbf{H}\, \delta \boldsymbol{d}_\mathrm{s}^\Gamma \,, \tag{3.60}$$

where $\delta \boldsymbol{d}^\Gamma$ is the virtual displacement vector on the discrete fluid or solid interface and $\mathbf{H}$ is the mapping matrix that depends on the chosen mapping method (as described below). The discrete virtual work of the fluid loads acting on Γ_f is written as

$$\delta W_\mathrm{f} = (\boldsymbol{t}_\mathrm{f}^\Gamma)^\mathrm{T}\, \delta \boldsymbol{d}_\mathrm{f}^\Gamma \,, \tag{3.61}$$

where $\boldsymbol{t}_\mathrm{f}^\Gamma$ is the discrete fluid loads computed on Γ_f using the fluid discretization. Analogously, the discrete virtual work of the solid acting on Γ_s is given by

$$\delta W_\mathrm{s} = (\boldsymbol{t}_\mathrm{s}^\Gamma)^\mathrm{T}\, \delta \boldsymbol{d}_\mathrm{s}^\Gamma \,, \tag{3.62}$$

and, since energy is conserved if $\delta W_\mathrm{f} = \delta W_\mathrm{s}$ for any arbitrary virtual displacement, it can be concluded that

$$\boldsymbol{t}_\mathrm{s}^\Gamma = \mathbf{H}^\mathrm{T}\, \boldsymbol{t}_\mathrm{f}^\Gamma \,. \tag{3.63}$$

This leads to a very simple condition for energy conservation at the interface: the mapping of the fluid loads on the structure should correspond to the transpose of the mapping matrix used for the displacement. In other words, only a one-way mapping, where the solid mesh is the source mesh and the fluid mesh is the target, is required to generate $\mathbf{H}$ and $\mathbf{H}^\mathrm{T}$. Moreover, Eq. (3.63) shows that the structural loads $\boldsymbol{t}_\mathrm{s}^\Gamma$ do not depend on the discretization method of the structure, but only on the discretization method of the fluid problem and the chosen mapping method used to transfer the displacement.

A general condition for the conservation of the total load across the interface can be derived as follows (for each physical dimension):

$$\begin{aligned}\boldsymbol{t}_{\mathrm{s}}^{\mathrm{tot}} &= \sum_{m=1}^{n_{\mathrm{s}}} \boldsymbol{t}_{\mathrm{s},m}^{\Gamma} \\ &= \sum_{m=1}^{n_{\mathrm{s}}} \sum_{l=1}^{n_{\mathrm{f}}} H_{ml}^{\mathrm{T}} \boldsymbol{t}_{\mathrm{f},l}^{\Gamma} \\ &= \sum_{l=1}^{n_{\mathrm{f}}} \sum_{m=1}^{n_{\mathrm{s}}} H_{ml}^{\mathrm{T}} \boldsymbol{t}_{\mathrm{f},l}^{\Gamma} \\ &= \sum_{l=1}^{n_{\mathrm{f}}} \boldsymbol{t}_{\mathrm{f},l}^{\Gamma} \sum_{m=1}^{n_{\mathrm{s}}} H_{ml}^{\mathrm{T}} \\ &= \boldsymbol{t}_{\mathrm{f}}^{\mathrm{tot}} \sum_{m=1}^{n_{\mathrm{s}}} H_{ml}^{\mathrm{T}} \,,\end{aligned} \tag{3.64}$$

from which we conclude that the strict equality $\boldsymbol{t}_{\mathrm{s}}^{\mathrm{tot}} = \boldsymbol{t}_{\mathrm{f}}^{\mathrm{tot}}$ holds only if the column-sum of $\mathbf{H}^{\mathrm{T}}$ is equal to one,

$$\sum_{m=1}^{n_{\mathrm{s}}} H_{ml}^{\mathrm{T}} = 1 \qquad \text{for all}\, l \,, \tag{3.65}$$

or, equivalently, if the row-sum of $\mathbf{H}$ is equal to one. This condition ensures that a rigid body solid translation $\boldsymbol{d}_{\mathrm{s}}^{\Gamma} = \boldsymbol{d}_0$, $\boldsymbol{d}_0$ being a uniform vector, is exactly interpolated on the fluid mesh since

$$\boldsymbol{d}_{\mathrm{f},l}^{\Gamma} = \sum_{m=1}^{n_{\mathrm{s}}} H_{lm}\, \boldsymbol{d}_{0,m} = d_0 \qquad \text{for all}\, l. \tag{3.66}$$

Although Eq. (3.65) ensures consistency of the displacement interpolation, requiring additionally energy conservation does not guarantee consistency of the load interpolation, for which the column-sum of $\mathbf{H}$ should be equal to one. Consequently, the conservative approach is, in general, only partially consistent. A fully consistent interpolation may be obtained if the mapping matrix for the loads is independently generated rather than obtained as the transposed of the mapping for displacements. In this case, two matrices $\mathbf{H}$ and $\mathbf{G}$ are generated for the mapping of the displacements and loads, respectively, which requires a two-way mapping. For the construction of $\mathbf{H}$, the mapping method is first applied with the solid mesh being the source and the fluid mesh being the target. Then, the reverse mapping is applied (fluid as source and solid as target) for the construction of $\mathbf{G}$. Because in general $\mathbf{G} \neq \mathbf{H}^{\mathrm{T}}$, the fully consistent mapping may not ensure the conservation of energy. Typically, a conservative approach is preferred for the interpolation of interface mechanical quantities, such as displacement and traction, while the interpolation of thermal quantities, such as temperature and heat flux, often relies on a consistent approach.

In the following, different mapping methods are described assuming that interface quantities are interpolated from the solid mesh (source) to the fluid mesh (target). This is associated with the construction of the matrix $\mathbf{H}$. The interpolation of fluid quantities back onto the solid interface is achieved through either a conservative interpolation based directly on $\mathbf{H}^{\mathrm{T}}$ or a fully consistent interpolation by constructing $\mathbf{G}$ with the reverse mapping.

Mapping methods

The coefficients of the interpolation matrices are computed according to the chosen mapping method for pairing nodes or elements of Γ_{f} with those on Γ_{s}, as discussed in the following. It is important to note that relations such as Eqs. (3.60) and (3.63) are still valid for a perfectly matching interface ($\Gamma_{\mathrm{f}} = \Gamma_{\mathrm{s}}$ at the discrete level). In this particular case, $\mathbf{H}$ (and potentially $\mathbf{G}$) is simply a boolean matrix. The mapping for a matching mesh is thus by construction simultaneously conservative and consistent.

Nearest neighbor The easiest way to map non-matching interfaces is based on a naive nearest neighbor (NN) approach. In the situation where a fine fluid mesh is defined as target and a coarser solid mesh as source, the NN mapping proceeds as follows: for each fluid nodes on Γ_f, the nearest structural nodes on Γ_s is determined and the corresponding entry of $\mathbf{H}$ is set to one. This leads to a very poor interface interpolation since several fluid nodes will be associated with the same structural quantities, as shown in Fig. 3.20. For instance, the interpolation of the solid

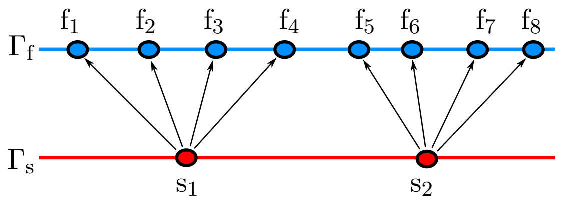


Figure 3.20: Illustration of the nearest neighbor mapping. Fluid nodes f_1 to f_4 will be assigned the same value from structural node s_1, f_5 to f_8 will be assigned data of s_2.

displacement field with the NN approach results in a non-acceptable stair-shape fluid interface reconstruction, as shown in Fig. 3.21. The NN method performs well only in the specific case of a rigid body translation of the solid interface. When interpolating fluid quantities onto the solid side (reverse mapping) in a fully consistent approach, the solid nodes would receive information only from a subset of fluid nodes.

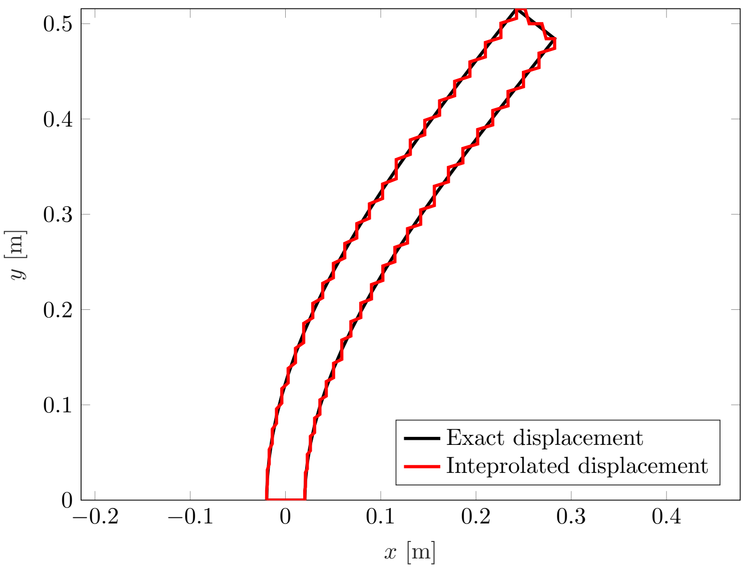


Figure 3.21: Illustration of the nearest neighbor interpolation leading to a stair-shape interface. Case of the bending of a vertically clamped beam.

Finite element interpolation A better mapping method can be developed based on a finite elements (FE) interpolation [61, 106] as depicted in Fig. 3.22. Considering again a fluid target and a solid source, the approach consists in projecting (with orthogonal projection as suggested by Beckert [106]) each fluid node f_l of Γ_f onto its closest structural element (e) of Γ_s. The natural coordinates (η, ξ) of the resulting projected point χ_l within the elements are then computed and the interpolation is thus performed using the shape functions N_m of (e),

$$\boldsymbol{d}^{\Gamma}_{\mathrm{f}_l} = \boldsymbol{d}^{\Gamma}_{\chi_l} = \sum_{m=1}^{n_\mathrm{e}} N_m(\chi_l)\, \boldsymbol{d}^{\Gamma}_{\mathrm{s}_m}\,, \tag{3.67}$$

and thus the coefficient of the matrix $\mathbf{H}$ are directly given by $H_{lm} = N_m(\chi_l)$. Because the

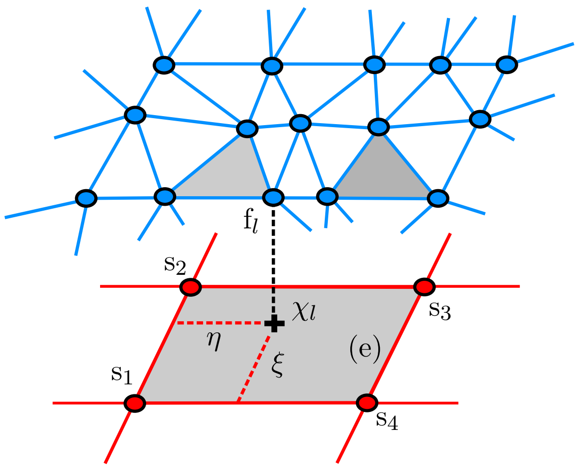


Figure 3.22: Orthogonal projection of a fluid node f_l onto its closest structural element (e) for interface mapping using element shape functions.

shape functions form a partition of unity on each structural element, $\sum_{m=1}^{n_\mathrm{e}} N_m(\chi_l) = 1$, the resulting interpolation is consistent (the row-sum of $\mathbf{H}$ is equal to one).

Mortar method Another interface mapping approach is obtained by a weighted residual method, where the continuity conditions are imposed in a weak fashion by introducing the Lagrange multiplier λ. For instance, the compatibility conditions for the displacement (same expression for the other quantities) can be written at the continuous level as

$$\int_\Gamma \lambda^\mathrm{T}\, (\boldsymbol{d}^{\Gamma}_\mathrm{f} - \boldsymbol{d}^{\Gamma}_\mathrm{s})\, d\Gamma = 0\,. \tag{3.68}$$

The discrete version of this expression implies that the field of the Lagrange multiplier (sometimes referred to as the *fourth* field of the problem [108]) has to be discretized as well. The mortar method [49, 61, 105, 108–111] usually discretizes λ with the trace on Γ_f or Γ_s of the discretization space of the fluid or solid quantities. For instance, when interpolating the displacement, the fluid target side is chosen as the discretization space on the support Γ_f. The resulting discretized version of Eq. (3.68) thus writes [61, 108]

$$\mathbf{D}\, \boldsymbol{d}^{\Gamma}_\mathrm{f} - \mathbf{M}\, \boldsymbol{d}^{\Gamma}_\mathrm{s} = \mathbf{0}\,, \tag{3.69}$$

which gives

$$\boldsymbol{d}^{\Gamma}_\mathrm{f} = \mathbf{D}^{-1}\, \mathbf{M}\, \boldsymbol{d}^{\Gamma}_\mathrm{s}\,. \tag{3.70}$$

In this expression, $\mathbf{D}$ is a square matrix whose coefficients are defined by integrals on Γ_f combining shape functions of the fluid N^f_l and Lagrange multiplier ϕ_k spaces. $\mathbf{M}$ is a rectangular matrix whose coefficients are defined by integrals, also on Γ_f, combining shape functions of the solid N^f_m and Lagrange multiplier spaces:

$$\begin{aligned} D_{kl} &= \int_{\Gamma_\mathrm{f}} \phi_k\, N^\mathrm{f}_l\, d\Gamma = \int_{\Gamma_\mathrm{f}} N^\mathrm{f}_k\, N^\mathrm{f}_l\, d\Gamma\,, \\ M_{km} &= \int_{\Gamma_\mathrm{f}} \phi_k\, N^\mathrm{s}_m\, d\Gamma = \int_{\Gamma_\mathrm{f}} N^\mathrm{f}_k\, N^\mathrm{s}_m\, d\Gamma\,. \end{aligned} \tag{3.71}$$

From Eq. (3.70), the identification of the mapping matrix $\mathbf{H}$ is straightforward, $\mathbf{H} = \mathbf{D}^{-1}\mathbf{M}$ and the interpolation is consistent given the property of shape functions to be a partition of unity [105]. The computation of $\mathbf{D}$ can be naturally performed since both shape function and support for the λ field and the fluid problem coincide. However, the computation of $\mathbf{M}$ is less direct since it mixes shape functions that are initially not defined on the same side (one on Γ_f and the other on Γ_s) and projection steps are required in order to express N_s in terms of coordinates underlying Γ_f [105,112]. More details on methods to compute $\mathbf{D}$ and $\mathbf{M}$ can be found in Heinstein *et al.* [112], for example. De Boer *et al.* [105] also reviewed a few methods based on Gauss point projection and element intersections. Several authors propose methods based on the construction of a common refinement of Γ_f and Γ_s [113,114] or *supermesh* [115,116]. Jaiman *et al.* [113,117] also compare the common-refinement method with node-projection based methods and highlight the better accuracy of the former as well as its intrinsic conservation property. Although the weighted residual/mortar method is mathematically more optimal than the FE interpolation [61], it requires computing the inverse of the matrix $\mathbf{D}$ or, in practice, solving the associated linear system, which might be a costly operation. Although Klöppel *et al.* [108] propose a dual variant of the mortar method leading to a diagonal form of $\mathbf{D}$ that reduces the cost associated to the computation of its inverse or to the solution of the linear system, the FE interpolation method has the advantage of explicitly computing the mapping matrix.

Boundary element interpolation The boundary element method (BEM) fills the space between fluid and structural interface grids with a fictitious and homogeneous elastic material [118–121]. The boundary elements relate the fluid interface nodal quantities and the structural quantities to the equations governing the behavior of the filling material described by a boundary integral equation [119, 120]. The BEM provides a consistent linear transformation between $\boldsymbol{d}_\mathrm{s}^\Gamma$ and $\boldsymbol{d}_\mathrm{f}^\Gamma$ that can be transposed to obtain a conservative approach. However the computation of the mapping matrix is quite complex since it is based on the solution of an additional fictitious physics.

Constant-Volume Tetrahedron method In the Constant Volume Tetrahedron (CVT) method [120], each fluid interface node is mapped with the three nearest structural neighbors. The triangles formed by the three structural nodes and the fluid node are combined in order to form a tetrahedron. The resulting relation between the fluid node and the three nearest structural nodes is that the projection of the fluid point onto the solid triangle moves linearly with the structural points whereas the out-of-plane component is chosen to conserve the volume of the tetrahedron. This method ensures correct mapping of rigid body motions but its formulation is intrinsically nonlinear so that a global matrix representation is not possible. Consequently, a linearization must be performed about a given structural position to obtain a suitable matrix [121] but the consistency of the mapping is then lost [119].

All the aforementioned mapping methods share a significant drawback: they are based on a mesh topology (node connectivity) representation. At the discrete level, the projection of source mesh nodes onto the target mesh elements depends on the definition of the interface normal. The normal and area discrepancies occurring for non-matching interface meshes, especially for curved interfaces and large grid-spacing mismatching, make the procedure less accurate and less robust [105, 113, 122]. In the case of the common refinement approach for the mortar method, an additional topology has even to be generated. In practice, these methods require access to the mesh topology through the individual solvers. Although topological information is well defined inside each coupled solver, it is not guaranteed that this information can be accessed and manipulated by an external coupling environment. For large problems requiring parallel partitioning, the management of interface topology becomes even more challenging due to the overlap of elements and the presence of ghost elements at the boundaries of each partition.

A very interesting alternative approach for the mapping of fluid-structure interfaces is the use of meshless methods. These methods are said to be meshless because no topological information or node connectivity is required to compute the mapping matrix; the procedure is only based on arbitrary point clouds and is thus free of any mesh projection/intersection procedure. Smith *et al.* [123] performed a significant litterature survey on several spline-based interpolation methods used for aeroelastic computations. In particular, they assessed different spline mapping functions in terms of accuracy, ease-of-use, robustness and cost-effectiveness. However, at the time of the survey, the methods were often limited to simplified geometries for which, typically, a wing was represented by a flat plate in both the structural and aerodynamical models with non-matching discretization. A small perturbation theory was used to model the flow so that only normal (out-of-plane) displacements are sufficient to describe both bending and torsion of the wing [119]. A recent extension to general three-dimensional (non-planar) interface interpolation has been developed through the use of radial basis functions.

Interface mapping with radial basis functions (RBF)

General theory of RBF can be found in Buhmann [124] and Wendland [125]. Interpolation with RBF has become a very powerful tool in multivariate approximation theory through scattered data because of its excellent approximation properties. They have been successfully applied to areas as diverse as computer graphics, geophysics, error estimation, the numerical solution of partial differential equations and even mesh deformation [107]. Their ability to treat arbitrarily scattered or gridded data makes them a perfect candidate for fluid-structure interface mapping based on general black-box solvers and modular coupling, and they have been successfully applied to fluid-structure interface mapping for aeroelastic simulations, from wing to full aircraft configurations [121,126–128]. The resulting coupling method thus keeps the dependence between the coupled solvers at its lowest level. Moreover, it is applicable to any kind of grids (structured and unstructured) and its parallel implementation is straightforward.

The basic principle of RBF interpolation is to transfer a field from a given set of points called centers to another set of evaluation nodes. In the context of fluid-structure interaction, when a quantity (the displacement is kept as an example) has to be interpolated from the solid to the fluid, the centers are the nodes located on the solid source interface mesh and the evaluation nodes are located on the target fluid mesh. The interpolated quantity is written as the weighted sum of basis functions [107,126,127]:

$$w(\boldsymbol{x}) = \sum_{k=1}^{n} \alpha_k \, \phi\left(||\boldsymbol{x} - \boldsymbol{x}_k||\right) + p(\boldsymbol{x}) \,, \tag{3.72}$$

where n is the number of source nodes, $\boldsymbol{x}$ the spatial coordinates, $\boldsymbol{x}_k$ the coordinates of the sources (centres), $p(\boldsymbol{x})$ a polynomial and ϕ the basis functions of the Euclidian distance $||\cdot||$. The coefficients α_k and the coefficients of the polynomial are determined by requiring an exact recovery of the interpolated function at the centres $\boldsymbol{x}_m$,

$$w(\boldsymbol{x}_m) = w_m \,, \tag{3.73}$$

and the additional requirements,

$$\sum_{k=1}^{n} \alpha_k \, q(\boldsymbol{x}_k) = 0 \,, \tag{3.74}$$

for any polynomial q with a degree less than or equal to that of polynomial p. The minimal degree of the polynomial depends on the choice of the basis function ϕ. A unique interpolant is given if the basis function ϕ is a conditionally positive definite function. A typical choice for the basis function is made by using functions that are conditionally definite positive of order less than or equal to two so that an interpolant can always be formed using linear polynomials p [126]:

$$p(\boldsymbol{x}) = \beta_0 + \beta_x x + \beta_y y + \beta_z z \,. \tag{3.75}$$

The choice of a linear polynomial has a significant consequence on fluid-structure interaction mapping because it guarantees a consistent mapping (exact recovery of constant function, such as rigid body translation).

The application of RBF interpolation for the mapping of the structural interface displacement $\boldsymbol{d}_s^\Gamma$ allows us to express the mapping matrix $\mathbf{H}$. When dealing with a discrete number of points, Eqs. (3.73) and (3.74) are written for the structural displacements in a matrix form as (for the x-direction, similar expressions in the other directions)

$$\begin{bmatrix} \boldsymbol{d}_\mathrm{s}^\Gamma \\ \mathbf{0} \end{bmatrix}_x = \begin{bmatrix} \mathbf{C}_\mathrm{ss} & \mathbf{P}_\mathrm{s} \\ \mathbf{P}_\mathrm{s}^\mathrm{T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix}_x , \tag{3.76}$$

where the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ vectors contain the unknown coefficients α_k and the coefficients of the polynomial p, respectively. The matrix $\mathbf{C}_\mathrm{ss}$ is a $n_\mathrm{s} \times n_\mathrm{s}$ (n_s being the number of points on Γ_s) matrix containing all the evaluations of the basis functions for distances between pairs of interface solid nodes such as $\mathbf{C}_\mathrm{ss}(l,m) = \phi\left(||\boldsymbol{x}_{\mathrm{s}_l} - \boldsymbol{x}_{\mathrm{s}_m}||\right)$ and the matrix $\mathbf{P}_\mathrm{s}$ is a $n_\mathrm{s} \times 4$ matrix whose rows are defined by the solid interface node coordinates $\begin{bmatrix} 1 & x_{\mathrm{s}_m} & y_{\mathrm{s}_m} & z_{\mathrm{s}_m} \end{bmatrix}$. Similarly, the same expression can be written for the fluid interface displacements using Eq. (3.72):

$$\boldsymbol{d}_{\mathrm{f},x}^\Gamma = \begin{bmatrix} \mathbf{C}_\mathrm{fs} & \mathbf{P}_\mathrm{f} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix}_x , \tag{3.77}$$

where, in this case, $\mathbf{C}_\mathrm{fs}$ is a $n_\mathrm{f} \times n_\mathrm{s}$ (n_f being the number of fluid points on Γ_f) matrix that contains the evaluation of the basis functions for distances between fluid and solid interface nodes such as $\mathbf{C}_\mathrm{fs}(l,m) = \phi\left(||\boldsymbol{x}_{\mathrm{f}_l} - \boldsymbol{x}_{\mathrm{s}_m}||\right)$ and $\mathbf{P}_\mathrm{f}$ is a $n_\mathrm{f} \times 4$ matrix whose rows are defined by the fluid node coordinates $\begin{bmatrix} 1 & x_{\mathrm{f}_m} & y_{\mathrm{f}_m} & z_{\mathrm{f}_m} \end{bmatrix}$. A simple combination of Eqs. (3.76) and (3.77) allows us to write the linear relation between $\boldsymbol{d}_\mathrm{f}^\Gamma$ and $\boldsymbol{d}_\mathrm{s}^\Gamma$ (for each spatial dimension):

$$\boldsymbol{d}_\mathrm{f}^\Gamma = \underbrace{\begin{bmatrix} \mathbf{C}_\mathrm{fs} & \mathbf{P}_\mathrm{f} \end{bmatrix} \begin{bmatrix} \mathbf{C}_\mathrm{ss} & \mathbf{P}_\mathrm{s} \\ \mathbf{P}_\mathrm{s}^\mathrm{T} & \mathbf{0} \end{bmatrix}^{-1}}_{\widetilde{\mathbf{H}}} \begin{bmatrix} \boldsymbol{d}_\mathrm{s}^\Gamma \\ \mathbf{0} \end{bmatrix} , \tag{3.78}$$

from which the mapping matrix $\mathbf{H}$ can be identified as the first $n_\mathrm{f} \times n_\mathrm{s}$ block of $\widetilde{\mathbf{H}}$. Obtaining the mapping matrix requires the inversion of a relatively small system. For a solid-to-fluid mapping, this matrix is of size $(n_\mathrm{s}+4) \times (n_\mathrm{s}+4)$ which is most of the time much smaller than the size of the systems involved in the computation of the solution for each physics. For a conservative approach, the transpose of the mapping matrix used to transfer the fluid loads to the structural interface is directly obtained as being the first $n_\mathrm{s} \times n_\mathrm{f}$ block of

$$\widetilde{\mathbf{H}}^\mathrm{T} = \begin{bmatrix} \mathbf{C}_\mathrm{ss} & \mathbf{P}_\mathrm{s} \\ \mathbf{P}_\mathrm{s}^\mathrm{T} & \mathbf{0} \end{bmatrix}^{-\mathrm{T}} \begin{bmatrix} \mathbf{C}_\mathrm{fs} & \mathbf{P}_\mathrm{f} \end{bmatrix}^\mathrm{T} . \tag{3.79}$$

For a fully consistent approach, the fluid-to-solid mapping $\mathbf{G}$ matrix is independently built by using the same procedure as for $\mathbf{H}$ but with a fluid donor and a solid target mesh. Thus, the matrix $\mathbf{G}$ is given by the first $n_\mathrm{s} \times n_\mathrm{f}$ block of

$$\widetilde{\mathbf{G}} = \begin{bmatrix} \mathbf{C}_\mathrm{sf} & \mathbf{P}_\mathrm{s} \end{bmatrix} \begin{bmatrix} \mathbf{C}_\mathrm{ff} & \mathbf{P}_\mathrm{f} \\ \mathbf{P}_\mathrm{f}^\mathrm{T} & \mathbf{0} \end{bmatrix}^{-1} , \tag{3.80}$$

which requires now the inversion of a larger $n_\mathrm{f} \times n_\mathrm{f}$ system.

Globally supported	
Volume spline (VS)	$\|\|\boldsymbol{x}\|\|$
Multi-quadric biharmonic (MQ)	$\sqrt{\|\|\boldsymbol{x}\|\| + a^2}$
Thin Plate Spline (TPS)	$\|\|\boldsymbol{x}\|\|^2 \log \|\|\boldsymbol{x}\|\|$
Locally supported	
Compact C0 (CPC0)	$(1-\xi)_+^2$
Compact C2 (CPC2)	$(1-\xi)_+^4\,(4\xi+1)$

Table 3.2: Examples of basis functions frequently used for fluid-structure interface mapping. For locally supported functions, $\xi = \frac{||\boldsymbol{x}||}{r}$ where r is the compact radius. The subscript + means that only positive contributions are considered (zero otherwise).

The RBF fluid-structure mapping is fully dependent on the choice of the basis function to compute the **C** block matrices. Many basis functions have been assessed by several authors for fluid-structure interface mapping [105, 107, 121, 126, 127] and especially for aeroelastic applications. The most frequent basis functions are summarized in Tab. 3.2, where a major distinction is made between globally and locally supported functions. Globally supported functions consider the contribution of every node, leading to a dense block matrix **C**. Conversely, locally supported functions introduce a fixed radius r that scales the distance between nodes,

$$\xi = \frac{||\boldsymbol{x}||}{r}\,, \tag{3.81}$$

and thus localize the interpolation by only considering the contributions of nodes that are located within a sphere of radius r with respect to the centers. Locally supported functions provide a sparse interpolation system for which the sparsity level is dictated by the value of the radius r. A large value of the radius yields a more accurate interpolation but also a denser system to store in memory and to solve. Moreover, excessively large radii lead to nearly singular matrices, because then all the entries of **C** are approximately equal to one. Conversely, small values of the radius imply sparse and light systems with lower accuracy. There is no clear rule for choosing the best value of the radius since its size might be problem-dependent. In practice, a reasonable fixed support radius for the fluid-structure-interaction problem has to guarantee a full coverage of the interpolation space. Each sphere with radius r should include at least all nearest neighbors of the considered centers. It would be very useful if we could vary the radius from center to center, but the theory only guarantees solvability for a fixed radius [126]. The dependence on the value of the support radius for locally supported functions can be rendered less important (and almost removed) by performing localized interpolations w_p on small overlapping patches of donor centers of size n_p. The union of all the patches covers all the donor interface nodes. The global interpolation w is then computed based on a partition of unity involving the local interpolations w_p over P patches,

$$w(\boldsymbol{x}) = \sum_{p=1}^{P} \eta_p(\boldsymbol{x}) w_p(\boldsymbol{x})\,, \tag{3.82}$$

where the coefficients η_p are constrained to form a partition of unity (PoU)

$$\sum_{p=1}^{P} \eta_p(\boldsymbol{x}) = 1\,. \tag{3.83}$$

A way to satisfy this last relation is to compute the coefficients as

$$\eta_p = \frac{\Psi_p(\boldsymbol{x})}{\sum\limits_{k=1}^{P} \Psi_k(\boldsymbol{x})}, \tag{3.84}$$

where the function $\Psi(\boldsymbol{x})$ may be freely defined as long as it is smooth [129]. A fair choice would be to use one of the functions listed in Tab. 3.2. Rendall *et al.* [129] propose a method to define the patches of centers. For instance, when the solid is the donor side, for each fluid nodes a patch containing the n_{s_p} nearest solid interface neighbors will be used to compute a local $\mathbf{C}_{\mathrm{ss}}^p$. By analogy, the n_{f_p} nearest fluid interface neighbors are used to compute a local $\mathbf{C}_{\mathrm{fs}}^p$. Then, the corresponding row of $\widetilde{\mathbf{H}}$ is assembled with the local matrices using the PoU and Eq. (3.82). The advantage of the localization is that the size associated to each local $\mathbf{C}$ matrix is reduced since the typical size of a patch is usually limited (20 to 50 points). Under these conditions, a direct inversion of the system formed with $\mathbf{C}_{\mathrm{ss}}^p$ can be affordable, leading to an explicit computation of the sparse mapping matrix $\mathbf{H}$, and $\mathbf{G}$ in the case of a fully consistent mapping.

3.5 Dynamic mesh treatment

As described in the previous section, a direct consequence of adopting a conforming mesh approach is the need for a dynamic adaptation of the fluid mesh to accommodate the displacement of the fluid-structure interface as the solid domain moves or deforms. Two main approaches can be considered. The first approach consists in performing a remeshing of the fluid domain: each time the fluid-solid interface deforms, a new fluid mesh is generated. For large three-dimensional meshes, the cost associated with a complete remeshing is obviously unacceptable, especially if the grid generator has to be called several times per time step in the case of a strongly coupled procedure. Additionally, mesh topology and connectivity are never conserved from one mesh to the other and thus the numerical errors introduced by interpolating the solution from the old mesh to the new one might become prohibitive. A second approach is to directly adapt the fluid mesh to the displacement of Γ_{f} by deforming the mesh, i.e., by continuously moving the grid nodes from the imposed displacement of the interface boundary to the inner volume mesh, and thus solve the fluid flow with the ALE formalism as introduced in Chapter 2. This approach fully preserves mesh topology and connectivity. However, mesh deformation can significantly degrade the mesh quality (degenerated cells, cells with negative volume, highly skewed cells, ...), with a major impact on the accuracy of the fluid solution or even on the convergence of the algorithm. This problem is even more acute for hybrid structured-unstructured meshes. Such meshes are typically used for viscous flows at high Reynolds number in the vicinity of solid surfaces to ensure a good resolution of the boundary layer. The grid cells close to the surface are usually rectangular (2D) or hexahedral (3D) with a very high aspect ratio and a strong stretching in the wall-normal direction, as shown in Fig. 3.23. The mesh deformation algorithm should ensure that the boundary layer cells keep their shape, size and orthogonality to preserve solution accuracy.

The quality and validity of the deformed mesh directly depend on the particular mesh deformation method considered. However, the numerical method must fulfill additional requirements linked to the ALE formalism when computing geometric parameters of the moving mesh, such as the node velocity $\boldsymbol{v}_\Omega$. This section first presents several mesh deformation techniques and their corresponding characteristics, and then introduces the concept of geometric conservation law.

3.5.1 Mesh deformation methods

Usual mesh deformation methods can be classified into two main classes [130]: physical analogy or interpolation. In the former category, the deformed mesh is obtained by solving a fictitious

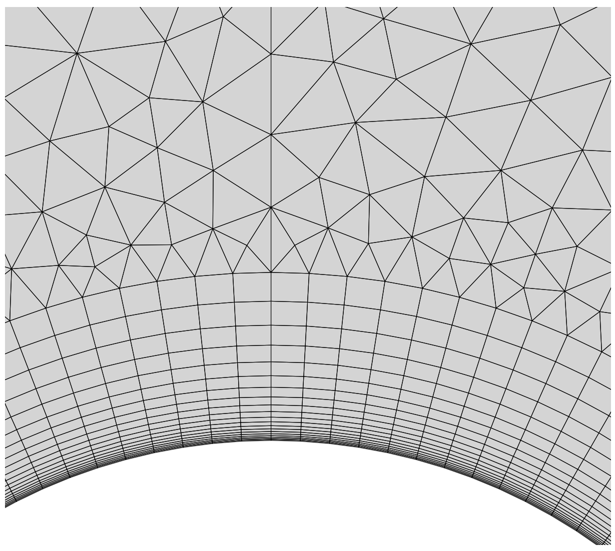


Figure 3.23: Unstructured two-dimensional fluid mesh with prismatic boundary layer mesh in the vicinity of the body surface.

physical problem governed by PDE's, where the displacement of the interface is represented by the boundary conditions. In the later category, the displacement of the interface is interpolated inside the domain mesh. Another particular category is the submesh method that combines physical analogy and interpolation. Physical analogy is used to deform a coarser version of the mesh and the resulting displacement field is then interpolated onto the initial finer level.

Physical-analogy-based methods

The physical analogy approaches the mesh deformation as a physical process governed by PDE's. Typically, the mesh is cast as a physical medium, such as a network of springs or a pseudo-elastic solid, on which governing equilibrium equations are solved to obtain the deformation field of that medium.

Spring analogy The simplest method is the spring analogy [131], where each edge connecting two adjacent nodes i and j is associated to a spring of stiffness k_{ij}, as illustrated in Fig. 3.24. In

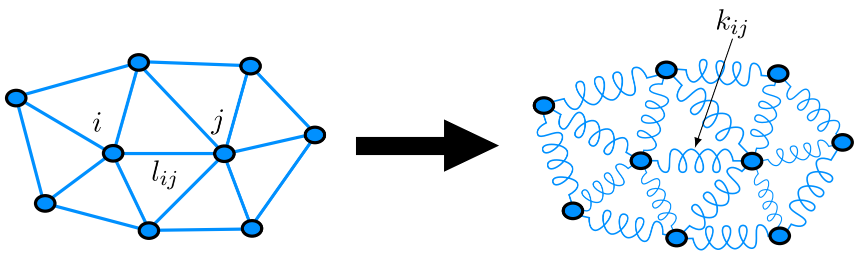


Figure 3.24: Illustration of the spring analogy method for deforming an unstructured fluid mesh in 2D.

order to prevent the nodes from collapsing into each other during the deformation, the stiffness is chosen to be inversely proportional to the length l_{ij} of the supporting edge,

$$k_{ij} \propto \frac{1}{l_{ij}}\,, \tag{3.85}$$

with $l_{ij} = ||\boldsymbol{x}_{\Omega,i} - \boldsymbol{x}_{\Omega,j}||$. Equilibrium of forces is then imposed at each node i of the fluid mesh, which leads to the system

$$\begin{aligned} \mathbf{K}(\delta \boldsymbol{X})_\Omega &= \mathbf{0} \qquad \text{on}\, \Omega_\text{f}\,, \\ \boldsymbol{X} &= \overline{\boldsymbol{X}} \qquad \text{on}\, \Gamma_\text{f}\,, \end{aligned} \tag{3.86}$$

where $\delta \boldsymbol{X}$ is the vector of unknown mesh node displacements, $\mathbf{K}$ the stiffness matrix containing the k_{ij} values and $\overline{\boldsymbol{X}}$ the displacement of the interface boundary. This standard spring analogy method has been shown to work well for 2D triangular meshes with a relatively small motion amplitude. For larger displacements, the method is not able to avoid grid line crossovers, because the definition (3.85) of the edge stiffness only prevents node collapsing. An improvement of the method consists in adding torsional springs with stiffness C_{ij} at each mesh node [132]. This additional stiffness can be explicitly related to the area A_{ijk} of the associated mesh triangles as

$$C_i^{ijk} = \frac{l_{ij}^2 l_{ik}^2}{4A_{ijk}^2}\,, \tag{3.87}$$

which provides additional rigidity to each element and prevents them to become flat during the deformation. This approach can be extended to three-dimensional unstructured meshes either by cutting each tetrahedron with triangles containing one of the associated nodes and then applying the 2D methodology to these triangles [133], or by adding a linear spring that connects each tetrahedron node to its opposite face [134], as illustrated in Fig. 3.25. Unlike some other mesh deformation methods, the spring analogy method is free of external parameter. It can be extended to other types of elements for structured meshes as well. However, despite the improvements, the method may fail to produce valid meshes for large interface displacements, especially in the presence of a fine boundary layer mesh in the vicinity of the solid interface.

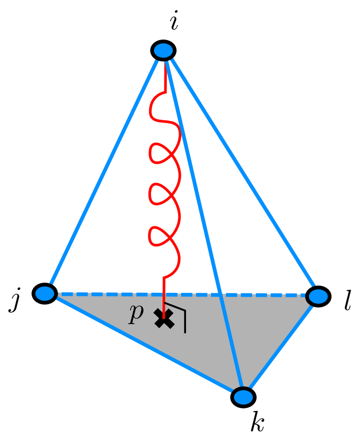


Figure 3.25: Spring analogy for 3D unstructured meshes. Additional linear spring connecting each vertex i with the opposite face in a tetrahedron. The edge springs are not represented but are part of the method as well.

Laplacian smoothing Another approach for deforming the fluid mesh by physical analogy is the Laplacian smoothing [135]. The grid nodes are relocated according to the simple Laplace equation

$$\nabla \cdot (\lambda \nabla \boldsymbol{x}_\Omega) = 0 \tag{3.88}$$

with the displacement of the fluid-structure interface Γ_f imposed as Dirichlet boundary condition. The parameter λ is here an arbitrary diffusion coefficient. A subsequent discretization of this Laplace equation leads to a system of equations similar to (3.86). A parameter λ that varies as a function of the local cell size or the distance from the solid interface can be used to better control the displacement of the grid nodes throughout the entire fluid domain. A very similar method is obtained by solving a fourth-order biharmonic equation [136] that extends the second-order Laplacian to control not only the mesh position but also the mesh spacing. Nonetheless, the extension to fourth-order has encountered only limited success due to a severe increase in cost and complexity compared to the standard second-order Laplacian. The principal advantage of Laplacian smoothing is that it is relatively simple to implement. However, the final mesh quality after deformation is still limited in case of large mesh motion [137, 138].

Pseudo-elasticity analogy The last widely used physical-analogy-based method is obtained by casting the fluid domain as an equivalent elastic solid [139–141]. The mesh deformation is then computed by solving the steady linear elasticity equation:

$$\begin{aligned} \nabla \cdot \sigma &= \mathbf{0} \qquad \text{on}\, \Omega_\mathrm{f}\,, \\ \delta \boldsymbol{x}_\Omega &= \overline{\delta \boldsymbol{x}_\Omega} \qquad \text{on}\, \Gamma_\mathrm{f}\,, \end{aligned} \tag{3.89}$$

where the Cauchy stress tensor is computed by the same Eqs. (2.22) and (2.23) as for a solid, and by using the Cauchy strain tensor defined on the mesh nodes displacement:

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\nabla \delta \boldsymbol{x}_\Omega + \nabla \delta \boldsymbol{x}_\Omega^\mathrm{T}\right) . \tag{3.90}$$

The mesh displacement on Γ_f is again imposed as Dirichlet boundary condition. The solution of Eq. (3.89) can be computed by a Finite Element (FE) formulation (see Section 3.7 for more details) leading again to an equation similar to Eq. (3.86) to be solved. Although the FE formulation used to solve the mesh deformation is an additional source of complexity, it can be limited to the linear case even in presence of large mesh motion. The rational is that no physical solution is sought but only a solution that preserves the mesh quality after deformation. For this purpose, an artificial Young's modulus (E_Ω) and Poisson coefficient (ν_Ω) are used to control the amount of deformation throughout the mesh. A variable Young's modulus is used to produce the desired effect in terms of stiffening the mesh against volume or shape changes. Local values of E_Ω can be set according to the volume or the Jacobian of the elements [139], or to the distance from a specified solid boundary. This can be used in particular to guarantee quasi-rigid motion of cells near the deforming solid interface. This preserves the boundary layer mesh where the local displacement may be much larger than the typical cell size. The pseudo-elastic casting method is therefore more efficient for relatively large mesh deformations and is applicable to any kind of grid (two- or three-dimensional, structured, unstructured, hybrid) provided that the underlying FE formulation is not restricted to any specific type of elements.

Interpolation-based methods

All the methods described above and based on physical analogies share the same drawback: a system of the size of the fluid domain has to be solved. Although the solution does not have to be physically accurate, this is obviously a significant additional cost that can become prohibitive for large problems, even if the methods can be parallelized. On the other hand, the second category of mesh deformation techniques is based on interpolation and tends to reduce the cost associated with the mesh dynamics. Interpolation schemes treat the fluid volume mesh as a problem of interpolating displacements from boundary points to other points inside the volume mesh. A significant advantage of this approach is that it does not require connectivity information and thus can be, in general, robustly applied to arbitrary mesh types, viscous boundary layer meshes or even meshes that contain general polyhedral elements or hanging nodes [130].

In the following, the inverse distance weighting, radial basis functions and sphere relaxation methods are described. The transfinite interpolation method, as described by Gaitonde *et. al.* [142], is a simple and robust mesh deformation method that is restricted to structured grids only and will therefore not be detailed in this work.

Inverse distance weighting (IDW) The IDW method is an explicit interpolation method [130, 143] where the displacement of the volume mesh nodes is described through a weighted average of all boundary node displacements as (the subscript Ω has been dropped for clarity)

$$\delta\boldsymbol{x}(\boldsymbol{x}) = \frac{\sum\limits_{j=1}^{n_b} w_j(\boldsymbol{x})\delta\boldsymbol{x}_j}{\sum\limits_{j=1}^{n_b} w_j(\boldsymbol{x})}, \tag{3.91}$$

where n_b is the number of boundary nodes (including those on Γ_f) and $\delta\boldsymbol{x}_j$ their displacement. The weights w_j are given by a function of the inverse of the distance between a volume mesh node and a boundary node:

$$w_j = w\left(\frac{1}{||\boldsymbol{x}-\boldsymbol{x}_j||}\right). \tag{3.92}$$

Witteveen [143] proposed the simple expression

$$w_j = \frac{1}{||\boldsymbol{x}-\boldsymbol{x}_j||^c}, \tag{3.93}$$

with c being a constant. Luke *et al.* [130] introduced the extended expression

$$w_j = A_j \cdot \left[\left(\frac{L_\mathrm{def}}{||\boldsymbol{x}-\boldsymbol{x}_j||}\right)^a + \left(\frac{\alpha L_\mathrm{def}}{||\boldsymbol{x}-\boldsymbol{x}_j||}\right)^b\right], \tag{3.94}$$

where A_j is the area weight assigned to node j, L_def is an estimated length of the deformation region, α is an estimated size of the near body influence region expressed as a fraction of L_def, and a and b are user-defined exponents. Details about how to choose these parameters can be found in the work of Luke *et al.* [130]. This extended version of the IDW interpolation was shown to perform well on three-dimensional hybrid meshes undergoing large boundary deformation. Due to its point-to-point explicit formulation, the associated cost is kept relatively low. Additionally, it is relatively easy to parallelize.

Radial basis functions Interpolation for fluid mesh deformation can also be performed using radial basis functions [144]. Following the discussion of Section 3.4.2, the node displacements in the volume are interpolated from the boundary with Eq. (3.72). The source centers are the nodes on the boundaries of the fluid domain and the interpolation nodes are the volume mesh nodes. For mesh deformation, it is usually recommended to discard the polynomial term of the RBF formulation to avoid its undesirable global coverage, which can cause the whole domain to move rather than deform [145]. RBF mesh deformation has demonstrated good performance for large boundary motion on hybrid meshes. However, this implicit method still requires the solution of a linear system. Although its size corresponds only to the number of boundary points instead of the entire volume mesh as in the case of physical-analogy-based methods, the associated cost might still become prohibitive. Moreover, the partition of unity approach is less desirable for mesh deformation because discontinuities or a loss of smoothness may appear in regions where interpolations overlap. In order to reduce the cost associated with the RBF mesh deformation, Rendall and Allen [146,147] have developed another procedure that aims at reducing the number of boundary points that are effectively considered to define the motion. A suitable reduced set of interface nodes that represents a pre-defined deformed geometry to a good degree of accuracy is selected by a greedy algorithm. This data reduction approach has recently gained much interest and some further improvements have been proposed such as a dynamic adaptive selection of the reduced set based on the current solid deformation [148–150] or a multiscale method that is not error-based and only depends on the initial geometry (not in the current deformation) [151].

Sphere relaxation A special kind of interpolation method can be found in the works of Zhou and Li [137, 152]. The mesh deformation is decomposed into several steps. First each volume node is given a layer index which is used to directly sort the nodes into different layers according to their distance from the boundaries. Each node has at least two layer indices corresponding to the deformed boundary and the fixed outer boundary, respectively. The nodes are then submitted to a pre-displacement by using the IDW approach as described by Eq. (3.91) with a weight being inversely proportional to the layer index. The next step consists in correcting locally the node positions by applying sphere (in 3D, disk in 2D) relaxation. Before the pre-displacement, a network of spheres, each centered at the nodes, is created in such a way that initially maximizes the tangencies and minimizes overlaps, as illustrated in Fig. 3.26. After the

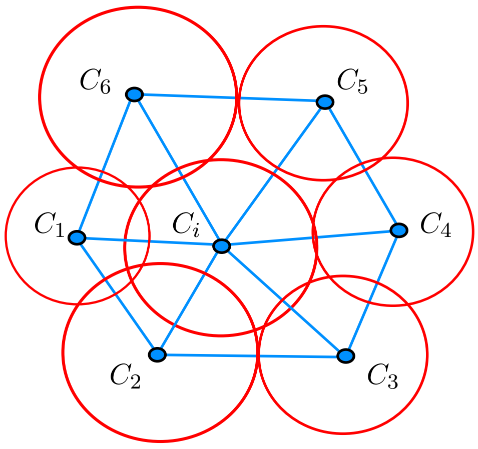


Figure 3.26: Sphere relaxation. Two-dimensional representation of spheres centered at the nodes and used for mesh deformation [152].

pre-displacement of the mesh nodes, attractive and repulsive interactions between spheres are computed according to the amount of overlap and deviation that has resulted from the pre-displacement. These interactions are used to smoothly relocate the nodes (sphere relaxation) in order to recover a configuration that maximizes tangencies and minimizes overlaps. The last step is a post-smoothing procedure that applies an additional node relocation so as to ensure good mesh quality and avoid negative cells.

Submesh methods

Finally, a special category of methods is obtained by mixing the physical analogy and interpolation in one single approach. Typical examples are mesh deformation methods that are based on a submesh approach. A coarser mesh that is easier to deform is first generated from the volume mesh. This coarse mesh can be obtained by standard triangulation [153–155] or can be defined as a Cartesian background mesh [156] which is more robust with respect to complex geometries. The nodes of the computational mesh are first simply mapped onto the submesh. Then, the submesh is deformed according to the motion of the boundary with the help of one of the aforementioned methods (e.g. pseudo-elasticity or RBF) and finally the displacement of the submesh nodes are interpolated back to the computational support. This interpolation can be performed using FE shape functions or, again, radial basis functions. The submesh method has been shown to preserve the quality of arbitrary two- and three-dimensional meshes at lower cost since only a coarse mesh has to be effectively deformed. The mapping procedure is robust even in presence of highly refined viscous boundary layer grids, since the relative position of a grid point in the corresponding submesh element is maintained. However, the interpolation procedure makes the topology characteristics of the coarse submesh visible in the computational mesh in case of large rigid-body movements [154].

3.5.2 Geometric conservation law

When selecting a method for integrating the fluid equations on a arbitrarily moving mesh, such as the ALE scheme, it is usually preferable that this scheme preserves the trivial solution of a uniform flow field. This basic requirement translates into a specific condition on the numerical scheme and on the procedure for updating the mesh position and velocity. This condition is usually referred to as the Geometric Conservation Law (GCL) [157–164]. For a general three-dimensional mesh, this law states that the change in volume (area in 2D) of each control volume between t_n and t_{n+1} must be equal to the volume swept by the cell boundary during $\Delta t = t_{n+1} - t_n$. Consequently, $\boldsymbol{x}_\Omega$ and $\boldsymbol{v}_\Omega$ cannot be updated solely based on mesh displacement when using the ALE formalism. When the fluid domain and equations are discretized using the Finite Volume Method (see Section 3.6 for more details), the GCL takes the form:

$$|\Omega_i^{n+1}| - |\Omega_i^n| = \int_{t_n}^{t^{n+1}} \int_{\partial\Omega_i} \boldsymbol{v}_\Omega \cdot \boldsymbol{n}\, dS\, dt = \sum_{j \in V(i)} \int_{t_n}^{t_{n+1}} \int_{\partial\Omega_{ij}} \boldsymbol{v}_\Omega \cdot \boldsymbol{n}\, dS\, dt\,, \tag{3.95}$$

where $|\Omega_i^n|$ is the volume of cell i at time t_n, $\partial\Omega_i$ is its boundary, $V(i)$ is the set of neighbor cells and $\partial\Omega_{ij}$ is the common face between cells i and j. The fully discrete version of the geometric conservation law (DGCL) is obtained when associated with a time integration scheme. In a series of papers by Farhat and coworkers [157–160], efforts were invested in generating first and second-order integration schemes, and particularly the related way of computing grid velocity, that satisfy the DGCL. It was shown that the DGCL not only preserves the state of a uniform flow, but also ensures that the order of accuracy of the time-integration scheme originally developed for a fixed grid is also achieved on the moving grid. For a time integration scheme of order p on a fixed mesh, satisfying the corresponding p-order DGCL would be a sufficient condition for this scheme to be at least first-order time accurate on a moving mesh [159, 160]. Consequently, a numerical scheme that satisfies the DGCL provides in general a higher accuracy than its counterpart violating the DGCL. It is important to note that the conservation of the temporal accuracy on moving grids has considerable consequences in aeroelastic problems where the energy exchange between the fluid and solid domains, e.g. for flutter assessment, is highly sensitive to time-accuracy. Additionally, unless restricted by numerical stability, a scheme that is DGCL-compliant allows larger time steps.

The (D)GCL has also been related to the stability of the corresponding numerical method. Lesoinne *et al.* [157] have shown that potential spurious oscillations may occur on the solution when violating the GCL. Later, Farhat *et al.* [160] demonstrated for several ALE schemes that satisfying the corresponding DGCL is a necessary and sufficient condition for a numerical scheme to preserve the stability of its fixed grid counterpart. They also confirmed that an ALE scheme violating the DGCL is bound to exhibit spurious oscillations and overshoots for practical computational time steps. Occasionally, such a non GCL-compliant scheme can also exhibit an unbounded behavior. For these reasons, and because the computational overhead associated with enforcing a DGCL is minimal, they finally recommended numerical methods to satisfy the DGCL when considering CFD applications on moving grids, such as fluid-structure interactions.

However, the implications of the (D)GCL have received controversial considerations [50, 161–163]. Geuzaine *et al.* [161], after further characterizing the DGCL, finally showed that it is neither a necessary nor a sufficient condition for an ALE numerical scheme to preserve on moving grids the order of accuracy of its time integration established on fixed grids. Boffi *et al.* [162] also provided opposite conclusions regarding the stability, stating that the DGCL is neither a necessary nor sufficient condition for stability (except for some particular schemes). Nevertheless, they still recognized that GCL-compliance is likely to improve accuracy of the numerical scheme and enhance its stability in some cases. Finally, although violating the GCL may not degrade the original time-accuracy of the numerical scheme, it may introduce extra artificial errors [164] so that the use of GCL-compliant ALE scheme are still encouraged in general. Considering the development of GCL-compliant numerical methods as a purely fluid solver requirement, a deep study of such schemes falls beyond the scope of this thesis and the interested reader can find many details in the aforementioned references.

3.6 The fluid solver

The numerical model representing the black-box fluid operator $\mathcal{F}$ is now detailed in the context of the SU2 solver that is used in this work. The SU2[3] solver is an open-source code originally developed at the Aerospace Design Lab of Stanford University [165–171] as a computational analysis and design package for solving partial differential equation and constrained optimization problems on general unstructured meshes. Although the framework is extensible to arbitrary sets of governing equations the core of the suite is a Reynolds-averaged Navier–Stokes (RANS) solver capable of simulating compressible and turbulent flows on dynamic meshes using ALE formalism. Available turbulence models are the Spalart-Allmaras [56] and the Shear Stress Transport $k-\omega$ [57] models whose detailed description can be found in Appendix A.

3.6.1 Spatial integration

In SU2, the unsteady ALE-RANS equations are spatially discretized using the Finite Volume Method (FVM) with a standard edge-based structure on a dual grid with control volumes constructed using a median-dual, vertex-based scheme as shown in Fig. 3.27. Median-dual control volumes are formed by connecting the centroids, face, and edge-midpoints of all cells sharing the particular node. The semi-discretized integral form of the RANS equations is given by

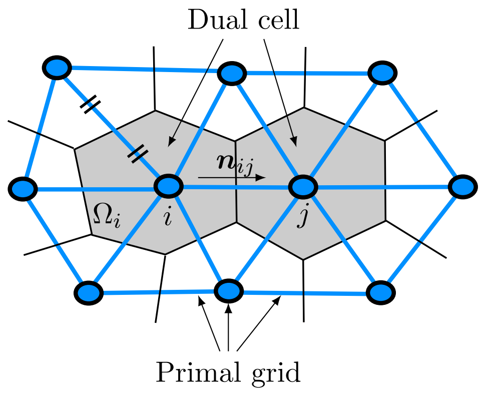


Figure 3.27: Schematic of the definition of a control volume on the dual mesh based on the primal mesh.

$$\int_{\Omega_i} \frac{\partial \boldsymbol{U}}{\partial t}\, d\Omega + \sum_{j \in V(i)} \left(\tilde{\boldsymbol{F}}^c_{ij} + \tilde{\boldsymbol{F}}^v_{ij} \right) \Delta S_{ij} - \boldsymbol{Q}|\Omega_i| = \int_{\Omega_i} \frac{\partial \boldsymbol{U}}{\partial t}\, d\Omega + \boldsymbol{R}_i(\boldsymbol{U}) = 0\,, \tag{3.96}$$

[3]Stanford University Unstructured

where $\boldsymbol{U}$ is a vector of conservative variables $[\rho, \rho\boldsymbol{v}, \rho E]^{\mathrm{T}}$, and $\boldsymbol{R}_i(\boldsymbol{U})$ is the residual. The quantities $\tilde{\boldsymbol{F}}^c_{ij}$ and $\tilde{\boldsymbol{F}}^v_{ij}$ are the projected numerical approximations of the convective and viscous fluxes, respectively, and $\boldsymbol{Q}$ is identified as the source term. The quantity ΔS_{ij} is the area of the face associated with the edge connecting node i and j, Ω_i is the volume of the control volume i and $V(i)$ is the set of neighboring nodes to node i. The convective fluxes, that include the mesh velocity for the ALE formulation, are evaluated by either central or upwind schemes. The Roe flux-difference-splitting scheme [172] evaluates the convective fluxes based on flow quantities reconstructed separately on both sides of the face of the control volume from values at the surrounding nodes:

$$\tilde{\boldsymbol{F}}^c_{ij} = \left(\frac{\boldsymbol{F}^c_i + \boldsymbol{F}^c_j}{2}\right) \cdot \boldsymbol{n}_{ij} - \frac{1}{2}\mathbf{P}\boldsymbol{\Lambda}\mathbf{P}^{-1}(\boldsymbol{U}_i - \boldsymbol{U}_j)\,. \tag{3.97}$$

In this expression, $\boldsymbol{n}_{ij}$ is the outward unit normal associated with the face between nodes i and j, $\mathbf{P}$ is the matrix of eigenvectors of the flux Jacobian matrix constructed using the Roe averaged variables and projected in the $\boldsymbol{n}_{ij}$ direction, and $\boldsymbol{\Lambda}$ is a diagonal matrix with entries corresponding to the absolute value of the eigenvalues of the flux Jacobian matrix. The discretization is first-order accurate in space but second-order accuracy is obtained via reconstruction of variables on the cell interfaces by using a Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) approach [173] with gradient limitation. Slope limiting is achieved through the Venkatarkishnan limiter [174] to preserve monotonicity in the solution by limiting the gradients during higher-order reconstruction. Another standard spatial scheme available in SU2 for evaluating the convective flux is the JST (Jameson-Schmidt-Turkel) scheme [175]. It uses a blend of two types of artificial dissipation which are computed using the difference in the undivided Laplacians (higher-order) of connecting nodes and the difference in the conserved variables (lower-order) on the connecting nodes. These two levels of dissipation are blended by using a pressure switch that triggers lower-order dissipation in the vicinity of shock waves. The final expression for the numerical flux using the JST method is

$$\tilde{\boldsymbol{F}}^c_{ij} = \boldsymbol{F}^c\left(\frac{\boldsymbol{U}_i + \boldsymbol{U}_j}{2}\right) \cdot \boldsymbol{n}_{ij} - \boldsymbol{d}_{ij}\,, \tag{3.98}$$

where the artificial dissipation along the edge ij is given by

$$\boldsymbol{d}_{ij} = \left(\varepsilon^{(2)}(\boldsymbol{U}_j - \boldsymbol{U}_i) - \varepsilon^{(4)}(\nabla^2\boldsymbol{U}_j - \nabla^2\boldsymbol{U}_i)\right)\phi_{ij}\lambda_{ij}\,, \tag{3.99}$$

in which λ and ϕ are a local spectral radius and a grid stretching parameter, respectively. Further details on the way these variables are computed can be found in Palacios *et al.* [165]. The convective terms for the turbulence models is discretized using a first-order upwind scheme.

In order to evaluate the viscous fluxes using the FVM, flow quantities and their derivatives are required at each face of the control volume. The values of the flow variables are thus averaged at the cell faces. The gradients of the flow variables are first calculated using either a Green-Gauss or a least-squares method at all grid nodes and then averaged to obtain the gradients at the cell faces. Source terms are simply approximated using piecewise constant reconstruction within each of the dual control volumes.

3.6.2 Time integration

Equation (3.96) is integrated in time using either an explicit or implicit Euler scheme (the explicit Runge-Kutta method is also available). For the implicit Euler scheme, the fully discretized version of Eq. (3.96) reads

$$\frac{|\Omega^n_i|}{\Delta t^n_i}\Delta\boldsymbol{U}^n_i = -\boldsymbol{R}_i(\boldsymbol{U}^{n+1})\,, \tag{3.100}$$

where $\Delta \boldsymbol{U}_i^n = \boldsymbol{U}_i^{n+1} - \boldsymbol{U}_i^n$. Since the residual at time $n+1$ is unknown, a first-order linearization has to be performed such that

$$\boldsymbol{R}_i(\boldsymbol{U}^{n+1}) = \boldsymbol{R}_i(\boldsymbol{U}^n) + \frac{\partial \boldsymbol{R}_i(\boldsymbol{U}^n)}{\partial t}\Delta t_i^n = \boldsymbol{R}_i(\boldsymbol{U}^n) + \sum_{j \in V(i)} \frac{\partial \boldsymbol{R}_i(\boldsymbol{U}^n)}{\partial \boldsymbol{U_j}} \Delta \boldsymbol{U}_j^n \,, \tag{3.101}$$

and the following linear system is solved to find the solution update:

$$\left(\frac{|\Omega_i^n|}{\Delta t_i^n} \delta_{ij} + \frac{\partial \boldsymbol{R}_i(\boldsymbol{U}^n)}{\partial \boldsymbol{U}_j} \right) \Delta \boldsymbol{U}_j^n = -\boldsymbol{R}_i(\boldsymbol{U}^n) \,. \tag{3.102}$$

A steady-state solution is obtained by a time-marching procedure using a specific numerical time step Δt_i^n. A local-time-stepping approach, that allows each cell in the mesh to advance at its own maximum time step, is used to accelerate the convergence to steady state. The evaluation of the local time step is based on the solution within the associated cell and a user-defined Courant-Friedrichs-Lewy (CFL) number [165].

For unsteady simulations, a second-order accurate dual time-stepping strategy [176, 177] is used. The unsteady problem is transformed into a steady problem at each physical time step, which is solved with the aforementioned time-marching procedure using convergence acceleration techniques. The following problem

$$\frac{\partial \boldsymbol{U}}{\partial \tau} + \boldsymbol{R}^*(\boldsymbol{U}) = 0 \,, \tag{3.103}$$

is then solved, where the dual time residual is given by

$$\boldsymbol{R}^*(\boldsymbol{U}) = \frac{3}{2\Delta t}\boldsymbol{U} + \frac{1}{|\Omega|^{n+1}} \left(\boldsymbol{R}(\boldsymbol{U}) - \frac{2}{\Delta t}\boldsymbol{U}^n |\Omega|^n + \frac{1}{2\Delta t}\boldsymbol{U}^{n-1}|\Omega|^{n-1} \right) . \tag{3.104}$$

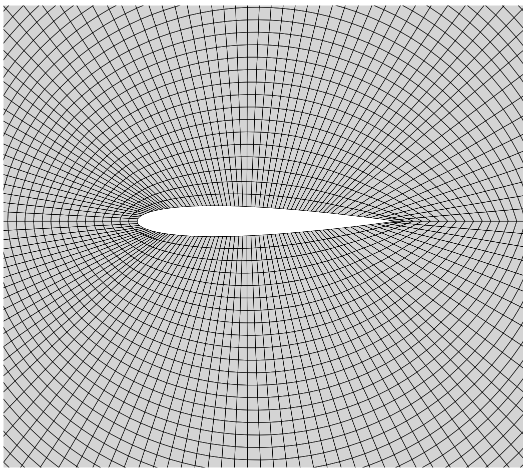
In these expressions, Δt is now the physical time step, τ is a fictitious numerical time to converge the steady state problem, $\boldsymbol{R}(\boldsymbol{U})$ still denotes the residual of the governing equations and $U = U^{n+1}$ once the steady state problem is solved.

3.6.3 Multigrid acceleration techniques

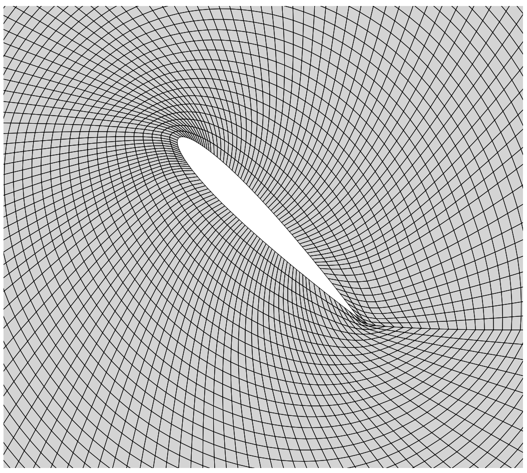
In order to speed up the damping of spatial low-frequency errors, SU2 relies on an agglomeration multigrid implementation that generates effective convergence at all length scales of a problem. It employs a sequence of grids of varying resolution that accelerates the convergence of the numerical solution of a set of equations by computing corrections to the finer-grid solutions on coarser grids recursively [178, 179]. A specific runtime agglomeration technique is used so that it is not necessary to manually create independent meshes for the coarse level; this task is completely automated in SU2 [165, 169].

3.6.4 Dynamic mesh computation

SU2 features a mesh deformation algorithm that is based on the pseudo-elasticity analogy as described in Section 3.5. The fictitious Young's modulus can be set as constant, inversely proportional to the cell volume or inversely proportional to the distance from the moving boundary. An adequate mesh quality is usually obtained with one of the last two options, even for relatively large mesh deformation, as illustrated in Fig. 3.28 for the 2D case of a pitching NACA 0012 airfoil. The mesh deformation is computed incrementally at each time step and can be subdivided into smaller deformation increments to gain robustness in case of large motion. The cost of the mesh deformation procedure can also be reduced by limiting the area of the fluid mesh that is allowed to deform. Usually a mesh region extending from the moving boundaries to a user defined limit is defined as the moving part and its outer region is considered fixed. This might



(a) Initial mesh.



(b) Deformed mesh.

Figure 3.28: Two-dimensional mesh deformation after a 45° rotation of a NACA 0012 airfoil using the elastic analogy implemented in SU2. This example uses a fictitious Young's modulus that is inversely proportional to the distance from the airfoil (moving boundary).

drastically decrease the cost associated with the mesh deformation by reducing the size of the linear system to be solved by the procedure. The SU2 solver also directly provides capabilities to describe the motion of the moving boundary that is limited to simple rigid body motion. This is typically used for one-way aeroelastic study in which the translation and rotation of the rigid boundary is controlled by a sine function with determined amplitude and frequency. For more complex rigid motion or when deformable solid boundaries are considered, an external coupling framework is needed.

The local grid velocity $\boldsymbol{v}_\Omega$, that is required for solving the ALE form of the governing equations, is computed based on node locations at prior time instances and using a finite differencing approximation that is consistent with the chosen dual time-stepping scheme [165]. For second-order accuracy in time, the mesh velocity is given by:

$$\boldsymbol{v}_\Omega = \frac{3\boldsymbol{x}_\Omega^{n+1} - 4\boldsymbol{x}_\Omega^{n} + \boldsymbol{x}^{n-1}}{2\Delta t}, \tag{3.105}$$

in which Δt is the physical time step. Finally, the numerical implementation of the GCL has been included as part of the dual-time stepping approach [170, 180].

3.7 Structural solver

The numerical models used to represent the black-box solid operator $\mathcal{S}$ are implemented in two in-house solvers: Metafor [58, 181–184] and GetDP [185, 186]. Both codes solve the governing equations for the structure in a Lagrangian formalism using the Finite Element Method (FEM). Metafor is designed to solve nonlinear thermo-mechanical problems involving large displacements and deformations of material, complex constitutive laws accounting for plastic and visco-plastic behavior and contact problems. GetDP is an open-source linear Finite Element software and a general environment for the treatment of discrete PDE-based problems including linear elasticity, heat conduction or electromagnetism, to cite a few.

3.7.1 Spatial integration

The governing equation (2.18) is discretized using the Finite Element Method. On each element e of the computational grid the displacement can be written as a combination of the nodal displacement $\boldsymbol{d}_I$,

$$\boldsymbol{d}_e = \sum_{I=1}^{N} \phi_I \boldsymbol{d}_I, \tag{3.106}$$

where N is the number of nodes associated with the element and ϕ_I the associated shape functions that depends on the type of the element. In order to improve the behavior of low-order elements for nonlinear problems, e.g to avoid shear locking for bending dominated problems, the Enhanced Assumed Strain (EAS) technique is implemented in Metafor [181, 187, 188]. This approach requires more computational effort but offers in return a better accuracy on coarse meshes.

Defining the vector of nodal displacements of the element as

$$\boldsymbol{q}_e = [\boldsymbol{d}_1^{\mathrm{T}} \ldots \boldsymbol{d}_N^{\mathrm{T}}]^{\mathrm{T}} \tag{3.107}$$

and the matrix of shape functions as

$$\mathbf{Q}_e = [\phi_1 \mathbf{I} \ldots \phi_N \mathbf{I}], \tag{3.108}$$

the displacement within the element takes the following discretized form:

$$\boldsymbol{d}_e = \mathbf{Q}_e \boldsymbol{q}_e. \tag{3.109}$$

The spatially discretized form of the governing equations are derived from the principle of virtual work (PVW) which is locally written as

$$\delta\mathcal{M}_e + \delta\mathcal{W}_e^{\text{int}} = \delta\mathcal{W}_e^{\text{ext}}\,, \tag{3.110}$$

by assuming no structural damping. The virtual work of the inertia force is

$$\delta\mathcal{M}_e = \int_{\Omega_e} \delta\boldsymbol{d}\rho\ddot{\boldsymbol{d}}\,d\Omega = \delta\boldsymbol{q}_e^{\mathrm{T}}\mathbf{M}_e\ddot{\boldsymbol{q}}_e\,, \tag{3.111}$$

where $\delta\boldsymbol{d}$ is a virtual kinematically admissible displacement and $\mathbf{M}_e$ the constant mass matrix,

$$\mathbf{M}_e = \int_{\Omega_e} \rho\mathbf{Q}_e^{\mathrm{T}}\mathbf{Q}_e\,d\Omega\,. \tag{3.112}$$

If $\boldsymbol{b}$ and $\boldsymbol{t}$ are the body forces and surface loads respectively, the virtual work of the external forces is

$$\delta\mathcal{W}_e^{\text{ext}} = \int_{\Omega_e} \delta\boldsymbol{d}^{\mathrm{T}}\boldsymbol{b}\,d\Omega + \int_{\partial\Omega_e} \delta\boldsymbol{d}^{\mathrm{T}}\boldsymbol{t}\,dS = \delta\boldsymbol{q}_e^{\mathrm{T}}\boldsymbol{f}_e^{\text{ext}} \tag{3.113}$$

with the energy-consistent vector of external forces

$$\boldsymbol{f}_e^{\text{ext}} = \int_{\Omega_e} \mathbf{Q}_e^{\mathrm{T}}\boldsymbol{b}\,d\Omega + \int_{\partial\Omega_e} \mathbf{Q}_e^{\mathrm{T}}\boldsymbol{t}\,dS\,. \tag{3.114}$$

The virtual work of internal forces is expressed as

$$\delta\mathcal{W}_e^{\text{int}} = \int_{\Omega_e} \delta\underline{\mathbf{E}}^{\mathrm{T}}\underline{\boldsymbol{\sigma}}\,d\Omega\,, \tag{3.115}$$

where the Voigt notation is used to express the strain $\mathbf{E}$ and stress $\boldsymbol{\sigma}$ tensors.

Remark

The Voigt notation allows us to express a second-order symmetric tensor as a vector. In general if

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\,, \tag{3.116}$$

then the corresponding Voigt notation is

$$\underline{\mathbf{A}} = \begin{bmatrix} a_{11} \\ a_{22} \\ a_{33} \\ a_{12} \\ a_{13} \\ a_{23} \end{bmatrix}\,. \tag{3.117}$$

Generally, $\delta\underline{\mathbf{E}}$ is related to $\delta\boldsymbol{q}_e$ through the relation

$$\delta\underline{\mathbf{E}} = \mathbf{B}(\boldsymbol{q}_e)\delta\boldsymbol{q}_e\,, \tag{3.118}$$

where $\mathbf{B}(\boldsymbol{q}_e)$ is the nonlinear strain matrix depending on the definition of the strain-displacement kinematic law. Hence, Eq. (3.115) becomes

$$\delta\mathcal{W}_e^{\text{int}} = \delta\boldsymbol{q}_e^{\mathrm{T}}\boldsymbol{f}_e^{\text{int}} \tag{3.119}$$

where the vector of internal forces can be defined as

$$\boldsymbol{f}_e^{\text{int}} = \int_{\Omega_e} \mathbf{B}^{\text{T}} \underline{\boldsymbol{\sigma}} \, d\Omega \,. \tag{3.120}$$

The stress tensor can be expressed according to a constitutive law. Although Metafor provides a wide range of constitutive models for nonlinear elasticity and plasticity [58,181,189], this thesis focuses only on purely elastic materials whose behavior is governed by Hooke's law as stated in Eqs. (2.22) and (2.23).

The discrete form of the PVW is valid for any arbitrary but kinematically admissible virtual displacement $\delta \boldsymbol{q}_e$ and is expressed on the current, i.e. deformed, configuration of the problem at any time. By assembling the contribution of inertia, internal and external forces of each finite element, the global nonlinear equation of motion can be expressed as:

$$\mathbf{M}\ddot{\boldsymbol{q}} + \boldsymbol{f}^{\text{int}}(\boldsymbol{q}) = \boldsymbol{f}^{\text{ext}}(\boldsymbol{q}) \,. \tag{3.121}$$

The nonlinear equation of motion is solved using a Newton-Raphson procedure at each time increment,

$$\begin{aligned} \mathbf{M}\ddot{\boldsymbol{q}} + \mathbf{K}_{\text{t}}(\boldsymbol{q}^k)\Delta\boldsymbol{q}^k &= \boldsymbol{f}^{\text{ext}} - \boldsymbol{f}^{\text{int}}(\boldsymbol{q}^k) \,, \\ \boldsymbol{q}^{k+1} &= \boldsymbol{q}^k + \Delta\boldsymbol{q}^k \,, \end{aligned} \tag{3.122}$$

in which $\mathbf{K}_{\text{t}}$ is the tangent stiffness matrix defined as the derivative of $\boldsymbol{f}^{\text{int}}$ with respect to $\boldsymbol{q}$. For a linear approach assuming small displacements and deformations, as it is the case in GetDP, the strain is described using the Cauchy strain tensor and the discrete equation of motion simply becomes

$$\mathbf{M}\ddot{\boldsymbol{q}} + \mathbf{K}\boldsymbol{q} = \boldsymbol{f}^{\text{ext}} \,, \tag{3.123}$$

with the constant stiffness matrix assembled from the elementary matrices

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}^{\text{T}}\mathbf{H}\mathbf{B} \, d\Omega \,. \tag{3.124}$$

In this expression, $\mathbf{H}$ is Hooke's matrix for a linear elastic material.

Similarly to the mechanical part, the Finite Element Method is also used to discretize the thermal part. If $\boldsymbol{T}$ expresses the vector of nodal temperatures, the resulting discretized heat equation can be written as

$$\mathbf{C}\dot{\boldsymbol{T}} + \mathbf{K}_{\text{T}}\boldsymbol{T} = \boldsymbol{Q}_{\text{T}} \,, \tag{3.125}$$

where $\mathbf{C}$ is the assembled heat capacity matrix, $\mathbf{K}_{\text{T}}$ is the conductivity matrix and $\boldsymbol{Q}_{\text{T}}$ is the vector of generalized heat sources that carries the coupling with the mechanical part (e.g. heat generation due to plastic deformation) when applicable. As stated in Section 2.4 the total deformation is also split into a mechanical part and a thermal part. In Metafor, the thermo-mechanical coupling is directly implemented and thus available without any additional coding effort. This is not the case for GetDP. Since it is designed to handle PDE's at a very general level, the coupling between momentum and heat equations has to be explicitly coded by the user.

3.7.2 Time integration

The time advancement of the thermo-mechanical system is performed in a staggered fashion [181, 188]. In contrast to monolithic schemes, which solve all the mechanical and thermal equations simultaneously, staggered schemes solve the mechanical and thermal equations sequentially in order to lower the CPU cost. Thus, the resolution of the coupled problem is achieved in two steps: first the mechanical problem is solved with isothermal or adiabatic thermal state which is then followed by the solution of the heat equation. In Metafor, the re-evaluation of internal stresses after the thermal step is possible for problems that are highly driven by thermal effects. For the resolution of the mechanical problem, algorithms of the Newmark family (see Appendix D) are usually selected and in particular the Chung-Hulbert algorithm. The thermal equation is solved at each time step by a midpoint generalized or trapezoidal numerical scheme. In GetDP, the mechanical problem is integrated in time using a Newmark method while the thermal problem is solved using a θ-scheme.

3.8 Simplified structural models

In many simplified FSI problems, the structure can be represented by dynamically-constrained rigid bodies (e.g., 2D pitching and plunging airfoil). In this case, the dynamics of the solid can be described by a small number of degrees of freedom, which greatly simplifies the structural computation. In particular, a complex computation of the solid deformation is not required and a much simpler algorithm can be used to compute the rigid body dynamics.

Such simplified problems are also considered in this work. Problems that can be described by a one or two degree-of-freedom rigid motion model are solved with a in-house integrator code based on either a generalized-α or a Runge-Kutta method (see Appendix D). In this case, a simplified spatial coupling is performed between the FVM fluid solver SU2 and the rigid body motion integrator. The discretized interface between the fluid domain and the rigid body is defined based on the fluid mesh only. At each coupling iteration, the interface nodal fluid loads are communicated to the rigid body integrator in order to compute the generalized forces and moments required for solving the equations of motion. The displacement of the fluid interface nodes are then obtained based on the computed rigid degrees of freedom. The rigid body motion is decomposed into the translation of a body-attached reference center O and the rotation around this point. The interface fluid node displacement between two time steps is then

$$\boldsymbol{x}_{n+1}^{\Gamma} - \boldsymbol{x}_{n}^{\Gamma} = \Delta\boldsymbol{x}_{n+1}^{\Gamma} = \Delta\boldsymbol{x}_{n+1}^{O} + (\mathbf{R} - \mathbf{I})\boldsymbol{r}_n \,, \tag{3.126}$$

where $\Delta\boldsymbol{x}^O$ is the translation of the center, which is also the translation of any points within the body, and $\boldsymbol{r}_n = \boldsymbol{x}_n^{\Gamma} - \boldsymbol{x}_n^{O}$. This typical rigid motion formulation is illustrated in Fig. 3.29. The matrix $\mathbf{R}$,

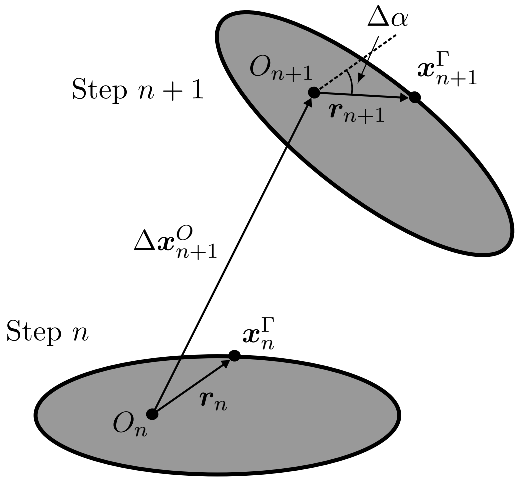

$\Delta\alpha$
O_{n+1}
$\boldsymbol{x}_{n+1}^{\Gamma}$
Step $n+1$
$\boldsymbol{r}_{n+1}$
$\Delta\boldsymbol{x}_{n+1}^{O}$
Step n
$\boldsymbol{x}_{n}^{\Gamma}$
$\boldsymbol{r}_n$
O_n

Figure 3.29: Rigid body kinematics for the simplified structural models.

$$\mathbf{R}(\Delta\alpha) = \begin{bmatrix} \cos\Delta\alpha & \sin\Delta\alpha \\ -\sin\Delta\alpha & \cos\Delta\alpha \end{bmatrix} , \tag{3.127}$$

is the classical rotation matrix for an incremental 2D rotation of angle $\Delta\alpha$ (positive clockwise) between two time steps around the center O.

Summary of chapter 3

In this chapter, the mathematical model has been extended to the numerical model. First, an introduction to the different levels of fidelity for computing FSI problems has been provided. The FSI model considered in this thesis is based on a high-fidelity approach, in which the nonlinear governing equations for each physics are discretized by, typically, the Finite Volume or Finite Element methods. An accurate representation of the interaction between the two physics can only be obtained by using a two-way coupling. A simplified, but obviously less accurate, coupling can still be performed with a one-way approach for which the structural motion is simply imposed in the fluid domain. For the numerical formulation, the distinction between the monolithic and the partitioned approach has been considered. While being accurate and numerically more stable, the monolithic approach usually requires the development of a specialized and complex code with limited flexibility and requires the mesh to be optimized for both physics at the same time. The partitioned approach is considered in this work because it provides more flexibility by using specialized solvers and fully independent meshes for each physics. However, the interface conditions have to be explicitly treated, which requires the development of an efficient communication framework for exchanging interface data between the two coupled modules.

The fixed point formulation for a mechanical coupling has been presented by introducing the Dirichlet-Neumann partitioning of the coupled problem. These operators are seen as an abstraction of the coupled solvers (Dirichlet for fluid and Neumann for solid). The partitioned coupling algorithm is obtained by calling these operators in a staggered fashion. A weak coupling is obtained when each operator is called once per time step which leads to a time-lagged advancement of the solutions, which is more prone to numerical instabilities. Conversely, a strong coupling procedure (block-Gauss-Seidel method) ensures the convergence of the interface data by evaluating the operators and exchanging their solutions several times per time step. This improves both the stability and the accuracy of the coupling procedure. Time step predictor can be used to further improve the coupling convergence rate.

Details on the numerical stability of the coupling procedure have been provided. For mechanical coupling, instabilities have been related to the added-mass effect which has been illustrated on a simplified FSI system. The impact of added-mass can be measured by the mass ratio as instabilities typically appear when the solid density approaches or falls below the fluid density. For cases with severe added-mass effect, it has been stated that even the BGS procedure cannot provide a converging coupling process. Thus, relaxation of the interface structural displacement has been introduced. A constant relaxation parameter can be provided, or a more efficient dynamic parameter can be computed using the Aitken's Δ^2 method. Newton-based coupling algorithms have been presented as a mean to further improve the coupling convergence. This method relies on first writing the coupled problem in a root finding form for the interface that can then be solved with a Newton-Raphson technique where the Jacobian (or its inverse) can be approximated based on the accumulation of the successive outputs from the coupled subsystems.

The review of the coupling algorithms has been extended to the thermal coupling. Unlike the mechanical coupling where a Dirichlet-Neumann partitioning is used most of the time, both Dirichlet-Neumann and Neumann-Dirichlet can be used for the thermal coupling. They have been referred to as the Flux Forward Temperature Back (FFTB) and the Temparature Forward Flux Back (TFFB) schemes, respectively, for which the fluid solver is taken as the reference for the direction of exchange. The stability of the thermal coupling schemes has also been reviewed and illustrated with a simple 1D CHT problem. Stability and convergence rate are related to the Biot number. It has been shown that FFTB is stable for Bi < 1 while TFFB is stable for Bi > 1. To improve the stability of these thermal schemes, relaxation can also be introduced through a numerical heat transfer coefficient that extends the stability margin. Optimal convergence rate is obtained when the numerical Biot number equals its physical counterpart, but the corresponding optimal numerical heat transfer coefficient cannot be known a priori in practice. Some relations have been proposed to compute the local optimal value, but their complexity requires further implementation effort. Specific time advancement procedures for thermal coupling have also been discussed. As the physical time scales of the fluid and solid domains usually significantly differ, efficient coupling can be obtained by reducing the data exchange frequency and by using different physical time steps within the coupled domains. When the unsteadiness of the fluid can be neglected in weakly transient problems, the advancement of the fluid solution can be reduced to a succession of steady-state computations (quasidynamic approach).

The treatment of the fluid-structure interface has been discussed as it is a key ingredient of any partitioned coupling procedure. Two main approaches have first been reviewed. The non-conforming interface meshes approach is often used in conjunction with a fluid solver based on the immersed-boundary method. In this case, the solid domain is embedded into a Cartesian fluid mesh that drastically simplifies the mesh generation procedure and alleviate the need for a dynamic mesh, but conversely adds complexity to the treatment of the boundary conditions to enforce continuity at the interface. Another drawback of the non-conforming IB approach is the difficulty to locally control the mesh refinement around the body while keeping the Cartersian structure, which leads to a mush faster increase of the mesh size for high Re cases. In this work, the conforming mesh approach is used. With this method, the fluid mesh conforms with the interface boundary which allows an explicit treatment of the interface conditions and eases local mesh refinement around the body, thus providing a locally accurate solution. However, the conforming mesh has to accommodate the motion of the solid boundary through a mesh deformation algorithm in conjunction with the ALE formulation of the governing equations. The mesh deformation process usually brings additional computational cost and its ability to generate valid or good quality mesh remains limited for large solid displacement.

In a partitioned approach using conforming body-fitted meshes, it is likely that the fluid and solid domains do not have a matching discretization at their interface. Several interpolation methods have been presented and discussed for both conservative and consistent solution transfer. Conservative interpolation ensures the conservation of the mechanical energy at the interface and is thus preferred for mechanical coupling. Consistent interpolation ensures the exact interpolation of a uniform field and is thus preferred for thermal coupling. Usually, the interpolation operator is expressed by an interpolation matrix which can then be constructed according to several mapping methods such as the Nearest-Neighbor, the Finite Element mapping, the Mortar method or the Radial Basis Function (RBF) interpolation. This last, considered in this thesis, has the advantage of not being based on mesh projection or node connectivity (meshless approach), but only relies on point clouds. This makes the RBF interpolation very suited for partitioned coupling of black-box solvers and eases the management of partitioned interface meshes (i.e. distributed for parallel computations). Setting up the RBF interpolation only requires computing distances and evaluating the basis functions. When interpolating data from a donor set of points to a target set of points, the mapping matrix is constructed by the evaluation of the basis function for the different pairs of donor and target nodes.

The ALE formalism used in the fluid domain for FSI is performed in conjunction with a dynamically deforming mesh, as a complete remeshing is too prohibitive (computationally costly, leading to topology changes). Several mesh deformation methods have been reviewed. Methods based on physical analogy such as the spring analogy, the Laplacian smoothing or the pseudo-elasticity analogy, are governed by PDE's or similar equilibrium equations. The first two methods are relatively easy to implement but remain limited in the quality of the resulting deformed mesh. The pseudo-elasticity analogy is more efficient but requires a full finite element implementation. Interpolation methods have also been considered because they tend to reduce the cost associated with mesh dynamics. Also, because the mesh deformation is reduced to the problem of interpolating displacements from boundary points to volume mesh points, they do not require connectivity information. Inverse distance weighting (IDW), RBF and sphere relaxation are typical examples of interpolation-based deformation methods. The IDW method is easier to implement and usually provides a reduced cost compared to RBF and sphere relaxation.

The importance of the Geometric Conservation Law (GCL) for ALE schemes has been briefly discussed. Despite contradicting results and claims, the development of GCL-compliant ALE schemes is still encouraged. GCL was originally introduced with the aim of preserving a uniform flow and preserving the order of accuracy of the time-scheme for moving grids. A direct consequence is that the grid velocity of the ALE formalism cannot be arbitrarily computed but must conforms to a specific GCL-compliant scheme. GCL was also related to the stability of the numerical schemes.

The fluid solver SU2 considered in this thesis has been briefly introduced. It is designed to solve the compressible unsteady RANS or Euler equations on a potentially deforming unstructured grid using the ALE formalism. Both Roe upwind and centered Jameson-Schmidt-Turkel schemes are available for the spatial discretization. Implicit Euler scheme is typically used for time-integration using a dual time-stepping approach for unsteady simulations. SU2 also features multi-grid acceleration techniques and a 3D mesh deformation capability based on the pseudo-elasticity analogy method. The dynamic mesh capability is implemented to be GCL-compliant.

Several structural solvers are considered in this work and have also been briefly described. Metafor is a nonlinear Finite Element code for thermo-mechanical problems involving large solid displacements and deformations. It uses the Enhanced Assumed Strain (EAS) technique to improve the behavior of low-order elements. The nonlinear part of the discretized equations of motion is solved by a Newton-Raphson procedure. Time-integration is performed via the Chung-Hilbert algorithm (Newmark family). For thermo-mechnical applications, thermal and mechanical problems are solved in staggered manner. The GetDP code is the second FE solver used in this thesis. It is designed to be a general environment for the treatment of discrete PDE-based problems. Only linear elastic materials can be treated with GetDP. Time integration of the mechanical problem is achieved by a Newmark method while the thermal problem is solved using a θ-scheme. Finally, one and two degree-of-freedom rigid motions, typically used for simplified VIV and aeroelastic models, can be solved by an in-house rigid body integrator code using either a generalized-α or a Runge-Kutta time integration method.

Part II

Development and implementation of the coupling environment

Chapter 4

The multi-code coupling

This chapter is dedicated to the transposition of the numerical models, algorithms and procedures into a partitioned computational framework in which independent fluid and solid solvers are coupled. In particular, it addresses how the black-box flexibility of the partitioned approach can be translated into a computational environment, where each code is specifically designed for the computation of a particular physics, and that makes efficient usage of modern and massively parallel programming architectures.

Some common considerations related to the development of coupling environments are first introduced in Section 4.1. Then, in Section 4.2, the most common methods used to communicate data between the solvers are presented. This is followed by the description, in Section 4.3, of the possible coupling technology that can be applied to bind and synchronize the solvers within a single framework. Finally, Section 4.4 provides a review of existing state-of-the-art coupling software or packages, and concludes about the rational for the development of a new coupling environment.

4.1 Common considerations

The practical application of the coupling algorithms requires the development of an efficient communication and synchronization environment that is superposed to the execution of each solver. In the partitioned approach, this coupling environment is referred to as the coupler. It is usually designed with the aim of managing all the coupling tasks such as solver calls, data exchange or mesh interpolation. The development of an efficient, robust and viable coupling tool should at least satisfy the following requirements:

- **Efficiency** The efficiency of the coupler can be related to minimizing the extra cost in CPU time introduced by the coupling tasks to reach a given accuracy in the coupled solution. Acceptable coupling efficiency can only be reached with proper formatting of exchanged data, usage of integrated solver calls and usage of efficient underlying linear algebra solutions for the interface treatment. When the coupled solvers are parallelized, the parallel scalability of the coupling should also be conserved and not be altered by the coupling environment.

- **Flexibility** The flexibility of the coupling tool is its capability to ensure compatibility with a wide range of different solvers (both fluid and solid). Hence, the substitution of one coupled solver by another should be done as transparently as possible without affecting the structure of the coupling layer. It means that, as soon as the compatibility is guaranteed, the same functionalities of the coupling tool can be accessed and reused regardless of the internal structure of the coupled code currently in use. However, a certain level of adaptivity should always be permitted in order to make possible the definition of user-customized algorithms or communication schemes. Flexibility is a key factor in the development of generic multi-physics applications.

- **Usability** The standard execution of the coupling framework and the definition of customized features should be performed with minimal effort from the user. This cannot be achieved without a clear and robust code structure whose accessible parts are provided by high-level code layers or APIs (Application Programming Interface). These APIs simplify the programming and facilitate the communication with other components by exposing deeply-coded objects or actions.

In the following sections, these three requirements are considered as a basis for the evaluation of the general performances of a coupling environment or of a particular technology used in that environment.

4.2 Multi-code communications methods

As opposed to algorithmic problems such as equation coupling or data interpolation, inter-code communication is ruled by technological and technical aspects. These aspects are of major importance since the efficiency of the coupling environment mostly depends on the chosen data transfer method. In particular, the chosen method impacts the data rate transfer and hardware usage, which are two critical factors for High Performance Computing (HPC) architectures. Several methods of communication are described hereafter and their advantage and limitations are discussed.

4.2.1 File communication

Using files as communication support is the simplest method to transfer data between independent software. Data to be exchanged are stored in files located on the hard-disk. At each communication, the sender solver first writes its interface solution as output in a file which is then read by the other (receiver) solver as input. This communication method is relatively slow in terms of data transfer speed due to limited access rate of the support medium. In a parallel approach, file I/O is usually a delicate task that requires blocking synchronization between the processes or the use of a dedicated parallel I/O library that still limits the parallel scalability. However, and when other communication methods are not possible for whatever reason, the file communication is still considered as an acceptable fall-back alternative.

4.2.2 MPI communication

The Message Passing Interface is the standard protocol used in parallelized software for distributed memory systems. When the execution of a single software is distributed among several processes (or ranks), each of them runs the same code that can be differentiated according to its process index. The processes of the distributed application are also capable of exchanging data using global (involving all the ranks) or point-to-point (from one rank to another) communications directly through the dynamic memory space and the hardware network. MPI communication can be either blocking or non-blocking. In the latter case, the execution of the code is continued even if the communication between two processes is not fully terminated. In the framework of multi-code coupling, MPI can be used to transfer data between distinct solvers if each of them is associated to one or several MPI processes. MPI processes are gathered into communicators which are distinct groups of processes sharing the same indexing and which are allowed to exchange data. Several communicators can share the same process. Rapid random memory access and high network transfer rates make this approach more efficient than the file-based communication. Additionally, as addressed in more details in the next section, it allows a more flexible and transparent data exchange within the coupling procedure.

4.2.3 Socket communication

Data exchange between solver instances can be performed via a Transfer Control Protocol (TCP) or Internet Protocol (IP) which are socket-based data transfers through a network. The data transfer speed thus depends on the hardware technology of the network. A socket is a software endpoint used to perform bidirectional communication between a server instance and a client instance. Sockets are associated with specific hardware ports on the computing unit to establish client-server communications. The TCP/IP communication protocol is not typical for massively parallel architectures, as opposed to MPI, since it requires the synchronization of the sender and receiver, among other prohibitive overheads such as port availability issues or network address identification. Although MPI outperforms TCP/IP for data exchange [190], the latter approach is still widely used for multi-code coupling.

4.3 Multi-code coupling technology

The means of coupling and exchanging data between distinct codes is not unique. Not only the support for the communication may differ, as addressed in the previous section, but also the way the communication and solver synchronization is managed varies. Several multi-code coupling procedures or technologies with different levels of complexity can be used. Advantages and limitations of the most common coupling technologies are described in the following.

4.3.1 Manual coupling

In this simplest approach, the coupling is not managed by a dedicated environment *per se*. The synchronization and the communication between the solvers are manually handled by the user and the file method is used as support for the communication. The scenario is illustrated in Fig. 4.1 and may be described as follows. The user first performs one computation with one

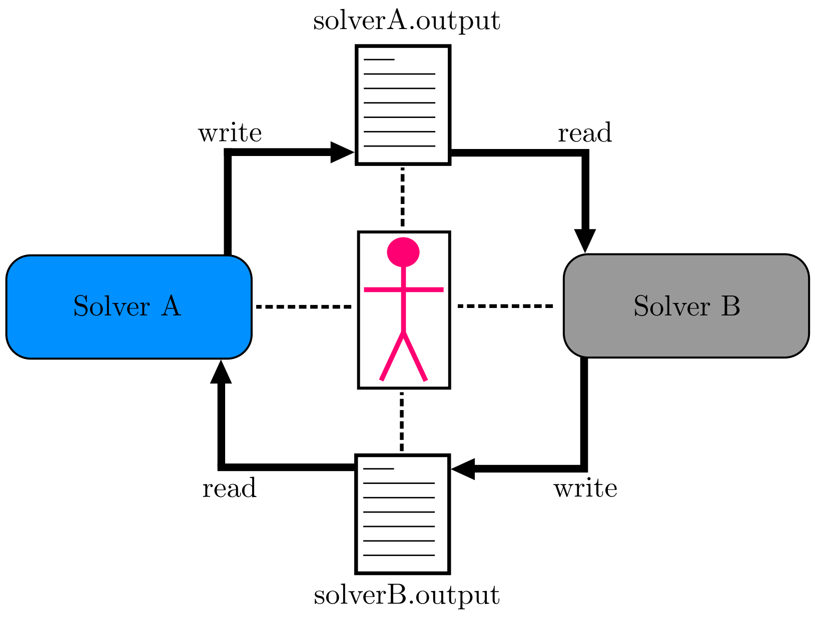


Figure 4.1: Illustation of the manual coupling based on files for data exchange.

of the coupled solvers, solver A, that stores the interface solution in an output file at the end of its computation. Then the user performs the computation with the other solver B using the interface data written by solver A as input. Another file is needed to store the data to be sent back as input to A for a new coupling cycle. In this approach, the solvers are usually not kept instantiated in the memory of the computing unit. Consequently, the computation at each solver call is fully restarted from the previous calculated solution. This implies the destruction of the computational data (mesh structure, solution field, simulation parameters) that needs to be re-generated at each coupling cycle[1], which is obviously highly inefficient and cumbersome. Furthermore, interface data treatment such as non-matching mesh interpolation has to be performed by an independent third party tool driven by the user as well. As an explicit action of the user is required at each cycle, the practical application of the manual coupling is usually limited to steady FSI computations where only a few coupling cycles are required to obtain a converged solution. For unsteady computations where several cycles are performed at each time step, this methodology is clearly impractical. Although the method can be applied with distributed solvers, the non-autonomous characteristic of the manual coupling completely cancels the benefit of the parallelization, especially if the costly parallel partitioning has to be re-generated at each cycle. Note that the coupling procedure could be automatized by scripting the coupling cycle. Nonetheless, although further limiting the manual intervention of the user, scripting does not remove the problem of restarting computations and does not increase the parallel efficiency. Given its low efficiency and many disadvantages, manual coupling is not further considered in this thesis.

4.3.2 Master-slave architecture

The master-slave architecture corresponds to the situation where the coupling is driven by one of the coupled solver which is then designated as the master. Any other coupled solver is seen as a slave component. This is depicted in Fig. 4.2. The master solver takes care of the coupling tasks and manages the computations to be performed by the slave solver. The control of the slave

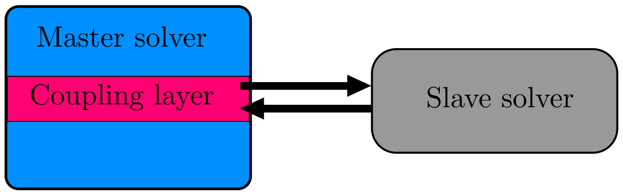


Figure 4.2: Illustration of the master-slave coupling methodology.

is handled by specific routines implemented inside the master. These specific routines actually play the role of the coupling environment. The link with the slave solver can be performed either by using sub-processing or a plug-in methodology. In the former approach, a child process is directly generated at runtime from the master instance to host and launch the slave solver. Pipes can be used to exchange data between master and child instances but the procedure is not straightforward to apply for the transfer of interface data. The plug-in methodology is designed to integrate the slave solver as a library in the framework of the master. This usually allows a better management of the coupled solvers and the possibility to use efficient communication methods such as MPI or TCP/IP. Because the synchronization of the solvers is dynamically handled by the master, no manual action from the user is required. However, as the coupling environment is embedded into one of the coupled solvers, the flexibility of this approach is limited. The coupling and the management of the parallel distribution when applicable, are intrinsically accommodated[2] to the master solver. Additionally, depending on the original code

[1]This presupposes that the coupled solver has the ability to restart from a stored solution, which is not always possible.

[2]The coupling layer is part of the code structure of the master, and cannot be transposed or reused for other codes.

design of the master, introducing coupling and communication routines at a relatively deep code level while conserving the backward structure might requires considerable development efforts. Due to its significant lack of flexibility, the master-slave coupling architecture is not further considered in this thesis.

4.3.3 Client-server architecture

One of the most common coupling technologies is based on a client-server architecture. In this case, each client solver is coupled with a central and independent server instance that gathers all the coupling tasks. All communications between the coupled solvers are run through the server which holds all the coupling data as well. Coupling algorithms and mesh interface treatment are executed at the server level. In this approach, only the compatibility with the server must be ensured for each candidate solver. The client-server coupling technology is depicted in Fig. 4.3. In order to communicate with the server, each coupled solver must use a client that is embedded

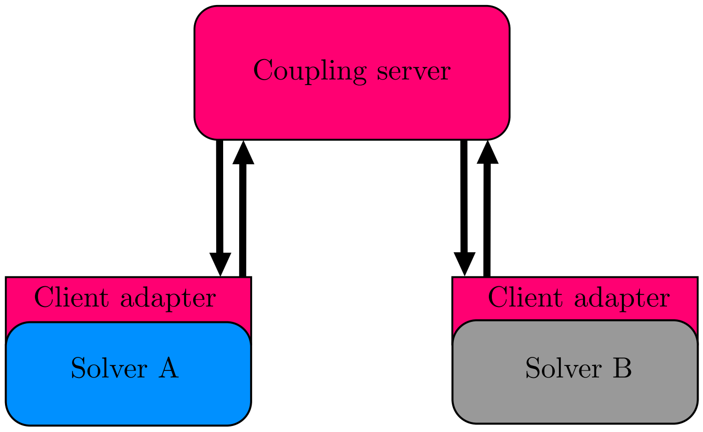


Figure 4.3: Client-server architecture with central coupling instance.

into a code adapter, and the client-server communication is usually based on TCP/IP or MPI. The adapter is composed of some specific coupling routines that have to be introduced into the coupled code, which requires additional development. Although being to a certain extent intrusive, this interfacing methodology provides a satisfying level of flexibility. Setting up the simulation is performed in several steps. First, the server is started. Then, the coupled solvers are launched and, finally, the communication network is established. The procedure is generally fully compatible with a parallel execution of the coupled solvers where each thread can be identified as a client for the server. However, and as the server is usually executed on one thread, the coupling tasks are serialized by the central instance, which significantly limits the parallel scalability of the coupled framework.

4.3.4 Unified architecture

The coupling within a unified architecture is the main focus of this work. It is at first sight similar to the client-server approach in a sense that the coupled solvers are bound to a third-party coupling tool. However, the basic idea of the unification is to replace the central server instance by a more efficient coupling environment which embeds the different codes into a single framework, as illustrated in Fig. 4.4. The main advantage of the unified methodology over the client-server approach is the possibility to manage direct peer-to-peer communication between solvers. For parallel execution of the solvers, the unified architecture also offers a proper parallelization of the coupling tasks, thus avoiding the typical serialization bottleneck of the conventional client-server coupling. Code adaption is still required in order to ensure compatibility between solvers that

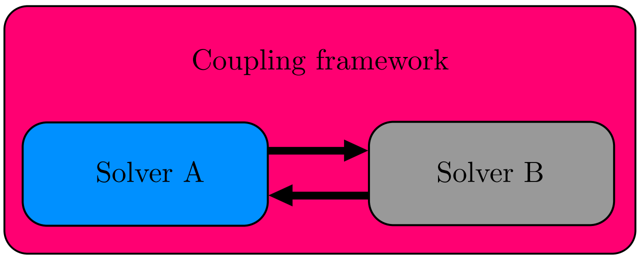


Figure 4.4: Unified coupling architecture. The coupling framework embeds the coupled solvers.

are usually compiled as libraries rather than executables. The direct integration of the solvers into the coupling environment allows an efficient management of the coupling procedures and the use of TCP/IP or MPI inter-communication methods. The computation can also be started as a whole, as opposed to the individual start of each component. Finally, the parallel structure of the coupled solvers (when applicable) can be exploited more efficiently to parallelize the data exchange.

4.4 State of the art

Although developing a partitioned coupling framework is far from being a trivial task, there is a vast amount of available multi-code coupling software. Section 4.4.1 reviews the most well-known and recent architectures, either commercial or open-source. This review is based on the one proposed by Uekermann in his thesis [190], who compared several coupling software based on the following characteristics:

- **API level** The API level goes from the low level which operates with explicit sending and receiving inter-communication routines (e.g. similarly to MPI), to the high level that provides complete coupling functionalities with a transparent user access against underlying low-level instructions. This criterion can be related to the ease of use and the flexibility of the coupling tool.
- **Coupling schemes available** The availability of built-in and ready-to-use coupling schemes, such as BGS or IQN for instance, is assessed.
- **HPC compatibility** The HPC character of a coupling software is directly related to the presence of a central serial instance that limits the parallel scalability as opposed to software with a full parallel communication layout.
- **Legal situation** This distinguishes commercial, in-house and open-source software.

The same classification criteria are used here except that two additional aspects, that are also considered by the author as important for multi-code coupling environment, are introduced in this work:

- **Communication protocol** This is related to one of the aforementioned communication methods which can be either files, TCP/IP or MPI.
- **Intrusive character** This aspect is related to introducing routines from the coupling environment into the source code of the coupled components. Depending on the intrinsic code structure of the component and the level of complexity of these coupling routines, the adaptation of the source code can be achieved with more or less development effort.

The review serves as a basis for defining the characteristics of the new coupling architecture, as further discussed and rationalized in Section 4.4.2.

4.4.1 Review of existing coupling tools

The following seven existing solutions are evaluated.

ADVENTURE_Coupler The ADVanced ENgineering analysis Tool for Ultra large REal world (ADVENTURE) [191,192] is an open-source project that gathers several tools and solvers for the analysis of different physics. The ADVENTURE_Coupler module was originally developed with the purpose of coupling other ADVENTURE fluid and solid solvers. Although many modules of the ADVENTURE project are open-source, it might not be the case for the coupler module, as already stated in Uekermann [190], since no clear access seems to be provided through the project web page. The coupling library is server-based but claims a high parallel efficiency as the server itself can be parallelized. The architecture is such that if N and M are the number of threads of the coupled solvers, the parallel server will need to be run on $M + N$ additional processes in order to establish the socket-based communications with each coupled rank. This actually doubles the total number of required cores to run a coupled computation, which might become significantly prohibitive for computing units with limited resource budget. The ADVENTURE_Coupler provides a coupling scheme based on the Broyden method, that belongs to the Newton family as IQN-ILS [192]. To perform coupling analysis, only a few communication libraries from the coupler have to be integrated into the solvers. Non-matching mesh treatment is performed using shape functions of elements. Examples of FSI coupling performed with ADVENTURE can be found in Yamada *et al.* [192] and Kataoka *et al.* [191].

EMPIRE Enhanced Multi Physics Interface Research Engine (EMPIRE) [193] is an open-source coupling software which is based on a client-server approach. The central server instance is not parallelized, which limits its HPC capabilities, but claims to be unlimited in the number of simultaneous codes that can be coupled. EMPIRE is divided into two components: the coupling server instance and an API library, acting as a client, that has to be implemented in the client codes. The communication between the clients and the server is supported by the MPI protocol. EMPIRE provides coupling schemes such as the relaxed BGS, as well as non-matching mesh interpolation capabilities based on the mortar method or the simplest nearest neighbor projection. A wind turbine aeroelastic study based on EMPIRE can be found in Sayed *et al.* [194].

MpCCI The Mesh-based parallel Code Coupling Interface (MpCCI) [195, 196] is the standard commercial software for code coupling. It is based on a client-server architecture and gathers many ready-to-use adapters for commercial solvers but also provides API which can be included in other codes. The code adapter consists of three distinct components: the coupling manager that controls the boundary values to be exchanged, the client that implements these intercommunications via TCP/IP sockets, and the code driver which is the code-specific part of the adapter and provides the access to the solver data structure. For this data access, specific coupling routines have to be implemented in the coupled solvers. Although being compatible with parallel code execution, the central instance is not parallelized, which limits its HPC capabilities. Non-matching mesh interpolation is performed with shape functions or projection. Coupling schemes are available but remain limited in terms of acceleration techniques [190]. However, and unlike most open-source solutions, MpCCI features a graphical user interface that helps the user setting up the coupled simulation. Typical applications of coupling between CFD and CSM commercial solvers with MpCCI can be found in the work of Debrabandere [37].

OpenPALM It is a generic open-source coupling framework designed for massively parallel applications [197,198]. OpenPALM can be considered as a unified architecture combining three main components: the PALM library that provides general management of the coupled applications and exchanged data, the CWIPI library for the treatment of data based on distributed (non-matching) meshes, and the PrePALM graphical user interface that helps setting up the coupled simulation and monitoring it at runtime. The scheduling and the communication between the different solvers are achieved by high-level primitives that have to be included in the coupled codes. The communication between the coupled solvers is performed by MPI. To the best knowledge of the author, OpenPALM provides non-matching mesh interpolation using projection schemes but no clear built-in coupling algorithm seems to be directly available. An example of application of the OpenPALM coupler to CHT in an industrial combustion chamber can be found in Duchaine *et al.* [197].

OASIS This is an open-source coupling tool originally designed for climate modeling applications but its development has been redirected towards more generic coupled applications [199, 200]. The recent versions of OASIS are HPC compliant due to its unified architecture. The coupling tasks are not performed by a centralized instance running on its own processes. Instead, library components are linked to each coupled codes to generate a fully parallel communication layer based on MPI. Coupling routines are introduced into the coupled codes via a rather low-level API. OASIS provides non-matching grid treatment through nearest-neighbor, bilinear or bicubic interpolation. There is no built-in coupling schemes available for fluid-structure interactions. Recent examples of climate coupling applications using OASIS can be found in Sanchez-Gomez *et al.* [201] and Ličer *et al.* [202].

preCICE The preCICE coupling framework is an open-source library-based coupling architecture which has been specifically designed to maintain an efficient parallel scalability by avoiding the use of a server instance [203, 204]. It is a unified architecture that can be used with black-box commercial solvers as well as open-source and in-house solvers. Parallel solvers are fully supported and the data exchanged layout is based on efficient point-to-point communication via MPI or TCP/IP. In order to couple solvers with preCICE, high-level API routines have to be integrated into the solver code. Non-matching mesh interpolation are based on nearest neighbor mapping or Radial Basis Functions. preCICE also provides built-in coupling schemes with convergence acceleration techniques such as Aitken's relaxation and IQN. Coupled applications using preCICE framework can be found in the paper of Mehl *et al.* [203] or in the thesis of Uekermann [190].

FUNtoFEM This a Python-based framework developed for both high-fidelity aeroelastic analysis and adjoint-based aeroelastic optimization [205, 206]. It has been used to couple the adjoint-enabled fluid solver FUN3D [207, 208] with the adjoint-enabled structural solver TACS [8]. The framework provides a high-level API which abstracts the aeroelastic analysis and design into two concepts, the model and the driver. The former stores the data-structure required for the computation, while the latter implements the coupling algorithm and orchestrates the data transfer between the components. The coupled solvers interact with the coupling framework using Python wrappers generated with Cython. The architecture is based on a parallel client-server approach which avoids serialization bottleneck when transferring interface data, but requires network communication (such as TCP/IP) to proceed.

A synthetic summary of the coupling software review can be found in Tab. 4.1, where the evaluation criteria of Uekermann [190] are used in addition to those proposed in this work. The criteria used to compare the coupling software are considered by the author as the most relevant in the particular field of multi-code applications. However, the list is not exhaustive and more common, but not less important, aspects such as portability, maintainability and expandability must also be considered as general requirements for the development of any kind of software.

	API level	HPC	Legal	Coup. schemes	Communication	Intrusive
ADVENTURE [192]	med	yes	in-house	yes	TCP/IP	yes
EMPIRE [193]	med	no	open	yes	MPI	yes
MpCCI [196]	med	no	comm.	yes	TCP/IP	yes
OpenPALM [198]	high	yes	open	no	MPI	yes
OASIS [200]	low	yes	open	no	MPI	yes
preCICE [204]	high	yes	open	yes	TCP/IP/MPI	yes
FUNtoFEM [205]	high	yes	open	yes	TCP/IP/MPI	no

Table 4.1: Summary of existing coupling software evaluated according to selected criteria.

4.4.2 Motivation for the development of a new coupling framework

The review performed in the previous section clearly shows that the development of a new coupling software is not a technological breakthrough. All the existing architectures listed in Tab. 4.1 perform well in almost every assessed criterion. However, several specific aspects for which there is room for improvement still deserve to be investigated. This justifies the presence of the last two columns in Tab. 4.1. Firstly, it shows that every coupling software depends on either TCP/IP or MPI protocols to perform inter-code communication. This means that somehow the data exchange is reduced to formal send/receive actions performed by the solvers and by the server instances (for server-based approaches), even if these actions are hidden behind high-level APIs. Thus, the achievement of the coupling fully depends on the availability of specific technologies and software libraries that are not fail-proof or that could introduce compatibility issues. Secondly, it is obvious that all, except FUNtoFEM, coupling packages are based on specific coupler-related routines to be introduced into the solver codes. Usually these routines are specifically developed under the form of APIs that minimize the invasiveness, but that do not avoid the need for a certain amount of specific accommodation of a part of the coupled solver source code.

In this thesis, the ambition is not to develop and provide the ultimate solution in terms of coupling software that could surpass all other existing tools. However, the objective is to propose, evaluate and develop an alternative which circumvents the two aforementioned drawbacks by:

1. not relying on any communication protocol such as TCP or MPI for performing inter-code data exchange, and
2. excluding any coupling-related routines to be introduced into the solver codes,

while maintaining the accessibility to all coupling functionalities through a high-level unified environment. A higher level of flexibility and usability can thus be reached, without altering the efficiency. This motivates the development of the coupling environment CUPyDO, which is a multi-languages coupling environment based on a Python wrapping approach. The strength of CUPyDO is its capacity to unify the coupled solvers into one single object-oriented framework in which they are not considered as independent executables but are treated and manipulated as simple programming objects. The detailed implementation of CUPyDO is presented in the next chapter.

The rational for developing a new FSI coupling tool can also be put under the light of the general development of the scientific research at the University of Liège. The purpose is then to provide a new tool to students and scientists for conducting fundamental research with minimal focus on technical or technological aspects. This can be achieved by providing not only the most accessible, ready-to-use and flexible tool but also by allowing the user to customize the underlying algorithms and the coupling schemes in an intuitive fashion. Many single-physics computational codes are today under intensive development at ULiège, and of course in other world-wide institutions, for design and research. This also calls for the development of a new modern and generic multi-physics platform that could allow these models to communicate and interact with minimal technical effort.

4.5 General challenges and basic requirements of the multi-code coupling for fluid-structure interaction

The development of an efficient coupling tool, either server-based or unified, faces several challenges. These can be either technical, e.g. the management of parallel communication, or purely numerical, such as the presence of added-mass coupling instabilities. In addition, the coupling between independent solvers cannot be achieved if these solvers do not satisfy some minimal requirements or do not provide some flexibility within their executions.

The first basic requirement is the capability of the coupled codes to restart the same time step several times with different boundary conditions and to be controlled while keeping their own data structure instantiated. Usually, the standard standalone execution of the code only permits a complete restart, i.e. with a re-generation of the data structure, from a previously computed solution. Depending on the structure of the coupled solver, this requirement may be considered as prohibitive and may demand considerable development effort.

As the coupling is based on the exchange of solver data, this data has to be exposed and made accessible from the outside of the code. This is not always easy depending on how deep the transferred data is defined in the overall code data structure. As interface data is defined through boundary conditions, accessibility and flexibility in the definition of these boundary conditions are required. When the meshes are not matching, additional data are needed. Meshless mappings (e.g., RBF-based) only require node coordinates while other mappings need additional topological information to perform mesh projection. As already mentioned in the previous chapter, access to solution derivatives and Jacobians is even more complicated to obtain in a black-box coupling fashion.

Assuming a conformal mesh treatment (see Section 3.4.1), the fluid solver must feature dynamic mesh capability and ALE formalism. This is again not always part of the standard requirement in the design of standalone fluid solver codes. The mesh dynamics could be handled by the coupling tool, but it would require access to the entire fluid domain topology instead of just the part belonging to the interface. Besides, the constraints on the numerical flow integration schemes inherited from the ALE formalism cannot really be managed by the coupling tool. Consequently, the ALE formalism should be part of the fluid solver capabilities.

Flexibility of the partitioned coupling approach cannot be achieved without a perfect independence between the coupled solvers, i.e., none of the solvers needs to know anything about the other one. Although this solver independence seems intuitive in the case of a server-client coupling due to the presence of the central instance in the role of the separator, it is less clear for unified architectures. In this case, the success of the coupling is ensured through a proper management of the inter-code communication via high-level API adapters and coupling routines. There is thus no direct knowledge, at the solver level, of what takes places below the API (abstraction principle). This becomes even more important for distributed execution involving parallel inter-communication.

Finally, all the aforementioned requirements show that, irrespective of the coupling architecture considered, minimal amount of code adaptation is always needed. When the original design of the code is not perfectly aligned with the structure of the coupling environment, a link to support inter-communication and execution control has to be added. Consequently, a fully non-intrusive coupling is inconceivable. However, the way this accommodation is performed is still open for consideration and improvement. This is the central point of this work, addressed through the development of CUPyDO, whose implementation is detailed in the next chapter.

Summary of chapter 4

In this chapter, multi-code communication methods and coupling technologies have been reviewed. First some common considerations, such as efficiency, flexibility and usability, have been highlighted as basic requirements for the tool used to couple independent solvers.

Data communication between codes can be typically based on files, MPI exchanges or TCP/IP socket communications. MPI communications are expected to be the most efficient, especially for HPC-compliant architectures. Four coupling methodologies have been considered. The manual coupling is a non-autonomous approach that has limited efficiency and usability due to the explicit action required by the user at each coupling step. In the master-slave architecture, the coupling is driven by one of the coupled solvers. The coupling process becomes then autonomous and efficient communication methods can be used. However, the coupling is usually accommodated to the master solver which creates a serious lack of coupling flexibility. The client-server architecture can improve this lack of flexibility. Different solvers can be coupled through specific code adapters to a central server that gathers all the coupling tasks and the communication of data. However, the parallel scalability of the coupled framework is limited, as the server is usually not parallelized itself. The unified architecture, followed in this thesis, is similar to the serve-client approach, but gathers the coupled solvers into one single framework which improves the communication between the solvers as well as the parallel scalability by offering direct peer-to-peer communication.

Several state-of-the-art existing coupling software have been reviewed and compared based on relevant characteristics such as their API level, the availability of coupling schemes, the HPC-compliance, or the intrusive character. By considering this review, it has been emphasized that, far from the idea of developing the ultimate solution, there is still room for improvements. This has motivated the development of CUPyDO, which is a multi-languages coupling environment based on a Python wrapping approach. The specificity of CUPyDO lies in the idea of turning the coupled solvers into black-box modules that are integrated into a unified environment.

Finally, some general challenges and requirements for multi-code coupling technologies in FSI have been discussed. Particularly, the flexibility in the control of the solvers, the data accessibility, the ALE formalism, the abstraction principle and the adaption of the coupled codes have been discussed.

Chapter 5

The coupling tool CUPyDO

This chapter describes the implementation of the coupling environment CUPyDO[1]. It is a multi-language C++/Python environment for coupling two existing and independent solvers into a unified architecture, i.e. with no central server instance. The particularity of CUPyDO is that the coupled codes are considered as libraries instead of executables and can thus be manipulated as simple programming objects. The main purpose is to leverage the coupling flexibility of the partitioned approach and the usability of the coupling tasks through the use of a Python wrapping approach as coupling mechanism. The coupler provides ready-to-use coupling algorithms as well as interpolation capabilities for non-matching fluid-structure interface meshes. Parallel functionalities are also available, based on communication (collective or peer-to-peer) between the processes of each solver involved in the computation. Python bindings for the Message Passing Interface (MPI) protocol as well as Python bindings for the PETSc library, used for all parallel linear algebra operations (mainly required for the mesh interpolation step), are available. Point searches and filtering are performed using binary trees for efficient computation of nearest neighbors during the mesh mapping.

5.1 Design strategy

In this thesis an open-source coupling environment based on the Python wrapping methodology is proposed as an alternative that does not fully rely on MPI or TCP/IP protocol for communication between the coupled modules. Emphasis is put on the modularity of the coupling mechanism with no explicit coupler-specific routines to be introduced into the coupled codes, thus minimizing intrusive code modification. User-friendliness is also ensured by providing ready-to-use coupling functionalities, such as interpolation methods for non-matching grids or iterative coupling algorithms, but also by limiting the amount of actions required by the user to set up and launch a coupled simulation. HPC-compliance is also part of the development chart by allowing the use of distributed solvers, parallel inter-communication and parallel management of the coupling tasks.

The main design drivers of CUPyDO can be formulated as follows, in order of priority:

1. minimize development effort to couple a given solver and limit source code invasiveness,
2. guarantee coupling flexibility and modularity,
3. maximize code reusability,

[1]CUPyDO is not an acronym but simply a reference to the antique god Cupid (latin Cupido), in the roman mythology, who is the symbol of love. With his bow and arrows, he had the power to make people fall in love with each other and to bring them together. The metaphor is then created with CUPyDO which is designed to couple codes in a unified framework. Since it is based on the Python language, the letter *i* was replaced by *y* in order to form the *Py* syllable as a reference to this programming language.

4. provide high-level coupling schemes,
5. provide high-level non-matching mesh interpolation schemes,
6. perform parallel inter-communications and coupling tasks,
7. ensure maintainability and expandability.

These characteristics are supported by the technical implementation based on a Python wrapping coupling procedure and built-in coupling tasks. These tasks are accessible at a high level to the user by hiding the complex management of exchanged data between the codes, especially when it is performed in parallel for distributed applications. The realization of the design objectives is also supported by the object-oriented (OO) structure of CUPyDO. This is a key point for establishing compatibility with each coupled solver while keeping their software independence. Furthermore, maintainability and expandability rely on the use of modern programming concepts implemented in CUPyDO. Finally, the development is conducted in such a way that external and powerful libraries can be integrated for the treatment of low-level functionalities such as basic linear algebra and MPI communications (mainly used for the treatment of non-matching meshes). The use of established libraries is obviously an additional guarantee of computational efficiency. Once again, these functionalities are hidden by the high-level layer accessible to the user and are thus fully transparent at runtime.

5.2 Multi-language programming and Python-wrapping procedure

The implementation of CUPyDO is based on a multi-language programming approach where the Python language is used at the highest coding level. It is typically used to interface the independent solvers, usually written in a compiled language, in one single and integrated framework. The flexibility of the Python language thus facilitates the synchronization of the solvers and the data exchange. With this technology, modules and functionalities of the solvers are wrapped into a Python layer that behaves as a code driver for CUPyDO.

5.2.1 Generating the Python wrapper

The Python-wrapping procedure can be illustrated as follows. Let us consider a simple C++ toy object that represents a solver to be coupled with CUPyDO:

```
//File: SolverExample.h
//C++ code for the definition of SolverExample class
#pragma once

#include <string>
//If required, headers from the core code can be used
#include "solverFunctions.h"

class SolverExample{
  double _data;
  std::string _tag;
public:
  SolverExample(std::string const& valTag);
  ¬SolverExample();
  void setData(double valData);
  double getData() const;
  void computeOneTimeStep(double timeStepSize);
};
```

This code can be interfaced and exposed to Python by using the Simplified Wrapper and Interface Generator (SWIG) tool [209]. SWIG takes the declarations found in C/C++ header files and uses them to generate the wrapper code needed by scripting languages (such as Python) in order to access the underlying C/C++ code. This is performed by an additional compilation step. The generated Python wrapper plays the role of a scripting API without using brute code translation nor interfering with the libraries and executables created during the primal compilation. The procedure is schematically illustrated in Fig. 5.1 for the simple toy C++ code. In this case the source code is composed of three files: the header (.h) used for the declaration

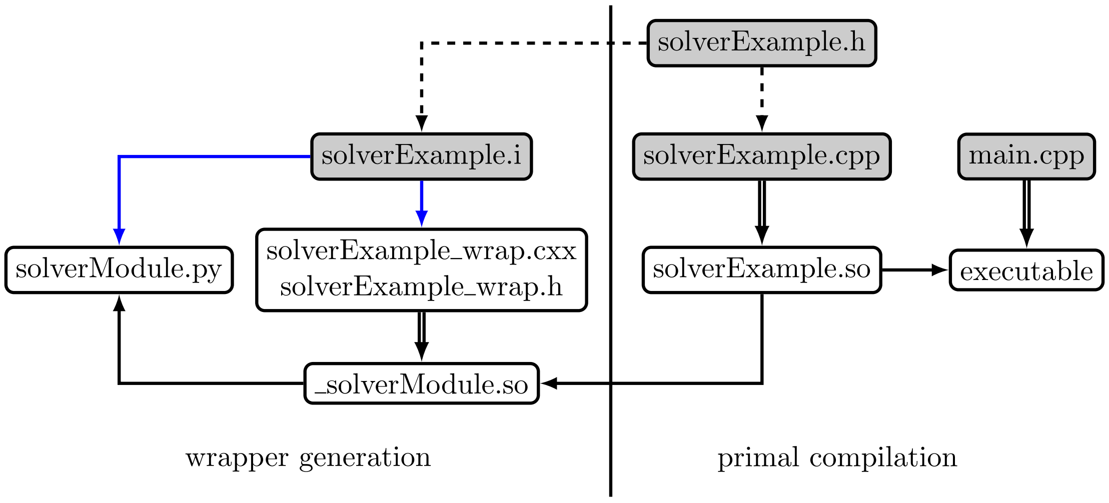


Figure 5.1: Compilation steps and generation of the Python wrapper for a simple C++ code. Grey boxes represent the source files (written by the developer). Dashed arrows are for include, double arrows are for compilation, simple arrows are for library link and blue arrows are for SWIG wrapper generation.

of the `SolverExample` class and two source files (.cpp), one for the class definition and one containing the `main` function for the executable. This source code can always be compiled as a standalone executable, as illustrated in the right part of Fig. 5.1. The generation of the Python wrapper is shown in the left part of Fig. 5.1. SWIG first takes a specific interface file (.i) as input. This file typically contains the declaration of the headers of the corresponding classes or functions to be exposed to Python. SWIG supports many C/C++ features such as constructor/destructor, virtual and static members, public inheritance, function and operator overloading, references, (smart) pointers, templates and namespaces, whose interfacing can be optionally tuned in the SWIG input file. Type mapping is also supported and defined through the input file. This includes not only basic types such as `double`, `integer` or `string` but also `std::vector`, `std::map` or static arrays that can be mapped into Python `list`, `dic` (dictionnary) or NumPy arrays [210], respectively. From the interface file, SWIG automatically generates a wrapper source code (.cxx and .h files) and a Python module. The wrapper source code contains many definitions and declarations used to resolve the interfacing and the type mapping between the C/C++ and Python languages. This new source code is also compiled into a library which is finally linked to the Python wrapping module of `SolverExample`.

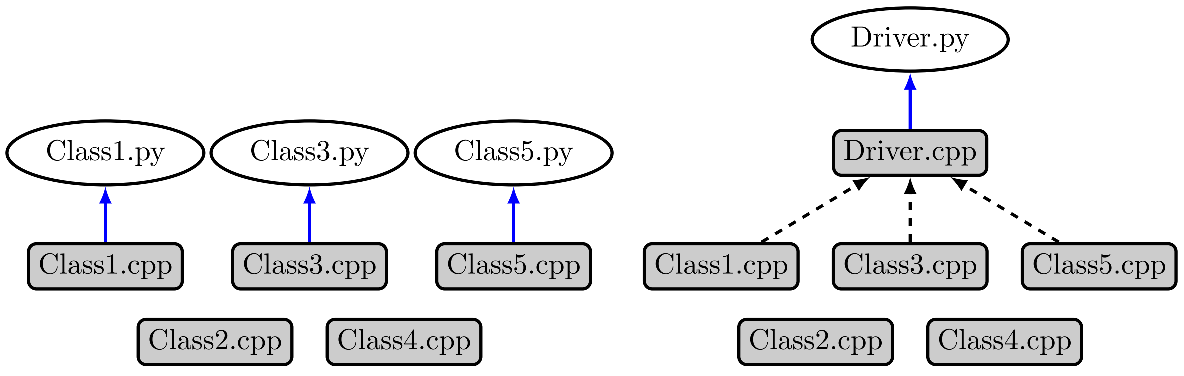
The key role of SWIG is to generate the Python wrapper by complex interfacing mechanisms and, at the same time, to avoid straight code translation. This opens the possibility for the user to manage the code with the inherent high-level flexibility of the Python language while it is executed under its own compiled language (C/C++). When properly managed, the interfacing overhead is almost negligible so that the computational efficiency of the original language is conserved. The Python wrapper of the toy class `SolverExample` is then used in Python with, for instance, the following code:

```python
#Python code that uses the wrapper of SolverExample
import solverModule

timeStepSize = 0.001

pysolver = solverModule.SolverExample("Put a tag here")
pysolver.setData(100.0)
data = pysolver.getData()
pysolver.computeOneTimeStep(timeStepSize)
```

For more complex codes, the generation of the Python wrapper can be performed in several manners depending on their inherent structure. The first approach for generating a wrapper, illustrated in Fig. 5.2(a), is by exposing each class[2] individually, resulting in several equivalent Python modules. Another way, as illustrated in Fig. 5.2(b), is to first develop a driver layer which includes the functionalities of several classes and then expose this layer to Python. The



(a) Wrapper based on multiple classes.

(b) Wrapper based on a driver layer.

Figure 5.2: Illustration of two manners of generating the Python wrapper depending on the exposed-code structure.

choice of one particular approach is mainly related to the original structure of the code and to the decisions that have oriented its design throughout its development. The multiple object approach is a regular choice for codes whose wrapper layer is developed at the same time as the underlying core structure. This has the advantage of providing higher flexibility to access the core functionalities. When the wrapper has to be developed on top of an existing and well-established core code structure, the driver layer approach may be easier to implement but may also decrease this flexibility. Finally, it is worth noting that only a restricted part of the code can be exposed if there is no interest for some routines to be accessible from Python.

Remark

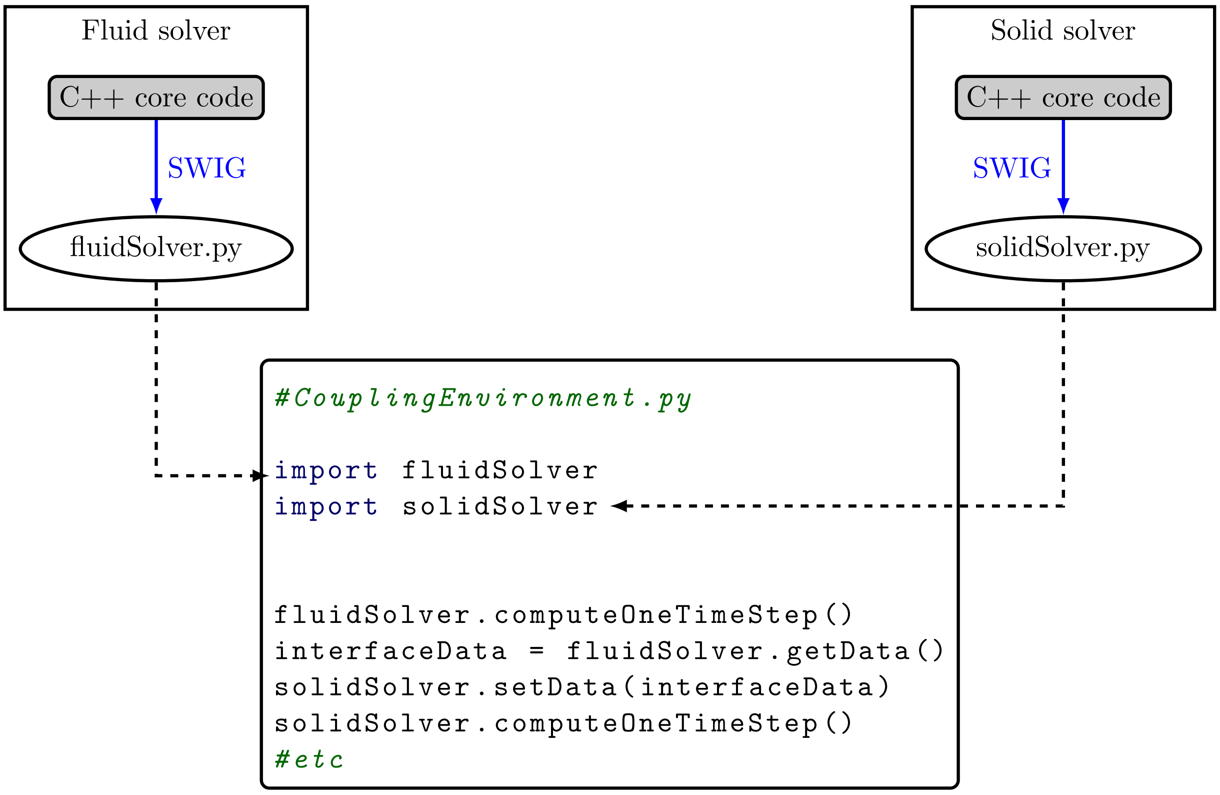
SWIG can interface C/C++ code with many other scripting languages such as Javascript, Perl, PHP, Tcl and Ruby. However, there is so far no possibility to directly expose other

[2]Objects and classes are usually specific to the C++ language. For the C language, this can be equivalently seen as group of functions.

compiled languages such as Fortran for instance. A typical work-around would be to generate a C/C++ API of the Fortran code which is then interfaced with Python.

5.2.2 Coupling with the Python wrapper

Python is one of the most used scripting language in the scientific (either academic or industrial) community and today Python wrappers are most commonly used to create high-level interpreted programming API, user interfaces, or as a tool for testing and prototyping C/C++ software. In this work, this technology is used as base technique to couple independent fluid and solid solvers, perform the synchronization of single-physics computations and exchange the interface data. The coupling mechanism is schematically represented in Fig. 5.3. In order to call the related coupling functionalities of each code, CUPyDO directly interacts with them through their wrappers as if they were simple Python objects. Inter-communication does not require any file I/O and the wrapped functionalities of each solver can thus be easily and intuitively managed in Python while the critical and computationally intensive calculations are performed under their own language. It should be emphasized that the solvers are not driven through basic OS system calls, although CUPyDO is flexible enough to technically permit such kind of basic interfacing. The coupling mechanism of CUPyDO is thus based on two principal aspects:



```
#CouplingEnvironment.py

import fluidSolver
import solidSolver


fluidSolver.computeOneTimeStep()
interfaceData = fluidSolver.getData()
solidSolver.setData(interfaceData)
solidSolver.computeOneTimeStep()
#etc
```

Figure 5.3: Schematic illustration of a coupling environment and its interaction with the respective fluid and solid Python wrappers.

1. the coupled codes are directly integrated into the coupling framework as computational objects or modules and,

2. the Python environment allows the manipulation of these objects and the coupling tasks in a very intuitive fashion.

This use of a Python framework as a coupling environment allows users and developers to reach the largest level of flexibility for performing top-level coupling tasks such as solver management and data exchange. First, the exchange data can be expressed under friendly Python-oriented formats such as lists or dictionaries, or in NumPy arrays for larger data sets on which algebraic operations have to be performed. This is allowed by the aforementioned type mapping capability of SWIG. Secondly, using a Python wrapping methodology is less intrusive than compiling the coupled code with an external API adapter coming from the coupler. Thirdly, the Python wrapper can be generated as a generic interfacing layer without being restricted to FSI coupling purposes. For instance, the solver wrapper can be used in a standalone fashion in order to define user-customized boundary conditions that thus do not require to be hard-coded in the source code. Finally, the coupling with commercial codes is technically conceivable since several of them, such as Abaqus [211] for instance, are already designed with a Python interface.

From a standard (i.e. not advanced) usage point of view, the coupling using Python does not require strong knowledge of the language since high-level functionalities are directly provided to the user by CUPyDO. However, one advantage of this approach is also to permit full customization of the coupling environment for more advanced users. The use of the Python language is then a reasonable compromise between efficiency and flexibility.

5.2.3 Development of the SU2 Python wrapper

The fluid solver SU2 used in this work has been described in Section 3.6. Under its standalone form, the source code is not designed to be coupled with CUPyDO and a Python wrapper has been developed for this purpose. The development of the SU2 Python wrapper is part of this thesis and is described in the following.

The SU2 Python wrapper is designed based on a driver object, as introduced in Section 5.2.1. A complete description of the structure of SU2 is beyond the scope of this work and many details can be found in some references such as Economon *et al.* [168] and Palacios *et al.* [165]. The general C++ class structure of the code is divided into several main classes:

- `CConfig` reads and store the problem configuration from input file,
- `COutput` manages the solution output of the simulation,
- `CGeometry` reads and processes the mesh input file, manages the dual grid and manages the multigrid geometry,
- `CIntegration` contains high-level loop for integrating in space and time general PDE's and also manages the multigrid acceleration,
- `CSolver` contains the solution procedure for particular PDE's (N-S mean flow, turbulence models, ...) and manages boundary conditions,
- `CNumerics` provides a wide range of discretization techniques for fluxes (convective and viscous) and source terms,
- `CIteration` manages the iterative process such as (pseudo-)time loops,
- `CVolumetricMovement` manages the mesh deformation.

In order to integrate the Python wrapper while keeping this inherent code structure, a `CDriver` class has been first created. The role of the `CDriver` is dual. On the one hand it defines high-level management of the aforementioned classes. Child classes of `CDriver` are defined to accommodate the underlying functionalities to specific applications such as general fluid flow, turbomachinery, adjoint, harmonic balance, multi-zone or multi-physics computations. On the other hand the `CDriver` provides an interface to specific functionalities and data structures defined at low level.

This is typically used to retrieve solution data defined at grid nodes or to define user-customized boundary conditions (e.g., non-stationary or non-uniform). The `CDriver` class is then exposed to Python using SWIG to create the Python wrapper, as depicted in Fig. 5.4. The advantage of the dual characteristic of the `CDriver` class is that it is used either by the `main` function that generates the SU2 executables or by a Python-based external environment such as CUPyDO. This minimizes the development overhead associated with the wrapper and maximizes code re-usability.

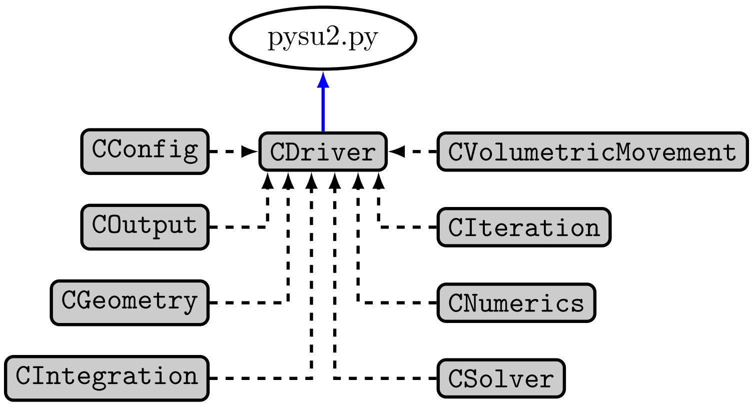


Figure 5.4: Generation of the SU2 Python wrapper based on the `CDriver` class.

The coupling routines of the SU2 Python wrapper are listed in Tab. 5.1. This gives an overview of the capabilities of SU2 for the interfacing with CUPyDO. Among these routines, some are mandatory to ensure the compatibility with the coupler. These routines constitute a common basis of functionalities that each coupled solver, and not only SU2, has to provide. Conversely, some other wrapper routines are optional and inherent to the structure of the solver code. In this section, the distinction between mandatory and optional routines is not made explicit as this will be addressed in the dedicated Section 5.5.

5.3 Architecture of CUPyDO

The Python wrapping methodology previously introduced is the foundation of the implementation of CUPyDO. Python wrapping is used to couple and interface different solvers but it is also used in the implementation of the coupler itself. In particular, CUPyDO is also composed of a low-level C++ kernel which is then wrapped and exposed to Python as high-level coupling functionalities. An overview of the object-oriented architecture of CUPyDO is depicted in Fig. 5.5, where the C++ code is represented with black boxes and the Python framework is represented with a larger grey box that includes the main Python classes of CUPyDO. The framework is also divided into three distinct layers: *Utilitiy* (U), *Core* (C) and *Interface* (I). Each part of the architecture is briefly introduced in this section, then described with more details in the next sections.

5.3.1 C++ kernel

The existence of a C++ kernel is justified by the decision of keeping any computationally intensive coupling routine at a low and more efficient language level. Typically, the C++ layer defines basic classes that handle the following tasks:

- `CInterpolator` for mesh mapping (including node searching) and mesh interpolation,
- `CInterfaceData` for management of fluid-structure interface data,

`CDriver` (constructor)	Initialize the SU2 computation (configuration, mesh, solver, ...) by instantiating the underlying classes.
`PostProcessing`	Exit the SU2 computation.
`Update`	Update the dual-time solution for next time step.
`ResetConvergence`	Reset the convergence monitoring. This is typically used when the same time step has to be restarted several times.
`Output`	Output the current solution to a file.
`DynamicMeshUpdate`	Perform a dynamic mesh deformation with grid velocity computation (ALE).
`StaticMeshUpdate`	Perform a static mesh deformation without grid velocity computation.
`BoundaryConditionsUpdate`	Process non-stationary boundary conditions and update multi-grid structure.
`Run`	Execute the solver for one time step or execute a steady (time-marched) computation.
`GetLift`	Get the lift force. Other aerodynamic forces and moments are accessible through similar functions (e.g. `GetDrag`).
`GetLiftCoeff`	Get the lift coefficient. Other aerodynamic coefficients are accessible through similar functions (e.g. `GetDragCoeff`).
`GetMovingMarker`	Get the index of the boundary patch marked as moving (moving fluid domain boundary).
`GetNumberVertices`	Get the number of vertices for a specified boundary patch.
`ComputeVertexData`	At a specified vertex, compute solution data which is not computed by the primal solution procedure of the solver. Data can be either forces or heat fluxes.
`GetVertexData`	Get data at a specified vertex. Data can be node index, coordinates, force components, temperature or heat flux components for instance.
`SetVertexData`	Set data at a specified vertex. Data can be node index, coordinates, force components, temperature or heat flux components for instance.

Table 5.1: Main coupling routines of the SU2 Python wrapper.

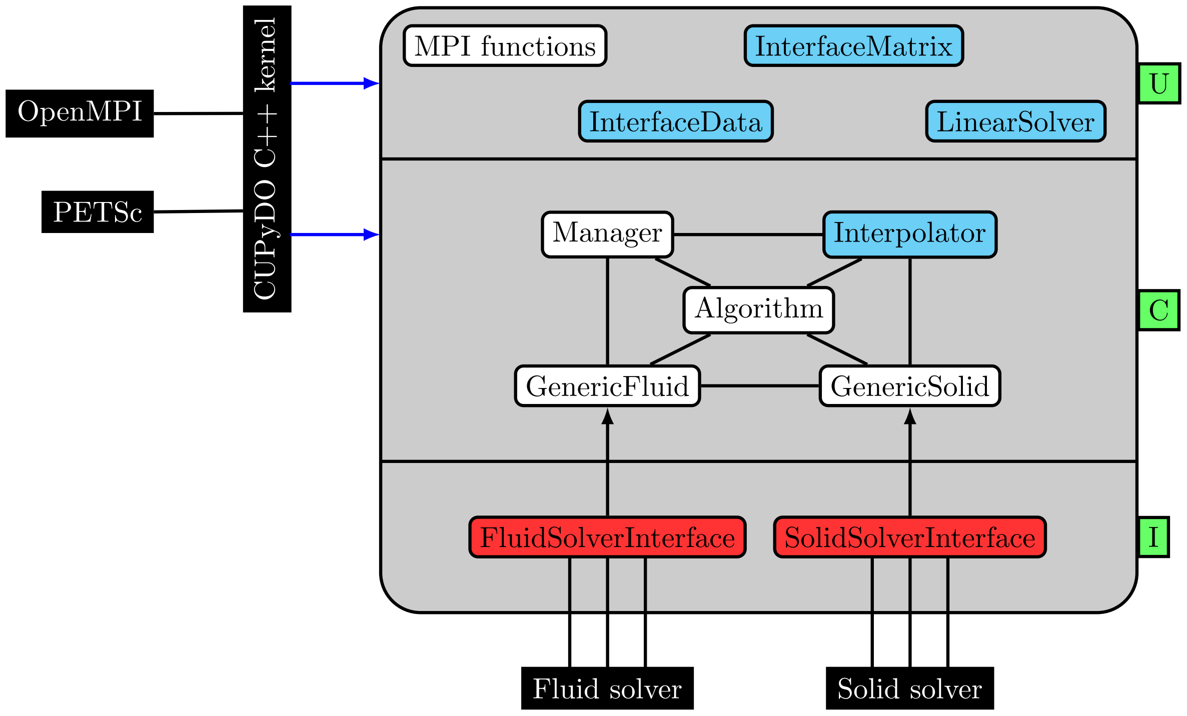


Figure 5.5: Overview of the main architecture of CUPyDO. Black boxes represent C++ code, blue boxes represent Python classes that inherit from the C++ kernel, white boxes represent Python-only classes and red boxes represent the interfacing Python layer. All these components are detailed in the next sections.

- `CInterfaceMatrix/CLinearSolver` for linear algebra.

Note that every class defined in the C++ kernel begins with the letter `C` in order to be distinguished from those defined in Python. These basic C++ classes are then wrapped and exposed to Python using SWIG.

The aforementioned tasks can be performed in parallel. Parallel HPC support is provided by well established external libraries that are OpenMPI and PETSc. These libraries are built with their corresponding built-in Python wrapper mpi4py [212] and petsc4py [213] and are thus naturally integrated in the hybrid structure of CUPyDO. A detailed description of the parallel capabilities of CUPyDO is given in Section 5.6.

As illustrated in Fig. 5.5, the classes defined and exposed from the kernel are used by some Python classes of the *Utility* and *Core* layers. These classes are coloured in light blue. The other classes are Python-only classes that do not require a computationally efficient C++ core. The Python kernel-based classes are usually built as children of the corresponding wrapper. For instance, the Python `Interpolator` class is a child class of the wrapper of the C++ `CInterpolator`. This is illustrated in Fig. 5.6 for an arbitrary class. This procedure offers the possibility to overload the low-level kernel classes to create high-level and more intuitive Python counterparts that conserve the intrinsic efficiency of the C++ language.

5.3.2 Python Utility layer

The *Utility* layer defines fundamental functionalities for CUPyDO. This includes MPI communication functions, interface data and matrix structures, as well as linear solvers.

MPI functions are directly based on the mpi4py wrapping module. The standard MPI routines for peer-to-peer and collective communications are accommodated to be used with the

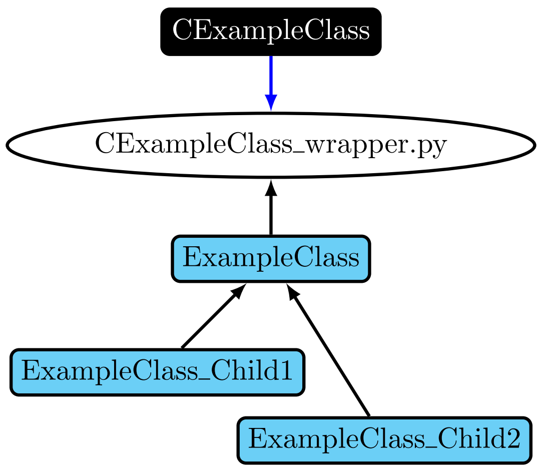


Figure 5.6: Wrapping procedure exposing a C++ kernel class to the Python framework of CUPyDO. Black box: C++ kernel class, blue arrow : SWIG wrapping action, light blue boxes: Python classes derived from the wrapper, black arrows indicate inheritance between classes and are pointing from child to mother classes.

interface data structure of CUPyDO.

The `InterfaceData` class is a child of the kernel `CInterfaceData` class which is designed to encapsulate in parallel the data being exchanged at the fluid-structure interface between the solvers. The data is stored in vector-like structures for which any standard arithmetical or algebraic operation is defined. This is supported by the PETSc library and the associated petsc4py wrapper module. The construction and the management of the interface data structure is detailed in the dedicated Section 5.4.

The `InterfaceMatrix` class is a child of the kernel `CInterfaceMatrix` class. It is designed to construct in parallel the interpolation matrix used for the mapping of non-matching interface meshes. Details about non-matching interface mesh capabilities of CUPyDO are developed in the dedicated Section 5.7.

The `LinearSolver` class is a child of the kernel `CLinearSolver` class. It is designed to solve the linear system involved in non-matching interface mesh interpolation. It is supported by the PETSc library and its wrapper petsc4py where Krylov-type iterative solvers are accommodated to be used with the `InterfaceData` and `InterfaceMatrix` structures.

5.3.3 Python Core layer

The Core layer is the central part of the coupling environment. In this part, the main classes are defined according to the different coupling tasks such as the management of the MPI partitioning and the communication network, the interpolation of the fluid-structure interface meshes, and the coupling algorithm.

The `Manager` class is designed to build the network describing the MPI partitioning of each solver. For example, this identifies the processes on which the fluid and solid solvers are running and, among all these processes, it distinguishes the subset of processes that effectively own fluid-structure interface grid nodes. Storing the number of interface nodes on each process and identifying the halo nodes[3] are also important tasks dedicated to the Manager.

The `Interpolator` class is a child of the kernel `CInterpolator` class. It uses the information built by `Manager` to construct the non-matching mesh interpolant that is called each time data has to be exchanged and interpolated from one interface grid to the other. The different schemes introduced in Section 3.3 for thermal coupling are also directly defined in the `Interpolator` class.

The `Algorithm` class is the central part of the *Core* layer. It is a Python-only base class containing the common functionalities for every particular coupling algorithm implemented in CUPyDO. In the scope of this thesis, the following coupling algorithms, that were formally described in Section 3.2, have been implemented:

- explicit weak coupling,
- block-Gauss-Seidel strong coupling with static relaxation,
- block-Gauss-Seidel strong coupling with dynamic Aitken's relaxation,

and they are used for either mechanical or thermal fluid-structure interactions. Note that for the BGS schemes, a second- or first-order time step predictor is available. Each specific coupling algorithm is implemented in a child class of `Algorithm` as illustrated in Fig. 5.7. The time-

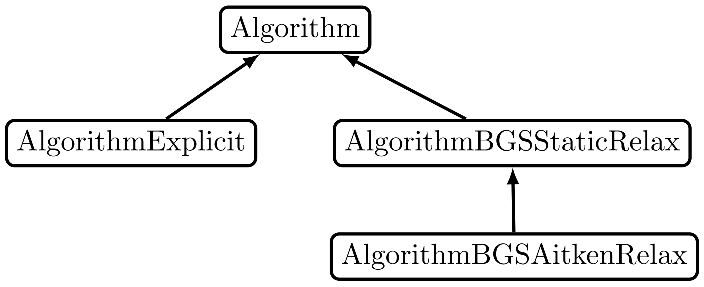


Figure 5.7: Structure of the `Algorithm` class.

marched strongly-coupled block-Gauss-Seidel algorithm such as implemented in CUPyDO is illustrated in Fig. 5.8. The corresponding high-level Python code of the inner coupling loop (BGS loop in Fig. 5.8) can be found in Appendix E for illustration. The algorithm structure strongly relies on the other classes of the *Core* layer. These are used by `Algorithm` in order to perform all the coupling tasks such as communication, mesh interpolation and sub-system computation. The object-oriented structure of CUPyDO allows the user to derive new coupling algorithms without knowing many details about inter-communication or mesh interpolation that can be used as black-box functionalities. The parallel data structure provided by the *Utility* layer allows these coupling tasks to be performed in parallel and thus avoid any serialization overhead at the coupling level when distributed solvers are used.

Remark

It should be mentioned that the IQN-ILS coupling algorithm with filtering is also implemented and available in CUPyDO by instantiating an `AlgorithmIQN_ILS` object. The detailed implementation is however not covered in this work as it is not the contribution of the present author. Details about IQN-ILS in CUPyDO can be found in the works of

[3]Halo nodes (or ghost nodes) are used at the boundaries of mesh partitions to support the communication of the solution in a distributed parallel computation.

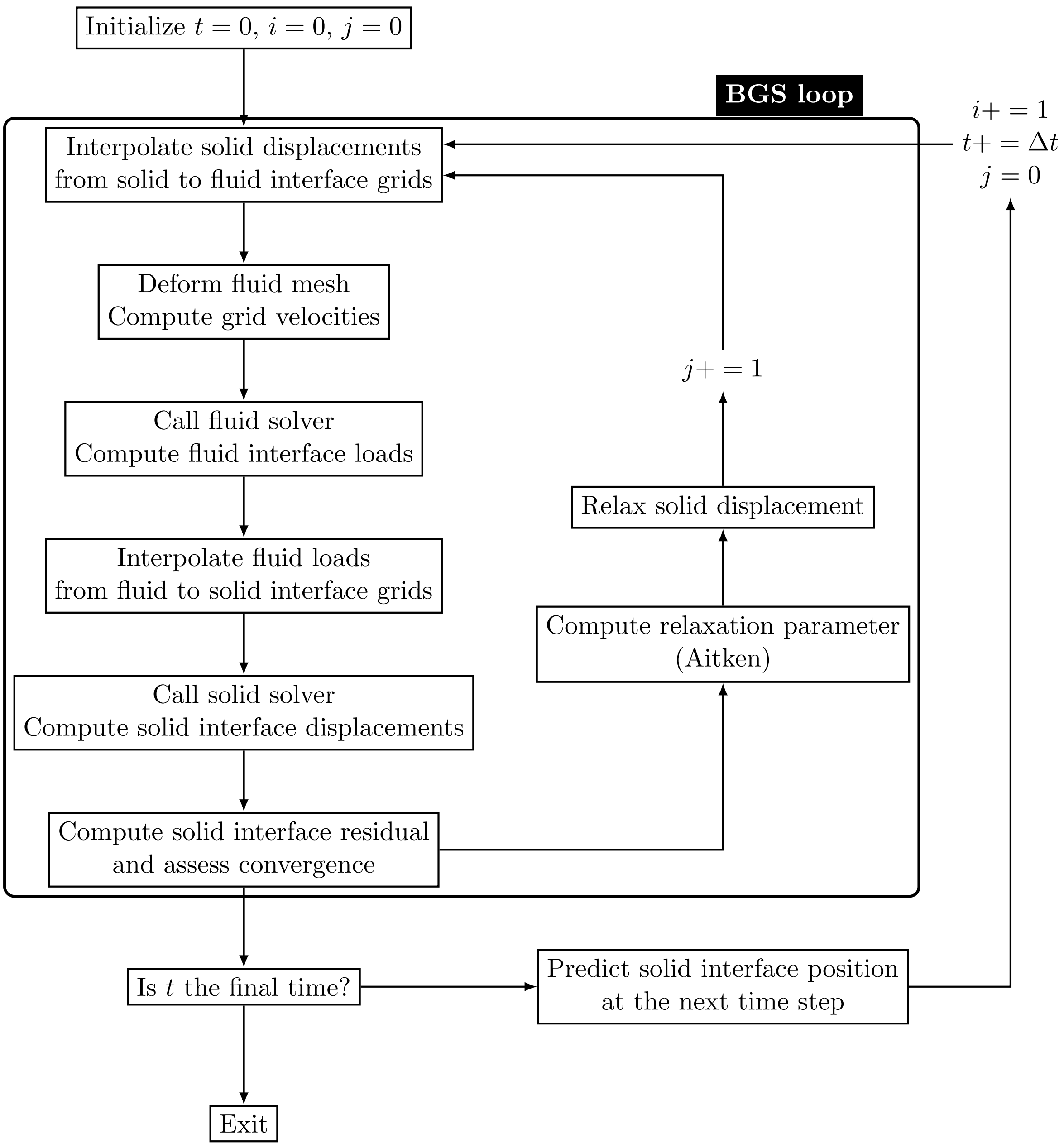

Initialize $t = 0$, $i = 0$, $j = 0$
BGS loop
Interpolate solid displacements from solid to fluid interface grids
Deform fluid mesh Compute grid velocities
Call fluid solver Compute fluid interface loads
Interpolate fluid loads from fluid to solid interface grids
Call solid solver Compute solid interface displacements
Compute solid interface residual and assess convergence
Compute relaxation parameter (Aitken)
Relax solid displacement
$j+ = 1$
$i+ = 1$
$t+ = \Delta t$
$j = 0$
Is t the final time?
Predict solid interface position at the next time step
Exit

Figure 5.8: Time-marching coupling algorithm based on the block-Gauss-Seidel scheme with Aitken's Δ^2 relaxation as implemented in CUPyDO (i is the time iterator and j is the FSI iterator).

M.L. Cerquaglia [2].

The `GenericFluid` and `GenericSolid` classes are used to represent in the *Core* layer the coupled fluid and solid solvers, respectively. They are almost pure virtual classes whose purpose is to ensure the compatibility between the interfaced solvers and the central coupling algorithm. The coupling methodology of CUPyDO and the role of the generic classes are detailed in the dedicated Section 5.5.

5.3.4 Python Interface layer

The *Interface* layer is an important part of the coupling environment since it ensures the flexibility of the coupling and the compatibility between the solvers and the coupling environment.

`SolverInterface` classes are children interfacing classes (red boxes in Fig. 5.7) that are directly derived from the generic classes of the *Core* layer. First, this inheritance ensures the compatibility between any coupled solver and the central `Algorithm` class. Then, the interfacing classes are overloaded with the specific wrapped functionalities coming from each coupled solver so that one interfacing class is required per coupled solver. The interfacing class plays the role of a plug-in layer and thus ensures the flexibility of the coupling. More details about solver interfacing are given in Section 5.5.

5.4 Interface data structure

The efficiency of a coupling environment based on a partitioned approach relies on a robust formatting of the data that is exchanged between the solvers. As this data is processed by the coupling algorithm, it has to be stored in structures on which algebraic operations can be applied. In CUPyDO, vector-like structures are defined for this purpose in the kernel `CInterfaceData` class which is exposed to Python as the `InterfaceData` class. This class is an extension of the parallel PETSc vector structure to a general multi-vector container as illustrated in Fig. 5.9. Each vector of the container is used to store data at the fluid-structure interface and

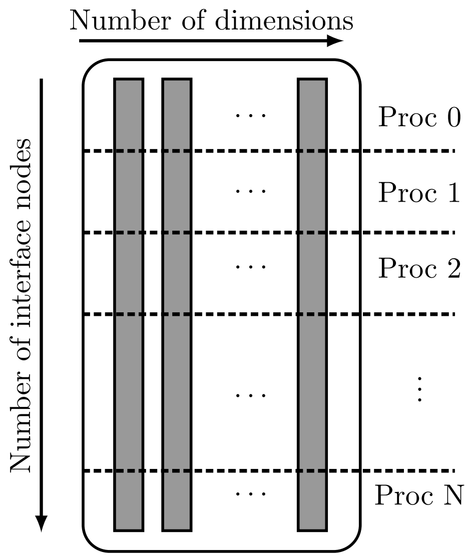


Figure 5.9: Schematic illustration of `InterfaceData` structure for storing multi-dimensional interface quantities.

its length thus corresponds to the number of nodes at the interface. The container can store as many vectors as needed in the same object. This is typically used for quantities with several dimensional components such as force or heat flux. When the coupling is performed in parallel involving distributed solvers (in a MPI sense) and consequently distributed fluid-structure interface, the `InterfaceData` container is also distributed and inherits the functionalities of the parallel PETSc vector structure. Each container is instantiated with three parameters: the number of interface nodes, the number of dimensions and the MPI communicator. The latter is the main support for inter-process data distribution. Data can be set to any vector index from any process, even if the process does not hold the index. For one particular vector index, data can be set for one particular dimension or for all the dimensions (i.e. all components) at the same time.

As already mentioned, the `InterfaceData` structure is designed to store any quantity defined at the fluid-structure interface. This can typically be coordinates, displacements, forces, heat fluxes, temperatures, or the residuals of the strongly-coupled algorithm as well. One `InterfaceData` object is instantiated per quantity on each side of the interface (fluid or solid). These objects are used for inter-solver communication, as illustrated in Fig. 5.10. By taking the

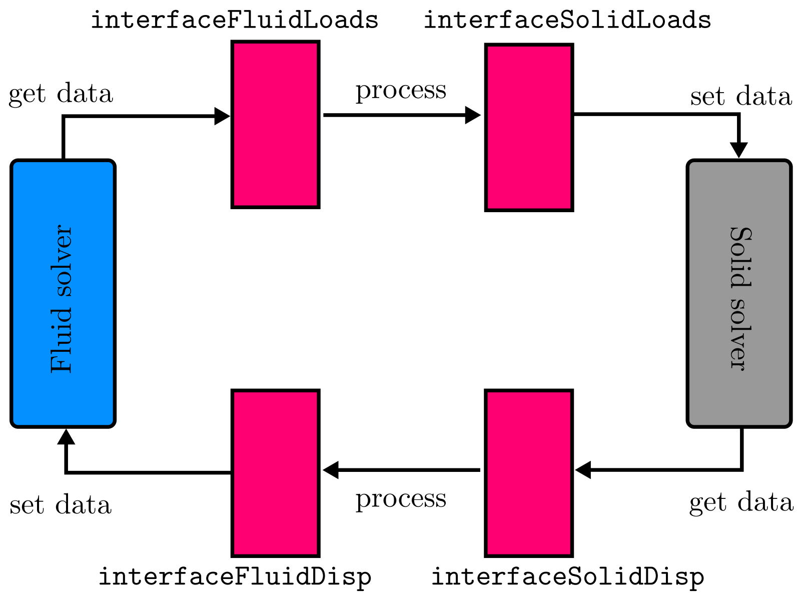


Figure 5.10: Usage of the `InterfaceData` class for inter-solver communications (example for the mechanical coupling). Each red box corresponds to one instance of `InterfaceData`.

example of the mechanical coupling, the procedure is described as follows. First, the physical interface quantities computed by the solvers (fluid loads and solid displacements) are obtained via dedicated accessors defined in the wrappers (see for example `GetVertexData` of the SU2 wrapper in Section 5.2.3). Data are then formatted into instances of `InterfaceData` and can be processed by the coupling environment. Formatting the data as vector structures allows to perform mathematical and algebraic operations (in parallel) on this data. Operators such as addition, subtraction or scalar multiplication (scaling) are thus defined in the `InterfaceData` class, as well as other vectorial operations, such as scalar product and norm computation. When they are combined with the `InterfaceMatrix` class (see Section 5.7 for more details), linear algebraic systems can be defined. A typical example of data processing is the treatment of interface mesh interpolation. After the processing step, the data is transferred to the solvers via dedicated modifiers defined in the wrappers as well (see for example `SetVertexData` in the SU2 wrapper in Section 5.2.3).

5.5 Coupling with black-box solvers and compatibility

The central idea behind the partitioned coupling approach is that the coupled solvers must be used as independent black-box tools. In CUPyDO, this is achieved by the Python wrapping methodology such as presented in Section 5.2. However, coupling flexibility cannot be achieved without a robust interfacing framework. The role of the *Utility* layer is to provide such robust and flexible interfacing with the solvers through their Python wrappers. It is a link between the coupled solvers and the central algorithmic part of the *Core* layer.

In order to preserve the high-level characteristic of the `Algorithm` structure, the coupled fluid and solid solvers are represented by unique generic base classes `GenericFluid` and `GenericSolid`, respectively. All the functionalities of these generic classes are, by construction, accommodated to the coupling algorithm. These abstract solver classes hide the intrinsic structure of each coupled solver and guarantee that no accommodation to any particular solver is introduced in the *Core* layer. This particular accommodation is instead achieved by the `SolverInterface` classes of the *Interface* layer. These specific classes are directly derived from the generic classes to inherit compatibility. They are then overloaded with the particular wrapper associated to each coupled solver, hence one interfacing class is required per coupled solver. The following code example illustrates the methodology for interfacing the SU2 solver, but the exact same philosophy can be applied with any other coupled solver, either fluid or solid:

```python
#File : SU2Interface.py

#Import SU2 Python wrapper
import pysu2
#Import generic fluid solver
from cupydo.genericSolvers import FluidSolver

#Create interface class that inherits from generic class
#Overload generic class with SU2 Python wrapper
class SU2Solver(FluidSolver):
    def __init__(self, [args]):
        (...)
        #Initialize SU2
        self.SU2 = pysu2.CFluidDriver()
        (...)

    def run(self):
        #Compute one time step
        self.SU2.run()

    (...)

    def meshUpdate(self, unsteady):
        #Perform mesh deformaiton
        if unsteady:
            self.SU2.DynamicMeshUpdate()
        else:
            self.SU2.StaticMeshUpdate()

    (...)
```

In this illustration, `__init__`, `run` and `meshUpdate` are examples of empty members defined in the generic fluid solver class and called by the coupling algorithm. They are filled with SU2 wrapper functions for accommodation so that for every generic member executed by the algorithm, the correct solver functionalities are also executed.

The additional code layer composing the interfacing classes offers direct compatibility with the core of CUPyDO as well as significant coupling flexibility. First, the individual Python wrappers of the solvers do not have to be designed while seeking direct accommodation with the coupling environment. The main reason is that, as previously mentioned, Python wrappers are usually developed for other purposes than FSI coupling. Thus, having the interfacing layer acting as a buffer relieves the design constraints imposed on the wrappers that are not required to be intrinsically FSI-oriented. Secondly, modifying the coupling environment, e.g. for maintenance or improvement, does not affect the individual Python wrappers and, reversely, any deep change in the core code of the coupled solvers has no direct impact on the central coupler.

The construction of an interfacing class is performed at a very high level, outside of any source code, hence requiring limited coding effort. Templates of interfacing classes are provided by CUPyDO to users or developers wishing to couple their solver if this solver provides a Python wrapper. The wrapper should only have minimal capabilities for controlling the solver execution and accessing the interface quantities, as listed below:

- Control the solver to compute one time step (potentially restart the same time step) or a steady-state solution with given boundary conditions,
- Provide identification of interface boundaries, e.g. with a tag or an index, and the number of vertices on these boundaries,
- Provide access to read or modify the following data at any interface vertex: indices (identifiers), coordinates, displacements, velocities, force components, temperatures and heat flux components,
- For the fluid solver only, control the mesh deformation according to the specified motion of the solid boundary.

5.6 Coupling parallelization

The coupling tool CUPyDO is designed to be HPC-compliant. Distributed simulations can be run in parallel using the MPI protocol on large computational units such as multi-core workstations or clusters. In the framework of a partitioned coupling approach, the parallelization is not straightforward and keeping a high level of flexibility while conserving good parallel scalability is challenging. When seeking high parallel performance, ensuring compatibility between the coupling tool and distributed solvers is not sufficient anymore: the coupling algorithm must also be distributed to avoid any serialization bottleneck. For fully-distributed architectures, load balancing, CPU idling and communication overhead are all factors influencing the parallel scalability and thus must be properly assessed.

The parallelization of CUPyDO is based on two types of communication between processes: intra- and inter-communications. Intra-communication refers to the communication between the processes inside one of the coupled solvers. This communication usually depends on the intrinsic parallelization of the solvers. They are used to compute the solution on partitioned single-physics domains and can be treated as black-box functionalities. This abstracted approach also allows CUPyDO to use pure serial solvers. Inter-communication refers to the communication between processes that are not inside the same solver. Such communication is typically used to exchange data in parallel between the coupled solvers and is related to the partitioning of the fluid-structure interface only. The parallel implementation of the coupler is developed to take into account the heterogeneous distribution of each coupled solver. Most of the time, the fluid domain will require many more processors than the solid domain. Consequently, different numbers of processors should be allocated for the fluid and solid solvers. The unified coupling architecture used in CUPyDO makes the management of the MPI partitioning more flexible than a server-client approach. Because CUPyDO couples the solvers as object libraries, it allows one process to host both fluid and solid solver instances, as they are never computing their solution at the same time. A coupling architecture requiring a segregated approach in which the solvers have to be hosted by distinct processes, would increase the total number of processors necessary for a simulation and the processor idling time. Idling time results from the fact that the coupling algorithm is executed in a staggered fashion: when a solver A is running, the other solver B must wait until A finishes computing the interface data required by B to be started. Fig. 5.11 illustrates the difference between segregated and integrated approaches.

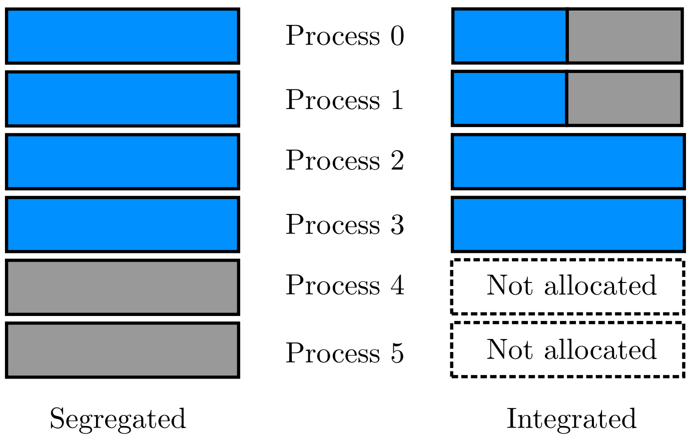


Figure 5.11: Segregated vs integrated processes solver distribution. In this example, blue and gray colors represent fluid and solid solvers, respectively.

The process distribution is based on the definition of several MPI communicators. A communicator defines a group of processes which are allowed to communicate between each other via peer-to-peer or collective communications. `MPI_COMM_WORLD` is the default communicator that owns all the processes involved in the coupled simulation. In CUPyDO it was intended that several communicators could be defined as subsets of `MPI_COMM_WORLD` to group processes involved in fluid and solid intra-communications: `FluidSolver_Comm` and `SolidSolver_Comm`, respectively. The process distribution and the number of required cores per domain is first set by the user, then the communicators are created and used to instantiate each individual solver in parallel. This also introduces an additional fundamental requirement that the coupled solvers must satisfy: they must accept to be launched on an arbitrary communicator instead of the default `MPI_COMM_WORLD`. If this cannot be fulfilled, as it is the case with the solvers used in this work, the computation can still be run in parallel but with less flexibility. Indeed, in this case the solvers are instantiated on the default `MPI_COMM_WORLD` and each of them are distributed across all the allocated processors. For example, if six cores are required by the user for the entire computation, each solver automatically uses these six cores. In other words, every process hosts both fluid and solid solver partitions. The number of cores to be used is then dictated by the size of the larger domain. This is most of the time the fluid domain, so that the solid domain might become over-distributed (e.g. it uses more processes than required for proper parallel scalability). Finally, it is important to note that CUPyDO enables coupling between parallel and serial solvers. In this case, the parallel solver is usually launched on the standard `MPI_COMM_WORLD` communicator and the serial solver is by default launched on the lowest MPI rank.

A central problem of the partitioned black-box coupling is that the coupling framework does not know a priori which solver process requires which coupling data. Consequently, an efficient communication network is created to identify which processes have to communicate between the solvers during data exchange phases. The `Manager` class of the *Core* layer is designed to build the network describing the MPI partitioning of each solver. Inspection rounds are performed in order to identify and determine the number of processes on which the fluid and/or solid solver partitions are running. Since only interface data are concerned with inter-communications, a subset of processes that effectively own fluid-structure interface nodes is also generated, as illustrated in Fig. 5.12. The identification of interface instances is typically achieved by interrogating

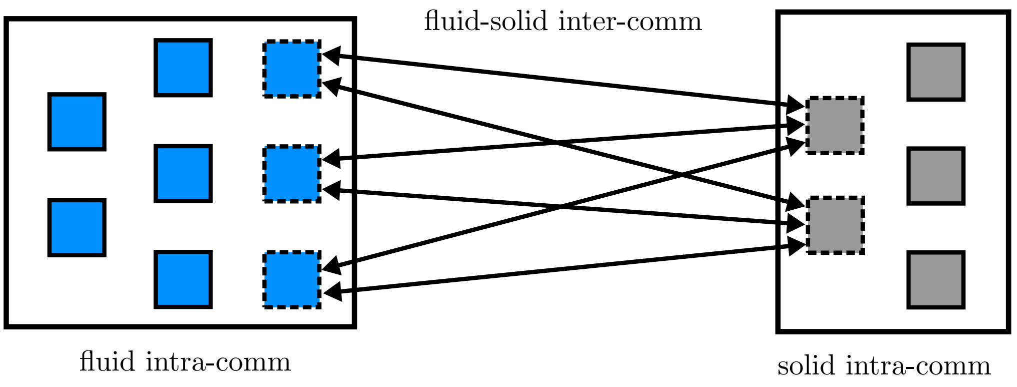


Figure 5.12: Illustration of intra- versus inter-communication. Blocks represent either a fluid (blue) or solid (grey) instance. Note that both fluid and solid instances can be hosted by the same MPI rank. Blocks with dashed boundaries represent instances that own fluid-structure interface nodes and are thus involved in the inter-communication procedure.

each solver partition, through their Python wrappers, for the number of interface nodes that are locally owned. By default this number is set to zero on partitions that do not own interface nodes. Interface node counting and halo node identification are also performed on each partition by the `Manager` during initialization. Interface halo node identification is a crucial step because halo nodes contain duplicated data that is hence not considered as physical for interface exchange. Local partitioning information is then gathered across all the processes of `MPI_COMM_WORLD` and stored under the form of Python `list()` or `dic()` entities in such a way that all information about interface partitioning and node distribution is accessible to all ranks.

The inter-communication performed between interface solver instances is supported by a dedicated communicator. This communicator must at least contain all processes owning a fluid and/or solid interface partition. In CUPyDO the default inter-communicator is the standard `MPI_COMM_WORLD` as it owns every solver instance. Defining a communicator to support inter-communications also allows the coupling tasks to be performed in a distributed fashion. This is typically the case for mesh interpolation, as it will be addressed in the next section, or other general tasks such as computing the coupling residual or performing interface relaxation and interface prediction. The parallel management of exchanged data is supported by the `InterfaceData` class as previously detailed in Section 5.6. The MPI communication is handled with Python and is supported by the mpi4py module. Communication, either peer-to-peer or collective, is embedded in high-level routines contained in the `Interpolator`, `Manager` and `Algorithm` classes as well as in the MPI functions of the *Utility* layer. Communication efficiency is also optimized through the specific data formatting used in the `InterfaceData` container class.

5.7 Interface mesh interpolation

Non-matching interface mesh interpolation is one of the most important capabilities of CUPyDO because it is the most computationally intensive part of the coupling process. Mesh interpolation is the central task of solver inter-communication. It is designed to work in parallel on distributed interface meshes while keeping a low computational overhead. The interpolation is based on Radial Basis Functions (RBF) which are implemented under the standard form (i.e. with no partition of unity localization). This method is chosen for its significant flexibility and its meshless characteristic. It is therefore the most appropriate method for coupling based on a black-box partitioned approach involving arbitrary kinds of interface grids. Finally, both conservative and fully consistent interpolations are implemented.

5.7.1 Parallel implementation of the RBF mesh interpolation

As described in Section 3.4.2, data interpolation is based on the definition of several matrices that are obtained by mapping the nodes of an interface donor mesh (subscript d in the following) onto an interface target mesh (subscript t in the following). For an interface quantity $\boldsymbol{s}$ to be interpolated from the donor to the target mesh, the mapping matrix $\widetilde{\mathbf{H}}$ is defined as:

$$\boldsymbol{s}_{\mathrm{t}} = \underbrace{\mathbf{B}\mathbf{A}^{-1}}_{\widetilde{\mathbf{H}}} \begin{bmatrix} \boldsymbol{s}_{\mathrm{d}} \\ \mathbf{0} \end{bmatrix} . \tag{5.1}$$

The matrix is constructed from two matrices $\mathbf{A}$ and $\mathbf{B}$,

$$\mathbf{A} = \begin{bmatrix} \mathbf{C}_{\mathrm{dd}} & \mathbf{P}_{\mathrm{d}} \\ \mathbf{P}_{\mathrm{d}}^{\mathrm{T}} & \mathbf{0} \end{bmatrix} \qquad \text{and} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{C}_{\mathrm{td}} & \mathbf{P}_{\mathrm{t}} \end{bmatrix} , \tag{5.2}$$

in which the blocks $\mathbf{C}$ contain the evaluation of basis functions and the blocks $\mathbf{P}$ contain the interface node coordinates (see Section 3.4.2). The basis functions currently available are the Thin Plate Spline (TPS) with global support,

$$\phi\left(||\cdot||\right) = ||\cdot||^2 \log\left(||\cdot||\right) , \tag{5.3}$$

and the Compact C2 (CPC2) with compact support of radius r,

$$\phi\left(||\cdot||\right) = \left(1 - \frac{||\cdot||}{r}\right)_+^4 \left(4\frac{||\cdot||}{r} + 1\right) . \tag{5.4}$$

These two functions are selected because they proved to provide the most accurate and robust results. In CUPyDO, interface mesh treatment is achieved in two consecutive steps: the mapping matrices $\mathbf{A}$ and $\mathbf{B}$ are first constructed during the pre-processing phase and, then, the actual interpolation is performed at every solver inter-communication.

Pre-process mapping

The construction of matrices $\mathbf{A}$ and $\mathbf{B}$ requires the evaluation of distances between interface nodes on the donor and target meshes. This is achieved only once at the beginning of the simulation where the two meshes are considered to be in a reference configuration. This means that the same mapping relation is conserved throughout the entire computation. This is valid, even for large interface displacement, as long as the relative position between nodes along the interface does not vary much.

Although there is no technical restriction to re-generate the mapping after each displacement of the interface, it would introduce a significant computational cost overhead. Each domain (fluid or solid) can be identified as the donor or the target side. A solid-to-fluid mapping is obtained when the solid side is identified as the donor side and the fluid side is identified as the target side. In order to build the reverse fluid-to-solid mapping, the procedure depends on the mapping type. For a conservative mapping, only the transposed matrix $\widetilde{\mathbf{H}}^{\mathrm{T}}$ is needed. Hence the **A** and **B** matrices are also simply transposed and no more mapping procedure is required. For a fully consistent mapping, the mapping procedure is repeated, this time with a fluid donor side and a solid target side in order to construct the matrix $\widetilde{\mathbf{G}}$ as detailed in Section 3.4.2. The interface mapping procedure currently implemented in CUPyDO is performed in parallel according to the intrinsic partitioning of the fluid-structure interface. The procedure is illustrated in Fig. 5.13 and described in the following.

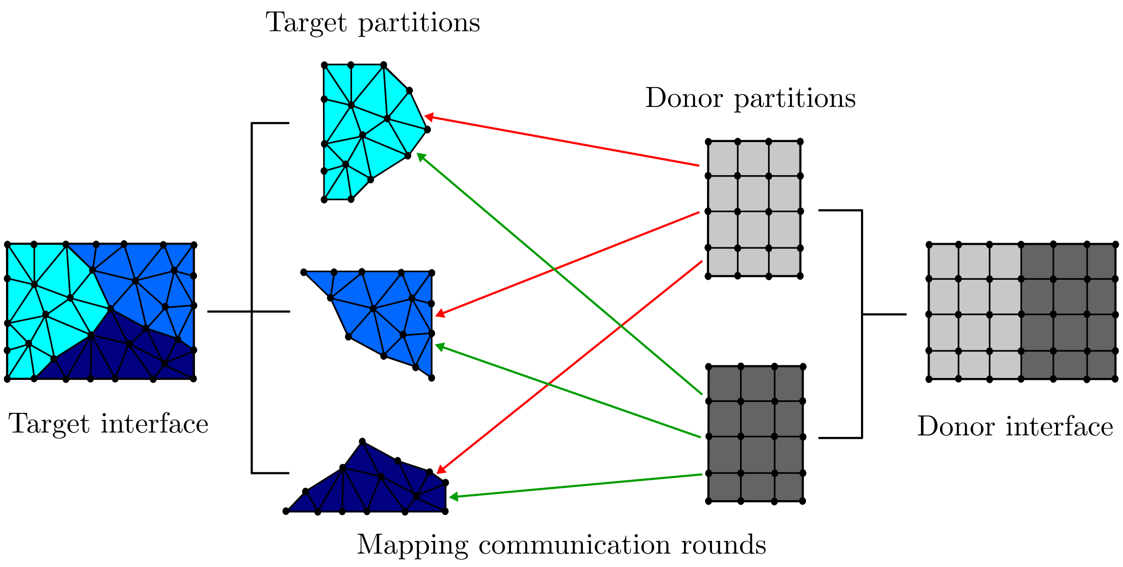


Figure 5.13: Non-matching mesh mapping for distributed interface partitions. In this simple example, two communication rounds (red and green arrows) are needed to map two solid interface partitions (in grey) with three fluid interface partitions (in blue).

The interface partitions are mapped in several communication rounds. The number of rounds corresponds to the number of interface partitions on the donor side. At each round, the corresponding donor interface partition (node coordinates) is sent to each target interface partition via peer-to-peer non-blocking communication. Using non-blocking communication allows overlapping of communication and computation. This means that the computation can be continued after a `MPI_Send` call without waiting for the corresponding `MPI_Receive` call. Once a target partition receives a donor partition, node searches and distance computations are performed to fill the corresponding elements in the blocks **C** and **P** of the matrices. When globally-supported basis functions are used, the search is performed on every node between the received donor and the local target partitions. Conversely, when locally-supported basis functions are used, the points on the received partitions are filtered to restrict the search on nodes included in the support radius, i.e. a sphere of radius r centered at the target nodes. An Alternating Digital Tree (ADT) [214] is used to perform efficient node search such as nearest neighbor or sphere neighbor search. Note that one ADT instance is locally created for each received partition. Distributed (parallel) ADT is not currently implemented in the framework of this thesis and is considered as part of future development. As an illustration of the multi-language flexibility of CUPyDO the communication rounds between the partitions are typically managed in Python using the bindings for MPI but node searches are executed in the C++ kernel to guarantee best efficiency.

The matrices **A** and **B** are defined in CUPyDO with the `InterfaceMatrix` class, which is a high-level implementation of the parallel capabilities of the PETSc matrix structure. The mapping matrices are then distributed and supported by the MPI communicator used for inter-solver communication. As mentioned in Section 5.6, this communicator is currently identified as the standard `MPI_COMM_WORLD`. Since the interface partitioning results from the intrinsic distribution of each coupled solver, heterogeneous interface node distribution is most of the time expected. In order to ensure a proper load balancing of the interface interpolation, a re-partitioning of the fluid-structure interface from a local distribution to a global distribution is performed. This re-partitioning takes place right after the aforementioned mapping procedure and is illustrated in Fig. 5.14. The matrices are globally assembled in parallel from the processes owning the

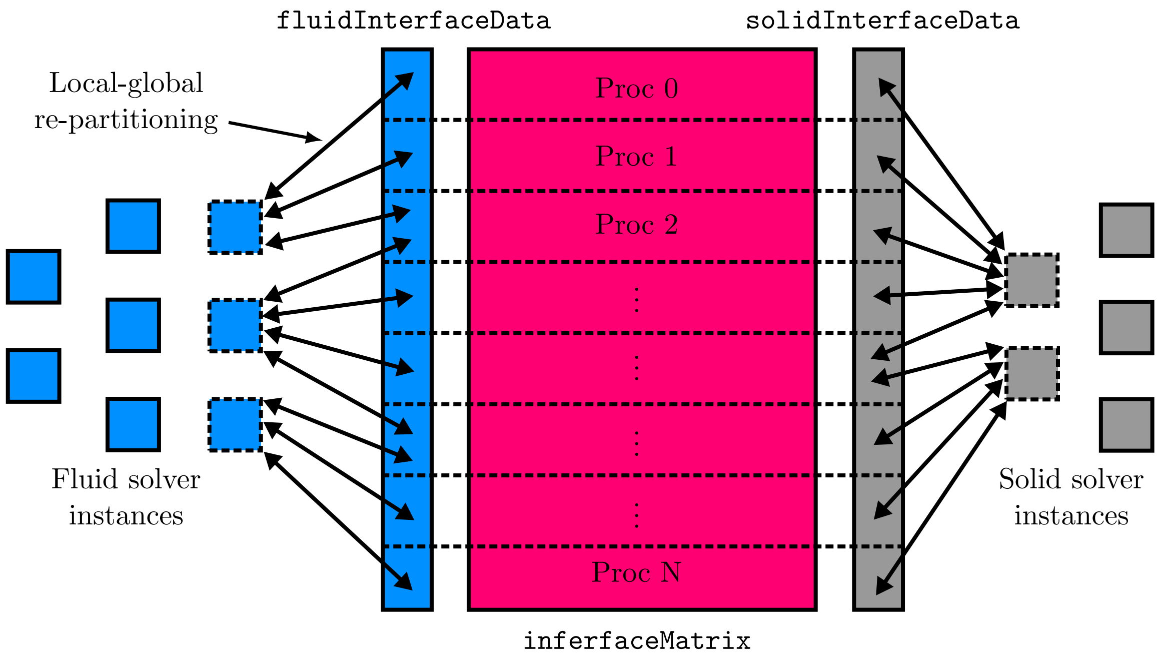


Figure 5.14: Parallel re-distribution of the fluid-structure interface for balanced mesh interpolation. Blocks with dashed boundaries represent instances that own fluid-structure interface nodes.

nodes belonging to the fluid-structure interface (boxes with dashed boundaries in Fig. 5.14), i.e. the processes that participates in the mapping procedure. This is called local-to-global re-partitioning. The complete mapping procedure can be formally described by Alg. 6.

Algorithm 6 Parallel non-mathcing mesh mapping sequence.

```
1: for all Donor interface partitions do
2:     Send interface nodes to each target partition
3:     for all Target interface partitions do
4:         Receive donor interface partition
5:         Perform local node mapping (compute elements of blocks $\mathbf{C}$ and $\mathbf{P}$)
6:     end for
7: end for
8: Perform interface re-partitioning for balanced interface node distribution
```

In the particular case of matching meshes, interpolation *per se* is not needed. However, a correspondence between fluid and solid nodes is still required since it is not expected, in general, that nodes having the same position own the same index between fluid and solid interfaces. The same mapping procedure is thus applied during the pre-processing phase, where a standard nearest neighbor search using the ADT is performed in order to pair the nodes at the same location on each side. In this case the matrix $\widetilde{\mathbf{H}}$ can be explicitly computed and features a simple boolean structure.

Inter-communication interpolation

Once the interface mapping is achieved after the pre-processing phase, interface interpolation based on relation (5.1) can be performed at any inter-communication step. In CUPyDO matrices $\mathbf{A}$ and $\mathbf{B}$ are explicitly computed, as previously described, but not matrix $\widetilde{\mathbf{H}}$. This would require the explicit inversion of $\mathbf{A}$ which might become a costly and inaccurate operation, especially for large systems. Moreover, a direct inversion of the matrix does not take advantage of its potential sparsity when locally supported basis functions are used for the interpolation. Consequently, the interpolation is performed in two sub-steps:

1. The linear system of Eq. (3.76) in Section 3.4.2, defined by matrix $\mathbf{A}$ and the donor interface data, is solved:
$$\mathbf{A}\boldsymbol{\gamma} = \begin{bmatrix} \boldsymbol{s}_\mathrm{d} \\ \mathbf{0} \end{bmatrix} \Rightarrow \boldsymbol{\gamma} = \mathbf{A}^{-1} \begin{bmatrix} \boldsymbol{s}_\mathrm{d} \\ \mathbf{0} \end{bmatrix} , \tag{5.5}$$

2. The result is multiplied by $\mathbf{B}$ to get the interpolated data on the target interface:
$$\boldsymbol{s}_\mathrm{t} = \mathbf{B}\boldsymbol{\gamma} . \tag{5.6}$$

These operations are performed in parallel based on the distributed structure of both the `InterfaceMatrix` class, used to represent the interpolation matrices, and the `InterfaceData` class, used to store the interface quantities to be exchanged and interpolated. As shown in Fig. 5.14, not only the `InterfaceMatrix` instances but also the `InterfaceData` instances are re-partitioned in the same fashion. After interpolation, the target interface data is redistributed to the respective target solver instances using reverse mapping (from global to local). Reverse mapping is performed by first gathering all data on the master MPI thread and then distributing the data with peer-to-peer communications to solver instances. A direct peer-to-peer communication strategy that avoids gathering the data is part of ongoing work. The complete inter-communication and interpolation sequence can be formally described by Alg. 7.

Algorithm 7 Mesh interpolation sequence during solvers inter-communication.

```
1: Get data from donor solver on local interface distribution
2: Assemble donor interface data in global distribution
3: Solve system Eq. (5.5)
4: Perform multiplication Eq. (5.6)
5: Redistribute target interface data to local target interface distribution
6: Set data to target solver on local interface distribution
```

The system given by Eq. (5.5) is solved in parallel using the FGMRES iterative solver of the PETSc library with a standard Jacobi preconditioning. It is managed by the `CLinearSolver` class and its Python counterpart `LinearSolver` in the *Utility* layer. It makes direct use of the available Python bindings of PETSc. A Jacobi-FGMRES approach was chosen as default because it is known to provide good convergence and proved to be robust in most cases, but any other PETSc solver could otherwise be used. Although a formal investigation of the performances of different PETSc solvers for RBF interface interpolation would represent an important contribution to the development of CUPyDO, this is not addressed in this thesis. Nevertheless, solver characterization could be envisioned as near future work.

Design of the `Interpolator` class

The aforementioned mapping and interpolation procedures are managed by the `Interpolator` class which is derived from the C++ kernel `CInterpolator` class. As already mentioned, the kernel mainly defines specific routines for efficient node search and mapping, but the high-level management is performed at the Python level. The `Interpolator` class is also a generic class whose children correspond to each type of interpolation (conservative or consistent) and to each basis function (CPC2 or TPS). The complete object-oriented structure is illustrated in Fig. 5.15 where boxes with dashed boundaries represent the children classes that can be instantiated. The base `Interpolator` class mainly defines the routines which are common to any type of

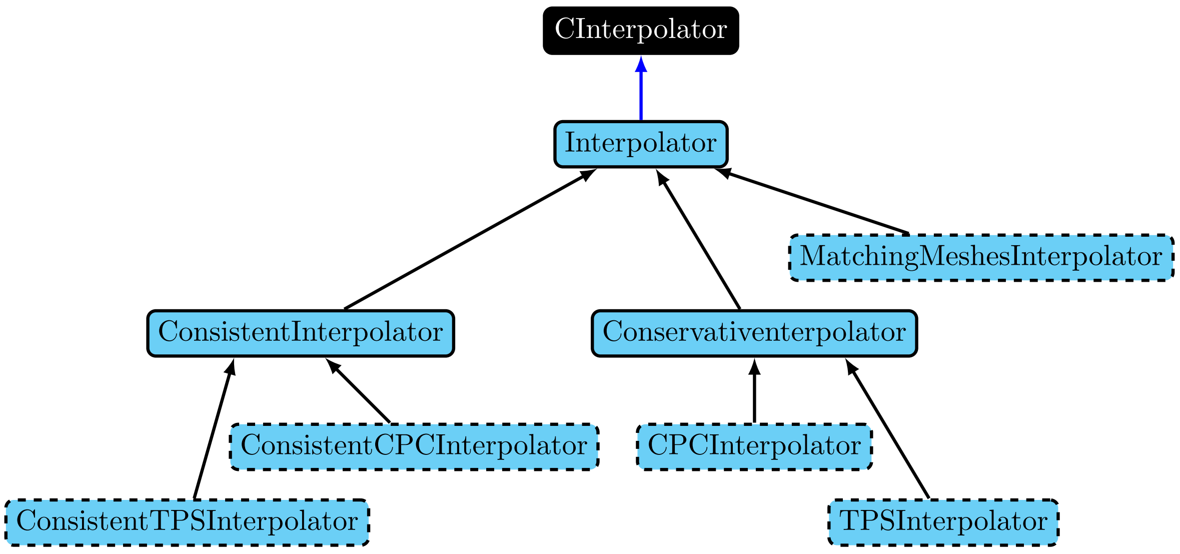


Figure 5.15: Object-oriented structure of the `Interpolator` class. Children class to be instantiated are identified with dashed boundaries.

interpolator such as those used to set/get data to/from the solvers. It also handles the repartitioning of the interface data. `ConsistentInterpolator` and `ConservativeInterpolator` are directly derived from the base class to define which particular type of mesh mapping is used for the interpolation. They both handle the parallel mesh mapping during the pre-processing phase as well as the interpolation during solver communication. The children of these two classes simply specify which basis function is in use and manage the associated node searches which are implemented in the kernel. Finally, the `MatchingMeshesInterpolator` is a special instantiable class to handle matching meshes.

The `Interpolator` class is also designed to manage the specific thermal coupling schemes, as introduced in Section 3.3, and implemented in CUPyDO. The following schemes are available: TFFB, FFTB, hFTB and hFFB. In contrast to the mechanical coupling, the way by which data is exchanged in the thermal coupling is not unique and depends on the chosen scheme. In this context, the `Interpolator` decides which type of thermal data (e.g. temperature or heat flux) is sent, interpolated and received by each solver. In the particular cases of hFTB and hFFB schemes, the `Interpolator` is also able to compute the specific Robin temperature introduced as $\hat{T}$ in Section 3.3 based on a user-defined numerical heat transfer coefficient $\tilde{h}$.

5.7.2 Verification of the RBF implementation

The interface interpolation implemented in the `Interpolator` of CUPyDO has been verified on a simple test case. A detailed study about accuracy of RBF methods to interpolate an analytical function over a 1D interface can be found in the works of de Boer *et al.* [105,107]. In particular, they concluded that using the globally-supported TPS basis function typically gives a more accurate interpolation (i.e. it gives a smaller interpolation error) than using the CPC2 basis function.

In this section, the purpose is to verify the implementation of the RBF interpolation in CUPyDO without considering the conservative and consistent aspects that will be addressed further in the next chapter of this thesis. The verification is thus limited to the interpolation of a two-dimensional displacement field from a source interface mesh to a target interface mesh. The chosen shape for the interface is a beam with dimension (length $\times$ width) $L \times l = 0.5 \times 0.04$ m^2. The interface is discretized using four distinct meshes with an increasing number of points along the length and the width:

- mesh A : 12×3,
- mesh B : 25×3,
- mesh C : 50×5,
- mesh D : 100×10.

Three displacement fields are used for the verification: a translation with arbitrary components (0.3, 0.1) (interpolation of a uniform field), a rotation of 90 degrees and a displacement resulting from the bending of the beam clamped at one of its extremity. In the latter case, the analytical expression of the displacement field,

$$\begin{aligned} d_x &= \frac{P}{6EI}\left(3\nu x^2(L-y) + (4+5\nu)l^2\frac{y}{4} + (3L-y)y^2\right) , \\ d_y &= \frac{-Px}{6EI}\left((6L-3y)y + (2+\nu)(x^2 - \frac{l^2}{4})\right) , \end{aligned} \tag{5.7}$$

is taken from Augarde *et al.* [215], where $E = 1906651$ Pa, $I = 6.66 \cdot 10^{-8}$ m^4, and $\nu = 0.4$ are the arbitrary Young modulus, cross-section inertia and Poisson coefficient, respectively. The fictitious load P corresponds to a distributed load on the free tip of the beam, with arbitrary value $P = 0.8$ N/m.

For each displacement case, several combinations of source-target mesh discretizations are assessed. The results are summarized in Tab. 5.2. Each combination of interface discretization

	Source	Target
From coarse to fine interface	12×3	12×3
	12×3	25×3
	12×3	50×5
	12×3	100×10
From fine to coarse interface	100×10	12×3
	100×10	25×3
	100×10	50×5
	100×10	100×10

Table 5.2: Summary of the different interface discretization combinations used during the verification tests.

is evaluated using either the TPS or CPC2 basis function (2×8 combinations). In the later case, the radius is chosen to be $r = 1/2\,L$. The impact of the CPC2 radius is tested as well with the values $r = \{0.25,\, 0.5,\, 0.75,\, 1\} \times L$ on a limited set of interface discretizations: 12×3 to be interpolated onto 100×10 and 100×10 to be interpolated onto 12×3 (2 mesh combinations $\times$ 4 values of radius). The total number of tests per displacement case is then equal to 24. For each test, the error is measured and defined as a comparison between the interpolated and exact displacements of the target interface:

$$e = \frac{\sqrt{\sum\limits_{i=1}^{N} ||\Delta \boldsymbol{d}_i||^2}}{\sqrt{\sum\limits_{i=1}^{N} ||\boldsymbol{d}_i^{\text{ex}}||^2}}, \tag{5.8}$$

where N is the number of points on the target interface and $\Delta \boldsymbol{d} = \boldsymbol{d}^{\text{int}} - \boldsymbol{d}^{\text{ex}}$ is the difference between interpolated and exact displacement of each interface. Finally, note that the tolerance of the iterative solver used for the interpolation is set to 10^{-10} so that any error of the same order or below is considered as marginal and negligible. The detailed results (measured errors) of the tests can be found in Appendix F.

The pure translation case is typically used to guarantee that the implemented interpolation is consistent (exact recovery of a uniform field). For pure translation, the maximum measured error for all the 24 cases is $e = 2.56 \cdot 10^{-10}$ which is considered as marginal. Therefore, the consistency of the interpolation is verified.

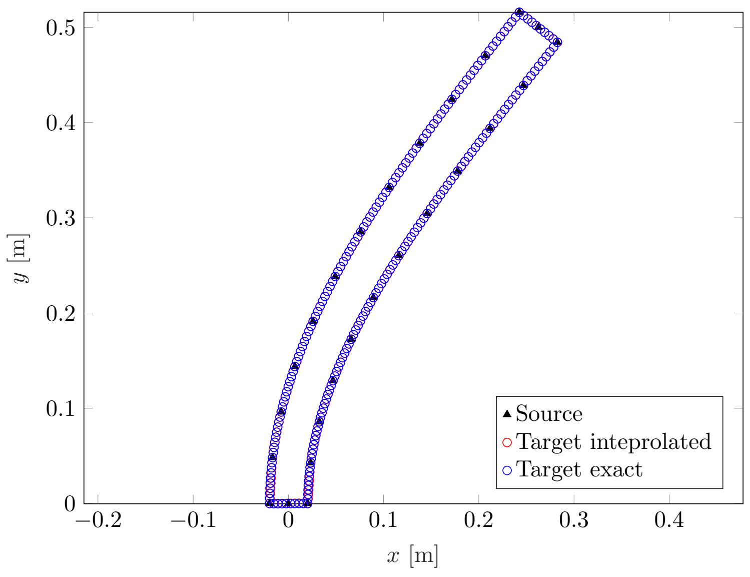
When applying a pure rigid body rotation, the maximum measured error for all the 24 cases is $e = 4.9 \cdot 10^{-7}$. The maximum error is obtained when interpolating the displacement from the finer mesh to the coarsest mesh using CPC2 with the lowest radius (worst tested case). Although the error cannot be considered as negligible, it remains far below (6 orders of magnitude) the maximum local displacement due to rotation. It is then concluded that the interpolation is extremely accurate for a pure rigid body motion.

The analysis of the errors measured for the case of the bending leads to the following general conclusion:

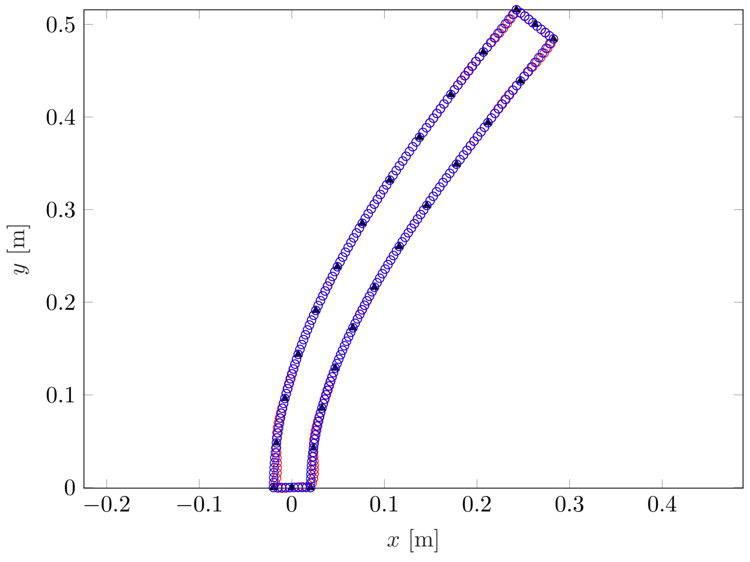
- The error usually increases when the discrepancy in the discretization of the interface between source and target increases.
- When interpolating on matching interfaces, the error has always the same order of magnitude as the linear solver tolerance.
- The error is usually smaller when interpolating from a fine to a coarse interface than interpolating from a coarse to a fine interface. In practice for FSI, this means that one could expect a more accurate interpolation of the fluid loads onto the structural mesh compared to the interpolation of the structural displacement onto the fluid mesh.
- The TPS basis function is typically more accurate (smaller error) than the CPC2 basis function. This is illustrated in Fig. 5.16 comparing the interpolated displacement field from the coarsest to the finest mesh obtained with TPS and CPC2 basis functions. The worst case of intepolation using CPC2 (Fig. 5.16 (b)) is intentionally chosen to highlight to major discrepancies located near the root and the tip of the beam. This reveals the effect of interpolating a field on a discontinuous geometrical support (presence of corners in this case).
- When using CPC2, the error decreases as the radius increases. This is illustrated in Fig. 5.17 for two coarse-fine mesh interface combinations.

5.8 Setting a fluid-structure computation with CUPyDO

Within the context of a partitioned approach, setting a coupled fluid-structure simulation with CUPyDO starts with configuring the coupled solvers independently. Typically, mesh, numerical



(a) TPS.



(b) CPC2 with $r = 0.25L$ (worst case). The largest discrepancy is observed at the root and the tip of the beam.

Figure 5.16: Interpolation of the displacement field (bending case) from the 12×3 mesh to the 100×10 mesh using two different basis functions.

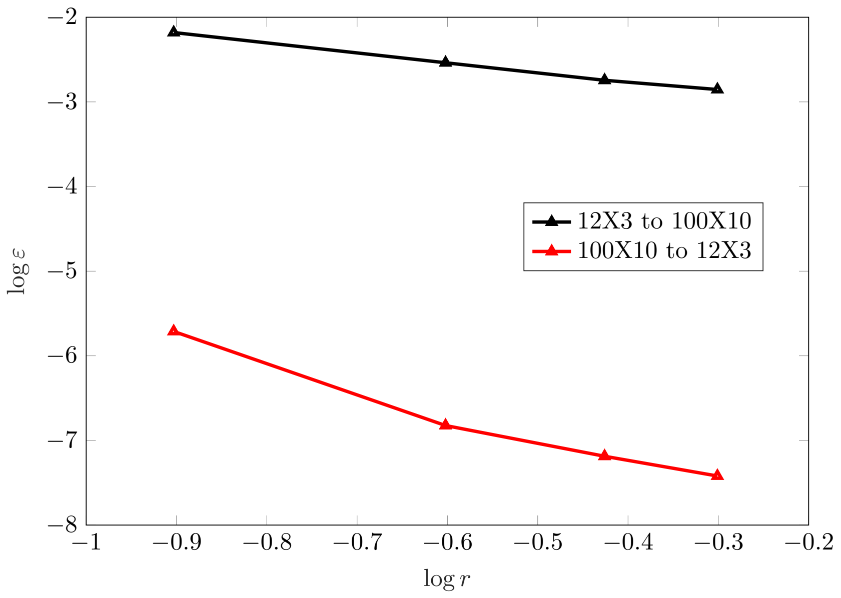


Figure 5.17: Interpolation error as a function of the radius for a CPC2 basis function used on two mesh combinations.

and physical parameters as well as initial and boundary conditions (except at the fluid-structure interface boundary) are set for each solver by using their intrinsic configuration system (i.e. configuration file). The coupling environment is then configured with a Python script that directly instantiates some of the classes described in the previous sections. The purpose of this section is to give an insight into how the modular structure of the coupling environment can be exploited to run a complete simulation. This is illustrated in the case of an unsteady strongly-coupled and mechanical only simulation using the block-Gauss-Seidel method with dynamic Aitken's Δ^2 relaxation. The interface meshes are supposed to be non-matching and the CPC2 conservative interpolation is used. As basic preliminaries to any computation, it is obviously supposed that the coupled solvers are properly set and installed on the computing unit, that they are compiled with their respective Python wrapper and finally that an interfacing module has been implemented in CUPyDO.

Step 1 : import the different modules of CUPyDO The different classes and functionalities of the coupling tool are grouped into distinct modules. Each module is associated with a major functionality such as management, interpolation or solver coupling. Importing these modules is equivalent to associating CUPyDO with the computation and can be performed as follows:

```python
#Import common utilities
import cupydo.utilities as cupyutil
#Import manager module
import cupydo.manager as cupyman
#Import interpolation module
import cupydo.interpolator as cupyinterp
#Import coupling algorithm
import cupydo.algorithm as cupyalgo
```

Step 2 : define coupling parameters A small set of coupling parameters have to be defined by the user to control the simulation and the coupling scheme. The parameters can be directly defined in the launching script as variables that will be passed as arguments to the different objects. These parameters are listed and detailed in the following (with arbitrary values):

```python
#Number of dimensions
nDim = 3
#FSI Tolerance for strong coupling
tolFSI = 1e-6
#Maximum number of coupling iterations
nFSIIterMax = 6
#Relaxation parameter
omega = 0.8
#RBF radius [m]
RBFRadius = 0.1
#Time evolution type
timeType = 'unsteady' #other option 'steady'
#Time step size [s]
timeStep = 0.001
#Physical simulation time [s]
timeTot = 2.0
```

The parameter `omega` may correspond either to the fixed relaxation parameter in case of static relaxation or to the parameter $\bar{\omega}$ used for dynamic relaxation and defined by Eq. (3.23) (the **max** criterion is chosen as default). The value of `omega` only depends on the selected type of coupling algorithm, as detailed further below. Also, it is important to note that the coupling at each time step is controlled either by the coupling tolerance or by a maximum allowed number of coupling iterations. This prevents time steps, where convergence cannot be reached but which are not diverging, from freezing the time-advancement. Note that these parameters are case-dependent and their optimal values are usually not known *a priori*.

Step 3 : initialize MPI and the parallel distribution of the solvers Prior to any computation, MPI has to be initialized (only for parallel simulations). The list of process ranks is specified for each coupled solver and is used to define the intra-communicator as subset of the general `MPI_COMM_WOLRD` communicator. As previously mentioned, this communicator is also used to parallelize the coupling tasks. In the following example, the fluid solver is instantiated on all the processors allocated to the computation while the solid solver is instantiated on two processors:

```python
#Initialize MPI
from mpi4py import MPI
worldComm = MPI.COMM_WORLD
worldGroup = worldComm.Get_group()
nProc = worldComm.Get_size()
#Allocate processors to fluid and solid solvers
fluidAllocRank = range(0, nProc)
solidAllocRank = [0, 1]
#Define fluid and solid communicators
fluidGroup = MPI.Group.Incl(worldGroup, fluidAllocRank)
solidGroup = MPI.Group.Incl(worldGroup, solidAllocRank)
fluidComm = comm.Create(fluidGroup)
```

```
solidComm = comm.Create(solidGroup)
```

Note that for a purely serial computation, an explicit definition of communicators is still required. In this case, they are simply set to `None`.

Remark
The methodology for defining separate communicators as described here above has been introduced as it is intended to be used with CUPyDO. However, it is important to note that it has neither been deployed nor formally tested because the available coupled solvers could not be launched on other communicators than the default `MPI_COMM_WOLRD`. As previously mentioned, parallel simulations are still operational using the default communicator for each coupled solver.

Step 4 : initialize the coupled solvers Each solver is individually initialized with its own set of parameters, typically mesh and configuration files. The initialization for coupling with CUPyDO is directly handled by the corresponding interface modules of the *Interface* layer. Solver initialization is typically managed by the constructor of the interfacing class and thus includes mesh reading and partitioning as well as any other pre-processing task such as setting up the numerical methods and integration schemes. After this phase, the solvers are supposed to be set and ready to compute their own solution. The following example illustrates the initialization process for arbitrary fluid and solid solvers:

```
#Initialize fluid solver
import cupydoInterfaces.fluidSolverInterface
#If the process owns a fluid solver instance, instantiate ...
    one solver partition
if fluidComm:
     fluidSolver = ...
    fluidSolverInterface.fluidSolverConstructor(fluidComm, ...
    [fluid parameters])

#Initialize solid solver
import cupydoInterfaces.solidSolverInterface
#If the process owns a solid solver instance, instantiate ...
    one solver partition
if solidComm:
     solidSolver = ...
    solidSolverInterface.solidSolverConstructor(solidComm, ...
    [solid parameters])
```

In practice, the interface classes are named according to each compatible solver, e.g. `SU2Interface` for SU2 or `MtfInterface` for Metafor.

Step 5 : instantiate the coupling manager Once the solvers are instantiated, they are able to communicate with the coupling environment. Then the coupling manager is instantiated in order to set the distributed communication network and to gather information about fluid-structure interface node distribution. The manager is called with the following line:

```
manager = cupyman.Manager(fluidSolver, solidSolver, nDim, ...
    timeType, worldComm, fluidComm, solidComm)
```

The manager mainly takes as arguments the solver instances as well as the communicators which are used to build the network. Activation of the thermal coupling (default is mechanical) is also achieved by setting the `thermal` boolean attribute to `True`:

```
manager.thermal = true
```

Step 6 : instantiate the interpolator By instantiating the interpolator, the mesh mapping and mapping matrices assembly are automatically launched as they are directly controlled by the constructor. In the following example, a conservative mapping method with CPC2 basis function is selected:

```
interpolator = cupyinterp.CPCInterpolator(manager, ...
    fluidSolver, solidSolver, RBFradius, worldComm, ...
    chtTransferMethod=None, heatTransferCoeff=1.0)
```

Because the interpolator relies extensively on inter-communication processes, it requires access to the manager and solver instances, that are passed as arguments. As a locally supported basis function is considered here, the value of the radius needs also to be specified. When thermal coupling is activated, the scheme is defined with the `chtTransferMethod` parameter (e.g. TFFB, hFTB, ...). In the present example, this parameter is set to `None` because only a mechanical coupling is considered. The parameter `heatTransferCoeff` is used to define the value of the numerical heat transfer coefficient, if applicable.

Step 7 : instantiate the coupling algorithm and launch the coupled simulation The final step consists in instantiating the coupling algorithm and then launching the complete simulation. This is achieved with the following code (block-Gauss-Seidel with dynamic Aitken's relaxation):

```
algorithm = cupyalgo.AlgorithmBGSAitkenRelax(manager, ...
    fluidSolver, solidSolver, interpolator, nFSIIterMax, ...
    timeStep, timeTot, omega, worldComm)
algorithm.run()
```

The algorithm instance depends on all other modules and also requires simulation control parameters as arguments. The coupled simulation is launched by simply calling the `run()` member which handles both the time-advancement (solver synchronization) and the coupling (solver communication). During the simulation, each solver independently manages its solution output, either standard or file outputs. Coupling information monitoring such as interpolation residual, coupling residual or computed relaxation parameter is also provided by CUPyDO at runtime. Coupling history is stored in a ASCII file where the number of coupling iterations, the value of the coupling residual and the dynamic relaxation parameter are written for each time step and/or for each coupling iteration.

Summary of chapter 5

This chapter has described in details the implementation of CUPyDO. First, some design strategies have been introduced so as to ensure fundamental requirements such as flexibility, high-level usability and maintainability of the code to be developed. Those strategies are supported by the technological corner stones on which CUPyDO is based, such as typically its Oriented-Object structure and the use of the Python-wrapping technique.

The Python-wrapping procedure has been illustrated. It has been shown, with simple examples, how C++ classes can be exposed to Python with minimal effort using the SWIG tool. The resulting Python layer consists in a module from which the C++ wrapped objects can be used in a very intuitive way, while being still executed under their own compiled language, providing the best efficiency. For complex codes, two manners of generating an equivalent Python wrapper have been presented. A wrapper layer can be generated on top of each class (or group of classes), or a core language driver layer can be first designed, which is then wrapped into Python.

The coupling methodology of CUPyDO using the Python wrapping has been presented. In order to call the related coupling functionalities of each code, CUPyDO directly interacts with them through their wrappers as if they were simple Python objects. This provides a high-level, flexible and intuitive management of the coupling tasks, while the computationnally intensive calculations are still performed in the compiled core part of the coupled solvers. Using a coupling Python layer provides several advantages such as a friendly data formatting (typically for inter-solvers communications), less intrusive code for the coupled solvers and the fact that the wrapper can still be designed for other general purpose than FSI coupling.

The development of the Python wrapper of the fluid solver SU2 has been presented. The wrapper is based on interfacing a C++ driver layer that gathers the functionalities of the other main C++ classes composing the solver code. The role of the SU2 driver layer is dual. On the one hand, it accommodates the low-level functionalities to a specific type of applications (e.g. external flow, turbomachinery flow, adjoint computation, harmonic balance, etc) and on the other hand it directly exposes low-level functionalities and data structure, which are typically used for FSI coupling, through the Python wrapper. The advantage of the dual characteristic of the SU2 driver layer is that it produces code that can be used by the core executable of the solver or by a Python-based external environment. This maximizes code re-usability.

The general architecture of CUPyDO has been described. It is composed of a C++ kernel and several Python layers. The C++ kernel gathers all the computatonnally intensive coupling tasks such as mesh interpolation or interface data management. The functionalities of the C++ kernel are then exposed to Python with the same wrapping procedure as for the coupled solvers. It is also used to link CUPyDO with external libraries such as OpenMPI or PETSc, typically for HPC support. The Python *Utility* layer defines fundamental functionalities, including MPI exchange functions, interface data and matrix structures, and linear solver definitions. The Python *Core* layer is the central part of the environment as it provides the high-level management of, for instance, the interface mesh interpolation and the coupling algorithms. The object-oriented structure of CUPyDO also allows the user to define new coupling algorithms with limited effort. The last Python *Interface* layer ensures the flexibility of the coupling by accommodating the wrapper of each coupled solver to the central algorithmic part of CUPyDO.

The interface data structure management of the *Utility* layer has been detailed. These structures are the main support for inter-communication as they are filled with data exposed by each coupled solver. Because they are manipulated by the coupling algorithm, interface data are stored into structures on which algebraic operations can be applied. Interface data are thus stored in vector-like structures, that are supported by the PETSc library if a MPI partitioning of the interface is applied.

The coupling mechanism of the *Interface* layer has also been detailed. In order to ensure compatibility between the core of CUPyDO and the coupled solvers, the object-oriented inheritance mechanism is used between the generic solver classes of the *Core* layer and the individual interfacing modules of the *Interface* layer. These inherited interfacing modules are filled with the wrappers of the associated coupled solvers, and act as buffers that reduce the design constraints on the wrappers and leverage modularity. Some minimal functionalities that coupled wrappers should satisfy have also been listed.

The management of parallel coupled computations with CUPyDO has been described. Assuming that the problem is partitioned with MPI, two types of communication between processes have been considered. While intra-communication is seen as black-box as they involve data exchange within the same coupled solver, inter-communication is used to exchange fluid-structure interface data between processes of distinct coupled solvers. The heterogeneous partitioning of each solver is taken into account and distinct MPI communicators for each coupled solver and for the interface can be defined as a subset of the general `MPI_COMM_WORLD`. The distribution of processes is based on an integrated approach, in which one process can host both a fluid and solid solver instance, as opposed to the segregated approach. The `Manager` class is specifically designed to build the network describing the MPI partitioning of each solver and to identify which processes are involved in inter-communication.

The implementation of the non-matching interface mesh interpolation of CUPyDO has been detailed. Interface interpolation is performed using the Radial Basis Function method. Two basis functions are used: the Thin Plate Spline (TPS) and the Compact C2 (CPC2). Non-matching interpolation is performed in two steps. In the first step, fluid and solid interface nodes are paired and mapped to construct the interpolation matrices. For parallel computation, the mapping is achieved through successive communication rounds from the partitions of the donor side to the partitions of the target side. The parallel data structure for matrices and vectors is supported by the PETSc library. Also, local-to-global mapping is used to re-balance the node distribution on the interface. In the second step, the associated linear system is solved at every inter-communication phase to achieve data interpolation from one interface to the other. The built-in FGMRES solver of PETSc is used to that purpose. The `Interpolator` class is the central class that implements the RBF interpolation. The object-oriented structure of CUPyDO is used to derive conservative and consistent interpolators. The different thermal coupling schemes are implemented in this class as well.

Finally, the detailed procedure for setting up a coupled FSI simulation with CUPyDO has been described. First, the coupled solvers must be individually configured by using their own configuration system. Then, the coupling environment is configured with a Python script that directly instantiates the main classes. The Python script can be built by following several steps. First, the different components of CUPyDO must be imported. Then, the coupling parameters can be defined, which is followed by the initialization of MPI (if applicable) and the definition of the parallel distribution. At that point, the coupled solvers are instantiated by calling their interfacing modules, and the coupling network is created by the `Manager`. Finally, the interface interpolator and the coupling algorithm are set. The coupled simulation is launched by calling the `run` function of the coupling algorithm.

Part III

Verification of the CUPyDO coupling environment and application to an aeroelastic case study

Chapter 6

FSI verification test cases for CUPyDO

This chapter is dedicated to the verification and validation of the CUPyDO coupling environment for both mechanical and thermal fluid-structure interaction applications. Referenced test cases are used to cover specific phenomena, such as VIV or flutter, that involve different fluid and flow conditions (e.g. subsonic vs transonic flows). Thermal applications with CHT are also considered. Each test case is set up to highlight one or several aspects of the coupling such as data communication, coupling synchronization, coupling relaxation, coupling versatility of the Python wrapping or interpolation of non-matching interface meshes. The central idea is to demonstrate that CUPyDO has all the required capabilities to produce accurate results for a wide range of physics and numerical models.

The first two test cases involve the coupling between the SU2 fluid solver and simplified rigid body models. A one degree-of-freedom cylinder immersed in a cross-flow is used to verify the accuracy of the strong coupling in simulating VIV and predicting the lock-in region. An airfoil with pitch and plunge degrees of freedom is then used to determine if CUPyDO is accurately predicting the aeroelastic flutter point. In the next test case, the complexity of the structural model is increased and a standard FEM model is used to predict the response of a beam attached in the wake of a rigid square cylinder immersed in a cross-flow. The substitution of the rigid body solver by Metafor demonstrates the modularity of the coupling environment. The efficiency and the robustness of Aitken relaxation is also evaluated. The AGARD 445.6 wing is used as the first three-dimensional aeroelastic test case. It also involves non-matching interface meshes, which are used to assess the interpolator on a realistic geometry. The last test case is a heated hollow cylinder immersed in a cross-flow. It is used to illustrate CHT applications and to verify the implementation of the thermal coupling schemes in CUPyDO.

Although the parallel capabilities of CUPyDO are used and assessed for the three-dimensional test cases, this thesis does not provide any formal HPC analysis. From a performance point of view, it is verified that the computing time overhead brought by the coupling tasks remains extremely limited when compared to the time spent by the solvers for computing their solutions. However, no parallel scalability tests are performed and this task must therefore be considered as part of future work in the development of CUPyDO. Moreover, the scalability of the coupled simulation strongly depends on the scalability of the individual solvers, which is outside the scope of the present work.

6.1 Vortex-induced vibration of a circular cylinder with one degree of freedom

The first case to be assessed is the response of a circular cylinder with one degree of freedom immersed in a cross-flow that illustrates the VIV phenomenon. The simplified structural model of an oscillating body is here considered for the numerical simulations with CUPyDO. The main reference for this case is the work of Dettmer and Perić [216] whose simulations are reproduced in this work.

This test case demonstrates that the implemented coupling architecture is able to predict vortex-induced vibration and, in particular, the lock-in phenomenon. It relies on the strongly coupled BGS algorithm for mechanical coupling. Matching interface meshes are used to verify that the energy is exactly conserved through data communication. Considering the absence of numerical coupling instabilities (Ma = 148.16) and the relatively small size of the time step compared to the characteristic time scales, a small number of coupling iterations (typically not higher than 3) is required to reach the given coupling tolerance.

6.1.1 Description of the simplified model

The two-dimensional dynamics of the one-degree-of-freedom oscillating cylinder is usually used to simplify the study of VIV phenomena. As illustrated in Fig. 6.1, a perfectly rigid circular cylinder of diameter D and mass m is immersed in a cross-flow of velocity U and is allowed to move vertically. The motion is constrained by a linear spring of stiffness k and a linear damper with constant c. The simple application of Lagrange's equations (Appendix B) gives

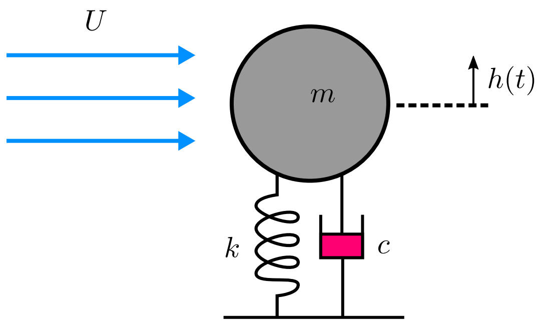


Figure 6.1: Oscillating rigid cylinder immersed in a cross flow. The motion is constrained by a standard spring-damping system.

the equation of motion

$$m\ddot{h} + c\dot{h} + kh = L\,, \tag{6.1}$$

in which h is the vertical position of the cylinder with respect to a reference position and $L(h, \dot{h}, t)$ is the resulting unsteady aerodynamic lift exerted by the flow on the cylinder and that depends nonlinearly on the solid motion.

From the structural point of view, one can define the natural frequency of the undamped system as

$$f_0 = \frac{1}{2\pi}\sqrt{\frac{k}{m}}\,, \tag{6.2}$$

and the damping coefficient

$$\zeta = \frac{c}{2\sqrt{km}}\,. \tag{6.3}$$

The fluid flow is usually characterized by its Reynolds and Mach numbers, while the Strouhal number is used to define a non-dimensional frequency (either the shedding frequency or the oscillation frequency of the cylinder). The coupling between the flow and structural parameters is quantified by the mass ratio which is expressed, in this particular case, as

$$\mathrm{Ma} = \frac{4m}{\pi \rho D^2 l}\,. \tag{6.4}$$

The denominator represents the mass of a fluid volume corresponding to the volume (area in 2D) of the cylinder. The out-of-plane length, l, of the cylinder is considered to be 1 m.

The oscillating cylinder is prone to VIV. The vortex shedding pattern that develops in the wake of the cylinder results in a harmonic lift. If it is assumed that the main frequency of the shedding is f_v, it is expected that the strongest fluid-body interaction takes place when $f_v \approx f_0$, which can be associated to with resonant behavior. As a first approximation the harmonic lift can be considered independent of the cylinder motion, so that

$$L(t) = \frac{1}{2}\rho D U^2 C_L(t)\,, \tag{6.5}$$

where the harmonic lift coefficient is

$$C_L(t) = C_L^0 \sin(2\pi f_v t)\,. \tag{6.6}$$

The lift coefficient amplitude C_L^0 and the shedding frequency f_v depend mainly on the Reynolds. Note that the oscillating lift coefficient also satisfies an oscillator equation:

$$\ddot{C}_L + (2\pi f_v)^2 C_L = 0\,. \tag{6.7}$$

As claimed by Païdoussis *et al.* [30], this is the central idea of the wake oscillator model. When the two oscillators, Eqs. (6.1) and (6.7), are decoupled, the response of the cylinder corresponds to the response of a system forced at frequency f_v,

$$h = h_0 \sin(2\pi f_v t)\,, \tag{6.8}$$

where the amplitude h_0 is illustrated in Fig. 6.2. However, experiments have shown that the

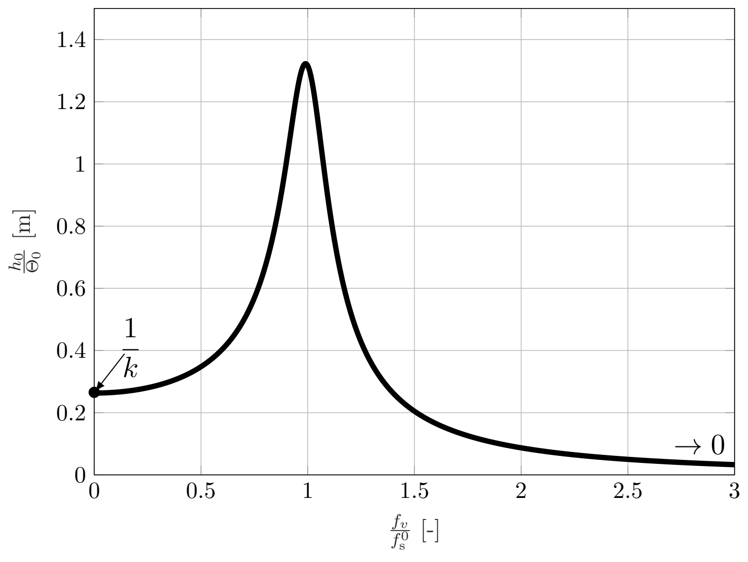


Figure 6.2: Schematic amplitude of the response of the cylinder as a function of the frequency of the harmonic loading. Numerical values are arbitrary. $\Theta_0 = \frac{1}{2}\rho D U^2 C_L^0$.

displacement of the cylinder can be of the order of a diameter which is large enough to significantly impact the vortex dynamics generating the lift. Hence, taking into account the coupling with the structure motion is important. This coupling can be achieved by introducing a forcing term into the oscillating lift equation [30],

$$\ddot{C}_L + (2\pi f_v)^2 C_L = \alpha \frac{\ddot{h}}{D}\,, \tag{6.9}$$

where α is a constant. A straightforward modal analysis of the set of linear equations formed by Eqs.(6.1) and (6.9) provides the dependence of the frequencies (shedding frequency and structural response) on the free-stream velocity, as shown in Fig. 6.3 (a). A critical free-stream velocity U_c can be defined as the velocity for which the shedding frequency is equal to the natural frequency of the structural system. This critical value is expressed using the definition of the Strouhal number Str as

$$U_\mathrm{c} = \frac{f_0 D}{\mathrm{Str}}\,. \tag{6.10}$$

It is found that in a range of velocities close to the critical velocity U_c, i.e. $f_v \approx f_0$, only one frequency exists in the system. In this range, there exists a mode with negative damping, as illustrated in Fig. 6.3 (b), which confirms the presence of coupling instabilities leading to high-amplitude responses.

6.1.2 Case description and simulation parameters

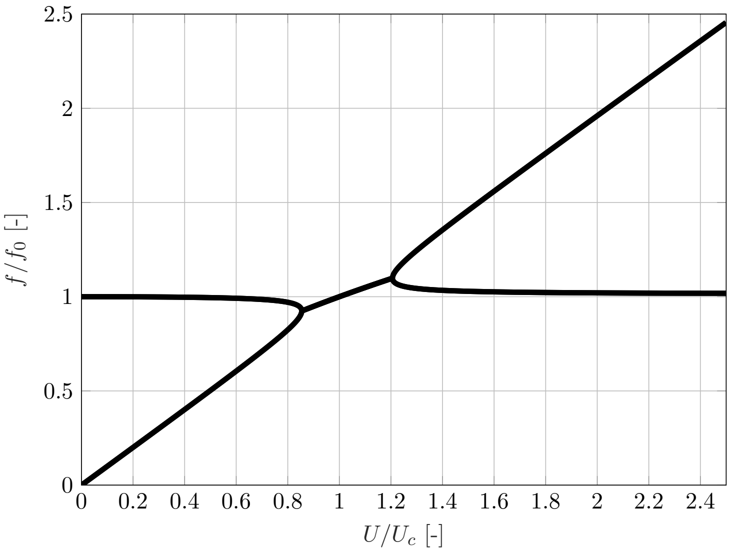
The purpose of the case is to simulate the response of the cylinder for different laminar flow Reynolds numbers in the range 90-120. The fluid and solid physical parameters are summarized in Tab. 6.1. The combination of these parameters gives a mass ratio Ma = 148.16, an undamped natural frequency $f_0 = 7.016$ Hz and a damping coefficient $\zeta = 0.0012$. The cylinder is immersed

Solid	Cylinder mass [kg]	m	0.298
	Cylinder diameter [m]	D	0.0016
	Spring stiffness [N/m]	k	579
	Damping [Ns/m]	c	0.0325
	Cylinder out-of-plane length [m]	l	1
Fluid	Density [kg m^{-3}]	ρ	1000
	Dynamics viscosity [Pa s]	μ	0.001

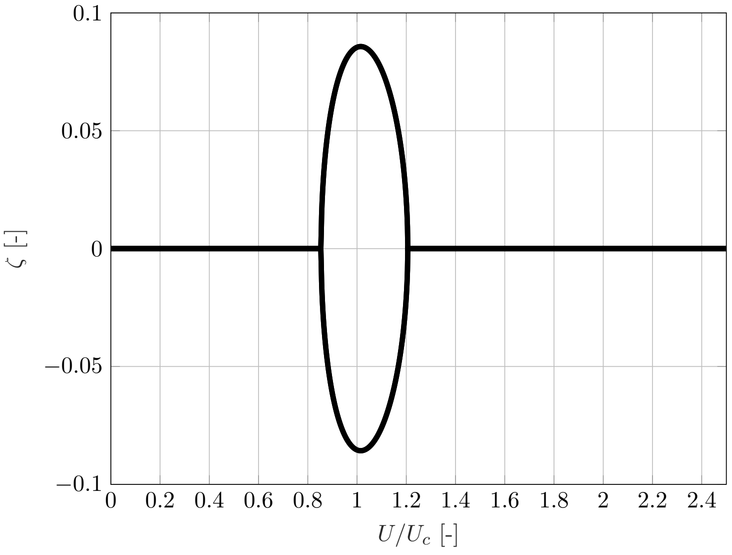
Table 6.1: Physical parameters for the one degree-of-freedom cylinder in a cross flow.

in a circular farfield flow domain where the outer boundary is located at 25 D, which corresponds to the common practice for minimizing the effects of boundary conditions. No-slip and adiabatic conditions are applied to the cylinder walls. The solver SU2 is used for the simulation of the fluid by solving the laminar Navier-Stokes equations. As depicted in Fig. 6.4, the domain is discretized with a structured O-mesh containing 25920 grid nodes (81 in the radial direction and 320 in the circumferential direction) with a clustering near the cylinder walls. Convective fluxes are evaluated with the second-order centered JST scheme. Time integration is performed with the second-order Euler implicit dual-time stepping scheme of SU2. The rigid body integrator introduced in Section 3.8 is used to compute the motion of the cylinder that is ruled by Eq. (6.1). Time integration of this equation is performed with the Runge-Kutta scheme. Initial conditions are such that the fluid flow is uniform and the cylinder is at rest, $h(t=0) = \dot{h}(t=0) = 0$.

The coupling is achieved with a strongly coupled BGS scheme. As the problem is characterized by a high mass ratio, no numerical instabilities are expected and, hence, no coupling under-relaxation is applied. A second-order interface predictor is used. Although the rigid body integrator only requires integrated fluid loads (in this case the resultant lift) to compute the motion of the cylinder, an interface mesh is intentionally defined on the solid side in order to receive



(a)



(b)

Figure 6.3: Dynamics of the coupled wake-cylinder oscillators. Evolution of the frequency (a) and damping (b) as a function of the flow velocity.

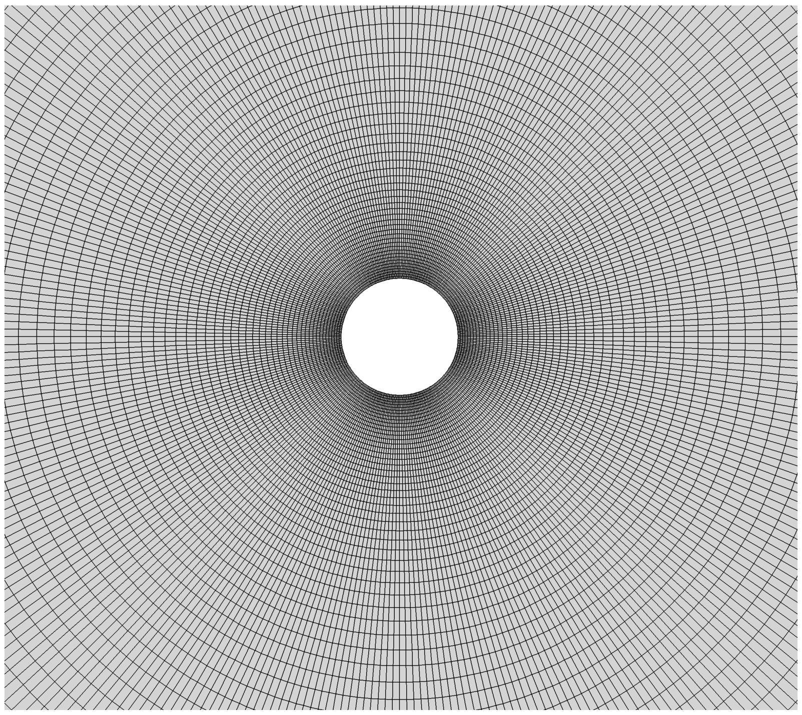


Figure 6.4: Illustration of the structured O-mesh mesh in the vicinity of the cylinder.

the nodal fluid loads. Integration is then performed by the rigid body solver. This allows us to test and verify, on a simplified case, the standard inter-communication procedures as it would be performed with a standard FEM solid solver. The two interface meshes have, by construction, matching discretizations so that no interpolation error is introduced in this case. Simulation parameters are summarized in Tab. 6.2. The value of the time step corresponds to 1/57 of the period of the free response of the cylinder in vacuum (i.e. in absence of surrounding fluid). The value of the time step was set according to Dettmer and Perić [216] who chose a small time step compared to the natural period to rule out significant inaccuracies due to time integration errors. Note that, for some flow Reynolds numbers, a large simulated time up to 400 s is necessary for the cylinder response to reach an established regime. The coupling tolerance corresponds to about $6 \cdot 10^{-4}\, D$.

Time step [s]	Δt	0.0025
Simulated physical time [s]	t_{tot}	Up to 400
Maximum number of coupling iterations per time step	$\hat{n}_{\text{FSI}}$	6
Coupling tolerance [m]	ε	10^{-6}

Table 6.2: Simulation parameters for the one degree-of-freedom cylinder in a cross flow.

In order to evaluate the capabilities of the fluid solver for this case, preliminary simulations involving a fixed cylinder have been performed. The reference of these simulations is the experimental study of Roshko [217] in which Reynolds numbers in the range $50-1400$ were considered for cylinder diameters in the range $0.0235-0.635$ cm. The fluid results obtained during the preliminary study are also used to validate the rigid-body integrator. The calculated lift force is applied on the moving cylinder over time to compute the uncoupled forced response which is then compared to an analytical solution.

6.1.3 Results

The results for the uncoupled cases are first presented, starting with the flow around a fixed cylinder and followed by the forced response of the cylinder to an imposed force. Results for the fully coupled case are subsequently detailed.

Fluid flow around a fixed cylinder

A first set of simulations are performed for a cylinder that is fixed. As the lock-in phenomenon is mainly related to the frequency of the vortex shedding f_v, this frequency is computed as a function of the Reynolds number in the range 90-120, for which the flow is laminar. The frequency of the vortex shedding is simply extracted from the time evolution of the lift coefficient by means of a Fourier analysis. The results are depicted in Fig. 6.5. The computed normalized shedding frequency as a function of Re is compared with very good agreement to an empirical relation based on experiments carried by Roshko [217]. This relation gives the Strouhal number,

$$\mathrm{Str} = \frac{f_v D}{U} = 0.212\left(1 - \frac{21.1}{\mathrm{Re}}\right), \tag{6.11}$$

as a function of the Reynolds number in the considered range. The good agreement between the computed results and the experimental data shows that the fluid solver is correctly set up for the next simulations with a moving cylinder. The shedding frequency is such that it equals the natural frequency of the system for Re $\approx$ 106. Thus, resonance is expected at this flow condition once the motion of the cylinder is allowed.

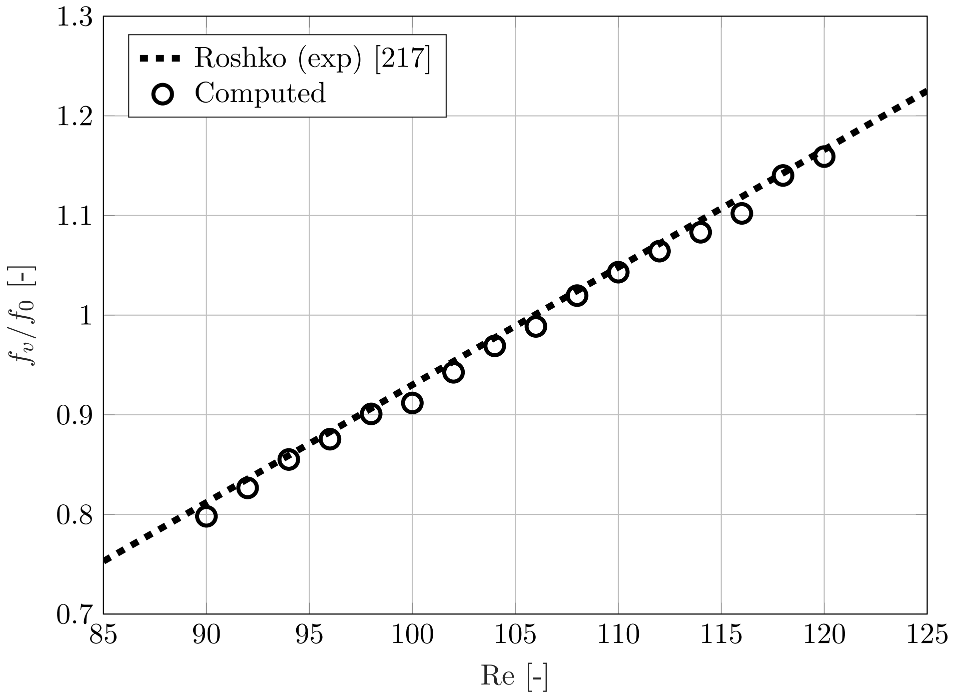


Figure 6.5: Normalized frequency of the vortex shedding as a function of the Reynolds number. Case of the fixed cylinder. Present results are compared to experimental data coming from Roshko [217].

Uncoupled forced response of the moving cylinder

The rigid-body integrator is used to compute the forced response of the moving cylinder based on the lift force computed with the previous flow simulations. The lift force imposed to the rigid body is of the form

$$L = \hat{L} \sin(\omega_v t) \ , \tag{6.12}$$

where $\omega_v = 2\pi f_v$ and $\hat{L}$ are taken from the previous computations at different Reynolds numbers. Analytically, the forced response of the cylinder can be expressed as

$$h = \hat{h} \sin(\omega_v t) \ , \tag{6.13}$$

in which the amplitude of the response is given as a function of the amplitude of the lift coefficient and the shedding frequency,

$$\hat{h} = \left| \frac{\hat{L}}{k} \cdot \frac{1}{1 - \frac{\omega_v^2}{\omega_0^2} + i2\zeta\frac{\omega_v}{\omega_0}} \right| \ . \tag{6.14}$$

The cylinder displacement reaches its maximum value for $f_v \rightarrow f_0$[1], as shown in Fig. 6.6 for different Reynolds numbers. As expected, resonance is reached at Re $\rightarrow$ 106 when the frequency of the vortex shedding and the natural frequency are coincident. A direct comparison with the analytical expression, Eq. (6.14), is also performed, showing excellent agreement and thus verifying the rigid-body integrator for this one-degree-of-freedom case.

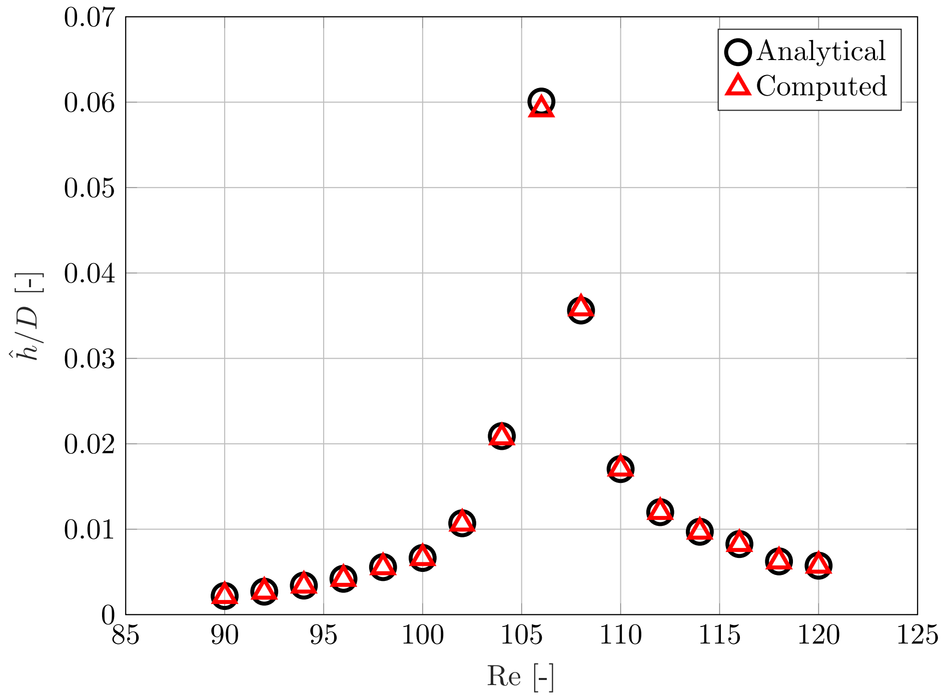


Figure 6.6: Normalized amplitude of the uncoupled forced response of the cylinder as a function of the flow Reynolds number.

Fully coupled fluid-structure system

Simulations are now performed under the same flow conditions but considering a cylinder free to move vertically. The BGS coupling procedure is found to be very efficient for every Reynolds

[1]The maximum is not exactly located at $f_v = f_0$ due to the presence of structural damping.

number tested: only two coupling iterations per time step are required to reach the prescribed coupling tolerance. The computed normalized shedding frequency as a function of Re is illustrated in Fig. 6.7 and compared to the results previously obtained by Dettmer and Perić [216]. Results for the moving cylinder show that the interaction causes the shedding frequency to lock onto the natural frequency of the cylinder ($f_v = f_0$) in the range Re $= 100 - 110$, which is referred to as the lock-in region. The lock-in phenomenon can be considered as a form of nonlinear resonance. When the shedding frequency approaches the natural frequency of the cylinder, the energy exchange between the flow and the structure increases significantly and locks the system on the resonant frequency. Outside the lock-in range, the shedding frequency falls back to values that correspond to the fixed configuration. We note a shift of the lock-in region towards higher Reynolds numbers compared to the results of Dettmer and Perić [216]. The difference in the mesh discretization between this work and the reference could explain the shift. Indeed, the mesh used in the present computations is much more refined than the finest mesh used by Dettmer and Perić (25600 cells here against 5374 in their work) and they have already noticed a clear sensitivity of the limit of the lock-in region to the mesh refinement. In the reference results, we also note the missing data for Re $= 96$, for an unknown reason.

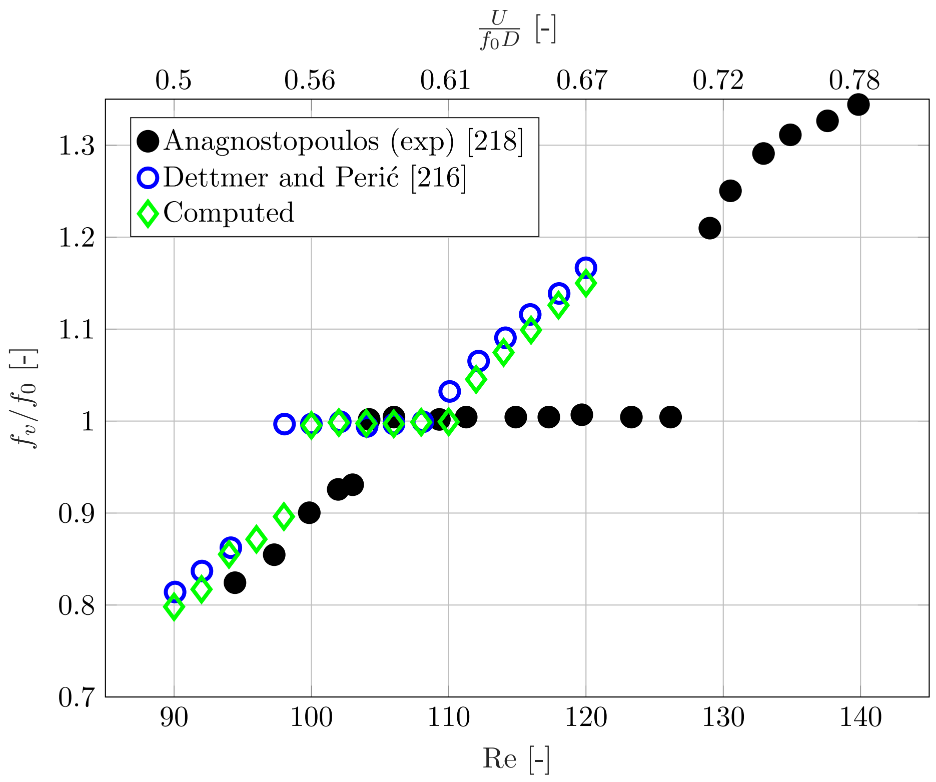


Figure 6.7: Normalized frequency of the vortex shedding as a function of the Reynolds number and the normalized velocity. Case of the moving cylinder.

Lock-in can also be observed in the amplitude of the cylinder response. This amplitude, normalized by the diameter, is shown in Fig. 6.8 as a function of the Reynolds number. The results computed by CUPyDO are again compared to those obtained by Dettmer and Perić [216] with good agreement, although we observe the same shift of the lock-in region. Inside the lock-in region, the amplitude of the response of the cylinder increases significantly up to $0.4D$, while it is nearly negligible outside this region. When increasing the Reynolds number from the lowest simulated value, the amplitude features a sharp transition between Re $= 98$ and Re $= 100$ when entering the lock-in region. Then it monotonically decreases until a second transition occurs between Re = 110 and Re = 112 when exiting lock-in region. A comparison between the response obtained with the fully coupled model and the forced uncoupled response clearly shows that neglecting the effect of the solid motion on the fluid flow leads to a significant error: the maximum amplitude obtained by the coupled solution is more than four times higher than the maximum amplitude of the uncoupled approach. High amplitude response is also maintained on a much larger range of Reynolds number that corresponds to the entire lock-in region. This clearly demonstrates the importance of taking into account the coupled effects in the design of systems potentially subjected to VIV.

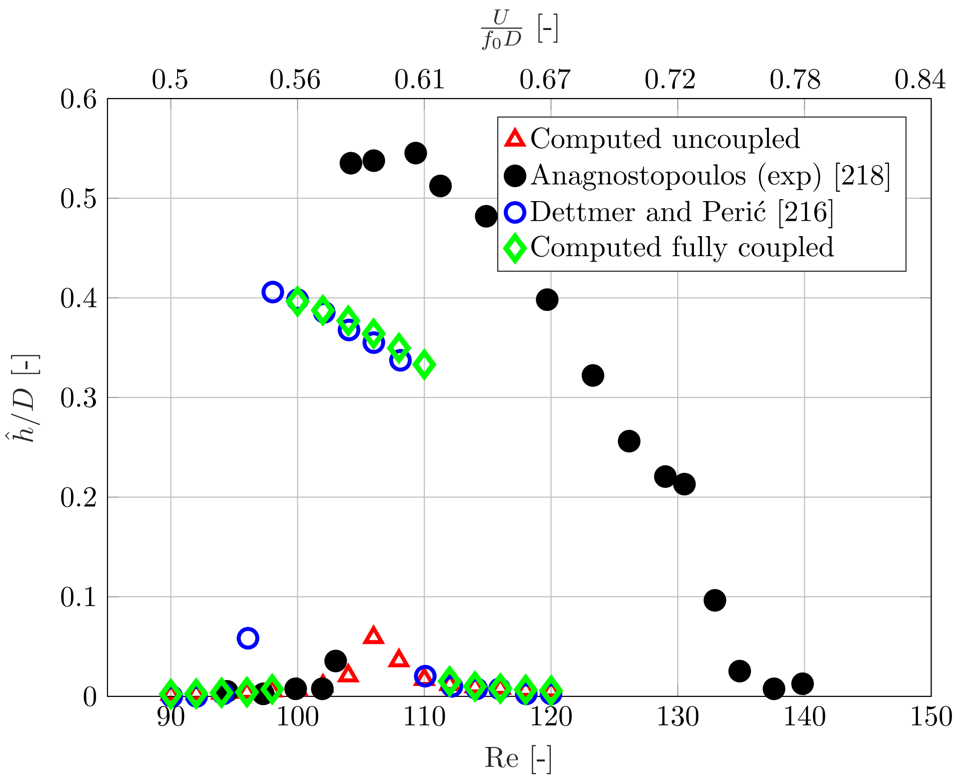
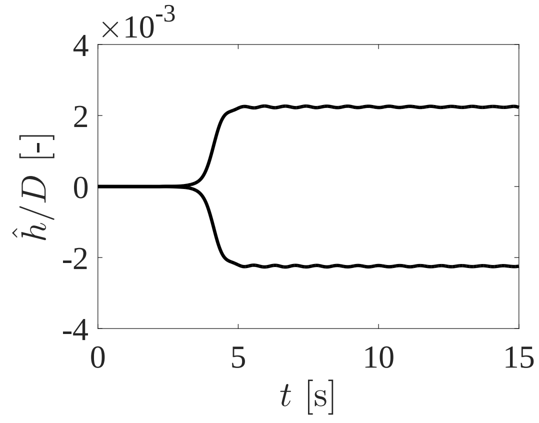


Figure 6.8: Normalized amplitude of the displacement of the cylinder as a function of the Reynolds number and the normalized velocity.

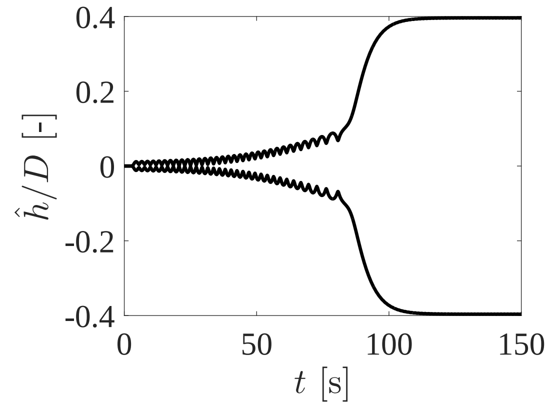
The computational results are also compared with the results from the experiment carried by Anagnostopoulos and Bearman [218]. Although the results follow the same qualitative trend, quantitative discrepancies are observed. The maximum amplitude obtained numerically differs from the experimental value by 27%. The computed lock-in region is also narrower and shifted towards smaller Reynolds number. These discrepancies were already addressed in Dettmer and Perić [216], who pointed out the significant differences between the numerical model and the experimental model. In the latter, the flow is actually a free-surface water channel (70 cm depth) in which the submerged (vertical) length of the cylinder is 12 cm [218]. They also highlight that no horizontal end plate had been fixed at the tip of the cylinder. Therefore, the vortex shedding at the lower end of the cylinder was in fact three dimensional.

The influence of the Reynolds number on the coupled system can be observed on a detailed analysis of the response. Fig. 6.9 illustrates the envelope[2] of the response for Reynolds numbers 90, 100, 104 and 114. The response for Re = 90 typically corresponds to the behavior obtained before reaching the lock-in region. Once the vortex shedding is established, the amplitude quickly stabilizes to a very low value. When entering the lock-in region at Re = 100, the amplitude

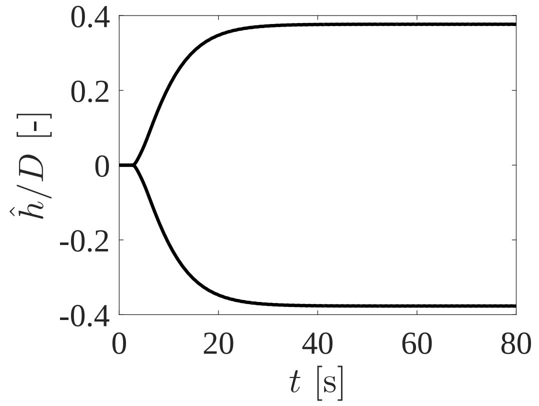
[2]The envelope here corresponds to the curve formed by the local extrema of the response.



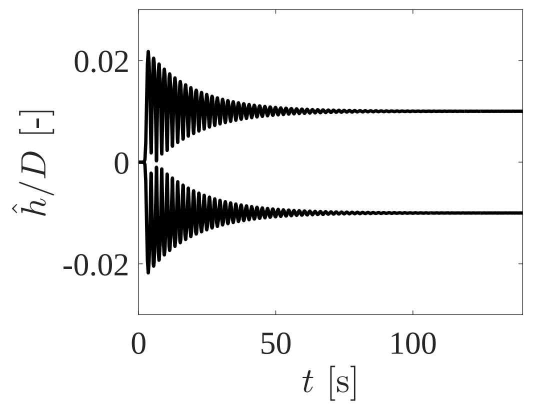
(a) Re = 90



(b) Re = 100



(c) Re = 104



(d) Re = 114

Figure 6.9: Envelope of the normalized amplitude of the cylinder response for four representative Reynolds numbers.

features a slow increase in time while being modulated by lower frequency waves. This transient phase is maintained until the amplitude abruptly transitions into a permanent locked-in state. Note however the long simulation time (more than 80 s) needed to reach this transition. Once inside the lock-in region, e.g. for Re = 104, the amplitude quickly increases to reach a plateau. Finally, beyond the lock-in region, e.g. for Re = 114, the amplitude first features a sharp increase before being damped to reach an established regime with a much lower amplitude than in the lock-in region. These trends were also observed by Dettmer and Perić [216]. The distinct patterns that the response of the cylinder features in a narrow range of Reynolds numbers demonstrate the complexity brought by the coupling of two physics in one single system.

6.1.4 Test case summary

This test case validates the coupling tool CUPyDO for the computation of VIV involving a simple one-degree-of-freedom rigid structure. Although the structural model remains simple, this case already clearly shows that the interaction between the two coupled physics can reach a significant level of complexity. This complexity is highlighted here by the existence of the lock-in region and the significant impact of the Reynolds number on the cylinder response in the vicinity of this region. These two key features are very well reproduced by the simulations.

This case also illustrates the significant error that can result from an uncoupled analysis of the system. By computing the fluid loads on a fixed structure and then simply applying these loads to the uncoupled rigid body model, the influence of the solid dynamics on the flow is not represented at all. The amplitude of the response of the cylinder is thus dramatically underestimated and the lock-in phenomenon cannot be captured. In practice, neglecting such coupling effects in the design of structures submitted to VIV, such as pipes, cables, towers or industrial chimneys, may have severe consequences for their long-term integrity.

6.2 Flutter of an airfoil with pitch and plunge degrees of freedom

This test case is used to evaluate the capabilities of CUPyDO (and by extension the coupled solvers) to simulate the coupled mode flutter phenomenon in transonic conditions. The classical Isogai wing section aeroelastic case (case A) [219,220], whose structural model is an airfoil with pitch and plunge degrees of freedom, is here used as reference.

The purpose of this test case is to provide a first assessment of the capabilities of CUPyDO to accurately simulate aeroelastic flutter. The focus here is on the transonic regime because it represents the typical flight conditions of standard commercial airliners. The aeroelastic study is also extended by comparing the accuracy of weak and strong coupling schemes implemented in CUPyDO for the prediction of the flutter inception.

6.2.1 Description of the simplified model

The two-dimensional dynamics of an airfoil with pitch and plunge degrees-of-freedom is introduced to study the coupled mode flutter of typical wing sections. The aeroelastic model is depicted in Fig. 6.10: a perfectly rigid airfoil with a chord $c = 2b$ is allowed to move vertically and to rotate around its elastic axis. The positions of the center of gravity and the elastic axis are respectively denoted by x_{cg} and x_{ea} and measured from the leading edge. The displacement h of the elastic axis is positive downwards and the pitch angle α is positive clockwise. The static imbalance S is defined as the product of the airfoil mass m with the distance $x_{\mathrm{cg}} - x_{\mathrm{ea}}$ between the center of gravity and the elastic axis. The structural restoring force is provided by a spring-damper system with stiffnesses K_h and K_α and damping coefficients C_h and C_α for the plunging and pitching mode, respectively.

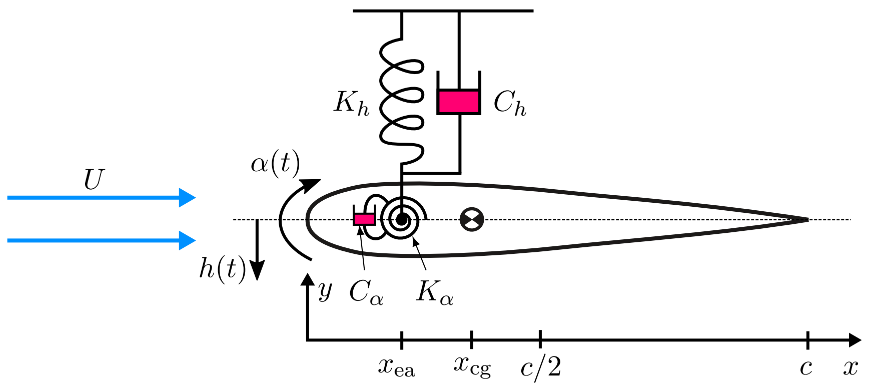


Figure 6.10: Airfoil with pitch and plunge degree-of-freedom immersed in a cross flow.

The linear equations of motion are obtained from Lagrange's equations [64]:

$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L, \\ S\ddot{h} + I_{\text{ea}}\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M, \end{aligned} \tag{6.15}$$

where I_{ea} is the moment of inertia of the airfoil around the elastic axis, L the aerodynamic lift (positive upwards) and M the aerodynamic moment with respect to the elastic axis (positive clockwise). The coupled system can be characterized by several non-dimensional parameters that are the normalized static imbalance χ and inertia r_α, the structurally-uncoupled natural frequency ratio $\bar{\omega}$ and the mass ratio Ma:

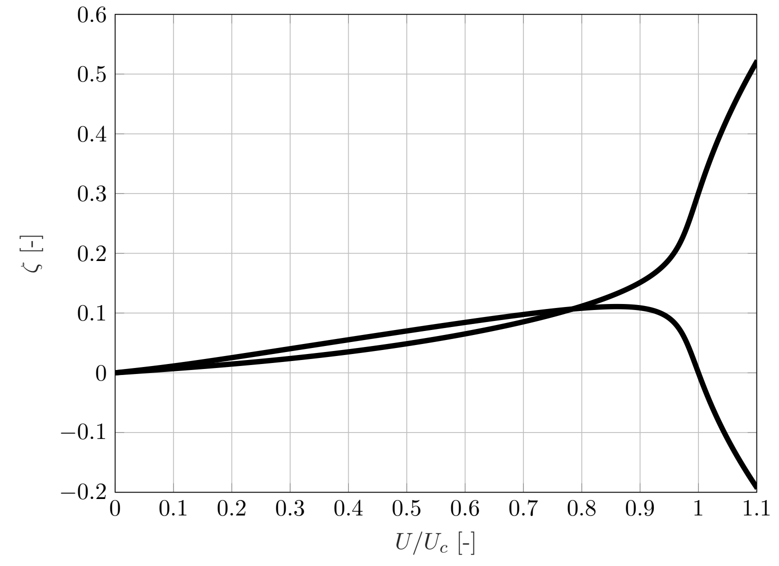
$$\chi = \frac{S}{mb}\,, \quad r_\alpha^2 = \frac{I_{\text{ea}}}{mb^2}\,, \quad \bar{\omega} = \frac{\omega_h}{\omega_\alpha}\,, \quad \text{Ma} = \frac{m}{\pi\rho b^2}\,. \tag{6.16}$$

In these expressions b is the half-chord, and $\omega_h = \sqrt{K_h/m}$ ($\omega_\alpha = \sqrt{K_\alpha/I_{\text{ea}}}$) is the uncoupled plunging (pitching) natural frequency.

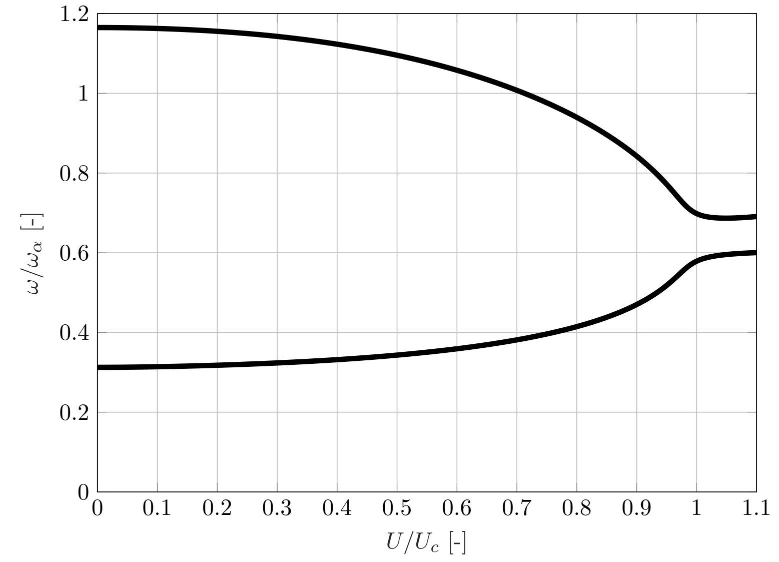
Solving the coupled problem given by Eq. (6.15) requires the computation of the aerodynamic lift and moment, which obviously depend on the airfoil motion:

$$L,\, M = f(\dot{h}, \dot{\alpha}, \alpha)\,. \tag{6.17}$$

By assuming an incompressible attached flow, thin airfoil theory [52] may be used to express the aerodynamic loads. The flow dynamics can be modelled by distinct approaches with different accuracy levels. The simplest quasi-steady approach assumes that the aerodynamic load at time t only depends on the position and the (relative) velocity of the airfoil at the same instant. Hence, the effect of the wake, i.e. the flow history, is not taken into account. The effect of the wake can be considered by decomposing the unsteady motion into a superposition of many small impulsive changes in pitch and plunge degrees of freedom, each of them generating a trailing edge vortex whose influence on the aerodynamic loads is represented by the Wagner function [35]. Theodorsen theory [35, 221] also introduces the dynamic effects of the wake but supposes a purely sinusoidal response of the airfoil. A modal analysis of the system provides the damping and frequency as a function of the free-stream velocity, as schematically represented in Fig. 6.11. The system damping ratio in Fig. 6.11 (a) shows that there exists a mode which becomes unstable ($\zeta < 0$) beyond a certain velocity identified as the critical flutter velocity U_c. For all velocities beyond the critical value, the response of the system to any initial perturbation remains unbounded in time (even in the presence of structural damping). At low flow velocity, the frequencies of each mode are close to the wind-off natural frequencies. When the fluid velocity increases, the frequencies typically start to converge towards each other. Negative damping and coalescence of the frequencies are typical features of the system when it approaches critical (flutter) conditions.



(a)



(b)

Figure 6.11: Dynamics of the pitch-and-plunge airfoil immersed in a cross flow. Variation of the damping (a) and frequency (b) as a function of the flow velocity.

6.2.2 Case description and simulation parameters

The Isogai wing section represents the dynamics of the outboard portion of a swept-back wing in the transonic regime. The airfoil is a symmetric NACA 64a010 profile with chord $c = 2b$. The elastic axis is placed in front of the airfoil at a distance $x_\mathrm{f} = -b$ from the leading edge. By following the definitions introduced in Section 6.2.1, the relevant physical parameters of the coupled system are listed in Tab. 6.3. The airfoil is immersed in a circular fluid domain with

Normalized static unbalance [-]	χ	1.8
Normalized inertia [-]	r_α	1.865
Natural frequency ratio [-]	$\overline{\omega}$	1
Mass ratio [-]	Ma	60
Natural pitching frequency [rad s^{-1}]	ω_α	100
Mach number [-]	M	From 0.75 to 0.895
Reynolds number [-]	Re	$12.56 \cdot 10^6$

Table 6.3: Structural and flow parameters for the Isogai wing section aeroelastic test case.

outer boundary extending to $25c$. Farfield flow conditions are imposed on this outer boundary. As the fluid is here assumed to be inviscid, a slip-wall condition ($\boldsymbol{v} \cdot \boldsymbol{n} = 0$) is imposed on the airfoil boundary. The Euler equations are solved with SU2 on the fluid domain which is discretized by a structured O-mesh of 21760 cells ($68\times$ 320 for the radial and circumferential direction, respectively) with a small stretching from the airfoil surface to the outer boundary. Convective fluxes are evaluated with the second-order centered JST scheme and time integration is performed using a second-order Euler implicit dual-time stepping scheme. The simulations are performed starting from a uniform flow. The rigid body integrator is used to predict the motion of the airfoil with the generalized-α scheme, starting from an initial pitch angle $\alpha_0 = 0.0174$ rad ($= 1^\mathrm{o}$).

The coupling is here achieved with either an explicit weak coupling or an implicit strong coupling using the BGS scheme without relaxation. The purpose is to evaluate the impact of this choice on the computed flutter airspeed. Similarly to the previous test case, matching interface meshes are used to transfer interface quantities (fluid loads and rigid body displacement). The parameters for the simulations are summarized in Tab. 6.4. The time step size is such that 39 time steps are needed to cover a period of the uncoupled pitch mode. The value of the coupling tolerance corresponds approximately to 10^{-4} times the displacement of the center of gravity associated with the initial perturbation α_0.

Time step [s]	Δt	0.0016
Simulated physical time [s]	t_tot	2
Maximum number of coupling iteration per time step	$\hat{n}_\mathrm{FSI}$	6
Coupling tolerance [m]	ε	$1 \cdot 10^{-6}$

Table 6.4: Simulation parameters for the Isogai wing section aeroelastic test case.

In order to evaluate the capabilities of SU2 for simulations of transonic flows around a moving airfoil, a preliminary computation using the same mesh and numerical methods is performed, in which the motion of the airfoil is imposed. The conditions of the simulation are taken from the experimental study conducted by Davis [222]. The imposed motion is a pitching motion around the quarter-cord with an angle

$$\alpha(t) = \overline{\alpha} + \hat{\alpha} \sin(\omega_0 t) \,, \tag{6.18}$$

where in this case $\overline{\alpha} = 0$, $\hat{\alpha} = 1.01$ and the reduced frequency

$$k_0 = \frac{\omega_0 c}{2U} = 0.202\,, \tag{6.19}$$

based on the free-stream flow velocity U. The Mach and Reynolds numbers are $\mathrm{M} = 0.796$ and $\mathrm{Re} = 12.56 \cdot 10^6$, respectively. Note that the coupler is intentionally not used for the one-way coupling between the motion of the airfoil and the flow solver. Instead, and in order to specifically assess the flow solver, the capability of SU2 to simulate moving rigid bodies is directly used to impose the motion of the airfoil.

The implementation of the pitch-plunge airfoil structural model is also verified by computing its uncoupled free response which can be compared to an equivalent model implemented in Matlab and integrated in time with Matlab built-in Runge-Kutta scheme.

6.2.3 Results

Results are first presented for the preliminary computations: the uncoupled free response of the structural model and the fluid flow around the airfoil with imposed motion. Then, an extensive study of the fully coupled system is provided with a specific focus on the flutter phenomenon and the results are compared with those from the literature.

Free response of the pitch-plunge airfoil model

The pitch-plunge airfoil model implemented in the rigid body integrator is tested and verified for the computation of the wind-off free response. For this test, the parameters as listed in Tab. 6.5 are taken from the work of Amandolese *et al.* [223] in which a similar model of a pitching and plunging flat plate has been studied. The free response of the airfoil is computed by using the

m [kg]	1.3511
I_{ea} [kg m^2]	$2.0711 \cdot 10^{-4}$
S [kg m]	0.0038
K_h [N/m]	2647.1
C_h [Ns/m]	0.2392
K_α [Nm/rad]	0.6622
C_α [Nms/rad]	$3.5134 \cdot 10^{-4}$

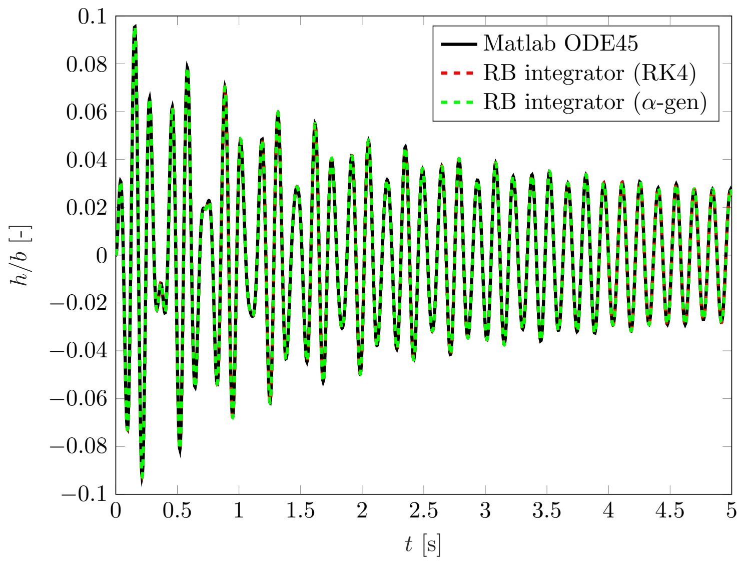
Table 6.5: Structural parameters used for the verification of the pitch-plunge airfoil model implemented in the rigid body integrator.

different time integrators (RK4 and α-gen) with a time step of $\Delta t = 0.001$ s. A pitch angle of $\alpha_0 = 0.174$ rad ($10°$) is arbitrarily chosen as initial condition. The results, illustrated in Fig. 6.12, are compared to those obtained from the built-in `ode45` integration routine of Matlab with excellent agreement.

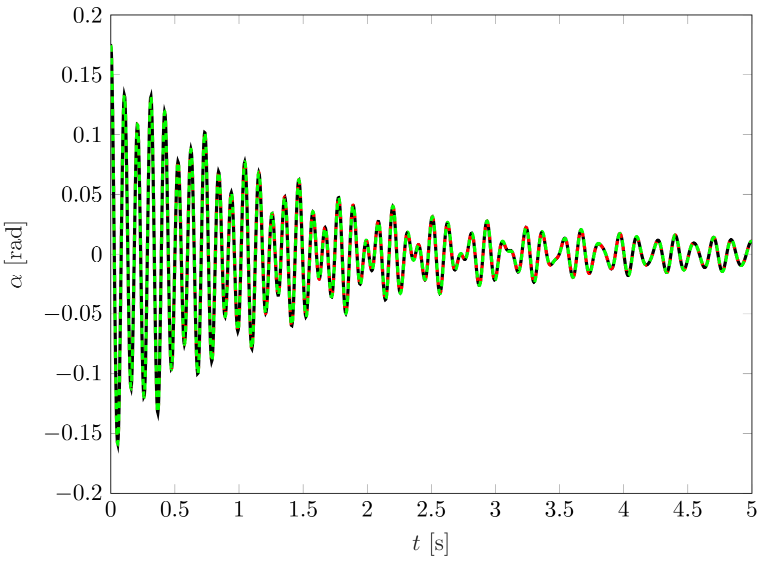
Fluid flow around the airfoil with imposed motion

The results of the preliminary one-way coupled simulations are now presented. The lift and moment coefficients,

$$\begin{aligned} c_l &= \frac{L}{0.5\rho c U^2}\,, \\ c_m &= \frac{M}{0.5\rho c^2 U^2}\,, \end{aligned} \tag{6.20}$$



(a) Plunge.



(b) Pitch.

Figure 6.12: Free response of the airfoil.

are shown as a function of the instantaneous angle of attack α in Fig. 6.13. The computed coefficients are compared to the experimental results from Davis [222] and to additional numerical results from McMullen *et al.* [224]. These latter authors performed the same numerical study, using a nonlinear frequency domain solver for the periodic solution of the Euler equations. For the lift coefficient (Fig. 6.13(a)), results computed with SU2 show good agreement with both experimental and computational references. Note that the mesh used here is 4 times more refined than the one used by McMullen *et al.* [224]. For the moment coefficient (Fig. 6.13(b)), the computational studies are still in good agreement with each other but show larger discrepancies with the experimental data. In this context, the discrepancy could be attributed to the fact that viscous effects are neglected. However, this is not confirmed by several viscous simulations that have been run by McMullen *et al.* [224] where the viscous results are extremely similar to the inviscid results. Pierce and Alonso [225] already used this case in the past for the development of an implicit preconditioned multigrid algorithm for unsteady turbulent Navier-Stokes calculations, and they obtained the same qualitative discrepancy compared to the experiment for the moment coefficient. They discussed the potential limitations of the turbulence model and also the accuracy of the experimental results by stating that the force coefficients are obtained by integrating the surface pressure measured on a sparse grid of pressure taps. No further investigation of the discrepancy could be found in McMullen *et al.* [224]. Nonetheless, the good match of computational data allows us to get sufficient confidence in the settings of SU2 for the simulations of transonic flows around moving airfoils and its applicability to transonic flutter.

Fully coupled aeroelastic analysis

At transonic flight conditions the aerodynamic loads are strongly nonlinear. Consequently, the expected aeroelastic phenomenon is nonlinear flutter, a generalization of the linearized flutter described previously. The system becomes unstable when, at its fixed point, the system Jacobian has at least one pair of complex conjugate eigenvalues with zero real part. At this critical condition, known as the nonlinear flutter point or the Hopf bifurcation point[3], the fixed point becomes unstable and a stable or unstable limit cycle with infinitesimal amplitude appears around it. In the supercritical Hopf case, the limit cycle is stable and its amplitude increases with airspeed. Aeroelastic systems can have more than one Hopf point as different pairs of eigenvalues can become unstable.

The flutter verification analysis is performed as follows. Several strongly-coupled FSI simulations at different transonic free-stream Mach numbers ($M = 0.7 - 0.9$) are performed for different values of the speed index

$$V^* = \frac{U}{b\omega_\alpha\sqrt{\mathrm{Ma}}} \tag{6.21}$$

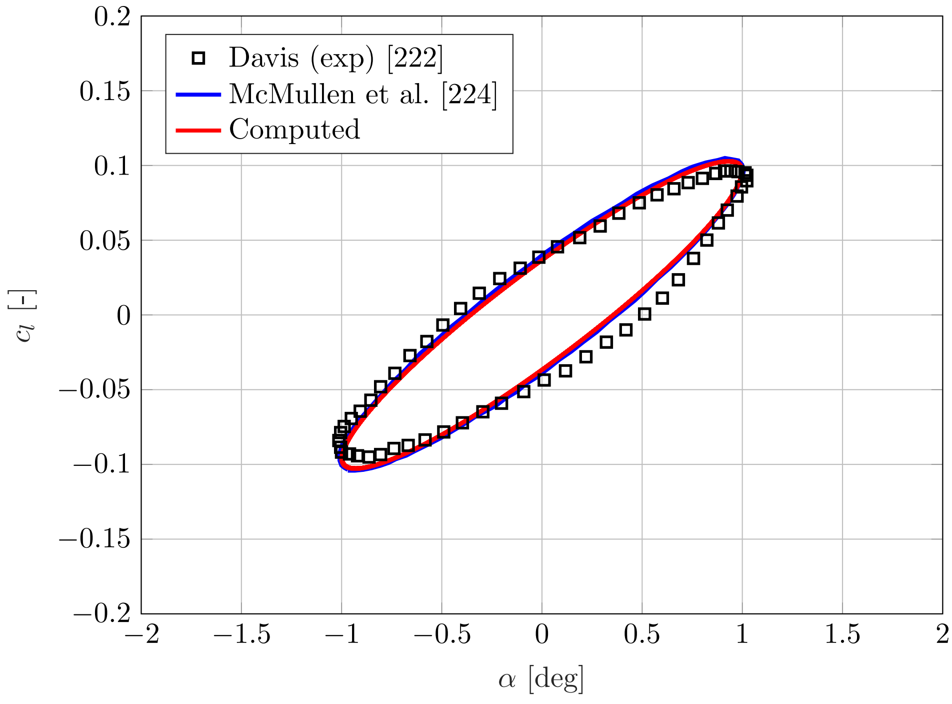
in order to predict the flutter point. As mentioned earlier, flutter occurs when a pair of complex conjugate eigenvalues becomes purely real or, equivalently, when one damping ratio becomes equal to zero. As this is a numerical solution we do not have access to the system Jacobian and hence to the eigenvalues but we can approximate the damping ratio using a simple signal processing approach. At a given Mach number, the damping coefficient ζ is computed for each speed index from the logarithmic decrement δ of the time response on the pitch and plunge degrees of freedom:

$$\zeta = \frac{1}{\sqrt{1+\left(\frac{2\pi}{\delta}\right)^2}}\,. \tag{6.22}$$

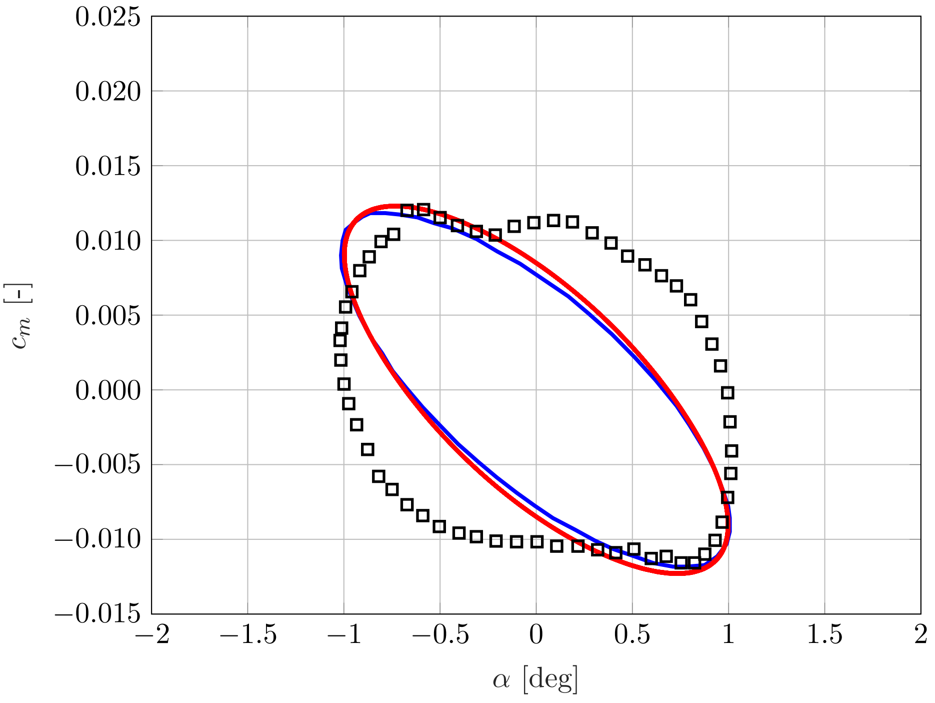
For an oscillating response of the system, the logarithmic decrement is computed by taking the ratio between two peaks of a quantity x separated by a multiple of the period T:

$$\delta = \frac{1}{n}\ln\frac{x(t)}{x(t+nT)}\,. \tag{6.23}$$

[3]More details about Hopf bifurcation will be given in the next chapter.



(a) Lift coefficient, positive upwards.



(b) Moment coefficient, positive clockwise.

Figure 6.13: Dynamic evolution of the lift and moment coefficient as a function of the angle of attack for an airfoil with an imposed sinusoidal pitching motion.

Note that the logarithmic decrement, originally well defined for one-degree-of-freedom system, works here approximately for the responses of this particular pitch-plunge airfoil system because one of the modes is highly damped and its effects disappear very quickly, such that the majority of the time response has a single harmonic component, as shown further below. The computed damping coefficient and corresponding speed index is plotted on a $\zeta - V^*$ diagram, as illustrated in Fig. 6.14. Determining flutter inception is performed by iteratively guessing a new speed index for which $\zeta \rightarrow 0$. Each new speed index to be simulated is determined by interpolating or extrapolating[4], from the diagram, the damping coefficients previously computed. The iterative procedure is continued until convergence up to an absolute tolerance of 10^{-4} is achieved on ζ, which corresponds to an absolute margin of error of approximately 0.001 on the flutter speed index. Note that for each simulation, three BGS iterations per time step are typically required to achieve the prescribed coupling tolerance. The computed flutter speed indices V_f^* are compared to values from the literature [226–229], in Fig. 6.15. The best approximation curve (spline

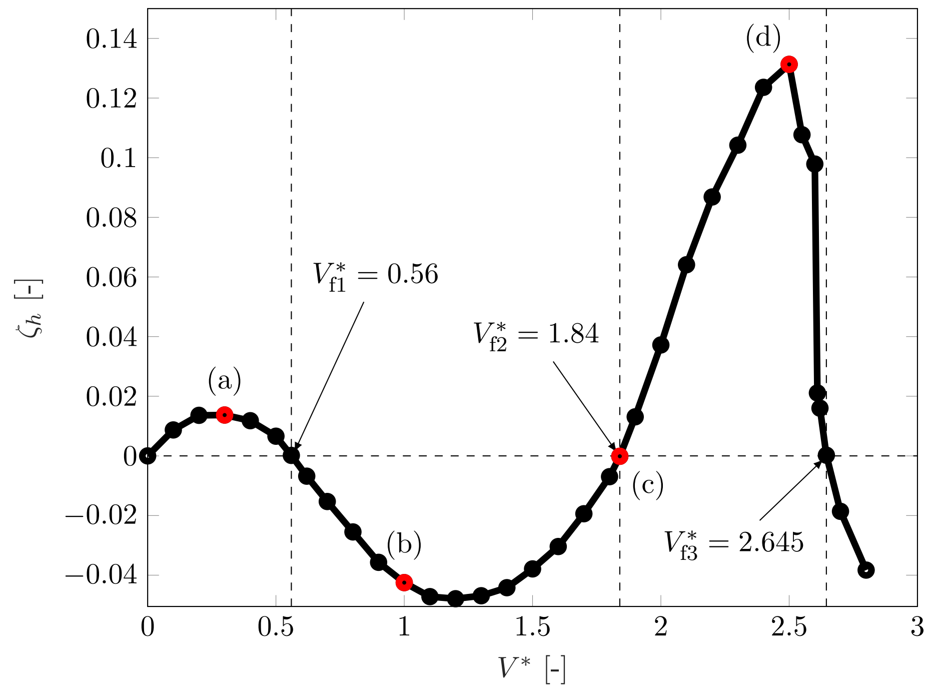


Figure 6.14: Damping coefficient, measured on the plunge degree of freedom, as a function of the speed index obtained at constant Mach number (M = 0.875). Letters (a) to (d) are used to identify speed indices that are used in Fig. 6.16.

interpolation) is a representation of the flutter boundary, i.e. the limit between the stable and unstable regions. It can be seen that the "transonic dip" and the typical "S-shape" of the flutter boundary are both well predicted for M = 0.7 − 0.9. For Mach numbers near the transonic dip (M = 0.8, 0.85 and 0.875), the agreement with all the references is excellent. An almost flat transonic dip is here captured that extends up to M = 0.888 which is a bit lower than the limit value M = 0.9 computed by Biao *et al.* [228]. Major discrepancies with the results of Thomas *et al.* [229] are found around M = 0.89, as they do not predict a flat transonic dip. For the second speed index at M = 0.875, we obtain a very good agreement with the results from Liu *et al.* [226] and Alonso *et al.* [227], while the value of Biao *et al.* seems over-predicted. At the beginning of the third flutter branch, around M = 0.85, the current results provide good agreement with the available references from Biao *et al.* and Thomas *et al.*. Discrepancies

[4]Interpolation and extrapolation are performed using the Matlab curve fitting built-in tool, based on spline interpolation.

appear again on this branch for higher Mach numbers, where the present results fall below the references. At M $\approx$ 0.9, the flutter speed indices from all references are scattered. This is also the only point for which the results obtained by CUPyDO provide a less satisfactory agreement with the results from Liu *et al.* and Alonso *et al.*. We also note that the coupling models used by Biao *et al.* and Thomas *et al.* are different than the approach followed by CUPyDO and the two other references. Indeed, Biao *et al.* solved the unsteady Euler equations coupled with the structural equations by using the first-order approximate boundary conditions [230] on a computational grid that does not vary in time. The model used by Thomas *et al.* is even more different and consists in solving the fluid equations in the frequency domain to generate a set of shape vectors containing the most dominant unsteady flow characteristics. These vectors are then used to construct a reduced-order model of the unsteady flow. The resulting reduced-order representation of the fluid dynamics is then coupled with the pitch-plunge airfoil model to form a reduced-order aeroelastic model.

The "S-shape" is a direct consequence of the occurrence of three flutter points after the transonic dip, for a Mach number in the range 0.85-0.888. Fig. 6.14 illustrates the typical shape of the damping coefficient, measured on the plunge degree of freedom, as a function of the speed index at Mach number M = 0.875. The aeroelastic response is characterized by four regions in the plot. The first region corresponds to stable (damped) responses of the airfoil when $\zeta > 0$. The end of the first stable region is located at $V^* = 0.56$ and corresponds to the first flutter point. Note that, for increasing values of V^*, the damping coefficient typically reaches a maximum value before decreasing to zero at flutter. The second region corresponds to an unstable time response of the airfoil when $\zeta < 0$. Again, the damping coefficient reaches an extremum before going back back to zero at the second flutter point for $V^* = 1.84$. The third region is a new stable region which is characterized by a significantly higher value of the maximum damping. However, the point of maximum stability is also much closer to the third flutter point ($V^* = 2.645$) causing a large drop of the damping coefficient when approaching this third flutter boundary. Such a rapid transition from highly stable to flutter conditions is referred to as *hard* flutter. Note that the same shape is recovered when measuring the damping on the pitch degree of freedom. Fig. 6.16 illustrates some typical aeroelastic responses corresponding to speed indices identified by the letters (a) to (d) in Fig. 6.14. Examples of decaying responses are depicted in Fig. 6.16(a) and Fig. 6.16(d), respectively, while unstable conditions are shown in Fig. 6.16(b) and Fig. 6.16(c). It can be seen from the unstable responses that the system reaches a limit cycle oscillation (LCO). This LCO is a consequence of nonlinear transonic aerodyanimc effects and, in particular, the presence of moving shocks along the chord. Fig. 6.17 illustrates this shock motion as the airfoil moves between two extreme positions during one limit cycle. A clear hysteresis cycle between the motion of the airfoil and the position of the two shocks on the upper and lower surfaces of the airfoil can be observed. The amplitude of the shock motion increases with the amplitude of the airfoil motion (unstable response) until the LCO. As shown in Fig. 6.18, the LCO regime is reached for any value of the speed index in the postcritical region, but the amplitude of this LCO varies with the speed index. In the first unstable region, the maximum LCO amplitude occurs at a velocity index slightly lower than that corresponding to the minimum damping (see Fig. 6.14). Moreover, a sharper transition between decaying response and LCO is observed at the flutter boundary corresponding to the lower velocity index. A similar sharp transition is also observed at the lower bound of the second unstable region. Additional simulations for larger values of V^* would be required to further analyze this region.

The frequency content of the response is analyzed through a Fourier analysis of the pitch and plunge response. Frequencies with a clear amplitude peak are then plotted as a function of the velocity index in Fig. 6.19. Note that both degrees of freedom have the same frequency content. In the first two regions (first stable and first unstable), only one frequency can be detected, as shown in Fig. 6.20(a). This frequency monotonically increases with the speed index. In the third region (second stable) a second higher frequency appears which initially decreases with V^* until

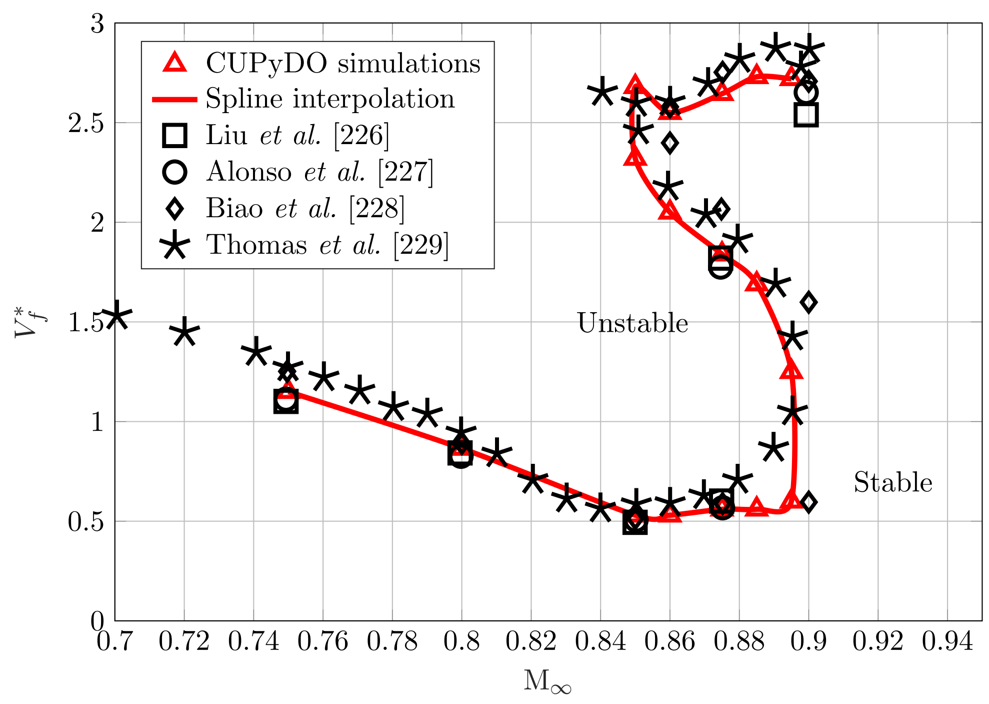
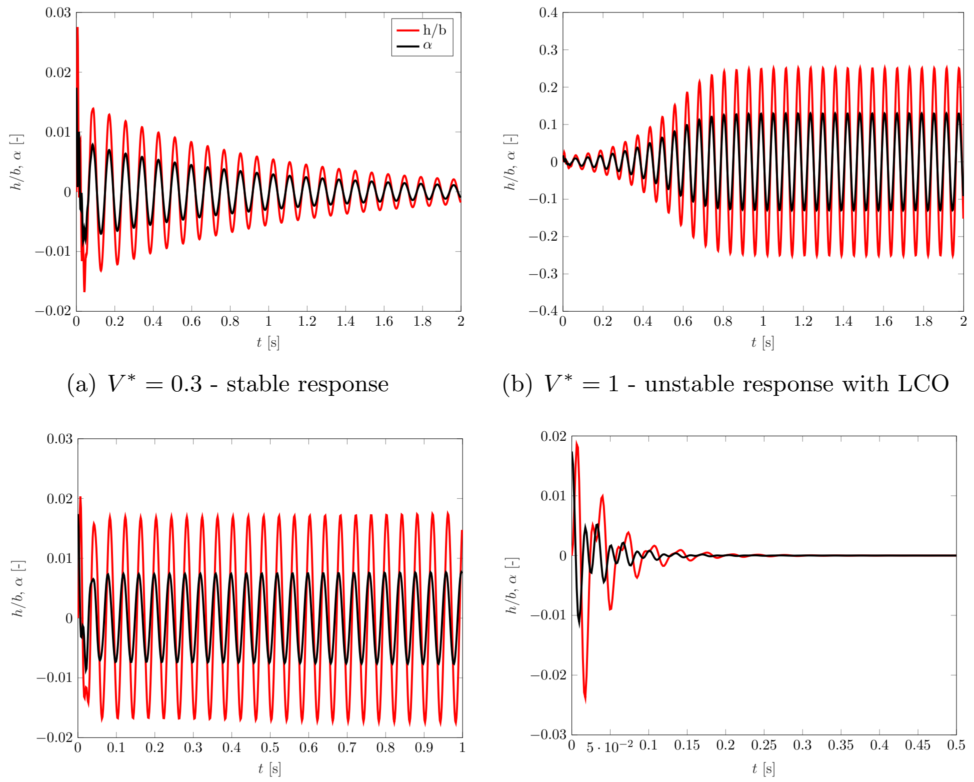


Figure 6.15: Flutter speed index as a function of the free-stream Mach number (flutter boundary). All results from the literature correspond to numerical studies performed with the same flow conditions as the present work.



(a) $V^* = 0.3$ - stable response

(b) $V^* = 1$ - unstable response with LCO

(c) $V^* = 1.84$ - unstable response very close to critical point (nonlinear flutter)

(d) $V^* = 2.5$ - highly stable response

Figure 6.16: Aeroelastic response of the airfoil for different speed indices at M = 0.875.

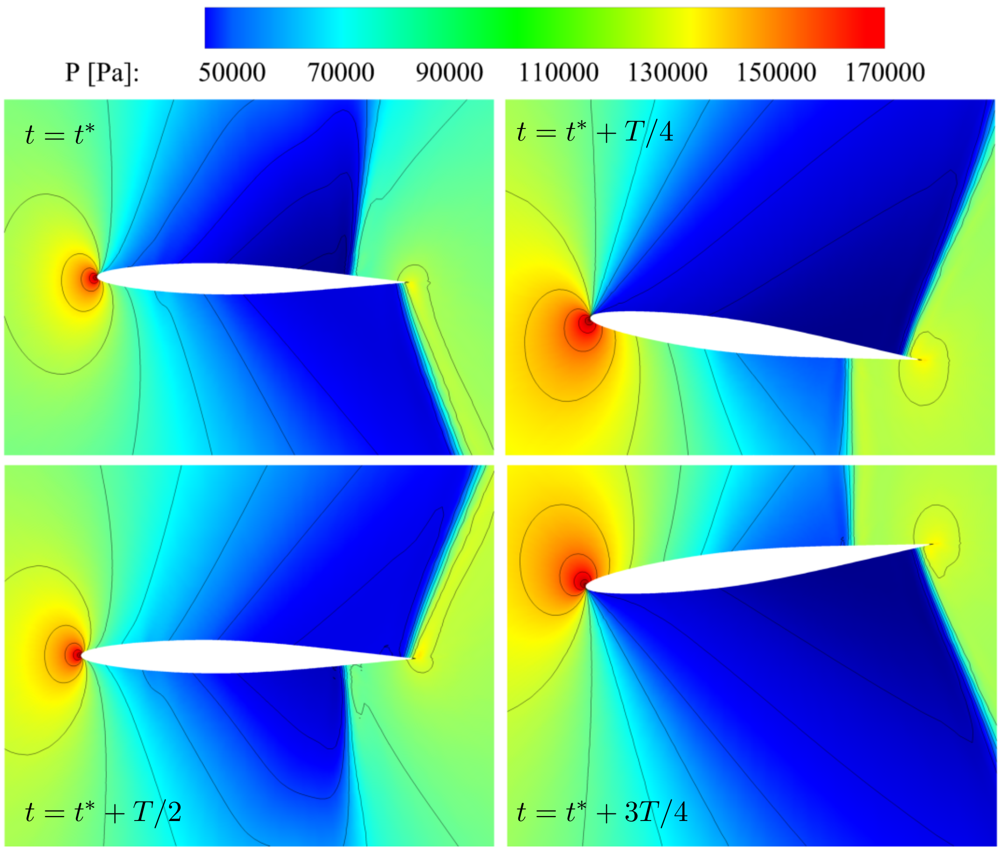


Figure 6.17: Pressure contour around the airfoil during one limit cycle to highlight the motion of the shocks on both sides of the airfoil. M = 0.875 and $V^* = 1$.

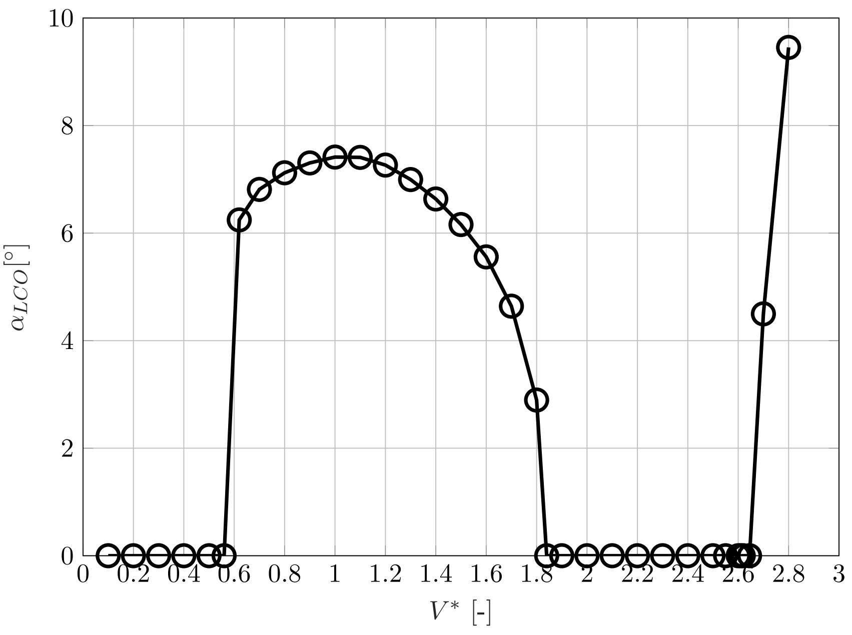


Figure 6.18: Amplitude of the LCO (pitch degree of freedom) as a function of the speed index for M = 0.875. Zero amplitude means no LCO.

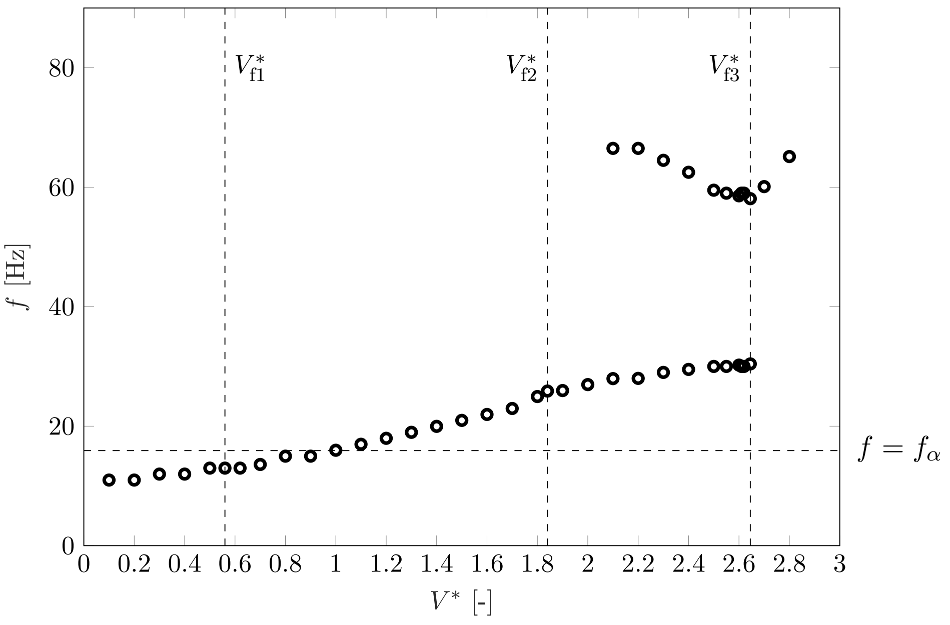
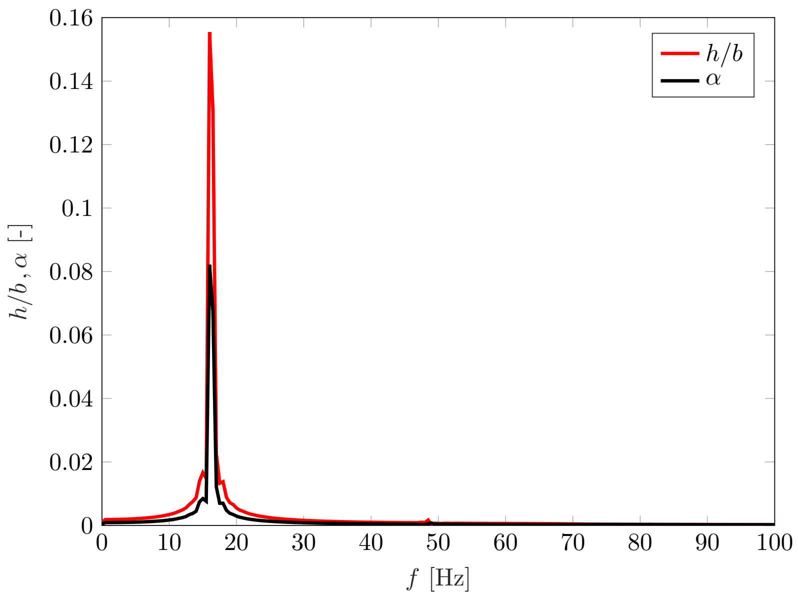


Figure 6.19: Main frequencies of the aeroelastic response as a function of the speed index at M = 0.875.

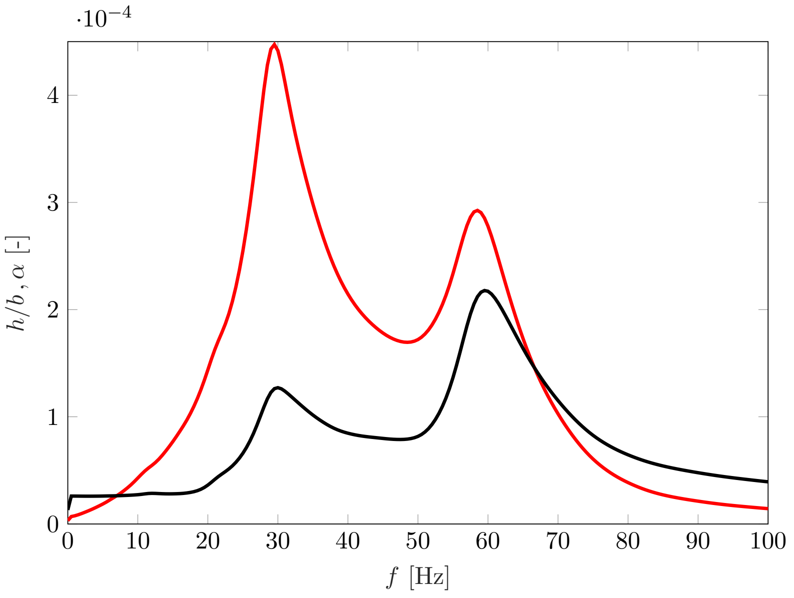
the third flutter point where it reaches a local minimum. Note that the high and low frequencies coexist in this region, as depicted in Fig. 6.20(b). Conversely, in the fourth region (second unstable), the low frequency vanishes and only the high frequency remains in the spectrum.

The aeroelastic response of the airfoil is also characterized by a phase shift between the pitch and plunge modes. As illustrated in Fig. 6.21, this phase shift between the two modes transitions from opposite-phase to in-phase as the speed index is increased. However, it is important to note that the opposite- or in-phase nature of the response depends on the reference for the orientation of the degrees of freedom. Therefore, one should focus on the existence of a transition between one state and the other (phase inversion) rather than on the state itself. The phase inversion tends to occur, in terms of speed indices, between V^*_{f2} and V^*_{f3}. This range typically also corresponds to the occurrence of the high frequency in the response of the airfoil and the range of maximum aeroelastic damping.

The weakly coupled scheme is now tested on the Isogai wing section. The main purpose is to assess the accuracy of explicit coupling schemes in the computation of flutter conditions compared to strong coupling schemes. In this case, the mass ratio is found to be high enough to ensure stable simulations with a weak coupling. Fig. 6.22 illustrates the aeroelastic response of the airfoil at M = 0.875 and $V^* = 0.56$ for both coupling schemes. It can clearly be observed that the weak coupling, although not altering the frequency of the response, does not provide the same level of aeroelastic damping. In this particular case, the aeroelastic damping obtained with the weak coupling is lower than that for the strong coupling so that the response becomes unstable. Tab. 6.6 summarizes the flutter speed indices obtained by the weak coupling. These results are compared to those obtained with strong coupling. The first flutter point for weak coupling occurs at a lower speed index than for the strongly-coupled scheme. Additionally, the two other flutter points cannot be captured with the weak coupling scheme as the response remains unstable for any $V^* > 0.44$. This clearly highlights the significant impact of the coupling scheme on the accuracy of the computed flutter properties of aeroelastic systems, as weak coupling has effectively linearized the system.



(a) $V^* = 1.0$



(b) $V^* = 2.5$

Figure 6.20: Frequency spectrum of the aeroelastic response for two different speed indices at M = 0.875.

	V^*_{f1}	V^*_{f2}	V^*_{f3}
Weak coupling	0.44	/	/
Strong coupling	0.56	1.84	2.645

Table 6.6: Comparison of flutter speed indices computed by strong and weak coupling schemes, M= 0.875.

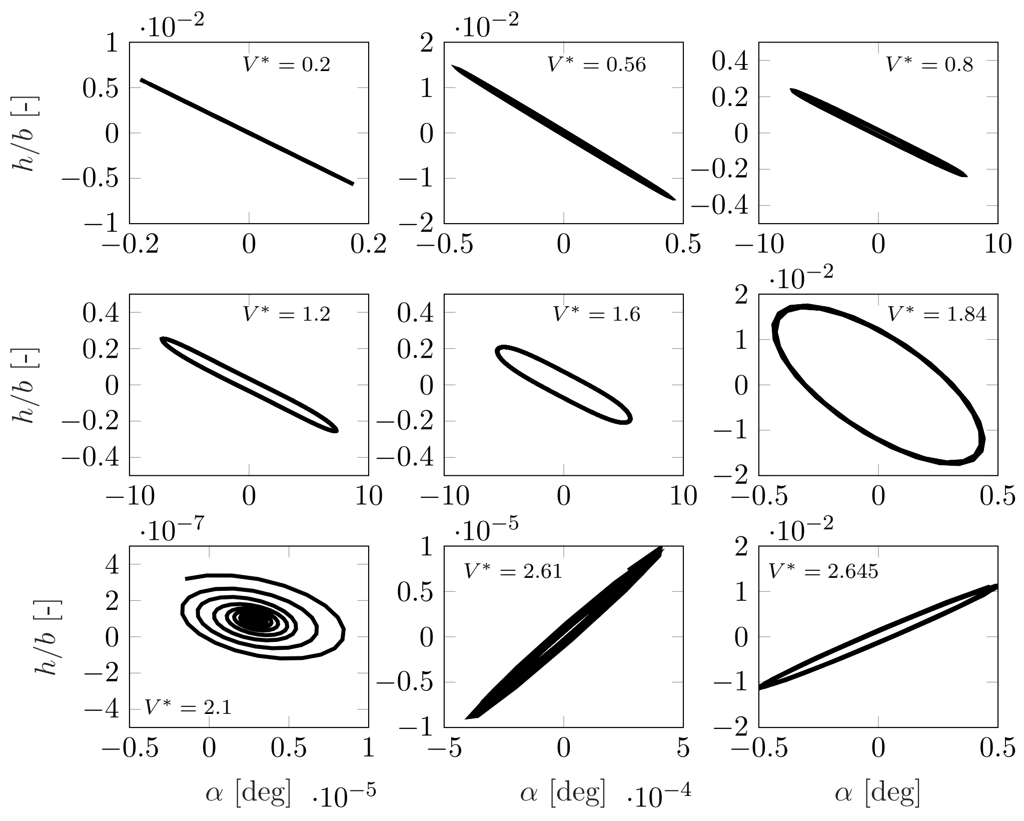


Figure 6.21: Phase shift between the pitch and plunge mode of the airfoil for several speed indices at M = 0.875.

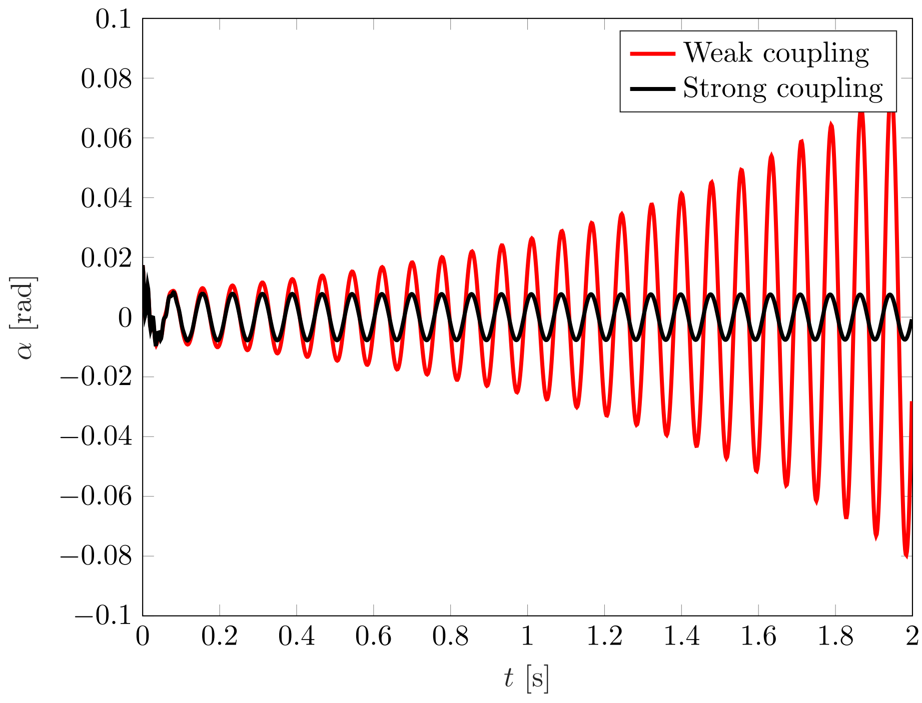


Figure 6.22: Comparison of the aeroelastic response obtained with weak and strong coupling. M = 0.875 and $V^* = 0.56$.

6.2.4 Test case summary

The Isogai wing section test was used to validate the coupling tool CUPyDO for the computation of aeroelastic flutter based on the simplified pitch-plunge airfoil model. This simple structural model provides a useful example of the coupled mode flutter mechanism. In particular, the Isogai wing section features a complex aeroelastic behavior that is characterized by the presence of several flutter points at transonic Mach numbers. In the unstable region, nonlinearities, such as the typical interaction between the moving shocks and the motion of the airfoil, lead to limit cycle oscillations whose amplitude can be related to the speed index.

The analysis of the frequency content of the aeroelastic response has revealed that the airfoil motion is dominated by a low frequency (comparable to f_α) which increases with the speed index. A higher frequency component only appears and becomes dominant when approaching the second flutter region. This change of frequency content is also accompanied by a phase inversion between the plunge mode and the pitch mode of the response.

Finally, it was demonstrated that the use of a weak coupling scheme has a significant influence on the aeroelastic damping and consequently on the flutter speed indices. The first flutter point was found to occur at a lower speed index, which corresponds to a conservative error from a design point of view. However, the weak coupling scheme cannot capture the two other flutter points present around M = 0.875. This analysis clearly highlights the importance of the correction brought by the iterative process of the strong coupling to fulfil the continuity conditions at the fluid-structure interface: a weak coupling is not sufficient for such flutter problems.

6.3 Vortex-induced vibration of a flexible cantilever

The study of the flexible cantilever attached to the downstream side of a perfectly rigid square cylinder is a classical two-dimensional benchmark test case for numerical simulations of FSI problems. A reference for this case can be found in the work of Habchi *et al.* [231]. Compared to the previous test cases, the following benchmark increases the complexity of the modelling as it involves a deformable solid instead of a rigid body motion with a very limited number of degrees of freedom.

This case focuses on the verification of a complete FVM-FEM coupling. It is therefore a first illustration of the versatility of CUPyDO as, from a software architecture point of view, the Metafor solver is simply substituted to the rigid-body integrator with no modification of the interfacing mechanism. High level solver accommodation is performed by calling the proper interfacing module of the *Interface* layer. From an algorithmic point of view, this case is used to evaluate the impact of the predictor order (including zero order i.e., no predictor at all) on the results and the performances of the coupling scheme. Finally, the test case is extended towards low mass ratios in order to intentionally introduce severe added-mass effects. The purpose is then to evaluate the efficiency and the robustness of the under-relaxation capability of the BGS coupling algorithm, and to highlight its limitations.

6.3.1 Case description and simulation parameters

The geometry of the computational domain and the boundary conditions are described in Fig. 6.23. In this case the chord of the square cylinder is $H = 0.01$ m. The physical properties of the solid and fluid are summarized in Tab. 6.7. The uniform incoming flow velocity is $U = 0.513$ m/s, which corresponds to a Reynolds number $\mathrm{Re} = UH/\nu_\mathrm{f} = 333$. The top and bottom sides of the domain are modeled as inviscid walls whereas a no-slip condition is imposed on solid boundaries (square cylinder and cantilever). The velocity and Reynolds number are such that an unsteady laminar Von Karman vortex street is generated in the wake of the cylinder with a well-defined shedding frequency. Therefore, the vortical structure of the wake generates harmonic aerodynamic loads that induce periodic oscillations of the flexible cantilever.

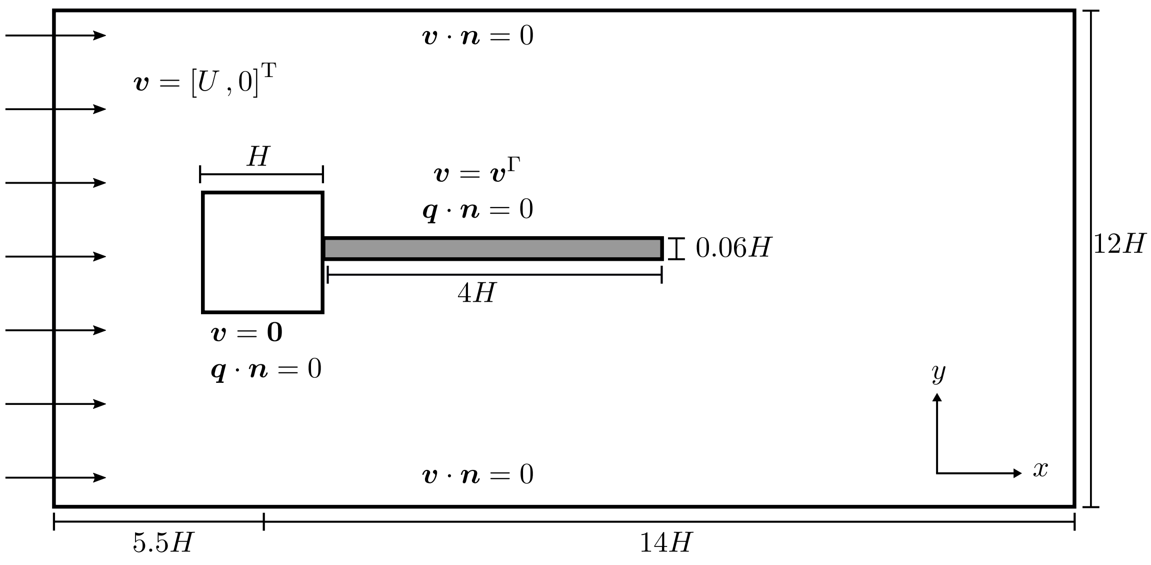

$\boldsymbol{v}\cdot\boldsymbol{n}=0$
$\boldsymbol{v}=[U\,,0]^{\mathrm{T}}$
H
$\boldsymbol{v}=\boldsymbol{v}^{\Gamma}$
$\boldsymbol{q}\cdot\boldsymbol{n}=0$
$0.06H$
$12H$
$4H$
$\boldsymbol{v}=\boldsymbol{0}$
$\boldsymbol{q}\cdot\boldsymbol{n}=0$
y
x
$\boldsymbol{v}\cdot\boldsymbol{n}=0$
$5.5H$
$14H$

Figure 6.23: Fluid domain geometry and boundary conditions for the flexible cantilever in the wake of a square cylinder ($H = 0.01$ m).

Solid	Density [kg m^{-3}]	ρ_{s}	100
	Young's modulus [Pa]	E	$2.5 \cdot 10^5$
	Poisson's ratio [-]	ν_{s}	0.35
Fluid	Density [kg m^{-3}]	ρ_{f}	1.18
	Kinematic viscosity [m^2 s^{-1}]	ν_{f}	$1.54 \cdot 10^{-5}$

Table 6.7: Flexible cantilever attached to a rigid square cylinder: physical properties of the solid and fluid.

The SU2 solver is used to solve the compressible laminar Navier-Stokes equations on the fluid domain which is discretized by a hybrid structured-unstructured grid with 15449 grid points. As illustrated in Fig. 6.24, the mesh is globally unstructured with a structured layer near the solid boundary. Convective fluxes are evaluated using the second-order centered JST scheme and time integration is performed with a second-order Euler implicit dual-time stepping scheme. The cantilever is modeled as pure elastic material and discretized with 240 $\times$ 10 (length $\times$ thickness) quadrilateral EAS elements. The solid problem is solved with the nonlinear Metafor solver. In order to eliminate any interpolation error, discretization is performed in both domains so as to have matching meshes at the fluid-structure interface.

The reference simulation parameters are summarized in Tab. 6.8. The chosen time step size corresponds to 122 time steps per period for the first bending mode of the beam and the chosen coupled tolerance represents 10^{-4} times the expected tip displacement of the cantilever. The simulation is performed starting from a uniform flow and no initial deformation of the cantilever.

Before computing the fully coupled system as described here above, preliminary studies are performed for each physics separately. The flow around a square cylinder in the absence of the cantilever is computed in order to validate the numerical settings of the flow solver. A simulation with a perfectly rigid cantilever is performed as well to be compared with the flexible (and coupled) case. The solid solver Metafor is also evaluated by computing the natural frequencies of the beam and by computing its wind-off free response.

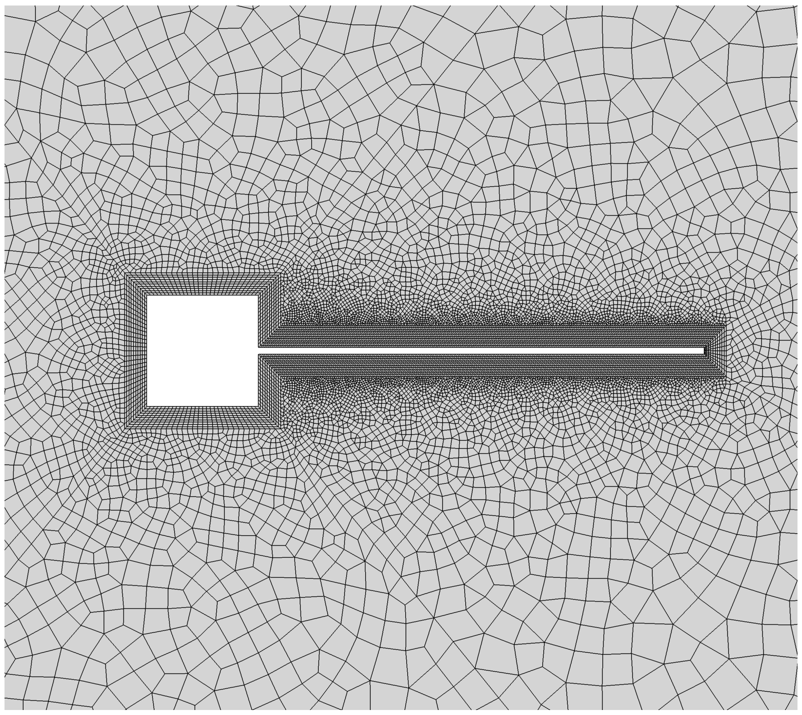


Figure 6.24: Mesh in the vicinity of the square cylinder with flexible cantilever.

Time step [s]	Δt	0.0025
Simulated physical time [s]	t_{tot}	15
Maximum number of coupling iteration per time step	$\hat{n}_{\mathrm{FSI}}$	6
Coupling tolerance [m]	ε	10^{-6}

Table 6.8: Simulation parameters for the VIV flexible cantilever.

6.3.2 Results

Results for the single physics preliminary computations are first presented in order to validate the numerical settings and the discretization of each domain. Then the fully coupled study is presented and reproduced with decreasing mass ratios to assess the performance of the coupling in presence of high added-mass effects.

Flow around a square cylinder with and without rigid cantilever

The results for the fluid computations involving perfectly rigid bodies are here presented. The simulation in the absence of the cantilever gives the following values for the RMS lift coefficient, mean amplitude of the drag coefficient and the Strouhal number: $C_{l,\mathrm{RMS}} = 0.82$, $\overline{C_d} = 1.94$ and $\mathrm{Str} = 0.13$, respectively. The mean drag and the Strouhal number computed in the present study are compared in Tab. 6.9 to other numerical and experimental data that can be found in the literature (e.g. [232–235]). The computed Strouhal number is in good agreement with the available experimental values (no numerical data available in Yuce and Kareem for the Strouhal number). However, the drag coefficient is over-predicted by the numerical simulations compared with the experiments. While, Yuce and Kareem did not provide any explanation on the origin of the over-predicted drag coefficient, the discrepancy can be related to the blockage effect which is already known to have an impact on the drag coefficient, as stated in Sohankar *et al.* [236]. The blockage coefficient has been found to be 12.5% in the case of Yuce and Kareem [235]. In

	Str	$\overline{C_d}$
Present (num. FVM laminar)	0.13	1.9
Yuce and Kareem [235] (num. FVM RANS SST k-ω)	/	2.1
Okajima [232, 233] (exp.)	0.12	1.6
Yen [234] (exp.)	0.12	1.7

Table 6.9: Flow around a square cylinder: Strouhal number and drag coefficients for a flow around a square cylinder: comparison between current computed values and various references (both experimental and numerical).

the present case, the blockage coefficient is 8% while it is reported to be much smaller, less than 4%, for the experiment of Okajima [232, 233].

When computing the fluid flow around the cylinder in the presence of the rigid cantilever, a significant modification of the vortex shedding and of the wake pattern behind the square is observed. Fig. 6.25 compares the two configurations (with and without cantilever) by showing the velocity vector field at the time corresponding to maximum lift. In the presence of the

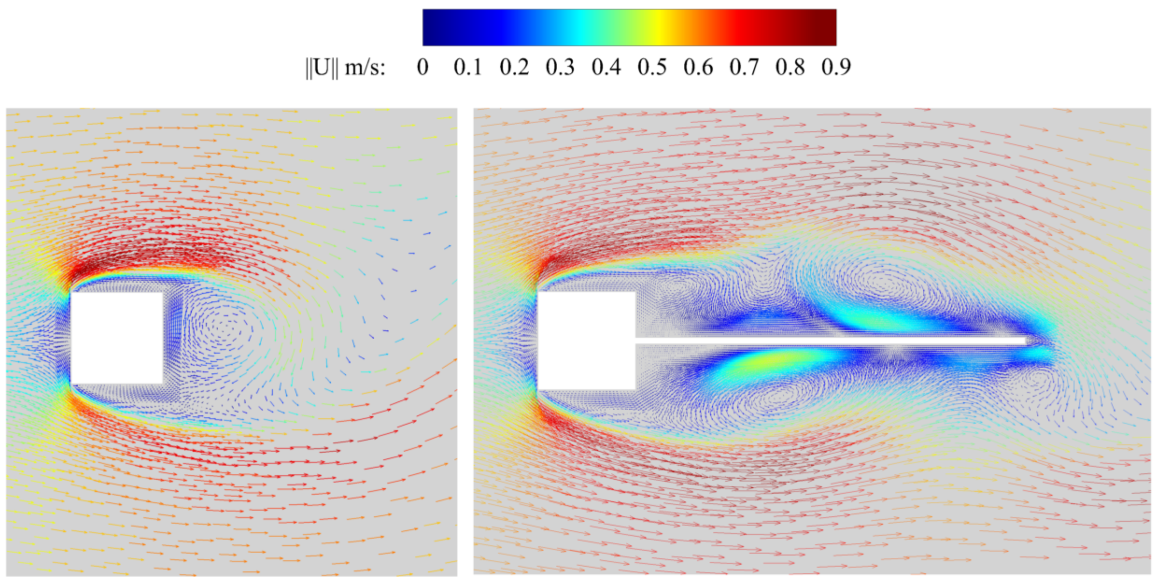


Figure 6.25: Velocity field of the computed flow around the square cylinder, left without rigid cantilever and right with rigid cantilever. Instantaneous view corresponding to the maximum lift.

cantilever, the shed vortices can only move horizontally along the cantilever before reaching the free wake. This decreases the frequency of the shedding from 6.8 to 5.8 Hz. This represents a decrease of 15% in terms of the Strouhal number, from 0.13 (without cantilever) to 0.11 (with cantilever).

Frequency analysis of the cantilever and wind-off free response

A linear modal analysis using the frequency analysis module of Metafor is performed on the flexible cantilever. The computed natural frequencies are compared to those given by the analytical solution for an equivalent 1D beam model [237],

$$2\pi f_i = \{1.875,\, 4.694,\, 7.855,\, 10.995\}^2 \sqrt{\frac{EI}{\rho_s A L^4}} \qquad i = 1, \cdots, 4\,, \tag{6.24}$$

for the first four natural frequencies. In this expression, I is the flexural inertia and A the cross-sectional area (considering an out-of-plane unit length). The computed natural frequencies are also compared to the results obtained with an equivalent modal analysis in the solver GetDP, in which the two classical assumptions for 2D problems, i.e. plane-stress (p$-\sigma$) or plane-strain (p$-\varepsilon$), are considered. Computations with Metafor can only be performed with the p$-\varepsilon$ assumption, as there is no p$-\sigma$ formalism directly available. However, both standard (STD) and enhanced assumed strain (EAS) elements are used in order to assess their impact on the natural frequencies. The results are compiled in Tab. 6.10 for the first four natural frequencies. The

1D beam model	GetDP $p-\sigma$	GetDP $p-\varepsilon$	Metafor $p-\varepsilon$ STD	Metafor $p-\varepsilon$ EAS
3.02	3.07	3.28	3.28	3.23
18.98	19.25	20.55	20.55	20.25
53.15	53.81	57.43	57.43	56.61
104.15	105.19	112.25	112.25	110.64

Table 6.10: Comparison between computed and analytical values for the first four natural frequencies of the cantilever. Frequencies are expressed in Hz.

plane-stress computation with GetDP shows very good agreement with the 1D beam model, which was established under the same assumption. When using the plane-strain assumption, the frequencies computed by GetDP and Metafor with standard elements are the same. The use of EAS elements in Metafor leads to slightly lower frequencies (a decrease of around 1.45% for all frequencies) as this particular type of elements is known to be less stiff than its standard counterpart. EAS elements are used for all subsequent computations as they provide an improved behaviour for bending dominated problems (see discussion in Section 3.7.1).

The free response of the cantilever is computed by an unsteady simulation. The response is obtained by applying a dead load at the tip of the cantilever during a short period of time (10 time steps) and by releasing it. The cantilever is then free to vibrate. Note that the dead load has been calibrated to obtain a maximum displacement that is close to the expected maximum displacement obtained in the coupled system. Fig. 6.26 illustrates the vertical displacement d_y of the cantilever tip as a function of time. The measured frequency of the response is 3.2 Hz which corresponds to the first natural frequency (first bending) obtained from modal analysis.

Fully coupled analysis

Coupled simulations are performed with the strongly-coupled BGS schemes without relaxation and with the use of the second-order predictor. Fig. 6.27 shows the computed tip displacement as a function of time. At the start of the simulation, a transient behavior is observed until the vortex shedding, and consequently the tip displacement, reaches an established regime where the displacement amplitude is modulated by lower frequency waves. This stems from the complex structure of the vortex shedding interacting with the large displacement of the cantilever.

A summary of results from the literature (e.g., [238–240]) is provided by Habchi *et al.* [231]. The oscillation frequency typically falls in the range 2.94$-$3.25 Hz, while the maximum amplitude of the tip displacement is in the range $0.95 - 1.15$ cm. Tab. 6.11 compares the present results with those from the literature. Note that all the authors used a partitioned BGS approach with a FVM-FEM coupling, except for Olivier *et al.* [240] who used a FVM-FVM coupling. The present computation predicts a maximum tip displacement $d_y = 1.14$ cm and a frequency

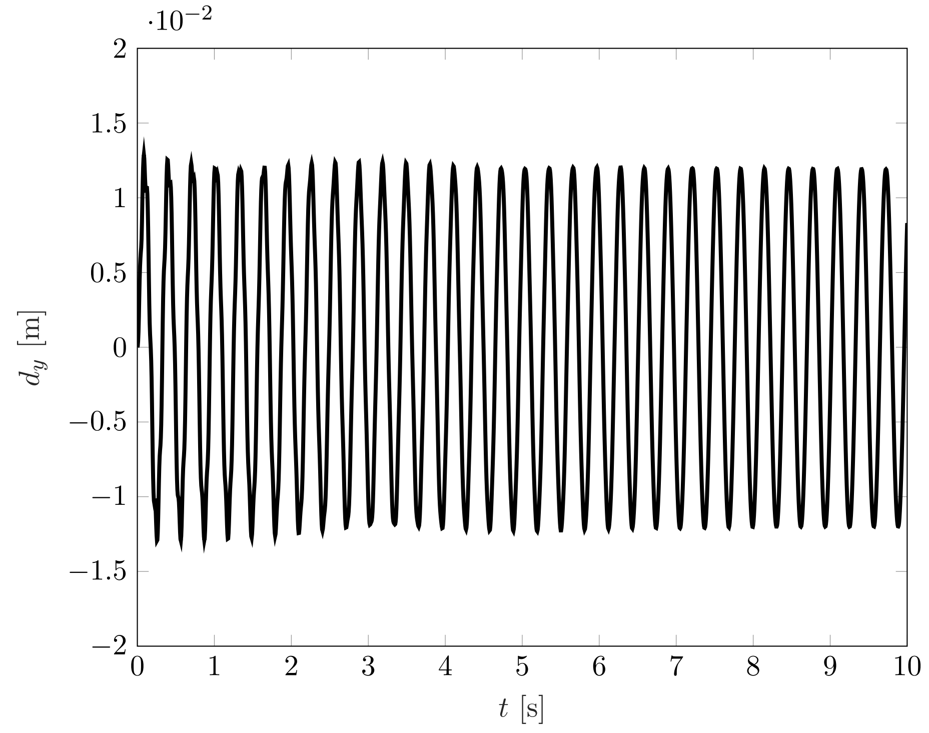


Figure 6.26: Vertical displacement of the cantilever tip as a function of time. Uncoupled free response.

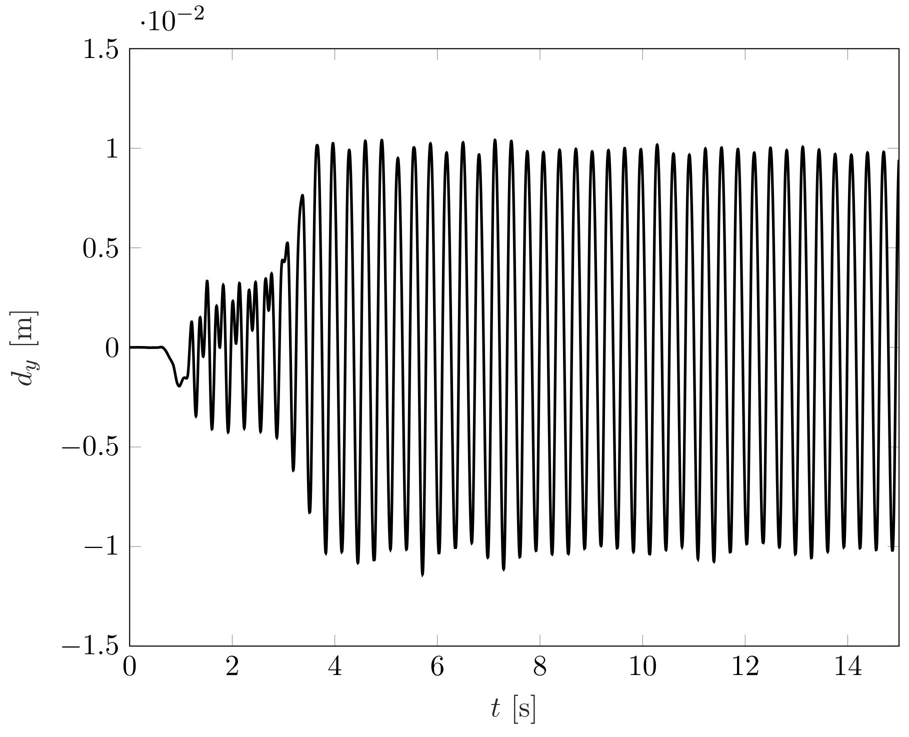


Figure 6.27: Vertical displacement of the cantilever tip as a function of time. Coupled vortex-induced response.

	max d_y [cm]	f [1/s]
CUPyDO	1.14	3.20
Sanchez *et al.* [241]	1.05-1.15	3.05-3.15
Habchi *et al.* [231]	1.02	3.25
Kassiotis *et al.* [238]	1.05	2.98
Wood *et al.* [239]	1.15	2.94
Olivier *et al.* [240]	0.95	3.17

Table 6.11: Comparison of the maximum tip displacement and oscillation frequency of the flexible cantilever between the present computation and results from the literature. The range of values obtained by Sanchez *et al.* corresponds to a parametric study on the relaxation parameter in the BGS algorithm.

$f = 3.20$ Hz, which falls within the range of results from the literature. The tip displacement corresponds to 28% of the length of the cantilever, and is thus considered as a large displacement. Fig. 6.28 shows the velocity magnitude contour at several time steps of a period T. The vortical flow structures moving along the deforming cantilever are ejected into the wake where they are slowly dissipated by viscous effects.

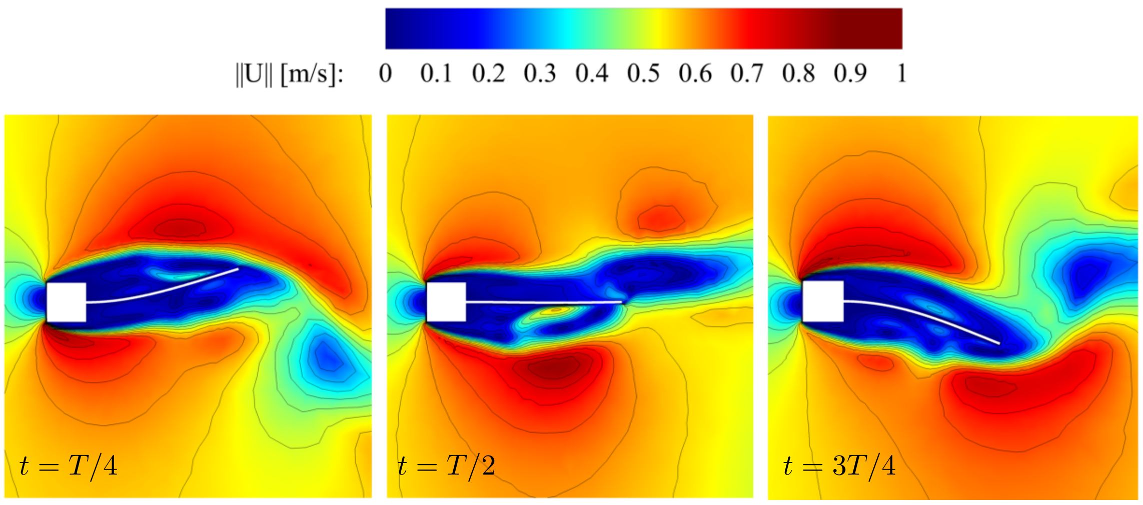


Figure 6.28: Flexible cantilever attached to a rigid square cylinder: velocity magnitude contour at three phases of a period.

Simulations under the same conditions are now performed with a first-order predictor and without predictor. Tab. 6.12 summarizes some general results. As expected, the most noticeable effect of the use of the predictor is a reduction of the mean number of coupling iterations per time step, while reaching similar mean values for the final coupling error at every time step.

The test case is now extended to cover lower mass ratios. So far the mass ratio for the computations was

$$\mathrm{Ma} = \frac{\rho_\mathrm{s}}{\rho_\mathrm{f}} = 84.7\,, \tag{6.25}$$

which is sufficiently high to get a numerically stable coupled computation without using under-relaxation. In this analysis, several additional mass ratios are tested, Ma = 8.5, 4.2, 1.7 and 0.8, by reducing the solid density (ρ_s = 10, 5, 2 and 1 kg/m^3 respectively) while keeping the ratio

	$\overline{N}_{\mathrm{FSI}}$	$\overline{\varepsilon}$ [m]
second-order	2.75	$6.47 \cdot 10^{-7}$
first-order	3.72	$5.56 \cdot 10^{-7}$
no predictor	5.51	$6.42 \cdot 10^{-7}$

Table 6.12: Summary of results obtained with different prediction orders: mean number of coupling iterations per time step and mean value of the final coupling error per time step

E/ρ_s, and thus the natural frequencies of the cantilever, constant. For each mass ratio, the effect of the predictor, the relaxation method and the relaxation parameter are assessed with respect to the mean number of FSI coupling iterations per time step. The results are summarized in Tab. 6.13. As expected, the convergence rate and numerical stability of the coupled simulations

Ma	Predictor	Relaxation	$\overline{\omega}$	$\overline{N}_{\mathrm{FSI}}$
8.5	Second-order	Static	1	X
	Second-order	Static	0.8	X
	Second-order	Static	0.5	8.3
	Second-order	Static	0.2	16.3
	Second-order	Aitken max	0.5	7.03
	Second-order	Aitken max	0.1	6.9
	No	Aitken max	0.1	8.8
4.2	Second-order	Static	0.5	X
	Second-order	Static	0.1	37.8
	Second-order	Aitken max	0.5	7.6
	Second-order	Aitken max	0.1	7.2
	Second-order	Aitken min	0.1	8.4
	No	Aitken max	0.5	10.5
1.7	Second-order	Aitken max	0.5	13.4
	Second-order	Aitken max	0.1	13.3
	Second-order	Aitken min	0.1	13.1
	No	Aitken min	0.1	15.9
0.8	Second-order	Aitken min	0.1	X
	No	Aitken min	0.1	31.9

Table 6.13: Flexible cantilever attached to a rigid square cylinder: analysis of the efficiency of the coupling algorithm with decreasing density ratio. $\overline{\omega}$ is the value of the relaxation parameter in case of static relaxation. In the case an Aitken's relaxation, it corresponds to the fixed value used in criterion Eq. (3.23). $\overline{N}_{\mathrm{FSI}}$ is the average number of coupling iterations per time step. The X symbol means that the coupling process diverges.

decrease significantly when reducing the mass ratio. This is clearly highlighted by a general increase of the mean number of coupling iterations. While no relaxation was used for the initial mass ratio in this study, stabilization by under-relaxation becomes more and more essential as Ma decreases. Without any relaxation, the simulation quickly diverges after a few time steps (denoted by the X symbol in Tab. 6.13).

For Ma = 8.5, the use of a static relaxation is sufficient to stabilize the iterative procedure if the value of the relaxation constant is below a certain limit (between 0.5 and 0.8). Note that in this study, the purpose is not to seek the exact limit of the relaxation constant that leads to convergence, as it would have required a significant number of computations. The purpose is rather to identify the general performance trends of the coupling algorithm. Low values of the relaxation parameter (e.g. $\overline{\omega} = 0.2$) bring excessive amount of relaxation, and thus lead to a significant increase in the number of coupling iterations. Using Aitken's relaxation with the max criterion leads to a further decrease in the mean number of coupling iterations and, unlike the static relaxation, the dynamic adaptation of ω performs slightly better with low values of $\overline{\omega}$. Finally, and as already observed, the absence of time step prediction slightly increases the mean number of coupling iterations.

For Ma = 4.2, static relaxation requires lower values of the relaxation parameter (e.g. $\overline{\omega} = 0.1$) to provide a convergent coupling, but it also drastically increases the number of coupling iterations. Using Aitken's relaxation here guarantees a convergent coupling process and efficiently decreases the number of coupling iterations. However, this number globally increases compared to the previous mass number using the same values of $\overline{\omega}$. The use of the *min* criterion is introduced, but it does not outperform the *max* criterion yet for this mass number. The same trend regarding the absence of time step prediction, i.e. an increase of the mean number of coupling iterations compared to cases with prediction, is still observed.

For Ma = 1.7, a significant increase in the mean number of iterations is observed for any relaxation technique and relaxation parameter. In this case, the min criterion provides a substantial reduction of the number of iterations compared to the max criterion.

Finally, the case Ma = 0.8 is so ill-conditioned because of the strong added-mass effects that only the min criterion with a low value of the relaxation parameter can provide a stable iterative coupling process, although the significant mean number of coupling iterations becomes prohibitive for practical applications. We also note that, in this case, the presence of the predictor acts as a destabilizing factor. This is most likely due to the higher sensivity of the coupling process to the guessed position of the solid interface at the beginning of each time step. Thus only an iterative coupling without time step prediction leads to convergence. However, it is important to note that the time step has been kept constant during the study and might need to be adapted to better represent the physics at lower mass ratio. When the flow changes rapidly, the extrapolation provided by the predictor leads to an overprediction that cannot be stabilized by the iterative coupling process. It is expected that a lower time step will bring back the benefit of using the predictor.

Fig. 6.29 shows the fluid velocity magnitude contour and the structural deformation for different mass ratios (Ma = 42.3, 4.2 and 0.8). The corresponding time for each plot is selected so as to correspond to the maximum amplitude of the deformation modes. One can clearly observe that the order of the deformation mode of the cantilever increases as the mass ratio decreases. However, this behavior is most likely due to increased beam flexibility (the stiffness has been decreased to maintain the ratio E/ρ_s constant) than the decreased mass ratio.

6.3.3 Test case summary

This test case was the first for which CUPyDO was used to couple a FVM fluid solver with a FEM structural solver. The substitution of the simple rigid body integrator by Metafor was performed in a straightforward fashion by taking advantage of the interfacing flexibility of the coupling tool.

The coupling was tested on the classical and widely studied case of the cantilever attached in the wake of a square cylinder. The solution computed for the reference case was shown to be in good agreement with results found in the literature.

The reference case was then adapted in order to purposely introduce added-mass effects and numerical instability, so as to assess the performance of the under-relaxation feature of the coupling algorithm. The different studies with decreasing mass ratios led to the following observations, that could be seen as best practice when using BGS coupling for cases with significant added-mass:

- static under-relaxation is the simplest option to use, but may lead to a relatively large

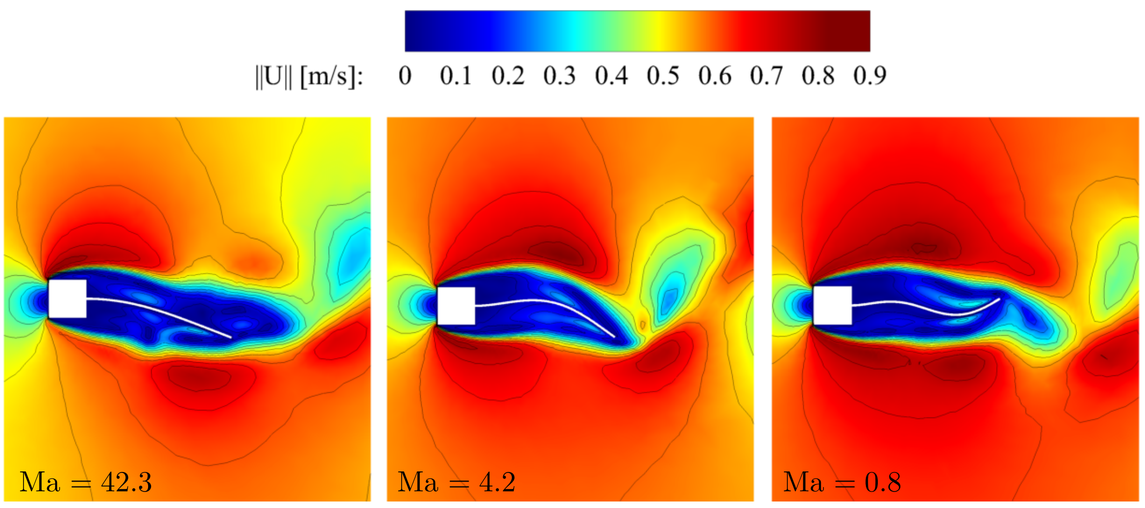


Figure 6.29: Flexible cantilever attached to a rigid square cylinder: velocity magnitude contour and deformation mode for three different mass ratios at the time of maximum deformation.

number of coupling iterations if the relaxation parameter is not optimal,

- Aitken's under-relaxation usually outperforms the static one,
- when using Aitken's under-relaxation, it is preferable to use the *max* criterion (provided it produces a stable coupling) as it leads to a lower number of coupling iterations than the *min* criterion,
- when the *max* criterion of Aitken cannot lead to a convergent coupling process, the *min* criterion is used to stabilize it,
- in general, using a (second-order) predictor can reduce the number of coupling iterations, but the predictor can be disabled to stabilize the coupling process in case of very strong added-mass effects, as in this situation the predictor acts as a destabilizing effect. However, a reduction of the time step will probably bring back the improvement of the time predictor.

6.4 Aeroelastic study of the AGARD 445.6 wing

The experimental AGARD 445.6 wing test case of Yates [242] is a frequently used three-dimensional validation case for transonic flutter simulations. It is therefore used to assess the three-dimensional coupling capabilities of CUPyDO. As the size of the meshes increases, parallelization is used to save substantial computational time (particularly for the fluid computation). The parallel coupling capabilities of CUPyDO are thus also evaluated, although no formal parallel scalability analysis is performed in this work.

Additionally, a non-matching mesh discretization is used at the fluid-structure interface. Consequently, RBF interpolation is used to transfer displacements and loads between the coupled solvers. Evaluation of the interpolation accuracy for both TPS and CPC2 basis functions is performed.

6.4.1 Case description and simulation parameters

The present computational study is based on the weakened model 3 of the wing [242]. This is a 45° swept-back wing whose geometry is depicted in Fig. 6.30 and whose geometrical properties are summarized in Tab. 6.14. The cross-section is a symmetric NACA 65a004 airfoil and the wing is clamped at the root. The fluid flow is considered inviscid and the compressible Euler

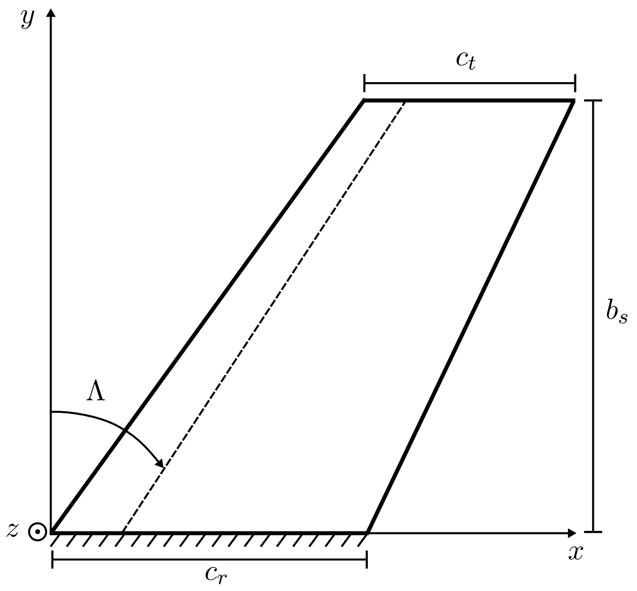


Figure 6.30: Geometry of the AGARD 445.6 wing.

Root chord [m]	c_r	0.559
Taper ratio [-]	λ	0.658
Tip chord [m]	c_t	0.368
Semi-span [m]	b_s	0.762
Aspect ratio [-]	AR	1.644
Sweep at quarter chord [°]	Λ	45
Wing surface [m^2]	S	0.353
Mean aerodynamic chord [m]	$\bar{c}$	0.470

Table 6.14: Geometrical properties of the AGARD 445.6 wing.

equations are thus solved. The fluid region of the problem is discretized using a structured O-mesh with a total number of 257521 grid nodes (a mesh convergence analysis has been performed to determine the final grid size). The fluid domain extends up to 25 times the root chord c_r from the wing in each direction. Farfield boundary conditions are applied on the outer boundary while a symmetry boundary condition is imposed on the plane in which the wing is clamped. The wing surface is discretized with 30, 80 and 20 cells in the spanwise, chordwise and thickness[5] direction respectively. The mesh around the wing is illustrated in Fig. 6.31. The SU2 solver is used to compute the flow: convective fluxes are evaluated using the second-order centered JST scheme and time integration is performed with the second-order Euler implicit dual-time stepping scheme.

The solid wing is modeled in Metafor with 8-node continuum EAS elements and an orthotropic elastic material whose properties are summarized in Tab. 6.15. It is discretized with 31, 17 and 2 cells in the spanwise, chordwise and thickness direction, respectively. EAS elements are used to conserve high accuracy in the structural solution while limiting the number of elements in the thickness of the wing.

[5]This refers to the number of cells used to discretize the rounded wing tip that is used here.

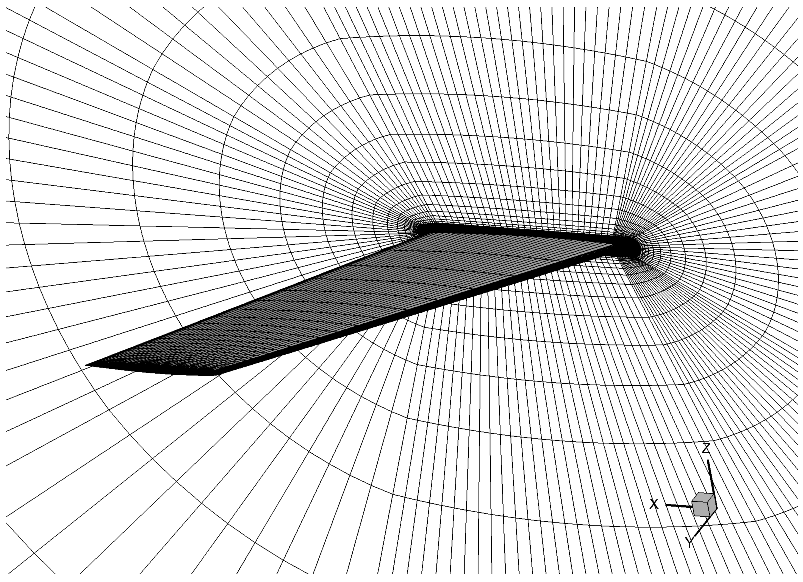


Figure 6.31: CFD mesh around the AGARD 445.6 wing.

Longitudinal Young's modulus [GPa]	E_1	3.151
Transverse Young's moduli [GPa]	E_2, E_3	0.4162
Shear moduli [GPa]	G_{12}, G_{13}, G_{23}	0.4392
Poisson's ratio [-]	$\nu_{12}, \nu_{13}, \nu_{23}$	0.31
Density [kg m^{-3}]	ρ_s	381.98

Table 6.15: Material properties for the AGARD 445.6 wing.

Strongly-coupled simulations are used to compute the flutter boundary of the wing. Similarly to the experimental investigation, a large range of Mach numbers, from M= 0.499 to M= 1.141, is simulated. Based on experimental conditions, the corresponding Reynolds numbers are in the range $\mathrm{Re} = 0.54 \cdot 10^6$ to $\mathrm{Re} = 1.89 \cdot 10^6$. As for the Isogai wing section test case, computations are performed at each Mach number for several speed indices and flutter is inferred from the damping coefficient extracted from the aeroelastic response. The speed index for the AGARD 445.6 wing test case is defined as

$$V^* = \frac{U}{0.5 c_r \omega_2 \sqrt{\mathrm{Ma}}}, \tag{6.26}$$

where c_r, ω_2 and Ma are the root chord, the natural frequency of the first torsional mode and the mass ratio, respectively. The mass ratio is given by $\mathrm{Ma} = m/\rho_\mathrm{f} V$, where $m = 1.863$ kg and $V = 0.130$ m^3 are directly taken from the work of Yates [242].

As the discretization of the interface is not matching, the mesh interpolator of the coupling tool is used to map the two interface meshes and to communicate the data. Preliminary steady FSI simulations are performed for assessing the performance of the different mesh interpolation methods for this case and in order to select the best-adapted method for the flutter case.

Flutter simulations are performed with a time step of 0.001 s ($\approx$ 105 time steps per period of the first bending mode) with no relaxation on the BGS coupling (value of Ma is in the range 33-260 depending on the fluid density). In order to limit the computation time of the simulation, a maximum number of four coupling iterations is imposed which is typically sufficient to reach a prescribed tolerance of 10^{-7} m (about 10^{-5} times the expected displacement amplitude at flutter). Simulations are performed in parallel on 16 cores (16 fluid instances, 1 solid) of a computing node with Intel Xeon E5-2650 processor (2 GHz, 16 threads).

Each simulation is initialized with a uniform flow and no deformation of the solid wing. During the first 0.01 s of simulated time, a vertical load is applied on an upstream portion of the wing tip in order to induce a small perturbation of a determined amplitude (around 0.66 % of the span). Then the loading is released and the wing is free to vibrate in the flow.

Preliminary fluid and solid simulations are performed to validate the mesh and the numerical settings. The steady transonic flow around a perfectly rigid wing at M = 0.96 and $\rho_\mathrm{f} = 0.0634$ kg/m^3 is computed with SU2 on several meshes with varying discretizations, as summarized in Tab. 6.16, and varying angles of attack (1, 2.5 and 5 degrees.) Then, a linear modal analysis of

	Spanwise	Chordwise	Thickness	Total number
Mesh A	15	40	10	46500
Mesh B	30	80	20	248000
Mesh C	45	95	20	410000
Mesh D	60	120	20	679280

Table 6.16: Mesh discretizations used for the preliminary fluid simulations. Number of wing surface cells and total number of cells (whole mesh).

the wing (solid domain) is performed in Metafor and compared to other numerical data found in the literature. Finally, as already introduced above, steady FSI computations are also performed to evaluate the performance of the different interpolation methods. The TPS and CPC2 (with several radii) basis functions are tested using a conservative approach. The flow conditions for the steady FSI study are taken from the work of Goura [243]: free-stream Mach number M= 0.8, angle of attack of 1°, free-stream velocity $U = 247.09$ m/s and free-stream fluid density $\rho_\mathrm{f} = 0.09411$ kg/m^3.

6.4.2 Results

The results of the preliminary studies (single physics and steady FSI) are first presented. They are then followed by the results of the flutter analysis.

Fluid simulations with a rigid wing

The results obtained with the single-physics steady fluid computations are depicted in Tab. 6.17 in terms of the lift and moment coefficients as a function of the angle of attack for the four meshes considered. The force coefficients are not significantly impacted by the mesh discretization for

	AoA = 1°		AoA = 2.5°		AoA = 5°	
	C_L	C_M	C_L	C_M	C_L	C_M
Mesh A	0.0802	-0.0882	0.2001	-0.2214	0.405	-0.4608
Mesh B	0.0775	-0.0841	0.1981	-0.2176	0.407	-0.4641
Mesh C	0.0806	-0.088	0.199	-0.2195	0.4089	-0.4672
Mesh D	0.081	-0.0892	0.1991	-0.2201	0.4093	-0.4675

Table 6.17: Lift and moment coefficients as functions of the angle of attack for the four fluid meshes. Preliminary steady fluid simulations with rigid wing.

the inviscid fluid model used in this case. Over all tested angles of attack, the mean differences between the maximum and minimum values regarding the mesh discretization are only 2.1% and 2.9% for the lift and moment coefficients, respectively (relative values with respect to the finest mesh).

Based on the results obtained from these preliminary fluid simulations, mesh B is judged accurate enough while keeping an affordable CPU cost. This trade-off between accuracy and computational cost is important because the cost of FSI simulations will be undoubtedly much higher than that of the present steady simulations. A reasonable mesh is thus necessary to ensure that unsteady FSI simulations remain feasible. One iteration on mesh B takes about half the time of an iteration on the finest mesh, mesh D. Therefore, mesh B is retained for all following computations.

Structural analysis of the wing

In order to asses the validity of the numerical settings of the solid solver, a modal analysis of the wing is performed in Metafor. The first four computed natural frequencies are compared with results from the literature in Table 6.18, showing good agreement with models coming from other references. Note that in these references, the structural model is based on plate elements instead of the volume elements used in this work.

Evaluation of the RBF interpolation on a steady aeroelastic case

The results for the steady FSI simulations are presented in Tab. 6.19 which shows the z-displacement (perpendicular to the wing plane) of the wing tip at both leading and trailing edges. The values of the support radius for the CPC2 function used in the present study are $r = 0.1$, 0.2, 0.4 and 0.7 which corresponds to 13.1%, 26.2%, 52.4% and 91.8% of the span, respectively. The computed values of the displacement of the wing tip are compared with the results obtained by Goura [243], who introduced the Constant Volume Tetrahedron (CVT) interpolation method, and Melville *et al.* [246] on the same test case. The results clearly show that, in this case, the impact of the basis function and the support radius on the displacement is marginal. The results are also in good agreement with the references, with a maximum discrepancy of 4.1% and 8.5% on the leading and trailing edge displacement, respectively. This can be

	f_1	f_2	f_3	f_4
Metafor	9.54	40.35	50.22	97.67
Yates [242]	9.60	38.10	50.70	98.50
Goura [243]	9.67	36.87	50.26	90.00
Beaubien *et al.* [244]	9.46	39.44	49.71	94.39
Zhang*et al.* [245]	9.57	38.17	48.35	91.55

Table 6.18: First four natural frequencies of the AGARD 445.6 wing from the present calculation and the literature (all numerical models). Frequencies are in Hz with f_1 and f_3 corresponding to the first and second bending modes, and f_2 and f_4 to the first and second torsion modes, respectively.

	Interpolation method	d_z LE [m]	d_z TE [m]	T [s]
Goura [243]	CVT	0.0112	0.0127	/
Melville *et al.* [246]	/	0.0112	0.0122	/
Present	TPS	0.01166	0.01324	10.7
	CPC2 $r = 0.1$	0.01164	0.01324	3.8
	CPC2 $r = 0.2$	0.01165	0.01323	6.0
	CPC2 $r = 0.4$	0.01166	0.01324	8.0
	CPC2 $r = 0.7$	0.01166	0.01324	9.8

Table 6.19: Vertical displacement of the wing tip at the leading and trailing edges (respectively denoted LE and TE). T is the average time spent by the interpolation at each FSI iteration.

explained by the use of different structural models (plate elements in the references vs volume elements in the present study) and the use of different mesh interpolation methods (such as the CVT in the work of Goura [243]). Tab. 6.19 also shows the average time, denoted by T, spent by the interpolation process at each FSI iteration. Note that this includes both fluid-to-solid and solid-to-fluid data interpolation at each iteration. As expected, this time increases with the CPC2 radius as the number of non-zero entries in the sparse interpolation matrix increases as well. This time is therefore maximum when using the TPS interpolation which provides a full dense interpolation matrix. Considering the low CPU time overhead of the interpolation, which is for the TPS 3.2% of the total CPU time of the simulation in this case study, the TPS basis function is kept for the flutter analysis to guarantee the best accuracy.

Flutter analysis of the AGARD wing

The fully coupled aeroelastic response of the AGARD wing is illustrated in Fig. 6.32, which shows the vertical displacement of the leading edge at the wing tip for three different speed indices in the vicinity of the flutter boundary at M = 0.96. As the flight condition is transonic, the aerodynamic loads are again nonlinear. Consequently, the post-critical responses are expected to converge towards limit cycles if the simulations are long enough. As the simulations are very computationally expensive, their duration is less than 20 cycles and therefore the full nature of the aeroelastic behaviour of the system is not captured. Within the simulated time window, three typical behaviors can be observed: a damped response for $V^* < V_\mathrm{f}^*$, a constant amplitude response for $V^* = V_\mathrm{f}^* = 0.281$ and an amplified response for $V^* > V_\mathrm{f}^*$, where V_f^* is the flutter speed index. As previously mentioned, the flutter condition is determined by successive evaluations of the damping coefficient ζ from the aeroelastic response as a function of the speed index. Fig. 6.33 shows the damping coefficient as a function of the speed index at $M = 0.96$. It first increases starting from low values at low speed indices, then reaches a maximum and finally

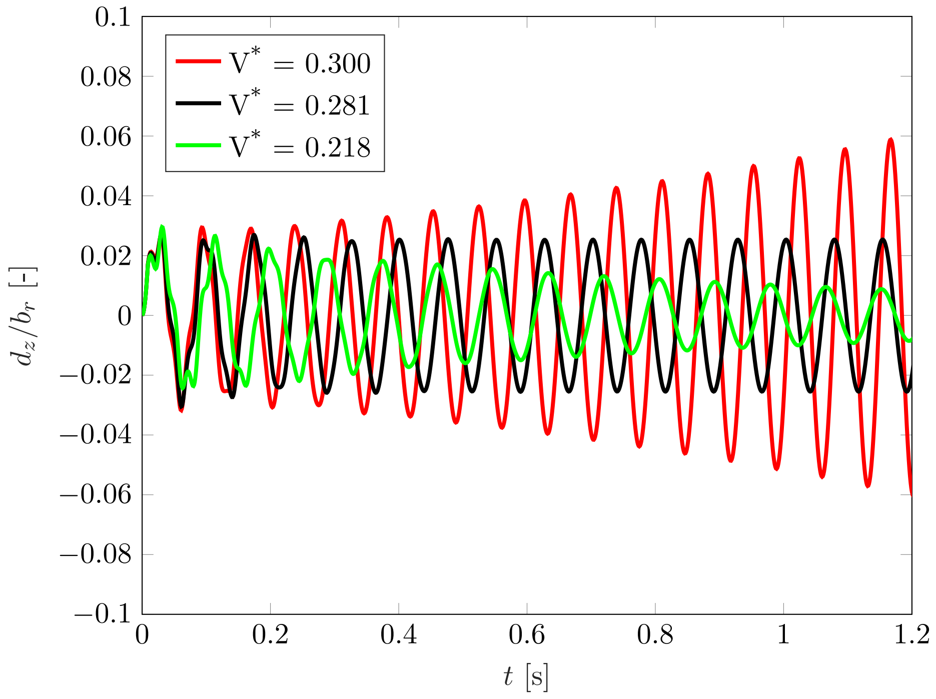


Figure 6.32: Aeroelastic response of the AGARD 445.6 wing for three speed indices V^* with M $= 0.96$ and $\rho_\mathrm{f} = 0.0634$ kg/m^3: vertical displacement d_z of the leading edge at the wing tip normalized by $b_r = c_r/2$. The flutter speed index is $V^*_\mathrm{f} = 0.281$

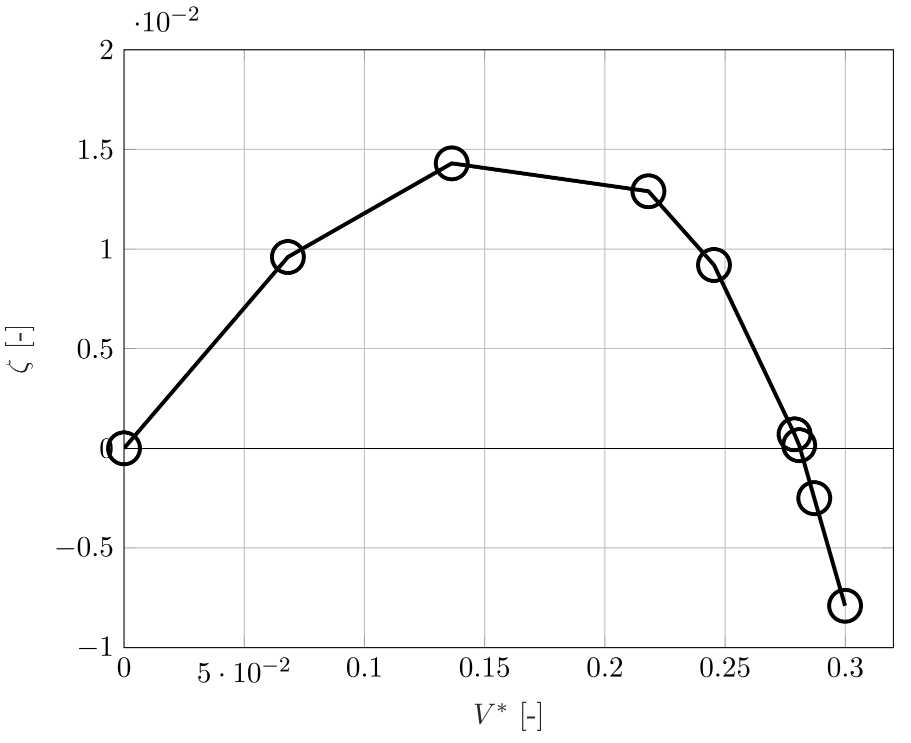


Figure 6.33: Computed damping coefficient of the AGARD 445.6 wing aeroelastic response as a function of the speed index for M $= 0.96$ and $\rho_\mathrm{f} = 0.0634$ kg/m^3.

drops until it crosses the $\zeta = 0$ axis which corresponds to the flutter point. Fig. 6.34 shows the pressure contours on the wing at $M = 0.96$ and $V^* = 0.3$ for three time instants in a cycle. Contours of the Mach number in the supersonic region are superposed at three wing sections in order to show its motion within one cycle and thus highlight the typical nonlinearities of the transonic regime.

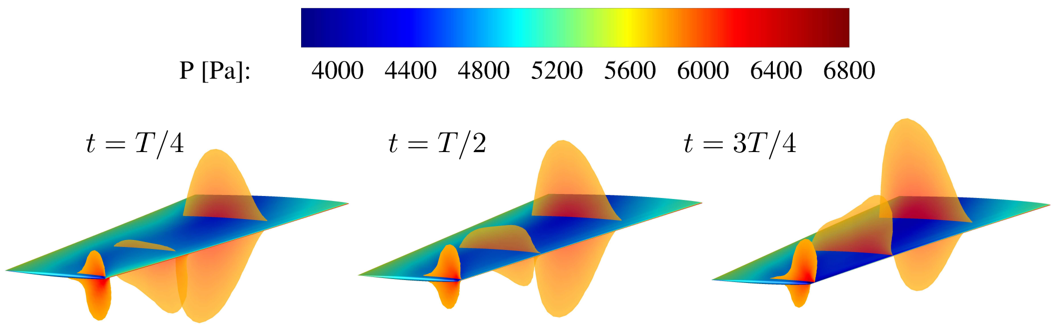


Figure 6.34: Surface pressure and Mach number in supersonic region for the AGARD 445.6 wing at three different time instants of a period with M= 0.96, $\rho_\mathrm{f} = 0.0634$ kg/m^3 and $V^* = 0.3$. The flow is from right to left.

Fig. 6.35 shows the computed flutter boundary that is compared to the experimental results [242] and to computational results obtained solving the Euler [226,244,247,248] or RANS [248, 249] equations. For all computational reference results, the structural part is modeled by a modal decomposition approach which differs from the fully time-integrated approach used in this thesis. It can be seen that the results obtained by coupling SU2 and Metafor with CUPyDO are in good agreement with experiments and with the other computational results found in the literature in the subsonic and transonic regimes. It is also important to note that the transonic dip is well-captured. However, larger discrepancies with the experimental data are observed for supersonic Mach numbers, especially at $M = 1.141$. Similarly to other computations, the flutter boundary is over-estimated at these Mach numbers. Although the origin of this discrepancy remains unclear, several explanations [226, 248] have been proposed, such as the impact of viscous effects (not accounted for in the present Euler simulations) and the complex nonlinear shock-boundary layer interaction. Even for RANS simulations, uncertainties remain concerning the turbulence model and the impact of transition. Additionally, the wing tip geometry, i.e. cut-off or rounded (rounded here), and the spatial discretization scheme (centered or upwind) [249] could also play a role in the supersonic regime. Finally, the effect of structural damping on the flutter boundary should be investigated, since no indication of its experimental value is given [247]. All these details are relevant for accurately capturing the flutter boundary at high Mach numbers but are in majority related to the fluid model. A detailed investigation of the flutter solution at supersonic speeds then falls beyond the scope of the present verification of the coupling environment. Note however that the performance in terms of coupling convergence is found to be highly satisfactory for all Mach numbers (3 iterations to reach the prescribed tolerance).

The performance of the coupled simulations in terms of CPU time was assessed at one of the flutter points: M= 0.96 and $V^* = 0.281$. The results are illustrated by the pie diagram in Fig. 6.36 which shows the relative contribution of the different tasks to the total CPU time of the simulation. Not surprisingly the computation of the fluid solution by SU2 is clearly the more time consuming task and represents about two thirds of the total time. It is followed by the ALE fluid mesh deformation (based here on a pseudo-elastic solid equilibrium approach) that contributes to almost one third of the total time. On the other hand, the contribution of the solid computation represents barely more than 2% of the total time and is thus insignificant. Finally, the FSI tasks,

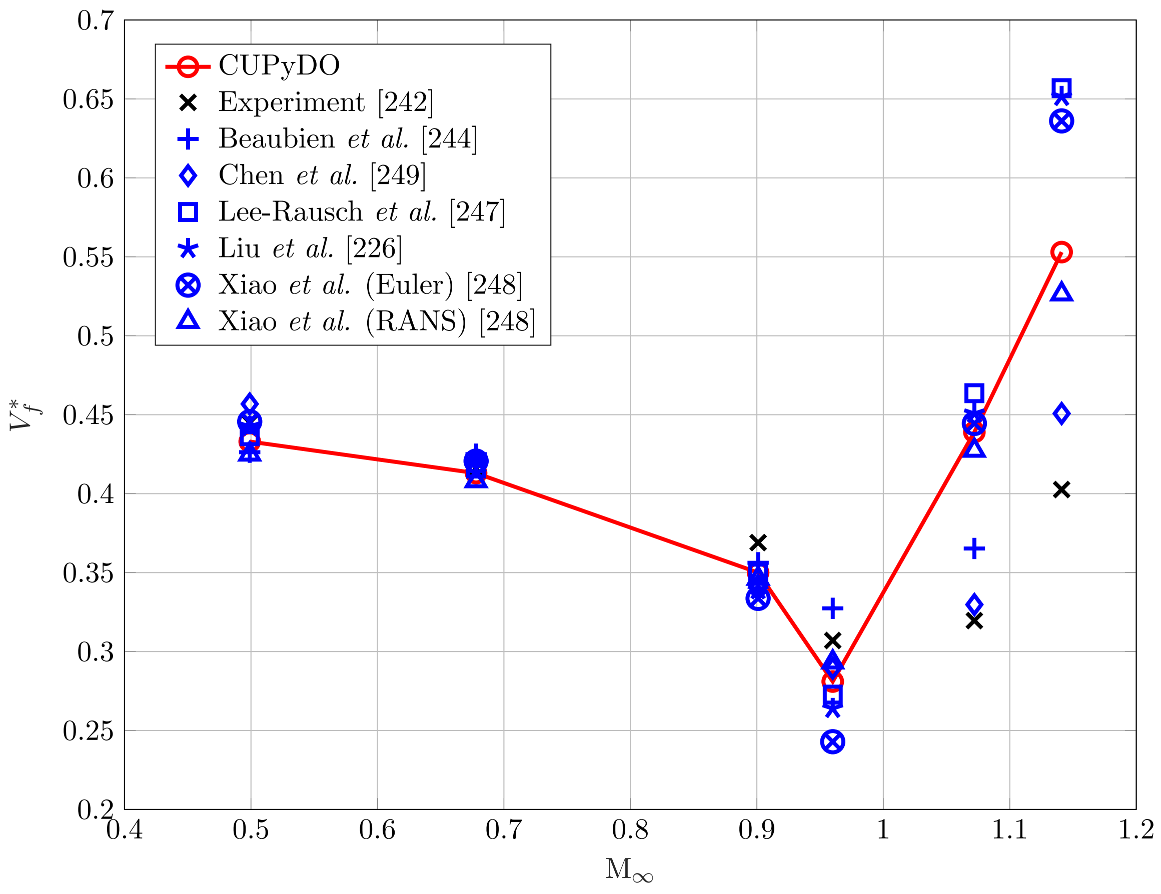


Figure 6.35: Flutter boundary of the AGARD 445.6 wing in the transonic regime.

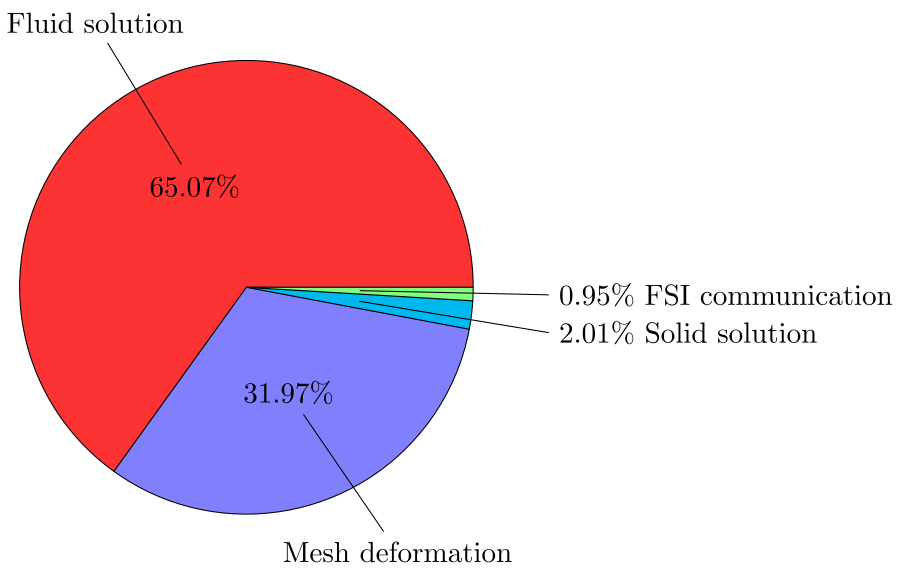


Figure 6.36: Typical distribution of the CPU time, with respect to the total CPU time, for each sub-system of a coupled simulation on the AGARD 445.6 flutter case.

which typically include data exchange, interface interpolation and coupling residual computation only represents a marginal cost (0.95% of the total time). This demonstrates that the overhead brought by the coupling environment *per se* is negligible. Note that the distribution of CPU time presented in Fig. 6.36 relates to the performance obtained for this specific test case. Therefore, it only gives general indications on the performance one could expect for other 3D cases. The CPU time is still highly dependent on other factors such as, for instance, the number of threads used by the computation, the mesh size, the numerical settings and models used in each solver, or their intrinsic algorithmic efficiency.

6.4.3 Test case summary

The AGARD 445.6 wing test case has been used to validate the coupling tool for flutter analysis on a general three-dimensional configuration with non-matching interface meshes. From a coupling capability point of view, the level of complexity encountered in this case is comparable to what must be achieved in industrial design. The results obtained in this study have been compared with a collection of references from the literature. The solution obtained with CUPyDO has been found to be in good agreement with these references in the subsonic and transonic flow regimes. In the supersonic regime, large discrepancies have been observed between all the results and can be in majority related to the fluid solution (typically the flow model and the numerical schemes) rather than the coupling environment itself. In the present case, the coupling process has demonstrated very good performances over the whole range of Mach numbers considered.

The relative contribution to CPU time of the different components of a typical 3D flutter calculation has then been analyzed. It has been found that the contributions of the computation of the fluid solution (65%) and the ALE mesh deformation (32%), dominate the total CPU cost while the computation of the solid solution represents only a very small percentage of the total cost (2%). Finally, the overhead brought by the coupling tasks is found to have the smallest contribution (1%).

6.5 Conjugate heat transfer with a circular cylinder immersed in cross-flow

A last verification case is now presented to assess the thermal coupling capabilities of CUPyDO. The test case is inspired by a numerical study performed by Nettis [78] which involves a conjugate heat transfer with a circular cylinder immersed in a laminar cross-flow.

The objective of this test case is to verify the thermal coupling schemes implemented in CUPyDO with a focus on their stability at different Biot numbers. Furthermore, this test introduces the coupling with GetDP as another compatible FEM structural solver to further demonstrate the versatility of CUPyDO.

6.5.1 Case description and simulation parameters

We consider a perfectly rigid hollow cylinder immersed in a laminar and uniform flow, as illustrated in Fig. 6.37. The ratio between the inner and outer diameter of the cylinder is $D_\mathrm{i}/D_\mathrm{o} = 0.5$. The temperature $T_\mathrm{i} = 350$ K is imposed on the inner boundary of the cylinder. The fluid domain is circular with a farfield diameter of $25\,D_\mathrm{o}$. The farfield fluid temperature is imposed at $T = 288.15$ K, so that the thermal exchange at the outer boundary of the cylinder defines a convective CHT problem. The fluid flow is characterized by a Reynolds number $\mathrm{Re} = 40$ (based on D_o), a Mach number $\mathrm{M} = 0.38$, and a Prandtl number $\mathrm{Pr} = 0.72$. Two distinct values of the ratio between fluid and solid thermal conductivities are considered in this study: $\lambda_\mathrm{s}/\lambda_\mathrm{f} = 4$, referred to as case A and reproducing the study of Nettis [78], and $\lambda_\mathrm{s}/\lambda_\mathrm{f} = 1/4$, referred to as case B. The purpose of evaluating two thermal conductivity ratios is that cases A

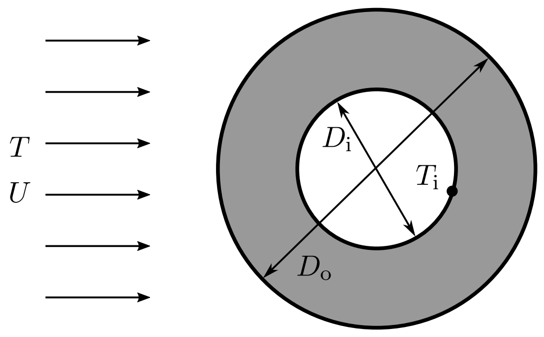


Figure 6.37: Heated hollow cylinder in a cross flow, geometry and boundary conditions.

and B provide a Biot number which is lower and higher than one, respectively. The performance of the thermal coupling schemes, as detailed in Section 3.3, is then evaluated.

Steady coupled simulations are performed because the Reynolds number is low enough to guarantee stationary flow in the wake of the cylinder. The SU2 solver is used to solve the compressible laminar Navier-Stokes equations in the fluid domain, which is discretized using a structured O-mesh with 6120 grid points. Convective fluxes are evaluated using the second-order centered JST scheme and time integration (time-marching for steady solution) is performed with the second-order Euler implicit scheme. The GetDP solver is used to solve the steady heat equation in the cylinder. The solid is discretized using quad elements with 21 X 121 points in the radial and circumferential directions, respectively. The maximum number of coupling iterations is intentionally set to a high value of 1000 so as not to bias the evaluation of coupling procedure for cases with a very low convergence rate. The coupling tolerance is set to 0.1 K for the temperature-back schemes (FFTB and hFTB) whereas it is set to 1 W/m^2 for the flux-back schemes (TFFB and hFFB), i.e. as low as 0.1% of the expected temperatures and fluxes along the interface.

In a first step, simulations of case A are performed using the four thermal coupling schemes implemented in CUPyDO and a variable numerical heat transfer coefficient, when applicable. The performance of the coupling procedure is measured by the number of coupling iterations that are required to reach the prescribed tolerance. The fluid and solid domains are also meshed so as to have matching discretizations at the interface to exclude any interpolation error. The solutions obtained for case A are compared to the reference solution provided by Nettis [78]. In a second step, the same study is performed for case B and the efficiency of the thermal coupling schemes is compared to case A. Finally, the mesh discretization of the solid interface is modified to consider non-matching meshes at the interface. The consistent interface mesh interpolation implemented in CUPyDO is thus used and tested on case A only, and the solution is compared to the matching interface case.

Preliminary fluid and solid simulations are also performed in order to verify the settings of the solvers. In the fluid, the steady flow around the cylinder is computed and the drag coefficient is then compared with other references. Both adiabatic and isothermal boundary conditions applied at the cylinder outer boundary are tested. In the solid, the temperature field is computed for different types of axisymmetric boundary conditions (heat flux imposed or temperature imposed) applied at the outer boundary. The solution is then compared to analytical expressions.

6.5.2 Results

The results obtained in the preliminary simulations are presented here, followed by the results obtained for the fully coupled system.

Steady laminar flow around the cylinder

The steady laminar fluid-only simulations are performed under the same flow conditions as those introduced for the coupled problem. One simulation is performed with adiabatic boundary conditions applied at the wetted surface of the cylinder. The other simulation is performed by applying a constant temperature $T_\mathrm{o} = T_\mathrm{i} = 350$ K instead. The computed drag coefficients are $c_d = 1.68$ and $c_d = 1.64$, respectively, which are in good agreement with a large amount of available experimental results in the literature, e.g. Finn [250]($c_d = 1.60$), Hoerner [251]($c_d = 1.65$) or Panton [252](collection of results in the range $c_d = 1.50 - 1.68$). The steady flow solution is represented in Fig. 6.38. For such a low Reynolds number, the flow is stable and features two symmetric contra-rotating vortices that develop in the near wake. Note that an

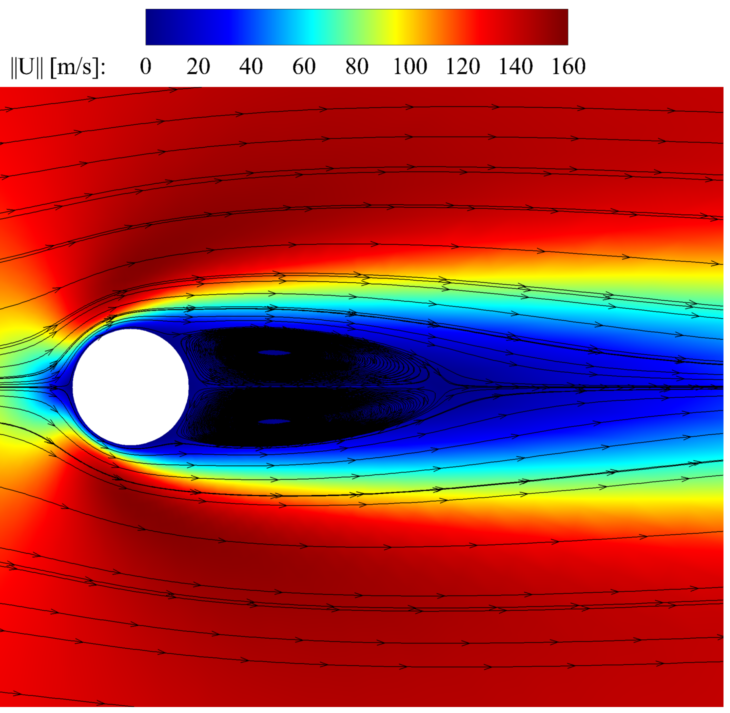


Figure 6.38: Steady laminar flow around the circular cylinder at Re = 40. Velocity magnitude contour with streamlines.

unsteady computation would lead to the exact same steady solution.

Thermal analysis of the hollow cylinder under prescribed thermal loads

The GetDP solver is then used to compute the temperature field within the cylinder submitted to axisymmetric thermal loads. At the outer boundary, either an imposed heat flux $q_\mathrm{o} = (\boldsymbol{q} \cdot \boldsymbol{n})_\mathrm{o}$, $\boldsymbol{n}$ pointing outwards the solid domain or an imposed temperature T_o are considered. At the inner boundary, a Dirichlet boundary condition is used: $T_\mathrm{i} = 350$ K. For this simple problem, an axisymmetric analytical solution for the radial temperature field exists (see Appendix G for details),

$$T(r) = A \ln r + B \,, \tag{6.27}$$

where the values of the constants A and B are given in Tab. 6.20 for the two different boundary conditions at $r = R_\mathrm{o} = D_\mathrm{o}/2$.

Fig. 6.39 illustrates the results obtained when setting $q_\mathrm{o} = -2000$ W/m^2, $T_\mathrm{o} = 288$ K and $\lambda_\mathrm{s} = 46$ W/mK (all arbitrary values). The results show perfect matching between the computed

	Heat flux imposed q_o	Temperature imposed T_o
A	$\frac{q_\mathrm{o} R_\mathrm{o}}{\lambda_\mathrm{s}}$	$\frac{T_\mathrm{i} - T_\mathrm{o}}{\ln(D_\mathrm{i}/D_\mathrm{o})}$
B	$T_\mathrm{i} - A\ln(D_\mathrm{i}/2)$	

Table 6.20: Constants A and B for the analytical solution, Eq. (6.27), of the uncoupled thermal problem in the hollow cylinder.

and analytical solutions, providing support to the choice of numerical settings for the subsequent fully coupled simulations.

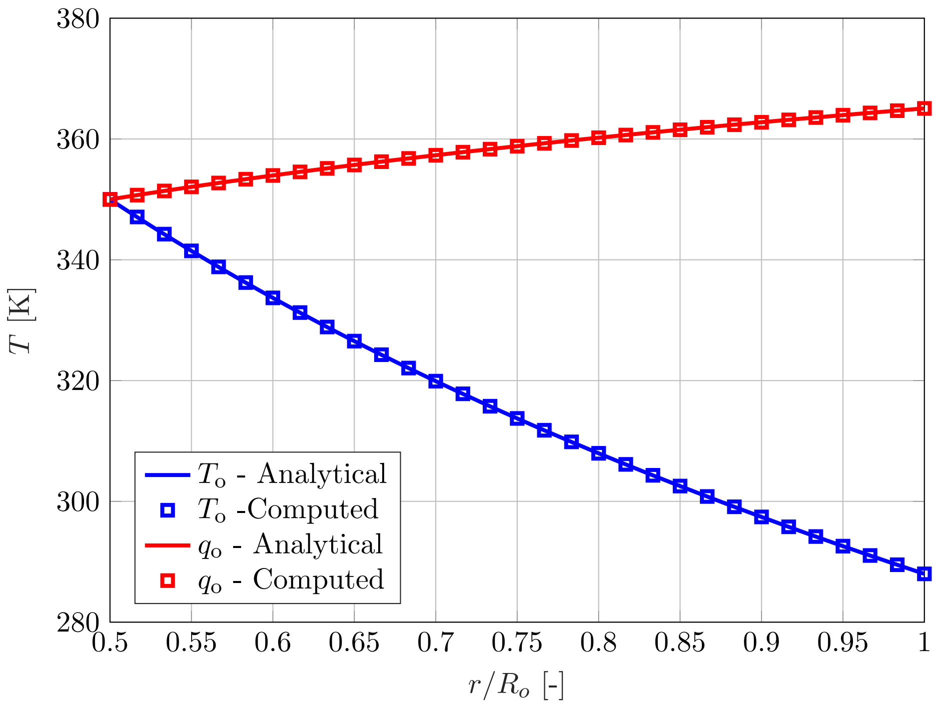


Figure 6.39: Radial distribution of the temperature in the hollow cylinder for two different types of boundary conditions applied at the outer boundary.

Coupled CHT results

The four thermal coupling schemes presented in Section 3.3 are tested on the coupled case A. Tab. 6.21 summarizes the number of coupling iterations required to reach the prescribed coupling tolerance. This case is chosen such that the local Biot number is below one everywhere on the surface of the cylinder (averaged computed value $\overline{Bi} = 0.18$). The results obtained for the FFTB and TFFB schemes confirm that only the FFTB sheme is stable in this situation. In order to stabilize the flux-back scheme, the numerical heat transfer coefficient $\tilde{h}$ can be used (hFFB scheme) but its optimal value is not known a priori and the coupling convergence rate remains dominated by the worst local condition, i.e. the points on the interface that are the farthest from the optimal condition[6]. In this study, the optimal global value is found to be $\tilde{h} \approx 30$. The number of iterations rapidly grows for smaller values of $\tilde{h}$ and the coupling process

[6]In Section 3.3, optimal condition corresponded to $\tilde{h} = h$ where h is the physical heat transfer coefficient of the thermal exchange.

	Value of $\tilde{h}$ (W/m^{2}K)	N_{FSI}
hFTB	0.5	8
	1	8
	2	9
	5	10
	10	10
	15	16
	30	14
	66	12
	80	11
	≥ 90	X
FFTB	-	8
hFFB	5	79
	10	47
	20	38
	30	33
	≥ 40	X
TFFB	-	X

Table 6.21: Comparison of the performance of different CHT coupling schemes - case A.

becomes unstable for any $\tilde{h} > 40$. However, the hFFB scheme, although converging at optimal conditions, still performs poorly when compared to the temperature-back schemes.

The TFFB scheme can also be stabilized through the same coupling under-relaxation as for mechanical coupling. Static relaxation is here applied and the optimal value of the relaxation parameter is found to be around 0.2. The corresponding number of coupling iterations is 37, which does not bring any improvement compared to the hFFB case with an optimal $\tilde{h}$.

The FFTB and hFTB schemes perform significantly better than their flux-back counterparts, with four times less coupling iterations at optimal conditions. For the hFTB scheme, the number of coupling iterations for $\tilde{h} \to 0$ reaches the same level as for the FFTB scheme. This behaviour is in agreement with the properties of the hFTB scheme as stated in Section 3.3. However, it was not expected that the optimal condition is also found for $\tilde{h} \to 0$, showing that the use of hFTB over TFFB does not bring any improvements in this particular case.

Fig. 6.40 shows the temperature distribution at the fluid-structure interface T_w obtained with the different coupling schemes. Good agreement is obtained between the present calculations and the results of Nettis [78] for which the hFTB scheme was used with $\tilde{h} = 10$. The temperature field inside the solid domain is illustrated in Figure 6.41 showing the typical distribution resulting from the convective heat transfer induced by the surrounding flow.

The study of case A is now extended to consider non-matching meshes at the fluid-solid interface. Non-matching discretization is obtained by coarsening the solid mesh in the circumferential direction from 31 nodes initially to 21 and 11 nodes. The coupled simulations are then performed under the same conditions as before, where the consistent RBF interpolation of CUPyDO is used to transfer the data between the two meshes. The CPC2 basis function is used with a radius of 0.3 m. The results are summarized in Fig. 6.42 showing the temperature distribution over the wetted surface of the cylinder for the FFTB coupling scheme. It is found that the impact of the circumferential solid discretization on the solution is only marginal, which confirms the efficiency of the consistent interpolation approach for thermal data transfer.

A new analysis is performed now on case B, which is chosen such that the local Biot number is higher than one everywhere on the surface of the cylinder (averaged computed value $\overline{\text{Bi}} = 2.04$). Tab. 6.22 summarizes the number of coupling iterations required to reach the prescribed tolerance

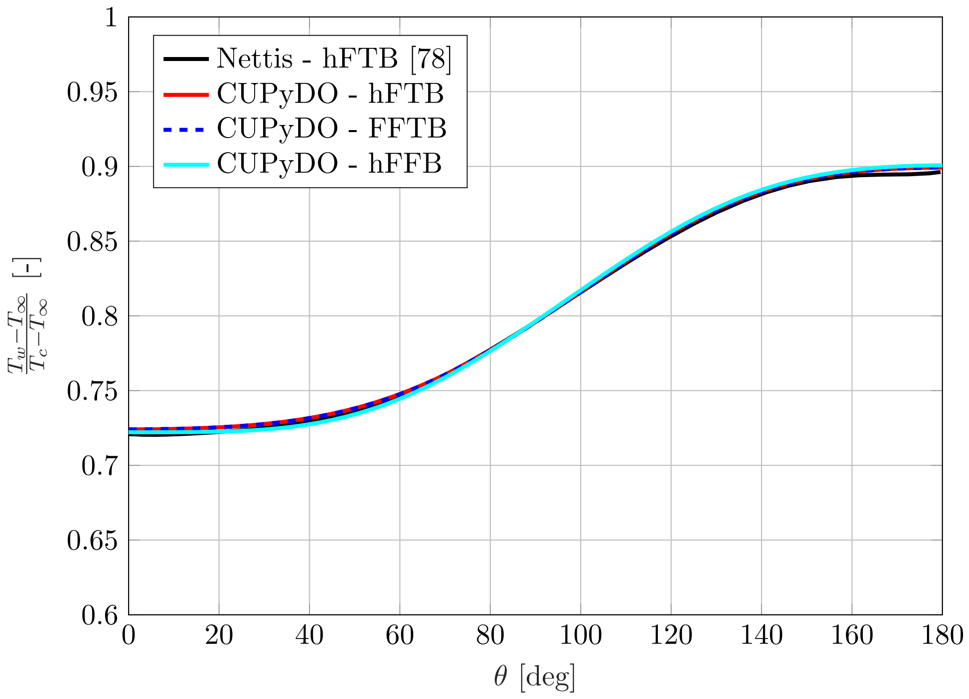


Figure 6.40: Case A - Temperature distribution over the wetted surface of the cylinder. The upstream stagnation point corresponds to $\theta = 0°$.

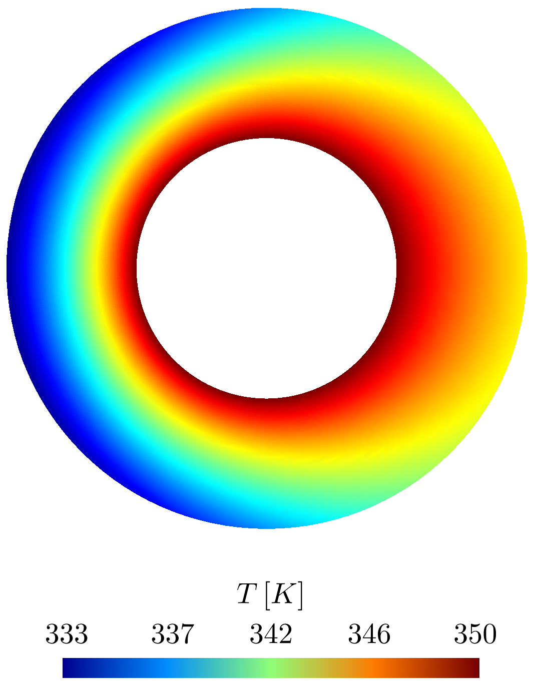


Figure 6.41: Temperature distribution inside the hollow cylinder.

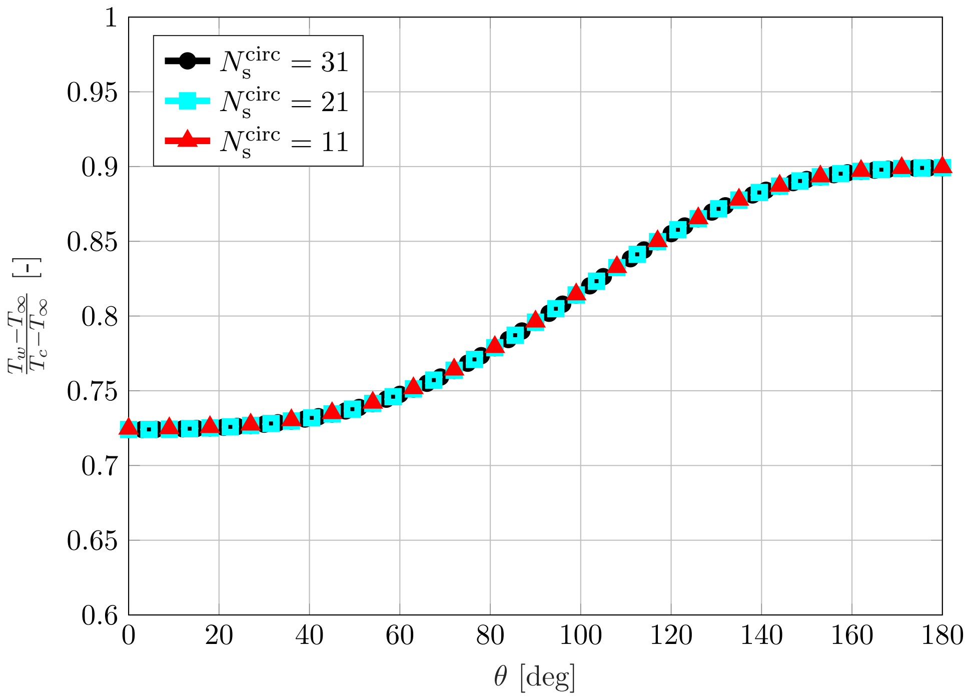


Figure 6.42: Case A - Temperature distribution over the wetted surface of the cylinder for three different circumferential discretizations of the solid surface interface mesh. FFTB scheme. The upstream stagnation point corresponds to $\theta = 0°$.

for each scheme. As expected, the relative efficiency of the temperature-back and flux-back schemes is reversed. For Bi > 1, the FFTB scheme is found to be unstable while the TFFB

	Value of $\tilde{h}$ (W/m^{2}K)	N_{FSI}
hFTB	0.05 to 1000	X
FFTB	-	X
hFFB	1	90
	2	44
	5	17
	7	13
	8	11
	9	11
	10	17
	11	32
	12	102
	$\geq$13	X
TFFB	-	8

Table 6.22: Comparison of the performance of different CHT coupling schemes - case B ($\overline{\text{Bi}} = 2.04$).

schemes provides a fast coupling convergence. In this particular case, it is also not possible to find a value of $\tilde{h}$ that leads to a stable coupling of the hFTB scheme. This makes any temperature-back scheme inefficient for this case. Optimal conditions can be found for the hFFB scheme, but still provide lower efficiency than TFFB. This again limits the benefit of thermal schemes using uniform $\tilde{h}$ for practical applications and calls for the development of locally optimized versions that involve local values of $\tilde{h}$. Fig. 6.43 illustrates the temperature distribution over the wetted surface of the cylinder for $\overline{\text{Bi}} = 2.04$.

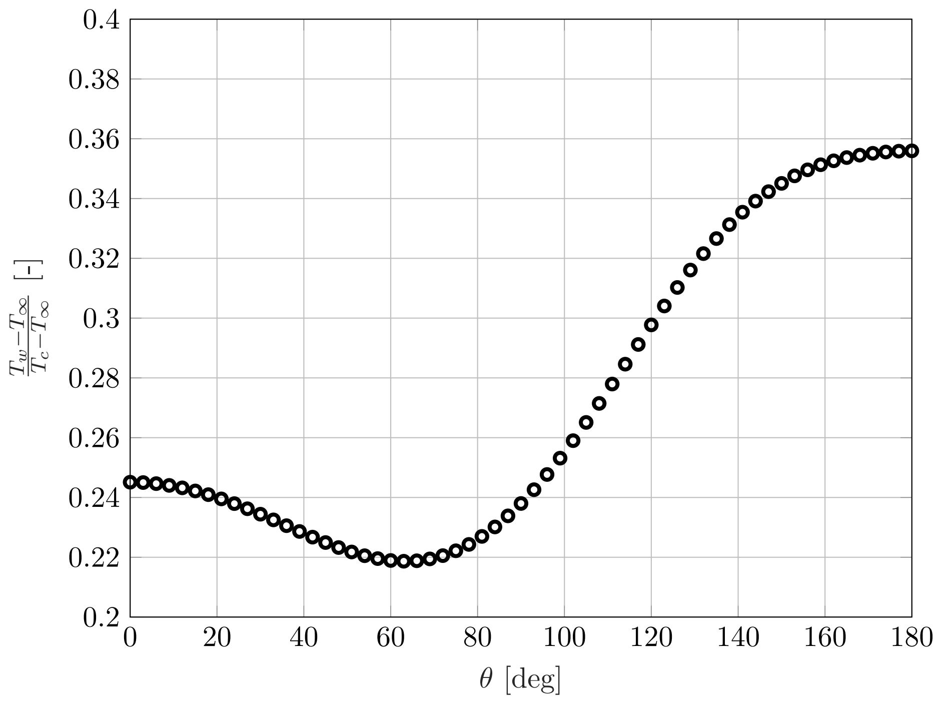


Figure 6.43: Case B - Temperature distribution over the wetted surface of the cylinder. The upstream stagnation point corresponds to $\theta = 0°$.

6.5.3 Test case summary

This verification case has tested the thermal capabilities of CUPyDO for CHT applications. The solid solver GetDP has been used, which further demonstrates the versatility of the coupling environment. Thermal exchanges between the hollow cylinder and the cross-flow have been simulated for two physical Biot numbers using the four coupling schemes implemented in CUPyDO. The performance of the schemes has been evaluated in terms of the number of coupling iterations, and their behavior has been compared to the expected trends that were established in Section 3.3 on an analytical 1D model. Results have confirmed that temperature-back schemes are efficient for $\text{Bi} < 1$ while, conversely, flux-back schemes become efficient for $\text{Bi} > 1$. It has also been shown that the use of a numerical heat transfer coefficient $\tilde{h}$ can stabilize the coupling procedure. However, it has been found that the use of a constant $\tilde{h}$ along the interface limits the benefit of the method, as it is related to the worst local condition along the interface. The same under-relaxation procedure as used for mechanical FSI can also be considered to stabilize the coupling, but it has not been found to perform significantly better than $\tilde{h}$-based schemes. Finally, consistent interpolation is successfully applied for the transfer of thermal data in the case of a non-matching interface discretization.

Summary of chapter 6

CUPyDO has been validated on both mechanical and thermal applications of various complexity. For the mechanical coupling, test cases focused mainly on vortex-induced vibrations and aeroelastic flutter. The results obtained in these studies have been compared to other works from the literature with good agreement. To compute the fluid part of the problem, the Euler or Navier-Stokes equations are solved in SU2 with the finite

volume method in a Arbitrary Lagrangian-Eulerian formulation. The high modularity of the framework has been demonstrated by using different structural solvers and models. Lagrangian linear/nonlinear finite element solvers such as GetDP or Metafor are used to solve the structural dynamic equilibrium for deformable solids and the heat equation for thermal conduction. A simpler integrator is used to compute constrained rigid body motions as in the Isogai aeroelastic test case for instance.

Through the different verification cases, the main coupling capabilities of CUPyDO, such as strong/weak coupling, Aitken's under-relaxation, thermal coupling schemes and non-matching interface mesh interpolation, have been assessed. The performance and the efficiency of the different features are highly satisfactory and several guidelines (or best practice) have been suggested. For flutter applications, the strong coupling scheme should be preferred over the weak coupling to reach a better accuracy on the flutter speed. For coupled problems with high added-mass effects, the Aitken's under-relaxation with the max criterion should be preferably used. The min criterion should be used as an alternative if a convergent coupling process can still not be obtained. Removing the time-step prediction could also improve stability in the most severe cases. Cases with severe added-mass effects can also be handled by the IQN-ILS coupling procedure developed in CUPyDO by M.L. Cerquaglia [2][a].

Fot CHT problems, the correct thermal scheme should be selected according to the characteristic Biot number (temperature-back for Bi< 1 and flux-back for Bi> 1). Thermal schemes based on a numerical heat transfer coefficient should be used only if stability cannot be obtained with the classical schemes. Similarly to the mechanical coupling, under-relaxation can also be envisaged. Finally, the RBF interpolation can be safely used for non-matching interface discretization with any of the two basis functions available. For mechanical applications, the conservative interpolation is preferably used as it is designed to conserve the mechanical energy and the total fluid load through the interface. For thermal applications consistent interpolation is preferred for exchanging temperature and heat flux.

The performance in terms of CPU time of a typical coupled simulation has been evaluated on a typical three-dimensional aeroelastic case. It has been clearly shown that the computation of the fluid solution (65.07%) and the fluid mesh deformation (31.97%) represent the largest contributions. The computation of the solid solution (2.01%) usually represents a negligible contribution while the overhead brought by the coupling tasks (0.95%) is marginal. Parallel coupled simulations have been successfully performed on 16 cores of a single cluster node. However, tests on a massively distributed architecture and a parallel scalability analysis should be considered in the future[b]. Finally, introducing a local computation of the optimal numerical heat transfer coefficient could improve the efficiency of the thermal coupling schemes as well.

[a]No detail about this development was provided in this thesis as the IQN-based coupling method is part of the scope of the thesis of M.L Cerquaglia.

[b]This is out of the scope of this thesis, but it should be mentioned CUPyDO was successfully run in parallel on up to four cluster nodes (64 cores).

Chapter 7

Aeroelastic study of a thin flat plate wing

This chapter presents a typical application of CUPyDO for the aeroelastic study of a thin flat plate wing. The main purpose is to compute the aeroelastic response of such a flexible system by focusing on the flutter boundary and on nonlinear limit cycle oscillations that low order models cannot efficiently represent. The present study is based on a preliminary experimental study that was conducted on several geometrical configurations of the plate, with an aspect ratio varying between 2 and 4 and a sweep angle between 0 and 45 degrees. Some of these configurations are simulated here under the same flow conditions as in the experiment.

This chapter is organized as follows. First, the context of the study, the methodology and the numerical settings are introduced. The relevant experimental results are also recalled. Then, preliminary computational studies are conducted in order to assess and validate the parametrization of each coupled solver. Finally, the aeroelastic numerical study is conducted with CUPyDO and the results are confronted to the available experimental data.

7.1 Context and case description

The aeroelastic response of a generic cantilever flat plate wing with a cord c, a span s, a thickness t and a sweep angle Λ is studied. The general geometry of the plate is illustrated in Fig. 7.1. The data for this study are directly taken from an experimental study performed by Dimitriadis *et al.* [253] in the low-speed wind tunnel of the University of Sydney. The geometrical parameters of the different plates used in this experimental study are given in Tab. 7.1. The present work focuses only on three of the high aspect ratio (AR) configurations with $\Lambda = 0°$, $20°$ and $45°$ denoted here configuration AR4-S0, AR4-S20 and AR4-S45, respectively. The AR4 will be dropped for conciseness. The flat plate wing is made of aluminum sheet with thickness of 1 mm.

Configuration	s [cm]	c [cm]	t [mm]	Λ [deg]
$\star$AR4-S0 (AR = 4)	80	20	1	0
AR4-S10 (AR = 4)	80	20	1	10
$\star$AR4-S20 (AR = 4)	80	20	1	20
$\star$AR4-S45 (AR = 4)	80	20	1	45
AR3-S0 (AR = 3)	60	20	1	0
AR2-S0 (AR = 2.25)	45	20	1	0

Table 7.1: Geometrical parameters of the flat plate wings investigated experimentally by Dimitriadis *et al.* [253] The configurations considered in the present numerical study are highlighted by the $\star$ symbol.

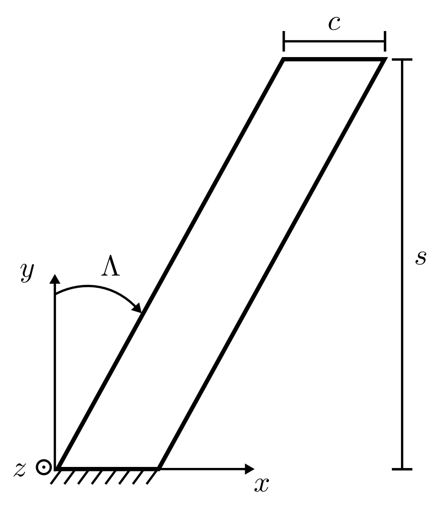


Figure 7.1: Geometry of the cantilever flat plate wing.

Due to the lack of more precise information about the material properties, an isotropic elastic material with standard values for aluminium is considered here, i.e. with density $\rho_{\mathrm{s}} = 2700$ kg/m^3, Young's modulus $E_{\mathrm{s}} = 69 \cdot 10^9$ Pa and Poisson's coefficient $\nu_{\mathrm{s}} = 0.346$. The wind tunnel tests were conducted with air under ambient mean temperature $T = 302.98$ K and pressure $P = 102302.4$ Pa which leads to an air density $\rho = 1.18$ kg/m^3 and dynamic viscosity $\mu = 1.86 \cdot 10^{-5}$ Pa·s (assuming a perfect gas). For the three plate configurations considered here, the maximum airspeed is $U_{\mathrm{max}} = 21$ m/s which gives a maximum chord-based Reynolds number $\mathrm{Re}_c = 266450$. The plates were clamped vertically to the floor of the wind tunnel so that the chord is aligned with the free-stream flow (i.e. zero angle of attack). The motion of the plate was captured by a single high-speed camera (250 fps) which tracked targets that were drawn on the wing. The position of the targets on each frame were extracted by a dedicated software. As explained by Dimitriadis *et al.* [253], using only one camera results in a 2D projection of the 3D wing motion. Nevertheless, these measurements were sufficiently accurate to determine the main quantities of interest, i.e., the flutter speed and flutter frequency.

The experimental study aimed to validate a flutter prediction method based on low-fidelity[1] Vortex Latice Method (VLM) and Doublet Lattice Method (DLM) approaches [253]. The flutter predictions obtained from these two techniques were compared to the experimental data and showed that both the VLM and DLM tend to be moderately conservative in the prediction of the flutter speed. At the critical flutter speed the plates underwent a Hopf bifurcation leading to limit cycle oscillations. General theory about bifurcations of aeroelastic systems can be found in the book of Dimitriadis [254]. As mentioned in the previous chapter, a Hopf bifurcation occurs when the fixed point of a system becomes unstable and limit cycle with infinitesimal amplitude appears around it. The supercritical Hopf bifurcation is characterized by an increase of the LCO amplitude and frequency with the free-stream airspeed at post-critical conditions. A supercritical aeroelastic bifurcation can be schematically illustrated by the black curve in Fig. 7.2 which shows the LCO amplitude of the response as a function of the free-stream airspeed. The critical airspeed is also identified as the flutter speed of the underlying linear system [254]: $U_{\mathrm{c}} = U_{\mathrm{f}}$. In contrast to the supercritical bifurcation, a subcritical bifurcation of nonlinear aeroelastic systems allows LCO to occur for free-stream airspeeds below the linear flutter threshold (red curve in Fig. 7.2). This is explained by the intrinsic properties of nonlinear systems whose trajectory response can be highly sensitive to the initial conditions. Typically for such systems, a LCO response can be obtained for $U < U_{\mathrm{f}}$ if one increases the initial perturbation beyond a given

[1]In this context, models that are not based on a CFD approach are considered as being low-fidelity models.

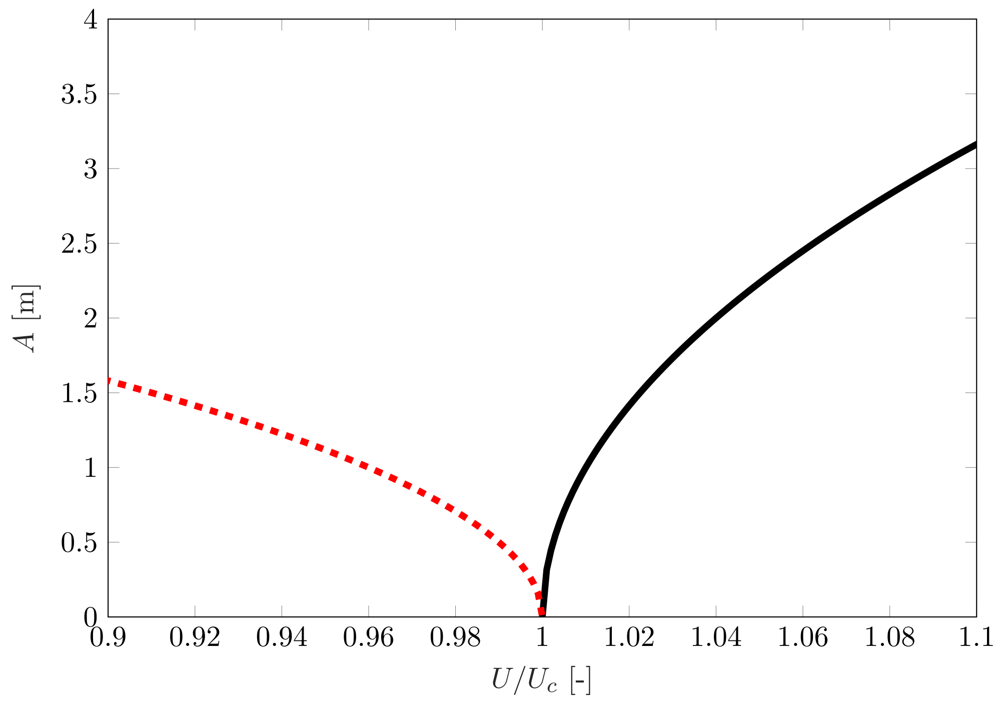


Figure 7.2: Typical behavior of the LCO amplitude as a function of the free-stream airspeed for a supercritical (black) and a subcritical (red) Hopf bifurcation. Values are arbitrary. U_c is the critical airspeed.

threshold. In practice, an aeroelastic bifurcation is caused by either structural or aerodynamic nonlinearities. Lee *et al.* [255] have given some typical examples. Structural nonlinearities are for instance related to the presence of stores, control surface freeplay, friction and geometrical or material hardening. Aerodynamic nonlinearities are more related to viscous effects, such as for example leading-edge recirculation bubbles, dynamic stall and wing tip vortices. For transonic/supersonic flows, shock dynamics and boundary layer-shock interaction are also significant sources of nonlinearities. In the experimental study, the plate wings started undergoing limit cycle oscillations at a specific critical airspeed that depended on the sweep angle and the aspect ratio. The bifurcation was identified as being supercritical, meaning that the amplitude of the LCO grew with airspeed and that the critical airspeed could be considered as the flutter speed of the underlying linear system [253]. The study performed in this thesis focuses on the same nonlinear aspects of the aeroelastic response. Particularly, the post-critical response of the plate is further analyzed with respect to the amplitude of the initial perturbation imposed to the system.

The numerical coupled model uses SU2 as the fluid solver, Metafor as the structural solver and CUPyDO for the coupling environment. The plate is immersed in a hemi-spherical fluid domain where the outer boundary extends to $25c$. Freestream conditions are imposed on the farfield boundary whereas a no-slip condition is imposed at the plate surface. In order to limit the constraints on the grid cells size near the clamping plan, which represents the floor of the wind tunnel, the boundary layer was neglected by imposing a slip condition (i.e. impermeability) on this boundary. The RANS equations with the SST $k-\omega$ turbulence model are solved in SU2. The fluid mesh used for the study is illustrated in Fig. 7.3 for the configuration S0. Note that the edges of the plate are kept sharp. The mesh is a structured O-mesh. The wing surface is discretized with 60, 30 and 10 cells in the spanwise, chordwise and thickness direction, respectively, with a clustering near the leading and trailing edges and near the wing tip. The radial direction (i.e. from the wing surface to the outer domain boundary) is discretized using 40

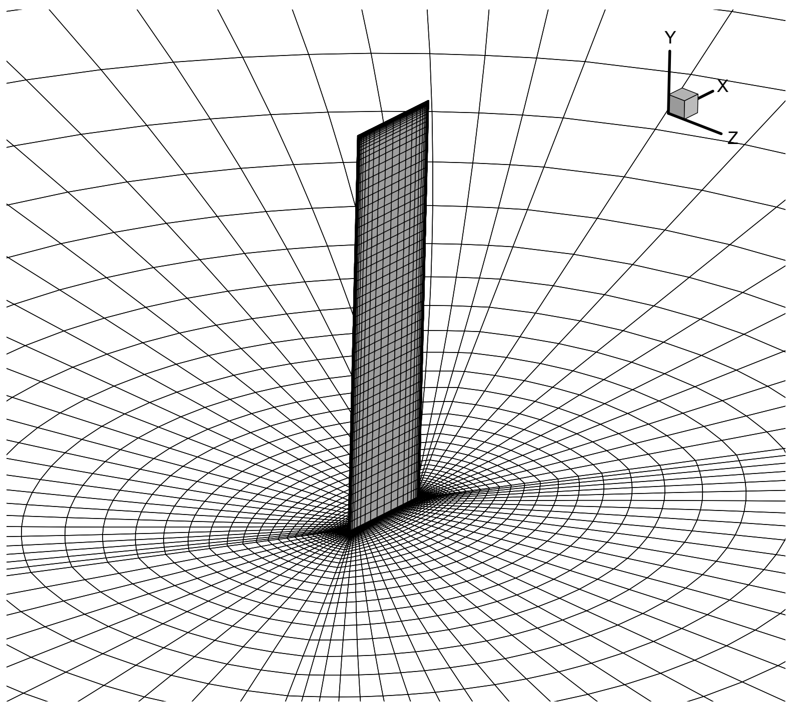


Figure 7.3: Fluid mesh around the wing plate (configuration S0).

cells with a clustering near the wing surface. The total number of grid points is 373981. Note that this mesh results from a mesh convergence study that is presented hereafter. Convective fluxes are evaluated using the second-order centered JST scheme and time integration is performed with the second-order Euler implicit dual-time stepping scheme. The structural part is modeled in Metafor with 8-node continuum EAS elements and discretized with 60, 30 and 2 cells in the spanwise, chordwise and thickness direction, respectively.

Strongly-coupled simulations are used to compute the aeroelastic response of the flat plate wing. The initialization process for the aeroelastic response is the same as the one used for the AGARD 445.6 validation test case. The simulation is started with a uniform flow and no deformation of the plate. During a given initial period t^*, a load is applied on an upstream portion of the plate tip in order to generate a perturbation in the z-direction (out-of-plane direction). Then the loading is released and the plate is let free to vibrate in the flow. The time evolution of the perturbation load $F(t)$ is described by the function

$$F(t) = \begin{cases} F^* \sin(\frac{2\pi}{T}t) & \text{if } t < t^*, \\ 0 & \text{otherwise,} \end{cases} \tag{7.1}$$

with $F^* = 0.25$ N and $T = 4t^*$ (the dead load is maximum at $t = t^*$ just before releasing it). The value of t^* is used as the control parameter for the amplitude of the initial perturbation. Using t^* rather than F^* as the control parameter is preferred because in the latter case an increase of the amplitude would lead to an impulse-like rather than a better-adapted smooth perturbation. Four different values of the perturbation time t^* are considered: 0.01, 0.02, 0.05 and 0.1 s. Note that an initial perturbation through a local dead load was preferred to a global initial displacement of the wing because it was found to be easier and faster to set up. A sine function is used in Eq. (7.1) to ensure a smooth loading profile over the perturbation period t^* and a vanishing time variation of the force at the beginning and at the end of this period.

The fluid forces and solid displacements at the interface are interpolated with the conservative RBF-CPC2 scheme. The control parameters of the simulations are summarized in Tab. 7.2.

Since no numerical instability due to added mass effect is expected (Ma = 2288), BGS without relaxation and a second-order predictor is used as coupling scheme. Simulations are performed in parallel on 16 cores (16 fluid instances, 1 solid) of a computing node with Intel Xeon E5-2650 processor (2 GHz, 16 threads).

Time step [s]	Δt	0.001
Simulated physical time [s]	t_{tot}	up to 10
Maximum number of coupling iterations per time step	n_{FSI}	10
Coupling tolerance [m]	ε	$1 \cdot 10^{-6}$
CPC2 radius [m]	r	0.2

Table 7.2: Control parameters for aeroelastic simulations of the flat plate wing.

7.2 Preliminary study

Several preliminary simulations are first performed to validate the mesh and the numerical settings of the solvers. First, uncoupled unsteady fluid simulations on a perfectly rigid plate are performed with different fluid mesh discretizations in order to determine the grid that offers the best balance between accuracy and computational cost (mainly CPU time). Then, a linear modal analysis of the plate structure is performed with several mesh discretizations.

7.2.1 Fluid simulations and mesh analysis

The preliminary unsteady fluid simulations are performed with the plate configuration AR4-S0 and a free-stream velocity $U = 20$ m/s. Two angles of attack are considered: 5° (low) and 30° (high). Five mesh discretizations are assessed in this study as summarized in Tab. 7.3. The

	Spanwise	Chordwise	Thickness	Radial	Total number
Mesh A	30	15	5	31	70425
Mesh B	40	20	7	31	116100
Mesh C	60	30	10	31	249000
Mesh D	60	30	10	41	364000
Mesh E	80	50	15	41	847200

Table 7.3: Mesh discretizations used for the preliminary fluid simulations. Number of wing surface cells, number of cells in the radial direction between the plate surface and the outer domain boundary, and total number of cells (entire mesh).

same temporal discretization as in the coupled problem (described in the previous section) is also used here.

Tab. 7.4 summarizes the lift and drag coefficients obtained by the different meshes. At the low angle of attack, the unsteady simulation provides a stationary flow solution. This is not the case at the high angle of attack as the flow is, as expected, significantly separated. The time-averaged values of the coefficients are provided in this case. The time spent by the solver for each iteration of the time-marching algorithm (i.e. a steady iteration within the time step)[2] is also indicated in the last column. The table also computes the maximum difference over the five meshes with respect to the value obtained on the finest mesh (E). This difference is barely higher than 3% for the lift coefficient and demonstrates that the meshes considered have a relatively low

[2]The number of cores (16) is kept constant for all computations.

impact on the pressure field. However, and not surprisingly, a significant difference is observed for the drag coefficient at low angle of attack, which reveals the impact of the discretization on the friction component of the drag. This effect is not observed at high angle of attack where the drag is dominated by the pressure component (profile drag) as the flow is separated. Considering

	AoA = 5°		AoA = 30°		t_{iter} [s]
	C_L	C_D	$\overline{C_L}$	$\overline{C_D}$	
Mesh A	0.423	0.0653	0.932	0.545	0.76
Mesh B	0.417	0.0643	0.939	0.543	1.21
Mesh C	0.421	0.0573	0.924	0.534	2.30
Mesh D	0.428	0.0457	0.936	0.541	4.32
Mesh E	0.431	0.0473	0.953	0.554	8.98
Maximum relative difference	3.04%	41.42%	3.02%	3.62%	/

Table 7.4: Stationary (low AoA) and time-averaged (high AoA) aerodynamic coefficients obtained with several mesh discretizations. t_{iter} is the time per steady iteration.

the relatively small variation of the coefficients between mesh D and E (less than 4% for mesh size ratio of 2.32), mesh D is preferred for the computation of unsteady aeroelastic cases as the computational cost of mesh E is expected to be too prohibitive for unsteady simulations with multiple fluid solutions and mesh deformation[3] required at each time step.

7.2.2 Modal analysis of the structure

A structural linear modal analysis is performed in Metafor to evaluate the discretization of the solid model of the plate wing. Three meshes are considered, whose discretization is detailed in Tab. 7.5. In Tab. 7.6, the first five natural frequencies computed for the plate in configuration

	Spanwise	Chordwise	Thickness	Total number
Mesh A'	30	15	2	900
Mesh B'	60	30	2	3600
Mesh C'	120	60	4	28800

Table 7.5: Mesh discretizations used for the preliminary structural modal analysis. Number of wing volume cells.

AR4-S45 are compared to those obtained by a finite element model used by Dimitriadis *et al.* [253]. This model was constructed in MSC Nastran using plate elements, as opposed to volume elements used in this study. Based on the results, the B' mesh is selected for the discretization of the structural model. The gain of using mesh C', i.e. a maximum variation of 1.8% on the fifth frequency, is not considered important enough to justify the cost increase. The accuracy of the structural model is assessed for the other plate configurations by performing the same modal analysis on the selected mesh. Results are reported in Tab. 7.7 and are again compared to the frequencies obtained by the finite elements analysis performed by Dimitriadis *et al.* [253]. A maximum deviation of 3.8% is observed on the fifth frequency of the configuration S45, which is acceptable for the aeroelastic study. Note that the error on the first three frequencies for all configurations is not higher than 2.3%. For $\Lambda \neq 0$ all modes combine both bending and torsion but mode 1 is mostly first bending and mode 3 is mostly first torsion [253].

[3]The robustness of the mesh deformation process can also severely decrease when fine RANS meshes are used in SU2.

	f_1	f_2	f_3	f_4	f_5
Dimitriadis *et al.* [253]	0.68	4.18	10.13	11.63	22.69
Mesh A'	0.73	4.54	10.97	13.10	26.99
Mesh B'	0.69	4.29	10.40	11.98	23.56
Mesh C'	0.69	4.25	10.29	11.86	23.15

Table 7.6: First five natural frequencies of the plate in configuration AR4-S45 computed with Metafor using different mesh discretizations and compared to the results of the finite element model of Dimitriadis *et al.* [253]. Frequencies are in Hz.

	AR4-S0		AR4-S20		AR4-S45	
	Reference [253]	Metafor	Reference [253]	Metafor	Reference [253]	Metafor
f_1	1.28	1.31	1.15	1.17	0.68	0.69
f_2	8.01	8.17	7.09	7.23	4.18	4.29
f_3	10.17	10.34	10.28	10.48	10.13	10.40
f_4	22.48	22.99	19.83	20.30	11.63	11.98
f_5	31.42	32.07	31.55	32.25	22.69	23.56

Table 7.7: First five natural frequencies computed with Metafor for all considered AR4 plate configurations using the selected mesh, i.e. mesh B'. Comparison with the results of the finite element model of Dimitriadis *et al.* [253]. Frequencies are in Hz.

7.2.3 Free wind-off response of the structure

The wind-off[4] response of the plate (configuration S0) to the aforementioned initial perturbation is assessed with respect to different perturbation durations, t^* in Eq. (7.1). The main purpose is to estimate the maximum wind-off amplitude that is reached for each tested value t^*. This is illustrated in Fig. 7.4 that shows the wind-off response of the plate (displacement of the leading edge at the plate tip) computed by Metafor with different perturbation times. In all cases, the response is composed of two main frequencies. Tab. 7.8 summarizes the maximum amplitude and the main frequencies of each response. The same frequency content is measured for all responses.

t^* [s]	$\max d_z/c$ [-]	f_1 [Hz]	f_2 [Hz]
0.01	0.0629	1.4	8.3
0.02	0.1277	1.4	8.3
0.05	0.3037	1.4	8.3
0.1	0.5406	1.4	8.3

Table 7.8: Maximum amplitude and main frequency content of the wind-off response for the configuration AR4-S0.

Note that the perturbation has been defined in such a way that it does not trigger any specific deformation mode, but it can be seen that mode 1 and mode 2 are mostly active in the wind-off response as the measured frequencies are very close to the first two natural frequencies. The maximum wind-off amplitude ranges between 6% and 54% of the chord. Fig. 7.5 shows the maximum wind-off amplitude as a function of t^*. It can be seen that, in the range of perturbation durations considered here, the maximum amplitude evolves almost linearly. Higher values of t^*, hence higher amplitude of the perturbation, have been considered, but no coupled solution has

[4]Response of the plate in a vacuum.

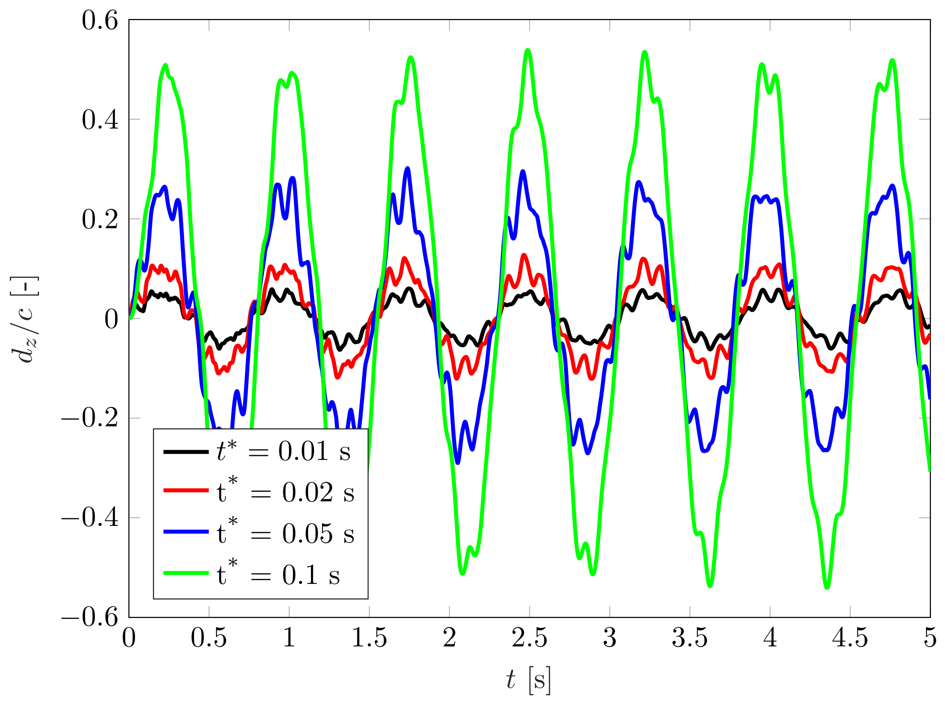


Figure 7.4: Wind-off response of the plate leading edge at the wing tip, configuration S0, with respect to different perturbation durations but with the same perturbation pattern (as described in Section 7.1).

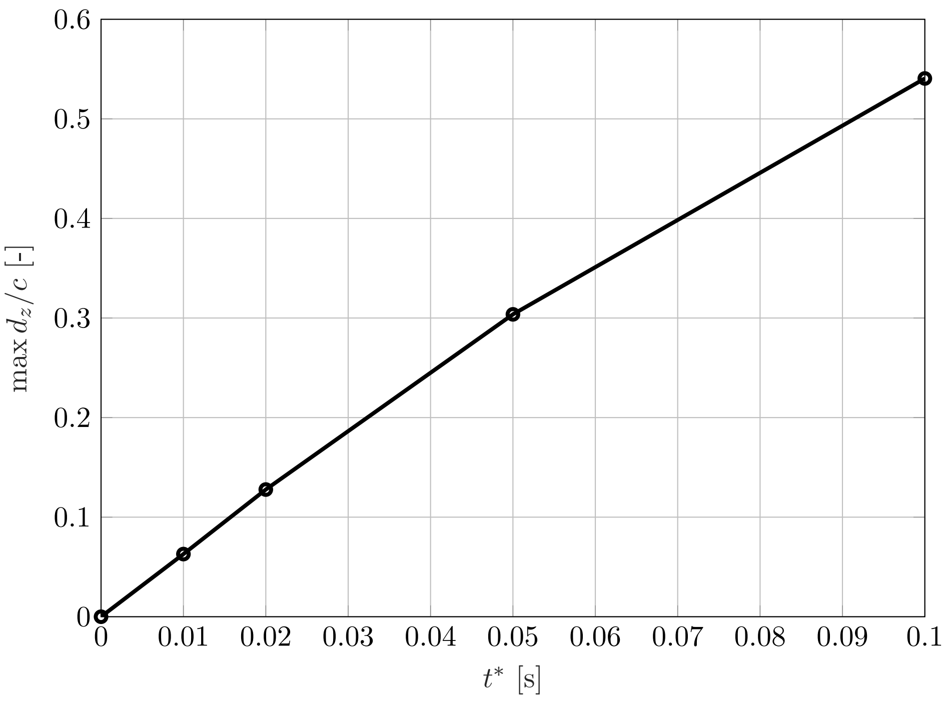


Figure 7.5: Maximum wind-off amplitude of the response as a function of t^* for the S0 plate configuration.

	VLM [253]		Experimental [253]		Computed	
	U_{f}	f_{f}	U_{f}	f_{f}	U_{f}	f_{f}
Plate S0	17.0	5.47	17.1	8.0	16.9	6.1
Plate S20	14.9	5.00	15.8	5.8	15.2	5.4
Plate S45	12.7	3.46	13.4	4.2	13.8	3.6

Table 7.9: Flutter characteristics (velocity U_{f} in m/s and frequency f_{f} in Hz) computed with CUPyDO for the different AR4 plate configurations. Results are compared with the VLM results and experimental measurements reported in Dimitriadis *et al.* [253].

been obtained due to the a lack of robustness in the fluid mesh deformation procedure that led to numerical instabilities and a rapid divergence of the fluid solution (more details are provided further below).

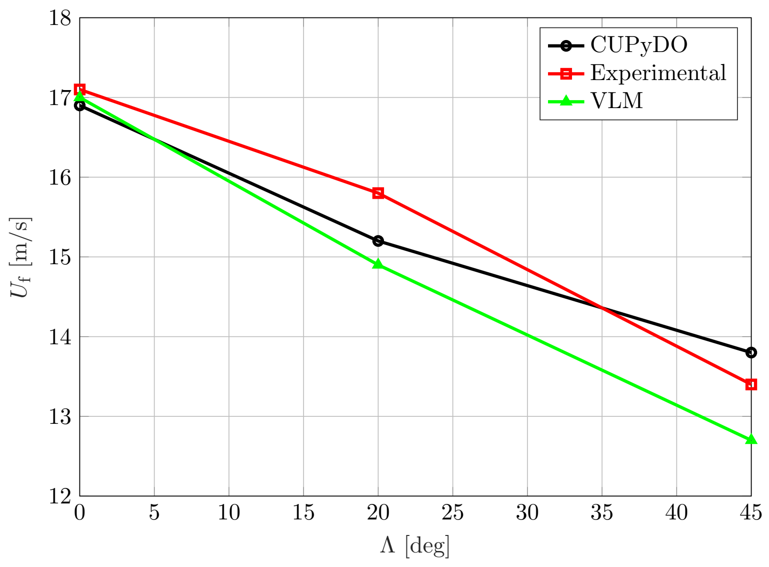
7.3 Unsteady aeroelasticity

The unsteady aeroelastic response of the fully coupled system is now considered. First, the flutter velocity and frequency are sought by computing the aeroelastic response to small perturbations. Then, the amplitude of the perturbation is increased at post-critical conditions and its impact on the limit cycle oscillations is studied.

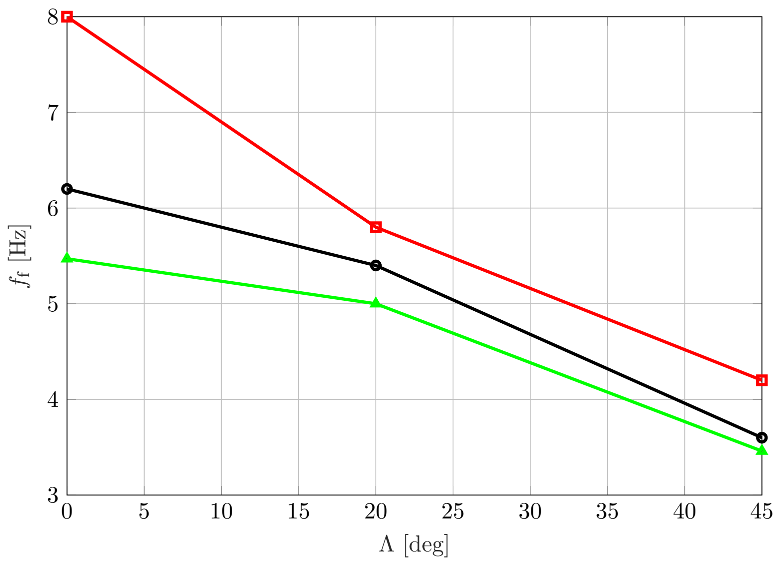
7.3.1 Flutter study

The flutter characteristics of the different plate configurations are now presented. As introduced in Section 7.1, an initial perturbation is given to the system by applying a dead load on the plate tip leading edge during a duration t^*. As only the linear flutter characteristics are sought at this stage, a small perturbation duration ($t^* = 0.01$ s) is chosen in order to provide the smallest amount of perturbation and thus limit the effects of the system nonlinearities (e.g. large structural displacements and flow separation). The damping is extracted from the aeroelastic response and flutter inception is determined using the same iterative process as described in Section 6.2.3. The iterative process is here continued until the flutter speed converges up to a tolerance of 0.1 m/s.

The flutter characteristics obtained for each plate configuration are reported in Tab. 7.9. The computed values are compared to the VLM results and the experimental measurements reported in the work of Dimitriadis *et al.* [253]. Fig. 7.6 illustrates the variation of the flutter speed and flutter frequency with the sweep angle for the three AR4 plates. The results computed with CUPyDO follow the same trend as the references, i.e. a decrease of the flutter velocity and frequency as the sweep angle increases. It is also important to note that the results computed by CUPyDO are consistently closer to the experimental results than the VLM results. The only exception is for the flutter speed at $\Lambda = 0°$ but the discrepancy with experimental results is very small (less than 1.16%) at this point. The errors on the flutter speed for $\Lambda = 20°$ and $\Lambda = 45°$ are also relatively low, i.e., 3.8% and 3.0%, respectively. When comparing the computed (CUPyDO) flutter frequencies to the experimental reference, the maximum error (23.7%) is found for the unswept configuration in contrast with the smallest error found for the flutter speed. The same trend is observed for the VLM results. The errors on the flutter frequency for $\Lambda = 20°$ and $\Lambda = 45°$ are 6.9% and 14.2%, respectively. This flutter study demonstrates the accuracy of the CFD-FEM coupling with CUPyDO for capturing the flutter speed of plate wings. Although the present results are close to the experimental measurements, the VLM approach provides a satisfactory level of accuracy for a computation time that is significantly smaller than a full high-fidelity FSI calculation. Indeed, the computation of the



(a) Flutter speed.



(b) Flutter frequency.

Figure 7.6: Flutter speed (a) and frequency (b) as a function of the sweep angle for the AR4 plates.

flutter speed with VLM requires only a few seconds, while the SU2-Metafor coupling requires roughly 12 hours of computing time to simulate one full cycle of the aeroelastic response on a node with 16 cores and for a mean value of 3 FSI iterations per time step to reach the given coupling tolerance[5]. Thus, other than for verification purposes, the use of a complete CFD-FEM coupling to capture the flutter characteristics of the plate wings seems hard to justify against the use of low-fidelity methods. The good predictions of the VLM can be explained in the present case by the linear mechanisms underlying the flutter inception (e.g., in-plane modes are neglected, small displacements, attached flow). However, a detailed description of the post-flutter aeroelastic response of the plate and flow cannot be accurately predicted by such simplified methods because of the increasing importance of nonlinear mechanisms, as shown in the next section. Under these conditions, high-fidelity (thus nonlinear) coupled models are required.

Fig. 7.7 compares the aeroelastic response of the wing in configuration S0 for different free-stream velocities $U \leq U_\mathrm{f}$ and a small duration of the initial perturbation ($t^* = 0.01$ s). For free-stream velocities below the flutter threshold, $U_\mathrm{f} = 16.9$ m/s, the initial perturbation is damped in time, with a damping rate that does depend on the speed itself. Typically, the initial perturbation is damped faster at $U = 10$ m/s than at $U = 16$ m/s. At the flutter speed, the initial perturbation quickly turns into limit cycle oscillations. The aeroelastic response is composed of a harmonic component which vibrates around a steadily recovering non-oscillatory component, slightly damped in time. This non-oscillatory component of the deformation can be identified as part of the aeroelastic response since it did not occur in the wind-off (i.e. structure only) simulations. The occurrence of a non-oscillatory aeroelastic deflection of the unswept plate was also confirmed by the experiment but the context in which this deflection appears is significantly different. As reported in Dimitriadis *et al.* [253], the straight plate tested in the wind tunnel undergoes significant static (constant in time) deflection for all airspeeds below the flutter threshold. The amplitude of the static deflection increases with the free-stream velocity and becomes as large as the chord for velocities close to the flutter threshold. In contrast, the non-oscillatory deformation computed here is not steady but decays (with varying decay rate though) at all free-stream airspeeds, including at the flutter threshold. Consequently, the amplitude of the computed non-oscillatory deflection is also much smaller (about 100 times smaller) compared to the experiments. Another point is that the static deflection observed experimentally occurs without any significant initial perturbation other than the natural perturbations contained in the free-stream flow of the wind tunnel (typically turbulent buffeting). Conversely, this deflection cannot be obtained in the CFD-FEM coupled model without intentionally perturbing the system, as any other marginal numerical perturbation would immediately be damped. This deflection is associated with a transient component of the response to the initial perturbation. It is therefore challenging to relate the computed non-oscillatory deflection with the one observed experimentally, as no experimental evidence could be found to explain the phenomenon. Dimitriadis *et al.* [253] have nevertheless made the following supposition: any imperfection in the flatness, angle of attack or installation of the flat plate in the wind tunnel will result in a small amount of twist under the action of the drag that acts in front the flexural axis. This initial twist will, in turn, generate a small amount of lift, also acting in front of the flexural axis, that will cause further twist and bending, until the deflection of the plate is stabilised by the internal structural loads. Note that this mechanism cannot be reflected by the VLM-based aeroelastic model, as the drag is neglected. In a CFD-FEM coupled model, such interaction can be represented but, in the present computations, the process appears to be decaying in time, potentially because there is no imperfection either in the plate or in its angle of attack. Since the plate is very flexible, the contribution of gravity to the buckling of the plate could be pointed out as a further destabilizing factor, which cannot be represented by the computational model as gravity is neglected in both the fluid and solid domains. Finally, the experimentally observed

[5]For the S0 plate, this typically represents 3.2 days for 1 s of simulated time.

static deflection of the plate has been pointed out by Dimitriadis *et al.* [253] as being a stiffening mechanism which results in an increase in the experimental flutter frequency compared to the numerical models where this mechanism does not occur.

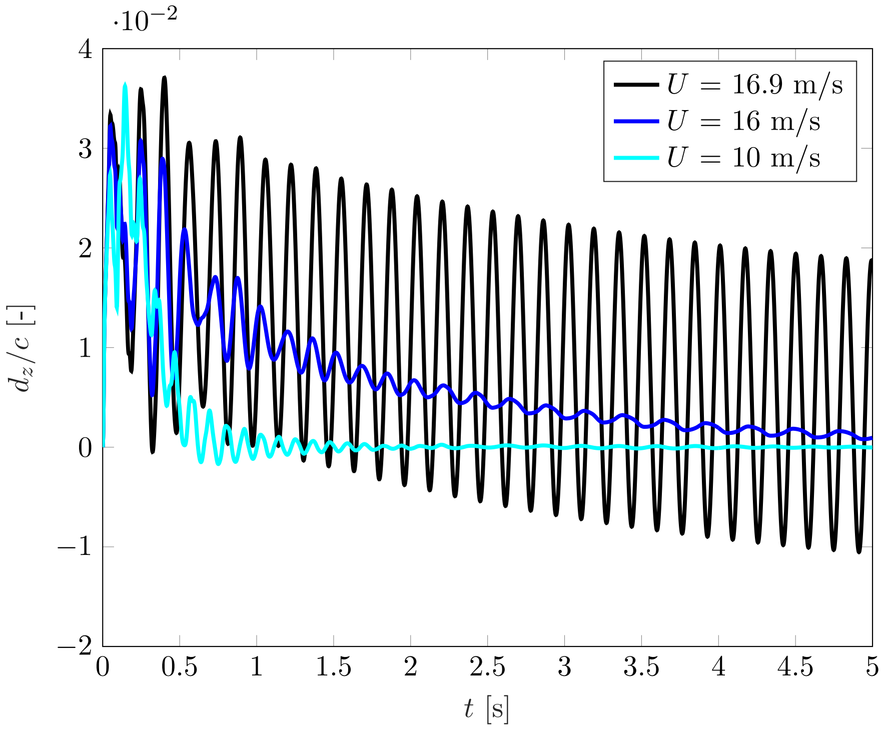


Figure 7.7: Aeroelastic response (out-of-plane displacement of the plate tip LE) of the configuration AR4-S0 for $U < U_\mathrm{f} = 16.9$ m/s and an initial perturbation using $t^* = 0.01$ s.

The aeroelastic responses of the swept configurations (S20 and S45) differ from the unswept case. As illustrated in Fig. 7.8 for the S45 configuration, the plate also features limit cycle oscillations at the flutter free-stream airspeed ($U_\mathrm{f} = 13.8$ m/s), but the oscillations do not occur around a non-oscillatory deflection of the plate. Hence, the LCO simply develops around the undeformed configuration of the plate. This trend is also confirmed by the experimental study, where no static deflection was recorded for any free-stream airspeed and any swept configuration [253]. At the flutter threshold, the LCO simply develops around the undeformed plate configuration.

In summary, the flutter characteristics obtained by the computational coupled model are in good agreement with those recorded by the experiment. The decreasing trend of both flutter speed and frequency with respect to the sweep angle is well captured. Also, it was shown that the aeroelastic response of the unswept plate (S0) features a non-oscillatory component that is decaying in time even at the flutter threshold. In absence of any experimental evidence, this mean component of the response could not be assumed to be the counterpart of the static deflection observed during the experiments for any free-stream airspeed up to the flutter threshold. Conversely, the response of the swept plate (S45) does not feature any significant mean component, in agreement with experimental measurements.

7.3.2 Study of limit cycle oscillations

The previous section was dedicated to the study of the flutter characteristics of the plate. The perturbation duration t^*, and hence the perturbation amplitude, was intentionally kept low in order to limit the effect of the nonlinearities that could arise for higher amplitudes. In this

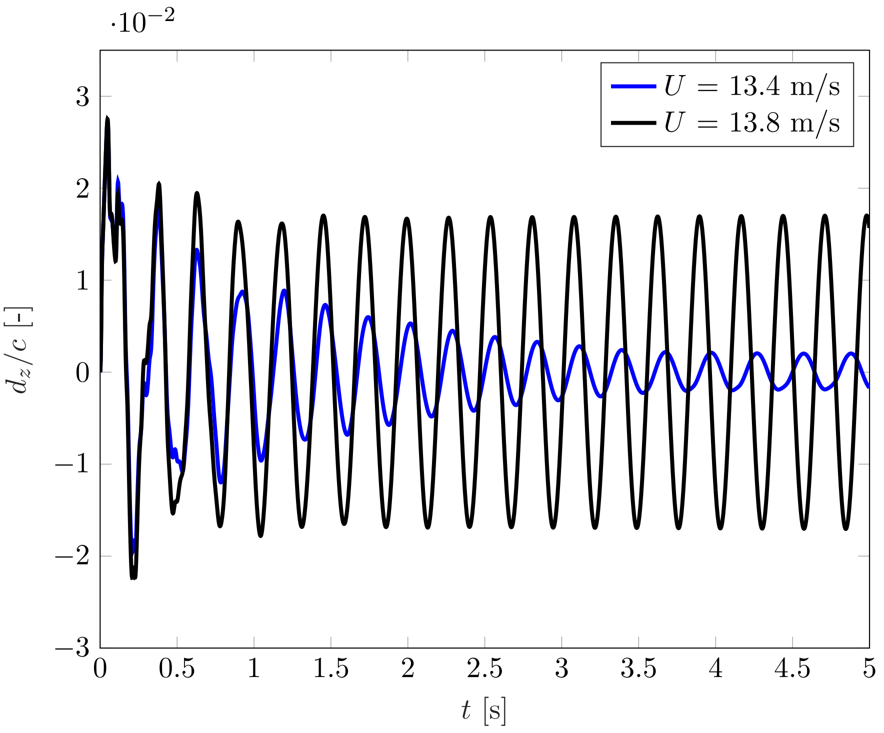


Figure 7.8: Aeroelastic response (out-of-plane displacement of the plate tip LE) of the configuration AR4-S45 for $U < U_\mathrm{f} = 13.8$ m/s and an initial perturbation using $t^* = 0.01$ s.

section, the impact of higher perturbation amplitudes on the aeroelastic response is assessed. Furthermore, aeroelastic responses for free-stream velocities higher than the flutter threshold are also simulated. Finally, this section also aims to provide more details on the flow around the deforming plate. The idea is to use flow visualization from CFD results, as experimental flow visualization had not been attempted and the VLM approach cannot provide the critical flow details of interest.

The following study focuses on the two plate configurations AR4-S0 and AR4-S45. The impact of the perturbation duration t^* on the unswept plate (S0) is first assessed. Fig. 7.9 illustrates the aeroelastic response at $U = 17.1$ m/s ($= 1.01U_\mathrm{f}$, i.e, very slightly above the computed flutter velocity) for different perturbation durations. For the two lowest values of t^*, the LCO response is the same as the one discussed for critical conditions in the previous section. After the transient phase, the response settles onto a limit cycle with the same frequency (6.2 Hz) and amplitude ($d_z/c = 0.018$) for both $t^* = 0.01$ s and $t^* = 0.02$ s. We also note the presence of a non-oscillatory deformation component, as previously discussed, that is still slightly damped in time although the free-stream velocity is set higher than the flutter threshold. This LCO is dominated by a bending motion of the plate, as shown in Fig. 7.10 by the contour of the z-displacement. Fig. 7.12 (a) depicts the z-displacement of both the leading and trailing edges at the plate tip, showing that their motion is quasi in-phase. Moreover, the larger motion amplitude of the trailing edge indicates the presence of some torsion (also seen in Fig. 7.10).

The aeroelastic response for the two highest values of t^* significantly differs from the small initial perturbation case. After the transient phase, the response converges to a limit cycle with higher frequency (9.8 Hz) and higher amplitude ($d_z/c = 0.046$) for both $t^* = 0.05$ s and $t^* = 0.1$ s. This new LCO is now dominated by a torsion mode of the plate, as illustrated by the contour of the out-of-plane displacement in Fig. 7.11. This is further demonstrated by the displacement of the leading and trailing edges of the plate tip that move now in opposite-phase, as depicted in Fig. 7.12 (b). A non-oscillatory component is also present in the response of the plate, but it

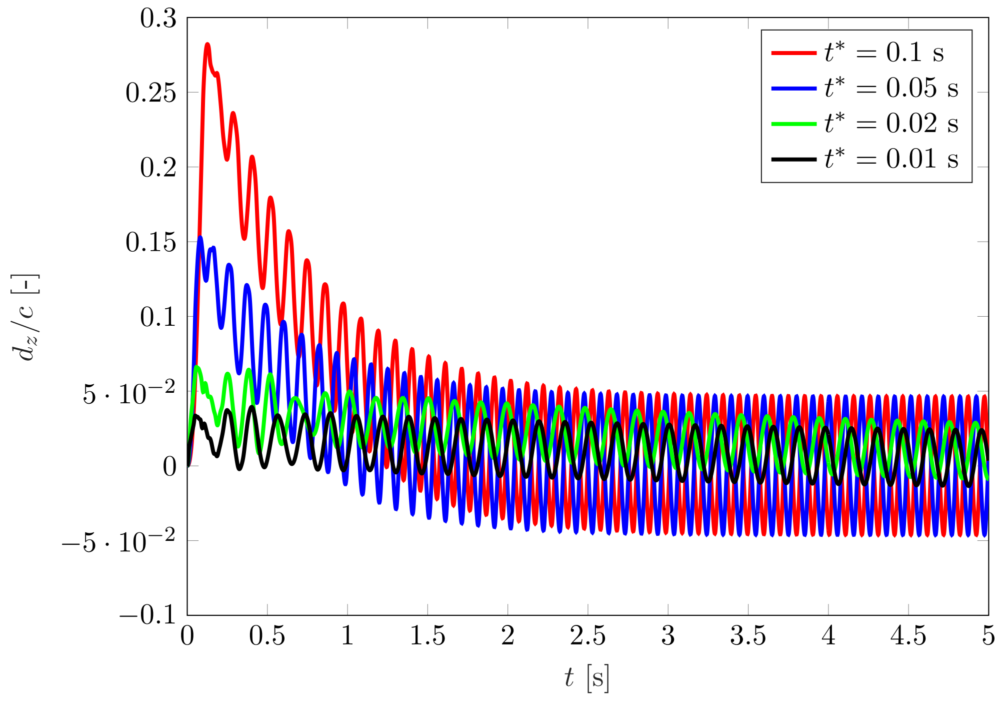


Figure 7.9: Aeroelastic response (z-displacement of the plate tip LE) of the configuration S0 at $U = 17.1$ m/s ($= 1.01U_\mathrm{f}$) for different initial perturbation durations.

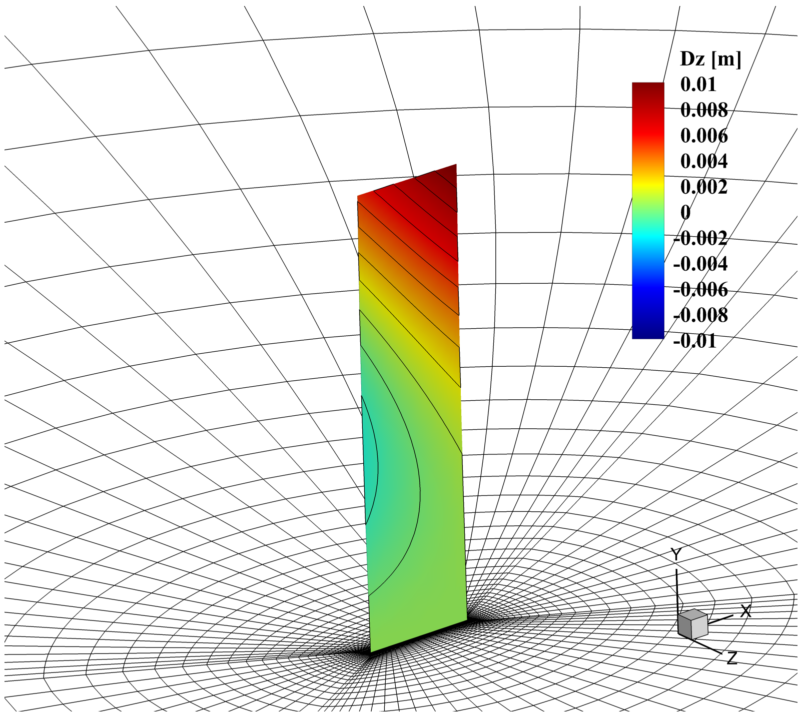


Figure 7.10: Contour of the z-displacement (peak value) for the S0 configuration at $U = 17.1$ m/s ($= 1.01U_\mathrm{f}$) and small initial perturbation duration ($t^* = 0.01$ s). The motion is dominated by bending.

is much more quickly damped than in the case of the LCO with lower frequency and amplitude.

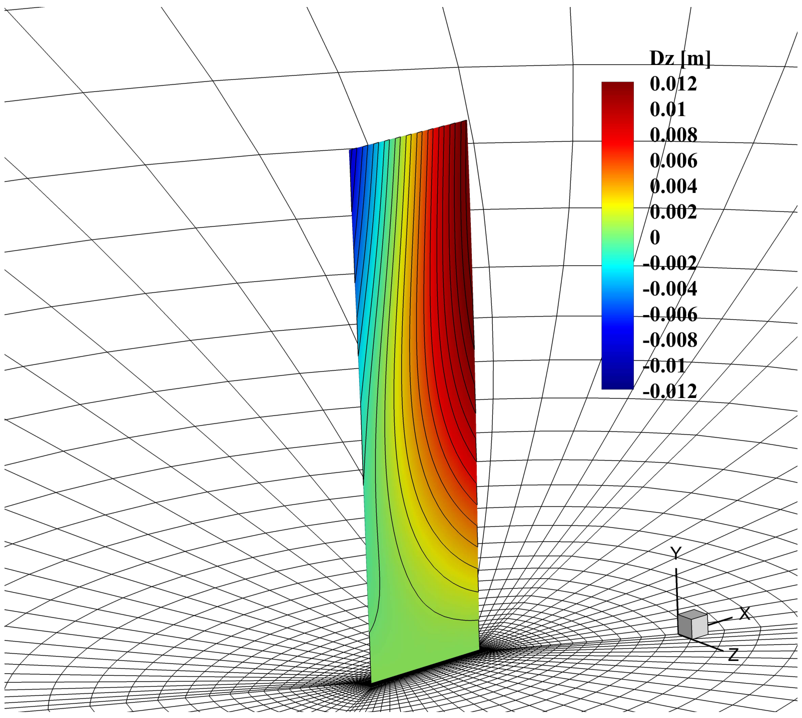
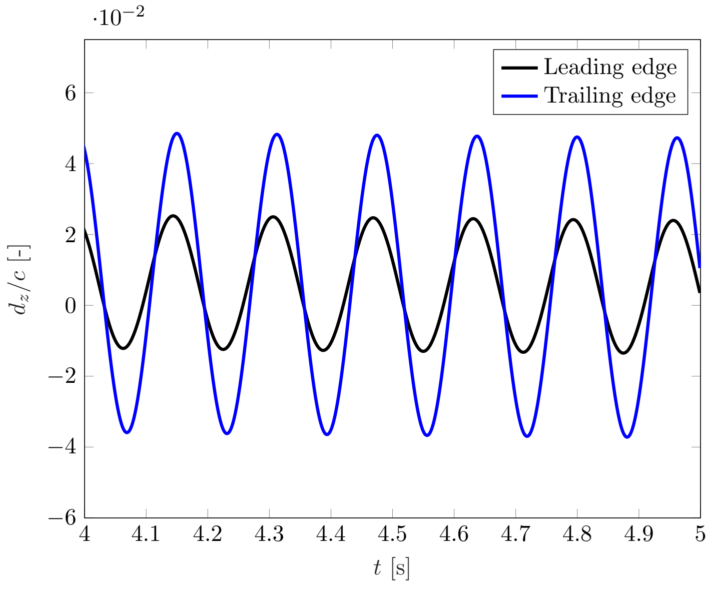
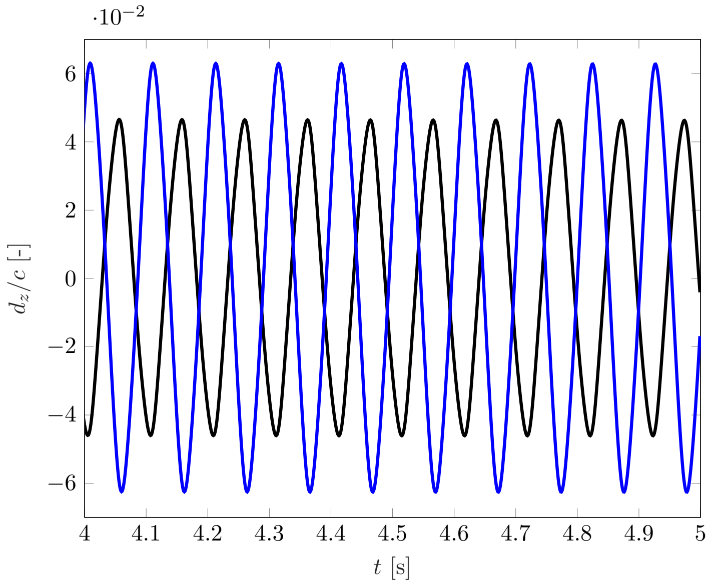


Figure 7.11: Contour of the z-displacement(peak value) for the S0 configuration at $U = 17.1$ m/s ($= 1.01U_\mathrm{f}$) and large initial perturbation duration ($t^* = 0.1$ s). The motion is dominated by torsion.

The computational analysis of the aeroelastic response of the plate subjected to different durations of the initial perturbation reveals the presence of two LCO that are characterized, for one particular free-stream airspeed, by two distinct frequencies and amplitudes. For a short initial perturbation, the response is attracted towards the low-frequency and amplitude LCO, while the response to longer initial perturbation is attracted towards the high-frequency and amplitude LCO. As introduced at the beginning of the chapter, the sensitivity of the trajectory response to initial conditions is seen as an intrinsic property of nonlinear aeroelastic systems. However, such sensitivity was explained in the context of a sub-critical bifurcation of the response for which $U < U_\mathrm{f}$. In the present simulations, an impact of the initial (structural) solution has been found at post-critical conditions ($U > U_\mathrm{f}$). The occurrence of a sub-critical bifurcation in the computational model has been investigated as well. No LCO response has been observed for any sub-critical free-stream velocity (the closest tested value to U_f is $0.98U_\mathrm{f}$) with a long duration of the initial perturbation (0.1 s), i.e. all responses were damped in time. Hence, the large set of parameters considered seems to clearly indicate that the bifurcation is supercritical, as observed in the experimental study. However, the occurrence of two distinct LCO responses was not observed during the experiment. To relate quantitatively the computed LCO to the one observed in the wind tunnel or to verify experimentally the existence of these two LCOs is very challenging for two main reasons. On the one hand, the actual motion of the plate is only partially captured by the camera. On the other hand, there is no mechanism to impose the same initial condition (perturbation) in the experiment as in the simulations. Qualitative analysis of videos recorded during the experiments suggests that the motion is dominated by bending and thus would correspond to the low-frequency LCO of the computational model, where the frequency discrepancy (comp. 6.2 Hz vs exp. 8.0 Hz) can be explained by the stiffening effect of the static deflection encountered in the experiments (as discussed previously). No experimental



(a) $t^* = 0.01$ s.



(b) $t^* = 0.1$ s.

Figure 7.12: Out-of-plane z-displacement of the leading and trailing edges at the wing tip for different initial perturbation durations at $U = 17.1$ m/s $= 1.01U_\text{f}$, configuration S0.

quantitative data on the amplitude is reported in Dimitriadis *et al.* [253], but the qualitative analysis of the recorded videos provides an estimation for the amplitude that is about one order of magnitude higher than the numerical solution. From a purely numerical point of view, the existence of two distinct LCO responses is confirmed in the simulated time window that was defined for this study (up to 10 s). It would be of interest to run some representative cases for a much longer time (e.g., at least 30 s of simulated time) in order to confirm that these LCOs have truly reached their stable steady state. For instance, it may happen that one type of LCO response transits, after a long time, towards the other type. However, running the simulations for such a long simulated time implies tremendous computational time (several weeks to months) which was not affordable for the numerous simulations that were needed in this study.

The same transition from low- to high-frequency LCO for configuration S0 can also be computationally triggered by increasing the free-stream velocity further beyond the flutter point, while keeping the smallest perturbation duration $t^* = 0.01$ s. This is depicted in Fig. 7.13 that shows the computed aeroelastic response of the plate for different post-critical velocities. Note that in this case the short initial perturbation is used for all velocities. It can be observed that, at $U = 21$ m/s ($1.24U_\mathrm{f}$), a transition from the low- to the high-frequency LCO occurs. This transition is not observed at lower velocity. The post-transition response has the same qualitative characteristics as the high-frequency LCO that was triggered at $U = 17$ m/s ($1.01U_\mathrm{f}$) by applying a longer initial perturbation: the motion is again dominated by torsion with a motion of the leading and trailing edges in opposite phase.

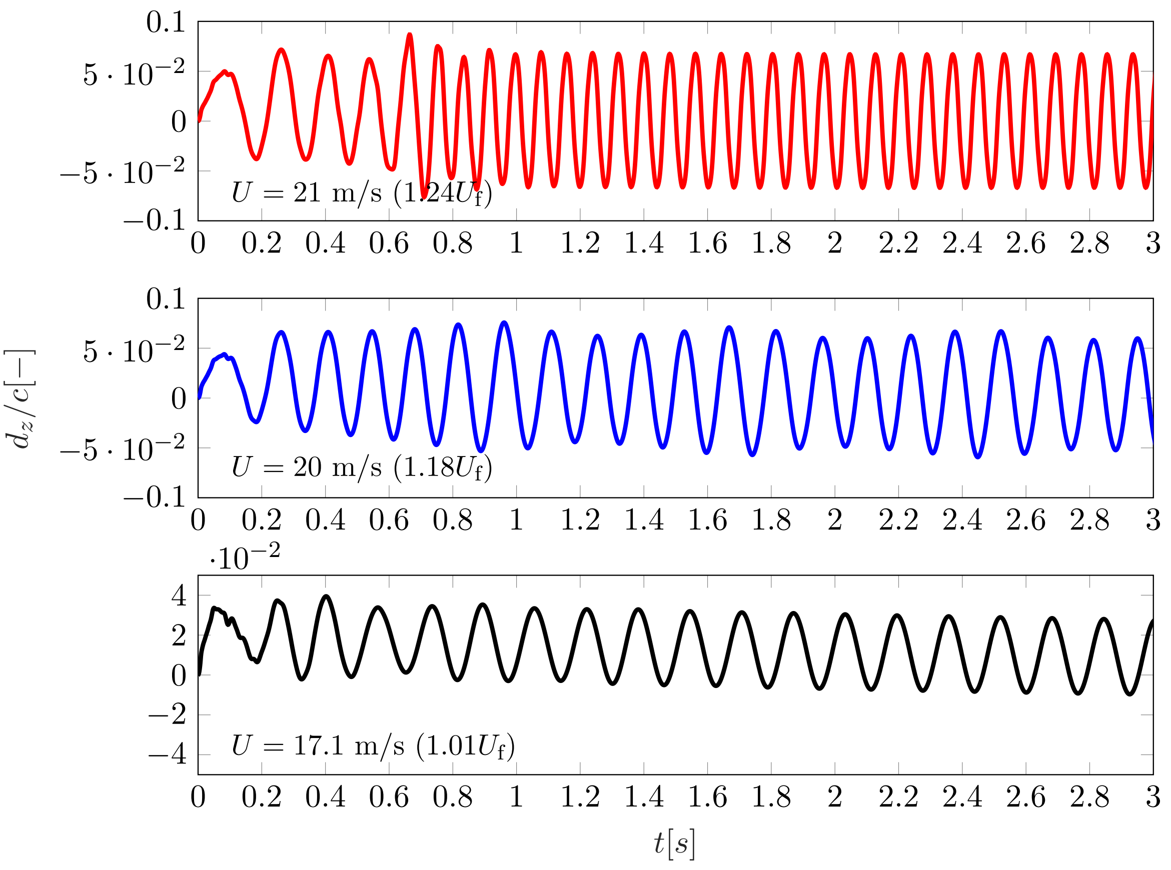
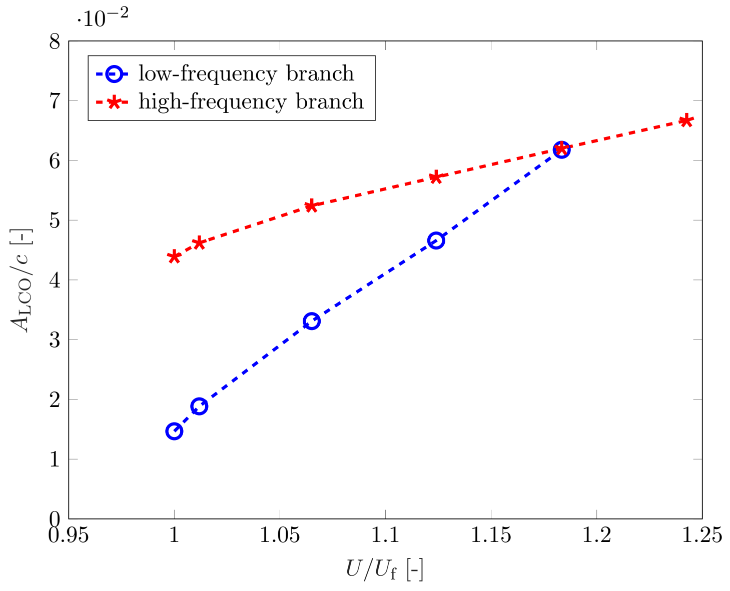
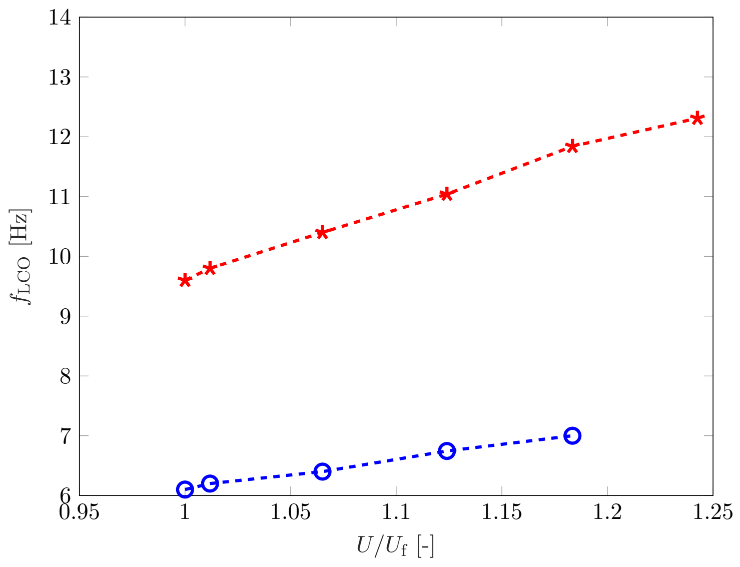


Figure 7.13: z-displacement at the plate tip of the leading edge for configuration S0 and for different post-critical free-stream velocities ($t^* = 0.01$ s). Note that the plotted simulation time is truncated at 3 s for clarity, but simulations have been run up to 5 s with no change in the response.

The post-critical limit cycle amplitude and frequency of the aeroelastic response of the S0 plate are summarized in Fig. 7.14 as a function of the free-stream airspeed. Similarly to the experiments[6], the computed LCOs show that the amplitude increases monotonically at supercritical airspeeds, which is characteristic of a supercritical bifurcation. The LCO frequency follows the same trend. Fig. 7.14 also represents the two LCO branches that can be triggered



(a) LCO amplitude.



(b) LCO frequency.

Figure 7.14: LCO amplitude and frequency of the aeroelastic response (z-displacement of the leading edge at the plate tip) as a function of the free-stream airspeed, normalized by the flutter speed, configuration S0.

either by varying the perturbation duration t^* or the free-stream airspeed, as discussed above. In the frequency plot, the two branches are well separated. On the other hand, the LCO amplitude of the low-frequency branch rapidly converges towards the high-frequency branch and merges into it around $U = 1.18U_{\mathrm{f}}$. For a post-critical airspeed up to $1.18U_{\mathrm{f}}$, there is a threshold in the initial perturbation duration (and hence in the perturbation amplitude) that determines

[6]Dimitriadis *et al.* [253] observed a growing LCO amplitude but do not provide quantitative data.

the branch on which the LCO frequency and amplitude will lock. The analysis of the responses to the four pre-determined values of t^* at several post-critical airspeeds up to $1.24U_\mathrm{f}$ allows us to roughly estimate this boundary between the two possible LCOs in the parameter space (t^*, U/U_f), as shown in Fig. 7.15. Four types of responses can be roughly identified. The first

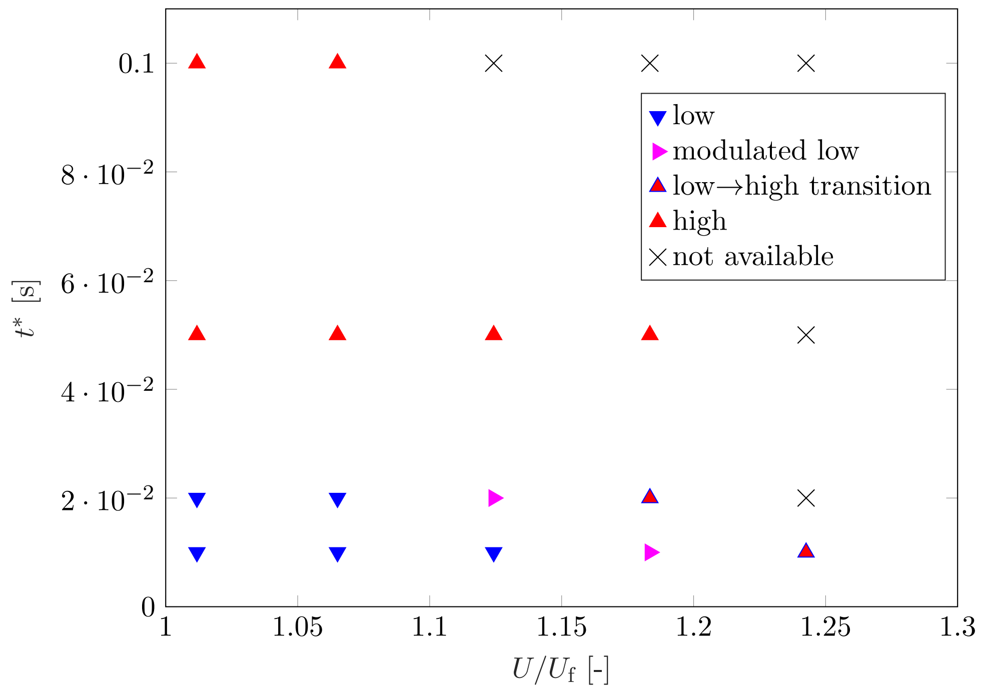


Figure 7.15: Types of LCO responses obtained with respect to the perturbation duration t^* and the free-stream airspeed U. Not available data are mostly computations that experience robustness issues and for which no result could be extracted.

three have already been presented and discussed: low-frequency LCO, high-frequency LCO and a transitional response from low- to high-frequency LCO. The fourth type of response is very similar to the low-frequency LCO but features a modulated amplitude which appears to be a precursory sign of transition towards high-frequency LCO if one increases the free-stream airspeed or the perturbation duration[7] (see case $U = 20$ m/s in Fig. 7.13). Fig. 7.15 clearly shows that the theoretical threshold value of t^* decreases when the free-stream velocity increases, up to the point where only the high-frequency response can be obtained at the lowest t^*. Note that no exploitable results could be obtained for simultaneously large initial perturbation and high velocity. The main reason is the large deformation amplitude during the transient phase (before reaching the LCO) that imposes a large mesh deformation. The mesh deformation process is incremental, i.e. each new deformed mesh is obtained from the deformed mesh at the previous coupling iteration. Once the quality of the mesh starts to deteriorate, this process can quickly lead to invalid meshes which cause the entire simulation to numerically diverge.

The flow around the plate in configuration S0 is now analyzed by means of flow vizualization. Particular characteristics of the flow, such as separation and vortex shedding, are usually sought to explain the occurrence of the two LCO branches. No massive flow separation other than the expected tip vortices (three-dimensional lifting effects) has been found to occur in the flow for both LCO branches and any free-stream airspeed considered (up to $U = 1.24U_\mathrm{f} = 21$ m/s). However, in the high-frequency response of the plate, one can observe a stable (i.e. that does not appear to detach and be shed in the wake) leading edge recirculation bubble that extends on the suction side of the plate. This is illustrated in Fig. 7.16 that shows the pressure contour and

[7]Or also potentially the simulated time window, as discussed earlier.

the (uncoloured) velocity streamlines on the plate in configuration S0 at velocity $U = 1.01U_\mathrm{f}$ and maximum deformation. The leading edge recirculation is marked on the surface by the low pressure yellow band while the core of the recirculation is well identified by the streamlines. Downstream of the leading edge recirculation bubble, the flow appears to stay attached to the plate. The generation of the LE recirculation bubble in the high-frequency LCO can be explained

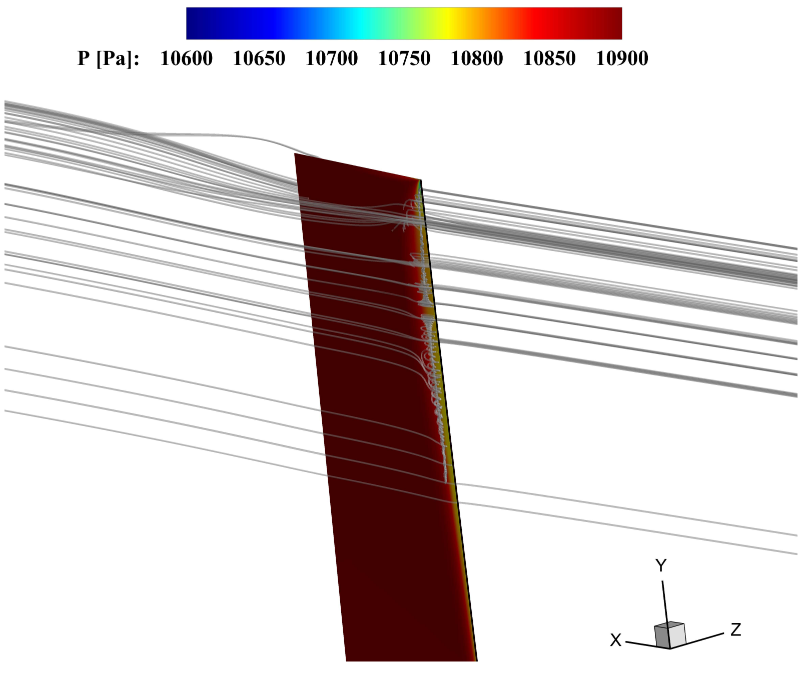


Figure 7.16: Pressure contour and velocity streamlines on the second half-span and at maximum deformation of the high-frequency response. $U = 1.01U_\mathrm{f}$, configuration S0. The flow is from right to left.

by the combination of two factors: on the one hand the torsion-dominated motion of the plate generates a higher local angle-of-attack, and on the other hand the sharp leading edge of the plate favours the detachment of the boundary layer. As the torsion, and so the effective local angle-of-attack, of the plate increase in the spanwise direction, the size of the recirculation bubble increases in that direction too. This can be observed in Fig. 7.17 that shows, on the suction side of the plate, the contour of the negative streamwise friction coefficient, that is again interpreted here as a marker of the separation. The pressure-coloured velocity streamlines are superposed to mark again the core of the recirculation bubble. In particular, the significant part of the recirculation zone begins at about 65% of the span and becomes largest between 75% and 93% of the span before decreasing towards the plate tip where it collapses due to the interaction with the tip vortices. Because of the periodic deformation of the plate, the recirculation bubble is not present during the entire cycle but periodically appears on the suction side of the plate. The existence of the LE recirculation bubble has been observed in high-frequency LCO for all free-stream airspeeds considered. The proposed explanation for the occurrence of two LCO branches in the computational model is therefore the nonlinear aerodynamic effect of the recirculation. By looking at how the LCO branches merge in Fig. 7.14 (a), we can reasonably assume that there exists a deformation amplitude threshold (a critical t^* here) above which the recirculation bubble can fully develop and increase locally the suction near the leading edge, therefore changing the response trajectory of the plate itself towards a torsion-dominated stall flutter. The suction effect brought by the recirculation bubble was already well illustrated in Fig. 7.16 by the leading edge low pressure band.

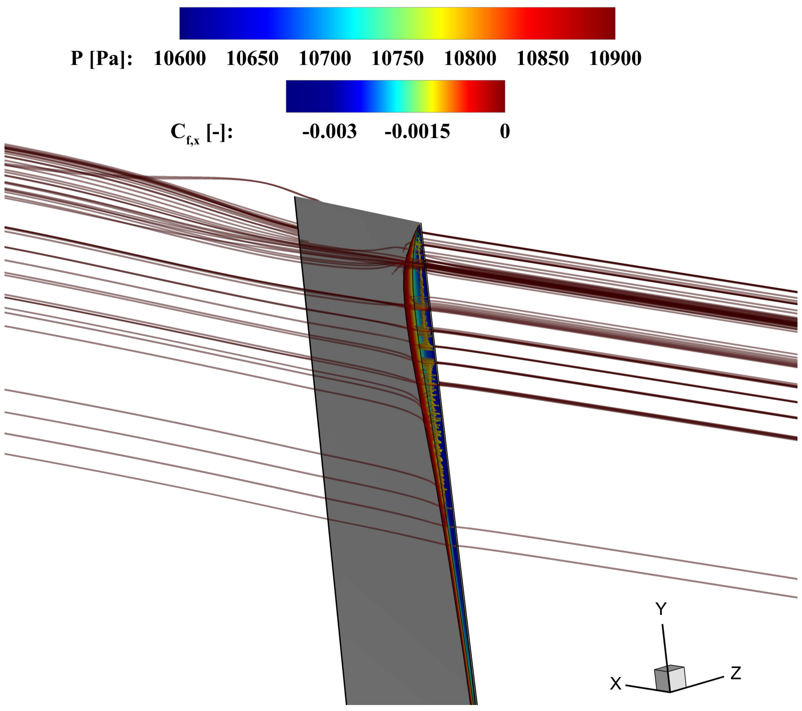

P [Pa]: 10600 10650 10700 10750 10800 10850 10900
$C_{f,x}$ [-]: -0.003 -0.0015 0
Y
X
Z

Figure 7.17: Contour of the streamwise negative skin friction coefficient and pressure-coloured streamlines on the second half-span and at maximum deformation of the high-frequency response. $U = 1.01U_\mathrm{f}$, configuration S0. The flow is from right to left.

The post-critical behaviour of the swept configuration S45 significantly differs from the one observed in the unswept case. Although the amplitude of its response is comparable to the S0 configuration at the flutter airspeed, it shows a sharp increase with respect to the free-stream airspeed. Computations at post-critical conditions reveals a fast growth of the amplitude in the range $U = 13.8 - 15$ m/s ($U_\mathrm{f} - 1.09U_\mathrm{f}$), as illustrated in Fig. 7.18. At a velocity 9% above the flutter threshold, the displacement of the leading edge at the plate tip exceeds $d_z/c = 0.13$, which is already twice as large as the displacement obtained for the S0 configuration at a velocity almost 25% above its own flutter airspeed ($d_z/c = 0.066$). A qualitative analysis of the records from the wind tunnel study also confirms that the response of the swept plate configuration is characterized by a higher amplitude when compared with the unswept configuration at similar free-stream airspeeds. As previously mentioned, such large amplitudes of the aeroelastic response represent a significant challenge for the numerical approach because the resulting large deformation of the computational grid leads to meshes of very poor quality. The fluid solution is thus strongly impacted: numerical instabilities start developing and eventually lead to the divergence of the solution, as illustrated by the case $U = 1.09U_\mathrm{f}$ in Fig. 7.18. Several simulation parameters were adjusted to try to stabilize the computation (typically turbulence model, CFL number, multi-grid levels, mesh deformation artificial stiffness and number of mesh deformation iterations), but none of these attempts were successful in significantly improving the robustness of the solver. Note that this lack of robustness is further exacerbated here by the use of a structured RANS mesh, as such meshes are more prone to deterioration when deformed.

The impact of the perturbation duration t^* is also assessed for the swept S45 configuration. In contrast to the unswept configuration, no LCO branch bifurcation has been observed in the range of t^* values tested. For the largest initial perturbation considered ($t^* = 0.1$ s), the aeroelastic response quickly reaches the limit cycle, as illustrated in Fig. 7.19 for the post-critical free-stream airspeed $U = 1.03U_\mathrm{f}$. The limit cycle of the swept plate S45 is characterized by a complex motion that combines both bending and torsion, as shown in Fig. 7.20. This figure

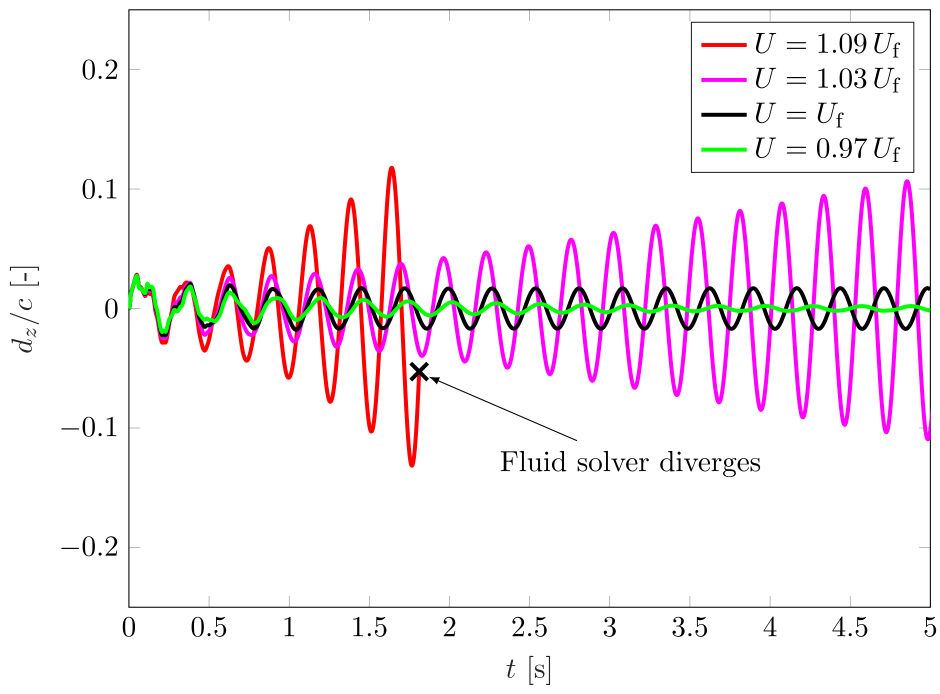


Figure 7.18: Aeroelastic response (z-displacement of the plate tip LE) of the configuration S45 at different free-stream velocities ($t^* = 0.01$ s).

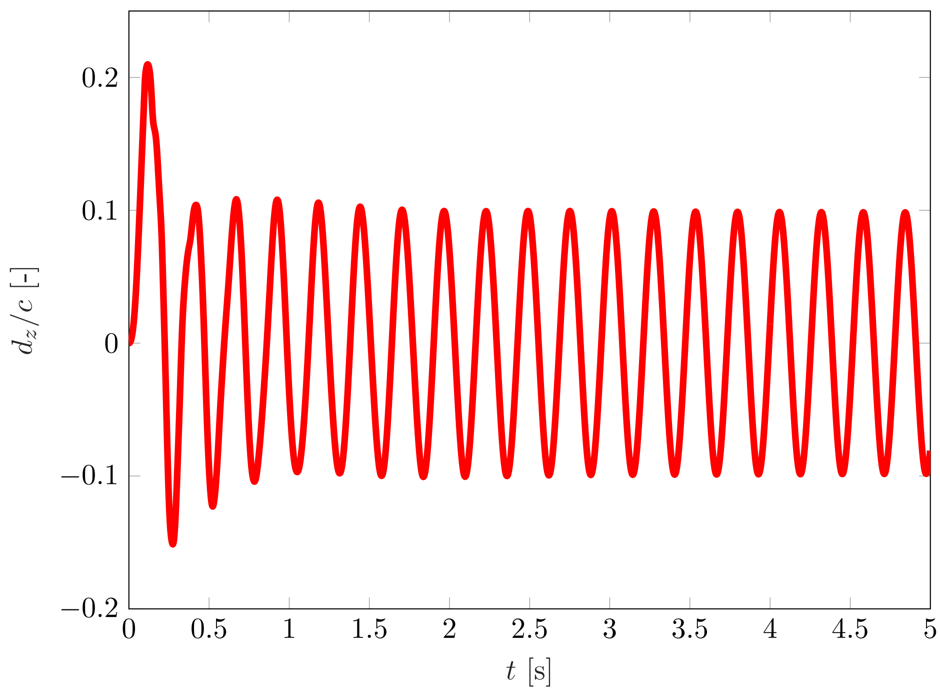


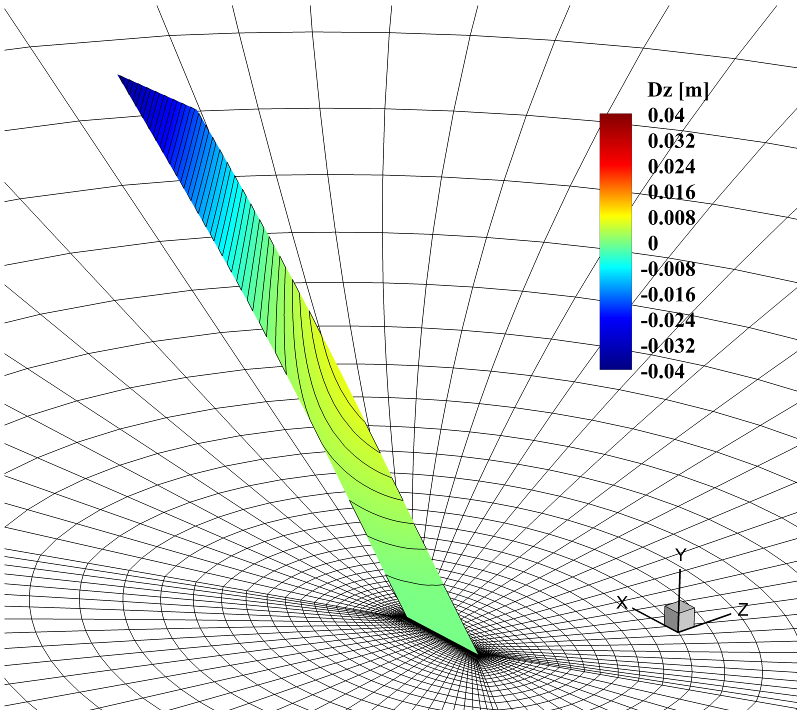
Figure 7.19: z-displacement of the leading edge at the plate tip for a free-stream airspeed $U = 1.03U_f$ and a perturbation duration $t^* = 0.1$ s; configuration S45.

also compares qualitatively the deformation of the S45 plate between the model simulated with CUPyDO and the experimental model in the wind tunnel at similar post-critical U. In both cases, the maximum amplitude of the deformation is clearly located at the trailing edge of the plate tip and the deformation pattern can be divided into three distinct parts. The first part extends from approximately 69% of the span to the plate tip (100%), where the plate is literally waving about in the z-direction. The second part extends from 22% to 69% of the span and mostly shows an out-of-plane displacement of the leading edge which is in opposite phase compared to the first part of the deformation pattern. The last part of the plate experiences only marginal deformation, as it is closer to the clamped extremity.

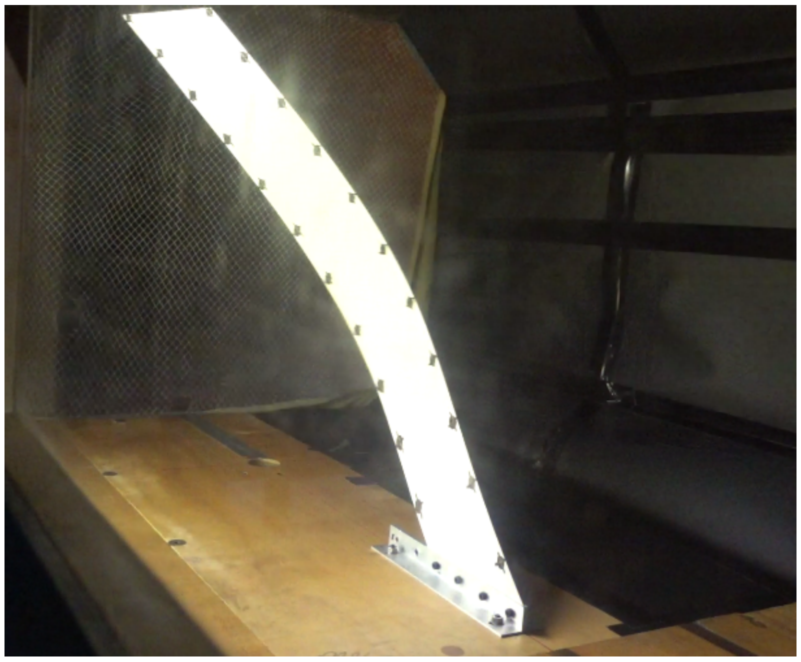
The flow around the plate in configuration S45 is also investigated by means of flow visualization. Fig. 7.21 shows the flow velocity streamlines around the plate at a free-stream velocity $U = 1.01U_\mathrm{f}$ and for an initial perturbation with $t^* = 0.1$ s. Similarly to the unswept configuration, no massive flow separation is observed, even at the highest amplitude obtained at $U = 1.09U_\mathrm{f}$[8]. The absence of flow separation can be explained by the combination of several factors. First, the amplitude of motion of the span sections located close to the clamped extremity of the plate is relatively small and the resulting local angle of attack is too low to produce flow separation. Then, for span sections further away from the root, the motion of the plate is such a complex combination of bending and torsion that the local angle of attack might remain under the dynamic stall angle. More specifically, for each two-dimensional span section, the analogy with a pitch-plunge flat plate[9] reveals that a nose-up-pitch-induced angle of attack could be compensated by an upward plunge motion, thus limiting the effective angle of attack. The third factor is a consequence of the significant three-dimensional effects of the flow field. Both the spanwise flow deflection towards the plate tip, induced by the swept geometry and the lift-induced tip vortices significantly influence the vortex dynamics and tend to smooth out the unsteadiness related to vortex shedding that would otherwise typically appear in a purely two-dimensional flow simulation around a plate section. This effect of tip vorticity was already highlighted in the work of Taira and Colonius [256] who performed flow simulations around three-dimensional rigid flat plate wings featuring comparable aspect ratios (up to 4) but lower Reynolds numbers (of the order of 10^2) than in the present case. Neither the occurrence of a low-speed LE vortex nor any particular change in the flow structure have been been observed for the S45 configuration in the range of U and t^* considered. In this configuration, the absence of two distinct LCO responses is then correlated to the absence of distinct and nonlinear flow features, that could otherwise appear when varying the free-stream airspeed or the perturbation duration in the range considered in this study.

[8]As mentioned and illustrated in Fig. 7.18, no higher deformation could be simulated due to the limited robustness of the mesh deformation procedure.

[9]The 2D pitch motion is the analogy of the 3D torsion of the plate and the 2D plunge motion is the analogy of the 3D bending.



(a) Contour of the plate surface z-displacement computed with CUPyDO ($U = 14$ m/s $= 1.01U_\mathrm{f}$).



(b) Snapshot of the plate motion in the wind tunnel ($U = 13.8$ m/s $= 1.03U_\mathrm{f}^\mathrm{exp}$).

Figure 7.20: Qualitative comparison of the plate LCO between computation with CUPyDO and experimental observation, configuration S45.

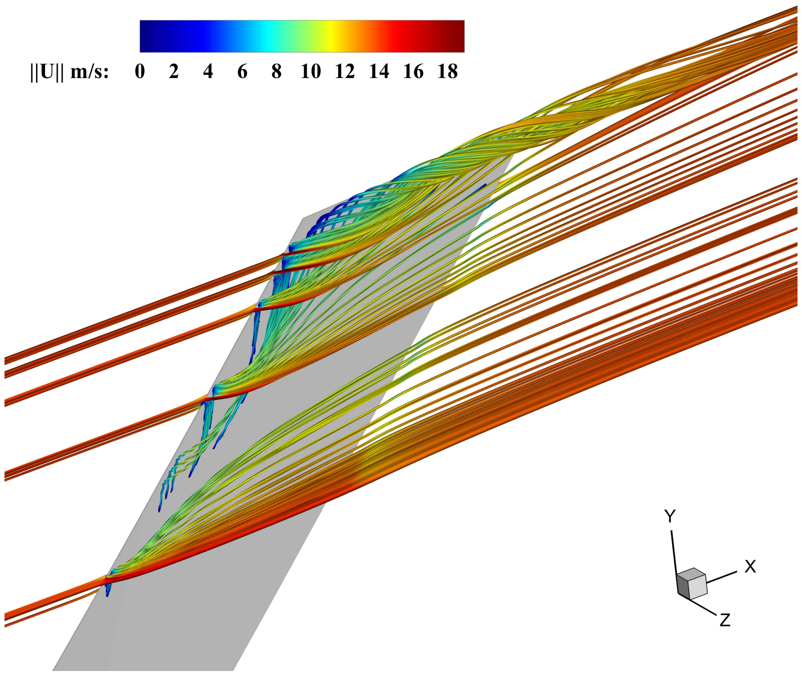


Figure 7.21: Flow velocity streamlines at the plate tip and crest amplitude of the deformation. $U = U_{\mathrm{f}}$, configuration S45.

Summary of chapter 7

The last chapter of this thesis presented the application of the coupling between SU2 and Metafor using CUPyDO for aeroelastic simulations of a very flexible cantilever flat plate wing. This test case was based on an experimental study performed by Dimitriadis *et al.* [253]. Several geometrical configurations of the plate with an aspect ratio of four and varying sweep angles have been considered.

Linear flutter properties (speed and frequency) have been determined by computing the aeroelastic response of the plate to small perturbations. Results are found to be in good agreement with the experimental observations and with other numerical results obtained by low-fidelity models based on the Vortex Lattice Method (VLM) and the Doublet Lattice Method (DLM). In particular, flutter characteristics follow the same trend: a decrease of the flutter speed and frequency with the sweep angle. Overall, the results computed by CUPyDO are closer to the experimental results than the VLM results are. The LCO flutter response of the unswept wing has been found to establish itself around a non-oscillatory and steadily decaying deformation component of the plate. Although sub-critical and critical static deflections of the plate were also experimentally observed for this configuration, relating the computed non-oscillatory deflection with the one observed experimentally is challenging as they differ in several aspects: the amplitude (up to 100 times larger in the experiment), the duration (permanent in the experiment, decays in time in the computational model) and the conditions for this deflection to occur (produced by the initial perturbation in the computational model while naturally developing in the experiment).

In a second step, the post-critical aeroelastic response of the plate has been computed for an unswept and a swept ($\Lambda = 45°$) configuration. At post-critical free-stream air-speeds, the response of the unswept plate features a supercritical bifurcation which can turn into two distinct LCO depending on the amplitude of the initial perturbation, controlled here by the perturbation duration t^*. The existence of threshold values of t^* has been shown for free-stream airspeeds up to $U = 1.24U_\mathrm{f}$. For t^* below the threshold value, the response locks on a low amplitude and low frequency LCO that oscillates around a decaying non-oscillatory component of the deformation. It has also been found that this LCO is dominated by a bending motion of the plate. For t^* above the threshold value, the response locks on a higher-amplitude and higher-frequency torsion-dominated LCO with no mean deformation. The threshold value of t^* has been found to decrease when increasing the post-critical free-stream airspeed up to a point where only the high-frequency LCO is obtained for the whole set of perturbation durations considered in this study. Flow visualization has been used to show that the high-frequency LCO response is related to the occurrence of a leading-edge recirculation bubble which is not observed in the low-frequency LCO. The occurrence of two distinct LCO responses has not been observed during the experiment, and the correspondence between one of the two computational LCO branches and the actual response of the plate in the wind tunnel is unclear. Qualitative deformation patterns and frequency measurements are comparable to the low-frequency computational response, but the experimental amplitude of deformation remains significantly larger than in the computational model.

The post-critical response of the S45 swept plate differs from the unswept configuration. Small initial perturbations are quickly amplified in time until they reach a limit cycle which is characterized by a much larger amplitude than in the unswept case. The post-critical amplification factor quickly increases with increasing free-stream airspeed. At 9% above the flutter airspeed, the maximum computed displacement of the leading edge at the plate tip is already twice as large as the displacement obtained for the unswept plate at a velocity almost 25% above its own flutter threshold. Furthermore, static deflection has been observed neither for the numerical simulations nor for the experimental measurements and a qualitative comparison of the deformation pattern between the computational and experimental responses has shown satisfactory agreement. Finally, the limit cycle response of the swept wing does not feature multiple branches as for the unswept wing, at least in the range of t^* and U for which proper results have been obtained. The absence of transition from one LCO to another has been corroborated by the absence of significant changes in the flow structure around the plate, such as the potential occurrence of a leading-edge recirculation zone, in contrast to the unswept case.

Finally, because of the body-fitted ALE strategy followed for the flow calculation, several LCOs with large amplitudes could not be properly simulated. In particular, large deformations of the structure tend to destabilize the incremental mesh deformation procedure of SU2, which finally leads to invalid meshes and the divergence of the flow solution. This highlights one of the typical limitations of the body-fitted ALE approach used by the fluid solver, but does not question the validity of the coupling approach used by CUPyDO.

Conclusion

Fluid-Structure Interaction problems have gained increasing interest from industrial designers and scientists because of their significant complexity making efficient designs more challenging. This thesis has focused on the computational approach for studying or designing FSI systems. Fluid and solid solvers are used to solve the governing equations for each physics with a high level of fidelity, but the complete multi-physics solution also requires coupling/interfacing effects to be accounted for by coupled computational models. Such models bring additional challenges that are related to, for instance, the coupling technology, the solver inter-communication or the coupling algorithm.

In this context, this thesis presented the development, verification and application of the tool CUPyDO that has been designed as a FSI coupling environment for single-physics black-box solvers. The partitioned approach followed in this work allows the intrinsic features of the individual solvers to be leverage, and a wide range of applications can be envisaged if the coupling interface is flexible enough. CUPyDO has been developed with a focus on computing efficiency, flexibility and user-friendliness. A modern programming technique is used, that combines C++ and Python, two standard object-oriented languages, into the same software architecture.

The fluid-structure interaction model

This thesis has been divided into three parts. The first part, composed of three chapters, is dedicated to the description of the fluid-structure interaction problem. Chapter 1 has described the main phenomenological aspects of the FSI problem. A classification has been proposed, depending on the characteristics of the fluid flow, but not all categories have been considered in this thesis. In particular, the examples discussed were restricted to steady external flows. Three mechanical FSI phenomena have been presented: VIV, galloping and flutter. Finally the importance of FSI in the engineering design process has been discussed.

Chapter 2 has presented the mathematical model of a coupled FSI system. The two principal strategies, i.e. the monolithic and the partitioned approaches, have been reviewed, as well as the general formalism for the description of the fluid and solid motion. While a Lagrangian formalism is still used for the solid, the ALE formalism combined with a moving mesh is used in the fluid domain. The governing equations for each physics have been presented before introducing the coupling conditions that express the continuity of displacement, force, temperature and heat flux through the fluid-solid interface. These interface conditions are the central part in the definition of the coupled problem. The most relevant non-dimensional parameters for FSI have finally been recalled.

Chapter 3 has been presented as an important review of the numerical aspects of FSI. Mechanical and thermal coupling algorithms have been presented based on the fixed point formulation of the coupled problem. Their numerical stability has also been discussed, and the mass number Ma and Biot number Bi have been shown to represent a good measure of the stability margin. For the mechanical coupling, the under-relaxed block Gauss-Seidel coupling algorithm is used to stabilize the iterative process in case of high added-mass effects. More efficient Newton-based methods such as the interface quasi-Newton with approximation of the inverse Jacobian from least-square (IQN-ILS), providing better convergence rate, have also been reviewed. For the thermal coupling, stability is achieved by selecting the appropriate scheme depending on the characteristic Biot number of the problem. Stability can further be improved by using a numerical heat transfer coefficient. Also for the thermal coupling, specific time advancement strategies that take into account the significant difference in physical time scales between the coupled domains have also been reviewed. The fluid-structure interface treatment can be achieved through two main approaches that have been discussed. The non-conformal approach deals with a structural domain that is embedded in a Cartesian fluid mesh. The fluid solver is thus based on the immersed-boundary method, which complexifies the treatment of the interface conditions. The approach that has been followed in this thesis is the conformal approach, in which the fluid mesh conforms with the structural boundary, making the interface treatment more explicit. The fluid solver is thus based on the ALE formalism with a dynamically deforming fluid mesh. Using a conformal interface mesh does not automatically imply that the discretization in each domain is matching. Interpolation is thus required to transfer solution data from one interface to the other. Several interpolation methods have been presented and discussed, for both conservative and consistent approaches. The RBF approach has been implemented in CUPyDO, as it is suited for partitioned coupling of black-box solvers and does not require knowledge of the interface topology. The fluid mesh dynamics of the ALE formalism has been reviewed as well. The mesh deformation problem can be expressed by partial differential equations, with specific boundary conditions, for the new position of the grid nodes. The pseudo-elasticity analogy is a typical example, which formulates the mesh deformation problem as a linear elastic solid deformation computed with a finite element approach. Another approach is to rely on an interpolation of the boundary displacement to the volume mesh, which advantageously does not require any information on the mesh topology. The same RBF approach can thus be applied for mesh deformation as for interface interpolation. Other interpolation methods for mesh deformation include the inverse distance weighting (IDW) or sphere relaxation. The importance of the Geometric Conservation Law (GCL) for ALE schemes has been briefly discussed and related to the accuracy and stability of the underlying numerical schemes. Despite contradictory results in the literature, the development of GCL-compliant ALE schemes is still encouraged. Finally, the single physics solvers used in this thesis have been introduced. For the fluid part, the SU2 solver is used to solve the compressible unsteady RANS or Euler equations on a potentially deforming unstructured grid using a GCL-compliant ALE formalism. The mesh deformation is achieved using the pseudo-elasticity analogy method. For the solid, several solvers have been considered. Metafor is a nonlinear finite elements code for thermo-mechanical problems involving large solid displacements and deformations. It has been used for mechanical FSI problems. GetDP is a general environment for the treatment of discrete PDE-based problems with the finite element method as well. GetDP has been used for thermal FSI problems. Finally, an in-house rigid body integrator code has been presented for solving one and two degree-of-freedom rigid motions that are characteristics of simplified VIV and aeroelastic models.

The coupling tool CUPyDO

The second part of this thesis, composed of two chapters, is related to the review of multi-code coupling technologies and the development of the coupling environment CUPyDO. Chapter 4 has first proposed a review of different inter-code communication methods and coupling architectures. For inter-code communication, the MPI protocol is expected to offer the best performances, especially for HPC-compliant architectures. The chosen coupling architecture relies on a unified approach as it offers better flexibility and direct communication between the solvers, that neither the master-slave nor client-server architectures can achieve. Several state-of-the-art coupling software have been reviewed and compared based on relevant characteristics such as the availability of coupling schemes, the HPC-compliance, or the intrusive character of the coupling tasks. This has motivated the exploration of an original coupling technology based on the Python wrapping of C/C++ code and the development of CUPyDO.

The implementation of CUPyDO has been detailed in Chapter 5. CUPyDO is designed as an hybrid C++/Python environment for the coupling of independent black-box solvers. Effort has been invested in the coupling flexibility and the modularity, which limits code invasiveness for the coupled solvers. By using a unified coupling approach, the coupled solvers can be manipulated as library objects instead of standalone executables, providing flexibility and usability to the coupling. The Python wrapping methodology, the central technology of CUPyDO, has been introduced and illustrated in the context of a very simple C++ example. It has been shown how the different classes of the core solvers code can be embedded into an API layer and exposed to Python by using SWIG, while conserving the object-oriented structure of the core language. The Python wrapping methodology has also been applied and detailed for the interfacing of the fluid solver SU2. The resulting Python layer is used as an API that interfaces the functionalities of the solver in a very intuitive way, while these functionalities remain executed under the original compiled language for best efficiency. The structure of CUPyDO is also based on the Python wrapping technology. It comes with several built-in coupling functionalities, such as coupling algorithms and non-matching mesh interpolation schemes, which are accessible from a high-level API that does not require strong knowledge of the Python language. For a more advanced usage of the environment, the structure of CUPyDO offers the possibility to customize the coupling tasks or even implement new functionalities that can be easily derived from existing ones. The detailed architecture of CUPyDO is composed of a C++ kernel, gathering all the computationally intensive coupling tasks, and a Python API that interfaces these core functionalities at a high level. The kernel is also used to link with external libraries such as OpenMPI and PETSc to support HPC capability. The Python API is divided into several layers. The Python *Utility* layer defines fundamental functionalities, such as MPI-related routines, interface data containers and linear algebra capabilities. The interface data containers are vector-like structures that support solver inter-communication as they can store and perform parallel arithmetical operations on data which are exposed by each coupled solver. The central part of the environment is the *Core* layer which provides the high-level management for the coupling algorithms, interface solution transfer and interface solution interpolation. The object-oriented structure of CUPyDO allows the user to generate new coupling algorithms directly in Python with limited effort. The *Interface* layer is specifically designed to ensure the compatibility between the core of CUPyDO and the coupled solvers while keeping a high level of flexibility. It is used to accommodate each independent solver wrapper to the algorithmic part of the environment. This is achieved by filling corresponding interface modules (one per coupled solver) with the Python-exposed functionalities. These interfacing modules directly inherit from a common generic solver class that is directly used by the *Core* layer. The management of parallel computation has been described by considering two types of communication between processes. While intra-communication is seen as black-box as they involve data exchange within an individual coupled solver, inter-communication is used to exchange data through the fluid-structure interface, between processes of distinct coupled solvers. New MPI communicators are defined as a subset of the standard `MPI_COMM_WORLD` in order to identify and tag groups of processors in which data exchange is needed, for both intra- and inter-communication. The communication network resulting from the heterogeneous MPI partitioning of each solver domain is stored and managed by the `Manager` class of CUPyDO. The non-matching interface mesh capability of CUPyDO is composed of two steps. The interpolation matrices are first computed by pairing and mapping interface fluid and solid nodes. For parallel computations, the mapping is achieved through successive communication rounds from the partitions of the donor side to the partitions of the target side. The parallel interface data structure for matrices and vectors is supported by the PETSc library through the functionalities that are interfaced in the *Utility* layer. In the second step, the linear system formed by the interpolation matrices is solved each time data need to be exchanged from one interface mesh to the other. The method implemented in the `Interpolator` class of CUPyDO is based on the use of Radial Basis Functions, which are well suited for black-box partitioned coupling and easy to apply in parallel. Both conservative and consistent interpolation schemes are available, using the Thin Plate Spline (TPS) or the Compact C2 (CPC2) basis functions. CUPyDO has been designed to provide a user interface that offers an adequate balance between usability and flexibility. Each coupled solver is individually configured by using their native configuration system (typically input data files). The coupling environment is then configured with a Python script that directly instantiates the main API classes by following a specific but intuitive sequence.

Verification and applications of CUPyDO

The third part of this thesis has been dedicated to the verification and application of CUPyDO. In Chapter 6, the framework has been verified and validated for both mechanically and thermally coupled applications of various complexity. In the fluid part, the Euler or Navier-Stokes equations are solved with SU2 using the finite volume method with an Arbitrary Lagrangian-Eulerian formulation. The high modularity of the coupling framework has been demonstrated by using different structural solvers and models. Lagrangian finite element solvers such as Metafor (nonlinear) and GetDP (linear) are used to solve the structural dynamic equilibrium for deformable solids and the heat equation for thermal conduction, respectively. A simpler integrator is used to compute constrained rigid body motions such as the plunging cylinder or the pitching-plunging airfoil. The main coupling capabilities of CUPyDO, such as strong/weak coupling, Aitken's under-relaxation, thermal coupling schemes and non-matching interface mesh interpolation, have been assessed through different verification cases. Several guidelines (or best practices) have been highlighted as well.

The coupling between SU2 and the rigid body integrator has been first tested. The study of the vortex-induced vibrations of the one-degree-of-freedom cylinder has clearly showed that, for Reynolds numbers falling in the lock-in region, a fully coupled model is required to accurately predict the amplitude of the motion of the structure, that is severely underestimated by a separate (uncoupled) analysis of each physics. The Isogai wing section model has been used to validate CUPyDO for aeroelastic flutter applications. The complex S-shape of the flutter boundary, due to the presence of several flutter points at highly transonic Mach numbers, has been successfully predicted. In the post-critical region, aerodynamic nonlinearities lead to limit cycle oscillations of the structure with an amplitude that depends on the speed index. The speed index has been found to have an impact on the frequency content of the response and the phase shift between the structural modes as well. Testing both strong and weak coupling schemes has revealed that, for accurately capturing the flutter boundary, a strongly coupled scheme is preferred.

The complete CFD-CSD coupling has been tested by substituting the Metafor solver to the rigid body integrator. The design of the interfacing layer of CUPyDO has made the procedure straightforward and has not required any adaptation of CUPyDO. The coupling was tested on the classical case of the cantilever attached in the wake of a square cylinder. The vortex-induced vibrations of the cantilever have been properly captured, and a good agreement has been observed for the frequency and the amplitude of the cantilever motion. Added-mass effects, and consequently numerical coupling instabilities and slow convergence rates, have been intentionally introduced by decreasing the mass ratio in order to test the performance of the Aitken's coupling relaxation implemented in CUPyDO. It has been found that Aitken's relaxation outperforms the static relaxation by stabilizing the coupling procedure and by decreasing the mean number of coupling iterations.

The coupling between SU2 and Metafor has been used again for aeroelastic flutter computations on the AGARD 445.6 wing case, a test case representative of industrial complexity (three-dimensional, non-matching interface meshes). The flutter boundary obtained in this study has been compared to a collection of results, both experimental and numerical, with good agreement in the subsonic and transonic regimes. In the supersonic regime, a large scatter is observed across the results found in the literature, which can mostly be attributed to an inaccurate fluid solution (flow model, numerical scheme, mesh...) rather than to deficiencies of the coupling environment itself. The conservative RBF interpolation has been successfully applied for transferring interface data between the two coupled solvers. The performance in terms of CPU time of a typical coupled simulation has been evaluated on a representative flight condition of the ARGARD wing. It has been shown that the largest contribution to the total CPU time stems for the computation of the fluid solution (65%) and from the fluid mesh deformation (32%). The computation of the solid solution (2%) represents a much smaller contribution while the overhead brought by the coupling task (1%) is marginal.

The thermal coupling capabilities of CUPyDO have been tested for the case of a heated hollow cylinder immersed in a steady cross flow. The coupling versatility of the tool has been further demonstrated by using the GetDP FEM code instead of Metafor, without changing anything in the coupling framework. The four available thermal coupling schemes have been tested for two distinct physical Biot numbers (smaller and larger than one). The coupling performance has been measured by the number of coupling iterations, and it has been confirmed that temperature-back schemes are the most efficient for $\mathrm{Bi} < 1$ cases while, reversely, flux-back schemes become more efficient for $\mathrm{Bi} > 1$. It has also been showed that the use of a numerical heat transfer coefficient can stabilized the coupling procedure if stability cannot be obtained with the standard schemes. Finally, consistent interpolation has been successfully applied for the transfer of thermal data in the case of a non-matching interface discretization.

The last application of CUPyDO in Chapter 7 has involved the coupling between SU2 and Metafor for the aeroelastic simulation of a very flexible cantilever flat plate wing. Different geometries with varying sweep angles have first been studied in the linear flutter regime by computing the response to small initial perturbations. Computed flutter velocities and frequencies have been compared with experimental observations and with a low-fidelity numerical model based on the vortex lattice method. The results have shown a decreasing flutter speed and frequency with respect to the sweep angle in good agreement with the reference data.

In a second step the post-critical aeroelastic response of the plate has been considered for two sweep angles: $0°$ and $45°$. For post-critical flow velocities up to 24% above the flutter airspeed, two distinct super-critical LCO branches have been identified for the unswept configuration, depending on the amplitude of the initial perturbation as measured by the initial perturbation duration. The low-amplitude/frequency branch has been shown to be dominated by a bending mode, while the high-amplitude/frequency branch has been shown to be dominated by a torsion mode. The theoretical threshold value of t^*, above which the LCO switches from the low- to the high-frequency branch, has been found to decrease with increasing free-stream airspeed. The response on the low-frequency LCO branch features a non-oscillatory component which slightly decays in time. Significant static deflection had been experimentally observed but could not be directly related to the transient non-oscillatory deformation component in the computational model, as they differ in several aspects such as amplitude, duration and conditions of occurrence. Only one type of LCO response had been observed during the experiments. This single LCO resembles more the low-frequency solution in terms of frequency and deformation pattern, but they significantly differ in their amplitude. In the computational model, the transition from the low- to the high-frequency LCO has been correlated to a change in the flow structure: the occurrence of a stable (i.e. not shed into the wake) leading edge recirculation zone that increases locally the suction.

The post-critical response of the swept plate ($\Lambda = 45°$) has been found to be significantly different from the unswept configuration. Small initial perturbations are quickly amplified in time until the dynamics reaches a limit cycle which is characterized by a much larger amplitude than in the unswept case. Moreover, the limit cycle response of the swept wing does not feature multiple branches and flow visualization has not revealed any significant change in the flow structure over the set of perturbation and free-stream airspeed considered. Static deflection has not been observed numerically for the swept plate, in agreement with experimental observations. Additionally, qualitative comparison of the deformation pattern between the computational and experimental response has shown satisfactory agreement.

Finally, several post-critical computations for both sweep angles have revealed that the robustness of the fluid mesh deformation remains limited when the structure undergoes large deformation. This highlights one of the typical disadvantages of using body-fitted ALE fluid meshes for FSI applications. Other coupling approaches should therefore be considered, as discussed in the following section.

Future perspectives

One of the major objectives of this thesis was to develop a robust tool that provides all the basic features required for the computational analysis of FSI problems in an academic or industrial context. This was achieved through the development of CUPyDO. However, the number of potential improvements and extensions one can imagine in the future of CUPyDO is large. This last section aims to provide a non-exhaustive list of possible developments and applications of CUPyDO that could be considered in the context of future research work.

In order to increase the capabilities of CUPyDO for FSI applications, the list of compatible fluid and solid solvers to be coupled should be enlarged. The interest would be to interface solvers based on other models than the FVM/FEM as presented in this thesis. Typically, on the fluid side, coupling solvers based on the immersed boundary approach would be of great interest for applications involving large rigid body motion such as store-release or flapping flight problems. FSI simulations involving free-surface flows have already been successfully performed by coupling a Lagrangian fluid solver based on the PFEM method with Metafor [2]. Fast steady aeroelastic computations of composite wings have also been achieved by coupling a fluid FEM-based code for the resolution of the full potential equations with a structural solver based on a modal decomposition [5]. In order to reduce further the computational cost, the coupling with low-fidelity flow models based on panel or lattice methods is also being tested with CUPyDO for fast parametric studies. On the solid side, the FEM solver for deformable solids has already been successfully replaced by a modal integrator computing the deformation of the solid using modal superposition [6]. This approach requires the use of a FEM solver only prior to the coupled simulation to compute once the modal modes of the structure.

The coupling environment has been designed for fluid-structure applications by exchanging surface data from one side of the interface to the other side. The coupling capabilities could however be extended towards general multiphysics applications, for example by coupling an electro-magnetic field, using for instance the GetDP solver, to a fluid flow. This typically would require the exchange of 3D volume data in addition to surface data as well as adding new dedicated solvers to the current list of compatible solvers to be coupled. Development of new coupling algorithms might also be required, which could raise again the question of numerical coupling stability.

The design capabilities of CUPyDO could be further extended towards optimization of coupled FSI problems. Optimization problems are now commonly solved for single-physics problems but coupled problems increase significantly the complexity of the optimization as sensitivities to the design variables, considering a gradient-based optimizer, must be computed for each coupled sub-system. The computation of the gradients can be achieved by several methods. A direct method would require the exact representation of the Jacobian of the governing equation, which is usually not affordable in practice. The standard finite difference method is easy to implement for black-box solvers and does not require expressing the Jacobian. However, the accuracy of the gradient calculation depends on a suitable choice of the numerical increment. Most importantly, for both direct and finite difference methods, the cost directly depends on the number of design variables [257]. Adjoint methods should thus be preferred, as the adjoint governing equations do not depend on the number of design variables. By using the same approach as for the direct coupled problem, an adjoint coupled problem could be obtained by coupling black-box fluid and solid adjoint solvers. On the fluid side, the SU2 solver can be envisaged as it already features adjoint capabilities, in which the derivatives are computed by Algorithmic Differentiation (AD) [257]. On the solid side, a new, possibly existing, structural solver featuring adjoint capabilities might be required, or the extension of Metafor and GetDP could be proposed. The coupling algorithm of CUPyDO will need to be extended as well for the coupling of adjoint solutions. The same iterative procedure such as the block Gauss-Seidel iterative coupling can be applied, but the cross dependencies (off-diagonal terms of the tangent matrix in Eq. (3.3) of the three field problem) must be computed. This may require some coupling tasks of CUPyDO (e.g. non-matching mesh interpolation) to be differentiated. For this purpose, AD can also be envisaged to be introduced in the C++ kernel of CUPyDO for the derivatives to be exposed in Python through the wrapper. Optimization for steady-state FSI has been recently achieved while unsteady FSI with adjoint is still an ongoing work.

The algorithmic capabilities of CUPyDO could also be extended or improved. In addition to the staggered iterative coupling, for which the coupled solvers compute their solution one at a time, a vectorial coupling could be envisaged. In this situation, the coupled solvers compute their solution at the same time and exchange their solutions between each time step. This strategy would however require the use of MPI even for non-parallelized solvers, so that each of them can be run on distinct cores (see segregated distribution from Section 5.6). Stability and coupling convergence rate improvements can be proposed for cases with high-added mass effect, for which the performance of BGS is still limited. In this context, the IQN-ILS coupling scheme has already been implemented but other coupling schemes could be envisaged. For thermal coupling schemes based on a numerical heat transfer coefficient, a procedure for locally computing this coefficient instead of imposing a constant and uniform value, would improve and even optimize the convergence rate of the coupled solution. The non-matching mesh capabilities can be improved as well. For instance, the dependence on the RBF radius can be lowered by implementing the variant based on the partition of unity (see end of Section 3.4.2). The CPU cost associated with the solution of the linear system could also be reduced by applying coarsening methods for selecting a representative subset of source nodes. Implementing other interpolation methods, potentially less costly, such as the explicit Inverse Distance Weighting to further increase the flexibility of CUPyDO could represent another avenue for future research. Finally, the usability of the coupling tool could be further improved by implementing a restart capability for coupled simulations. This is not a trivial task as the restart solution of each coupled solver must be re-synchronized properly to conserve the order of accuracy of the time schemes.

Finally, a more-in-dept analysis of the HPC capabilities of CUPyDO could be performed. Parallel scalability on massively distributed architectures could be assessed. However, as the scalability of the coupled system strongly depends on the intrinsic scalability of the sub-systems (i.e. the black-box solvers) the study should only focus on the coupling tasks, typically the parallel non-matching mesh mapping and interpolation, both being supported by the linear algebra kernel linked to the PETSc library. Also, and in order to improve the usability and portability of the coupling tool, replacing PETSc by another parallel linear algebra engine, could be envisaged. It was found indeed that running PETSc on Windows platforms can become cumbersome. Ideally, the new candidate should come with a Python wrapper as it was the case for PETSc. The Trilinos library [258] from Sandia National Laboratories may be a potential candidate.

List of Figures

List of Tables

Bibliography

[1] A. Bejan, J.D. Charles, and S. Lorente. The evolution of airplanes. *Journal of Applied Physics*, 116(4), 2014. https://doi.org/10.1063/1.4886855.

[2] M.L. Cerquaglia, D. Thomas, R. Boman, V. Terrapon, and J.-P. Ponthot. A fully partitioned Lagrangian framework for FSI problems characterized by free surfaces, large solid deformations and displacements, and strong added-mass effects. *Computer Methods in Applied Mechanics and Engineering*, 348:409–442, 2019. https://doi.org/10.1016/j.cma.2019.01.021doi:10.1016/j.cma.2019.01.021.

[3] M.L. Cerquaglia. *Development of a fully-partitioned PFEM-FEM approach for fluid-structure interaction problems characterized by free surfaces, large solid deformations, and strong added-mass effects.* PhD thesis, Université de Liège, Belgium, 2019.

[4] A. Crovato, H.S. Almeida, G. Vio, G.H. Silva, A.P. Prado, C. Breviglieri, H. Güner, P.H. Cabral, R. Boman, V.E. Terrapon, and G. Dimitriadis. Effect of levels of fidelity on steady aerodynamic and static aeroelastic computations. *Aerospace*, 7, 2020. https://doi.org/10.3390/aerospace7040042.

[5] A. Crovato. *Steady Transonic Aerodynamic and Aeroelastic Modeling for Preliminary Aircraft Design.* PhD thesis, Université de Liège, Belgium, 2020.

[6] H. Güner. *Unsteady aerodynamic modeling methodology based on Dynamic Mode Interpolation (DMI) for transonic flutter calculations.* PhD thesis, Université de Liège, Belgium, 2020.

[7] H. Güner, D. Thomas, G. Dimitriadis, and V.E. Terrapon. Unsteady aerodynamic modeling methodology based on dynamic mode interpolation for transonic flutter calculations. *Journal of Fluid and Structures*, 84:218–232, 2019. https://doi.org/10.1016/j.jfluidstructs.2018.11.002.

[8] G.J. Kennedy and J. Martins. A parallel finite-element framework for largescale gradient-based design optimization of high-performance structures. *Finite Elements in Analysis and Design*, 87:56–73, 2014. https://doi.org/10.1016/j.finel.2014.04.011.

[9] M.A. Abdelrhman. Modeling coupling between eelgrass zostera marina and water flow. *Marine Ecology Progress Series*, 338:81–96, 2007.

[10] I.E. Garrick and W.H. Reed III. Historical development of aircraft flutter. *Journal of Aircraft*, 18(11):897–912, 1981.

[11] Shigehiko Kaneko, Tomomichi Nakamura, Fumio Inada, Minoru Kato, Kunihiko Ishihara, Takashi Nishihara, and Mikael A. Langthjem, editors. *Flow-Induced Vibrations.* Academic Press, Oxford, Second edition, 2014. https://doi.org/10.1016/B978-0-08-098347-9.00011-4.

[12] D.G. Gorman, J.M. Reese, and Y.L. Zhang. Vibration of a flexible pipie conveying viscous pulsating fluid flow. *Journal of Sound and Vibration*, 230(2):379–392, 2000.

[13] L. Lui and F. Xuan. Flow-induced vibration analysis of supported pipes conveying pulsating fluid using precise integration method. *Mathematical Problems in Engineering*, 2010, 2010.

[14] S. Zhou, T-J. Yu, X-D. Yang, and W. Zhang. Global dynamics of pipes conveying pulsating fluid in the supercritical regime. *International Journal of Applied Mechanics*, (2), 2017.

[15] M.P. Païdoussis. *Fluid-Structure Interaction: Slender structures and axial flow*, volume 1. Elsevier Academic Press, Second edition, 2014.

[16] F. Xie, X. Zheng, M.S. Triantafyllou, Y. Constantinides, and G. Em Karniadakis. The flow dynamics of the garden-hose instability. *Journal of Fluid Mechanics*, 800:595–612, 2016.

[17] X. Li, B. Sun, H. You, and L. Wang. Evolution of Rolls-Royce air-cooled turbine blades and feature analysis. *Procedia Engineering, 2014 Asia-Pacific International Symposium on Aerospace Technology, APISAT2014*, 99:1482–1491, 2015.

[18] R.D. Blevins. *Flow-Induced Vibrations*. Nostrand Reinhold Co., New York, 1977.

[19] T. Sarpkaya. Vortex-Induced Oscillations: A selective review. *Journal of Applied Mechanics*, 46(2):241–258, 1979. `https://doi.org/10.1115/1.3424537`.

[20] C.H.K. Williamson. Vortex shedding in the cylinder wake. *Annual Review of Fluid Mechanics*, 28:477–539, 1996. `https://doi.org/10.1146/annurev.fl.28.010196.002401`.

[21] P.W. Bearman. Circular cylinder wakes and vortex-induced vibrations. *Journal of Fluids and Structures*, (5):648–658, 2011. `https://doi.org/10.1016/j.jfluidstructs.2011.03.021`.

[22] R. Bourguet and M.S. Triantafyllou. Vortex-induced vibrations of a flexible cylinder at large inclination angle. *Philosophical transactions Series A, Mathematical, physical, and engineering sciences*, 373(2033), 2015. `https://doi.org/10.1098/rsta.2014.0108`.

[23] Jiaqing Kou, Weiwei Zhang, Yilang Liu, and Xintao Li. The lowest reynolds number of vortex-induced vibrations. *Physics of Fluids*, 29(4):041701, 2017. `https://doi.org/10.1063/1.4979966`.

[24] M. Van Dyke. *An album of fluid motion*. The Parabolic Press, Stanford, California, USA, 1982.

[25] M.S. Triantafyllou, R. Bourguet, J. Dahl, and Y. Modarres-Sadeghi. Vortex-Induced Vibrations. In M.R. Dhanak and N.I Xiros, editors, *Handbook of Ocean Engineering*, chapter 36, pages 819–849. Spring International, 2004. `https://doi.org/10.1007/978-3-319-16649-0`.

[26] Jing Xu, Dongshi Wang, Hui Huang, Menglan Duan, Jijun Gu, and Chen An. A vortex-induced vibration model for the fatigue analysis of a marine drilling riser. *Ships and Offshore Structures*, 12(S1):S280–S287, 2017. `https://doi.org/10.1080/17445302.2016.1271557`.

[27] Yongle Li, Haojun Tang, Qiaoman Lin, and Xinzhong Chen. Vortex-induced vibration of suspenders in the wake of bridge tower by numerical simulation and wind tunnel test. *Journal of Wind Engineering and Industrial Aerodynamics*, 164(Sup C):164 – 173, 2017. `https://doi.org/10.1016/j.jweia.2017.02.017`.

[28] P. D'Asdia and S. Noè. Vortex-induced vibration of reinforced concrete chimneys: in sity experimentation and numerical previsions. *Journal of Wind Engineering and Industrial Aerodynamics*, 74-76:765–776, 1998. `https://doi.org/10.1016/S0167-6105(98)00069-5`.

[29] V. Holland, T. Tezdogan, and E. Oguz. Full-scale CFD investigations of helicql strakes as a means of reducing the vortex induced forces on a semi-submersible. *Ocean Engineering*, 137:338–351, 2017. `https://doi.org/10.1016/j.oceaneng.2017.04.014`.

[30] Païdoussis Michael P., Price Stuart J., and de Langre Emmanuel. *Fluid-Structure Interaction: Cross-Flow-Induced Instabilities*. Cambridge University Press, New York, First edition, 2011.

[31] J.P. Den Hartog. *Mechanical vibrations*. McGraw-hill, New York, 1956.

[32] C.B. Rawlins. Galloping conductors. In *Transmission Line Reference Book*, chapter 4, Wind-induced conductor motion. Palo Alto, CA: Electic Power Research Institute, 1979.

[33] C. Scruton. Wind effects on structures. *Proceedings Institution of Mechanical Engineers*, 185:301–317, 1971.

[34] G.V. Parkinson. Wind-induced instability of structures. *Philosophical Transactions of the Royal Society*, 269:395–409, 1971.

[35] Y.C Fung. *An introducion to the theory of elasticity*. Dover Publications, Inc, New York, 1993.

[36] E.H Dowell, R Clark, D. Cox, H.C. Curtiss, J.W. Edwards, K.C. Hall, D.A. Peters, R. Scanlan, E. Simiu, F. Sisto, and T.W. Strganac. *A modern course in aeroelasticity*, volume 116 of *Solid mechanics and its applications*. Kluwer Academic Publisher, fourth revised and enlarged edition, 2004.

[37] F. Debrabandere. *Computational methods for industrial Fluid-Structure Interactions*. PhD thesis, Université de Mons, Belgium, 2014.

[38] W. Shyy, Y. Lian, S.K. Chimakurthi, J. Tang, C.E.S Cesnik, B. Stanford, and Ifju P.G. Flexible wings and fluid-structure interactions for micro-air vehicles. In D. Floreano, JC. Zufferey, M. Srinivasan, and Ellington C., editors, *Flying Insects and Robots*, pages 143–157. Springer, Berlin, Heidelberg, 2009. `https://doi.org/10.1007/978-3-540-89393-6_11`.

[39] T.P. Combes, A.S. Malik, G. Bramesfeld, and M.W. McQuilling. Efficient fluid-structure interaction method for conceptual design of flexible, fixed-wing micro-air-vehicle wings. *AIAA Journal*, 53(6):1142–1454, 2015. `https://doi.org/10.2514/1.J053125`.

[40] F-B. Tian, H. Dai, H. Luo, J.F. Doyle, and B. Rousseau. Fluid-structure interaction involving large deformations: 3D simulations and applications to biological systems. *Journal of Computational Physics*, 258:451–469, 2014. `https://doi.org/10.1016/j.jcp.2013.10.047`.

[41] A. Abdelkefi. Aeroelastic energy harvesting: A review. *International Journal of Engineering Science*, 100:112–135, 2016. `https://doi.org/10.1016/j.ijengsci.2015.10.006`.

[42] K. Shoele and R. Mittal. Energy harvesting by flow-induced flutter in a simple model of an inverted piezoelectric flag. *Journal of Fluid Mechanics*, 790:582–606, 2016. `https://doi.org/10.1017/jfm.2016.40`.

[43] J.M. McCarthy, S. Watkins, A. Deivasigamani, and S.J. John. Fluttering energy harvesters in the wind: A review. *Journal of Sound and Vibration*, 361:355–377, 2016. `https://doi.org/`.

[44] Q. Xiao and Q. Zhu. A review on flow energy harvesters based on flapping foils. *Journal of Fluids and Structures*, 46:174–191, 2014. `https://doi.org/10.1016/j.jfluidstructs.2014.01.002`.

[45] T.B. Le and F. Sotiropoulos. Fluid-structure interaction of an aortic heart valve prosthesis driven by an animated anatomic left ventricle. *Journal of Computational Physics*, 244:41–62, 2013. `https://doi.org/10.1016/j.jcp.2012.08.036`.

[46] L. Radtke, A. Larena-Avellaneda, T. Kölbel, E.S. Debus, and A. Düsterl. Cardiovascular fluid-structure interaction: A partitioned approach utilizing the p-fem. *Proceedings in Applied Mathematics and Mechanics*, 14(1):493–494, 2014. `https://doi.org/10.1002/pamm.201410234`.

[47] S.R. Idelsohn, E. Oñate, R. Rossi, J. Marti, and F. Del Pin. New computational challenges in fluid-structure interactions problems. In J Eberhardsteiner, C Hellmich, HA Mang, and J Périaux, editors, *ECCOMAS Multidisciplinary Jubilee Symposium*, volume 14 of *Computational Methods in Applied Sciences*, pages 17–31. Springer Netherlands, Dordrecht, 2009. `https://doi.org/10.1007/978-1-4020-9231-2_2`.

[48] P.B. Ryzhakov, R. Rossi, S.R. Idelsohn, and E. Oñate. A monolithic Lagrangian approach for fluid-structure interaction problems. *Computational Mechanics*, 46:883–899, 2010. `https://doi.org/10.1007/s00466-010-0522-0`.

[49] G. Hou, J. Wang, and A. Layton. Numerical methods for fluid-structure interaction - A review. *Communications in Computational Physics*, 12(2):337–377, 2012. `https://doi.org/10.4208/cicp.291210.290411s`.

[50] J. Donéa, A. Huerta, J-Ph. Ponthot, and A. Rodríguez-Ferran. Arbitrary Lagrangian-Eulerian methods. In R. Stein, R. de Borst, and T.J.R. Hughes, editors, *Encyclopedia of Computational Mechanics*. Wiley, 2004. `https://doi.org/10.1002/0470091355`, ISBN 9780470091357.

[51] R. Boman and J-P. Ponthot. Finite element simulation of lubricated contact in rolling using the arbitrary Lagrangian–Eulerian formulation. *Computer Methods in Applied Mechanics and Engineering*, 193(39-41):4323–4353, 2004. `https://doi.org/10.1016/j.cma.2004.01.034`.

[52] J.D. Jr Anderson. *Fundamentals of aerodynamics*. McGraw-Hill, New York, fifth edition, 2011.

[53] E. Garnier, N. Adams, and P. Sagaut. *Large Eddy Simulation for Compressible Flows*. Scientific Computation. Springer Netherlands, 2009. `https://doi.org/10.1007/978-90-481-2819-8`.

[54] J. Smagorinsky. General circulation experiments with the primitive equations.I. The basic experiment. *Monthly Weather Review*, 91(3):99–164, 1963. `https://doi.org/10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2`.

[55] P.A. Durbin and B.A. Pettersson-Reif. Reynolds-Averaged Navier-Stokes equations. In *Statistical theory and modeling for turbulent flows*. Wiley, second edition, 2010. ISBN 978-0-470-68931-8.

[56] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, 1992. `https://doi.org/10.2514/6.1992-439`.

[57] F.R. Menter. Zonal two equation $k - \omega$ turbulence model for aerodynamic flows. In *AIAA Paper 1993-2906. 23rd Fluid Dynamics, Plasmadynamics, and Lasers Conference*, Orlando, Florida, USA, 1993. `https://doi.org/10.2514/6.1993-2906`.

[58] J-P. Ponthot. Unified stress update algorithms for the numerical simulation of large deformation elasto-plastic and elasto-viscoplastic processes. *International Journal of Plasticity*, 18(1):91–126, 2002. `https://doi.org/10.1016/S0749-6419(00)00097-8`.

[59] A. Bertram and R. Glüge. *Solid Mechanics: Theory, Modeling and Problems.* Springer, first german edition, 2013.

[60] A. Ibrahimbegovic. *Nonlinear Solid Mechanics: Theoretical Formulations and Finite Element Solution Methods*, volume 160 of *Solid mechanics and its applications.* Springer, 2009.

[61] C. Farhat, M. Lesoinne, and P. Le Tallec. Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: momentum and energy conservation, optimal discretization and application to aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 157(1-2):95–114, 1998. `https://doi.org/10.1016/S0045-7825(97)00216-8`.

[62] M. Lesoinne and C. Farhat. Stability analysis of dynamic meshes for transient aeroelastic computations. In *AIAA Paper 93-3325. 11th AIAA Computational Fluid Dynamics Conference*, pages 309–314, Orlando, Florida, USA, July 6-9 1993. `https://doi.org/10.2514/6.1993-3325`.

[63] U. Küttler and W. Wall. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43:61–72, 2008. `https://doi.org/10.1007/s00466-008-0255-5`.

[64] R.L. Bisplinghoff, H. Ashley, and R.L. Halfman. *Aeroelasticity.* Dover Publications, 1996.

[65] P. Causin, J.F. Gerbeau, and F. Nobile. Added-mass effect in the design of partitioned algorithms for fluid-structure problems. *Computer Methods in Applied Mechanics and Engineering*, 194:4506–4527, 2005. `https://doi.org/10.1016/j.cma.2004.12.005`.

[66] C. Förster, W.A. Wall, and E. Ramm. The artificial added-mass effect in sequential staggered fluid-structure interaction algorithm. In *European Conference on Computational Fluid Dynamics ECCOMAS CFD*, TU Delft, The Netherlands, 2006.

[67] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels. Stability of a coupling technique for partitioned solvers in FSIapplications. *Computers and Structures*, 86(23-24):2224–2234, 2008. `https://doi.org/10.1016/j.compstruc.2008.05.005`.

[68] B.M. Irons and R.C. Turck. A version of the Aitken accelerator for computer iteration. *International journal for Numerical Methods in Engineering*, 1:275–277. `https://doi.org/10.1002/nme.1620010306`.

[69] A.E.J Bogaers, S. Kok, B.D. Reddyd, and T. Franz. Quasi-Newton methods for implicit bock-box FSI coupling. *Computational Methods in Applied Mechanics and Engineering*, 279:113–132, 2014. `https://doi.org/10.1016/j.cma.2014.06.033`.

[70] J. Degroote, R. Haelterman, S. Annerel, C. Bruggeman, and J. Vierendeels. Performance of partitioned procedures in fluid–structure interaction. *Computers and Structures*, 88(7-8):446–457, 2010. `https://doi.org/10.1016/j.compstruc.2009.12.006`.

[71] J. Vierendeels, L. Lanoye, J. Degroote, and P. Verdonck. Implicit coupling of partitioned fluid–structure interaction problems with reduced order models. *Computers and Structures*, 85(11-14):970–976, 2007. `https://doi.org/10.1016/j.compstruc.2006.11.006`.

[72] A.E.J. Bogaers, S. Kok, B.D. Reddy, and T. Franz. An evaluation of quasi-newton methods for application to FSI problems involving free surface flow and solid body contact. *Computers and Structures*, 173:71–83, 2016. `https://doi.org/10.1016/j.compstruc.2016.05.018`.

[73] J. Degroote, K-J. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. *Computers and Structures*, 87(11-12):793–801, 2009. `https://doi.org/10.1016/j.compstruc.2008.11.013`.

[74] J. Degroote, S. Annerel, and J. Vierendeels. Stability analysis of Gauss–Seidel iterations in a partitioned simulation of fluid–structure interaction. *Computers and Structures*, 88(5-6):263–271, 2010. `https://doi.org/10.1016/j.compstruc.2009.09.003`.

[75] R. Haelterman, A.E.J. Bogears, K. Scheufele, B. Uekermann, and M. Mehl. Improving the performance of the partitioned QN-ILS procedure for fluid–structure interaction problems: Filtering. *Computers and Structures*, 171:9–17, 2016. `https://doi.org/10.1016/j.compstruc.2016.04.001`.

[76] F. Lindner, M. Mehl, K. Scheufele, and B. Uekermann. A comparison of various quasi-Newton schemes for partitioned fluid-structure interaction. In *VI International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems 2015*, Venice, Italy, 2015.

[77] S. Badia, A. Quaini, and A. Quarteroni. Modular vs. non-modular preconditioners for fluid–structure systems with large added-mass effect. *Computer Methods in Applied Mechanics and Engineering*, 197(49-50):4216–4232, 2008. `https://doi.org/10.1016/j.cma.2008.04.018`.

[78] L. Nettis. *Conjugate Heat Transfer: Strategies and Applications.* PhD thesis, Politecnico Di Bari, 2011.

[79] T. Verstraete, Z. Alsalihi, and R.A. Van den Braembussche. A conjugate heat tranfer method applied to turbomachinery. In *European Conference on Computational Fluid Dynamics ECCOMAS CFD 2006*, TU Delft, The Netherlands, 2006.

[80] T. Verstraete and R.A. Van den Braembussche. A novel method for the computation of conjugate heat transfer with coupled solvers. In *International Symposium on Heat Transfer in Gas Turbine Systems*, Antalya, Turkey, 2009. `https://doi.org/10.1615/ICHMT.2009.HeatTransfGasTurbSyst.570`.

[81] T. Verstraete and S. Scholl. Stability analysis of partitioned methods for predicting conjugate heat transfer. *International Journal of Heat and Mass Transfer*, 101:852–869, 2016. `https://doi.org/10.1016/j.ijheatmasstransfer.2016.05.041`.

[82] M.B. Giles. Stability analysis of numerical interface conditions in fluid–structure thermal analysis. *International Journal for Numerical Methods in Fluids*, 25(4):421–436, 1997. `https://doi.org/10.1002/(SICI)1097-0363(19970830)25:4<421::AID-FLD557>3.0.CO;2-J`.

[83] S.K. Godunov and V.S. Ryabenkii. *The Theory of Difference Schemes - An introduction.* North-Holland, Amsterdam, 1964.

[84] M-P. Errera and G. Turpin. Temporal multiscale strategies for conjugate heat transfer problems. *Journal of Coupled Systems and Multiscale Dynamics*, 1(1):89–98, 2013. `https://doi.org/10.1166/jcsmd.2013.1005`.

[85] M-P. Errera, M. Lazareff, J-D. Garaud, T. Soubrié, C. Douta, and T. Federici. A coupling approach to modeling heat transfer during a full transient flight cycle. *International Journal of Heat and Mass Transfer*, 110:587–605, 2017. `https://doi.org/10.1016/j.ijheatmasstransfer.2017.03.048`.

[86] G. Gimenez, M-P. Errera, D. Baillis, Y. Smith, and F. Pardo. A coupling numerical methodology for weakly transient conjugate heat transfer problems. *International Journal of Heat and Mass Transfer*, 97:975–989, 2016. `https://doi.org/10.1016/j.ijheatmasstransfer.2016.02.037`.

[87] M-P. Errera and S. Chemin. Optimal solutions of numerical interface conditions in fluid–structure thermal analysis. *Journal of Computational Physics*, 245:431–455, 2013. `https://doi.org/10.1016/j.jcp.2013.03.004`.

[88] M-P. Errera and F. Duchaine. Comparative study of coupling coefficients in dirichlet–robin procedure for fluid–structure aerothermal simulations. *Journal of Computational Physics*, 312:218–234, 2016. `https://doi.org/10.1016/j.jcp.2016.02.022`.

[89] R. Corral and Z. Wang. An efficient steady state coupled fluid-solid heat transfer method for turbomachinery applications. *International Journal of Thermal Sciences*, 130:59–69, 2018. `https://doi.org/10.1016/j.ijthermalsci.2018.04.003`.

[90] R.R. El Khoury, M. Errera, K. El Khoury, and M. Nemer. Efficiency of coupling schemes for the treatment of steady state fluid-structure thermal interactions. *International Journal of Thermal Sciences*, 115:225–235, 2017. `https://doi.org/10.1016/j.ijthermalsci.2017.02.001`.

[91] F. Duchaine, A. Corpron, L. Pons, V. Moureau, F. Nicoud, and T. Poinsot. Development and assessment of a coupled strategy for conjugate heat transfer with Large Eddy Simulation: Application to a cooled turbine blade. *International Journal of Heat and Fluid Flow*, 30(6):1129–1141, 2009. `https://doi.org/10.1016/j.ijheatfluidflow.2009.07.004`.

[92] J. Degroote, A. Swillens, P. Bruggeman, R. Haelterman, P. Segers, and J. Vierendeels. Simulation of fluid–structure interaction with the interface artificial compressibility method. *Numerical Methods in Biomedical Engineering*, 26(3-4):276–289, 2009. `https://doi.org/10.1002/cnm.1276`.

[93] S. Jaure, F. Duchaine, G. Staffelbach, and L.Y.M. Gicquel. Massively parallel conjugate heat transfer methods relying on large eddy simulation applied to an aeronautical combustor. *Computational Science and Discovery*, 6(1), 2013. `https://doi.org/10.1088/1749-4699/6/1/015008`.

[94] E. Radenac, J. Gressier, and P. Millan. Methodology of numerical coupling for transient conjugate heat transfer. *Computers and Fluids*, 100:95–107, 2014. `https://doi.org/10.1016/j.compfluid.2014.05.006`.

[95] B. Roe, R. Jaiman, A. Haselbacher, and P.H. Geubelle. Combined interface boundary condition method for coupled thermal simulations. *International Journal for Numerical Methods in Fluids*, 57(3):329–354, 2008. `https://doi.org/10.1002/fld.1637`.

[96] B. Baqué, M. Errera, A. Roos, and F. Feyel. Simulation of transient conjugate heat transfer via a temporal multiscale approach. *International Journal for Multiscale Computational Engineering*, 11(4):333–345, 2013. https://doi.org/10.1615/IntJMultCompEng.2013004653.

[97] C.S. Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25(3):220–252, 1977. https://doi.org/10.1016/0021-9991(77)90100-0.

[98] C.S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002. https://doi.org/10.1017/S0962492902000077.

[99] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37:239–261, 2005. https://doi.org/10.1146/annurev.fluid.37.061903.175743.

[100] Y. Kim and C.S. Peskin. Penalty immersed boundary method for an elastic boundary with mass. *Physics of Fluids*, 19, 2007. https://doi.org/10.1063/1.2734674.

[101] H. Luo, R. Mittal, X. Zheng, S.A. Bielamowicz, R.J. Walsh, and J.K. Hahn. An immersed-boundary method for flow-structure interaction in biological systems with application to phonation. *Journal of Computational Physics*, 227(22):9303–9332, 2008. https://doi.org/10.1016/j.jcp.2008.05.001.

[102] L. Wang, G.M.D. Currao, F. Han, A.J. Neely, J. Young, and F-B. Tian. An immersed boundary method for fluid–structure interaction with compressible multiphase flows. *Journal of Computational Physics*, 346:131–151, 2017. https://doi.org/10.1016/j.jcp.2017.06.008.

[103] R. Ghias, R. Mittal, and H. Dong. A sharp interface immersed boundary method for compressible viscous flows. *Journal of Computational Physics*, 225(1):528–553, 2007. https://doi.org/10.1016/j.jcp.2006.12.007.

[104] R. Mittal, H. Dong, M. Bozkurttas, F.M. Najjar, A. Vargas, and A. von Loebbecke. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of Computational Physics*, 227(10):4825–4852, 2008. https://doi.org/10.1016/j.jcp.2008.01.028.

[105] A. de Boer, A.H. van Zuijlen, and H. Bijl. Review of coupling methods for non-matching meshes. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1515–1525, 2007. https://doi.org/10.1016/j.cma.2006.03.017.

[106] A. Beckert. Coupling fluid (CFD) and structural (FE) models using finite interpolation elements. *Aerospace Science and Technology*, 4(1):13–22, 2000. https://doi.org/10.1016/S1270-9638(00)00111-5.

[107] A. de Boer, A.H. van Zuijlen, and H. Bijl. Radial basis functions for interface interpolation and mesh deformation. In Barry Koren and Kees Vuik, editors, *Advanced Computational Methods in Science and Engineering*, pages 143–178. Springer, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-03344-5_6.

[108] T. Klöppel, A. Popp, U. Küttler, and W.A. Wall. Fluid–structure interaction for non-conforming interfaces based on a dual mortar formulation. *Computer Methods in Applied Mechanics and Engineering*, 200(45-46):3111–3126, 2011. https://doi.org/10.1016/j.cma.2011.06.006.

[109] C. Bernardi, Y. Maday, and A.T. Patera. A new nonconforming approach to domain decomposition: the mortar element method. In H. Brezis and J-L. Lions, editors, *Collège de France Seminar*. Pitman, 1990.

[110] C. Lacour and Y. Maday. Two different approaches for matching nonconforming grids: the mortar element method and the FETI method. *BIT Numerical Mathematics*, 37(3):720–738, 1997. https://doi.org/10.1007%2FBF02510249.

[111] F.P.T. Baaijens. A fictitious domain/mortar element method for fluid–structure interaction. *International Journal for Numerical Methods in Fluids*, 35(7):743–761, 2001. https://doi.org/10.1002/1097-0363(20010415)35:7<743::AID-FLD109>3.0.CO;2-A.

[112] M.W. Heinstein and T.A. Laursen. A three dimensional surface-to-surface projection algorithm for non-coincident domains. *International Journal for Numerical Methods in Biomedical Engineering*, 19(6):421–432, 2003. https://doi.org/10.1002/cnm.601.

[113] R.K. Jaiman, X. Jiao, P.H. Geubelle, and E. Loth. Conservative load transfer along curved fluid–solid interface with non-matching meshes. *Journal of Computational Physics*, 218(1):372–397, 2006. https://doi.org/10.1016/j.jcp.2006.02.016.

[114] X. Jiao and M.T. Heath. Common-refinement-based data transfer between non-matching meshes in multiphysics simulations. *International Journal for Numerical Methods in Engineering*, 61(14):2402–2427, 2004. https://doi.org/10.1002/nme.1147.

[115] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, and C.R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Computer Methods in Applied Mechanics and Engineering*, 198(33-36):2632–2642, 2009. https://doi.org/10.1016/j.cma.2009.03.004.

[116] P.E. Farrell and J.R. Maddison. Conservative interpolation between volume meshes by local galerkin projection. *Computer Methods in Applied Mechanics and Engineering*, (1-4):89–100, 2011. https://doi.org/10.1016/j.cma.2010.07.015.

[117] R.K. Jaiman, X. Jiao, P.H. Geubelle, and E. Loth. Assessment of conservative load transfer for fluid–solid interface with non-matching meshes. *International Journal for Numerical Methods in Engineering*, 64(15):2014–2038, 2005. https://doi.org/10.1002/nme.1434.

[118] P. Chen and Jadoc I. Interfacing of fluid and structural models via innovative structural boundary element method. *AIAA Journal*, 36(2):282–287, 1998. https://doi.org/10.2514/2.7513.

[119] M. Sadeghi, F. Liu, K.L. Lai, and H.M. Tsai. Application of three-dimensional interfaces for data transfer in aeroelastic computations. In *22nd Applied Aerodynamics Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences*, Rhodes Island, Greece, 2004. https://doi.org/10.2514/6.2004-5376.

[120] G.S.L. Goura, K.J. Badcock, M.A. Woodgate, and B.E. Richards. A data exchange method for fluid-structure interaction problems. *The Aeronautical Journal*, 105(1046):215–221, 2001. https://doi.org/10.1017/S0001924000025458.

[121] T.C.S. Rendall and C.B. Allen. Unified fluid–structure interpolation and mesh motion using radial basis functions. *International Journal for Numerical Methods in Engineering*, 74(10):1519–1559, 2008. https://doi.org/10.1002/nme.2219.

[122] N. Maman and C. Farhat. Matching fluid and structure meshes for aeroelastic computations: a parallel approach. *Computers and Structures*, 54(4):779–785, 1995. https://doi.org/10.1016/0045-7949(94)00359-B.

[123] M.J. Smith, D.H. Hodges, and C.E.S. Cesnik. Evaluation of computational algorithms to interface between cfd and csd methodologies. Technical report, Wright-Patterson Air Force report, WL-TR-96-3055, 1995.

[124] M Buhmann. *Radial Basis Functions.* Cambridge University Press, 2003.

[125] H. Wendland. *Scattered Data Approximation.* Cambridge University Press, 2004.

[126] A. Beckert and H. Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125–134, 2001. `https://doi.org/10.1016/S1270-9638(00)01087-7`.

[127] R. Ahrem, A. Beckert, and H. Wendland. A meshless spatial coupling scheme for large-scale fluid-structure-interaction problems. *Computer Modeling in Engineering and Sciences*, 12(2):121–136, 2006. `https://doi.org/10.3970/cmes.2006.012.121`.

[128] Cordero-Gracia M., M. Gómez, and E. Valero. A radial basis function algorithm for simplified fluid-structure data transfer. *International Journal for Numerical Methods in Engineering*, 99(12):888–905, 2014. `https://doi.org/10.1002/nme.4708`.

[129] T.C.S Rendall and C.B. Allen. Improved radial basis function fluid–structure coupling via efficient localized implementation. *International Journal for Numerical Methods in Engineering*, 78(10):1188–1208, 2009. `https://doi.org/10.1002/nme.2526`.

[130] E. Luke, E. Collins, and E. Blades. A fast mesh deformation method using explicit interpolation. *Journal of Computational Physics*, 231(2):586–601, 2012. `https://doi.org/10.1016/j.jcp.2011.09.021`.

[131] J.T. Batina. Unsteady Euler airfoil solutions using unstructured dynamic meshes. *AIAA Journal*, 28(8):1381–1388, 1990. `https://doi.org/10.2514/3.25229`.

[132] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163(1-4):231–245, 1998. `https://doi.org/10.1016/S0045-7825(98)00016-4`.

[133] C. Degand and C. Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers and Structures*, 80(3-4):305–316, 2002. `https://doi.org/10.1016/S0045-7949(02)00002-0`.

[134] C.L. Bottasso, D. Detomi, and R. Serra. The ball-vertex method: a new simple spring analogy method for unstructured dynamic meshes. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4244–4264, 2005. `https://doi.org/10.1016/j.cma.2004.08.014`.

[135] R. Löhner and C. Yang. Improved ALE mesh velocities for moving bodies. *Communications in Numerical Methods in Engineering*, 12(10):599–608, 1996. `https://doi.org/10.1002/(SICI)1099-0887(199610)12:10<599::AID-CNM1>3.0.CO;2-Q`.

[136] B.T. Helenbrook. Mesh deformation using the biharmonic operator. *International Journal for Numerical Methods in Engineering*, 56(7):1007–1021, 2003. `https://doi.org/10.1002/nme.595`.

[137] X. Zhou and S. Li. A new mesh deformation method based on disk relaxation algorithm with pre-displacement and post-smoothing. *Journal of Computational Physics*, 235:199–215, 2013. `https://doi.org/10.1016/j.jcp.2012.10.024`.

[138] S. Sun, S. Lv, Y. Yuan, and M. Yuan. Mesh deformation method based on mean value coordinates interpolation. *Acta Mechanica Solida Sinica*, 29(1):1–12, 2016. `https://doi.org/10.1016/S0894-9166(16)60002-2`.

[139] K. Stein, T. Tezduyar, and R. Benney. Mesh moving techniques for fluid-structure interactions with large displacements. *Journal of Applied Mechanics*, 70(1):58–63, 2003. https://doi.org/10.1115/1.1530635.

[140] K. Stein, T.E. Tezduyar, and R. Benney. Automatic mesh update with the solid-extension mesh moving technique. *Computer Methods in Applied Mechanics and Engineering*, 193(21-22):2019–2032, 2004. https://doi.org/10.1016/j.cma.2003.12.046.

[141] R.P. Dwight. Robust mesh deformation using the linear elasticity equations. In H. Deconinck and E. Dick, editors, *Computational Fluid Dynamics 2006*. Springer, Berlin, Heidelberg, 2006. https://doi.org/10.1007/978-3-540-92779-2_62.

[142] A.L. Gaitonde and S.P. Fiddes. A three-dimensional moving mesh method for the calculation of unsteady transonic flows. *The Aeronautical Journal*, 99(984):150–160, 1995. https://doi.org/10.1017/S0001924000027135.

[143] J.A.S. Witteveen. Explicit and robust inverse distance weighting mesh deformation for cfd. In *48th AIAA Aerospace Sciences Meeting*, Orlando, Florida, 2010. https://doi.org/10.2514/6.2010-165.

[144] A. de Boer, M.S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers and Structures*, 85(11-14):784–795, 2007. https://doi.org/10.1016/j.compstruc.2007.01.013.

[145] G.A. Strofylas, G.I. Mazanakis, S.S. Sarakinos, G.N. Lygidakis, and I.K. Nikolos. Using improved radial basis functions methods for fluid-structure coupling and mesh deformation. In *ECCOMAS Congress 2016 VII European Congress on Computational Methods in Applied Sciences and Engineering*, Cretes Island, Greece, 2016. https://doi.org/10.7712/100016.1905.9110.

[146] T.C.S. Rendall and C.B. Allen. Efficient mesh motion using radial basis functions with data reduction algorithms. *Journal of Computational Physics*, 228(17):6231–6249, 2009. https://doi.org/10.1016/j.jcp.2009.05.013.

[147] T.C.S. Rendall and C.B. Allen. Parallel efficient mesh motion using radial basis functions with application to multi-bladed rotors. *International Journal for Numerical Methods in Engineering*, 81(1):89–105, 2010. https://doi.org/10.1002/nme.2678.

[148] T. Gillebaart, D.S. Blom, A.H. van Zuijlen, and H. Bijl. Adaptive radial basis function mesh deformation using data reduction. *Journal of Computational Physics*, 321:997–1025, 2016. https://doi.org/10.1016/j.jcp.2016.05.036.

[149] X. Gao, Dong Y., C. Xu, M. Xiong, Z. Wang, and X. Deng. Developing a new mesh deformation technique based on support vector machine. *International Journal of Computational Fluid Dynamics*, 31(4-5):246–257, 2017. https://doi.org/10.1080/10618562.2017.1328734.

[150] J. Niu, J. Lei, and J. He. Radial basis function mesh deformation based on dynamic control points. *Aerospace Science and Technology*, 64:122–132, 2017. https://doi.org/10.1016/j.ast.2017.01.022.

[151] L. Kedward, C.B. Allen, and T.C.S. Rendall. Efficient and exact mesh deformation using multiscale rbf interpolation. *Journal of Computational Physics*, 345:732–751–, 2017. https://doi.org/10.1016/j.jcp.2017.05.042.

[152] X. Zhou and S. Li. A novel three-dimensional mesh deformation method based on sphere relaxation. *Journal of Computational Physics*, 298:320–336, 2015. https://doi.org/10.1016/j.jcp.2015.05.046.

[153] X. Liu, N. Qin, and H. Xia. Fast dynamic grid deformation based on delaunay graph mapping. *Journal of Computational Physics*, 211(2):405–423, 2006. https://doi.org/10.1016/j.jcp.2005.05.025.

[154] E. Lefrançois. A simple mesh deformation technique for fluid–structure interaction based on a submesh approach. *International Journal for Numerical Methods in Engineering*, 75(9):1085–1101, 2008. https://doi.org/10.1002/nme.2284.

[155] Y. Wang, N. Qin, and N. Zhao. Delaunay graph and radial basis function for fast quality mesh deformation. *Journal of Computational Physics*, 294:149–172, 2015. https://doi.org/10.1016/j.jcp.2015.03.046.

[156] H. Fang, C. Gong, C. Yu, C. Min, X. Zhang, J. Liu, and L. Xiao. Efficient mesh deformation based on cartesian background mesh. *Computers and Mathematics with Applications*, 73(1):71–86, 2017. https://doi.org/10.1016/j.camwa.2016.10.023.

[157] M Lesoinne and C. Farhat. Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aeroelastic computations. *Computer Methods in Applied Mechanics and Engineering*, 134(1-2):71–90, 1996. https://doi.org/10.1016/0045-7825(96)01028-6.

[158] B. Koobus and C. Farhat. Second-order time-accurate and geometrically conservative implicit schemes for flow computations on unstructured dynamic meshes. *Computer Methods in Applied Mechanics and Engineering*, 170(1-2):103–129, 1999. https://doi.org/10.1016/S0045-7825(98)00207-2.

[159] H. Guillard and C. Farhat. On the significance of the geometric conservation law for flow computations on moving meshes. *Computer Methods in Applied Mechanics and Engineering*, 190(11-12):1467–1482, 2000. https://doi.org/10.1016/S0045-7825(00)00173-0.

[160] C. Farhat, P. Geuzaine, and C. Grandmont. The discrete geometric conservation law and the nonlinear stability of ALE schemes for the solution of flow problems on moving grids. *Journal of Computational Physics*, 174(2):669–694, 2001. https://doi.org/10.1006/jcph.2001.6932.

[161] P. Geuzaine, C. Grandmont, and C. Farhat. Design and analysis of ALE schemes with provable second-order time-accuracy for inviscid and viscous flow simulations. *Journal of Computational Physics*, 191(1):206–227, 2003. https://doi.org/10.1016/S0021-9991(03)00311-5.

[162] D. Boffi and L. Gastaldi. Stability and geometric conservation laws for ALE formulations. *Computer Methods in Applied Mechanics and Engineering*, 193(42-44):4717–4739, 2004. https://doi.org/10.1016/j.cma.2004.02.020.

[163] S. Etienne, A. Garon, and D. Pelletier. Perspective on the geometric conservation law and finite element methods for ALE simulations of incompressible flow. *Journal of Computational Physics*, 228(7):2313–2333, 2009. https://doi.org/10.1016/j.jcp.2008.11.032.

[164] R. Ma, X. Chang, L. Zhang, X. He, and M. Li. On the geometric conservation law for unsteady flow simulations on moving mesh. *Procedia Engineering*, 126:639–644, 2015. https://doi.org/10.1016/j.proeng.2015.11.253.

[165] F. Palacios, J.J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranak, A. Campos, S. Copeland, T. Economon, A. Lonkar, T. Lukaczyk, and T. Taylor. Stanford University Unstructured (SU2: An open-source integrated computational environment for multi-physics simulation and design. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Gravepine, Texas, 2013. `https://doi.org/10.2514/6.2013-287`.

[166] F. Palacios, T.D. Economon, A.C. Aranake, S.S. Copeland, A.K. Lonkar, T.W. Lukaczyk, D.E. Manosalvas, K.R. Naik, A. Santiago Padrón, B. Tracey, A. Variyar, and J.J. Alonso. Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows. In *52nd Aerospace Sciences Meeting, AIAA SciTech*, National Harbor, Maryland, 2014.

[167] T.D. Economon, F. Palacios, and J.J. Alonso. Unsteady continuous adjoint approach for aerodynamic design on dynamic meshes. *AIAA Journal*, 53(9):2437–2453, 2015. `https://doi.org/10.2514/1.J053763`.

[168] T.D. Economon, F. Palacios, S.R. Copeland, T.W. Lukaczyk, and J.J. Alonso. SU2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016. `https://doi.org/10.2514/1.J053813`.

[169] T.D. Economon, D. Mudigere, G. Bansal, A. Heinecke, F. Palacios, J. Park, M. Smelyanskiy, J.J. Alonso, and P. Dubey. Performance optimizations for scalable implicit RANS calculations with SU2. *Computers and Fluids*, 129:146–158, 2016. `https://doi.org/10.1016/j.compfluid.2016.02.003`.

[170] R. Sanchez, H.L. Kline, D. Thomas, A. Variyar, M. Righi, T.D. Economon, J.J. Alonso, R. Palacios, G. Dimitriadis, and V. Terrapon. Assessment of the fluid-structure interaction capabilities for aeronautical applications of the open-source solver SU2. In *ECCOMAS Congress, VII European Congress on Computational Methods in Applied Sciences and Engineering*, Crete Island, Greece, June 2016. `https://doi.org/10.7712/100016.1903.6597`.

[171] M. Pini, S. Vitale, P. Colonna, G. Gori, A. Guardone, T.D. Economon, J.J. Alonso, and F. Palacios. SU2: the open-source software for non-ideal compressible flows. *Journal of Phyics: Conference Series*, 821(1), 2017. `https://doi.org/10.1088/1742-6596/821/1/012013`.

[172] P.L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2), 1981. `https://doi.org/10.1016/0021-9991(81)90128-5`.

[173] B. van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32(1), 1979. `https://doi.org/10.1016/0021-9991(79)90145-1`.

[174] V. Venkatakrishnan. On the accuracy of limiters and convergence to steady state solutions. In *31st Aerospace Sciences Meeting, Aerospace Sciences Meetings*, Reno, NV, 1993. `https://doi.org/10.2514/6.1993-880`.

[175] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the euler equations by finite volume methods using runge kutta time stepping schemes. In *14th Fluid and Plasma Dynamics Conference, Fluid Dynamics and Co-located Conferences*, Palo Alto, CA, 1981. `https://doi.org/10.2514/6.1981-1259`.

[176] A. Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. In *10th Computational Fluid Dynamics Conference*, Honolulu, HI, USA, 1991. `https://doi.org/10.2514/6.1991-1596`.

[177] A. Jameson and S. Schenectady. An assessment of dual-time stepping, time spectral and artificial compressibility based numerical algorithms for unsteady flow with applications to flapping wings. In *19th AIAA Computational Fluid Dynamics*, San Antonio, Texas, 2009. `https://doi.org/10.2514/6.2009-4273`.

[178] D.J. Mavripilis. Multigrid techniques for unstructured meshes. Technical report, Inst. for Computer Applications on Science and Engineering, NASA Langley Research Center TR-95-27, Hampton, VA, 1995.

[179] D.J. Mavripilis. On convergence acceleration techniques for unstructured meshes. Technical report, Inst. for Computer Applications on Science and Engineering,NASA Langley Research Center, ICASE Rept. TR-98-44, Hampton, VA, 1998.

[180] R. Biedron and J. Thomas. Recent enhancements to the FUN3D flow solver for moving-mesh applications. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, Orlando, Florida, 2009. `https://doi.org/10.2514/6.2009-1360`.

[181] L. Adam and J-P. Ponthot. Thermomechanical modeling of metals at finite strains: First and mixed order finite elements. *International Journal of Solids and Structures*, 42:5615–5655, 2005. `https://doi.org/10.1016/j.ijsolstr.2005.03.020`.

[182] P. Bussetta, D. Marceau, and J-P. Ponthot. The adapted augmented Lagrangian method: a new method for the resolution of the mechanical frictional contact problem. *Computational Mechanics*, 49(2):259–275, 2012. `https://doi.org/10.1007/s00466-011-0644-z`.

[183] R. Boman and J-P. Ponthot. Efficient ALE mesh management for 3D quasi-Eulerian problems. *International Journal For Numerical Methods in Engineering*, 92:857–890, 2012. `https://doi.org/10.1002/nme.4361`.

[184] Y. Crutzen, R. Boman, L. Papeleux, and J-P. Ponthot. Continuous roll forming including in-line welding and post-cut within an ALE formalism. *Finite Elements in Analysis and Design*, 143:11–31, 2018. `https://doi.org/10.1016/j.finel.2018.01.005`.

[185] P. Dular, C. Geuzaine, F. Henrotte, and W. Legros. A general environment for the treatment of discrete problems and its application to the finite element method. *IEEE Transactions on Magnetics*, 34(5):3395–3398, 1998. `https://doi.org/10.1109/20.717799`.

[186] C. Geuzaine. GetDP: a general finite-element solver for the de Rham complex. In *PAMM Volume 7 Issue 1. Special Issue: Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting, Zürich*, pages 1010603–1010604, 2007. `https://doi.org/10.1002/pamm.200700750`.

[187] Q.V. Bui, L. Papeleux, and J.P. Ponthot. Numerical simulation of springback using enhanced assumed strain elements. *Journal of Materials Processing Technology*, 153-154:314–318, 2004. `https://doi.org/10.1016/j.jmatprotec.2004.04.342`.

[188] P.P. Jeunechamps and J.P. Ponthot. An efficient implicit approach for the thermomechanical behavior of materials submitted to high strain rates. *Journal de Physique IV France*, 134:515–520, 2006. `https://doi.org/10.1051/jp4:2006134079`.

[189] L. Adam and J.P. Ponthot. A coupled thermo-viscoplastic formulation at finite strains for the numerical simulation of superplastic forming. *Journal of Materials Processing Technology*, 139(1-3):514–520, 2003. `https://doi.org/10.1016/S0924-0136(03)00529-6`.

[190] B.W. Uekermann. *Partitioned Fluid-Structure Interaction on Massively Parallel Systems*. PhD thesis, Technische Universität München, Germany, 2016.

[191] S. Kataoka, S. Minami, H. Kawai, T. Yamada, and S. Yoshimura. A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid–structure interactions. *Computational Mechanics*, 53(6):1299–1310, 2014. `https://doi.org/10.1007/s00466-013-0973-1`.

[192] T. Yamada, G. Hong, S. Kataoka, and S. Yoshimura. Parallel partitioned coupling analysis system for large-scale incompressible viscous fluid–structure interaction problems. *Computers and Fluids*, 141:259–268, 2016. `https://doi.org/10.1016/j.compfluid.2016.03.030`.

[193] T. Wang, S. Sicklinger, R. Wüchner, and K.-U. Bletzinger. Concept and realization of coupling software EMPIRE in multi-physics co- simulation. In *Computational Methods in Marine Engineering*, pages 289–298, 2013.

[194] M. Sayed, Th. Lutz, E. Krämer, Sh. Shayegan, A. Ghantasala, R. Wüchner, and K.-U. Bletzinger. High fidelity cfd-csd aeroelastic analysis of slender bladed horizontal-axis wind turbine. *Journal of Physics: Conference Series*, 753, 2016. `https://doi.org/10.1088/1742-6596/753/4/042009`.

[195] Fraunhofer Institute for Algorithms and Germany Scientific Computing SCAI, Sankt Augustin. MpCCI 4.5.0-1 documentation. `https://www.mpcci.de/content/dam/scai/mpcci/documents/MpCCIdoc-4_5_0.pdf`.

[196] W. Joppich and M. Kürschner. MpCCI - A tool for the simulation of coupled applications. *Concurrency and Computation: Practice and Experience*, 18(2):183–192, 2006. `https://doi.org/10.1002/cpe.913`.

[197] F. Duchaine, S. Jauré, D. Poitou, E. Quémerais, G. Staffelbach, T. Morell, and L. Gicquell. Analysis of high performance conjugate heat transfer with the OpenPALM coupler. *Computational Science and Discovery*, 8(1), 2015. `https://doi.org/10.1088/1749-4699/8/1/015003`.

[198] A. Piacentini, T. Morel, M. Thévenin, and F. Duchaine. O-PALM: an open source dynamic parallel coupler. In *IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems*, pages 885–895, Kos Island, Greece, 2011.

[199] S. Valcke. The OASIS3 coupler: a European climate modelling community software. *Geoscientific Model Development*, 6(2):373–388, 2013. `https://doi.org/10.5194/gmd-6-373-2013`.

[200] A. Craig, S. Valcke, and L. Coquart. Development and performance of a new version of the OASIS coupler, OASIS3-MCT3.0. *Geoscientific Model Development*, 10(9):3297–3308, 2017. `https://doi.org/10.5194/gmd-10-3297-2017`.

[201] E. Sanchez-Gomez, C. Cassou, Y. Ruprich-Robert, E. Fernandez, and L. Terray. Drift dynamics in a coupled model initialized for decadal forecasts. *Climate Dynamics*, 46(5-6):1819–1840, 2016. `https://doi.org/10.1007/s00382-015-2678-y`.

[202] M. Ličer, P. Smerkol, A. Fettich, M. Ravdas, A. Papapostolou, A. Mantziafou, B. Strajnar, J. Cedilnik, M. Jeromel, J. Jerman, S. Petan, V. Malačič, and S. Sofianos. Modeling the ocean and atmosphere during an extreme bora event in northern Adriatic using one-way and two-way atmosphere–ocean coupling. *Ocean Science*, 12(1):71–86, 2016. `https://doi.org/10.5194/os-12-71-2016`.

[203] M. Mehl, B. Uekermann, H. Bijl, D. Blom, B. Gatzhamme, and A. van Zuijlen. Parallel coupling numerics for partitioned fluid–structure interaction simulations. *Computers and Mathematics with Applications*, 71(4):869–891, 2016. `https://doi.org/10.1016/j.camwa.2015.12.025`.

[204] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, K. Shukaev, and B. Uekermann. preCICE – A fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250–258, 2016. `https://doi.org/10.1016/j.compfluid.2016.04.003`.

[205] K.E Jacobson. *Adjoint-based aeroelastic optimization with high-fidelity time-accurate analysis*. PhD thesis, Georgie Institute of Technology, 2019.

[206] K.E. Jacobson, J.F. Kiviaho, M.J. Smith, and G.J. Kennedy. An aeroelastic coupling framework for time-accurate aeroelastic analysis and optimization. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Kissimmee, Florida, 2018.

[207] R.T. Biedron, J-R. Carlson, J.M. Delarga, P.A. Gnoffo, D.P. Hammond, W.T. Jones, B. Kleb, E.M. Lee-Rausch, E.J. Nielsen, M.A. Park, C.L. Rumsey, J.L. Thomas, K.B. Thompson, and A.W. William. Fun3d manual: 13.6. `https://fun3d.larc.nasa.gov/papers/FUN3D_Manual-13.6.pdf`.

[208] W.K. Anderson and D.L. Bonhaus. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids*, 23(1):1–21, 1994. `https://doi.org/10.1016/0045-7930(94)90023-X`.

[209] D.M. Beazley. SWIG : An easy to use tool for integrating scripting languages with C and C++. In *4th Tcl/Tk Workshop*, Monterey, CA, USA, July 1996. `https://www.usenix.org/legacy/publications/library/proceedings/tcl96/full_papers/beazley/index.html`.

[210] S. van der Walt, S.C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. `https://doi.org/10.1109/MCSE.2011.37`.

[211] Abaqus analysis user's guide V6.14, online documentation. SIMULIA, `http://abaqus.software.polimi.it/v6.14/books/usb/default.htm`.

[212] L. Dalcin, R. Paz, M. Storti, and J. D'Elía. MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008. `https://doi.org/10.1016/j.jpdc.2007.09.005`.

[213] L. Dalcin, R. Paz, P. Kler, and A. Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011. `https://doi.org/10.1016/j.advwatres.2011.04.013`.

[214] K. Han, Y.T. Feng, and D.R.J. Owen. Performance comparisons of tree-based and cell-based contact detection algorithm. *Engineering Computations*, 24(2):165–181, 2007. `https://doi.org/10.1108/02644400710729554`.

[215] C.E. Augarde and A.J. Deeks. The use of Timoshenko's exact solution for a cantilever beam in adaptive analysis. *Finite Elements in Analysis and Design*, 44(9-10):595–601, 2008. `https://doi.org/10.1016/j.finel.2008.01.010`.

[216] W. Dettmer and D. Perić. A computational framework for fluid–rigid body interaction: Finite element formulation and applications. *Computer Methods in Applied Mechanics and Engineering*, 195(13-16):1633–1666, 2006. `https://doi.org/10.1016/j.cma.2005.05.033`.

[217] A. Roshko. On the development of turbulent wakes from vortex streets. Technical report, National Advisory Committee for Aeronautics, NACA, 1953.

[218] P. Anagnostopoulos and P.W. Bearman. Response characteristics of a vortex-excited cylinder at low Reynolds numbers. *Journal of Fluids and Structures*, 6:39–50, 1992. `https://doi.org/10.1016/0889-9746(92)90054-7`.

[219] K. Isogai. On the transonic-dip mechanism of flutter of a sweptback wing. *AIAA Journal*, 17(7):793–795, 1979. `https://doi.org/10.2514/3.61226`.

[220] K. Isogai. Transonic-dip mechanism of flutter of a sweptback wing: Part II. *AIAA Journal*, 19(9):1240–1242, 1981. `https://doi.org/10.2514/3.7853`.

[221] T. Theodorsen. General theory of aerodynamic instability and the mechanism of flutter. *NASA Report No 496*, 1940. `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19800006788.pdf`.

[222] S. S. Davis. Naca 64 a010 (nasa ames model) oscillatory pitching. Technical report, AGARD Report No. 702, 1982.

[223] X. Amandolese, S. Michelin, and M. Choquel. Low speed flutter and limit cycle oscillations of a two-degree-of-freedom flat plate in a wind tunnel. *Journal of Fluids and Structures*, 43:244–255, 2013. `https://doi.org/10.1016/j.jfluidstructs.2013.09.002`.

[224] M. McMullen, A. Jameson, and J.J. Alonso. Application of a non-linear frequency domain solver to the Euler and Navier-Stokes equations. In *40th AIAA Aerospace Sciences Meeting and Exhibit, Aerospace Sciences Meetings*, Reno, NV, USA, 2002. `https://doi.org/10.2514/6.2002-120`.

[225] N.A. Pierce and J.J. Alonso. Efficient computation of unsteady viscous flows by an implicit preconditioned multigrid method. *AIAA Journal*, 36(3):401–408, 1998. `https://doi.org/10.2514/2.377`.

[226] F. Liu, J. Cai, Y. Zhu, H.M. Tsai, and A.S.F. Wong. Calculation of wing flutter by a coupled fluid-structure method. *Journal of Aircraft*, 38(2):334–342, 2001. `https://doi.org/10.2514/2.2766`.

[227] J.J. Alonso and A. Jameson. Fully-implicit time-marching aeroelastic solution. In *AIAA Paper 94-056. 32nd Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, 10-13 January, 1994. `https://doi.org/10.2514/6.1994-56`.

[228] Z. Biao, Q. Zhide, and G. Chao. Transonic flutter analysis of an airfoil with approximate boundary method. In *26th international congress of the aeronautical sciences*, 2008. `http://www.icas.org/ICAS_ARCHIVE/ICAS2008/PAPERS/230.PDF`.

[229] J.P. Thomas, K.C. Hall, and E.H. Dowell. Reduced-order aeroelastic modeling using proper-orthogonal decompositions. *Presented at CEAS/AIAA/ICASE/NASA Langley International Forum on Aeroelasticity and Structural Dynamics*, 1999. `http://people.duke.edu/~jthomas/papers/papers/podairfoil.pdf`.

[230] C. Gao, S. Luo, F. Liu, and D.M. Schuster. Calculation of unsteady transonic flow by an euler method with small disturbance boundary conditions. In *41st Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, 6-9 January, 2003. `https://doi.org/10.2514/6.2003-1267`.

[231] C. Habchi, S. Russeil, D. Bougeard, J-L. Harion, T. Lemenand, A. Ghanem, D. Della Valle, and H. Peerhossaini. Partitioned solver for strongly coupled fluid-structure interaction. *Computers and Fluids*, 71:306–319, 2013. `https://doi.org/10.1016/j.compfluid.2012.11.004`.

[232] A. Okajima. Strouhal numbers of rectangular cylinders. *Journal of Fluid Mechanics*, 123:379–398, 1982. `https://doi.org/10.1017/S0022112082003115`.

[233] A. Okajima. Numerical analysis of the flow around an oscillating cylinder. In *Proceedings of the 6th International Conference on Flow-Induced Vibration*, April, 10-12, 1995.

[234] S.C. Yen, K.C. San, and T.H. Chuang. Interactions of tandem square cylinders at low Reynolds numbers. *Experimental Thermal and Fluid Science*, 32(4):927–938, 2008. `https://doi.org/10.1016/j.expthermflusci.2007.07.001`.

[235] M.I. Yuce and D.A. Kareem. A numerical analysis of fluid flow around circular and square cylinders. *Journal - American Water Works Association*, 108(10):E546–E554, 2016. `https://doi.org/10.5942/jawwa.2016.108.0141`.

[236] A. Sohankar, C. Norberg, and L. Davidson. Numerical simulation of unsteady low-Reynolds number flow around rectangular cylinders at incidence. *Journal of Wind Engineering and Industrial Aerodynamics*, 69-71:189–201, 1997. `https://doi.org/10.1016/S0167-6105(97)00154-2`.

[237] M. Geradin and D. Rixen. *Mechanical Vibrations, Theory and application to structural dynamics.* Wiley and Sons, Second edition, 1997.

[238] C. Kassiotis, A. Ibrahimbegovic, R. Niekamp, and H. Matthies. Nonlinear fluid-structure interaction problem. Part I : implicit partitioned algorithm, nonlinear stability proof and validation examples. *Computational Mechanics*, 47(3):305–323, 2011. `https://doi.org/10.1007/s00466-010-0545-6`.

[239] C. Wood, A.J. Gil, O. Hassan, and J. Bonet. Partitioned block-Gauss-Seidel coupling for dynamic fluid-structure interaction. *Computers and Structures*, 88:1367–1382, 2010. `https://doi.org/10.1016/j.compstruc.2008.08.005`.

[240] M. Olivier, G. Dumas, and J. Morissette. A fluid-structure interaction solver for nano-air-vehicle flapping wings. In *AIAA Paper 2009-3676. 19th AIAA Computational Fluid Dynamics Conference*, pages 1–15, San Antonio, USA, June 2009. `https://doi.org/10.2514/6.2009-3676`.

[241] R. Sanchez, R. Palacios, T.D. Economon, H.L. Kline, J.J. Alonso, and F. Palacios. Towards a fluid-structure interaction solver for problems with large deformations within the open-source SU2 suite. In *AIAA 2016-0205. 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, January, 4-8, 2016. `https://doi.org/10.2514/6.2016-0205`.

[242] E.C. Yates. AGARD standard aeroelastic configuration for dynamic response I - Wing 445.6. *AGARD Report 765*, 1988. `http://www.dtic.mil/dtic/tr/fulltext/u2/a199433.pdf`.

[243] G.S.L. Goura. *Time marching analysis of flutter using computational fluid dynamics.* PhD thesis, University of Glasgow, 2001.

[244] R.J. Beaubien, F. Nitzsche, and D. Feszty. Time and frequency domain solutions for the AGARD 445 wing. In *International Forum on Aeroelasticity and Structural Dynamics (IFASD)*, Munich, Germany, 2005. `https://www.researchgate.net/publication/228737999_Time_and_frequency_domain_flutter_solutions_for_the_AGARD_4456_wing`.

[245] B. Zhang, W. Ding, J. Shengcheng, and J. Zhang. Transonic flutter analysis of an AGARD 445.6 wing in the frequency domain using the Euler method. *Engineering applications of computational fluid mechanics*, 10(1):244–255, 2016. `http://dx.doi.org/10.1080/19942060.2016.1152200`.

[246] R.B. Melville, S.A. Morton, and D.P. Rizzetta. Implementation of a fully-implicit, aeroelastic Navier-Stokes solver. In *13th Computational Fluid Dynamics Conference*, Snowmass Village, CO, USA, 1997. `https://doi.org/10.2514/6.1997-2039`.

[247] E.M. Lee-Rausch and J.T. Batina. Calculation of AGARD wing 445.6 flutter using Navier-Stokes aerodynamics. In *AIAA paper 93-3476. 11th Applied Aerodynamics Conference*, Monterey, CA, USA, 1993. `https://doi.org/10.2514/6.1993-3476`.

[248] J. Xiao and C. Gu. Wing flutter simulations using an aeroelastic solver based on the predictor-corrector scheme. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(11):1193–1210, 2010. `https://doi.org/10.1243/09544100JAERO756`.

[249] X. Chen, G-C. Zha, and M-T. Yang. Numerical simulation of 3-D wing flutter with fully coupled fluid-structure interaction. *Computers and Fluids*, 36(5):856–867, 2007. `https://doi.org/10.1016/j.compfluid.2006.08.005`.

[250] R.K. Finn. Determination of the drag on a cylinder at low Reynolds numbers. *Journal of Applied Physics*, 24(6):771–773, 1953. `https://doi.org/10.1063/1.1721373`.

[251] S.F. Hoerner. *Fluid-dynamic drag: practical information on aerodynamic drag and hydrodynamic resistance.* Hoerner fluid dynamics, First edition, 1965.

[252] R.L. Panton. *Incompressible flow.* Wiley, Fourth edition, 2013.

[253] G. Dimitriadis, N.F. Giannelis, and G.A. Vio. A modal frequency-domain generalised force matrix for the unsteady Vortex Lattice method. *Journal of Fluids and Structures*, 76:216–228, 2018. `https://doi.org/10.1016/j.jfluidstructs.2017.10.010`.

[254] G. Dimitriadis. *Introduction to Nonlinear Aeroelasticity.* Wiley, First edition, 2017. `https://doi.org/10.1002/9781118756478`.

[255] B.H.K. Lee, S.J. Price, and Y.S. Wong. Nonlinear aeroelastic analysis of airfoils: bifurcation and chaos. *Progress in Aerospace Sciences*, 35:205–334, 1999. `https://doi.org/10.1016/S0376-0421(98)00015-3`.

[256] K. Taira and T. Colonius. Three-dimensional flows around low-aspect-ratio flat-plate wings at low reynolds numbers. *Journal of Fluid Mechanics*, 623:187–207, 2004. `https://doi.org/10.1017/S0022112008005314`.

[257] R. Sanchez, T. Albring, R. Palacios, N.R. Gauger, T.D. Economon, and J.J. Alonso. Coupled adjoint-based sensitivities in large-displacement fluid-structure interaction using algorithmic differentiation. *International Journal for Numerical Methods in Engineering*, 113:1081–1107, 2018. `https://doi.org/10.1002/nme.5700`.

[258] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, R.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, and K.S. Stanley. An overview of the Trilinos project. *ACM Transaction on Mathematical Software*, 31(3):397–423, 2005. `https://doi.org/10.1145/1089014.1089021`.

Appendix A

RANS turbulence models

This appendix describes the Spalart-Allmaras and the Shear Stress Transport $k - \omega$ RANS models as implemented in the SU2 solver.

Spalart-Allmaras

The standard one-equation SA model computes the turbulent viscosity μ_t as:

$$\mu_\mathrm{t} = \rho \tilde{\nu} f_{v_1} \,, \tag{2}$$

with the following definitions:

$$f_{v_1} = \frac{\chi^3}{\chi^3 + c_{v_1}^3} \,, \qquad \chi = \frac{\tilde{\nu}}{\nu} \,. \tag{3}$$

The quantity $\nu = \mu/\rho$ is defined as the molecular kinematic viscosity. The new variable $\tilde{\nu}$ is obtained by solving a transport equation where the convective, viscous, and source terms are given by

$$\boldsymbol{F}^\mathrm{c} = \boldsymbol{v}\tilde{\nu} \,, \qquad \boldsymbol{F}^\mathrm{v} = -\frac{\nu + \tilde{\nu}}{\sigma} \nabla \tilde{\nu} \,, \qquad Q = c_{b_1} \hat{S} \tilde{\nu} - c_{w_1} f_w \left(\frac{\tilde{\nu}}{d_S} \right)^2 + \frac{c_{b2}}{\sigma} |\nabla \tilde{\nu}|^2 \,. \tag{4}$$

The production term is defined as

$$\hat{S} = |\boldsymbol{\omega}| + \frac{\tilde{\nu}}{\kappa^2 d_S^2} f_{v_2} \,, \tag{5}$$

where $\boldsymbol{\omega} = \nabla \wedge \boldsymbol{v}$ is the fluid vorticity, d_S is the distance to the nearest wall and

$$f_{v_2} = 1 - \frac{\chi}{1 + \chi f_{v_1}} . \tag{6}$$

The function f_w is computed as

$$f_w = g \left[\frac{1 + c_{w_3}^6}{g^6 + c_{w_3}^6} \right] \,, \tag{7}$$

where $g = r + c_{w_2}(r^6 - r)$ and

$$r = \frac{\tilde{\nu}}{\hat{S} \kappa^2 d_S^2} \,. \tag{8}$$

Finally, the set of closure constants for the model is given by

$$\sigma = 2/3 \,, \quad c_{b_1} = 0.1355 \,, \quad c_{b_2} = 0.622 \,, \quad \kappa = 0.41 \,, \tag{9}$$

$$c_{w_1} = \frac{c_{b_1}}{\kappa^2} + \frac{1 + c_{b_2}}{\sigma} \,, \quad c_{w_2} = 0.3 \,, \quad c_{w_3} = 2 \,, \quad c_{v_1} = 7.1 \,. \tag{10}$$

Generally, at the far-field boundary, the turbulent viscosity is imposed as some fraction of the laminar viscosity, while $\tilde{\nu}$ is set to zero at viscous walls.

Shear Stress Transport $k-\omega$

The Menter SST turbulence model is a two-equation model that expresses the turbulent viscosity as a function of the turbulent kinetic energy k and its specific dissipation rate ω. More specifically, it consists of the blending of the traditional $k-\omega$ and $k-\varepsilon$ models. The eddy viscosity, which includes the shear stress limiter, is given by

$$\mu_\mathrm{t} = \frac{\rho a_1 k}{\max(a_1\omega; SF_2)}\,, \tag{11}$$

where $S=\sqrt{2S_{ij}S_{ij}}$, $\mathbf{S}$ being the rate-of-strain tensor, and F_2 is the second blending function which is defined as

$$arg_2 = \max\left(2\frac{\sqrt{k}}{0.09\omega y}; \frac{500\nu}{y^2\omega}\right)\,, \tag{12}$$

$$F_2 = \tanh(arg_2^2)\,. \tag{13}$$

In those expressions, the variable y is the shortest distance to the nearest solid surface. The convective, viscous, and source terms for the turbulent kinetic energy are

$$\boldsymbol{F}^c = \rho k\boldsymbol{v}\,, \qquad \boldsymbol{F}^v = -(\mu+\sigma_k\mu_\mathrm{t})\nabla k\,, \qquad Q = P - \beta^*\rho\omega k\,, \tag{14}$$

where P is the production of turbulent kinetic energy. The convective, viscous, and source terms for the specific dissipation are given by

$$\boldsymbol{F}^c = \rho\omega\boldsymbol{v}\,, \qquad \boldsymbol{F}^v = -(\mu+\sigma_k\mu_\mathrm{t})\nabla\omega\,, \tag{15}$$

$$Q = \frac{\gamma}{\nu_\mathrm{t}}P - \beta^*\rho\omega^2 + 2(1-F_1)\frac{\rho\sigma_{w_2}}{\omega}\nabla k\nabla\omega\,, \tag{16}$$

where F_1 is the first blending function which is defined by

$$arg_1 = \min\left(\max\left(\frac{\sqrt{k}}{0.09\omega y}; \frac{500\nu}{y^2\omega}\right); \frac{4\rho\sigma_{\omega_2}k}{CD_{k\omega}y^2}\right)\,, \tag{17}$$

$$F_1 = \tanh(arg_1^4)\,. \tag{18}$$

Some closure constants of the model are directly given by

$$a_1 = 0.31\,, \quad \beta^* = 0.09\,, \quad \sigma_{\omega_2} = 0.856\,, \tag{19}$$

whereas the remaining set of constants are blended functions according to the value of F_1:

$$\begin{aligned}\sigma_k &= 0.85\ldots 1\,,\\ \sigma_\omega &= 0.5\ldots 0.856\,,\\ \gamma &= 0.55\ldots 0.44\,,\\ \beta &= 0.075\ldots 0.0828\,.\end{aligned} \tag{20}$$

Appendix B

Lagrange's equation

The dynamics of a system discretized in a finite set of degrees of freedom $\boldsymbol{q} = [q_1, \ldots, q_n]$ can be represented by Lagrange's equation

$$\frac{d}{dt}\left(\frac{\partial \mathcal{T}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{T}}{\partial q_i} + \frac{\partial \mathcal{V}}{\partial q_i} + \frac{\partial \mathcal{D}}{\partial \dot{q}_i} = Q_i(t) \qquad i = 1, \ldots, n\,, \tag{21}$$

where $\mathcal{T}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t)$ is the total kinetic energy of the system, $\mathcal{V}(\boldsymbol{q}, t)$ is the potential energy, $\mathcal{D}(\dot{\boldsymbol{q}}, t)$ is the dissipation function and $\boldsymbol{Q}$ is the generalized external forces.

Example of the 1D oscillator

The use of Lagrange's equation is illustrated by a simple one-dimensional oscillator, as depicted in Fig. 22. The body oscillator of mass m is allowed to move vertically. The motion is described

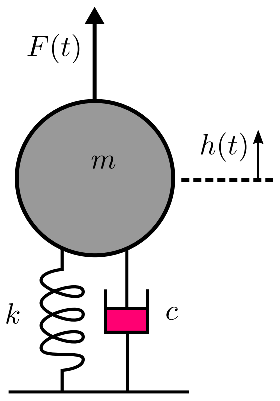


Figure 22: One-dimensional oscillator of mass m, with stiffness k and damping c.

by the variable $h(t)$. The dynamics of the motion is constrained by a linear stiffness k and a linear damping c. A vertical force $F(t)$ is applied to the oscillator. In this case the kinetic energy, potential energy and dissipation function are given by:

$$\begin{aligned} \mathcal{T} &= \frac{1}{2} m \dot{h}^2 \\ \mathcal{V} &= \frac{1}{2} k h^2 \\ \mathcal{D} &= \frac{1}{2} c \dot{h}^2\,. \end{aligned} \tag{22}$$

The application of Eq. (21) leads to the second order ordinary differential equation of motion

$$m\ddot{h} + c\dot{h} + kh = F(t) \tag{23}$$

which corresponds to the standard oscillator equation.

Appendix C

Simplified approach for the characterization of the added-mass effect

The concept of added-mass effect is introduced using a simplified fluid-structure model. A rigid body of mass m attached to a spring of stiffness k is immersed in a still, incompressible, inviscid fluid without body force, as illustrated in Fig. 23, where the fluid domain is supposed to extend to infinity. Considering only a small structural motion in the fluid, the nonlinear advection term of the Euler equations can be neglected. This leads to the simplified system of fluid equations (conservation of mass and momentum):

$$\begin{aligned} &\nabla \cdot \boldsymbol{v} = 0, \\ &\rho_{\mathrm{f}} \frac{\partial \boldsymbol{v}}{\partial t} = -\nabla p. \end{aligned} \tag{24}$$

It is assumed that the solid rigid displacement $\boldsymbol{\eta}$ is under the form $\boldsymbol{\eta} = q(t)\boldsymbol{\phi}$, where q is an unknown function of time only and $\boldsymbol{\phi} = \boldsymbol{e}_y$ a well defined modal shape ensuring a vertical rigid displacement. A simple application of Lagrange's equation leads to a standard oscillator equation for q,

$$m\ddot{q} + kq = L\,, \tag{25}$$

where L is the resultant force applied by the fluid projected along the solid motion. The

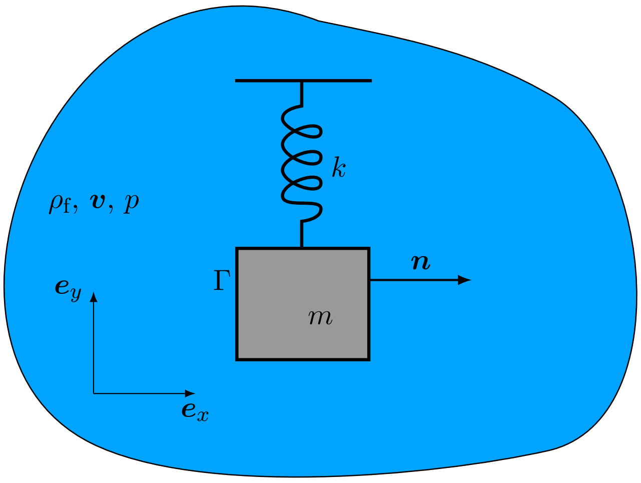


Figure 23: Simplified fluid-solid coupled problem.

kinematic and dynamic coupling conditions (2.27) on the fluid-solid interface Γ are here given by

$$\begin{aligned} \boldsymbol{v} \cdot \boldsymbol{n} &= \dot{q}\, \boldsymbol{e}_y \cdot \boldsymbol{n}\,, \\ L &= \int_\Gamma -p\boldsymbol{n} \cdot \boldsymbol{e}_y \, d\Gamma\,, \end{aligned} \tag{26}$$

where the dynamic conditions expresses the resultant force acting on the solid as a contribution of the fluid pressure distribution projected towards the vertical motion. In these expressions, $\boldsymbol{n}$ is the unit normal on Γ pointing outwards from the solid. Taking the time derivative of the kinematic coupling condition and using the momentum equation of the fluid dynamic leads to

$$\nabla p \cdot \boldsymbol{n} = -\rho_\mathrm{f}\, \ddot{q}\, \boldsymbol{e}_y \cdot \boldsymbol{n}\,. \tag{27}$$

This equation suggests expressing the pressure field as

$$p = \ddot{q}\, \phi_p\,, \tag{28}$$

where ϕ_p is the modal shape of the fluid pressure field. Introducing this expression into the dynamic coupling conditions allows us to rewrite the oscillator equation under the form:

$$(m + m_a)\, \ddot{q} + kq = 0\,, \tag{29}$$

where $m_a = \int_\Gamma \phi_p\, \boldsymbol{n} \cdot \boldsymbol{e}_y \, d\Gamma$ is the added mass related to the fluid pressure field. Equation (29) describes the motion of the solid in the fluid as its free response in vacuum (i.e. without a surrounding medium) with a modified inertia term accounting for fluid-solid interaction, as illustrated in Fig. 24. In this simple case, a direct consequence of the presence of the fluid on the free response is a modification of the natural frequency of the oscillator.

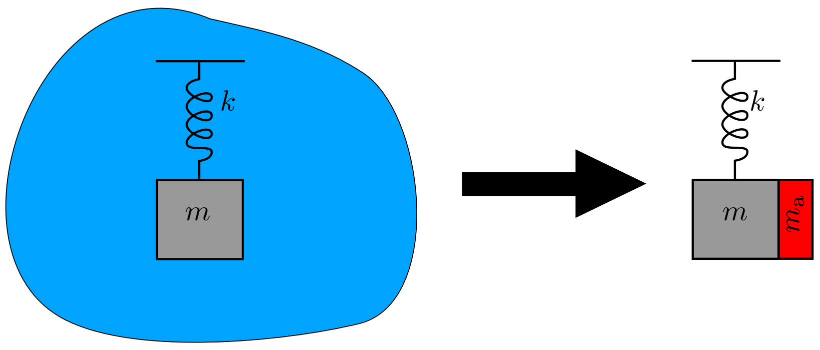


Figure 24: Illustration of added-mass effect for the simplified fluid-solid coupled problem.

Appendix D

Temporal integration schemes

This appendix describes two common temporal integration schemes: the second-order implicit generalized-α method from the Newmark finally and the fourth-order explicit Runge-Kutta.

Newmark family schemes

Algorithms of the Newmark family are popular methods for simulating systems whose dynamics is represented by a second order nonlinear differential equation:

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)\,, \tag{30}$$

with the initial conditions $\boldsymbol{q}(t_0) = \boldsymbol{q}_0$ and $\dot{\boldsymbol{q}}(t_0) = \dot{\boldsymbol{q}}_0$. The left-hand side has the form of an inertia term and the right-hand side is a general nonlinear term. At each time step $n+1$, the standard Newmark method computes the dynamic response $\boldsymbol{q}_{n+1}$, $\dot{\boldsymbol{q}}_{n+1}$, $\ddot{\boldsymbol{q}}_{n+1}$ by solving the equations

$$\begin{aligned}
\mathbf{M}(\boldsymbol{q}_{n+1})\ddot{\boldsymbol{q}}_{n+1} &= \boldsymbol{f}(\boldsymbol{q}_{n+1}, \dot{\boldsymbol{q}}_{n+1}, t_{n+1})\,, && (31)\\
\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + \Delta t\dot{\boldsymbol{q}}_n + \Delta t^2(0.5-\beta)\ddot{\boldsymbol{q}}_n + \Delta t^2\beta\ddot{\boldsymbol{q}}_{n+1}\,, && (32)\\
\dot{\boldsymbol{q}}_{n+1} &= \dot{\boldsymbol{q}}_n + \Delta t(1-\gamma)\ddot{\boldsymbol{q}}_n + \Delta t\gamma\ddot{\boldsymbol{q}}_{n+1}\,, && (33)
\end{aligned}$$

where β and γ are two user-defined parameters that control accuracy and stability. In practice, these two parameters are defined based on a third parameter $\alpha > 0$ which is a measure of the numerical damping of the algorithm:

$$\begin{aligned}
&\gamma = 0.5 + \alpha\,,\\
&\beta = 0.25(\gamma + 0.5)^2\,.
\end{aligned} \tag{34}$$

In order to solve the system of equations (31)-(33) with a Newton-Raphson procedure, the nonlinear equation (31) is written in a residual form:

$$\boldsymbol{r}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t) = \mathbf{M}\ddot{\boldsymbol{q}} - \boldsymbol{f}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) = \mathbf{0}\,, \tag{35}$$

that can then be linearized around an approximate solution $\boldsymbol{q}^*$, $\dot{\boldsymbol{q}}^*$, $\ddot{\boldsymbol{q}}^*$:

$$\boldsymbol{r}(\boldsymbol{q}^* + \Delta\boldsymbol{q}, \dot{\boldsymbol{q}}^* + \Delta\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}^* + \Delta\ddot{\boldsymbol{q}}, t) = \boldsymbol{r}(\boldsymbol{q}^*, \dot{\boldsymbol{q}}^*, \ddot{\boldsymbol{q}}^*, t) + \mathbf{M}\Delta\ddot{\boldsymbol{q}} + \mathbf{C}_\mathrm{t}\Delta\dot{\boldsymbol{q}} + \mathbf{K}_\mathrm{t}\Delta\boldsymbol{q}\,, \tag{36}$$

with the tangent damping and stiffness matrices

$$\mathbf{C}_\mathrm{t} = -\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{q}}} \qquad \text{and} \qquad \mathbf{K}_\mathrm{t} = -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\,. \tag{37}$$

The solution at time step $n+1$ is computed based on a predictor-corrector approach and a Newton-Raphson procedure that solves the linearized residual equation. The prediction is first computed using Eqs. (32) and (33) in which we set $\ddot{\boldsymbol{q}}_{n+1} = \mathbf{0}$:

$$\begin{aligned}
\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + \Delta t\dot{\boldsymbol{q}}_n + \Delta t^2(0.5-\beta)\ddot{\boldsymbol{q}}_n\,, && (38)\\
\dot{\boldsymbol{q}}_{n+1} &= \dot{\boldsymbol{q}}_n + \Delta t(1-\gamma)\ddot{\boldsymbol{q}}_n\,, && (39)
\end{aligned}$$

then the corrector is computed based on the linearized equation and two increments from Eqs. (32) and (33):

$$
\begin{aligned}
\Delta \boldsymbol{q}_{n+1} &= -\mathbf{S}_{\mathrm{t}}^{-1} \boldsymbol{r}(\boldsymbol{q}_{n+1}, \dot{\boldsymbol{q}}_{n+1}, \ddot{\boldsymbol{q}}_{n+1})\,, && (40)\\
\Delta \dot{\boldsymbol{q}}_{n+1} &= \frac{\gamma}{\Delta t \beta} \Delta \boldsymbol{q}_{n+1}\,, && (41)\\
\Delta \ddot{\boldsymbol{q}}_{n+1} &= \frac{1}{\Delta t^2 \beta} \Delta \boldsymbol{q}_{n+1}\,. && (42)
\end{aligned}
$$

In this expressions, $\mathbf{S}_{\mathrm{t}}$ is the tangent operator

$$
\mathbf{S}_{\mathrm{t}} = \frac{1}{\Delta t^2 \beta} \mathbf{M} + \frac{\gamma}{\Delta t \beta} \mathbf{C}_{\mathrm{t}} + \mathbf{K}_{\mathrm{t}}\,. \qquad (43)
$$

The complete algorithm is depicted in Fig. 25. The Newmark algorithm is second order accurate if $\alpha = 0$ but does not provide numerical damping. When $\alpha > 0$, the algorithm has numerical damping but is only first-order accurate in time. The generalized-α method is an extension

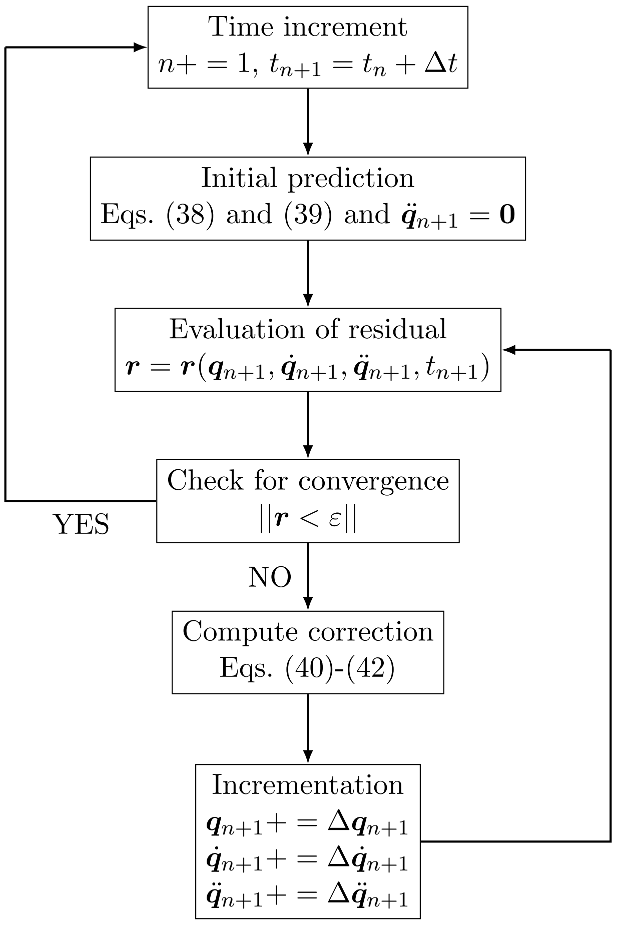

Time increment
$n+ = 1$, $t_{n+1} = t_n + \Delta t$
Initial prediction
Eqs. (38) and (39) and $\ddot{\boldsymbol{q}}_{n+1} = \mathbf{0}$
Evaluation of residual
$\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{q}_{n+1}, \dot{\boldsymbol{q}}_{n+1}, \ddot{\boldsymbol{q}}_{n+1}, t_{n+1})$
Check for convergence
$||\boldsymbol{r} < \varepsilon||$
YES
NO
Compute correction
Eqs. (40)-(42)
Incrementation
$\boldsymbol{q}_{n+1} + = \Delta \boldsymbol{q}_{n+1}$
$\dot{\boldsymbol{q}}_{n+1} + = \Delta \dot{\boldsymbol{q}}_{n+1}$
$\ddot{\boldsymbol{q}}_{n+1} + = \Delta \ddot{\boldsymbol{q}}_{n+1}$

Figure 25: Newmark algorithm.

of the standard Newmark that combines second-order accuracy and numerical damping. The procedure is the same as in the standard case, but with the equations

$$
\begin{aligned}
\mathbf{M}(\boldsymbol{q}_{n+1}) \ddot{\boldsymbol{q}}_{n+1} &= \boldsymbol{f}(\boldsymbol{q}_{n+1}, \dot{\boldsymbol{q}}_{n+1}, t_{n+1})\,, && (44)\\
\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + \Delta t \dot{\boldsymbol{q}}_n + \Delta t^2 (0.5 - \beta) \boldsymbol{a}_n + \Delta t^2 \beta \boldsymbol{a}_{n+1}\,, && (45)\\
\dot{\boldsymbol{q}}_{n+1} &= \dot{\boldsymbol{q}}_n + \Delta t (1 - \gamma) \boldsymbol{a}_n + \Delta t \gamma \boldsymbol{a}_{n+1}\,, && (46)\\
(1 - \alpha_m) \boldsymbol{a}_{n+1} + \alpha_m \boldsymbol{a}_n &= (1 - \alpha_f) \ddot{\boldsymbol{q}}_{n+1} + \alpha_f \ddot{\boldsymbol{q}}_n\,, && (47)
\end{aligned}
$$

to solve, where a new acceleration-like vector $\boldsymbol{a}$ has been introduced. The generalized-α method also introduces two algorithmic parameters α_m and α_f that are also used in the definition of γ and β

$$\begin{aligned} \gamma &= 0.5 + \alpha_f - \alpha_m \,, \\ \beta &= 0.25(\gamma + 0.5)^2 \,. \end{aligned} \tag{48}$$

Some typical values of α_m and α_f are associated with certain classical methods in structural dynamics:

- the standard Newmark procedure is recovered for $\alpha_f = \alpha_m = 0$,
- the Hilber-Hughes-Taylor (HHT) method is obtained for $\alpha_m = 0$ and $\alpha_f \in [0; 1/3]$,
- the Chung-Hulbert method is obtained when defining α_m and α_f from a unique value of the spectral radius at infinite frequencies $\rho_\infty \in [0; 1]$ (1 for no dissipation and 0 for a complete damping of high-frequency content) such that

$$\alpha_m = \frac{2\rho_\infty - 1}{\rho_\infty + 1} \qquad \text{and} \qquad \alpha_f = \frac{\rho_\infty}{\rho_\infty + 1} \,. \tag{49}$$

4th-order Runge-Kutta method

The 4th-order Runge-Kutta (RK4) method is an explicit time integration method that was developed in order to solve a first order system of the form:

$$\dot{\boldsymbol{g}} = \boldsymbol{f}(\boldsymbol{g}, t) \,, \tag{50}$$

with the initial condition $\boldsymbol{g}(t_0) = \boldsymbol{g}_0$. At each time step $n + 1$, the RK4 method computes the state of the system as

$$\boldsymbol{g}_{n+1} = \boldsymbol{g}_n + \frac{\Delta t}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) \,, \tag{51}$$

where the k_i vectors are given by

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{g}_n, t_n) \,, \\ \boldsymbol{k}_2 &= \boldsymbol{f}(\boldsymbol{g}_n + \frac{\Delta t}{2}\boldsymbol{k}_1, t_n + \frac{\Delta t}{2}) \,, \\ \boldsymbol{k}_3 &= \boldsymbol{f}(\boldsymbol{g}_n + \frac{\Delta t}{2}\boldsymbol{k}_2, t_n + \frac{\Delta t}{2}) \,, \\ \boldsymbol{k}_4 &= \boldsymbol{f}(\boldsymbol{g}_n + \Delta t \boldsymbol{k}_3, t_n + \Delta t) \,. \end{aligned} \tag{52}$$

The central idea of the method is to explicitly compute the next state of the system $\boldsymbol{g}_{n+1}$ as the sum of the current state $\boldsymbol{g}_n$ and the product between the time step and an estimate of the slope. This slope is estimated by a weighted sum of different slopes along the interval $[t_n, t_{n+1}]$: $\boldsymbol{k}_1$ the slope at the beginning of the interval, $\boldsymbol{k}_2$ and $\boldsymbol{k}_3$ are slopes at the middle of the interval, and $\boldsymbol{k}_4$ is the slope at the end of the interval.

The RK4 method can be used to solve a second-order differential equation

$$\ddot{\boldsymbol{g}} = \boldsymbol{f}(\boldsymbol{g}, \dot{\boldsymbol{g}}, t) \,, \tag{53}$$

with the initial conditions $\boldsymbol{g}(t_0) = \boldsymbol{g}_0$ and $\dot{\boldsymbol{g}}(t_0) = \dot{\boldsymbol{g}}_0$, by expressing the problem as a system of two first-order equations

$$\begin{aligned} \dot{\boldsymbol{g}} &= \boldsymbol{h}(\boldsymbol{g}, \boldsymbol{d}, t) = \boldsymbol{d} \,, \\ \dot{\boldsymbol{d}} &= \boldsymbol{s}(\boldsymbol{g}, \boldsymbol{d}, t) = \boldsymbol{f}(\boldsymbol{g}, \boldsymbol{d}, t) \,, \end{aligned} \tag{54}$$

or, in the form equivalent to Eq. (50),

$$\dot{\boldsymbol{G}} = \boldsymbol{F}(\boldsymbol{G}, t) \,, \tag{55}$$

with $\boldsymbol{G} = [\boldsymbol{g}, \boldsymbol{d}]^{\mathrm{T}}$ and $\boldsymbol{F} = [\boldsymbol{h}, \boldsymbol{f}]^{\mathrm{T}}$.

Appendix E

The following code illustrates how a coupling algorithm can be developed at a high-level by using the object-oriented structured of CUPyDO. This code is schematic and the exact syntax may slightly differ from what can be actually found in the source code. Note that `self` represent the `Algorithm` class itself.

```python
def BGSLoop(self):

    self.FSIIter = 0
    self.FSIConv = False
    self.errValue = 1e12

    while ((self.FSIIter < nbFSIIter) and (not ...
   self.criterion.isVerified(self.errValue))):
        #Communicate/interpolate solid displacement onto the ...
   fluid mesh
        self.solidToFluidMechaTransfer()

        #Deform volume fluid mesh and compute grid velocity
        self.FluidSolver.meshUpdate()

        #Call fluid solver for one time step
        self.FluidSolver.run(self.time-self.ΔT, self.time)

        #Communicate/interpolate fluid loads onto the solid ...
   mesh
        self.fluidToSolidMechaTransfer()

        #Call solid solver for one time step
        self.SolidSolver.run(self.time-self.ΔT, self.time)

        #Compute the coupling residual
        res = self.computeSolidInterfaceResidual()
        self.errValue = self.criterion.update(res)

        #Assess coupling convergence
        self.FSIConv = self.criterion.isVerified(self.errValue)

        #Perform solid interface relaxation
        self.relaxSolidPosition()

        #Write coupling monitoring data
        self.writeRealTimeData()

        self.FSIIter += 1

```

Appendix F

Results of the verification test for the RBF interpolation in CUPyDO

The following tables summarize the errors obtained during the verification tests of the RBF interpolation implemented in CUPyDO. The details related to these tests can be found in Section 5.7.2.

Pure translation

Source discretization	Target discretization	TPS	CPC2 ($r = 0.5L$)
12×3	12×3	$1.68e^{-12}$	$2.3e^{-11}$
12×3	25×3	$1.18e^{-12}$	$2.3e^{-11}$
12×3	50×5	$1.04e^{-12}$	$2.4e^{-11}$
12×3	100×10	$1.15e^{-12}$	$2.4e^{-11}$
100×10	12×3	$1.54e^{-10}$	$8.5e^{-11}$
100×10	25×3	$1.16e^{-10}$	$6.9e^{-11}$
100×10	50×5	$1.05e^{-10}$	$9.4e^{-11}$
100×10	100×10	$9.29e^{-11}$	$9.9e^{-11}$

Table 10: Variable mesh discretization, TPS and CPC2 basis functions.

Source discretization	Target discretization	Radius	Error
12×3	100×10	$0.25L$	0
		$0.5L$	$2.40e^{-11}$
		$0.75L$	$2.07e^{-11}$
		L	$1.2e^{-11}$
100×10	12×3	$0.25L$	$2.5e^{-10}$
		$0.5L$	$8.5e^{-11}$
		$0.75L$	$1.2e^{-10}$
		L	$1.5e^{-10}$

Table 11: CPC2 basis function with variable radius.

Pure rotation

Source discretization	Target discretization	TPS	CPC2 ($r = 0.5L$)
12×3	12×3	$4.7e^{-12}$	$1.2e^{-13}$
12×3	25×3	$4.4e^{-12}$	$6.2e^{-13}$
12×3	50×5	$4.5e^{-12}$	$4.6e^{-13}$
12×3	100×10	$4.4e^{-12}$	$5.4e^{-13}$
100×10	12×3	$1.3e^{-10}$	$2.6e^{-11}$
100×10	25×3	$1.0e^{-10}$	$2.2e^{-11}$
100×10	50×5	$9.1e^{-11}$	$3.5e^{-11}$
100×10	100×10	$9.7e^{-11}$	$9.9e^{-11}$

Table 12: Variable mesh discretization, TPS and CPC2 basis functions.

Source discretization	Target discretization	Radius	Error
12×3	100×10	$0.25L$	$2.2e^{-12}$
		$0.5L$	$5.40e^{-13}$
		$0.75L$	$5.3e^{-14}$
		L	0
100×10	12×3	$0.25L$	$4.9e^{-7}$
		$0.5L$	$2.6e^{-11}$
		$0.75L$	$4.9e^{-11}$
		L	$6.6e^{-11}$

Table 13: CPC2 basis function with variable radius.

Bending

Source discretization	Target discretization	TPS	CPC2 ($r = 0.5L$)
12×3	12×3	$2.7e^{-12}$	$4.6e^{-13}$
12×3	25×3	$1.2e^{-3}$	$2.8e^{-3}$
12×3	50×5	$1.3e^{-3}$	$2.9e^{-3}$
12×3	100×10	$1.3e^{-3}$	$2.9e^{-3}$
100×10	12×3	$6.4e^{-8}$	$1.5e^{-7}$
100×10	25×3	$3.3e^{-7}$	$1.6e^{-7}$
100×10	50×5	$8.8e^{-7}$	$1.3e^{-6}$
100×10	100×10	$1.1e^{-10}$	$5.8e^{-10}$

Table 14: Variable mesh discretization, TPS and CPC2 basis functions.

Source discretization	Target discretization	Radius	Error
12×3	100×10	$0.25L$	$6.6e^{-3}$
		$0.5L$	$2.9e^{-3}$
		$0.75L$	$1.8e^{-3}$
		L	$1.4e^{-3}$
100×10	12×3	$0.25L$	$1.9e^{-6}$
		$0.5L$	$1.5e^{-7}$
		$0.75L$	$6.5e^{-8}$
		L	$3.8e^{-8}$

Table 15: CPC2 basis function with variable radius.

Appendix G

Analytical solution for the axisymmetric thermal problem in a hollow cylinder

A hollow cylinder with inner and outer diameter denoted by D_i and D_o, respectively, is here considered. It is proposed to solve the steady heat equation in the solid domain Ω_s under the assumption of an axisymmetric problem and with the following boundary conditions: a constant uniform temperature T_i imposed at the inner boundary and either a constant uniform temperature T_o or a constant uniform heat flux $q_\mathrm{o} = (\boldsymbol{q} \cdot \boldsymbol{n})_\mathrm{o}$ imposed at the outer boundary. The problem is depicted in Fig. 26. Note that both boundary conditions are represented on the outer boundary but only one is considered at a time.

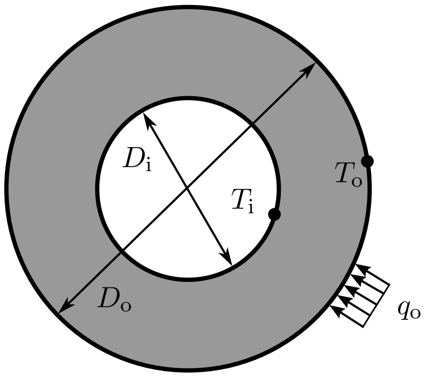


Figure 26: Geometry of the hollow cylinder with boundary conditions for the thermal analysis.

The governing equations reduces here to the simple Laplace's equation,

$$\nabla^2 T = 0\,, \tag{56}$$

which reads in polar coordinates (r, θ)

$$\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right)\,, \tag{57}$$

when considering an axisymmetric problem $(\partial \cdot /\partial\theta = 0)$. A straightforward solution of this equation is given by

$$T(r) = A \ln r + B\,, \tag{58}$$

in which A and B are constants depending on the boundary conditions. When the temperature is imposed on the outer boundary,

$$A = \frac{T_\mathrm{i} - T_\mathrm{o}}{\ln\left(D_\mathrm{i}/D_\mathrm{o}\right)} \tag{59}$$

while

$$A = \frac{q_\mathrm{o} R_\mathrm{o}}{\lambda} \tag{60}$$

when a heat flux is imposed on the outer boundary. In any case, the constant B is given by

$$B = T_\mathrm{i} - A \ln(D_\mathrm{i}/2)\,. \tag{61}$$