Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

FSI procedures for Civil Engineering Applications.

TREBALL FINAL DE MÀSTER

Treball realitzat per:
Rubén Zorrilla Martínez

Dirigit per:
Dr. Riccardo Rossi
Dra. Antonia Larese

Màster en:
Mètodes Numèrics en Enginyeria

Barcelona, juny 2016

Departament d'Enginyeria Civil i Ambiental

FSI procedures for Civil Engineering Applications

Rubén Zorrilla Martínez

Master's degree on Numerical Methods in Engineering

Escola Tècnica Superior d'Enginyers de Camins Canals i Ports de Barcelona

Universitat Politècnica de Catalunya

Supervisors: Dr. Riccardo Rossi and Dr. Antonia Larese.

June 2016

Acknowledgements

First of all, I want to acknowledge *Catalunya-La Pedrera* foundation for awarding me with one of their excellence master grants. Furthermore, I want to also acknowledge professor Eugenio Oñate as well as the *International Center for Numerical Methods in Engineering* (CIMNE) for the economic support during the materialization of this work.

Secondly, my gratitude also goes to my supervisors, Dr. Riccardo Rossi and Dr. Antonia Larese. I really appreciate the confidence they put on me since the very beginning and their time and effort dedication.

Moreover, there are more people that have selflessly helped during the realisation of this master thesis. I thank all of them but I must specially recognise Dr. Alfred Bogaers for the guidelines in the implementation of the Multivector Quasi-Newton method as well as my office colleagues Vicente Mataix and Andrés Adam for their daily collaboration.

Last but not least, my deepest and most sincere dedication goes to my family, specially to my father Juan Fco. for being and example of tireless fight during the last years, to my mother Maria Luisa for her unconditional love, to Anna for being always by my side and to the memory of my grandmother Hortensia.

Abstract

Several engineering problems need to account for Fluid-Structure Interaction (FSI) in order to have reliable results. Is for this reason that its simulation in an efficient and reliable manner is, since the last decades, a field of study in the computational engineering science.

In this work a fluid-structure interaction black box resolution environment has been developed. In the black box FSI resolution, both the solid and fluid solvers are conceived as black boxes that take and input data to give back a solution. This allows to focus on the interaction mechanisms, taking advantage of a high reusability of existing codes. In this work the solid mechanics and fluid dynamics modules of Kratos Multiphysics have been used as field solvers.

Several coupling schemes and strategies have been developed, all of them based on the previous concept of black box coupling. Moreover, the capability of solving strongly coupled non-linear problems was also requested. These kind of problems imply a high influence in the solution of both the fluid and the solid domains because of its coupling, and likely drives to large displacements in the structure.

Complementary, it must be highlighted that the difficulty of the coupling is added to the inherent complexity of the fluid and solid problems, leading to a high resolution cost. Consequently, this work specially focuses in the analysis of the computational effort of each one of the implemented methods, among which the recently developed Multivector Quasi-Newton method stands out due to its efficiency.

Finally, several strongly coupled problems present in the literature have been solved in order to assess the performance of the implemented strategies.

Resumen

El problema de interacción fluido estructura está presente en múltiples procesos físicos y tecnológicos que suceden a diario. Es por este motivo que su simulación de una manera eficaz y fidedigna es desde las últimas décadas un tema de estudio en el ámbito de la ingeniería computacional.

En este trabajo se ha desarrollado un entorno de resolución del problema de interacción fluido estructura en el que los métodos de resolución de ambos dominios son concebidos como cajas negras, que a partir de unos datos de entrada devuelven una solución. Esto permite focalizar únicamente en los mecanismos de interacción y reutilizar códigos existentes, tal y como se hace en este trabajo, donde se han empleado los módulos de mecánica de sólidos computacional y de dinámica de fluidos de Kratos Multiphysics.

Así pues se han desarrollado diferences estrategias de acoplamiento, todas ellas basadas en el anterior concepto de cajas negras. Además, se estableció como requisito adicional la capacidad de solucionar problemas no-lineales fuertemente acoplados. Esta tipología de problemas implica una gran afectación en las soluciones de ambos dominios debido a su acoplamiento, y suele derivar en grandes desplazamientos en la estructura.

Complementariamente, se debe destacar que a la inherente complejidad del problema del sólido y la estructura se añade la de su acoplamiento, lo que resulta en un elevado coste computacional en su resolución. En consecuencia, se ha hecho especial hincapié en el análisis del coste computacional de los métodos implementados, entre los cuales se destaca el recientemente desarrollado *Multivector Quasi-Newton method* por su eficiencia.

Finalmente, se han reproducido múltiples problemas presentes en la literatura, todos ellos no-lineales y fuertemente acoplados, a fin de evalúar el rendimiento de cada una de las estrategias desarrolladas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Fluid-Structure Interaction problem

The Fluid-Structure Interaction problem, known by its acronym FSI, describes the existent interaction between a deformable body and the moving fluid that surrounds it. One can find lots of FSI problem examples in the nature, as diverse as the blood flood in the human cardiovascular system or the flapping of birds wings. In the engineering field, there are also multiple FSI problem examples such as the aeroelastic phenomena exerted by the wind over plane airfoils and light bridges or the design of immersed structures.

Regarding the nature of the FSI problem, it belongs to the so called coupled problems family. Hence, the resolution of the FSI problem implies to solve both the solid mechanics problem as well as the fluid dynamics one, taking into account that their solutions are coupled and depend on each other. As a consequence, the coupling difficulties, which may be determinant in multiple cases, are added to the inherent complexities of each one of the problems.

Due to such complexity, which makes its analytical resolution impossible in the majority of the cases, the FSI problem resolution is typically handled by means of experimental or numerical techniques. The main experimental tool in the assessment of the FSI problem are the wind tunnels. Wind tunnels have been a very useful tool for the structural aeroelastic design since its apparition. However, their construction and operation costs are hugely large, meaning that few companies can afford them. As a cheaper alternative, one can perform reduced scale tests but there are still certain material or geometrical behaviours that are impossible to scale.

In this context, numerical simulation techniques, as the ones presented in this work, have become an extremely good alternative to the wind tunnel tests since their apparition. The main advantage of numerical simulation is the possibility of performing a vast amount of tests at a much cheaper price. Furthermore, thanks to parallelization technology and supercomputers, the amount of time required to perform a FSI simulation is increasingly reduced. However, they are still prohibitive in some cases, meaning that more research effort must be put on the FSI problem optimization for the full technology transfer to the industry, which requires FSI simulation times according to the project schedules.

1.2 FSI simulation: Black-box partitioned schemes

The numerical resolution techniques for solving an FSI problem can be roughly divided in two main groups. The former is the monolithic resolution, in which the solid and fluid problems are merged as a unique large one. The latter are the so called partitioned schemes, which consist in keeping both problems separated and communicate their solutions by means of an intermediate procedure.

It has to be said that both techniques have advantages and disadvantages. Thus, monolithic solvers are understood as the best solvers to deal with strongly coupled problems, in which one field solution largely depends on the other one and vice versa. However, they have the large disadvantage of requiring the development of a complete new solver for the coupled FSI case.

On the contrary, partitioned schemes have the great advantage of allowing large code reusability since only minor modifications have to be done within each solver. One step further are the so called black-box partitioned coupling techniques, in which the fluid and solid solvers are understood as a black boxes that take some input data to give back a solution. As a consequence, no modifications are necessary to the existent solvers and only the coupling interface between them has to be programmed.

Apart from the reduction in time and human effort during the development, the black box partitioned strategies have the advantage of using existent or even commercial solvers for each one of the fields. This is translated in the use of already optimized and tested specific solvers for both the solid and fluid domains, driving to a better overall performance of the main FSI solver.

1.3 Objectives

The main objective of this work is the development of a black box FSI coupling resolution environment, starting from the computational fluid dynamics and computational solid mechanics technology already implemented in Kratos Multiphysics simulation framework.

Due to the inherent large computational effort required to solve the FSI problem, this work also focuses on the optimization of the coupling procedure. Therefore, several methods must be tried in order to find the most computationally efficient and stable one. To do that, some of the most common FSI benchmark examples present in the literature are reproduced to check the results and to assess the computational performance of the developed strategies.

Besides, this work also has some extra transversal objectives. The first one is to reach a deeper knowledge in the numerical simulation of the fluid dynamics and solid mechanics problems, which is understood to be necessary for the proper development of their coupling. On the other hand, this work aims to get into contact with Kratos Multiphysics framework as well as with their coding languages, Python and C++.

Last but not least, it has to be said that the developments of this work are expected to be implemented within the new Kratos GUI (Graphical User Interface), which has been recently released. Then, this work also includes some parallel tasks oriented towards the creation of the FSI Kratos GUI module.

1.4 Contents

In the following lines, the contents of the present document are briefly depicted.

Chapter 2 contains a review on the main aspects involving the numerical simulation of the fluid dynamics and the solid mechanics problems using the finite element method. This chapter also includes the peculiarities of the FSI problem resolution.

In chapter 3 the different FSI coupling strategies developed in this work are presented. Thus, the Dirichlet-Neumann and the Neumann-Neumann coupling schemes are depicted. In addition, the formulation of the black box residual minimization techniques used all along this work (relaxation schemes, Jacobian Free Newton-Krylov methods and Quasi-Newton methods) is developed.

Chapter 4 collects the results discussion. In particular the cavity flow problem domain decomposition, the Mok and Turek-Hron benchmarks as well as a 3D hemodynamics problem are presented.

Complementary, chapter 5 summarizes the developments in the last version of the new Kratos GUI and sketches the future FSI GUI module.

Finally, chapter 6 states the conclusions and the future work lines.

Chapter 2

State of the art

This chapter is aimed to be a review on the state of the art of the Fluid-Structure Interaction (FSI) simulation. Due to the multidisciplinary nature of the FSI problem, the chapter has been divided in four sections. In the first one, a general overview on continuum mechanics is done. The second and third ones review the main aspects of the Finite Element (FE) simulation of the structural and fluid problems. Finally, the fourth section is devoted to the implications of the solid and structural problems coupling.

2.1 Review on continuum mechanics

This section is intended to be a brief review on the continuum mechanics basics that are needed for the comprehension of the rest of the work. Thus, the kinematics involving a body in motion as well as the strain and stress measures are presented. Finally, both the balance and constitutive equations used along the work are also commented. A more detailed review on continuum mechanics can be found in [33], [18] or [46] among many other books. Particularly, in chapter 4 of [13] a quite good explanation about strain and stress measures can be found.

2.1.1 Kinematics

Kinematics is the study of motion and deformation of a body without regard to the forces responsible for such action. The position at time $t = 0$ is called *initial configuration* and is denoted as Ω_0. Otherwise stated, the initial configuration is taken as the so called *reference configuration.* The reference configuration is needed to refer the movement equations from the initial configuration Ω_0 to the *current configuration* Ω.

Taking into account that the continuum body is considered to be composed by a set of particles called *material points*, the position vector of any material point in the reference configuration is defined by $\mathbf{X}$. The value of $\mathbf{X}$ is

$$\mathbf{X} = X_i \mathbf{e}_i = \sum_{i=1}^{n_{dim}} X_i \mathbf{e}_i \tag{2.1}$$

where n_{dim} are the number of dimensions and $\mathbf{e}_i$ are the unit base vectors of a rectangular Cartesian coordinate system. $\mathbf{X}$ coordinates are called *material* or *Lagrangian coordinates.* The motion of the body is described by the *deformation map* $\boldsymbol{\Phi}$. As can be seen in Eq. 2.2, given the position of a particle in the reference configuration, the deformation map turns back the so called *spatial* or *Eulerian coordinates* in the current configuration. The expression of the deformation map is

$$\mathbf{x} = \boldsymbol{\Phi}(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t) \tag{2.2}$$

where $\mathbf{x}$ are the spatial coordinates whose value is

$$\mathbf{x} = x_i \mathbf{e}_i = \sum_{i=1}^{n_{dim}} x_i \mathbf{e}_i \tag{2.3}$$

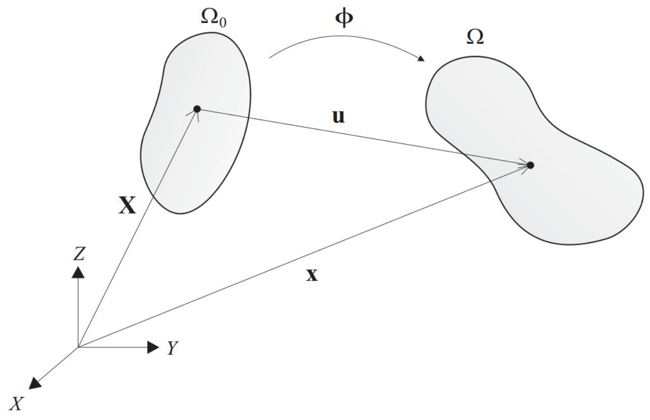


Figure 2.1: Configurations of a body. Image taken from [44].

Two possible descriptions of the movement arise from the two presented type of coordinates:

- Material or Lagrangian description: Material coordinates $\mathbf{X}$ and time t are taken as independent variables. Typically used in solid mechanics.
- Spatial or Eulerian description: Spatial coordinates $\mathbf{x}$ and time t are taken as independent variables. Typically used in fluid mechanics.

On the other hand, the difference between the current and the reference configuration gives the *displacement* $\mathbf{u}$ which can be expressed in material coordinates as

$$\mathbf{u}(\mathbf{X},t) = \mathbf{x} - \mathbf{X} = \boldsymbol{\Phi}(\mathbf{X},t) - \mathbf{X} \tag{2.4}$$

The *material velocity* is the rate of change of the position vector. It is obtained as the *material* or *total time derivative* (derivative when $\mathbf{X}$ is held constant) of the position which is expressed as

$$\mathbf{v}(\mathbf{X},t) = \frac{\partial \mathbf{x}(\mathbf{X},t)}{\partial t} = \frac{\partial \mathbf{u}(\mathbf{X},t)}{\partial t} = \dot{\mathbf{v}}(\mathbf{X},t) \tag{2.5}$$

In the same way, the *material acceleration* is rate of change of the velocity vector, what is to say the material time derivative of the velocity given by

$$\mathbf{a}(\mathbf{X},t) = \frac{\partial \mathbf{v}(\mathbf{X},t)}{\partial t} = \dot{\mathbf{v}}(\mathbf{X},t) = \ddot{\mathbf{u}}(\mathbf{X},t) \tag{2.6}$$

For the case of an Eulerian description of the variables, e.g. the fluid dynamics case, the material time derivative of any variable expressed in terms of spatial coordinates $\mathbf{x}$ and time t can be obtained with

$$\frac{D(\bullet)}{Dt} = \frac{\partial(\bullet)}{\partial t} + \mathbf{v}\cdot\nabla(\bullet) \tag{2.7}$$

where the first term is the *spatial time derivative* meanwhile the second one is the so called *convective term.*

Associated to the movement from the reference configuration Ω_0 to the current configuration Ω there is a change in the size and/or the shape of the body. This change is called *deformation* and is measured via the *deformation gradient tensor* given by

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \nabla \mathbf{x} \quad \textit{or} \quad F_{ij} = \frac{x_i}{X_j} \tag{2.8}$$

The deformation gradient above can be also expressed in terms of the displacements by means of Eq. 2.4 as follows

$$\mathbf{F} = \frac{\partial \mathbf{X}}{\partial \mathbf{X}} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}} = \mathbf{I} + \nabla \mathbf{u} \quad or \quad F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial X_j} \tag{2.9}$$

Finally, it is interesting to pinpoint another widely used quantity related to **F** which is the *Jacobian determinant* obtained as

$$J = det(\mathbf{F}) \tag{2.10}$$

2.1.2 Strain measures

The strain is defined as the measure of the geometrical deformation caused by the forces applied on a continuum body. For Lagrangian descriptions, the essential strain measure is the *Green-Lagrange strain tensor* defined as

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) \quad or \quad E_{ij} = \frac{1}{2}(F_{ij}^T F_{ij} - \delta_{ij}) \tag{2.11}$$

The Green-Lagrange strain tensor can be rewritten in terms of the displacements by means of Eq. 2.9 yielding

$$E_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} + \frac{\partial u_k}{\partial X_i}\frac{\partial u_k}{\partial X_j}\right) \tag{2.12}$$

For small deformation problems, the non-linear term in Eq. 2.12 can be neglected yielding the so called *infinitesimal strain tensor* defined as

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i}\right) \tag{2.13}$$

On the other hand, the *spatial velocity gradient tensor* is given by

$$\mathbf{l} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}} \quad or \quad l_{ij} = \frac{\partial v_i}{x_j} \tag{2.14}$$

and can be decomposed into a symmetric and a skew-symmetric part by means of

$$\mathbf{l} = \frac{1}{2}(\mathbf{l} + \mathbf{l}^T) + \frac{1}{2}(\mathbf{l} - \mathbf{l}^T) = \mathbf{d} + \mathbf{w} \tag{2.15}$$

The symmetric term **d** in Eq. 2.15 is called *rate of deformation tensor* while the skew-symmetric **w** is called *spin or rate or rotation tensor.*

2.1.3 Stress measures

The stress is defined as the amount of force per unit area. In continuum mechanics field, its value is given by the so called *surface traction vector* denoted by $\mathbf{t}$. Furthermore, the *Cauchy's theorem* states that there exists a tensor field $\boldsymbol{\sigma}$, known as *Cauchy stress tensor*, such that for each unit normal vector $\mathbf{n}$ its surface traction vector is obtained as

$$\mathbf{t} = \mathbf{n} \cdot \boldsymbol{\sigma} = \boldsymbol{\sigma}^T \cdot \mathbf{n} \tag{2.16}$$

Regarding the Cauchy stress tensor, it is interesting to point out that $\boldsymbol{\sigma} = \boldsymbol{\sigma}^T$ due to the conservation of angular momentum.

The Cauchy's theorem can also be expressed in the reference configuration as

$$\mathbf{t}_0 = \mathbf{n}_0 \cdot \mathbf{P} \tag{2.17}$$

where $\mathbf{P}$ is the so called *nominal stress tensor*. It is a must to comment that the nominal stress tensor is not symmetric unlike the Cauchy stress tensor. The transpose of the nominal stress tensor is known as *first Piola-Kirchhoff stress tensor*. The *second Piola-Kirchhoff stress tensor* $\mathbf{S}$ is symmetric and defined as

$$\mathbf{F}^{-1} \cdot \mathbf{t}_0 = \mathbf{n}_0 \cdot \mathbf{S} \tag{2.18}$$

Finally, it is interesting to state some useful transformation between the previous stress tensors. Such transformation are listed below.

$$\boldsymbol{\sigma} = J^{-1}\mathbf{F} \cdot \mathbf{P} = J^{-1}\mathbf{F} \cdot \mathbf{S} \cdot \mathbf{F}^T \tag{2.19}$$

$$\mathbf{P} = J\mathbf{F}^{-1} \cdot \boldsymbol{\sigma} = \mathbf{S} \cdot \mathbf{F}^T \tag{2.20}$$

$$\mathbf{S} = J\mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} = \mathbf{P} \cdot \mathbf{F}^{-T} \tag{2.21}$$

2.1.4 Conservation equations

The conservation equations state that certain physical magnitude must be always satisfied in the whole problem domain. Thus, they are expressed as an integral relation in the entire domain Ω. Due to the additive property of the integral, the conservation equations must be also satisfied in any subdomain of the whole problem domain, allowing to be expressed as partial differential equations.

In this subsection, the mass, linear momentum, angular momentum and energy conservation equation are presented in their partial differential form. A better explanation of the derivation of such equations can be found in [18] or [33].

Mass conservation

The *mass conservation equation* states that if there are no mass sources or sinks, the mass of any material domain must remain constant. Hence, the mass can be expressed in different configurations as

$$m = \int_{\Omega} \rho(\mathbf{X}, t) d\Omega = \int_{\Omega_0} \rho(\mathbf{X}, t) J d\Omega_0 = \int_{\Omega_0} \rho_0(\mathbf{X}) d\Omega_0 \tag{2.22}$$

Note that time dependency of the mass is located in the density ρ. Hence, the mass conservation equation can be expressed in terms of the density as

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 \tag{2.23}$$

The previous equation is also known as *continuity equation* and is found applying the *Reynold's transport theorem* to Eq. 2.22.

In addition, there is the case of *incompressible materials.* In these cases, the density keeps constant in time and the material time derivative in Eq. 2.23 vanishes. Hence, the continuity equation for the incompressible case is

$$\nabla \cdot \mathbf{v} = 0 \tag{2.24}$$

Linear momentum conservation

The *conservation of linear momentum*, also known as *balance of linear momentum*, states that the rate of change of the linear momentum is equal to the total applied force. Somehow, the conservation of linear momentum can be viewed as the Newton's second law. The balance of linear momentum is expressed as

$$\frac{D}{Dt} \int_{\Omega} \rho \mathbf{v}(\mathbf{x}, t) d\Omega = \int_{\Omega} \rho \mathbf{b}(\mathbf{x}, t) d\Omega + \int_{\Gamma} \mathbf{t}(\mathbf{x}, t) d\Gamma \tag{2.25}$$

Applying the Reynold's lemma, the mass conservation equation and the Gauss' divergence theorem to the linear momentum definition in Eq. 2.25 yields the *momentum equation* which reads as follows

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{2.26}$$

In static problems, the acceleration term in Eq. 2.26 is negligible. When this term is dropped, the momentum equation is called *equilibrium equation.*

In an Eulerian description, the material time derivative in Eq. 2.26 is developed by means of Eq. 2.7 yielding the so called *Eulerian formulation* widely used in fluid mechanics. The Eulerian formulation can be expressed as

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{2.27}$$

Note that the Eulerian formulation implies the evaluation of the divergence with respect to the spacial coordinates since Eq. 2.27 is in the current configuration.

On the other hand, in a Lagrangian description the material time derivative can be directly computed as the partial time derivative. In nonlinear solid mechanics, this is the so called *updated Lagrangian formulation.* The *updated Lagrangian formulation* reads

$$\rho \frac{\partial \mathbf{v}}{\partial t} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{2.28}$$

Finally, the conservation of linear momentum can also be expressed in the reference configuration for Lagrangian coordinates. In nonlinear solid mechanics, this is called *total Lagrangian formulation* and is expressed as

$$\rho_0 \frac{\partial \mathbf{v}}{\partial t} = \nabla_0 \cdot \mathbf{P} + \rho_0 \mathbf{b} \tag{2.29}$$

where $\nabla_0 \cdot (\bullet)$ is the divergence taken with respect to material coordinates and $\mathbf{P}$ is the nominal stress tensor.

Angular momentum conservation

The *conservation of angular momentum* is obtained taking the cross product of the current position vector $\mathbf{x}$ by each term of the linear momentum Eq. in 2.26, yielding

$$\frac{D}{Dt}\int_{\Omega}\mathbf{x}\times\rho\mathbf{v}(\mathbf{x},t)d\Omega=\int_{\Omega}\mathbf{x}\times\rho\mathbf{b}(\mathbf{x},t)d\Omega+\int_{\Gamma}\mathbf{x}\times\mathbf{t}(\mathbf{x},t)d\Gamma \tag{2.30}$$

As has been commented before, it can be proved that the previous equation leads to

$$\boldsymbol{\sigma}=\boldsymbol{\sigma}^{T} \tag{2.31}$$

meaning that the Cauchy stress tensor is always symmetric by definition. This implies that

$$\mathbf{F}\cdot\mathbf{P}=\mathbf{F}^{T}\cdot\mathbf{P}^{T} \tag{2.32}$$

which is in general not symmetric. Moreover, using the previously presented stress tensors transformations, one can obtain that

$$\mathbf{S}=\mathbf{S}^{T} \tag{2.33}$$

meaning that the second Piola-Kirchhoff stress is also a symmetric tensor.

Energy conservation

The *conservation of energy principle* requires that the power of the total energy of a body, which is the sum of the kinetic and the internal energy, equals the power of the applied forces plus the power of the extra energy entering the domain such as the heat sources or radiation.

On one hand, the power of the total energy is given by

$$\mathcal{P}^{tot}=\mathcal{P}^{int}+\mathcal{P}^{kin}=\frac{D}{Dt}\int_{\Omega}\rho w^{int}d\Omega+\frac{D}{Dt}\int_{\Omega}\frac{1}{2}\rho\mathbf{v}\cdot\mathbf{v}d\Omega \tag{2.34}$$

On the other hand, the *1st law of thermodynamics* also states that

$$\mathcal{P}^{tot}=\mathcal{P}^{ext}+\mathcal{P}^{heat}=\int_{\Omega}\mathbf{v}\cdot\rho\mathbf{b}d\Omega+\int_{\Gamma}\mathbf{v}\cdot\mathbf{t}d\Gamma+\int_{\Omega}\rho s d\Omega-\int_{\Gamma}\mathbf{n}\cdot\mathbf{q}d\Gamma \tag{2.35}$$

Note that in this case only heat power has been considered as extra energy for the sake of simplicity. Equalling Eq. 2.34 to Eq. 2.35 yields the conservation of energy equation, which can be expressed in Eulerian partial differential form as

$$\rho \frac{Dw^{int}}{Dt} = \boldsymbol{\sigma} : \mathbf{d} - \nabla \cdot \mathbf{q} + \rho s \tag{2.36}$$

For a purely mechanical process as the ones presented in this work, the energy equation turns to be

$$\rho \frac{Dw^{int}}{Dt} = \boldsymbol{\sigma} : \mathbf{d} \tag{2.37}$$

The previous equation shows that the Cauchy stress tensor $\boldsymbol{\sigma}$ and the rate of deformation tensor $\mathbf{d}$ are conjugate in power. Complementary, the same purely mechanical process energy equation can be also expressed in Lagrangian coordinates as

$$\rho_0 \dot{w}^{int} = \mathbf{P} : \dot{\mathbf{F}}^T = \mathbf{S} : \dot{\mathbf{E}} \tag{2.38}$$

showing that the nominal stress tensor $\mathbf{P}$ is conjugate in power to the material time derivative of the deformation gradient tensor $\dot{\mathbf{F}}$ and that the second Piola-Kirchhoff stress tensor $\mathbf{S}$ is conjugate in power to the material time derivative of the Green-Lagrange strain tensor $\dot{\mathbf{E}}$.

2.1.5 Constitutive equations

So far, the presented equations are insufficient to describe the mechanical behaviour of any material. Therefore, the previous set of equations have to be completed with the so called *constitutive equations*, which specify the mechanical properties of a material and its stress-strain relation.

It has to be said that there are lots of constitutive laws in the existing bibliography and each one is suitable for one type of material present in the nature.

In this work, only purely mechanical processes are treated meaning that in the presented constitutive equations the stress state uniquely depends on the kinematic state. In the next subsections, the constitutive equations used in this work are discussed.

Linear elasticity

Linear elasticity theory is suitable for problems in where the analysed body undergoes small changes of shape and there is barely difference between the reference and the current configurations. In this cases the linearised infinitesimal strain tensor presented in Eq. 2.13 can be used. Thus, the energy conservation equation reads

$$\rho_0 \dot{w}^{int} = \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}} \tag{2.39}$$

The term $\rho_0 w^{int}$ in the previous equation is usually denoted by W^{int} which is the so called *strain energy function*. For the linear elasticity case, the strain energy function depends only on the infinitesimal strain tensor and is defined as

$$W^{int} = \frac{1}{2}\boldsymbol{\varepsilon} : \mathbb{C} : \boldsymbol{\varepsilon} \tag{2.40}$$

where $\mathbb{C}$ is a 4th order tensor collecting the elastic constants. For an isotropic material, as the ones considered in this work, the 81 components of $\mathbb{C}$ reduce to only two constants. These constants are λ and μ and are called *Lamé parameters*. Thus, for an isotropic material $\mathbb{C}$ is defined as

$$\mathbb{C} = \lambda \mathbf{I} \otimes \mathbf{I} + 2\mu \mathbb{I} \tag{2.41}$$

where $\mathbf{I}$ is the second order identity tensor while $\mathbb{I}$ is the fourth order symmetric identity tensor.

Applying the definition of the strain energy function in Eq. 2.40 into Eq. 2.39 one obtains that the constitutive equation that relates strain and stresses for the linear elastic case is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon} \tag{2.42}$$

Finally, the constitutive tensor in Eq. 2.41 can be substituted in Eq. 2.42. This yields the final linear elastic constitutive equation which can be expressed as

$$\boldsymbol{\sigma} = \lambda tr(\boldsymbol{\varepsilon})\mathbf{I} + 2\mu\boldsymbol{\varepsilon} \tag{2.43}$$

Nonlinear elasticity

Some problems may involve small strains and large deformations coming from large displacements and rotations. In this cases, the mechanical response can be modelled by means of the *Kirchhoff - Saint Venant* material model, which is a generalization of the linear elasticity theory to the *nonlinear elasticity* case.

Thus, the strain energy function is a generalization of the one presented in Eq. 2.40 and reads

$$W^{int} = \frac{1}{2}\mathbf{E} : \mathbb{C} : \mathbf{E} \tag{2.44}$$

Besides, the counterpart of equation 2.42 is

$$\mathbf{S} = \mathbb{C} : \mathbf{E} \tag{2.45}$$

where $\mathbb{C}$ is the same fourth order tensor depicted in Eq. 2.41. Therefore, the constitutive equation for nonlinear elastic materials is

$$\mathbf{S} = \lambda tr(\mathbf{E})\mathbf{I} + 2\mu\mathbf{E} \tag{2.46}$$

Finally, it is interesting to point out some expressions that allow to express λ and ν in terms of other elastic variables widely used in elasticity theory such as the Young modulus E, the Poisson ratio ν or the bulk modulus K. These expressions are listed below.

$$\mu = \frac{E}{2(1+\nu)} \qquad \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \qquad K = \lambda + \frac{2}{3}\mu \tag{2.47}$$

Newtonian fluid

In *Newtonian fluids*, the stress state is linearly related with the rate of deformation instead of the proper value of the deformation. Besides, the stress state can be divided in two contributions: the pressure contribution and the viscous contribution.

The former, comes from the *thermodynamic pressure* p and appears even in fluids at rest. In this case, the stress tensor components are orthogonal to the boundaries and can be obtained as

$$\boldsymbol{\sigma} = -p\mathbf{I} \tag{2.48}$$

The latter, is related with the fluid movement meaning that that moving fluids develop an additional stress component due to viscosity. This extra stress component is linearly related with the strain rate tensor $\mathbf{d}$ as follows

$$\boldsymbol{\sigma} = -p\mathbf{I} + \mathbb{C}\mathbf{d} \tag{2.49}$$

where $\mathbb{C}$ is once again the same fourth order tensor depicted in Eq. 2.41. Rearranging terms and using the Stokes assumption $\lambda + \frac{2}{3}\mu = 0$ one obtains the next constitutive equation for Newtonian fluids

$$\boldsymbol{\sigma} = -\left(p + \frac{2}{3}\mu\nabla \cdot \mathbf{v}\right)\mathbf{I} + 2\mu\mathbf{d} \tag{2.50}$$

For incompressible fluid flows, the incompressibility condition in Eq. 2.24 can be applied yielding the simpler form

$$\boldsymbol{\sigma} = -p\mathbf{I} + 2\mu\mathbf{d} \tag{2.51}$$

2.2 Computational structural mechanics

This section reviews the numerical resolution of the structural mechanics problem using the *Finite Element Method* (FEM). For the sake of generality, the finite deformation assumption has been considered.

First of all, the variational or weak form of the problem is stated using the *Principle of Virtual Work* (PVW). Then, the space discretization via finite elements approximations as well as the most common time discretization schemes are also presented. Finally, the problem resolution with numerical techniques is also addressed.

The main references for this section have been [13] and [46]. Besides, [36] includes a pretty clear explanation of the FE method basis and its application to the infinitesimal deformation case. Complementary, in [44] all the computational solid mechanics concepts involved in the FSI problem simulation are discussed.

2.2.1 Total Lagrangian weak form

In order to construct finite element approximations of the structural problem solution, it is necessary to write the formulation in a Galerkin (weak) or variational form. Since in the finite deformation assumption the reference and current configurations are not almost equal, which is the case in the infinitesimal deformation assumption, one have the choice of writing the formulation either in the reference or in the current configuration.

In this work, the total Lagrangian formulation is adopted. The total Lagrangian method is suitable for finite displacements and small strains. However, it can be also applied to large elastic strains if an appropriate hyperelastic constitutive law is used [13]. Moreover, for such finite deformation case, it becomes the simplest approach since the integrals are expressed over the reference configuration domains, which will remain constant during the deformation process. Later on, the results in terms of the reference configuration can be transformed and written in terms of the deformed configuration.

First of all, the problem in strong form must be stated. Therefore, starting from the linear momentum equation in total Lagrangian configuration depicted in Eq. 2.29 and adding some boundary and initial conditions, the strong form of the structural problem in index notation can be expressed as

$$\begin{aligned} \rho_0 \ddot{u}_i &= \frac{\partial P_{ji}}{\partial X_j} + \rho_0 b_i && \textit{in } \Omega_0 \\ u_i &= \bar{u}_i^0 && \textit{at } t = 0 \\ u_i &= \bar{u}_i && \textit{on } \Gamma^D \\ t_i &= \bar{t}_i && \textit{on } \Gamma^N \end{aligned} \tag{2.52}$$

where Ω_0 is the reference structure domain, Γ^D is the Dirichlet boundary and Γ^N is the traction or Neumann boundary.

First of all, the spaces for both the test functions $\delta u_i(\mathbf{X})$ and the trial displacement functions $u_i(\mathbf{X}, t)$ must be stated. The space of test functions is defined as

$$\delta u_i(\mathbf{X}) \in \mathcal{U}_0, \;\; \mathcal{U}_0 = \left\{ \delta u_i \mid \delta u_i \in \mathcal{C}^0(\mathbf{X}),\; \delta u_i = 0 \textit{ on } \Gamma^D \right\} \tag{2.53}$$

while the space of displacement functions is defined as

$$u_i(\mathbf{X}, t) \in \mathcal{U}, \;\; \mathcal{U} = \left\{ u_i \mid u_i \in \mathcal{C}^0(\mathbf{X}),\; u_i = \bar{u}_i \textit{ on } \Gamma^D \right\} \tag{2.54}$$

Note that the spaces of test and trial functions are similar except that the test displacement functions vanishes wherever the displacement is prescribed.

Hence, taking the product of the test displacement by the linear momentum equation and integrating over the whole reference configuration domain, yields the next Galerkin-type variational form

$$\int_{\Omega_0} \delta u_i \left(\frac{\partial P_{ji}}{\partial X_j} + \rho_0 b_i - \rho_0 \ddot{u}_i \right) d\Omega_0 = 0 \tag{2.55}$$

However, the first term of the previous equation does not accomplish the regularity requirements stated in Eq. 2.54 since the trial displacement functions need to be $\mathcal{C}^1$. Thus, integration by parts is applied to such term and the next final variational form is obtained

$$\int_{\Omega_0} \left(\delta F_{ij} P_{ji} - \delta u_i \rho_0 b_i + \delta u_i \rho_0 \ddot{u}_i \right) d\Omega_0 - \int_{\Gamma_0^N} \delta u_i \bar{t}_i d\Gamma_0 = 0 \tag{2.56}$$

Eq. 2.56 is the PVW, which can be written as

$$\delta \mathcal{W}^{int} - \delta \mathcal{W}^{ext} + \delta \mathcal{W}^{kin} = 0 \tag{2.57}$$

Note that the PVW states that the virtual internal work $\delta \mathcal{W}^{int}$ plus the virtual kinematic work $\delta \mathcal{W}^{kin}$ are in equilibrium with the virtual external work $\delta \mathcal{W}^{ext}$. These virtual works can be expressed in matrix form as

$$\delta \mathcal{W}^{int} = \int_{\Omega_0} \delta \mathbf{F}^T : \mathbf{P} d\Omega_0 = \int_{\Omega_0} \delta \mathbf{E} : \mathbf{S} d\Omega_0 \tag{2.58a}$$

$$\delta \mathcal{W}^{ext} = \int_{\Omega_0} \rho_0 \delta \mathbf{u} \cdot \mathbf{b} d\Omega_0 + \int_{\Gamma_0^N} \delta \mathbf{u} \cdot \bar{\mathbf{t}}_0 d\Gamma_0 \tag{2.58b}$$

$$\delta \mathcal{W}^{kin} = \int_{\Omega_0} \rho_0 \delta \mathbf{u} \cdot \ddot{\mathbf{u}} d\Omega_0 \tag{2.58c}$$

Note that in Eq. 2.58a some identities have been applied to express the virtual internal work in an alternative way. This will be quite useful for the space discretization. The step-by-step proof of this identity can be found in [44].

2.2.2 Finite element space discretization

In this subsection the finite element discretization of the total Lagrangian formulation in Eq. 2.56 is presented in such a way valid for both the 2D and 3D cases. This subsection tries to summarize chapter 5 of [46] in where a much deeper and clarifying explanation can be found.

The basic idea of the FE discretization is to divide the computational domain, in this case the reference domain Ω_0, in a finite number of conformant elements. Within each element, the motion can be approximated by

$$x_i^h(\mathbf{X},t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{X}) x_i^I(t) \qquad \forall i = 1, n_{dim} \tag{2.59}$$

where $N_I(\mathbf{X})$ are the *shape functions* of each node, n_{nodes} is the number of nodes in each finite element, $x_i^I(t)$ are the nodal values of the i-component of motion at node I and n_{dim} is the number of problem dimensions (2 or 3 in this work).
In a similar fashion, the displacements approximation reads

$$u_i^h(\mathbf{X},t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{X}) u_i^I(t) \qquad \forall i = 1, n_{dim} \tag{2.60}$$

and its corresponding velocity and acceleration approximations are

$$\dot{u}_i^h(\mathbf{X},t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{X}) \dot{u}_i^I(t) \qquad \forall i = 1, n_{dim} \tag{2.61}$$

$$\ddot{u}_i^h(\mathbf{X},t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{X}) \ddot{u}_i^I(t) \qquad \forall i = 1, n_{dim} \tag{2.62}$$

Besides this, the space of test functions, which is time independent, can be also discretized as

$$\delta u_i^h(\mathbf{X}) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{X}) \delta u_i^I \qquad \forall i = 1, n_{dim} \tag{2.63}$$

Once arrived to this point, it is convenient to state the *matrix* or *Voigt notation* to represent the stress, strain and variation of strain. For a general 3D case, the six-component form of the second Piola-Kirchhoff stress tensor $\mathbf{S}$ is

$$\mathbf{S} = \begin{bmatrix} S_{11} & S_{22} & S_{33} & S_{12} & S_{23} & S_{31} \end{bmatrix}^T \tag{2.64}$$

and the Green-Lagrange strain tensor $\mathbf{E}$ is

$$\mathbf{E} = \begin{bmatrix} E_{11} & E_{22} & E_{33} & 2E_{12} & 2E_{23} & 2E_{31} \end{bmatrix}^T \tag{2.65}$$

The variation of the Green-Lagrange strain tensor $\delta\mathbf{E}$ is similarly obtained as

$$\delta\mathbf{E} = \begin{bmatrix} \delta E_{11} & \delta E_{22} & \delta E_{33} & 2\delta E_{12} & 2\delta E_{23} & 2\delta E_{31} \end{bmatrix}^T \tag{2.66}$$

and can be computed taking the variation of the Green-Lagrange strain tensor in Eq. 2.11. Considering that $\delta F_{ij} = \frac{\partial \delta u_i}{\partial X_j}$, the variation of the Green-Lagrange strain can be expressed in terms of the displacement. Finally, if the previous displacements discretization is applied, yields the next matrix form of $\delta\mathbf{E}$

$$\delta\mathbf{E} = \hat{\mathbf{B}}^I \delta\mathbf{u}^I \tag{2.67}$$

where $\hat{\mathbf{B}}^I$ is the *nodal nonlinear strain-displacement matrix* which can be decomposed as

$$\hat{\mathbf{B}}^I = \mathbf{B}^I + \mathbf{B}^I_{NL} \tag{2.68}$$

being $\mathbf{B}^I$ identical to the infinitesimal deformation strain-displacement matrix and $\mathbf{B}^I_{NL}$ the remaining nonlinear part. These matrices can be found in chapter 5 of [46].

Recovering the PVW terms stated in Eq. 2.58 and substituting Eq. 2.67 into Eq. 2.58a yields the *internal forces vector* $\mathbf{f}^{int}$ whose components are obtained as

$$\mathbf{f}^{int}_I = \int_{\Omega_0} \hat{\mathbf{B}}^T_I \mathbf{S} d\Omega_0 \tag{2.69}$$

It is interesting to point out that the nonlinear behaviour arises in the previous internal forces vector. Such nonlinearity might come from either a nonlinear constitutive relation or large displacement or strains. A detailed explanation about the nonlinearity sources in the equilibrium equation can be found in [34].

Taking Eq. 2.58b and carrying out the same procedure, the *external force vector* $\mathbf{f}^{ext}$ is obtained as

$$\mathbf{f}^{ext}_I \int_{\Omega_0} N_I \rho_0 \mathbf{b} d\Omega_0 + \int_{\Gamma^N_0} N_I \bar{\mathbf{t}}^0 d\Gamma_0 \tag{2.70}$$

meanwhile the *inertial force vector* $\mathbf{f}^{kin}$ can be obtained from Eq. 2.58c and its components are

$$\mathbf{f}^{kin}_I = \int_{\Omega_0} N_I \rho_0 N_J d\Omega_0 \ddot{\mathbf{u}}_J = \mathbf{M}_{IJ} \ddot{\mathbf{u}}_J \tag{2.71}$$

where **M** is a *mass matrix* defined by

$$M_{IJ} = \int_{\Omega_0} N_I \rho_0 N_J d\Omega_0 \tag{2.72}$$

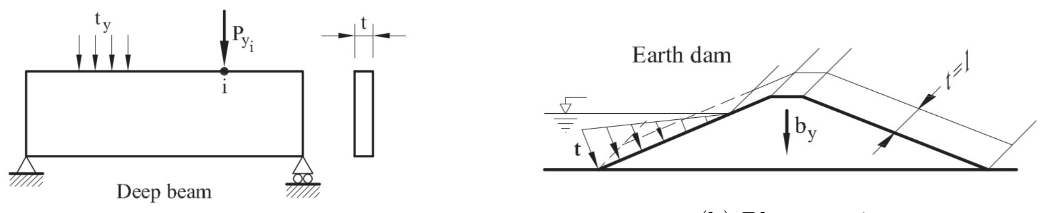
Finally, the *semi-discrete equations of motion*, which remain to be discretized in time, are given by

$$\mathbf{f}^{int}(\mathbf{S}) + \mathbf{M}\ddot{\mathbf{u}} = \mathbf{f}^{ext} \tag{2.73}$$

Last but not least, it is interesting to comment that this subsection presents the FE discretization in a general manner. If any further information about isoparametric quadrilateral, bricks, triangular and tetrahedral FE implementations is needed, the reader is encouraged to review either [36] or Annex A of [46]. For other types of FE discretizations such as beams, shells or membranes, more specific analysis can be found in [37] or [44].

2.2.3 Plane strain/stress cases

The 3D general case presented above may be simplified to a 2D one if loading, geometry, material behaviour and boundary conditions do not vary along the third coordinate. Then, two possible cases arise: plane stress and plane strain.



(a) Plane stress.

(b) Plane strain

Figure 2.2: 2D solid problem examples. Image taken from [36].

As can be seen in figure 2.2, plane stress theory can be applied to those structures in where the third dimension is much lower than the other two (e.g. slender beams). On the contrary, plane strain theory can be applied to those structures in where the third dimension is much larger than the other two (e.g. constant cross section gravity dams).

In plane strain problems, the strain in the third direction (the one orthogonal to the plane in where the problem is computed) is null, meaning that $\boldsymbol{\varepsilon}_{33} = 0$ in the infinitesimal deformation case or $\mathbf{E}_{33} = 0$ in the finite deformation one. The material constitutive matrix is the same but restricted to the 2D terms.

On the other hand, in plane stress problems the stress in the third direction is null, what is to say that $\boldsymbol{\sigma}_{33} = 0$ in the infinitesimal deformation case or $\mathbf{S}_{33} = 0$ in the finite deformation one. To do that, the material constitutive matrix is modified to satisfy this zero stress condition.

2.2.4 Time discretization

Time discretization schemes can be classified as *explicit* and *implicit*. Explicit schemes obtain the solution at time step t_{n+1} from the known values of the solution at time step t_n, meaning that no system of equations needs to be solved. Despite the fact that their implementation is simpler, explicit methods are conditionally stable and lead to larger computational cost since they require extremely small Δt to ensure stability.

On the other hand, implicit schemes obtain the solution at time step t_{n+1} not only from known values at time t_n but also considering the values at t_{n+1}. This implies to solve a system of equations at every time step but improves both the stability (in general, implicit schemes are unconditionally stable) and the precision. However, it has to be said that their implementation is more weird.

In this work, only implicit schemes have been considered due to their better performance. In the next subsections, the some of the most common time discretization schemes used in structural mechanics are pointed out. This section is mainly based in [44], where a very comprehensive explanation can be found. A much deeper analysis on time integrators can be found in [34].

Newmark method

Among the several time integration methods, the Newmark method is one of the most popular in structural dynamics. On one hand, the semi-discrete equations of motion to be solved at time t_{n+1} are given by

$$\mathbf{f}^{int}(\mathbf{u}_{n+1}) + \mathbf{M}\ddot{\mathbf{u}}_{n+1} = \mathbf{f}^{ext}(\mathbf{u}_{n+1}) \tag{2.74}$$

On the other hand, the displacements $\mathbf{u}_{n+1}$ and their time derivatives are approximated with the Newmark formulas below

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \dot{\mathbf{u}}_n + \Delta t^2 \left(\frac{1}{2} - \beta\right) \ddot{\mathbf{u}}_n + \Delta t^2 \beta \ddot{\mathbf{u}}_{n+1} \tag{2.75}$$

$$\dot{\mathbf{u}}_{n+1} = \dot{\mathbf{u}}_n + \Delta t \left(1 - \gamma\right) \ddot{\mathbf{u}}_n + \Delta t \gamma \ddot{\mathbf{u}}_{n+1} \tag{2.76}$$

where Δt is the time step and γ and β are the parameters that determine the stability and accuracy of the scheme. The method is second-order accuracy and unconditionally stable for $\gamma = 1/2$ and $\beta = 1/4$.
From Eq. 2.75 one can isolate $\ddot{\mathbf{u}}_{n+1}$ and substitute it into Eq. 2.76 to isolate $\dot{\mathbf{u}}_{n+1}$. Then, the acceleration term in Eq. 2.74 can be substituted, yielding the next algebraic system just in terms of the known displacements $\mathbf{u}_n$ and the sought displacements $\mathbf{u}_{n+1}$

$$\begin{aligned}\mathbf{f}^{int}(\mathbf{u}_{n+1}) + \frac{1}{\beta \Delta t^2}\mathbf{M}\mathbf{u}_{n+1} - f^{ext}(\mathbf{u}_{n+1}) = \\ \mathbf{M}\left[\frac{1}{\beta \Delta t^2}\mathbf{u}_n + \frac{1}{\beta \Delta t}\dot{\mathbf{u}}_n + \left(\frac{1}{2\beta} - 1\right)\ddot{\mathbf{u}}_n\right]\end{aligned} \tag{2.77}$$

Once the displacements $\mathbf{u}_{n+1}$ have been obtained, the velocity and acceleration at the current time step can be obtained as a post-process with the Newmark formulas above.

Bossak method

The Bossak method is an extension of the previously presented Newmark scheme, in where $\alpha_B = 0$. Thus, the Bossak scheme uses the Newmark formulas depicted in Eqs. 2.75 and 2.76 but the semi-discrete equation of motion is modified as follows

$$\mathbf{f}^{int}(\mathbf{u}_{n+1}) + \mathbf{M}\ddot{\mathbf{u}}_{n+1-\alpha_B} = \mathbf{f}^{ext}(\mathbf{u}_{n+1}) \tag{2.78}$$

where

$$\ddot{\mathbf{u}}_{n+1-\alpha_B} = (1 - \alpha_B)\ddot{\mathbf{u}}_{n+1} + \alpha_B \ddot{\mathbf{u}}_n \tag{2.79}$$

being

$$\alpha_B \in \left[-\frac{1}{3}, 0\right] \qquad \gamma = \frac{1 - 2\alpha_B}{2} \qquad \beta = \frac{(1 - \alpha_B)^2}{4} \tag{2.80}$$

Finally, substituting Eq. 2.79 into 2.78 and using the previously stated Newmark formulas when necessary, yields

$$\begin{aligned}\mathbf{f}^{int}(\mathbf{u}_{n+1}) + \frac{1-\alpha_B}{\beta\Delta t}\mathbf{M}\mathbf{u}_{n+1} - \mathbf{f}^{ext}(\mathbf{u}_{n+1}) = \\ \mathbf{M}\left[\frac{1-\alpha_B}{\beta\Delta t}\mathbf{u}_n + \frac{1-\alpha_B}{\beta\Delta t}\dot{\mathbf{u}}_n + \left(\frac{1-\alpha_B}{2\beta} - 1\right)\ddot{\mathbf{u}}_n\right]\end{aligned} \tag{2.81}$$

Generalised-α method

The generalised-α method achieves high-frequency dissipation while minimizing unwanted low-frequency dissipation. Once again, the Newmark formulas in Eqs. 2.75 and 2.76 are retained whereas the semi-discrete equations of motion are modified as follows

$$\mathbf{f}^{int}(\mathbf{u}_{n+\alpha_f^s}) + \mathbf{M}\ddot{\mathbf{u}}_{n+\alpha_m^s} = \mathbf{f}^{ext}(\mathbf{u}_{n+\alpha_f^s}) \tag{2.82}$$

where

$$\mathbf{u}_{n+\alpha_f^s} = (1-\alpha_f^s)\mathbf{u}_n + \alpha_f^s\mathbf{u}_{n+1} \tag{2.83}$$

$$\ddot{\mathbf{u}}_{n+\alpha_m^s} = (1-\alpha_m^s)\ddot{\mathbf{u}}_n + \alpha_m^s\ddot{\mathbf{u}}_{n+1} \tag{2.84}$$

Values of α_f^s and α_m^s for low frequency dissipation can be found in [7] or [44] as well as optimal values of γ and β for second-order accuracy.

Finally, the algebraic equations to be solved for $\mathbf{u}_{n+1}$ are given by

$$\begin{aligned}\mathbf{f}^{int}(\mathbf{u}_{n+\alpha_f^s}) + \frac{\alpha_m^s}{\beta\Delta t^2}\mathbf{M}\mathbf{u}_{n+1} - \mathbf{f}^{ext}(\mathbf{u}_{s+\alpha_f^s}) = \\ \mathbf{M}\left[\frac{\alpha_m^s}{\beta\Delta t^2}\mathbf{u}_n + \frac{\alpha_m^s}{\beta\Delta t}\dot{\mathbf{u}}_n + \left(\frac{\alpha_m^s}{2\beta} - 1\right)\ddot{\mathbf{u}}_n\right]\end{aligned} \tag{2.85}$$

2.2.5 Problem resolution: Newton-Raphson iterative scheme

This subsection briefly pinpoints the methodology to solve the non-linear system of equations that appears after the space and time discretization of the momentum equation. Among the many procedures to solve non-linear systems of equations, the widely used *Newton-Rahpson iterative method* has been used in this work.

In the Newton-Rahpson method the non-linear system to solve is firstly written in residual form. For the sake of simplicity, let us consider the system that results after applying the Newmark time discretization scheme (Eq. 2.77), whose residual is expressed as

$$\mathbf{R}_{n+1} = \mathbf{f}^{ext}(\mathbf{u}_{n+1}) - \frac{1}{\beta\Delta t^2}\mathbf{M}\mathbf{u}_{n+1} - \mathbf{f}^{int}_{n+1} + \mathbf{M}\left[\frac{1}{\beta\Delta t^2}\mathbf{u}_n\frac{1}{\beta\Delta t}\dot{\mathbf{u}}_n\left(\frac{1}{2\beta}-1\right)\ddot{\mathbf{u}}_n\right] \tag{2.86}$$

On the other hand, the previous residual can be expanded by means of Taylor's series. If the high order terms are neglected, the residual is approximated as

$$\mathbf{R}^i_{n+1} \approx \mathbf{R}^{i-1}_{n+1} + \left.\frac{\partial \mathbf{R}_{n+1}}{\partial \mathbf{u}_{n+1}}\right|^i \cdot \Delta\mathbf{u}^i_{n+1} \tag{2.87}$$

where $i-1$ is supposed to be a previous known iteration and i the iteration in where the solution is sought. Moreover, the *tangent stiffness matrix* can be defined as

$$\mathbf{K}^T = \frac{\partial \mathbf{R}}{\partial \mathbf{u}} \tag{2.88}$$

Considering that the previous residual in Eq. 2.87 must be zero, substituting the tangent stiffness matrix in Eq. 2.88 and rearranging terms the next algebraic system arises

$$\left(\mathbf{K}^T\right)^{i-1}_{n+1} \cdot \Delta\mathbf{u}^i_{n+1} = -\mathbf{R}^{i-1}_{n+1} \tag{2.89}$$

As can be seen, the previous system of equations is solved for $\Delta\mathbf{u}^i_{n+1}$, which is a correction of the previous iteration solution. Thus, the procedure is carried out until any convergence criteria (e.g. absolute or relative variation in $\Delta\mathbf{u}$) is reached, meaning that the residual tends to zero. The solution at the current iteration can be easily updated as

$$\mathbf{u}^i_{n+1} = \mathbf{u}^{i-1}_{n+1} + \Delta\mathbf{u}^i_{n+1} \tag{2.90}$$

Note that for the first iteration $i = 1$ of time step $n+1$ a prediction is needed. This problem can be easily overcome taking the previous step solution as initial guess for the first iteration at the current step, meaning that $\mathbf{u}^0_{n+1} = \mathbf{u}_n$.

Additionally, it is interesting to spend some words concerning the computation of the tangent stiffness matrix. Considering again Eq. 2.86 and disregarding the element formulation, the tangent stiffness matrix can be split as

$$\mathbf{K}^{T} = \mathbf{K}^{T,ext} - \mathbf{K}^{T,kin} - \mathbf{K}^{T,int} \tag{2.91}$$

Note that the external forces contribution $\mathbf{K}^{T,ext}$ is null if the external forces are conservative, what is to say that they do not depend on the solution. On the other hand, the $\mathbf{K}^{T,kin}$ contribution according to the considered Newmark scheme can be easily obtained as

$$\mathbf{K}^{T,kin} = \frac{1}{\beta \Delta t^2}\mathbf{M} \tag{2.92}$$

Regarding the $\mathbf{K}^{T,int}$ contribution, it has to be said that it completely depends on the considered constitutive law as well as element type. However, $\mathbf{K}^{T,int}$ is usually split as

$$\mathbf{K}^{T,int} = \mathbf{K}^{T,Mat} + \mathbf{K}^{T,Geo} \tag{2.93}$$

where $\mathbf{K}^{T,Mat}$ depends on the material constitutive law, which can be either linear or non-linear, whereas $\mathbf{K}^{T,Geo}$ stores the geometrical non-linearity contribution.

2.3 Computational fluid Dynamics

In this section, the finite element numerical resolution of the fluid dynamics problem is reviewed for the viscous-incompressible case. Thus, the Navier-Stokes equations as well as their weak form are firstly stated. Then, the space discretization via finite elements approximations and the time discretization are presented. Finally some of the most common stabilization techniques are briefly commented.

The main references that have been used in this bibliographic research are [17] and [38].

2.3.1 Viscous incompressible flows

Navier-Stokes equations

The motion of a fluid is governed by the balance of mass, momentum and energy equations. When the thermal effects are negligible, as is assumed all along this work, the energy equation is uncoupled and only the mass and momentum continuity equations, which combination results in the Navier-Stokes equations, need to be solved to obtain the velocity field $\mathbf{v}$ and the pressure field p.

Therefore, the Navier-Stokes equations can be obtained substituting the constitutive law for a Newtonian incompressible fluid stated in Eq. 2.51 into the Eulerian description of the linear momentum conservation stated in Eq. 2.27. Hence, the *general form of the incompressible Navier-Stokes equations* for a fluid with constant density ρ in a domain Ω for time $t > 0$ reads

$$\begin{cases} \dfrac{\partial \mathbf{v}}{\partial t} - \nabla \cdot \left[\nu \left(\nabla \mathbf{v} + \nabla \mathbf{v}^T\right)\right] + (\mathbf{v} \cdot \nabla) \cdot \mathbf{v} + \nabla p = \mathbf{b} \\ \nabla \cdot \mathbf{v} = 0 \end{cases} \tag{2.94}$$

Note in the previous equation that $\nu = \mu/\rho$ is called kinematic viscosity and p is in fact the pressure divided by the density (henceforth simply called pressure). Considering that such kinematic viscosity ν is constant, the diffusive term can be rearranged, yielding the Navier-Stokes equations form that is considered in this work. Finally, the Navier-Stokes equations for constant viscosity incompressible flows are

$$\begin{cases} \dfrac{\partial \mathbf{v}}{\partial t} - \nu \Delta \mathbf{v} + (\mathbf{v} \cdot \nabla) \cdot \mathbf{v} + \nabla p = \mathbf{b} \\ \nabla \cdot \mathbf{v} = 0 \end{cases} \tag{2.95}$$

or in index notation

$$\begin{cases} \dfrac{\partial v_i}{\partial t} - \nu \Delta v_i + \displaystyle\sum_{j=1}^{d} v_i \frac{v_i}{x_j} + \frac{\partial p}{x_i} = b_i \quad i = 1, \ldots, n_{dim} \\ \displaystyle\sum_{j=1}^{n_{dim}} \frac{\partial v_j}{\partial x_j} = 0 \end{cases} \tag{2.96}$$

Besides, in order to have a well posed problem it is necessary to assign the initial condition

$$\mathbf{v}(\mathbf{x}, 0) = \mathbf{v}_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega \tag{2.97}$$

where $\mathbf{v}_0$ is a given divergence-free vector field, together with suitable boundary conditions such as

$$\begin{cases} \mathbf{v}(\mathbf{x},t) = \phi(\mathbf{x},t) \quad \forall \mathbf{x} \in \Gamma_D \\ \left(\nu \dfrac{\partial \mathbf{v}}{\partial \mathbf{n}} - p\mathbf{n}\right)(\mathbf{x},t) = \psi(\mathbf{x},t) \quad \forall \mathbf{x} \in \Gamma_N \end{cases} \tag{2.98}$$

being ϕ and ψ given vector functions while Γ_D and Γ_N are a partition of the domain boundary such that $\partial\Omega = \Gamma_D \cup \Gamma_N$. As usual, $\mathbf{n}$ is the outwards unit normal vector to $\partial\Omega$.

Note that no initial condition is required for the pressure field. This is due to the fact that no time derivative of the pressure appears in the governing equations. When $\partial\Omega = \Gamma_D$, meaning that only Dirichlet boundary conditions are imposed, pressure is only present by its gradient and thus it is determined up to a constant. In this cases, it is usual to fix the pressure in one point to uniquely define the pressure field.

Furthermore, it has to be said that when the viscous effects are negligible (e.g. far from boundaries of the flow field) the diffusive term can be neglected, yielding the so called *Euler equations*. On the other hand, for low Reynolds number[1] flows, the non-linear convective term can be neglected yielding the so called *Stokes equations*.

Viscous incompressible flows main difficulties

Before introducing the numerical techniques used to solve the Navier-Stokes equations, it is interesting to state which are the main difficulties in the numerical simulation of viscous incompressible flows. On one hand, the presence of the non-linear and non-symmetric term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ in the momentum equation is the first difficulty to deal with, specially for high Reynolds number flows. Thus, the standard Galerkin formulations used may suffer from instabilities in convection-dominated flows, requiring the use of stabilization techniques such as the SUPG, GLS or OSS methods.

On the other hand, the incompressibility condition given by the mass conservation equation is another source of difficulty. As can be seen in Eq. 2.95 the incompressibility condition is in fact a restriction in the velocity field $\mathbf{v}$, which must be divergence-free. Thus, the presence of the pressure p in the momentum equation has the purpose of introducing an extra degree of freedom needed to satisfy the incompressibility condition. In other words, the pressure is acting as a Lagrange multiplier that enforces the incompressibility constraint, meaning that the velocity and pressure unknowns are coupled.

[1]For a characteristic velocity v_c and a characteristic length l_c, Reynolds number is defined as $Re = \rho v_c l_c / \mu$

Despite there are ways to avoid this coupling solving exclusively for the velocity (Penalty methods), in this work the traditional *mixed finite elements method*, which keeps velocity and pressure as unknowns is considered. This method presents numerical difficulties due to the saddle-point nature of the resulting variational problem, which implies that the velocity and pressure interpolation spaces must be compatible. This compatibility requirement is the so called *Ladyzhenskaya-Babuska-Brezzi condition*, commonly referred as *LBB-condition* or *inf-sup condition*. However, there exists FE formulations that allow to use velocity and pressure interpolation pairs that do not satisfy such LBB-condition. A much more detailed explanation regarding the analysis and resolution of saddle-point problems can be found in [38]. Regarding the LBB-condition, a very clarifying explanation as well as a collection of the most popular interpolation pairs can be found in [17].

2.3.2 Weak formulation of the Navier-Stokes equations

The strong form of the Navier-Stokes equations consists in the mass and momentum balance equations in Eq. 2.95 or 2.96, the initial condition in Eq. 2.97 and the Dirichlet and Neumann boundary conditions in 2.98. To develop the weak form, the space of test functions is defined as

$$\delta v_i(\mathbf{x}) \in \mathcal{V}_0, \quad \mathcal{V}_0 = \left\{ \delta v_i \mid \delta v_i \in H^1(\Omega), \; \delta v_i = 0 \; on \; \Gamma_D \right\} \tag{2.99}$$

where $H^1(\Omega)$ is the Hilbert space containing the functions whose components and their first derivatives are square-integrable. Note that once again the test function vanishes at the Dirichlet boundaries. On the other hand, the space of trial solutions for the velocity is defined as

$$v_i(\mathbf{x}) \in \mathcal{V}, \quad \mathcal{V} = \left\{ v_i \mid v_i \in H^1(\Omega), \; v_i = \phi \; on \; \Gamma_D \right\} \tag{2.100}$$

Thus, taking the linear momentum balance equation in Einstein notation, multiplying it by each component of the test function $\delta \mathbf{v}$ and integrating in Ω yields

$$\int_\Omega \delta v_i \frac{\partial v_i}{\partial t} d\Omega - \int_\Omega \delta v_i \nu \Delta v_i d\Omega + \int_\Omega \delta v_i v_j \frac{\partial v_i}{\partial x_j} d\Omega + \int_\Omega \delta v_i \frac{\partial p}{\partial x_i} d\Omega = \int_\Omega \delta v_i b_i d\Omega \tag{2.101}$$

Note that the second term involves the laplacian of the velocity field, increasing the regularity required for its interpolation. Same happens in the pressure derivative in the fourth term. Hence, these terms can be integrated by parts as

$$-\int_\Omega \delta v_i \nu \Delta v_i d\Omega = \int_\Omega \nu \frac{\partial \delta v_i}{\partial x_j}\frac{\partial v_i}{\partial x_j} d\Omega - \int_{\partial\Omega} \nu \delta v_i \frac{\partial v_i}{\partial x_j} n_j d\partial\Omega \tag{2.102a}$$

$$\int_\Omega \delta v_i \frac{\partial p}{\partial x_i} d\Omega = -\int_\Omega p \frac{\partial \delta v_i}{\partial x_i} d\Omega + \int_{\partial\Omega} p \delta v_i n_i d\partial\Omega \tag{2.102b}$$

Using the previous two relations in Eq. 2.101 one obtains

$$\begin{aligned}\int_\Omega \delta v_i \frac{\partial v_i}{\partial t} d\Omega + \int_\Omega \nu \frac{\partial \delta v_i}{\partial x_j}\frac{\partial v_i}{\partial x_j} d\Omega + \int_\Omega \delta v_i v_j \frac{\partial v_i}{\partial x_j} d\Omega + \int_\Omega p \frac{\partial \delta v_i}{\partial x_i} d\Omega = \\ \int_\Omega \delta v_i b_i d\Omega + \int_{\partial\Omega} \delta v_i \left(\nu \frac{\partial v_i}{\partial x_j} n_j - p n_i\right) d\partial\Omega \quad \forall \delta v \in \mathcal{V}_0\end{aligned} \tag{2.103}$$

or in vector notation

$$\begin{aligned}\int_\Omega \frac{\partial \mathbf{v}}{\partial t} \cdot \delta\mathbf{v} d\Omega + \int_\Omega \nu \nabla\mathbf{v} \cdot \nabla\delta\mathbf{v} d\Omega + \int_\Omega [(\mathbf{v}\cdot\nabla)\mathbf{v}] \cdot \delta\mathbf{v} d\Omega - \int_\Omega p \nabla \cdot \delta\mathbf{v} d\Omega = \\ \int_\Omega \mathbf{b} \cdot \delta\mathbf{v} d\Omega + \int_{\partial\Omega} \delta\mathbf{v} \cdot \left(\nu \frac{\partial \mathbf{v}}{\partial \mathbf{n}} - p\mathbf{n}\right) d\partial\Omega \quad \forall \delta\mathbf{v} \in \mathcal{V}_0\end{aligned} \tag{2.104}$$

where is easy to notice that the boundary term corresponds to the Neumann boundary condition ψ stated in Eq. 2.98, which is in fact the traction vector $\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n}$ being $\boldsymbol{\sigma}$ the sum of the viscous and hydrostatic stress for a Newtonian fluid depicted in Eq. 2.51.

Similarly, the weak form of the continuity equation can be obtained. To that purpose, its space of test functions is firstly defined as

$$\delta p(\mathbf{x}) \in \mathcal{Q}, \quad \mathcal{Q} = \left\{ \delta p \mid \delta p \in L^2(\Omega), \int_\Omega \delta p d\Omega = 0 \right\} \tag{2.105}$$

where $L^2(\Omega)$ is the is the space containing the functions that are square-integrable. The trial function for the pressure is also contained in $\mathcal{Q}$ meaning that

$$p(\mathbf{x}, t) \in \mathcal{Q} \tag{2.106}$$

Then, the Galerkin weak form of the incompressible mass conservation equation can be expressed in index notation as

$$\int_\Omega \delta p \frac{\partial v_i}{\partial x_i} d\Omega = 0 \quad \forall \delta p \in \mathcal{Q} \tag{2.107}$$

or in vector notation as

$$\int_\Omega \delta p \nabla \cdot \mathbf{v} = 0 \quad \forall \delta p \in \mathcal{Q} \tag{2.108}$$

2.3.3 Finite elements in viscous incompressible flows

Finite elements space discretization

In this section the finite element space discretization of the Galerkin weak form of the Navier-Stokes equations in Eqs. 2.104 and 2.108 is presented. Thus, the velocity interpolation reads as follows

$$v_i^h(\mathbf{x}, t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{x}) v_{iI}(t) \quad \forall i = 1, n_{dim} \tag{2.109}$$

being $N_I(\mathbf{x})$ the nodal shape functions in Eulerian coordinates and $v_{iI}(t)$ the velocity field nodal values. In a similar way, the test function, which is time-independent, is discretized as

$$\delta v_i^h(\mathbf{x}, t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{x}) \delta v_{iI} \quad \forall i = 1, n_{dim} \tag{2.110}$$

Additionally, the acceleration is approximated as

$$\frac{\partial v_i^h}{\partial t}(\mathbf{x}, t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{x}) \dot{v}_{iI}(t) \quad \forall i = 1, n_{dim} \tag{2.111}$$

On the other hand, the pressure is approximated by

$$p(\mathbf{x}, t) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{x}) p_I(t) \tag{2.112}$$

while its time-independent test function approximation reads

$$\delta p(\mathbf{x}) = \sum_{I=1}^{n_{nodes}} N_I(\mathbf{x}) \delta p_I(t) \tag{2.113}$$

Once the previous space approximations have been defined, one can recover the momentum equation Galerkin weak form in Eq. 2.104 (or 2.103) to write its FE approximation. Considering that the variation δv_{iI} is arbitrary, the following algebraic system of equations arises

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{K}(\mathbf{v})\mathbf{v} - \mathbf{G}\mathbf{p} = \mathbf{f}^{ext} \tag{2.114}$$

being $\mathbf{M}$ the *mass matrix*, $\mathbf{K}(\mathbf{v})$ the *stiffness matrix* obtained as the addition of the convective ($\mathbf{K}^c$) and viscous ($\mathbf{K}^\nu$) stiffness matrices, $\mathbf{G}$ the *pressure matrix* and $\mathbf{f}^{ext}$ the *volumetric external force vector*. The coefficients of such arrays are obtained as

$$M_{ijIJ} = \delta_{ij} \int_\Omega N_I N_J d\Omega \tag{2.115a}$$

$$K^c_{ijIJ} = \delta_{ij} \int_\Omega N_i \mathbf{v}_h \frac{\partial N_J}{\partial x_j} d\Omega \tag{2.115b}$$

$$K^\nu_{ijIJ} = \delta_{ij} \nu \int_\Omega \frac{\partial N_I}{\partial x_j} \frac{\partial N_J}{\partial x_j} d\Omega \tag{2.115c}$$

$$G_{iIJ} = \int_\Omega \frac{\partial N_I}{\partial x_i} N_j d\Omega \tag{2.115d}$$

$$f^{ext}_{iI} = \int_\Omega b_i N_I d\Omega \tag{2.115e}$$

On the other hand, the mass conservation equation Galerkin weak form in Eq. 2.108 (or Eq. 2.107) can be also discretized in a similar fashion, yielding the next algebraic system of equations

$$\mathbf{G}^T \mathbf{v} = 0 \tag{2.116}$$

where $\mathbf{G}^T$ is the so called *divergence matrix*, which is in fact the transpose of the pressure matrix. The coefficients of the divergence matrix can be obtained as

$$G^T_{jIJ} = \int_\Omega N_I \frac{\partial N_J}{\partial x_j} d\Omega \tag{2.117}$$

Once arrived to this point, it is interesting to point out that the same index notation as the one used in the solid mechanics problem discretization have been used. Therefore, minor indices i and j refer to the problem dimensions and run from 1 to n_{dim} meanwhile capital indices I and J refer to the element nodes and run from 1 to n_{nodes}.

Finally, the coupled Navier-Stokes equations algebraic system to be solved is

$$\begin{cases} \mathbf{M}\dot{\mathbf{v}} + \mathbf{K}(\mathbf{v})\mathbf{v} - \mathbf{G}\mathbf{p} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v} = 0 \end{cases} \tag{2.118}$$

Note that the previous system is the one solved if a monolithic resolution scheme is selected.

LBB-condition fulfilment and common interpolation pairs

Once arrived to this point, it is a must to spend a few words regarding the most common velocity and pressure interpolation pairs used in the mixed FE discretization of the Navier-Stokes equations. As was commented before, the incompressibility condition $\nabla \cdot \mathbf{v} = 0$ in the Navier-Stokes equations introduces a compatibility requirement between the pressure and the velocity interpolations. This is the aforementioned LBB-condition or inf-sup condition.

Thus, in order to have $\mathbf{v}$ and p uniquely determined, the pressure interpolation space $\mathcal{Q}^h$ and the velocity interpolation space[2] $\mathcal{V}^h$ must satisfy that $dim\mathcal{Q}^h \leq dim\mathcal{V}^h$. However, this is a necessary but not sufficient condition which can be interpreted as the larger the velocity space the higher the probability of the interpolation pairs to be compatible.

The sufficient condition that ensures that both $\mathbf{v}$ and p are uniquely determined is the LBB-condition. The LBB-condition states that

$$\inf_{\delta p^h \in \mathcal{Q}^h} \sup_{\delta v^h \in \mathcal{V}^h} \frac{(\delta p^h, \nabla \cdot \delta v^h)}{\|\delta p\|_0 \|\delta v^h\|_1} \geq \alpha > 0 \tag{2.119}$$

where α is independent of the mesh size. Once again the reader is referred to [17] or [38] for a much more detailed explanation about the LBB-condition fulfilment.

In the four figures below, which depict some common mixed FE interpolation pairs, symbols $\bullet$ represent velocity interpolation points while symbols $\square$ represent pressure interpolation points. Figures 2.3 and 2.4 depict some interpolation pairs that have continuous pressure approximation. However, the approximations in figure 2.3 do not satisfy the inf-sup condition. On the other hand, figures 2.5 and 2.6 represent interpo-

[2]These interpolation spaces are the space discretized version of the previously defined spaces in Eq. 2.99 and 2.106

lation pairs with discontinuous pressure approximation. In these cases, the pairs that satisfy the inf-sup condition are collected in figure 2.6. Hence, it can be noticed that the satisfaction of the LBB-condition disregards the continuity of the pressure field but is related with the difference in the interpolation orders of the velocity and pressure fields.

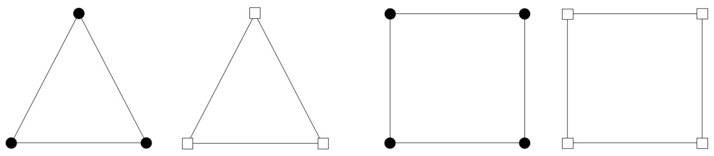


Figure 2.3: Continuous pressure interpolation pairs which do not satisfy the inf-sup condition. Image taken from [38].

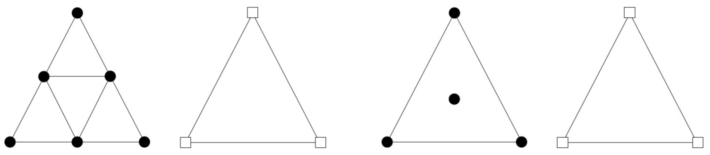


Figure 2.4: Continuous pressure interpolation pairs which do satisfy the inf-sup condition. Image taken from [38].

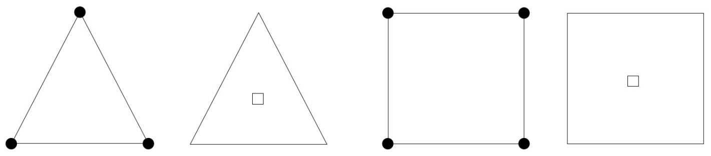


Figure 2.5: Discontinuous pressure interpolation pairs which do not satisfy the inf-sup condition. Image taken from [38].

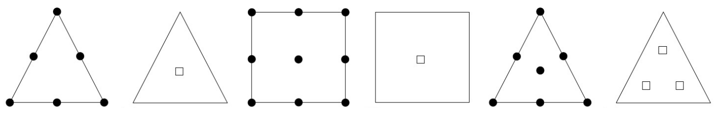


Figure 2.6: Discontinuous pressure interpolation pairs which do satisfy the inf-sup condition. Image taken from [38].

2.3.4 Time discretization schemes

In this subsection the time discretization ideas presented for the structural mechanics case are followed. Thus, only implicit schemes are considered for the fluid dynamics time discretization. Recovering Eq. 2.118 it can be noticed that the equations to be solved at each time step t_{n+1} are

$$\begin{cases} \mathbf{M}\dot{\mathbf{v}}_{n+1} + \mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.120}$$

Among the many time discretization schemes, the *θ-family method* and the *Backward Differentiation method* (BDF) are presented for the fluid dynamics case. Besides, the *Bossak scheme* and *Generalised-α scheme* are recovered from the structural mechanics problem and applied to the fluid dynamics one. More information regarding the classical time discretization schemes can be found in [17].

θ-family method

Assuming that at time t_n the solution fields $\mathbf{v}_n$ and p_n are known, the time derivative of the velocity can be linearly interpolated as

$$\dot{\mathbf{v}}_{n+\theta} = (1-\theta)\dot{\mathbf{v}}_n + \theta\dot{\mathbf{v}}_{n+1} \cong \frac{\mathbf{v}_{n+1} - \mathbf{v}_n}{\Delta t} \tag{2.121}$$

being $\Delta t = t_{n+1} - t_n$. Similarly, the velocity field can be be approximated as

$$\mathbf{v}_{n+\theta} = (1-\theta)\mathbf{v}_n + \theta\mathbf{v}_{n+1} \tag{2.122}$$

Substituting the previous acceleration and velocity interpolations in Eq. 2.121 and 2.122 into 2.120 one obtains the next set of discrete equations

$$\begin{cases} \mathbf{M}\frac{1}{\Delta t}(\mathbf{v}_{n+1} - \mathbf{v}_n) + \mathbf{K}(\mathbf{v}_{n+\theta})\mathbf{v}_{n+\theta} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+\theta} = 0 \end{cases} \tag{2.123}$$

The scheme unconditionally stable for $1/2 \leq \theta \leq 1$. Taking $\theta = 1$ the scheme is first-order accurate and is known as the *Backward Euler method.* On the other hand, taking $\theta = 1/2$ yields a second-order accurate scheme, known as the *Crank-Nicolson* method. Only for $\theta = 1/2$ the θ-methods are second order accurate.

Backward Differentiation method

Backward differentiation methods are also known as Gear schemes. The Backward differentiation methods are classified according to the approximation order of the time derivative. Thus, in the 1st order Backward Differentiation method (BDF1) the time derivative of the velocity is approximated as

$$\dot{\mathbf{v}}_{n+1} = \frac{\mathbf{v}_{n+1} - \mathbf{v}_n}{\Delta t} \tag{2.124}$$

and substituted into Eq. 2.120, yielding the next system of equations to be solved at each time step

$$\begin{cases} \mathbf{M}\dfrac{1}{\Delta t}(\mathbf{v}_{n+1} - \mathbf{v}_n) + \mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.125}$$

Note that the BDF1 time discretized system in Eq. 2.125 equals the θ-family time discretized system in Eq. 2.123 when $\theta = 1$.

For second order accuracy, the method is called BDF2. In the BDF2, the velocity time derivative is approximated as

$$\dot{\mathbf{v}}_{n+1} = \frac{3\mathbf{v}_{n+1} - 4\mathbf{v}_n + \mathbf{v}_{n-1}}{2\Delta t} \tag{2.126}$$

Hence, substituting the BDF2 acceleration approximation into Eq. 2.120 yields the system of equations

$$\begin{cases} \mathbf{M}\dfrac{1}{2\Delta t}(3\mathbf{v}_{n+1} - 4\mathbf{v}_n + \mathbf{v}_{n-1}) + \mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.127}$$

Finally, it has to be said that the BDF2 needs more initialization values that just $\mathbf{v}_0$ since $\mathbf{v}_{n-1}$ appears. This problem can be easily overcome using the BDF1 in the first time step and move then to the BDF2 in the second time step.

Bossak method

Taking the discrete system of equations in Eq. 2.118 and applying the Bossak scheme yields the next semi-discrete system to be solved

$$\begin{cases} \mathbf{M}\dot{\mathbf{v}}_{n+1-\alpha_B} + \mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.128}$$

where the time derivative of the velocity $\dot{\mathbf{v}}_{n+1-\alpha_B}$ is defined as

$$\dot{\mathbf{v}}_{n+1-\alpha_B} = (1-\alpha_B)\dot{\mathbf{v}}_{n+1} + \alpha_B\dot{\mathbf{v}}_n \tag{2.129}$$

As was commented in the structural mechanics time discretization section, the Bossak scheme derives from the Newmark method. Thus, the Newmark formulae for the velocity in Eq. 2.76 is recovered. Rearranging terms, the acceleration can be approximated as

$$\dot{\mathbf{v}}_{n+1} = \frac{1}{\Delta t\gamma}\left(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t(1-\gamma)\dot{\mathbf{v}}_n\right) \tag{2.130}$$

Substituting Eq. 2.130 into Eq. 2.129 the acceleration $\dot{\mathbf{v}}_{n+1-\alpha_B}$ can be expressed only in terms of $\mathbf{v}_{n+1}$ and the previous step known values $\mathbf{v}_n$ and $\dot{\mathbf{v}}_n$.
Finally, one can recover the system of equations in Eq. 2.128 and apply the previous relations. This yields the next final system to be solved

$$\begin{cases} \frac{(1-\alpha_B)}{\Delta t\gamma}\mathbf{M}\mathbf{v}_{n+1} + \mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} + \\ \qquad + \frac{(1-\alpha_B)}{\Delta t\gamma}\mathbf{M}\mathbf{v}_n + \left[\frac{(1-\alpha_B)(1-\gamma)}{\gamma} - \alpha_B\right]\mathbf{M}\dot{\mathbf{v}}_n \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.131}$$

Recall that α_B and γ are the time integration parameters of the Bossak method referred in the structural mechanics time discretization section.

Generalised-α method

In this case, the generalised-α method is applied on a first-order system. The application of the generalised-α method to first-order system of equations, can be found in [25]. As was commented in the structural mechanics section, the generalised-α method has high-frequency dissipation while the low-frequency dissipation is minimized, something desirable in analysis with long time periods. In the generalised-α method, the semi-discrete equations are

$$\begin{cases} \mathbf{M}\dot{\mathbf{v}}_{n+\alpha_m^f} + \mathbf{K}(\mathbf{v}_{n+\alpha_f^f})\mathbf{v}_{n+\alpha_f^f} - \mathbf{G}\mathbf{p}_{n+1} = \mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}_{n+1} = 0 \end{cases} \tag{2.132}$$

where the previous acceleration and velocities are defined as

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \Delta t(1-\gamma^f)\dot{\mathbf{v}}_n + \Delta t\gamma^f\dot{\mathbf{v}}_{n+1} \tag{2.133}$$

$$\mathbf{v}_{n+\alpha_f^f} = (1-\alpha_f^f)\mathbf{v}_n + \alpha_f^f\mathbf{v}_{n+1} \tag{2.134}$$

$$\dot{\mathbf{v}}_m^f = (1-\alpha_m^f)\dot{\mathbf{v}}_n + \alpha_m^f\dot{\mathbf{v}}_{n+1} \tag{2.135}$$

Isolating $\dot{\mathbf{v}}_{n+1}$ in Eq. 2.133 and substituting it into Eq. 2.135 yields

$$\dot{\mathbf{v}}_{n+\alpha_m^f} = \frac{\alpha_m^f}{\Delta t\gamma^f}(\mathbf{v}_{n+1} - \mathbf{v}_n) + \left(1 - \frac{\alpha_m^f}{\gamma^f}\right)\dot{\mathbf{v}}_n \tag{2.136}$$

Finally, substituting Eq. 2.134 and Eq. 2.135 into 2.132 allow to express the problem in terms of the unknowns $\mathbf{v}_{n+1}$ and p_{n+1}. Optimal values of the integration parameters γ^f, α_m^f and α_f^f for second-order accuracy, high-frequency dissipation and optimal low-frequency dissipation can be found in [44].

2.3.5 Pressure segregation methods

Despite that pressure segregation methods are not used in this work it is interesting to briefly outline them. The aim of pressure segregation methods is to decouple the pressure from the balance of linear momentum equation. Therefore, more than one system of equation arises but with less total computational effort.

Starting from the BDF1 time discretized N-S equations in Eq. 2.125, one can split the linear momentum equation in the equivalent form

$$\begin{cases} \mathbf{M}\frac{1}{\Delta t}(\tilde{\mathbf{v}}_{n+1} - \mathbf{v}_n) + \mathbf{K}(\tilde{\mathbf{v}}_{n+1})\tilde{\mathbf{v}}_{n+1} - \mathbf{G}\mathbf{p}_n = \mathbf{f}^{ext} \\ \mathbf{M}\frac{1}{\Delta t}(\mathbf{v}_{n+1} - \tilde{\mathbf{v}}_{n+1}) - \mathbf{G}(\mathbf{p}_{n+1} - \mathbf{p}_n) = 0\mathbf{G}^T\mathbf{v}_{n+1} \quad = 0 \end{cases} \tag{2.137}$$

considering the auxiliary velocity $\tilde{\mathbf{v}}_{n+1}$ and the approximation

$$\mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1} \approx \mathbf{K}(\tilde{\mathbf{v}}_{n+1})\tilde{\mathbf{v}}_{n+1} \tag{2.138}$$

Rearranging terms, one arrives to

$$\begin{cases} \mathbf{M}\dfrac{1}{\Delta t}(\tilde{\mathbf{v}}_{n+1} - \mathbf{v}_n) + \mathbf{K}(\tilde{\mathbf{v}}_{n+1})\tilde{\mathbf{v}}_{n+1} - \mathbf{G}\mathbf{p}_n = \mathbf{f}^{ext} \\ \qquad\qquad\qquad -\Delta t \mathbf{L}(\mathbf{p}_{n+1} - \mathbf{p}_n) = \mathbf{G}^T(\tilde{\mathbf{v}}_{n+1}) \\ \mathbf{M}\dfrac{1}{\Delta t}(\mathbf{v}_{n+1} - \tilde{\mathbf{v}}_{n+1}) - \mathbf{G}(\mathbf{p}_{n+1} - \mathbf{p}_n) = 0 \end{cases} \tag{2.139}$$

where $\mathbf{L}$ is an approximation of the Laplacian operator [8] and the last equation is in fact an explicit velocity update. This is the so called *Fractional Step scheme*. In [44] the application of the fractional step scheme combined with the BDF2 and the generalised-α method can be found. An stability analysis of the method is carried out in [8].

Finally, it is interesting to pinpoint the *Predictor-Corrector method*. The Predictor-Corrector method is a decoupled iterative scheme which goal is to converge to the monolithic solution. Considering again the BDF1 time discretization method, the Predictor-Corrector scheme reads

$$\begin{cases} \mathbf{M}\dfrac{1}{\Delta t}(\mathbf{v}_{n+1,i} - \mathbf{v}_n) + \mathbf{K}(\mathbf{v}_{n+1,i-1})\mathbf{v}_{n+1,i} - \mathbf{G}\mathbf{p}_{n+1,i-1} = \mathbf{f}^{ext} \\ \qquad\qquad\qquad -\Delta t \mathbf{L}(\mathbf{p}_{n+1,i} - \mathbf{p}_{n+1,i-1}) = \mathbf{G}^T\mathbf{v}_{n+1,i} \end{cases} \tag{2.140}$$

where i holds for the current iteration and $i-1$ for the known previous one. The Predictor-Corrector method is widely developed in [12].

2.3.6 Stabilization techniques

As commented before, two are the main sources of numerical instabilities in the numerical approximation of the Navier-Stokes equations. The former is related with the non-linear convective term, which might generate numerical oscillations in the velocity field when high Reynolds number convection-dominated flows are considered. These oscillations are minimized as the mesh is refined. However, this implies a computational effort increase. Therefore, the use of stabilization techniques is the optimal way to handle these oscillations.

The other instability source is related to the incompressibility condition constraint and the LBB-condition, which requires the velocity and pressure approximation spaces to be compatible in order to avoid spurious pressure modes. Thus, one have two options, either to use interpolation pairs that satisfy the LBB-condition or circumvent it using stabilization techniques, allowing the use of simpler approximations, as the ones presented in figure 2.3. These approximations, which do not satisfy the LBB-condition, have advantages such as continuous pressure interpolation or a simpler implementation, due to the fact that the same interpolation is used for the velocity and pressure fields.

In this work, the *Variational Multiscales method* (VMS) has been used as main stabilization technique. In the next subsection, the VMS method is introduced. However, there are several stabilization techniques that can be used as well. Some examples are the *Steamline-upwind/Petrov-Galerkin method* (SUPG), the *Galerkin Least Squares* (GLS), the *Orthogonal Subscales method* (OSS) or the *Finite Increment Calulus* (FIC), which are briefly outlined as well.

Variational Multiscales method

The starting point of any multiscale method is the decomposition of the unknowns. Hence the velocity $\mathbf{v}$ and the pressure p are decomposed as

$$\mathbf{v} = \mathbf{v}_h + \tilde{\mathbf{v}}, \quad p = p_h + \tilde{p} \tag{2.141}$$

where $\mathbf{v}_h$ and p_h belong to the finite element approximation spaces $\mathcal{V}^h$ and $\mathcal{Q}^h$ and $\tilde{\mathbf{u}}$ and $\tilde{p}$ are the *subscales.* The FE components of the decomposition $\mathbf{v}_h$ and p_h are the so called *resolved scales.* Conversely, the subscales are the *unresolved* scales, which introduce an artificial diffusion that stabilizes the formulation.

For the better comprehension of the VMS method, it is quite useful to analyse are the implications of introducing the decomposition in Eq. 2.141 into the non-linear convective term of the Navier-Stokes equations. The complete weak formulation of the N-S equations considering the subscale can be found in [10]. Focusing only in the non-linear term, it can be rewritten as

$$\nabla \cdot (\mathbf{v} \otimes \mathbf{v}) = \nabla \cdot (\mathbf{v}_h \otimes \mathbf{v}_h) + \nabla \cdot (\mathbf{v}_h \otimes \tilde{\mathbf{v}}) + \nabla \cdot (\tilde{\mathbf{v}} \otimes \mathbf{v}_h) + \nabla \cdot (\tilde{\mathbf{v}} \otimes \tilde{\mathbf{v}}) \tag{2.142}$$

where

- $\nabla \cdot (\mathbf{v}_h \otimes \mathbf{v}_h)$ is the term appearing in a standard Galerkin approximation.

- $\nabla\cdot(\mathbf{v}_h\otimes\tilde{\mathbf{v}})$ introduces the numerical stability that controls the convective derivative and pressure gradient.
- $\nabla\cdot(\tilde{\mathbf{v}}\otimes\mathbf{v}_h)$ drives to global momentum equilibrium.
- $\nabla\cdot(\tilde{\mathbf{v}}\otimes\tilde{\mathbf{v}})$ have barely difference with similar terms in LES turbulence models. As a consequence, the question of whether this term is acting as a Large Eddy Simulation (LES) turbulence model arises [10][22].

Regarding the velocity time derivative in the N-S equations, it turns to be $\partial_t\mathbf{v} = \partial_t\mathbf{v}_h + \partial_t\tilde{\mathbf{v}}$ when the decomposition in Eq. 2.141 is considered. Then, one can keep the subscales time derivative (dynamic subscales) or neglect it (quasi-static subscales). It is proved that dynamic subscales yield the correct long term stability of the space-discrete scheme [11].

On the other hand, the selection of the subscales spaces is also crucial. For instance, in the *Orthogonal Subscales* (OSS) method, the space of the subscales is selected to be orthogonal to the FE space, what is to say that $\tilde{\mathcal{V}} \in \mathcal{V}_h^{\perp}$. This has implications in the kinetic energy balance separation between scales. The foundations of the OSS method can be found in [9] and [10].

The main reference in this subsection has been [11], which plainly introduces the reader to the VMS methods formulation and summarizes years of research on the OSS method. For further details on the VMS stabilization technique, the reference papers are [19] and [20].

Other stabilization techniques

This subsection briefly outlines some of the most common stabilization alternatives to the VMS method. One stabilization technique that has been widely used in convection-dominated problems is the *streamline-upwind/Petrov-Galerkin method* (SUPG). This method consists in introducing a numerical diffusion but only along the streamlines. As is depicted in [5], the method can be applied to the residual form of the Navier-Stokes equations.

A more general stabilization approach is the *Galerkin Least Squares method* (GLS). In the GLS, the stabilization terms are obtained minimizing the sum of the square residual of the momentum equation. However, it has to be taken into account that the GLS requires FE discretization for both space and time. A detailed explanation on the GLS method can be found in [21].

So far, all the presented stabilization methods require the introduction of artificial diffusion terms. This is not the case in the *Finite Increment Calculus method* (FIC), in where the stabilization terms are considered as a natural and intrinsic contribution to the original differential equations, meaning that the terms obtained can be interpreted in a more physical manner. The FIC method main reference is [35].

The formulation and implementation in FSI problems of all the methods outlined in this subsection can be found in [44].

2.3.7 Problem resolution: linearization techniques

Once the Navier-Stokes equations have been discretized, the problem to be solved is a non-linear system of equations. In this case, the non-linearity is located in the convective term $\mathbf{K}(\mathbf{v}_{n+1})\mathbf{v}_{n+1}$.

The presence of the non-linearity implies the use of an iterative scheme. Considering that i is the sought iteration and $i-1$ is a the previous known one, the convective term can be linearised as

- $\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^{i-1}_{n+1}$: there exists a solution but convergence is very slow and only for low Reynolds numbers.
- $\mathbf{K}(\mathbf{v}^{i}_{n+1})\mathbf{v}^{i-1}_{n+1}$: there might not be stable solution as the element size decreases (the linear form associated to the problem is not coercive).
- $\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^{i}_{n+1}$: the iteration is said to be of *Picard's type* (fixed point method).

Among the previous three presented linearization techniques, the most useful and extended one is the Picard's method. Lets consider again the BDF1 discretization in Eq. 2.125 to show the Picard's strategy, which reads as follows

$$\begin{cases} \mathbf{M}\dfrac{1}{\Delta t}(\mathbf{v}^{i}_{n+1}-\mathbf{v}_n)+\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^{i}_{n+1}-\mathbf{G}\mathbf{p}^{i}_{n+1}=\mathbf{f}^{ext} \\ \mathbf{G}^T\mathbf{v}^{i}_{n+1}=0 \end{cases} \tag{2.143}$$

Despite the Picard's iteration can achieve convergence in the majority of cases, its convergence rate is linear. Thus, it is convenient to take advantage of the Newton-Raphson's method, which has quadratic convergence. Considering once again BDF1 as discretization scheme, the Newton-Raphson's linearization reads as follows

$$\begin{cases} \mathbf{M}\dfrac{1}{\Delta t}(\mathbf{v}^i_{n+1}-\mathbf{v}_n)+\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^i_{n+1}+\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^{i-1}_{n+1}-\mathbf{G}\mathbf{p}^i_{n+1}= \\ \qquad\qquad\qquad\qquad\qquad\qquad =\mathbf{f}^{ext}+\mathbf{K}(\mathbf{v}^{i-1}_{n+1})\mathbf{v}^{i-1}_{n+1} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{G}^T\mathbf{v}^i_{n+1}=0 \end{cases} \tag{2.144}$$

2.4 Fluid structure interaction

This section reviews the main aspects regarding the coupling and resolution of the fluid-structure interaction problem. In this context, the *Arbitrary Lagrangian-Eulerian* (ALE) form of the Navier-Stokes equations is firstly presented. Then, the FSI problem is stated as well as the existing approaches to solve it. Finally, some of the common issues involved in the FSI problem resolution are pointed out.

2.4.1 ALE framework in computational fluid dynamics

As has been commented before, in CFD analysis the Navier-Stokes equations are usually solved in their Eulerian form, since it avoids the extremely large element distortion associated to the flow convection. When solving an FSI problem, one has to take into account that the skin of the structure, henceforth referred as *solid interface*, would have a displacement due to the fluid loads. Considering, that no penetration of the fluid into the body is allowed, such structure displacement is translated to a displacement of the *fluid interface*.

In those cases in where the structural displacements are expected to be infinitesimal, their affectation to the mesh can be neglected, meaning that the FSI problem can be solved keeping the Eulerian formulation in the CFD part. On the contrary, when the structural displacements are finite, the Eulerian formulation turns to be a poor approximation and the necessity of other techniques arises.

Consequently, the ALE framework has been widely used in the FSI context when the large displacements assumption is taken in the structure. Despite the fact that in this work the ALE method has been selected, there exist several alternative techniques that can be also successfully applied. Among them, the novel embedded formulation [1] has to be mentioned.

The main feature of the ALE framework is the apparition of a new body configuration, the so called *ALE configuration*, which domain is $\hat{\Omega}$. The coordinates of such domain $\hat{\Omega}$

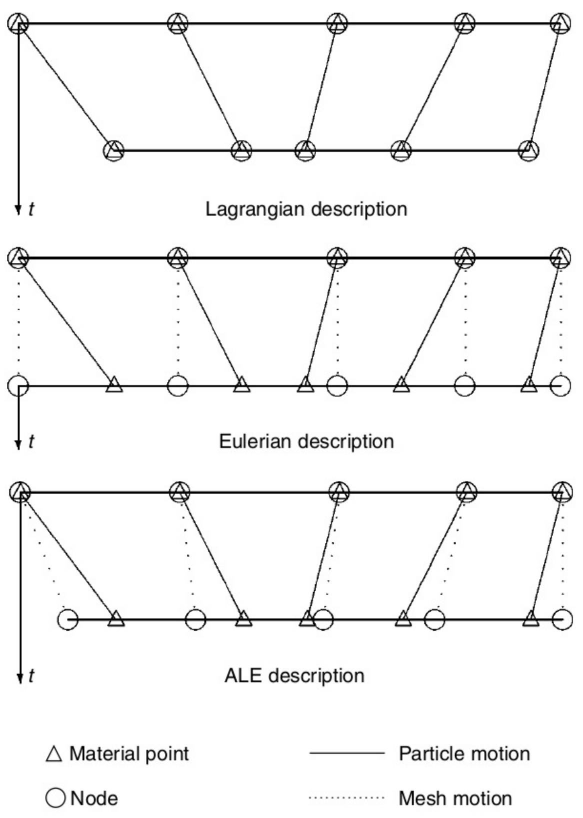


Figure 2.7: Lagrangian, Eulerian and ALE frameworks comparison. Image taken from [17].

are the *ALE coordinates* and are denoted by χ. As a consequence, two different motions appear in ALE methods: *material motion* and *mesh motion* (figure 2.7). Material motion is described as in Eq. 2.2 while the mesh motion is described as

$$\mathbf{x} = \hat{\mathbf{\Phi}}(\boldsymbol{\chi}, t) \tag{2.145}$$

By means of the previous equation, the *mesh displacement* can be obtained as

$$\hat{\mathbf{u}} = \mathbf{x} - \boldsymbol{\chi} = \hat{\mathbf{\Phi}}(\boldsymbol{\chi}, t) - \boldsymbol{\chi} \tag{2.146}$$

The *mesh velocity* and the *mesh acceleration* can be obtained as the first and second time derivative of Eq. 2.146. Besides this, the material time derivative is also rewritten in ALE framework as

$$\frac{D(\bullet)}{Dt} = \frac{\partial(\bullet)}{\partial t} + \mathbf{c} \cdot \nabla(\bullet) \tag{2.147}$$

where $\mathbf{c}$ is the *convective velocity* obtained as the difference between the material and the mesh velocities $\mathbf{c} = \mathbf{v} - \hat{\mathbf{v}} = \mathbf{v} - \mathbf{v}_{mesh}$. Once the convective velocity has been defined, the balance of linear momentum equation in ALE framework can be stated as

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{c} \cdot \nabla \mathbf{v} \right) = \nabla \cdot \sigma + \rho \mathbf{b} \tag{2.148}$$

For the sake of simplicity, only the essentials of the ALE formulation have been outlined in this section. A much further description of the method can be found in [3] or [17].

2.4.2 Problem formulation and transmission conditions

This subsection recalls the concepts stated in chapters 2.2 and 2.3 to state the FSI problem. First of all, let us divide the problem domain in two disjoint domains such that $\Omega = \Omega_f \cup \Omega_s$ being Ω_f the fluid domain and Ω_s the solid one. The boundary of the problem domain can be also separated in $\Gamma = \Gamma_f \cup \Gamma_s \cup \Gamma_{int}$ being Γ_{int} the interface between fluid and structure. As usual, in both solid and fluid domains the boundary can be split in Dirichlet (Γ^D) and Neumann (Γ^N) boundaries. Figure 2.8 schematically depicts the previous domains and boundaries.

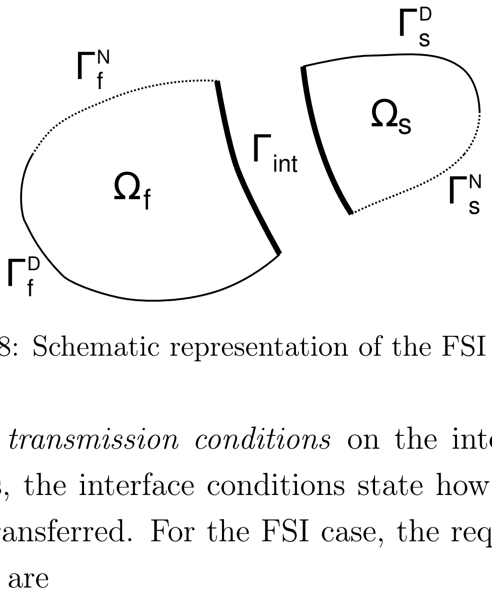


Figure 2.8: Schematic representation of the FSI problem.

On the other hand, the *transmission conditions* on the interface boundary must be defined. In general terms, the interface conditions state how the information between problem subdomains is transferred. For the FSI case, the required transmission conditions at the interface Γ_{int} are

- continuity of displacements: $\mathbf{u}_s = \mathbf{u}_f$ *on* Γ_{int}
- continuity of tractions: $\mathbf{n} \cdot \boldsymbol{\sigma}_s = \mathbf{n} \cdot \boldsymbol{\sigma}_f$ *on* Γ_{int}

Note that the continuity of displacements can be formulated equivalently in terms of the velocity. Moreover, it is interesting to point out that all the components of the displacement must be continuous at the interface since the viscous flow consideration is assumed. On the contrary, when the inviscid flow assumption is considered, only the normal component needs to be continuous, due to the slip nature of the interface.

Recovering the fluid dynamics governing equations in ALE form as well as the solid mechanics governing equations, both together with their respective boundary and initial conditions, as well as the transmission conditions above one can formulate the complete FSI problem as

$$\left\{\begin{aligned}
\frac{\partial \mathbf{v}_f}{\partial t} + (\mathbf{v}_f - \mathbf{v}_{mesh}) \cdot \nabla \mathbf{v}_f - \nu_f \Delta \mathbf{v}_f + \nabla p = \mathbf{b}_s \quad & \textit{in } \Omega_f \\
\nabla \cdot \mathbf{v}_f = 0 \quad & \textit{in } \Omega_f \\
\rho_{0,s} \frac{\partial \mathbf{v}_s}{\partial t} = \nabla_0 \cdot \mathbf{P} + \rho_{0,s} \mathbf{b}_s \quad & \textit{in } \Omega_s \\
\mathbf{v}_f(\mathbf{x}, t) = \phi(\mathbf{x}, t) \quad & \textit{on } \Gamma_f^D \\
\left(\nu_f \frac{\partial \mathbf{v}_f}{\partial \mathbf{n}} - p\mathbf{n}\right)(\mathbf{x}, t) = \psi(\mathbf{x}, t) \quad & \textit{on } \Gamma_f^N \\
\mathbf{u}_s(\mathbf{x}, t) = \bar{u}_s(\mathbf{x}, t) \quad & \textit{on } \Gamma_s^D \\
\mathbf{t}_s(\mathbf{x}, t) = \bar{\mathbf{t}}_s(\mathbf{x}, t) \quad & \textit{on } \Gamma_s^N \\
\mathbf{u}_f(\mathbf{x}, t) = \mathbf{u}_f(\mathbf{x}, t) \quad & \textit{on } \Gamma_{int} \\
\mathbf{t}_s(\mathbf{x}, t) = \mathbf{t}_f(\mathbf{x}, t) \quad & \textit{on } \Gamma_{int} \\
\mathbf{v}_f(\mathbf{x}, 0) = \mathbf{v}_f^0(\mathbf{x}) \quad & \textit{in } \Omega_f \textit{ at } t = 0 \\
\mathbf{u}_s(\mathbf{x}, 0) = \bar{\mathbf{u}}_s^0(\mathbf{x}) \quad & \textit{in } \Omega_s \textit{ at } t = 0
\end{aligned}\right. \tag{2.149}$$

Last but not least, it has to be said that considering an ALE framework in the fluid domain implies the apparition of the *mesh movement problem*. The particularities of this problem are addressed in chapter 2.4.5.

2.4.3 FSI approaches

The existent approaches to solve the FSI problem can be roughly divided in two types: *monolithic approaches* and *partitioned approaches*. The monolithic approach consists in solving all the equations of the FSI problem (Eq. 2.149) in a unified system while the partitioned approaches use independent solvers for each solid and fluid fields.

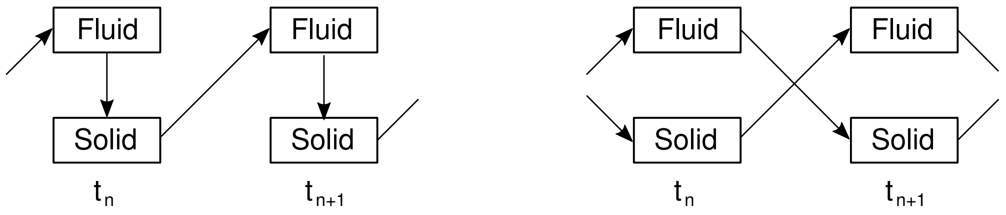
Monolithic approaches

Monolithic approaches have the advantage of larger stability and robustness, meaning that convergence is usually optimal. Due to this reason, the trend in the literature is to use this kind of approaches in complex coupling situations such as hemodynamics [4]. However, monolithic schemes lead to large system of equations, which turn to be extremely expensive in terms of the computational cost. Besides, the development of a monolithic solver requires lots of efforts since both the existing solid and fluid solvers require major modifications to perform in a monolithic manner [41].

Partitioned approaches

On the other hand, partitioned approaches keep a separate field solver for the fluid and the solid problems. This has the great advantage of code reusability, meaning that optimal and widely tested, or even commercial, solvers can work as black boxes and the unique thing that is left to be done is the coupling algorithm between them.

According to the nature of such coupling algorithm, one can distinguish two types of partitioned approaches: explicit or implicit partitioned schemes. In the *explicit partitioned schemes*, also known as *staggered approaches* or *weakly coupled partitioned schemes*, the information interchange between fields is transferred at the end of the time step (figure 2.9). As a consequence, the computational cost is very cheap but the interface convergence rates are poor, even with the use of predictors.



(a) Serial approach. (b) Parallel approach.

Figure 2.9: Loosely coupled partitioned schemes overview.

On the contrary, *implicit partitioned schemes*, also known as *strongly coupled partitioned schemes*, are intended to be an intermediate solution between the monolithic approaches and the fully explicit ones [41]. In this case, the coupling algorithm between the fluid and solid solvers implies the iteration until a convergence criterion is reached (figure 2.10).

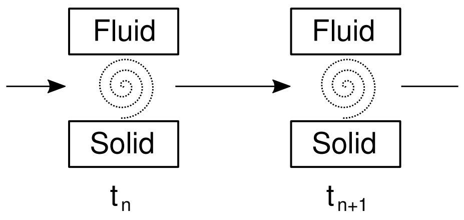


Figure 2.10: Strongly coupled partitioned scheme overview.

Despite the fact that have much better convergence ratios than the loosely coupled methods, the strongly coupled schemes may diverge in some cases depending on the strategy used. Furthermore, these methods have the disadvantage of its computational cost, which in some cases can be larger than the monolithic scheme one. Such large computational cost comes from the fact that it is needed to solve the fluid and structure problems at each iteration of the coupling scheme.

2.4.4 The added mass effect

This subsection overviews the added mass effect problem. The main reference has been [23], where a further description of the added mass effect, the foundation and application of the techniques to deal with it and several numerical examples can be found.

The added mass effect is an issue associated to the incompressible flow assumption. In a rigorous way, the *added mass effect* is defined as the addition of virtual mass to the coupled system because of the movement of a body surrounded by an incompressible fluid. Such incompressible flow assumption is the source of the added mass effect problem, since the inertial forces associated to the accelerating or decelerating body are translated to a wider region of the fluid domain. This can be viewed as an extra or virtual mass which was not present in the original system.

However, the term added mass effect has been widely used in the literature to name those instability issues that tipically occur in FSI problems combining incompressible fluids and structures with similar densities (e.g. hemodynamics problems). As pointed in [23], another particular source of added mass effects is the use of pressure segregation methods within a monolithic scheme.

As can be seen in figure 2.11, the added mass effect affects to all the FSI approaches that have been defined in the previous section. Thus, when loosely coupled partitioned approaches are used, the system may become unstable when the added mass effect is significant. In the strongly coupled partitioned approaches case, the convergence might be ruined by the added mass effect. Finally, in monolithic schemes the system might become ill-conditioned or it could not converge if pressure segregation methods are used.

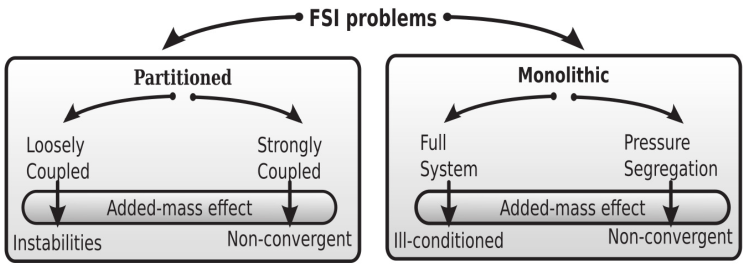


Figure 2.11: Added mass effect affectation scheme according to FSI procedures. Image taken from [23].

2.4.5 Fluid ALE mesh particularities

Mesh movement problem

As has been pointed before, in an ALE framework the solid interface movement implies to update the fluid mesh after each time iteration. This update consists in solving an extra problem, which varies according to the mesh updating techniques. Despite there are several techniques to update the mesh, all of them share the same boundary conditions at the interface

$$\begin{aligned} \mathbf{u}_{mesh}(\mathbf{x},t) &= \mathbf{u}_s(\mathbf{x},t) \qquad && on\ \Gamma_{int} \\ \mathbf{u}_{mesh}(\mathbf{x},t) &= 0 && on\ \Gamma = \Gamma_f^D \cup \Gamma_f^N \end{aligned} \tag{2.150}$$

As can be seen, the first boundary condition in Eq. 2.150 states that the mesh is attached to the fluid-structure interface. On the other hand, the second one, which holds for the rest of the fluid domain boundary, states null movement at the external boundaries. It has to be said that in the external boundaries the mesh movement only needs to be null in the normal direction, meaning that the mesh nodes can vary their position within the boundary plane if it is needed.

On the other hand, it is crucial that the fluid mesh update accommodates the interface changes in a reliable way. How the mesh can be best updated depends on several factors, such as the interface and overall geometry complexity, how severe the interface movement is, and how the initial mesh was generated. In general, the mesh update could have two components: moving the mesh as long as it is possible and remeshing (i.e., generating fully or partially a new set of nodes and elements) when the element distortion becomes too high [2].

It can be guessed that remeshing must be avoided as much as possible due to its high computational cost, which becomes even larger in 3D cases. Thus, robust and efficient mesh movement techniques are essential in FSI problems that consider ALE formulation in the fluid domain.

Among the many mesh updating techniques, a specially interesting one is the *stiffness* or *structural methods*, in which a structural problem is solved considering as interface boundary conditions the solid interface movement. Thus, a stiffness matrix is generated from the specific mesh Lamé coefficients. The stiffness methods are particularly interesting in FSI applications, since the smaller the element is, the larger is its stiffness. Since the finest elements are usually placed near the solid domain, this avoids the element distortion in the interface surroundings. A deeper explanation about the stiffness mesh updating technique, as well as the problem equations and expressions of the mesh Lamé parameters can be found in [2]. Complementary, [42] contains a review on the main features of these methods.

Another family of mesh updating techniques is the so called *elliptic methods* (also known as Laplacian or Poisson methods), which formulation is detailed in [41]. These methods consists in mapping between the computational space and the physical domain through a parameter space, that can be used to control the quality of the mesh. Then, an elliptic equation is solved for each displacement component until a convergence is satisfied.

Finally, it has to be said that there exist other typologies of mesh updating techniques, such as the transfinite mapping method or simple interpolations. A general overview on the existing mesh updating techniques can be found in [17].

Mapping strategies

The use of partitioned approaches in the resolution of the FSI problem requires the proper transference of information between subdomains to guarantee the convergence. There is no problem when interface matching meshes are used, since the information can be directly transferred node-by-node. Unfortunately, this might not be the case due to the need of different mesh resolutions in the fluid and solid domains. Therefore, a mapping technique is required in order to apply the interface boundary conditions.

According to [45], consistency, which specifies that a constant field should be mapped exactly, is the basic criterion for mapping algorithms. Another criterion is the conservation of energy, which helps to derive special mapping operators for traction and force. Thus, the direct use of mapping algorithms is called *consistent mapping*, while the use of mapping techniques derived from the energy conservation is called *conservative mapping*.

There are several mapping strategies that can be used, from the simpler *Nearest Neighbour interpolation* to the use of *mortar elements*. In the following lines, some of the most popular mapping techniques are outlined.

The *Nearest Neighbour interpolation* is probably the simplest method of transferring information between meshes. It is based in a search algorithm so that the closest node to an origin mesh node is found in the destiny mesh. Once the closest destiny node has been found for all origin nodes, the values are directly transferred without any interpolation. This search procedure is likely expensive so it is advised to optimize it using *tree data structures* such as the kd-tree.

The second mapping techniques to be commented are the *Point-to-Element projection schemes*. This family contains the Node-Projection scheme and the Quadrature-Projection scheme. In the Node-Projection scheme, the origin nodes are directly projected towards the destiny interface. On the other hand, in the Quadrature-Projection scheme, an auxiliary quadrature is generated in the origin interface and such quadrature points are the ones to be projected onto de destiny interface. Once the projections have been carried out, the values are simply interpolated.

Another mapping technique is the *Common-Refinement based scheme* (C-R). In the C-R scheme the position of the fluid interface nodes is projected onto the solid interface. Then an auxiliary mesh, which elements are in between the solid nodes and the projected fluid nodes, is generated. Once the auxiliary mesh has been obtained, the mapped values can be computed.

A completely different family of methods are the so called *mortar elements*. The standard mortar method applies the Galerkin approach which can minimize the L_2 norm of the deviation between two fields. In [45] the application and performance assessment of mortar and dual mortar methods in the FSI context can be found.

For further information, the reader is aimed to review [24] and [45]. Moreover, the application of some of the commented interpolation techniques to problems similar to the ones presented in this work can be found in [27].

Chapter 3

Methodology

This chapter collects the main aspects of the FSI procedures implemented in this work. First of all, it is important to clearly state that the priority has been to develop a FSI framework with high code reusability capabilities. As a consequence, monolithic coupling approaches, which require the implementation of a new whole solver, were completely rejected since the very beginning, driving towards the use of partitioned coupling schemes.

In the partitioned coupling schemes a specific solver is used for each one of the coupled problems. One step further is to consider these solvers as *black boxes* that are able to take an input information to give back an output data, without the necessity of any access to their interior procedures.

This black boxes idea is the key concept of the coupling approaches that are to be presented, since it allows the junction of different widely tested and optimized codes for each one of the problem fields. Besides, the development of the coupled solver is cheaper in economic and time consumption terms due to the possibility of code reusability. Is for these reasons that partitioned approaches have become more and more popular in the industry field during the last times. The black box coupling concept is further detailed in [4]. A general review on algorithms for strong coupling procedures can also be found in [28].

In this chapter, the Dirichlet-Neumann and Neumann-Neumann strongly coupled partitioned schemes are firstly presented and its residual form is developed. Then, the implemented iterative techniques to minimize such interface residual are discussed.

3.1 Partitioned strongly coupled approaches developed

In this section the partitioned strongly coupled approaches that have been used all along this work are presented. On one hand, the *Dirichlet-Neumann scheme* (D-N) is discussed. On the other hand, the *Neumann-Neumann* (N-N) scheme is also developed.

3.1.1 Dirichlet-Neumann iterative scheme

The D-N scheme steps can be roughly divided in

1. Solve the fluid problem from a suitable prediction of the interface velocity as interface Dirichlet B.C.

2. Extract the interface fluid fluxes

3. Apply the interface fluid fluxes as a Neumann B.C. on the solid interface

4. Solve the solid problem to obtain the resultant step interface displacement

5. Extract the solid interface velocity as next fluid interface velocity prediction

It has to be said, that the D-N scheme can be solved in the other way around, that is to say, that Neumann B.C.'s are applied on the fluid interface and Dirichlet B.C.'s are applied on the solid problem. This is suitable in those cases in where the fluid density ρ_f is much larger than the solid density ρ_s. However, this is not the usual case in FSI problems, where the solid density is normally several orders of magnitude larger than the fluid one.

For the sake of simplicity, the solid problem previously stated in 2.73 as well as the fluid problem stated in 2.118 can be firstly expressed in compact form according to their interior and interface contributions. Thus, the compact version of the solid problem reads

$$\begin{bmatrix} \mathbf{S}_{II} & \mathbf{S}_{I\Gamma} \\ \mathbf{S}_{\Gamma I} & \mathbf{S}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{u}_I \\ \mathbf{u}_\Gamma \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^S \\ \mathbf{f}_\Gamma^S \end{bmatrix} \tag{3.1}$$

where $\mathbf{u}_\Gamma$ are the solid interface DOF's and $\mathbf{u}_I$ the rest of solid DOF's. On the other hand, the compact version of the fluid problem is

$$\begin{bmatrix} \mathbf{F}_{II} & \mathbf{F}_{I\Gamma} \\ \mathbf{F}_{\Gamma I} & \mathbf{F}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{v}_I \\ \mathbf{v}_\Gamma \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^F \\ \mathbf{f}_\Gamma^F \end{bmatrix} \tag{3.2}$$

where $\mathbf{v}_\Gamma$ are the fluid interface DOF's and $\mathbf{v}_I$ the rest of fluid DOF's. It is important to clearly state that in Eq. 3.2 the degrees of freedom $\mathbf{v}_I$ and $\mathbf{v}_\Gamma$ are considered as the whole unknowns of the problem, meaning that both the fluid velocity and pressure unknowns are compacted as interior DOF's and interface DOF's.

Taking the compact form of the fluid problem in Eq. 3.2 at time step $n+1$, the interior solution field $\mathbf{v}_I^{n+1}$ can be computed from a suitable interface velocity prediction $\mathbf{v}_\Gamma^\star$ as

$$\mathbf{F}_{II}\mathbf{v}_I^{n+1} = \mathbf{f}_I^{F,n+1} - \mathbf{F}_{I\Gamma}\mathbf{v}_\Gamma^\star \tag{3.3}$$

Then, the interface fluxes are directly obtained as

$$\mathbf{f}_\Gamma^{F,n+1} = \mathbf{F}_{\Gamma I}\mathbf{v}_I^{n+1} + \mathbf{F}_{\Gamma\Gamma}\mathbf{v}_\Gamma^\star \tag{3.4}$$

Note that the fluid problem can be viewed as an operator such that, given a fluid interface velocity prediction $\mathbf{v}_\Gamma^\star$, gives back a fluid interface flux $\mathbf{f}_\Gamma^{F,n+1}$. Henceforth, let us rename the D-N fluid problem as the operator $\mathbf{f}_\Gamma^{n+1} = \mathsf{F}(\mathbf{v}_\Gamma^\star)$.

On the other hand, the solid problem can be stated in terms of the fluid interface nodal fluxes $\mathbf{f}_\Gamma^{F,n+1}$. Recovering the compact form of the solid problem in Eq. 3.1 as well as the obtained $\mathbf{f}_\Gamma^{F,n+1}$ interface fluxes from the fluid problem, the solid problem reads

$$\begin{bmatrix} \mathbf{S}_{II} & \mathbf{S}_{I\Gamma} \\ \mathbf{S}_{\Gamma I} & \mathbf{S}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{u}_I^{n+1} \\ \tilde{\mathbf{u}}_\Gamma^{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^{S,n+1} \\ \mathbf{f}_\Gamma^{S,n+1} - \mathbf{f}_\Gamma^{F,n+1} \end{bmatrix} \tag{3.5}$$

Note that the fluid interface fluxes $\mathbf{f}_\Gamma^{F,n+1}$ are subtracted to the RHS due to the interface equilibrium requirement. As in the fluid problem case, the solid problem can be viewed as an operator such that, given an interface flux $\mathbf{f}_\Gamma^{n+1}$, turns an interface displacement $\tilde{\mathbf{u}}_\Gamma^{n+1}$. Henceforth, let us rename the D-N solid problem as the operator $\tilde{\mathbf{u}}_\Gamma^{n+1} = \mathsf{S}^{-1}(\mathbf{f}_\Gamma^{n+1})$.

Once arrived to this point, one can state the D-N algorithm via the previously defined fluid and solid problem operators. Therefore, the D-N coupled problem can be expressed as

$$\tilde{\mathbf{u}}_\Gamma^{n+1} = \mathsf{S}^{-1}(\mathsf{F}(\mathbf{v}_\Gamma^\star)) \tag{3.6}$$

or in terms of the solid interface velocity[1] $\tilde{\mathbf{v}}_{\Gamma}^{n+1}$

$$\tilde{\mathbf{v}}_{\Gamma}^{n+1} = \mathsf{S}^{-1}(\mathsf{F}(\mathbf{v}_{\Gamma}^{\star})) \tag{3.7}$$

As said above, in this work strong coupling partitioned schemes are considered. Hence, the D-N algorithm turns to be an iterative procedure to be solved at each time step. Therefore, the D-N coupling operator turns to be

$$\tilde{\mathbf{v}}_{\Gamma,i+1}^{n+1} = \mathsf{S}^{-1}(\mathsf{F}(\mathbf{v}_{\Gamma,i}^{n+1})) \tag{3.8}$$

where the prediction $\mathbf{v}_{\Gamma}^{\star}$ is the previous iteration solution $\mathbf{v}_{\Gamma,i}^{n+1}$. Besides, the D-N algorithm iteration residual can be defined as

$$\mathbf{r}_{\Gamma,i+1}^{n+1} = \tilde{\mathbf{v}}_{\Gamma,i+1}^{n+1} - \mathbf{v}_{\Gamma,i}^{n+1} \tag{3.9}$$

In a parallel fashion, one can alternatively define the residual as

$$\begin{aligned} \mathbf{r}_{\Gamma,i+1}^{S,n+1} &= \mathsf{S}^{-1}(\mathbf{f}_{\Gamma,i}^{n+1}) - \mathbf{v}_{\Gamma,i}^{n+1} \\ \mathbf{r}_{\Gamma,i+1}^{F,n+1} &= \mathsf{F}(\mathbf{v}_{\Gamma,i}^{n+1}) - \mathbf{f}_{\Gamma,i}^{n+1} \end{aligned} \tag{3.10}$$

The scheme depicted in Eq. 3.9 is known as *Gauss-Seidel iteration* and requires a sequential resolution of the fluid and solid problems. On the other hand, the scheme in Eq. 3.10 is known as *Jacobi iteration* and allows the parallel resolution of the fluid and solid problems. In this work, the Gauss-Seidel iteration has been mainly used because of its better convergence behaviour.

Finally, in an ALE framework, as the one considered in this work, the mesh problem has to be solved once the interface displacement solution has been obtained. Thus, the mesh solver becomes in a sort of interior nodes mapper which recomputes their coordinates from the obtained interface displacement of the D-N coupled problem.

For further information regarding the D-N scheme implementation and its application to FSI problems the reader is encouraged to review [26] and [44].

[1]Once the solid problem has been solved in terms of the displacement unknowns, the velocities can be straightforwardly obtained via the time integration scheme formulas.

3.1.2 Neumann-Neumann iterative scheme

First of all, for the sake of simplicity let us consider the compact forms previously defined in Eq. 3.1 and 3.2. Slightly modifying both of them, the N-N fluid problem reads

$$\begin{bmatrix} \mathbf{F}_{II} & \mathbf{F}_{I\Gamma} \\ \mathbf{F}_{\Gamma I} & \mathbf{F}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{v}_I^{F,n+1} \\ \mathbf{v}_\Gamma^{F,n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^{F,n+1} \\ \mathbf{f}_\Gamma^{F,n+1} + \mathbf{f}_\Gamma^\star \end{bmatrix} \tag{3.11}$$

while the N-N solid problem is

$$\begin{bmatrix} \mathbf{S}_{II} & \mathbf{S}_{I\Gamma} \\ \mathbf{S}_{\Gamma I} & \mathbf{S}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{u}_I^{S,n+1} \\ \mathbf{u}_\Gamma^{S,n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^{S,n+1} \\ \mathbf{f}_\Gamma^{S,n+1} - \mathbf{f}_\Gamma^\star \end{bmatrix} \tag{3.12}$$

being $\mathbf{f}_\Gamma^\star$ a suitable approximation of the interface fluxes required to the fluid and solid subdomains equilibrium.

Rearranging terms, both the fluid and the solid domain can be expressed in terms of the interface unknowns. Hence, after some algebra, the fluid problem reads

$$\left(\mathbf{F}_{\Gamma\Gamma} - \mathbf{F}_{\Gamma I}\mathbf{F}_{II}^{-1}\mathbf{F}_{I\Gamma}\right)\mathbf{v}_\Gamma^{n+1} = \mathbf{f}_\Gamma^{F,n+1} + \mathbf{f}_\Gamma^\star - \mathbf{F}_{\Gamma I}\mathbf{F}_{II}^{-1}\mathbf{f}_I^{n+1} \tag{3.13}$$

while the solid problem (in velocity terms) reads

$$\left(\mathbf{S}_{\Gamma\Gamma} - \mathbf{S}_{\Gamma I}\mathbf{S}_{II}^{-1}\mathbf{S}_{I\Gamma}\right)\mathbf{v}_\Gamma^{n+1} = \mathbf{f}_\Gamma^{S,n+1} - \mathbf{f}_\Gamma^\star - \mathbf{S}_{\Gamma I}\mathbf{S}_{II}^{-1}\mathbf{f}_I^{n+1} \tag{3.14}$$

being the operator acting over the interface unknown in the LHS the so called interface or Schur complement operator. In this context, it can be noticed that in the N-N scheme both the fluid and the solid problems act as an operator such that, given a suitable interface flux prediction, gives back either an interface displacement or an interface velocity. Henceforth, let us define the N-N fluid problem operator $\mathbf{v}_\Gamma^{F,n+1} = \mathsf{F}^{-1}(\mathbf{f}_\Gamma^\star)$ and the N-N solid problem operator $\mathbf{v}_\Gamma^{S,n+1} = \mathsf{S}^{-1}(-\mathbf{f}_\Gamma^\star)$.

It is important to point out that the resolution of the presented N-N scheme consists in minimizing the interface velocity residual since the interface flux equilibrium is imposed by definition. Thus, the N-N residual to minimize by means of an iterative scheme is defined as

$$\mathbf{r}_{\Gamma,i+1}^{n+1} = \mathbf{v}_{\Gamma,i+1}^{F,n+1} - \mathbf{v}_{\Gamma,i+1}^{S,n+1} \tag{3.15}$$

that is to say

$$\mathbf{r}^{n+1}_{\Gamma,i+1} = \mathsf{F}^{-1}(\mathbf{f}^{i+1}_{\Gamma}) - \mathsf{S}^{-1}(-\mathbf{f}^{i+1}_{\Gamma}) \tag{3.16}$$

In addition, it is interesting to note in 3.16 that the presented N-N algorithm allows to solve the fluid and solid problems in a parallel way.

Finally, it is worth observing that the presented N-N algorithm is equivalent to impose the interface velocity continuity $\mathbf{v}^{F,n+1}_{\Gamma} = \mathbf{v}^{S,n+1}_{\Gamma}$ via Lagrange multipliers in a monolithic resolution. Indeed, the Lagrange multipliers can be viewed as the interface flux required to fulfil the interface velocity continuity constraint. Similar implementation and applications of the N-N scheme presented in this work can be found in [27].

3.2 Residual minimization techniques

In these section, the *Aitken relaxation* scheme, the *Jacobian-Free Newton-Krylov methods* (JFNK) and the *Multivector Quasi Newton method* (MVQN) are presented as residual minimization techniques. These techniques fulfil the black box solver requirement stated in the introduction of this chapter. Thus, they have the ability of iteratively minimize the interface residual without any access to the interior of neither the fluid nor the solid solvers.

3.2.1 Relaxation techniques

Relaxation techniques are nothing but the line search step of a non-linear solver. Typically, relaxation techniques are used in combination with the D-N iterative scheme. The D-N relaxation algorithm used in this work is presented in Alg. 1. As can be seen, it consists on solving the iterative D-N coupled problem in Eq. 3.8 but considering a relaxed initial approximation at each iteration. Thus, the initial approximation at iteration $i+1$ turns to be

$$\mathbf{v}^{F,n+1}_{\Gamma,i+1} = \mathbf{v}^{F,n+1}_{\Gamma,i} + w\left(\tilde{\mathbf{v}}^{S,n+1}_{\Gamma,i} - \mathbf{v}^{F,n+1}_{\Gamma,i}\right) = \mathbf{v}^{F,n+1}_{\Gamma,i} + w\mathbf{r}^{n+1}_{\Gamma,i} \tag{3.17}$$

being w the so called relaxation parameter and $\mathbf{r}^{n+1}_{\Gamma,i+1}$ the current iteration residual in 3.9.

The relaxed initial approximation computation depends on the relaxation technique used. The simplest one is the so called *fixed relaxation method* in where a constant relaxation parameter w is used. However, there exists dynamic techniques, such as the *Aitken relaxation method*, in where the relaxation parameter is updated at each iteration.

Algorithm 1 D-N relaxation scheme.

```
1:  for $t \leq t_{end}$ do:
2:      $i = 0$
3:      $\mathbf{v}_{\Gamma,0}^{S} = \mathbf{v}_{\Gamma}^{S,n}$
4:      while $i \leq i_{max}$ do
5:          solve fluid: $\mathbf{f}_{\Gamma,i+1}^{F,n+1} = \mathsf{F}(\mathbf{v}_{\Gamma}^{S,i})$
6:          solve solid: $\tilde{\mathbf{v}}_{\Gamma,i+1}^{S,n+1} = \mathsf{S}^{-1}(-\mathbf{f}_{\Gamma,i+1}^{F,n+1})$
7:          compute $\mathbf{r}_{\Gamma,i+1}^{n+1} = \tilde{\mathbf{v}}_{\Gamma,i+1}^{S,n+1} - \mathbf{v}_{\Gamma}^{S,i}$
8:          if $\|\mathbf{r}_{\Gamma,i+1}^{n+1}\| \leq \|\mathbf{r}_{tol}\|$ then:
9:              Break
10:         else:
11:             if not fixed, compute $w_{i+1}^{n+1}$
12:             update $\mathbf{v}_{\Gamma,i+1}^{S,n+1} \rightarrow \mathbf{v}_{\Gamma,i}^{F,n+1} + w_{i+1}^{n+1}\mathbf{r}_{\Gamma,i}^{n+1}$
13:             $i = i + 1$
14:     $t = t + \Delta t$
```

Fixed relaxation parameter

The simplest and most ineffective method is to choose a fixed relaxation parameter w for all the time steps. The relaxation parameter has to be small enough to ensure convergence but as large as possible to minimize the computational effort. Hence, the main difficulty of the method is to choose a suitable relaxation parameter, since the optimal value is completely problem dependent and impossible to know a priori.

Aitken relaxation scheme

As is pointed in [26], the Aitken relaxation scheme has proven to be astonishingly simple and efficient. The main idea of the Aitken relaxation is to use information from two previous iterations to compute the relaxation parameter w_{i+1} at the current one.

Thus, the Aitken dynamic relaxation parameter in a vector problem can be obtained as

$$w_{i+1} = -w_i \frac{(\mathbf{r}_{\Gamma,i})^T(\mathbf{r}_{\Gamma,i+1} - \mathbf{r}_{\Gamma,i})}{\|\mathbf{r}_{\Gamma,i+1} - \mathbf{r}_{\Gamma,i}\|^2} \tag{3.18}$$

where the residuals $\mathbf{r}_{\Gamma,i+1}$ and $\mathbf{r}_{\Gamma,i}$ are defined as is done in Eq. 3.9. Further details on the derivation of the Aitken relaxation parameter can be found in [26]. The performance of the Aitken relaxation is assessed and compared against other line search methods in [30].

Regarding the initialization of the relaxation parameter at the beginning of the analysis, it is a common practice to perform the first iterations with a fixed parameter. In this work, a value of $w_0 = 0.825$ has been considered. This value has been used in [26] as fixed relaxation parameter in similar problems to the ones presented. Furthermore, it is also common to initialize the relaxation parameter at each time step as the last one obtained in the previous step.

3.2.2 Newton family methods

The methods presented in this subsection are suitable for both the D-N and N-N algorithms presented above. For the sake of a lighter notation, from now on and otherwise stated, the step is always $n+1$ and both the residual and the unknown are defined in the interface Γ. Hence, let us consider any interface residual $\mathbf{r}$ coming from an interface unknown $\mathbf{u}$. Considering a 1st order Taylor expansion, the residual $\mathbf{r}$ can be approximated as

$$\mathbf{r}_{i+1} \approx \mathbf{r}_i + \frac{\partial \mathbf{r}_{i+1}}{\partial \mathbf{u}} \Delta \mathbf{u}_{i+1} \tag{3.19}$$

Taking into account that one wants the previous residual to be null, it becomes in

$$-\mathbf{r}_i = \mathbf{J}_i \Delta \mathbf{u}_{i+1} \tag{3.20}$$

where the previous Jacobian matrix has been denoted as $\mathbf{J}_i$. Then, the $i+1$ solution update, which can be obtained from the inversion of the Jacobian in 3.20, is applied as follows to obtain the current iteration solution [2]

[2]Recall that in this case $\mathbf{u}_{i+1}$ represents any unknown interface variable, since the method can be equivalently applied to a problem formulated either in terms of the displacements (velocities) or the nodal fluxes.

$$\mathbf{u}_{i+1} = \mathbf{u}_i + \Delta\mathbf{u}_{i+1} \tag{3.21}$$

It is interesting to point out that the inherent non-linearity of the FSI problem is handled via the iterative nature of the method. In other words, in a linear problem the method must always converge in one iteration.

So far, nothing else but the standard Newton-Raphson method for non-linear problems has been presented. Despite the extensive evidence of the good performance of the Newton-Raphson method, in the black box algorithms context it has the great disadvantage of requiring the explicit Jacobian computation. As pointed before, in black box coupling techniques, the access to the solver is limited, meaning that the Jacobian is usually not available or if it is, its computation and storage becomes expensive in computational effort terms. In this context, the methods in which the system Jacobian is approximated rather than computed arise as an extremely good solution for this issue.

In the next lines, two methods in where the Jacobian is not explicitly computed are presented. The former is the *non-linear Jacobian Free GMRES* (JFNK). The latter is the recently developed *Multivector Quasi-Newton method* (MVQN). Besides, there exist other Quasi-Newton approaches that might be used for the resolution of FSI problems. Some of them are the *Broyden method* or the *Broyden-Fletcher-Goldfarb-Shanno method* (BFGS). The application of these methods to strongly coupled procedures can be respectively found in [28] and [30].

Non-linear Jacobian Free GMRES

The *Generalised Minimal Residual method* (GMRES) belongs to the Newton-Krylov family of linear solvers. Newton-Krylov solvers can be defined as iterative methods for large linear systems that use the j^{th} Krylov subspace from

$$\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0) = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^0\mathbf{r}_0, \ldots, \mathbf{A}^{j-1}\mathbf{r}_0\} \tag{3.22}$$

where $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ is the initial linear residual for a given initial approximation $\mathbf{x}_0$ of the solution $\mathbf{x}$. There are multiple Newton-Krylov solvers such as the *Conjugate Gradient* (CG), the *Arnoldi method*, the *Lanczos method* or the mentioned *Generalised Minimal Residual* (GMRES).

In this work the GMRES algorithm has been selected as main Newton-Krylov linear system solver due to its widely proven well performance. However, it is worth mentioning that any other Newton-Krylov solver could be used in a similar fashion.

The main idea of the GMRES method is to write the sought solution in terms of an orthonormal base, which is found via Arnoldi iteration, to the one in Eq. 3.22. Moreover, the GMRES algorithm has the interesting feature of finding a root in a maximum number of iterations equal to the number of unknowns, always providing that the problem converges. Further information regarding the formulation and performance analysis of the GMRES linear solver can be found in [40].

It is interesting to note that in Eq. 3.20 the Jacobian is not explicitly needed but its projection onto a vector. Then, the Jacobian computation can be avoided approximating such projection using finite differences. Thus, the finite differences Jacobian projection approximation reads

$$\mathbf{J}_i \Delta\mathbf{u}_{i+1} \approx \frac{\mathbf{R}(\mathbf{u}_i + \varepsilon\Delta\mathbf{u}_{i+1}) - \mathbf{R}(\mathbf{u}_i)}{\varepsilon} \tag{3.23}$$

where $\mathbf{R}()$ is any residual function and ε is a small perturbation. It is mandatory to comment that this small perturbation has the function of linearising the problem. Taking into account that the GMRES is conceived as a method to solve linear system of equations, a non-linear residual, as the one that arises in a FSI problem, cannot be considered.

In this context, the perturbation value ε can be viewed as the distance covered by the correction, meaning that if ε is sufficiently small $\mathbf{R}(\mathbf{u}_i + \varepsilon\Delta\mathbf{u}_{i+1}) \approx \mathbf{r}_{i+1}$, that is to say that the two previous residual problems are close to be equal implying that the Jacobian approximation is almost linear and can be solved with the GMRES method.

However, the selection of a value for such perturbation ε is one of the disadvantages of the presented method since it is completely problem dependent and might condition the convergence. Hence, it must be small enough to linearise the residual but as large as possible to reduce the computational effort. In [30] a formulae for the computation of ε is used. However, it is also recommended to consider a ε value which order is around half the machine precision. Taking this statement into account, values $\varepsilon = 10^{-6}$ and $\varepsilon = 10^{-7}$ have been used in this work.

Thus, to solve the residual equation in Eq. 3.20 with the GMRES method and the Jacobian approximation in Eq. 3.23 yields the non-linear Jacobian-Free GMRES algorithm, in where the non-linearity of the problem is handled by means of an outer loop, which iterates until a non-linear tolerance criteria tol_{NL} is fulfilled or up to a maximum number of non-linear iterations.

Summarizing, the main features of the presented Jacobian-Free non-linear GMRES strategy implementation are listed below

- The non-linearity is handled by means of an external loop that carries on until a given non-linear convergence criterion is fulfilled.
- The capability of the GMRES to iteratively solve large linear systems of equations is exploited in the resolution of each one of the linearised residual equations.
- The Jacobian computation is avoided approximating its projection onto a vector via finite differences.
- The Jacobian of the interface residual is linearised provided that ε is sufficiently small.

Finally, the non-linear Jacobian-free GMRES algorithm implemented in this work is presented in Alg. 2.

Multivector Quasi-Newton method

According to [4], the MVQN method is a Quasi-Newton method that can be viewed as a generalisation of the Broyden scheme. Comparing to the Broyden scheme, the MVQN has the advantage of exactly reproduce all the current step information, as opposed to the last observation of the current step information. This leads to an inherent consistency condition with Newton's method. Therefore, the MVQN method is based in approximating the inverse of the Jacobian in Eq. 3.20 with the information stored in the so called observation matrices, which are defined as follows

$$\mathbf{V}_i = [\Delta\mathbf{r}_i \; \Delta\mathbf{r}_{i-1} \ldots \Delta\mathbf{r}_2 \; \Delta\mathbf{r}_1] \tag{3.24a}$$

$$\mathbf{W}_i = [\Delta\mathbf{u}_i \; \Delta\mathbf{u}_{i-1} \ldots \Delta\mathbf{u}_2 \; \Delta\mathbf{u}_1] \tag{3.24b}$$

where $\Delta\mathbf{r}_i = (\tilde{\mathbf{u}}_i - \mathbf{u}_i) - (\tilde{\mathbf{u}}_{i-1} - \mathbf{u}_{i-1})$ and $\Delta\mathbf{u}_i = \mathbf{u}_i - \mathbf{u}_{i-1}$, being $\tilde{\mathbf{u}}$ the solution of the D-N operator in Eq. 3.6. Moreover, the MVQN can be also applied to the residual form of the N-N algorithm. For the N-N residual case, $\Delta\mathbf{r}_i = (\mathbf{u}_i^F - \mathbf{u}_i^S) - (\mathbf{u}_{i-1}^F - \mathbf{u}_{i-1}^S)$ while $\Delta\mathbf{f}_i = \mathbf{f}_i - \mathbf{f}_{i-1}$. Then, the inverse Jacobian approximation can be computed by means of the previous observation matrices as

Algorithm 2 Non-linear Jacobian-Free GMRES.

```
1:  $i = 0$
2:  $\mathbf{u}_0 = \mathbf{u}^n$ *(initial guess from prev. time step)*
3:
4:  **while** $i \leq i_{NL,max}$ **do**:
5:      $\mathbf{u}_i = \mathbf{u}_{i-1}$ (*NL iteration guess*)
6:
7:      **Base residual computation:**
8:      *Solve fluid:* $\mathsf{F}(\mathbf{u}_i)$
9:      *Solve solid:* $\mathsf{S}(\mathbf{u}_i)$
10:     *Base residual:* $\mathbf{r}_i = \mathsf{F}(\mathbf{u}_i) - \mathsf{S}(\mathbf{u}_i)$
11:
12:     **Linearised correction:**
13:     *Create the callable:* $\mathbf{J}\Delta\mathbf{u} = \frac{\mathsf{R}(\mathbf{u}_i + \varepsilon\Delta\mathbf{u}_i) - \mathbf{r}_i}{\varepsilon}$
14:     *GMRES minimization:* $\Delta\mathbf{u} = \mathsf{GMRES}(\mathbf{J}\Delta\mathbf{u}_i, -\mathbf{r}_i)$
15:     *NL iteration guess update:* $\mathbf{u}_i = \mathbf{u}_i + \Delta\mathbf{u}$
16:
17:     *Convergence check:*
18:     **if** convergence = True **then**:
19:         **Break**
```

$$\mathbf{J}_{i+1}^{n+1} = \mathbf{J}^{n} + \left(\mathbf{W}_i - \mathbf{J}^{n}\mathbf{V}_i\right)\left(\mathbf{V}_i^T\mathbf{V}_i\right)^{-1}\mathbf{V}_i^T \tag{3.25}$$

The pseudo-code of the MVQN method is depicted in Alg. 3. Note the necessity of performing a fixed point iteration at the very beginning of the problem to have enough information to initialize the observation matrices. However, this is only needed at the first time step, since in the steps onwards the first iteration update can be performed with the stored previous step Jacobian. Besides, an initial approximation of the inverse Jacobian is also required. In this case $\mathbf{J}^0 = -\mathbf{I}$.

In addition, in [4] it is stated that the number of vectors in the observation matrices should never exceed the number of interface DOFs to avoid an over-determined problem. According to the convergence behaviour of the MVQN method this rarely occurs, but if it happens to be the case, the last column, which contain the older information, can be dropped at each iteration.

Finally, it has to be said that the MVQN can be also applied to the block iterative equations form as opposed to the residual form described here. According to [4], the block form is slightly more stable but requires to solve two sets of linear equations. Thus, the residual equations as described here are expected to be slightly faster as the system size increases.

Algorithm 3 Multivector Quasi-Newton method.

```
1: $i = 1$
2: $\mathbf{u}_0 = \mathbf{u}^n$
3: Empty $\mathbf{V}_i$ and $\mathbf{W}_i$
4:
5: while $i \leq i_{nl,max}$ do:
6:     Solve the coupled problem$\rightarrow$ Get $\mathbf{r}_i$
7:     Check convergence:
8:     if convergence = True then:
9:         Break
10:    else:
11:        if step $= 1$ and $i = 1$ then:
12:            $\mathbf{u}_1 = \mathbf{u}_0 + w\mathbf{r}_0$
13:        else if step $\neq 1$ and $i = 1$ then:
14:            $\mathbf{J}_2^{n+1} = \mathbf{J}^n$
15:            $\Delta\mathbf{u}_2 = \mathbf{J}_2^{n+1}\mathbf{r}_1$
16:            $\mathbf{u}_2 = \mathbf{u}_1 + \Delta\mathbf{u}_2$
17:        else:
18:            $\mathbf{V}_i$ and $\mathbf{W}_i$ updates
19:            $\mathbf{J}_{i+1}^{n+1} = \mathbf{J}^n + (\mathbf{W}_i - \mathbf{J}^n\mathbf{V}_i)\left(\mathbf{V}_i^T\mathbf{V}_i\right)^{-1}\mathbf{V}_i^T$
20:            $\Delta\mathbf{u}_{i+1} = \mathbf{J}_{i+1}^{n+1}\mathbf{r}_i$
21:            $\mathbf{u}_{i+1} = \mathbf{u}_i + \Delta\mathbf{u}_{i+1}$
```

Chapter 4

Tests and results

This chapter collects all the tests carried out to validate the implementation of the methodologies described in chapter 3. First of all, the *Kratos Multiphysics* framework as well as the main aspects of the implementation are described. Secondly, this section also collects the tests performed along with their results.

Regarding the developed simulations, four different problems are presented. The domain decomposition of the well-known cavity flow problem is firstly discussed to check the correctness of the implementation due to its simplicity. Secondly, a first FSI case is presented to assert the performance of the implementation in an FSI context. Besides, as FSI 2D final test the Turek-Hron benchmark [43] is reproduced. Finally, a preliminary 3D case reproducing the blood flow in a human vein is also depicted.

4.1 Code implementation

4.1.1 Kratos Multiphysics and GiD framework

All the code developed in this work has been implemented using the *Kratos Multiphysics* (Kratos) software [14] [16], maintained and developed by CIMNE researchers for more than ten years ago. Kratos is an open-source framework for the implementation of numerical methods for the resolution of engineering problems. Moreover, it is intended to be extremely modular to allow the collaborative development by large teams of researchers. Such modularity is achieved thanks to the *core* and *applications* combination. Thus, the standard tools or procedures such as databases, linear algebra or search structures come as a part of the core and are available as building blocks in the development of applications, which focus on the solution of the problem of interest.

In addition, Kratos aims to the optimization and the computational effort reduction. By this reason, it has two implementation levels. A higher one coded in Python, which is used as scripting language due to its simplicity, and a lower one coded in C++ which represents the majority of the code. Furthermore, Kratos also includes parallelization in both OpenMP and MPI languages.

On the other hand, the *GiD* v12 commercial software has been used as pre and post-processor. GiD is also developed by CIMNE researchers and works in combination with Kratos thanks to a specific *problemtype*. GiD Kratos problemtype allows to introduce all the problem settings such as materials, boundary conditions or solution strategies needed to perform a simulation in Kratos.

Further information regarding Kratos Multiphysics and GiD can be found in their official websites [15] and [29].

4.1.2 Implementation aspects

Recovering the black boxes concept introduced in chapter 3, the *SolidMechanicsApplication* and *FluidMechanicsApplication*, already implemented in Kratos, are used as black box solvers for the solid and fluid domains of the FSI problem. Thus, the main effort of the presented implementation, which has been programmed in the Python top level of Kratos, is related with the coupling between these two applications.

Despite the fact that C++ has a much more optimized performance than Python, this is not a large disadvantage in this case, since the most computationally expensive tasks (the solid and fluid problems resolution) are executed at the C++ level by their corresponding applications. In this manner, one can take advantage of the simplicity and flexibility of Python language during the implementation.

However, it has to be said that at the beginning of this work some functionalities were missing in the existent applications. Apart of some minor modifications, the main implementation carried out within Kratos was the creation of a punctual load condition to be applied at the RHS of the fluid problem for the imposition of nodal fluxes in the N-N coupling algorithm.

Regarding the Python level implementation, it can be roughly summarized in four main types of scripts that each one of the presented simulations must have. These files are listed below.

- Main file: This script, called `MAIN_FILE_FSI.py`, launches the simulation.
- Strategy scripts: These scripts collect the coupling resolution. Thus, they contain the implemented coupling strategies distributed in the next set of files
  - `mvqn_strategy.py`
  - `jfnk_strategy.py`
  - `relaxation_strategy.py`

  as well as the residual definitions in `residual_definitions.py`.
- Fluid class script: This script (`FluidProblemClass.py`) acts as a communicator between the main script and the Kratos fluid black box solver. Thus, it is a collection of instructions to be called in the main file to perform tasks such as initialize the problem, solve it, advance in time or write the output files.
- Solid class script: In a similar fashion to the fluid class script, the solid class script (`SolidProblemClass.py`) acts as a communicator between the main script and the Kratos fluid black box solver.
- Auxiliary scripts: Contain utilities to perform secondary tasks such as the communication between solid and fluid at the interface.

In addition, it has to be said that the main scripts pointed before have been programmed in an object oriented fashion, meaning that any strategy or new residual definition can be straightforwardly added if their syntax is respected. Thus, any new residual must have a constructor as well as a `ComputeResidual` method that given an input array gives back a residual. For the coupling strategies, they must have a constructor besides the methods `ExecuteInitializeSolutionStep`, `InterfaceSolutionUpdate` and `ExecuteFinalizeSolutionStep`.
The main files and scripts described above can be found in annex A.

4.2 Cavity flow problem

4.2.1 Problem description

In this section the coupling approaches developed in chapter 3 are used to perform the domain decomposition of the well-known cavity flow problem. Despite this is not a FSI problem, its domain decomposition approach is specially useful to check correctness thanks to the known solution. Moreover, the computational cost of each one of the strategies developed can also be assessed by means of this problem.

As depicted in figure 4.1, the cavity flow problem consists in a fluid contained in a unit length square domain with Dirichlet boundary conditions on all its sides: three stationary and one moving. Depending on how the moving side condition is imposed, one can distinguish two different cases. The former is the lid-driven cavity flow problem in where the corners of the moving side have velocity different than zero. The latter is the shear-driven cavity flow problem, in where the corners of the moving side have zero velocity. This last one has been selected in this work. In [17] a deep analysis of the cavity flow problem simulation can be found.

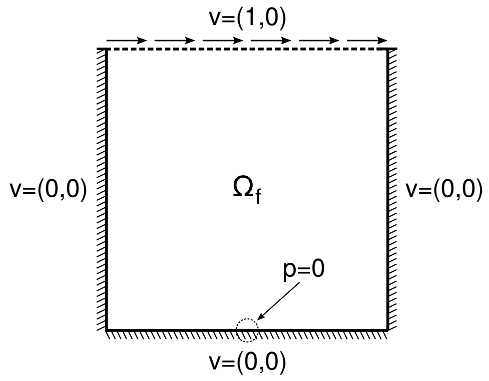
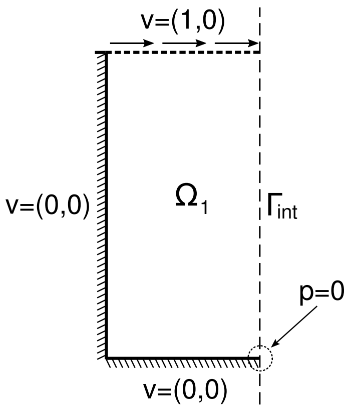


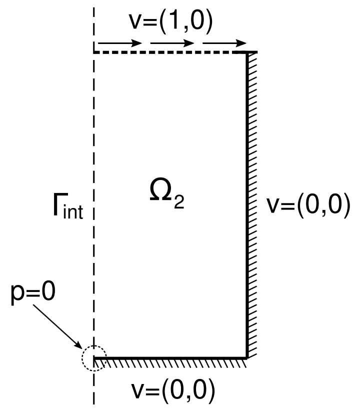
Figure 4.1: Cavity flow problem.

Thus, the presented problem consists in two disjoint domains in which the Navier-Stokes equations are solved considering that their solutions are coupled at the interface. Figure 4.2 depicts the geometry of the coupled partitioned problem. Regarding the Reynolds number, the characteristic velocity, density and viscosity were set such that $Re = 100$. As a consequence, the convective term contribution is almost null, meaning that the flow is close to be of Stoke's type. Thus, no stability nor turbulence problems are expected.

Regarding the mesh, a structured mesh of 1720 elements (20×40 equally spaced edge



(a) Left domain.



(b) Right domain

Figure 4.2: Cavity flow problem domain decomposition.

divisions) has been used at each subdomain. The element type is a Q1P1 triangular element for fluids. The time as well as material parameters of the simulation are listed below.

- $t_{tot} = 1sec$
- $\Delta t = 0.01sec$
- $\rho = 1000kg/m^3$
- $\nu = 0.01m^2/s$

Finally, the convergence criterion has been set as the L^2-norm of the interface velocity residual in accordance to the algorithms presented in chapter 3. Thus, the convergence criterion is $\|\mathbf{r}\| = \|\mathbf{v}_{\Gamma,1} - \mathbf{v}_{\Gamma,2}\| \leq tol_{NL}$, being $tol_{NL} = 1e-05$.

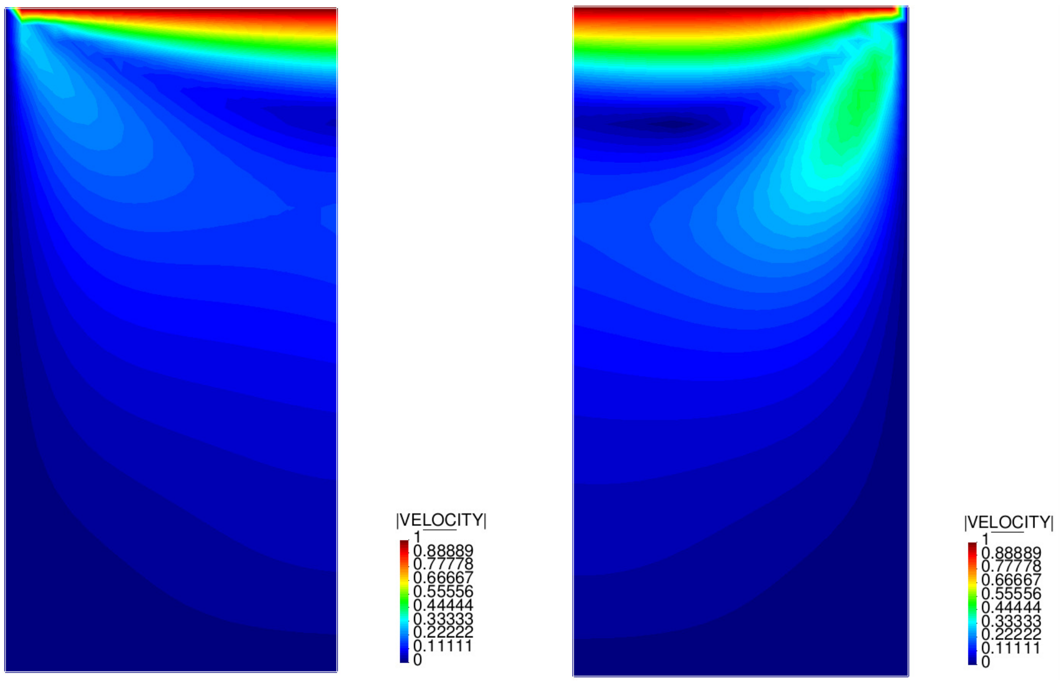
4.2.2 Results assessment

To assess the performance of the aforementioned coupling algorithms and residual minimization techniques, the partitioned cavity flow problem described above has been solved by means of the next strategies

- Dirichlet-Neumann coupling with Jacobian Free non-linear GMRES residual minimization (D-N JFNK).
- Dirichlet-Neumann coupling with Aitken relaxation (D-N Aitken).

- Dirichlet-Neumann coupling with Multivector Quasi-Newton Method residual minimization (D-N MVQN).
- Neumann-Neumann coupling with Jacobian Free non-linear GMRES residual minimization (N-N JFNK).
- Neumann-Neumann coupling with Multivector Quasi-Newton method residual minimization (N-N MVQN).

All the D-N iterative schemes related above are developed in a Gauss-Seidel fashion. Despite this, Jacobi iterative schemes were also tried but no convergence was achieved.
First of all, one of the obtained partitioned solutions is presented in figure 4.3. In this case, it has been selected to show the solution obtained with the N-N JFNK coupling approach as a sample of all the partitioned solutions computed, since no differences can be appreciated. As can be seen, the partitioned solution matches the expected cavity flow problem solution.



(a) Left domain solution.   (b) Right domain solution.

Figure 4.3: $\|v\|$ partitioned results for $Re = 100$ and $t = 1$ (N-N JFNK algorithm).

Complementary, figure 4.4 depicts the obtained v_x, v_y and pressure results at an horizontal cross section placed at $y = 0.8$. As commented above, there is barely difference between the five implemented techniques. Moreover, the results perfectly match the complete reference solution. Figures 4.4a, 4.4b, 4.4c and 4.4d also shown that the interface velocity continuity requirement is fulfilled by all the coupling techniques developed.

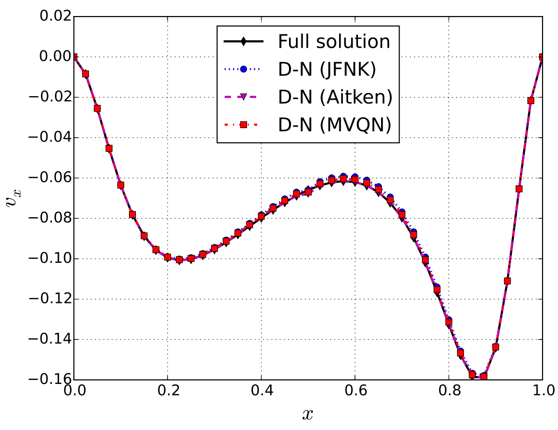
The same cross section data has been extracted from the pressure field (figures 4.4e and 4.4f). As was the case for the velocity results, the partitioned pressure results match the reference complete solution for all the coupling strategies developed. However, it has to be said that a little discontinuity, which becomes smaller as the mesh is refined, appears at the interface values. Taking into account that having two disjoint domains implies that the nodes at the interface location are doubled and considering that only velocity continuity is explicitly imposed, it is possible to have two different pressure values at the interface. Thus, as long as the mesh is refined, the interface pressure must converge to a unique value as has been proved.

On the other hand, the computational performance of each one of the developed coupling strategies has been assessed using the number of residual evaluations and the CPU spent time. The tests have been performed with a Quad-Core AMD FX-8350 processor.

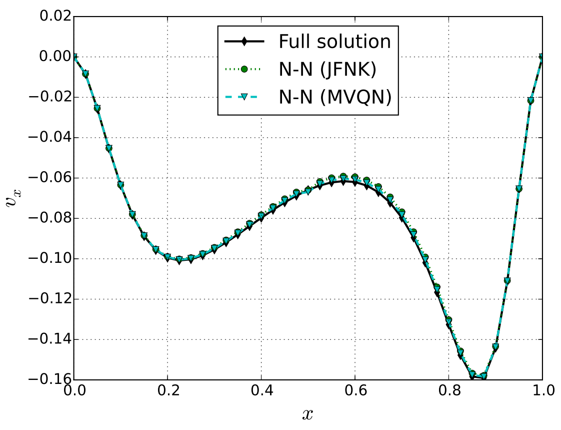
Figures 4.5 and 4.6 collect the number coupled problem evaluations, that is to say, the residual evaluations and the CPU spent time assessment. As can be seen, there are barely variation in the results evolution, meaning that all the presented strategies have almost constant number of problem evaluations.

Besides, it is interesting to point out that the MVQN method performs similarly with both the D-N and N-N schemes. However, this is not the case for the JFNK one, which computational cost is much more reduced when the D-N scheme is used (figure 4.6b). Therefore, it can be observed that the problem linearisation in the JFNK strategy appears to affect more the N-N scheme. Furthermore, it has to be said that independently of the minimization strategy, the N-N scheme is much more sensitive to the initial guess, leading to a much higher number of coupled problem evaluations to reach the equilibrium at the beginning of the problem.

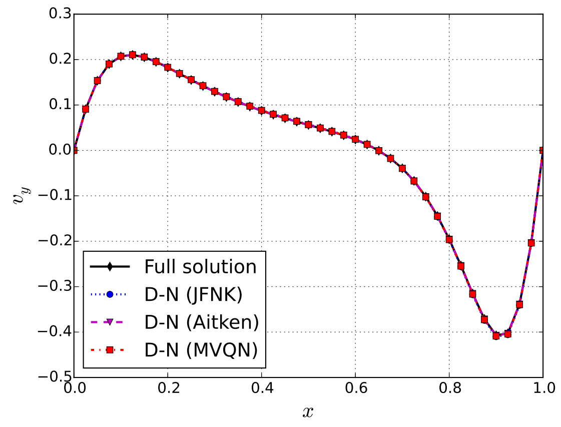
For the general computational effort assessment, it is useful to study the total resolution spent time (figure 4.6b). As can be seen, the superior performance of the two MVQN approaches is overwhelming. Compared with the JFNK strategy, the computational effort reduction of the MVQN method is around the 75% for the D-N scheme and around the 85% for the N-N one. Moreover, it has to be said that considering its simplicity, the Aitken relaxation scheme has an extremely good performance, which in this case is even better than the JFNK one.



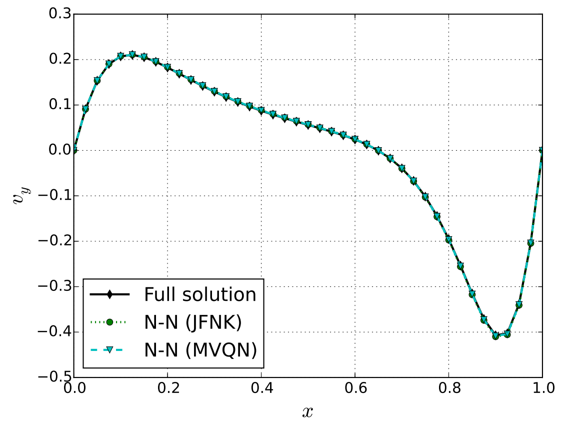
(a) v_x Dirichlet-Neumann algorithms.



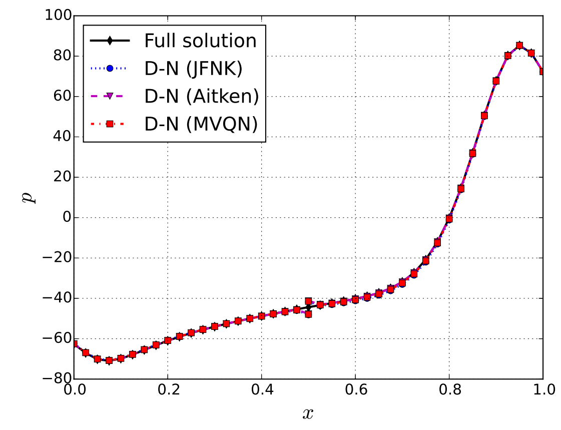
(b) v_x Neumann-Neumann algorithms.



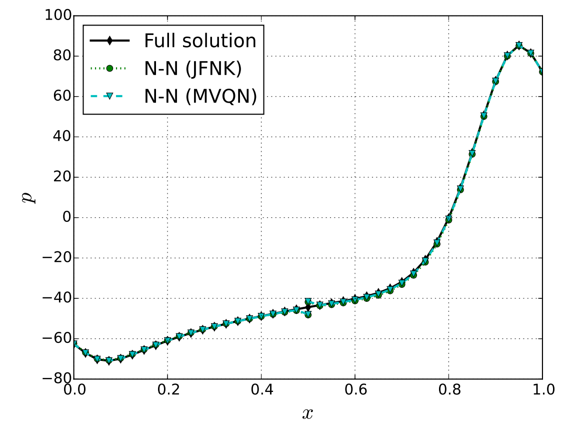
(c) v_y Dirichlet-Neumann algorithms.



(d) v_y Neumann-Neumann algorithms.

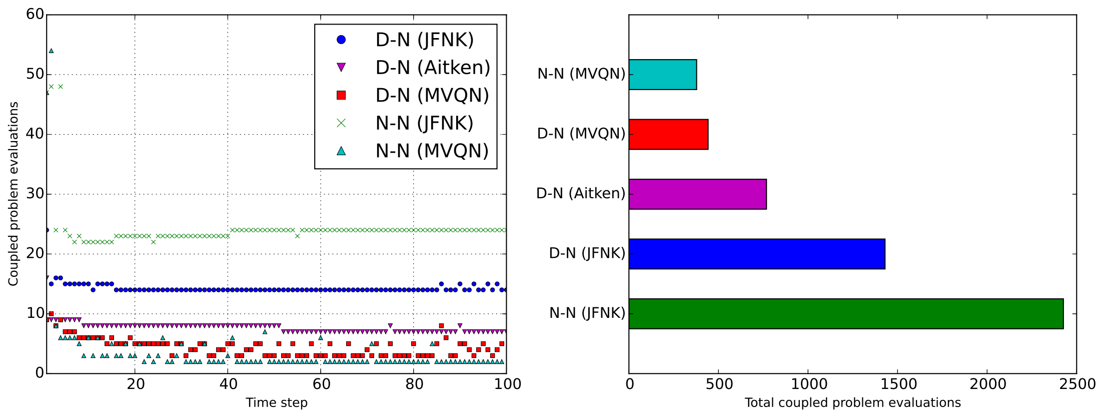


(e) Pressure Dirichlet-Neumann algorithms.



(f) Pressure Neumann-Neumann algorithms.

Figure 4.4: Cross section results comparison ($t = 1$, $y = 0.8$).



(a) Step-by-step evolution. (b) Total evaluations.

Figure 4.5: Coupled problem evaluations comparison.

Finally, it is interesting to point out the reasons that make the MVQN method perform so well compared with the other schemes. The JFNK approach makes a correction based on a linearised residual while the MVQN method is approximating the proper inverse Jacobian considering the original non-linear residual. Moreover, the MVQN method requires only one residual evaluation per non-linear iteration while the JFNK one does as linearised residual evaluations as GMRES iterations, leading to a greater total computational cost. The previous statement also applies for the Aitken relaxation, but what makes the MVQN method to overcome the Aitken relaxation performance is the fact that it is considering information from all the previous non-linear iterations while the Aitken method only considers two non-linear iterations.

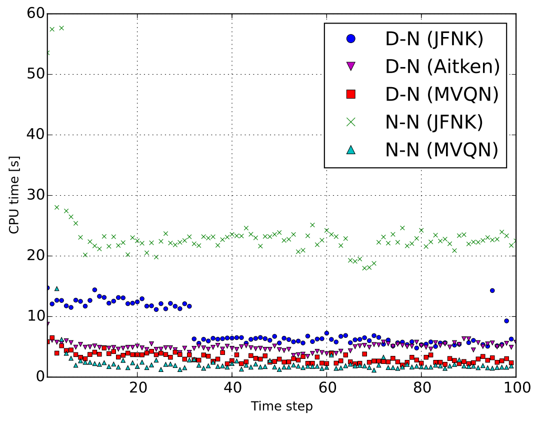
4.2.3 Alternative coupling strategies

This section briefly comments the alternative coupling strategies that were tried to deal with the interface pressure discontinuity. It has to be said that these are some preliminary tests before a deeper study of the presented alternatives.

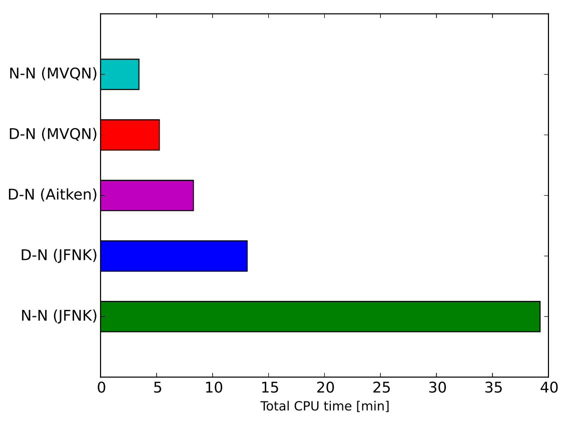
The first trial was to define the N-N algorithm residual in a weak sense as

$$r_\Gamma = \int_\Gamma W\left(v_{\Gamma,1} - v_{\Gamma,2}\right) d\Gamma = 0 \tag{4.1}$$

As can be seen, instead of minimizing the residual nodally, which is equivalent to minimize the L^∞-norm of the interface residual, the L^2-norm is minimized in this case.



(a) Step-by-step evolution.



(b) Total CPU time.

Figure 4.6: Coupled problem CPU times comparison.

To do that, an auxiliary interface mesh, which in the 2D cavity problem case is composed by 1D elements, is used to discretize the weak residual depicted in Eq. 4.1. After the space discretization, the residual to be minimized is

$$\mathbf{r}_\Gamma = \mathbf{M}\mathbf{r}_{nodes} \tag{4.2}$$

where $\mathbf{M}$ is a mass matrix which components are defined as

$$M_{ij} = \int_{\Gamma^e} N_i(x)N_j(x)d\Gamma \tag{4.3}$$

The previous approach led to exactly the same velocity and pressure results that the previously presented ones. Thus, it was decided to append the interface pressure difference $r_{\Gamma,p} = p_{\Gamma,1} - p_{\Gamma,2}$ to the weak residual. In this way, an extra DOF is added to the interface problem to explicitly impose the pressure continuity. Despite the fact that this approach worked in some cases, it do not manage to converge all the times.

This behaviour was associated to the fact that in the incompressible case, the pressure is linked to the velocity to enforce the divergence free condition. Then, adding it as an interface DOF implies to add an extra restriction to the minimization problem without any further information.

Finally, the last idea was to define the residual in the interface edges instead of defining it nodally. Despite this makes the continuity requirement weaker since there are less DOFs, the interface pressure discontinuity remained without noticeable differences.

4.3 Channel with flexible wall problem

4.3.1 Problem description

This model problem was firstly proposed by Mok in 2001 [31]. It consists in a 2D convergent fluid channel that contains a flexible wall structure attached to its bottom wall. The main challenge of the test is that the densities of the fluid and the structure have similar order of magnitude, leading to a strongly coupled problem in where large interaction between the two fields appears.

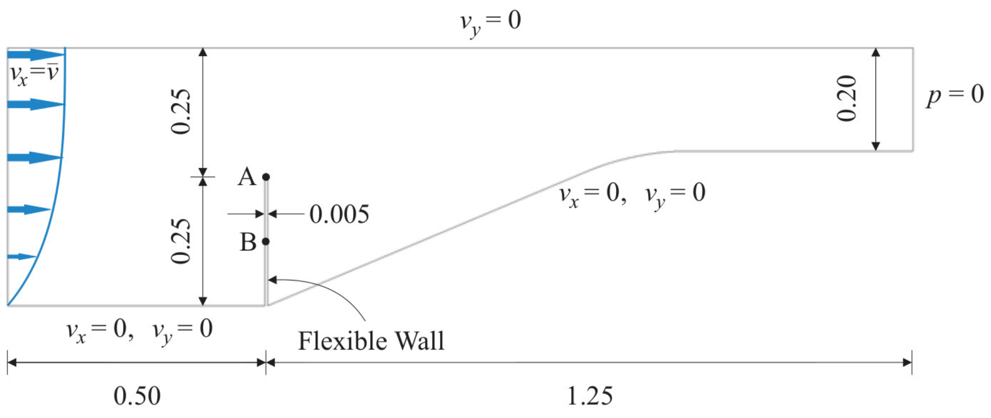


Figure 4.7: Mok benchmark geometry and boundary conditions. Image taken from [44].

As depicted in figure 4.7, the top edge is set as a slip boundary while the inferior one is set as non-slip. Besides, the pressure is set to zero at the right edge. Regarding the left side inlet, it is considered to be parabolic according to the next equation

$$v(y,t) = 4\bar{v}y(1-y) \tag{4.4}$$

where $\bar{v}$ is a time dependent reference velocity such that

$$\bar{v} = \begin{cases} \frac{0.06067}{2}\left(1 - \cos\frac{\pi t}{10}\right) & if \ t \leq 10 \\ 0.06067 & otherwise \end{cases} \tag{4.5}$$

Table 4.1 collects the fluid and solid material properties. Regarding their constitutive equations, the fluid is considered as a simple Newtonian fluid while a linear elastic plane stress law with unit thickness has been used in the solid. Furthermore, geometrical non-linearity is considered since large displacements are expected.

Table 4.1: Mok problem material data.

Structure		Fluid	
ρ^s	1500.0 kg/m^3	ρ^f	956.0 kg/m^3
E	2.3$e10^6$ N/m^2	μ	0.145 $Pa \cdot s$
ν	0.45		

The simulation runs from 0 to 25 seconds with a fixed time step of $\Delta t = 0.1$ seconds. For both the fluid and solid problems, the Kratos default Bossak scheme has been used.

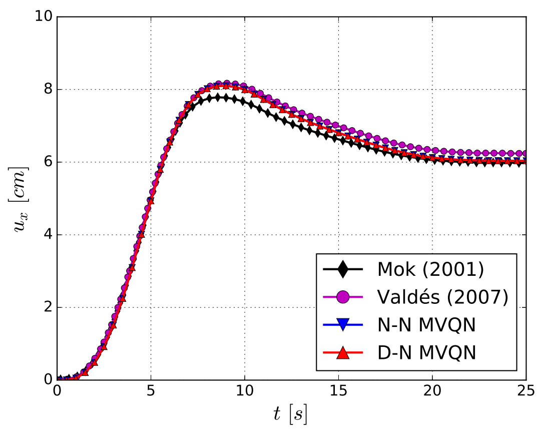
Finally, a mesh composed by 6428 linear triangular elements (Q1P1 element) has been used in the fluid domain. In this way, the fluid mesh is similar to the one presented in [44] and a comparable framework is set. The solid mesh is conformed by 404 linear large displacement quadrilateral elements in order to match the fluid nodes at the interface. The mesh movement is carried out using a non-linear structural technique already implemented in Kratos.

4.3.2 Results assessment

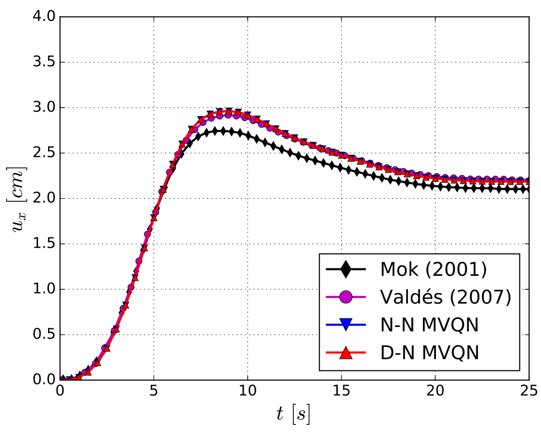
First of all, it has to be clearly stated that for the sake of simplicity, it was decided to show the Mok test results assessment with the D-N MVQN and N-N MVQN strategies, since barely difference can be found with the solution obtained with the other three approaches. Moreover, the interface residual criterion stated for the cavity problem has been kept, since it was considered to be restrictive enough for an FSI problem.

The results assessment has been carried out as is done in [31] and [44]. Thus, the x-component of the velocity v_x as well as the pressure p have been extracted from points A and B, which are located at the tip and the middle point of the windward side of the flexible wall (figure 4.7).

Figure 4.8 compares the obtained x-component displacement evolution in points A and B with the reference solutions. In can be noticed that both D-N MVQN and N-N MVQN strategies have no apparent differences between them. If they are compared with the reference solutions, the point A u_x evolution is in perfect accordance with the one in [44] but as the problem evolves, it becomes closer to the one in [31] (figure 4.8a). For the point B case, it perfectly matches the solution given in [44], which is slightly deviated from the one given in [31] (figure 4.8b).



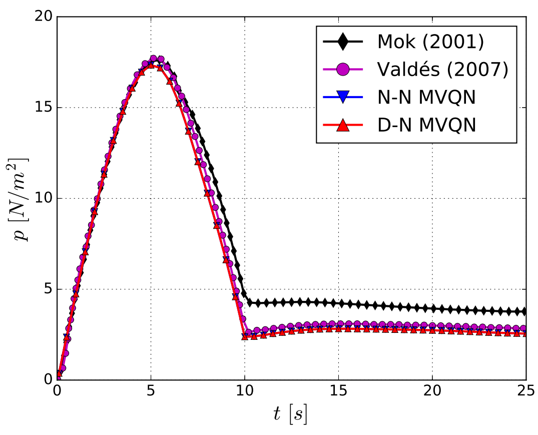
(a) Point A.



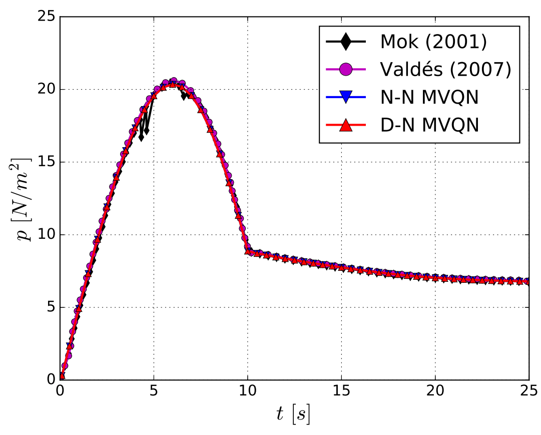
(b) Point B.

Figure 4.8: x-displacement component u_x evolution comparison.

The same results comparison has been carried out for the pressure (figure 4.9). As can be seen in figure 4.9a, both obtained pressure evolutions perfectly match the solution given in [44]. For the point B case (figure 4.9b), the four analysed solutions match. Besides, the pressure oscillations that appear in the original solution do not appear.



(a) Point A.



(b) Point B.

Figure 4.9: Pressure p evolution comparison.

Furthermore, some time snapshots of the obtained solution are shown in figure 4.13 and 4.14. Figure 4.13 depicts the v_x velocity component evolution. This component allows to study how the flow is detached from the top part of the elastic wall and generates a vortex in the leeward side of the wall. Regarding the pressure field evolution (figure 4.14), it is interesting to note the pressure gradient between the windward and the leeward parts of the wall, and how it diminishes when the inlet becomes constant ($t = 10\ s$) and the flow is stabilized.

Once arrived to this point, it can be said that the obtained solutions are correct and match the reference ones, specially the one given by [44]. Taking into account that this solution was confirmed by [39], it can be asseverated that the implementation of the FSI framework is correct.

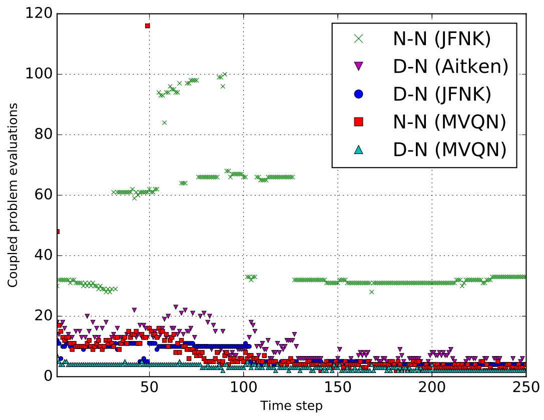
Complementary, a computational effort and convergence assessment has been carried out as well. In this case, the performance analysis have been done with an Intel i7-4790 Quad-Core processor. The previous example absolute residual criterion and tolerance are used.

Figure 4.10a depicts the step-by-step evolution of the coupled problem evaluations. Clearly, the N-N JFNK strategy is the one that have the poor performance. Regarding the other four algorithms, the D-N MVQN and the D-N JFNK ones have a similar evolution trend, in which the required coupled problem evaluations slightly varies along the simulation. For the N-N MVQN it can be observed again that it requires more residual evaluations at the initial steps of the problem but as long as the problem evolves it starts to decrease. The Aitken relaxation has slight variations without a clear pattern.

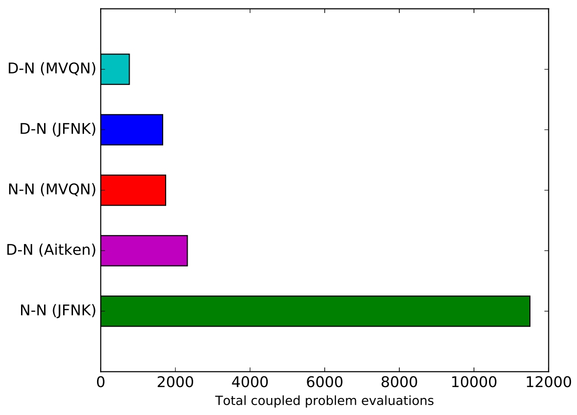
Furthermore, if the total number of coupled problem evaluations is analysed it can be noted that the D-N MVQN requires around one half of the D-N JFNK residual evaluations and around one third of D-N Aitken ones. If both MVQN schemes are compared, the D-N MVQN method performs approximately twice better than the N-N MVQN method.

Additionally, figure 4.11 collects the CPU time assessment. As can be observed, the results are pretty similar to the coupled problem evaluations ones. Studying the total CPU time, it can be asseverated that the MVQN technique performance is overwhelming for both the D-N and the N-N schemes. Indeed, it represents a reduction in the computational effort around the 50% compared to the other D-N schemes (Aitken and JFNK) and around 80% for the N-N scheme case. Consequently, it can be asserted that the most efficient approach is again the D-N MVQN method. Regarding the N-N scheme, it has to be highlighted that it is extremely optimized when the MVQN method is used.

On the other hand, figure 4.12 depicts the convergence rate of the JFNK and MVQN

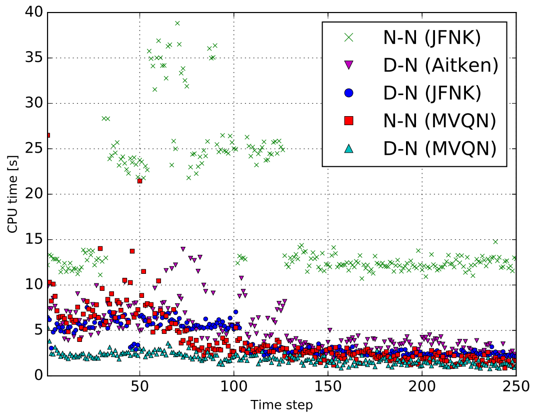


(a) Step-by-step evolution.

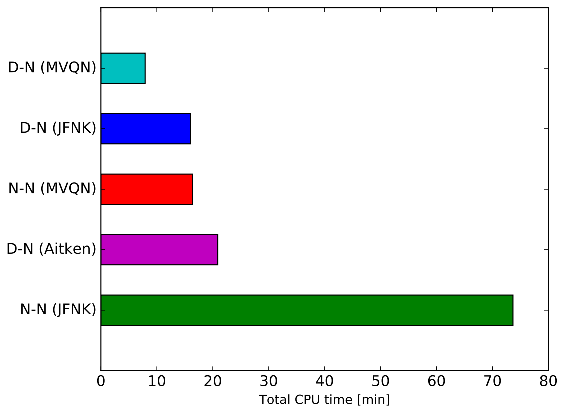


(b) Total evaluations.

Figure 4.10: Coupled problem evaluations comparison.



(a) Step-by-step evolution.



(b) Total CPU time.

Figure 4.11: CPU time comparison.

strategies with both the D-N and N-N coupling schemes. First of all, it can be noted that in all the presented cases the convergence is supralinear. Moreover, the JFNK strategy has a much higher convergence order than the other approaches. Despite this, it requires a larger computational effort due to the above mentioned higher non-linear iteration cost, coming from the fact that each GMRES iteration requires one evaluation of the residual. Comparing the D-N and N-N schemes, the D-N always have a better convergence rate, specially in the MVQN method case.

Finally, it is important to point out some of the particularities that have been observed during the development of the previous example. Regarding the MVQN method, it has to be said that the initial inverse Jacobian approximation widely affects the convergence of the very first steps, meaning that the lack of information yields an initial rate of convergence lower than the expected one. Thus, it is advisable to initialize the problem in a soft manner as is done in the presented case. Secondly, one must take care with the problem tolerances, meaning that if the overall non-linear tolerance is similar to either the fluid or solid solver ones, the method might start to struggle around the non-linear tolerance value without overcoming it.

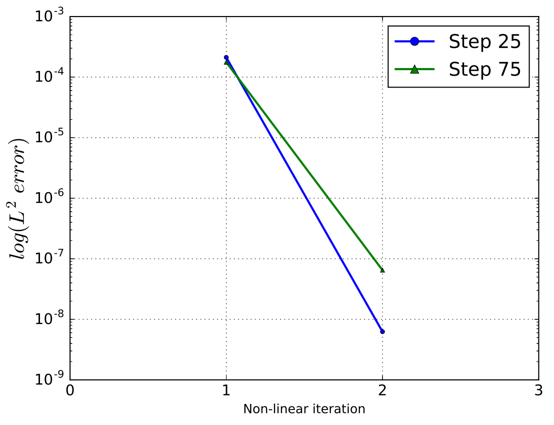
4.4 Turek-Hron benchmark

4.4.1 Problem description

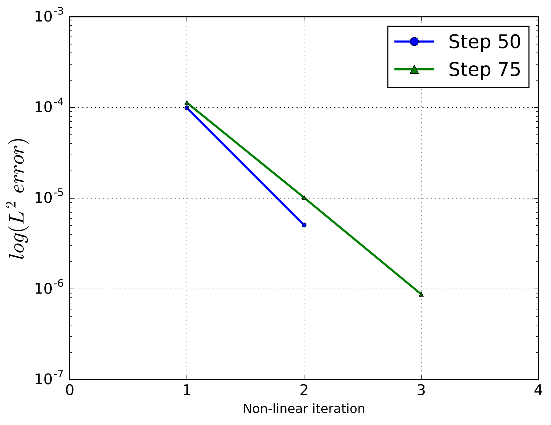
The well-known Turek-Hron benchmark, which is presented in [43], has been selected as final 2D test for the FSI framework developed due to its complexity. There are three configurations of the Turek-Hron benchmark, known as FSI1, FSI2 and FSI3, which involve the same geometry but different settings.

The complexity of the test comes from either the large displacements that the immersed structure develop (FSI2) or by the high frequency of its oscillations (FSI3). Moreover, it has to be taken into account that the solid behaviour considers both geometrical and material non-linearities.

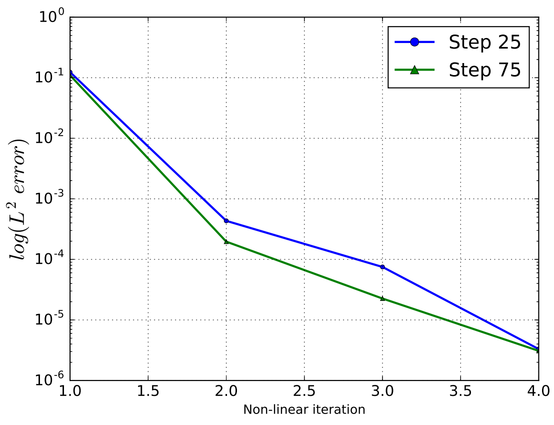
The problem geometry is depicted in figure 4.15. As can be seen in figure 4.15a, the solid geometry consists in a radius $r = 0.05$ cylinder which center is placed at coordinates $(0.2, 0.2)$. Regarding the flag, its thickness is $h = 0.02$ and its length is $l = 0.35$. The flag right bottom corner is placed at coordinates $(0.6, 0.19)$. On the other hand, the fluid domain (figure 4.15b) consists in a rectangular box of length $L = 2.5$ and height



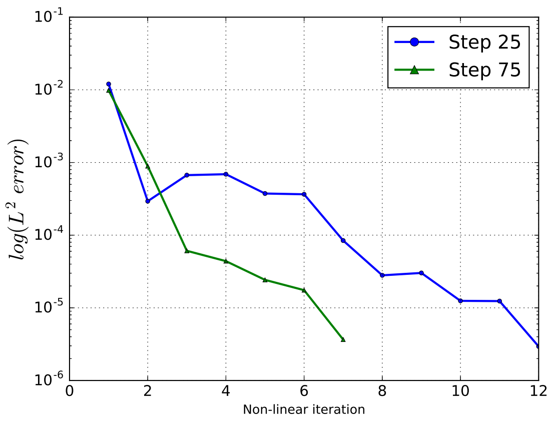
(a) D-N JFNK.



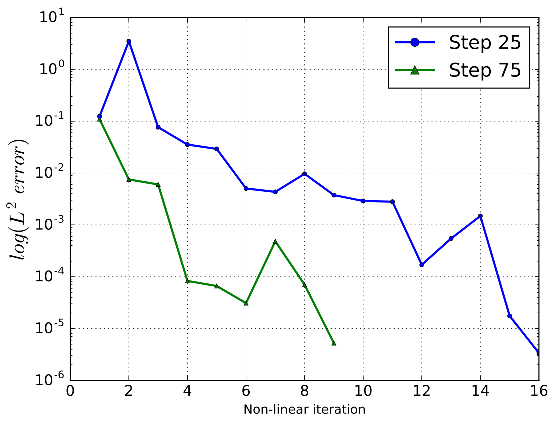
(b) N-N JFNK.



(c) D-N MVQN.

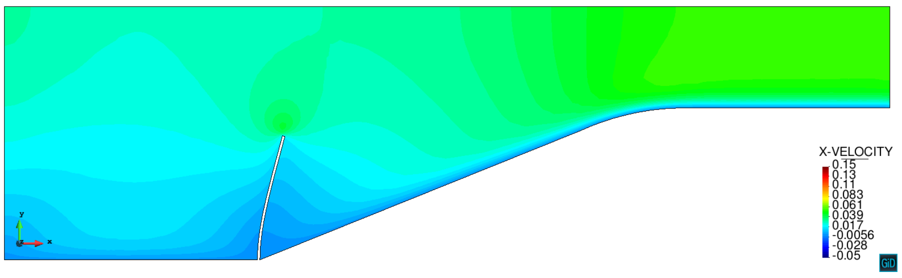


(d) N-N MVQN.

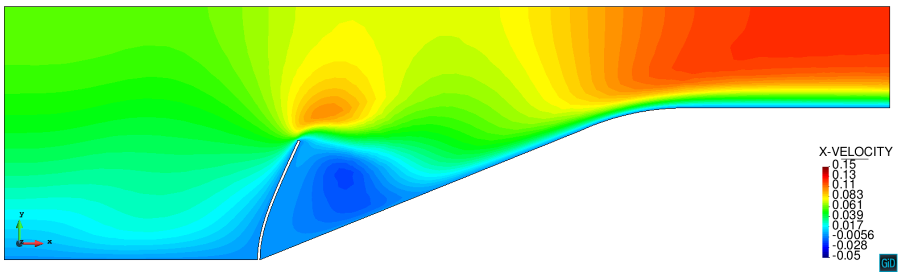


(e) D-N Aitken relaxation.

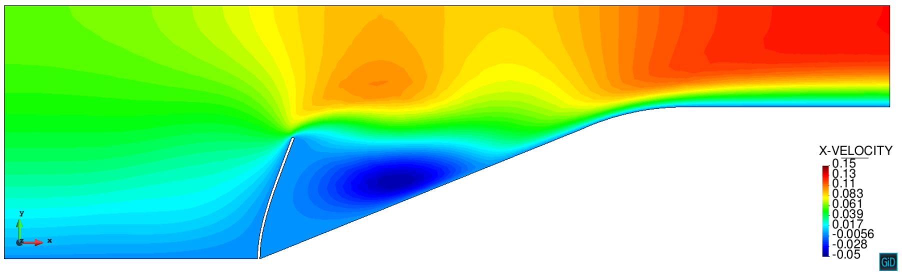
Figure 4.12: Convergence assessment for the JFNK and MVQN methods.



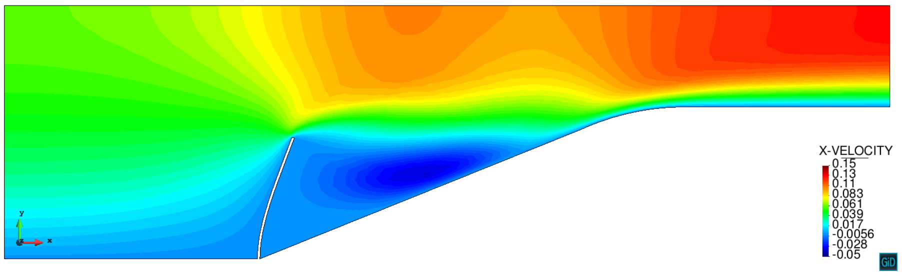
(a) $t = 5\ s$.



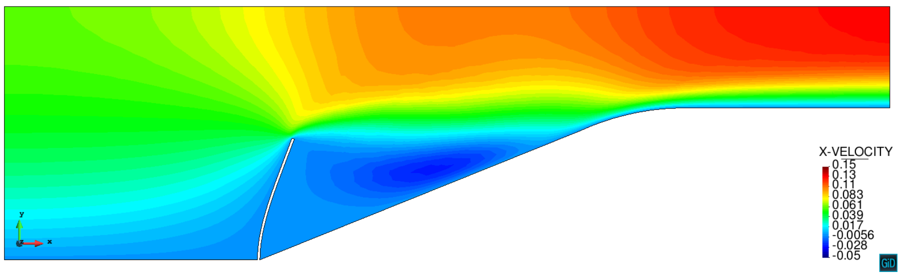
(b) $t = 10\ s$.



(c) $t = 15\ s$.

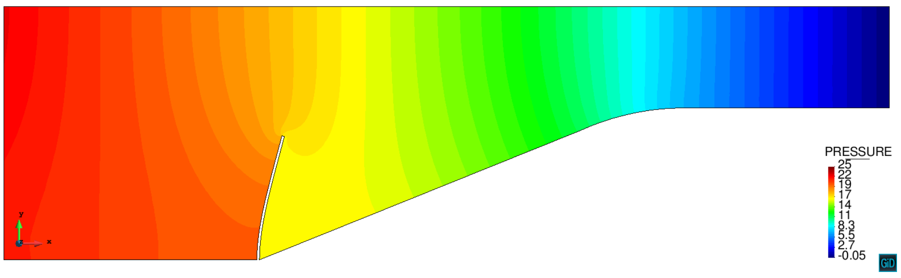


(d) $t = 20\ s$.

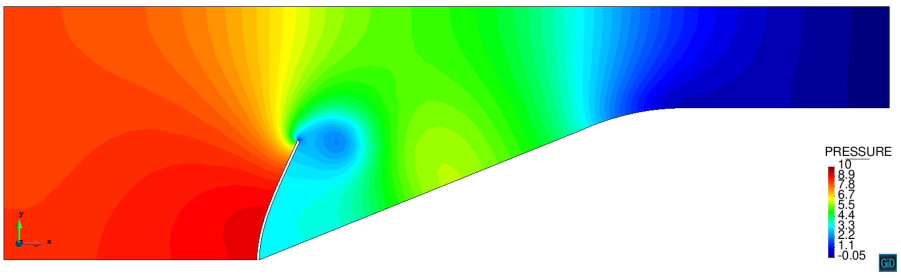


(e) $t = 25\ s$.

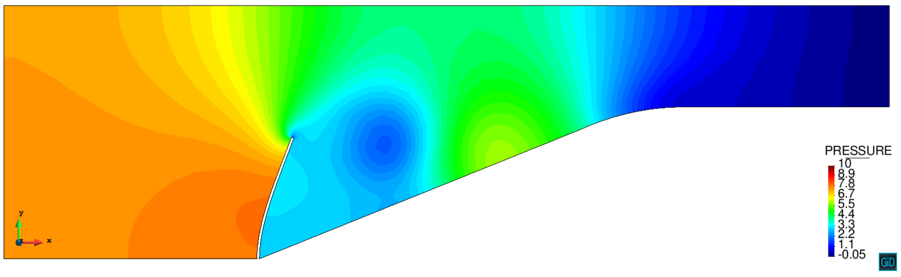
Figure 4.13: Mok test obtained v_x $[m/s]$ fields (real scale deformation).



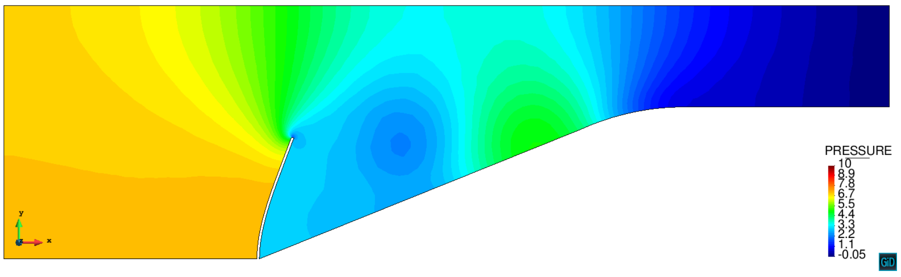
(a) $t = 5\ s$.



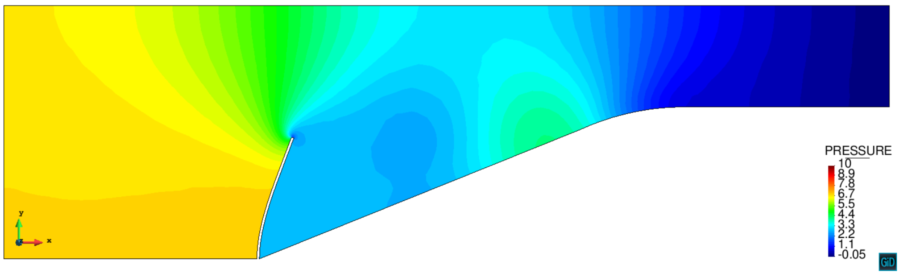
(b) $t = 10\ s$.



(c) $t = 15\ s$.



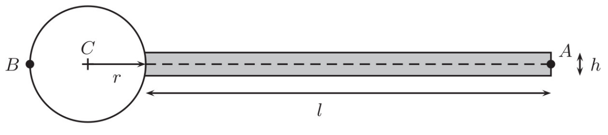
(d) $t = 20\ s$.



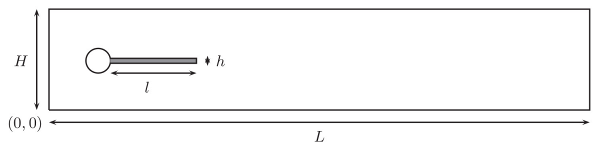
(e) $t = 25\ s$.

Figure 4.14: Mok test obtained p $[Pa]$ fields (real scale deformation).

$H = 0.41$ (all the previous measures are given in meters).



(a) Solid domain.



(b) Fluid domain.

Figure 4.15: Turek-Hron benchmark geometry. Image taken from [43].

Regarding the fluid problem boundary conditions, both top and bottom edges as well as the cylinder are considered as non-slip boundaries while the pressure is set to zero in the right end edge. Besides, a parabolic inlet is imposed in the left edge according to the next equation

$$v(y,t) = 1.5\bar{v}\frac{4}{0.1681}y(0.41 - y) \tag{4.6}$$

where $\bar{v}$ is a time dependent reference velocity such that

$$\bar{v} = \begin{cases} \bar{u}\frac{1-\cos\left(\frac{\pi t}{2}\right)}{2} & if\ t \leq 2 \\ \bar{u} & otherwise \end{cases} \tag{4.7}$$

The fluid is considered to be of Newtonian type and its material parameters are a density $\rho^f = 1000kg/m^3$ and a kinematic viscosity of $\nu^f = 0.001m^2/s$. On the other hand, and hyperelastic Kirchhoff-Saint Venant plane strain constitutive law has been considered in the moving flag. The solid parameters will be discussed later on since they vary depending on the test performed.

Finally, the same elements as the ones considered in the previous tests have been used. Thus, the fluid mesh is made by triangular Q1P1 linear elements while the solid mesh is made by linear quadrilateral large displacement elements. As pointed before, both elements and constitutive laws were already implemented in Kratos.

4.4.2 Model calibration

Due to the complexity of the Turek-Hron benchmark, lots of previous calibration tests, which are also addressed in [43], are required to obtain the expected solution. This subsection collects the results of the calibration tests performed.

CSM problem calibration

According to [43], the computational solid mechanics problem is calibrated introducing a body force $b = [0, -2, 0]$ and studying the displacement evolution of point A (see figure 4.15a). In CSM1 a static problem is solved considering $\rho^s = 1000 kg/m^3$, $\nu = 0.4$ and $E = 1.4e{+}06 N/m^2$. CSM2 has the same material parameters to CSM1 but $E = 5.6e{+}06 N/m^2$. The last test CSM3 is equal to CSM1 test but under dynamic conditions. Tables 4.2, 4.3, 4.4 and 4.5 collect the obtained results of such CSM1, CSM2 and CSM3 calibration tests. For the two CSM1 and CSM2 static tests the displacements u_x and u_y are depicted. On the other hand, for the dynamic test CSM3 the mean value and amplitude of u_x and u_y are analysed. The reference solutions for these values are

- CSM1: $u_x = -0.007187\ m$, $u_y = -0.0661\ m$
- CSM2: $u_x = -0.000469\ m$, $u_y = -0.01697\ m$
- CSM3: $u_x = -0.014305 \pm 0.014305\ m$, $u_y = -0.063607 \pm 0.06516\ m$

Once the CSM tests have been performed, it can be stated that the structural solver settings are in accordance to the ones given by the reference solution. Regarding the mesh convergence study, the mesh number 4 has been selected as reference mesh, since it is the unique that ensures a deviation smaller than a 5%, which has been set as the maximum difference threshold.

CFD problem calibration

As was the case for the solid mechanics problem, the fluid dynamics one needs also to be calibrated. To do that, in [43] there are also collected the CFD1, CFD2 and CFD3 calibration tests. Such tests consist in solving the fluid problem considering the cylinder as well as the flag as rigid objects. Regarding the inlet reference velocities, there are $\bar{u} = 0.2$, $\bar{u} = 1$ and $\bar{u} = 0.2 m/s$ for CFD1, CFD2 and CFD3 respectively.

Table 4.2: CSM1 calibration test results.

Mesh	n_{nodes}	$u_x[m]$	$u_y[m]$	rel. err. $u_x[\%]$	rel. err. $u_y[\%]$
0	93	-0.004204	-0.05078	41.5	23.18
1	205	-0.005766	-0.05935	19.78	10.21
2	427	-0.006468	-0.06283	10.01	4.94
3	819	-0.006798	-0.06441	5.42	2.56
4	1331	-0.006936	-0.06506	3.46	1.57

Table 4.3: CSM2 calibration test results.

Mesh	n_{nodes}	$u_x[m]$	$u_y[m]$	rel. err. $u_x[\%]$	rel. err. $u_y[\%]$
0	93	-0.0002695	-0.012895	42.54	24.01
1	205	-0.0003731	-0.015160	20.46	10.67
2	427	-0.0004203	-0.016092	10.39	5.17
3	819	-0.0004425	-0.016516	5.64	2.68
4	1331	-0.0004520	-0.016692	3.64	1.64

Tables 4.6 and 4.7 collect the solid (cylinder plus flag) drag and lift obtained results. The reference solutions of the tests performed are

- CFD1: Drag:14.29 N, Lift:1.119 N
- CFD2: Drag:136.7 N, Lift:10.53 N
- CFD3: Drag:439.45 ± 5.62 N, Lift:-11.89 ± 437.81 N

As was expected, both the the CFD1 and CFD2 tests drive to a stationary solution in all the meshes considered. Besides, it is interesting to point out that the lift calibration requires much more refinements than the drag one. Hence, at least meshes labelled as 4 and 5 are required in CFD1 test to ensure an lift error below the 5%. For the CFD2 test,at least meshes 7 and 8 are required. Finally, CFD3 was performed using the most refined mesh 8. The obtained results were in accordance to the reference ones.

Table 4.4: CSM3 calibration test results ($\Delta t = 0.02$).

Mesh	n_{nodes}	$u_x[m]$	$u_y[m]$	rel. err. $u_x[\%]$	rel. err. $u_y[\%]$
0	93	-0.008607 ± 0.008607	-0.049869 ± 0.050854	39.83 ± 39.83	21.60 ± 21.96
1	205	-0.011676 ± 0.011677	-0.057788 ± 0.058945	18.35 ± 18.37	9.15 ± 9.54
2	427	-0.013087 ± 0.013087	-0.060965 ± 0.062293	8.51 ± 8.51	4.15 ± 4.40
3	819	-0.013754 ± 0.013755	-0.062395 ± 0.063882	3.85 ± 3.84	1.91 ± 2.05
4	1331	-0.014036 ± 0.014036	-0.062991 ± 0.064460	1.88 ± 1.88	0.97 ± 1.07

Table 4.5: CSM3 calibration test results ($\Delta t = 0.01$).

Mesh	n_{nodes}	$u_x[m]$	$u_y[m]$	rel. err. $u_x[\%]$	rel. err. $u_y[\%]$
0	93	-0.008657 ± 0.008658	-0.049770 ± 0.051164	39.48 ± 39.48	21.75 ± 21.48
1	205	-0.011803 ± 0.011803	-0.057756 ± 0.059403	17.49 ± 17.49	9.20 ± 8.84
2	427	-0.013197 ± 0.013197	-0.060937 ± 0.062695	7.75 ± 7.75	4.20 ± 3.78
3	819	-0.013864 ± 0.013864	-0.062396 ± 0.064191	3.08 ± 3.08	1.90 ± 1.49
4	1331	-0.014137 ± 0.014138	-0.062990 ± 0.064801	1.17 ± 1.17	0.97 ± 0.55

4.4.3 Results assessment

This section collects the results of the performed Turek-Hron FSI tests. It has to be said that only the D-N MVQN approach, which has been proven to be the most computationally efficient one, has been used in this section due to the large computational cost of the presented experiment. By the same reason, the convergence criterion has been kept but the tolerance has been reduced to 10^{-4}. Regarding the mesh solver, the same non-linear pseudo-structural technique has been used again. The comparison magnitudes for the results assessment are the flag tip (point A) displacement as well as the whole body drag and lift. The settings to perform the three Turek and Hron FSI benchmarks are collected in table 4.8.

FSI1 benchmark

According to [43], the FSI1 test can be firstly solved as an initial validation due to its simplicity, which comes from the low Reynolds number considered. Thus, the FSI1 test was firstly performed using a coarse mesh (number 4 in the previous calibration tests)

Table 4.6: CFD1 calibration test results ($\Delta t = 0.01$).

Mesh	n_{nodes}	Drag $[N]$	Lift $[N]$	rel. err. Drag [%]	rel. err. Lift [%]
1	1300	0.136	0.0002	98.8	99.99
2	1769	14.63	1.31	2.38	16.71
3	2339	14.66	0.81	2.59	27.61
4	2906	14.50	1.12	1.47	0.1
5	4615	14.42	1.10	0.91	1.7

Table 4.7: CFD2 calibration test results ($\Delta t = 0.01$).

Mesh	n_{nodes}	Drag $[N]$	Lift $[N]$	rel. err. Drag [%]	rel. err. Lift [%]
2	1769	146.83	1.93	7.08	81.67
3	2339	146.36	0.02	7.06	99.81
4	2906	142.31	2.54	4.18	75.88
5	4615	140.81	8.91	3.00	15.38
6	7101	138.23	12.17	1.11	15.57
7	9795	137.92	9.92	0.89	5.79
8	13573	137.62	10.60	0.67	0.66

and the settings given by [43], which are collected in table 4.8.

Table 4.9 summarizes the results of the FSI1 test. As can be seen, the expected stationary solution was obtained. Comparing the obtained values against the reference ones, the maximum relative error in all the studied magnitudes is less than the 3%. Thus, it can be stated that the FSI implementation is well calibrated and ready to move to the much more requesting FSI2 and FSI3 benchmarks.

FSI2 benchmark

As pointed in [43], the interesting point of FSI2 test is that it must drive to periodic oscillations in both the fluid and the solid domains, despite the fact that CFD2 has a stationary solution. Therefore, FSI2 is specially interesting to check the affectation of the interaction mechanisms.

Because of the expected oscillatory behaviour a much finer mesh is needed to capture

Table 4.8: Turek & Hron FSI tests settings.

Parameter	FSI1	FSI2	FSI3
ρ^s $[kg/m^3]$	1000	10000	1000
ν^s	0.4	0.4	0.4
E $[N/m^2]$	1.4e+06	1.4e+06	5.6e+06
ρ^f $[kg/m^3]$	1000	1000	1000
ν^f $[m^s/s]$	0.001	0.001	0.001
Δt $[s]$	0.01	0.002	0.001
$\bar{u}$ $[m/s]$	0.2	1.0	2.0
Re	20	100	200

Table 4.9: FSI1 reference and obtained control values.

Magnitude	Reference	Obtained	rel. err [%]
$u_{x,A}$ $[m]$	0.0227e-03	0.02331e-03	2.69
$u_{y,A}$ $[m]$	0.8209e-03	0.82336e-03	0.30
Drag $[N]$	14.295	14.497	1.41
Lift $[N]$	0.7638	0.7707	0.90

such phenomenology. Thus, the previously referred refined mesh number 8 has been used. The particular settings to perform the FSI2 test are collected in table 4.8. However, for the sake of reducing the computational cost, the time step given in [43] has been modified to $\Delta t = 0.0025$ s.

Table 4.10 collects the maximum and minimum obtained values for both the drag and the lift. As can be seen, the drag bounds are in very good accordance with the reference values. Regarding the lift, the maximum and minimum values relative errors is a slightly less accurate, being around the 6.5%. On the other hand, table 4.11 collects the maximum and minimum results for the point A displacements. It can be noted that the u_y results are very close to the reference one, being the relative error for both the maximum and minimum values less than the 3%. For the u_x displacement component, the relative errors are larger, with a maximum around the 7%. Taking into account that in the u_x case the absolute error magnitude order is 10^{-4}, these results have been considered to be precise enough.

Table 4.10: FSI2 drag and lift results assessment.

	Drag			Lift		
	Reference $[N]$	Obtained $[N]$	rel. err [%]	Reference $[N]$	Obtained $[N]$	rel. err [%]
max	292.71	304.14	3.90	238.41	253.54	6.35
min	137.41	137.44	0.02	-237.19	-253.35	6.81

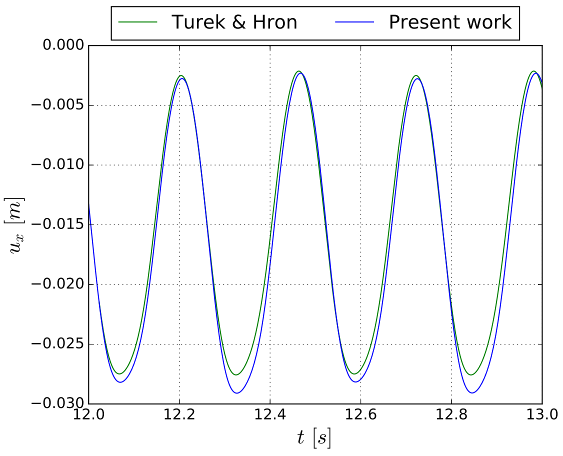
Table 4.11: FSI2 u_x and u_y results assessment.

	u_x			u_y		
	Reference $[m]$	Obtained $[m]$	rel. err [%]	Reference $[m]$	Obtained $[m]$	rel. err [%]
max	-0.00215	-0.00231	7.44	0.0830	0.0852	2.65
min	-0.02755	-0.02910	5.63	-0.0804	-0.0828	2.99

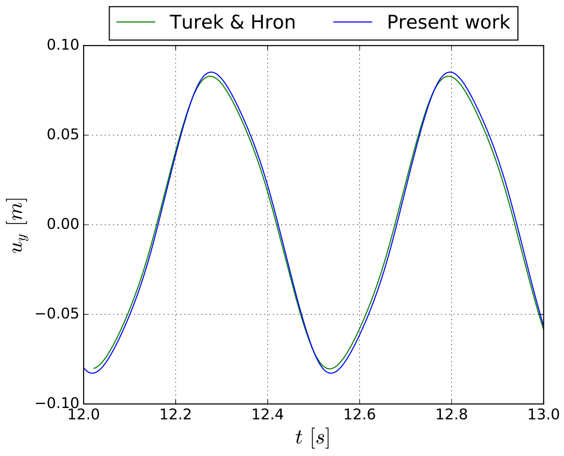
As pointed in [43], the evolution of the control magnitudes must be studied at least for a period of time. Thus, in figure 4.16 the time evolution of these variables is collected. As can be seen, the obtained evolution has barely difference with the reference one, meaning that minor variations only occur at the maximum and minimum values as depicted in tables 4.10 and 4.11. Furthermore, a Fast Fourier Transform (FFT) has been performed using the four control variables data series yielding a characteristic period of oscillation equal to 0.25 s for the u_x and drag data series while it is 0.5 s for the u_y and lift ones. These values represent a relative error of 3.5% and 0% respectively.

Complementary, in figure 4.17 some real scale snapshots of the velocity field are shown for a full period of oscillation. It can be noticed how the vortex shredding yields large oscillations in the elastic flag. Thus, the initial small perturbation coming from considering a deformable flag generates a perturbation in the pressure field, what is to say a vertical force disequilibrium, that generates the body vibration (figure 4.18). Besides, it is interesting to recall that these vortex pattern do not appear if the same Reynolds number ($Re = 100$) is considered in a CFD analysis, showing the relevance that the FSI mechanisms might have in those cases close to the $Re = 90$ limit value.

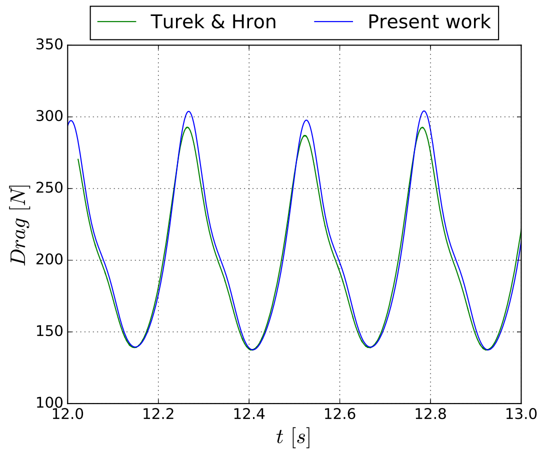
Finally, it is interesting to pinpoint that the resolution of this case has been specially challenging because of the large flag displacements, which in the most finer meshes made the fluid elements close to the flag tip to swap. Hence, it was a difficult task to obtain a mesh fine enough to ensure precision but coarse enough to avoid element swap.



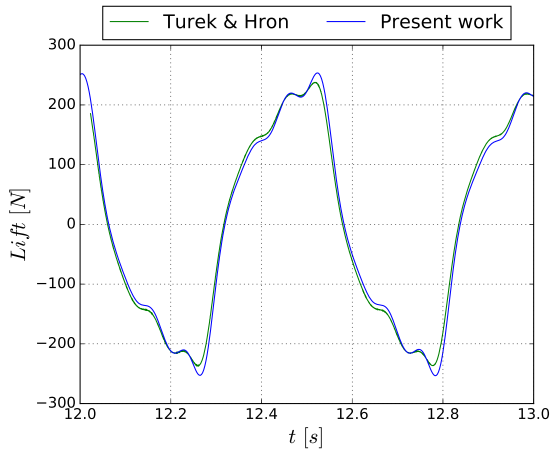
(a) Point A u_x.



(b) Point A u_y.

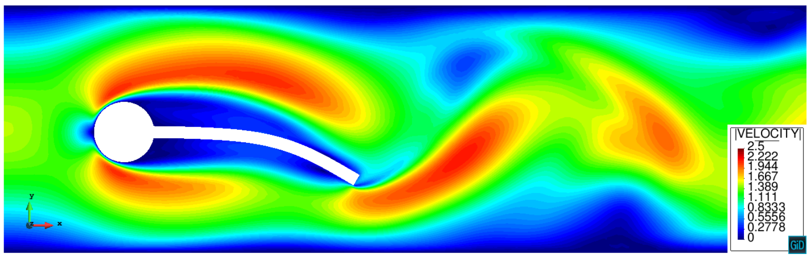


(c) Cylinder and flag total drag.

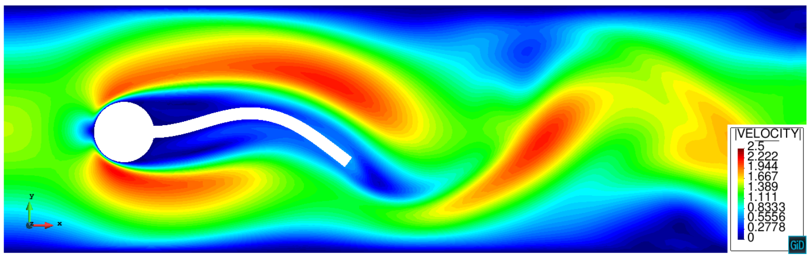


(d) Cylinder and flag total lift.

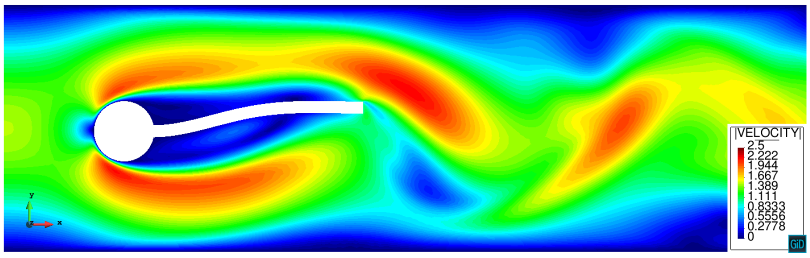
Figure 4.16: FSI2 test results assessment.



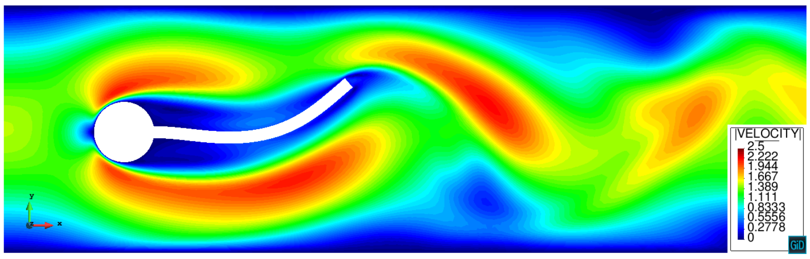
(a) $t = 12.0\ s$.



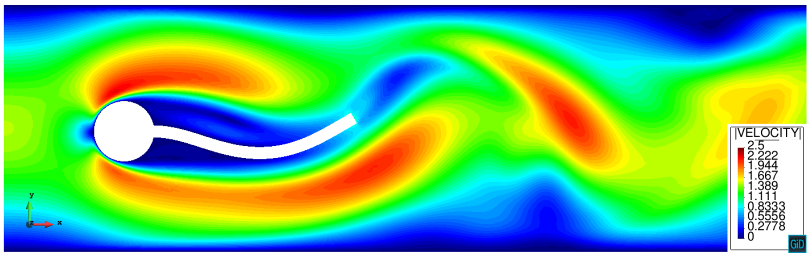
(b) $t = 12.1\ s$.



(c) $t = 12.2\ s$.

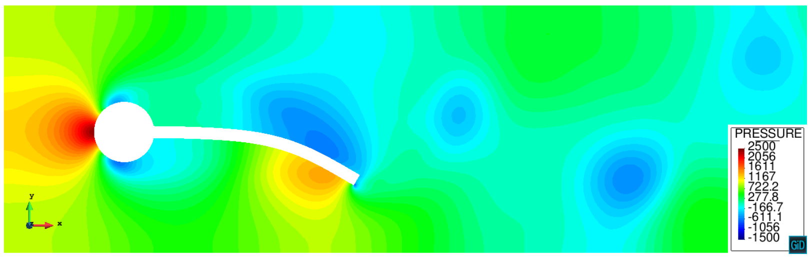


(d) $t = 12.3\ s$.

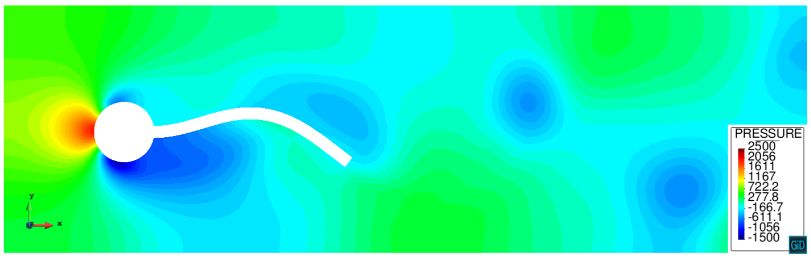


(e) $t = 12.4\ s$.

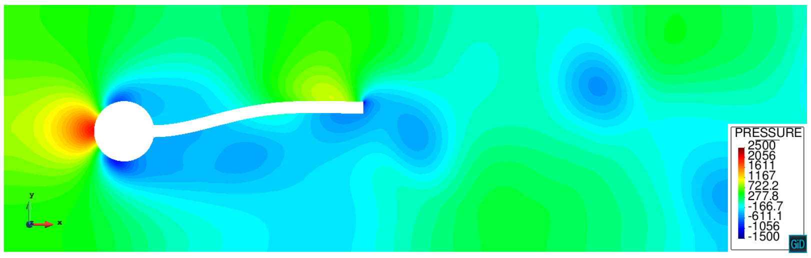
Figure 4.17: FSI2 test $\|v\|$ $[m/s]$ fields for a period of oscillation (real scale deformation).



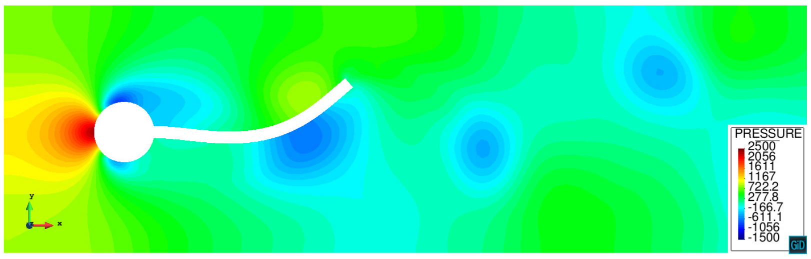
(a) $t = 12.0\ s$.



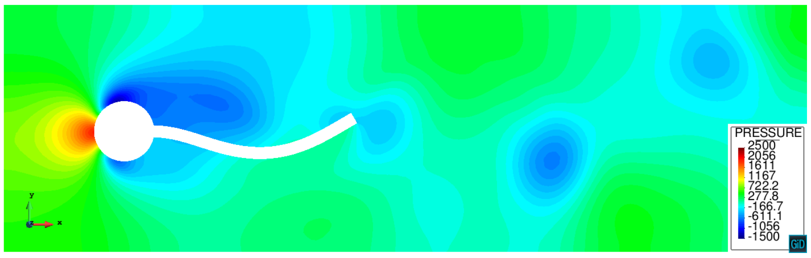
(b) $t = 12.1\ s$.



(c) $t = 12.2\ s$.



(d) $t = 12.3\ s$.



(e) $t = 12.4\ s$.

Figure 4.18: FSI2 test p $[Pa]$ fields for a period of oscillation (real scale deformation).

FSI3 benchmark

Complementary, the FSI3 benchmark has been also performed. As described in [43], FSI3 test must drive to large deformation and complex oscillations. Once again, the FSI3 settings are collected in table 4.8 except the time increment, which has been resized to $\Delta t = 0.002\ s$ for the sake of reducing the computational cost.

Tables 4.12 and 4.13 collect the maximum and minimum values of the comparison magnitudes. As can be noticed in table 4.12 the drag and lift results are almost in perfect accordance with the reference ones, with a maximum relative error in the maximum lift around the 3%. Regarding the displacements of the flag tip, the relative errors of the u_y displacement are around the 5% while the u_x ones are around the 18% and 10% for the maximum and minimum values. Despite these values may seem unacceptable, it has to be taken into account the order of magnitude of the absolute error is 10^{-4}, meaning that the results can be considered as good as the drag and lift ones.

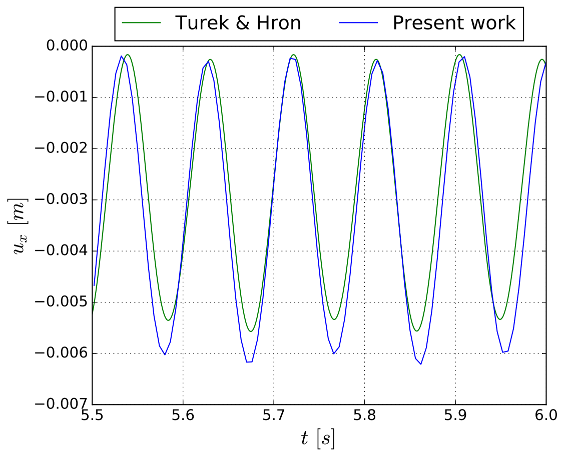
Table 4.12: FSI3 drag and lift results assessment.

	Drag			Lift		
	Reference $[N]$	Obtained $[N]$	rel. err [%]	Reference $[N]$	Obtained $[N]$	rel. err [%]
max	488.24	489.56	0.27	156.41	−161.14	3.02
min	432.76	431.52	0.29	−151.41	−152.14	0.48

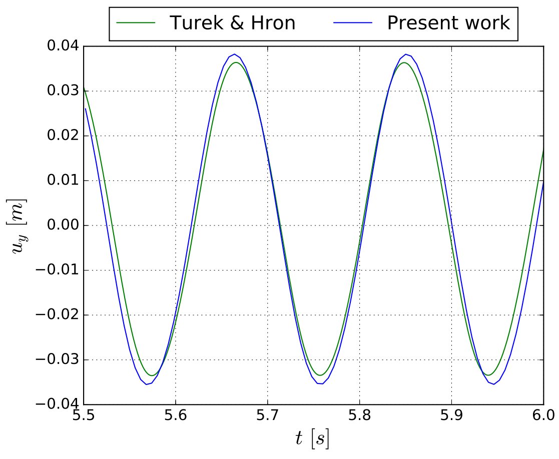
Table 4.13: FSI3 u_x and u_y results assessment.

	u_x			u_y		
	Reference $[m]$	Obtained $[m]$	rel. err [%]	Reference $[m]$	Obtained $[m]$	rel. err [%]
max	−0.00016	0.00019	18.75	0.03646	0.03826	4.94
min	−0.0056	0.0062	10.71	0.03352	0.03551	5.94

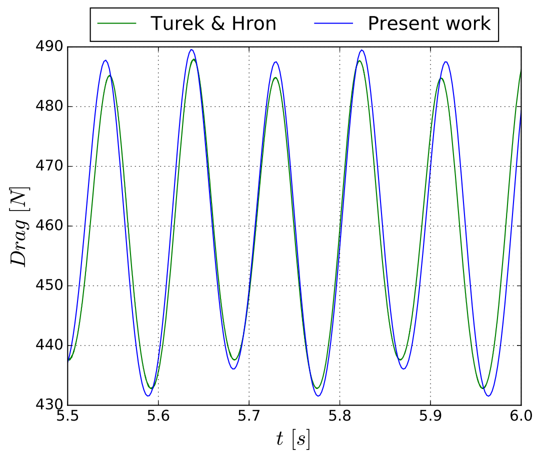
Complementary, figure 4.19 collects the previous results time evolution for a few periods of oscillation. As can be seen, the obtained results time evolutions also match the reference ones. Furthermore, a FFT has been performed with the obtained time evolution results. The obtained period of oscillation for the u_x displacement and the drag is 0.1 s while the u_y displacement and lift one is 0.16667 s. With respect to the reference ones, this represents a 10.5% and 7.5% of relative error. Taking into account



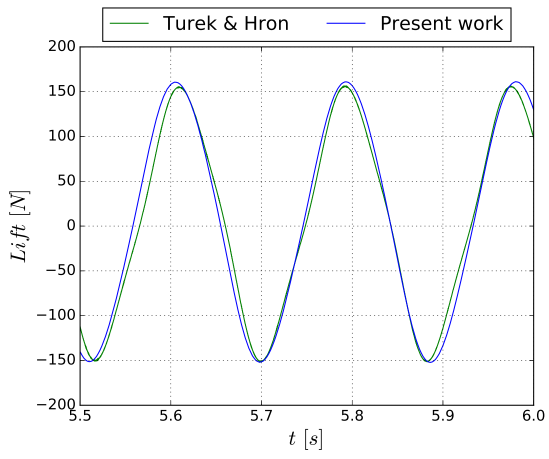
(a) Point A u_x.



(b) Point A u_y.



(c) Cylinder and flag total drag.



(d) Cylinder and flag total lift.

Figure 4.19: FSI3 test results assessment.

that a larger Δt than the original one have been considered and that the FFT may induce some numerical errors, the results have considered to be acceptable.

Besides, it is interesting to point out that no rest of high frequencies were found in the FFT analysis. Thus, it may be interesting to repeat the analysis with a time integration scheme without high frequency dissipation such as the Newmark's, since the Bossak's one introduces numerical damping, which seems to be unnecessary in this case.

Finally, the $\|v\|$ and p contours are shown for a full period of oscillation. As can be seen, the expected periodic flow behaviour appears in the simulation. Besides, it is interesting to compare the FSI3 and FSI2 flap displacement amplitude. As can be seen, the FSI3 flap displacement is much smaller than the FSI2 case, despite the larger

Reynolds number, proving that the amplitude value widely depends in how close the structure and vortex characteristic frequencies are.

4.5 Pressure pulse in compliant vessel

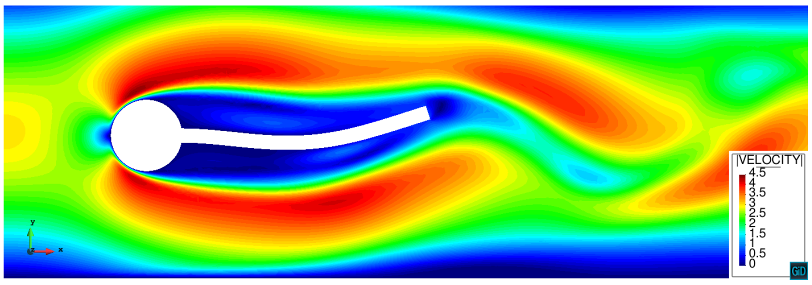
4.5.1 Problem description

So far, all the presented examples were 2D cases. In this section a 3D hemodynamics case is presented to show the further capabilities of the implemented strategies. Due to the large computational cost of solving a 3D case, only the D-N MVQN strategy is taken into account due to its much better performance. Besides, it has to be said that non-matching meshes are also considered, thanks to the possibility of using a 3D mapper already implemented in Kratos.

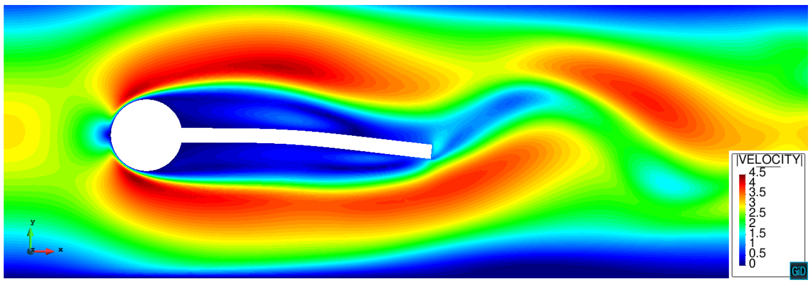
This problem was originally proposed in [32] and later reproduced in [44]. Its aim is to simulate the the fluid-structure interaction arising in the modelling of blood flow in human cardiovascular system. As described in [44], the problem consists of a thin elastic vessel, which in this case has been modelled with a solid shell recently implemented in Kratos by V. Mataix, conveying the blood flow, which is modelled as an incompressible fluid using the Navier-Stokes equations.

It has to be said that the aim of this problem is not to study the real physiological blood flow phenomena but to show the capabilities of the implemented strategies to deal with 3D problems. Furthermore, this problem is also intended to further test the solid shell recently implemented in Kratos.

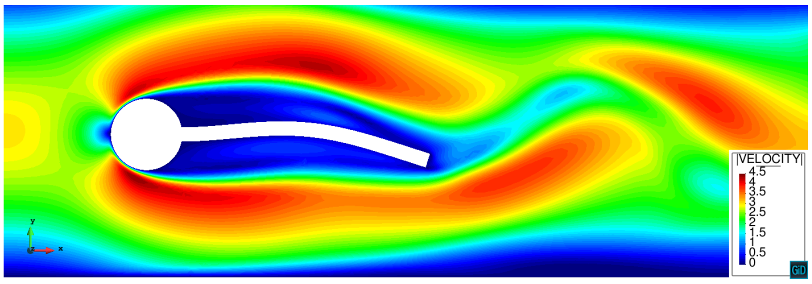
Regarding the geometry, it consists in a straight cylinder of radius $r_0 = 0.005\ m$ which length and thickness are $L = 0.05\ m$ and $t = 0.001\ m$. The blood physical parameters are $\rho^f = 1000\ kg/m^3$ and dynamic viscosity $\mu^f = 0.003\ kg/ms$, yielding a kinematic viscosity $\nu^f = 3e\text{-}06\ m^2/s$. Regarding the solid parameters, the density is $\rho^s = 1200\ kg/m^3$ while the Poisson and Young modulus are $\nu^s = 0.3$ and $E = 3e05\ Pa$. Regarding the boundary conditions, both sides of the vein are clamped (radial displacements allowed) and an overpressure of $p = 1333.2\ Pa$ is imposed at the inlet boundary for $0.003\ s$. Despite that in [44] the pressure pulse is imposed in a sudden manner, it has been decided to introduce it in a more natural way with a sinusoidal function such that



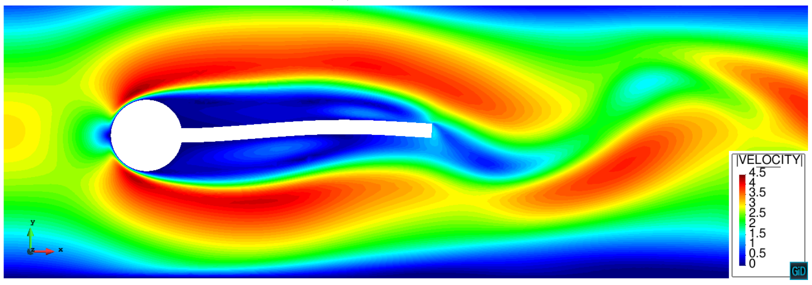
(a) $t = 5.50\ s$.



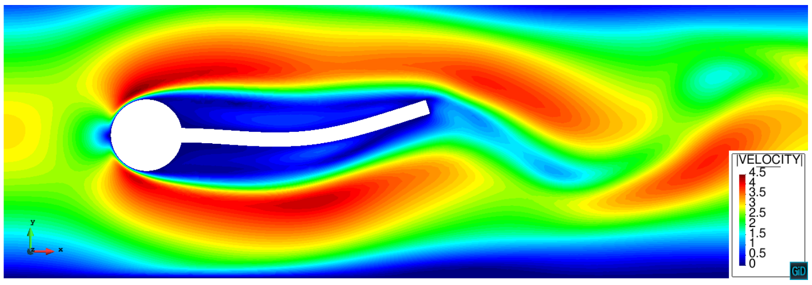
(b) $t = 5.54\ s$.



(c) $t = 5.58\ s$.

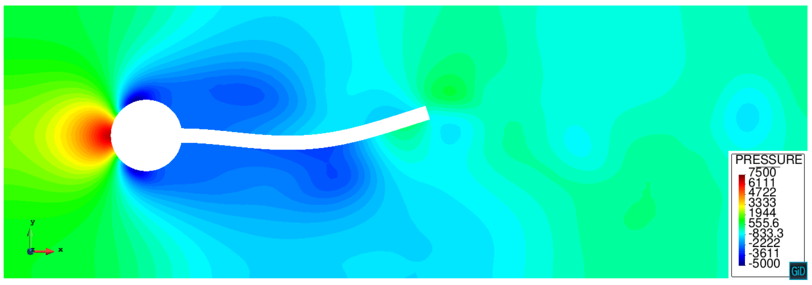


(d) $t = 5.62\ s$.

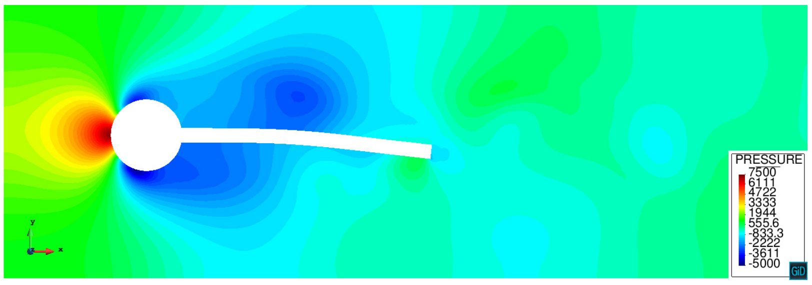


(e) $t = 5.66\ s$.

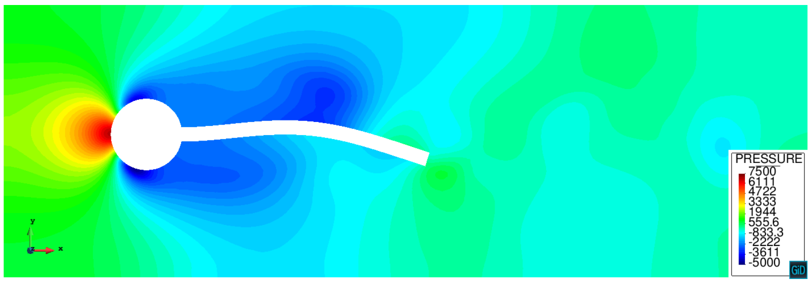
Figure 4.20: FSI3 test $\|v\|$ $[m/s]$ fields for a period of oscillation (real scale deformation).



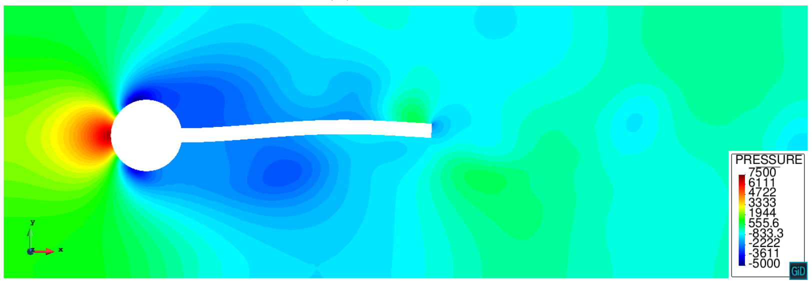
(a) $t = 5.50\ s$.



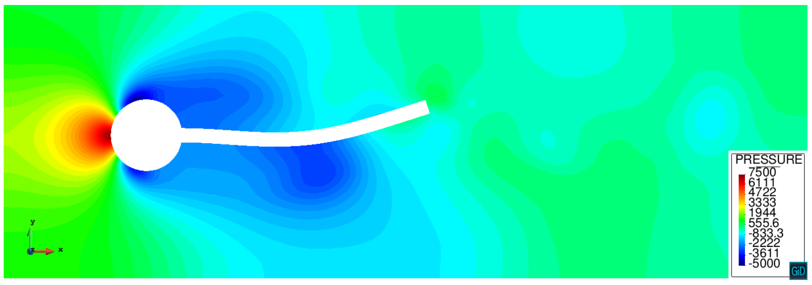
(b) $t = 5.54\ s$.



(c) $t = 5.58\ s$.



(d) $t = 5.62\ s$.



(e) $t = 5.66\ s$.

Figure 4.21: FSI3 test obtained p $[Pa]$ fields for a period of oscillation (real scale deformation).

$$p = \begin{cases} 1333.2 \sin\left(\frac{0.5\pi t}{0.00025}\right) & if\ t \leq 0.00025 \\ 1333.2 & if\ 0.00025 < t \leq 0.00275 \\ 1333.2\left(1 - \sin\left(\frac{0.5\pi(t-0.00275)}{0.00025}\right)\right) & if\ 0.00275 < t \leq 0.003 \\ 0.0 & otherwise \end{cases} \tag{4.8}$$

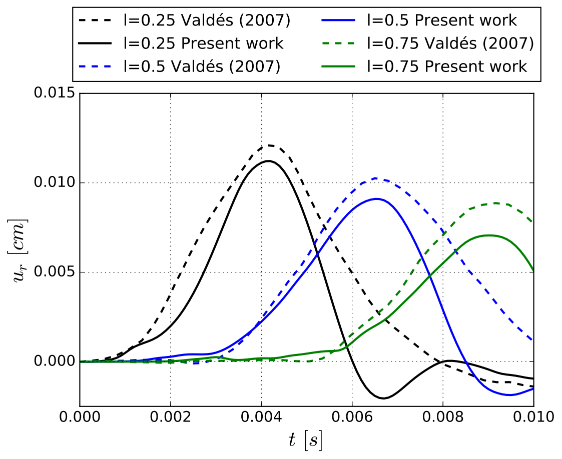
Moreover, it is interesting to point out that in such a symmetric problem it is advisable to use a symmetric mesh in both domains. Otherwise, a little geometrical perturbation in the pressure field may drive to an undesired solution. Thus, a radially symmetric mesh of 3842 nodes has been used in the fluid. In the solid domain, the mesh is also radially symmetric and has 1488 nodes. About the element types, the previously commented solid shell is considered in the fluid while a VMS tetrahedral linear element with same velocity and pressure interpolations has been used in the fluid domain.

Last but not least, the Bossak default scheme in Kratos has been used again for time discretization purposes. The total time of the simulation is $t = 0.01$ while the time increment is $\Delta t = 0.0001$. Besides, the convergence criterion has been changed to a relative one due to the low order values of the problem. Thus, the convergence criterion is $\|\Delta r\|/\|r\| \leq 10^{-4}$.

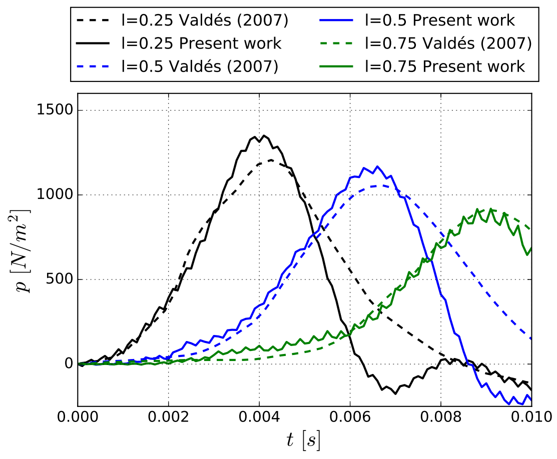
4.5.2 Results assessment

Figure 4.22 collects a comparison between the results in [44] and the obtained ones for three control points placed at $0.25l$, $0.5l$ and $0.75l$, being l the tube length. Regarding the radial displacements (figure 4.22a), it can be seen that the obtained results are similar to the reference ones. The major differences appear after the peak value when the vein section is recovering its shape. Besides, this vein retraction is much clear in the presented solution and can be clearly noted by the negative radial displacements. This behaviour is more similar to real hemodynamics and has been also observed in similar problems in the literature [6].

On the other hand, the pressure evolution is also assessed in figure 4.22b. As can be noted, the pressure trend matches the radial displacements evolution but some oscillations appear in the solution. Regarding the nature of these oscillations, it can be asserted that they are not numeric, since one oscillation is developed in several time steps. Moreover, studying the initial steps solution one can note that in the region close to the outlet, the velocity field is not stationary but points backwards, generating a small wave that is understood to interact to the main pressure wave that is travelling forward, generating the small pressure oscillations mentioned above. Several ideas arise to solve this issue, the most straightforward one is to solve a few CFD steps before the introduction of the pressure pulse and the FSI analysis to avoid the fluid to be completely at rest. An alternative depicted in [6] is to add a simplified 1D version of the presented FSI problem to the outlet. In this way one can have the outlet pressure controlled and its reflection is avoided. Furthermore, one can also implement non-reflecting boundary conditions at the outlet.



(a) Radial displacement.

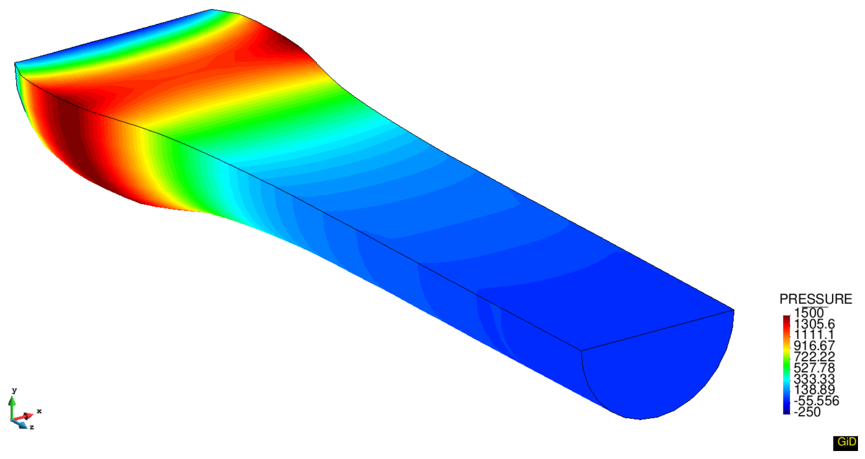


(b) Pressure.

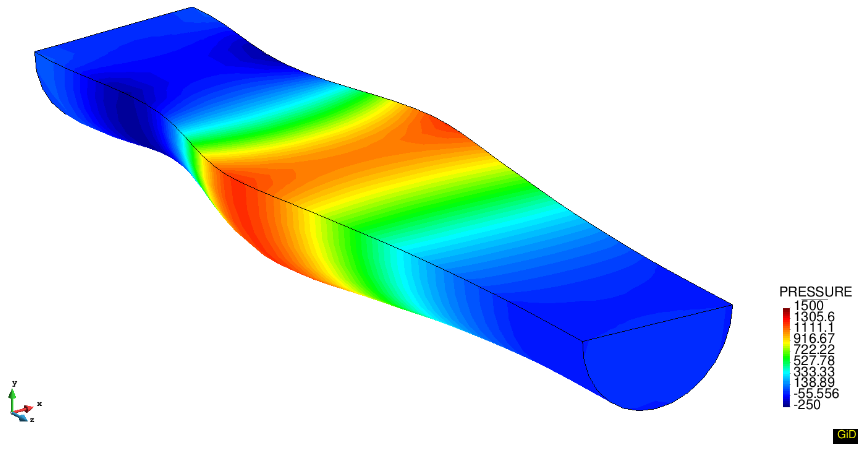
Figure 4.22: Vein flow results comparison.

Taking into account the inherent complexity of the presented problem and the fact that the issues in the solution are not associated to the FSI implementation, it can be asseverated that the presented coupling methodology also works in 3D. In addition, it is interesting to point out the utility that the solid shell elements have in this kind of simulations, since they allow to model thin geometries with a unique layer of elements as is done in shell elements but without double sided surfaces, something that makes the mapping easier.

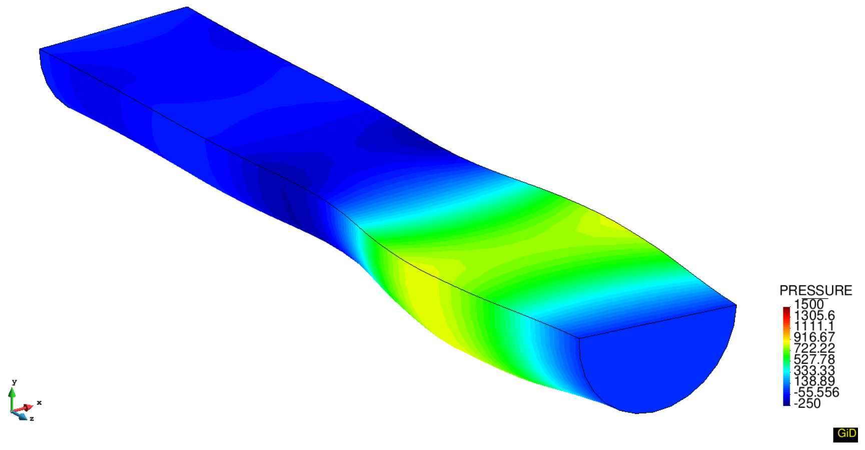
Finally, some snapshots of the obtained coupled solution are shown below. Figures 4.23 and 4.24 show the fluid pressure and velocity contours for $t = 0.003$, $t = 0.006$ and $t = 0.009$. Furthermore, the vein wall displacements fields are shown in figure 4.25 for the same time instants.



(a) $t = 0.003\ s$.

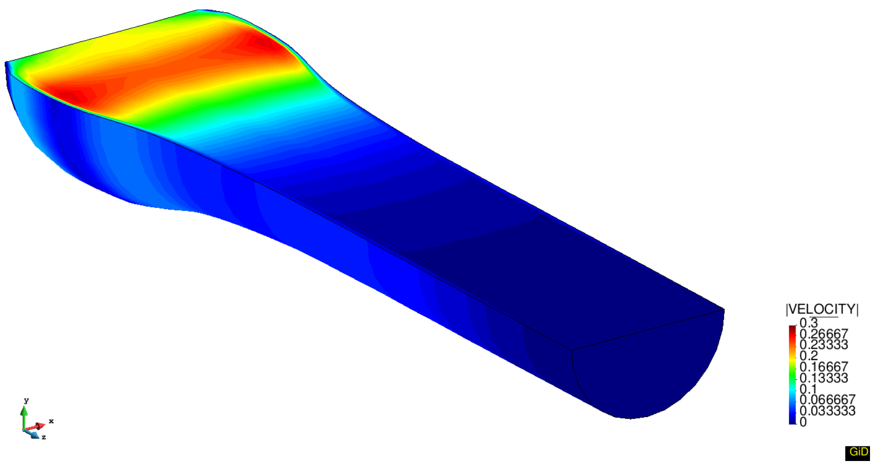


(b) $t = 0.006\ s$.

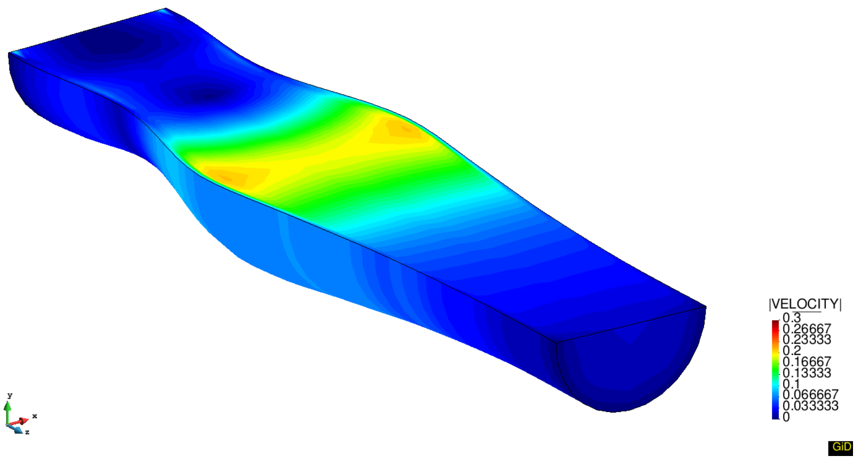


(c) $t = 0.009\ s$.

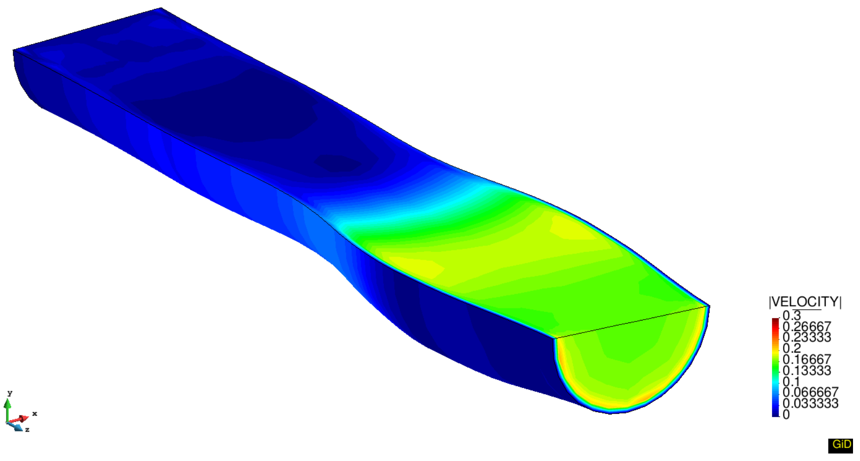
Figure 4.23: Fluid pressure p $[Pa]$ snapshots (deformation scaled 20 times).



(a) $t = 0.003\ s$.

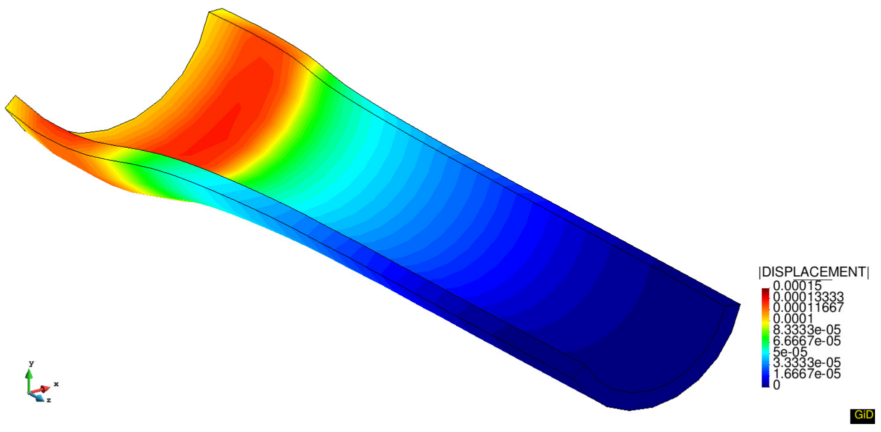


(b) $t = 0.006\ s$.

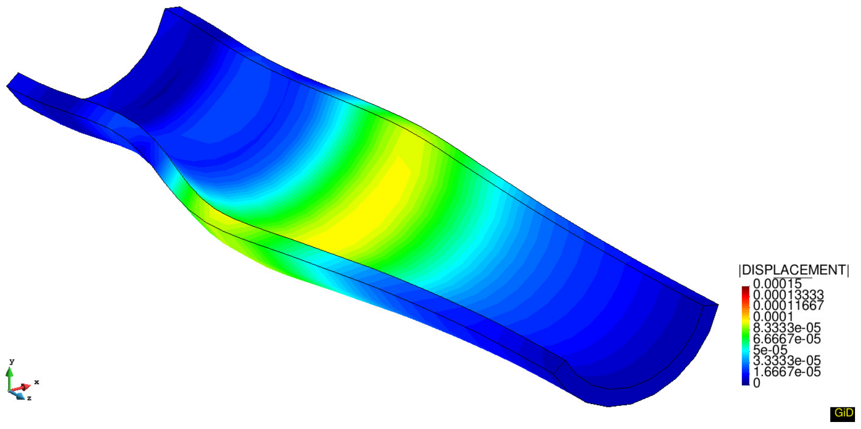


(c) $t = 0.009\ s$.

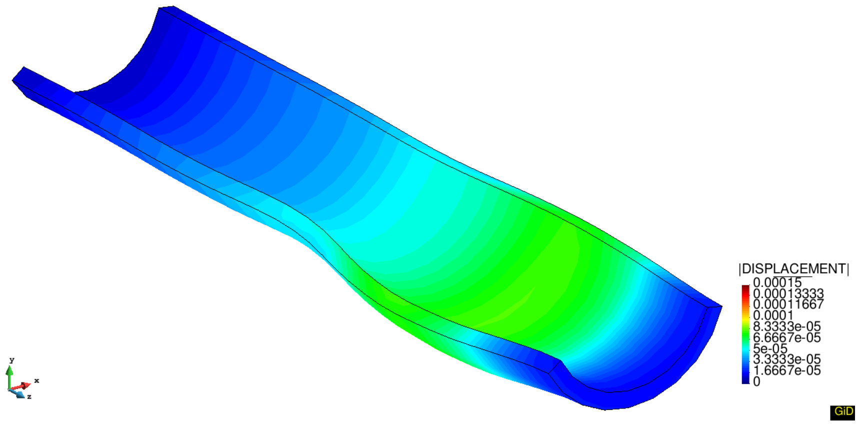
Figure 4.24: Fluid velocity $\|v\|$ $[m/s]$ snapshots (deformation scaled 20 times).



(a) $t = 0.003\ s$.



(b) $t = 0.006\ s$.



(c) $t = 0.009\ s$.

Figure 4.25: Vein wall displacements $\|u\|$ $[m]$ snapshots (deformation scaled 20 times).

Chapter 5

Towards the FSI GiD interface

This chapter tries to summarize the developments carried out in the new Kratos GUI, which has been recently released. Despite the fact that these tasks have a complete different nature to the ones previously described, it is interesting to spend a few words to depict them, since all the FSI strategies developed all along this work are expected to be implemented in the new Kratos GUI in a similar manner.

Before any explanation, it has to be clearly stated that the development of this new interface has been carried out by a group of CIMNE developers. The tasks described in this document are the ones performed by the author. All the developments commented in this sections are already tested and available in Kratos repository. For further information or downloads the reader is referred to Kratos official website [15].

5.1 GiD problem types description

One of the advantages of GiD is its modularity, which avoids to program any new interface from the scratch. Therefore, to program a GUI for any solver only requires the creation of a problem type, meaning that utilities such as geometry or mesh generation are common for all the problem types. In this context, GiD problem types can be viewed as custom modules attached to standard GiD that add extra capabilities to the standard GiD interface to communicate it with any custom code.

Thus, the purpose of a GiD problem type can be roughly summarized in generating the input files in accordance to the requirements of an external solver. Moreover, a problem type must also include the graphical interface custom menus.

In general terms, one can distinguish two types of files in any problem type. The former are the .xml and .spd files, which work in a website programming fashion in the sense that .spd files structure the interface tree and the .xml ones store its information. The latter are the .tcl scripts, which is the main programming language in GiD. These scripts are the ones that load the interface tree contents and write the solver input files when the simulation is run.

5.2 New Kratos GUI

In this section the most relevant new capabilities of the new Kratos GUI are depicted. To do that, it is interesting to list the major modifications with respect to the previous Kratos problem types. Therefore, the most relevant features of the new Kratos GUI are

- the use of JSON string as input data format.
- the use of Kratos processes for the loads and boundary conditions imposition.
- the organization of the model using submodelparts instead of meshes.

In the next subsections each one of these new features is briefly described. Furthermore, the modifications carried out in the original Kratos code to make it fit into the new format are also explained.

5.2.1 JSON string input format

The *JavaScript Object Notation*, commonly referred as JSON, is a human-readable open-standard format for data exchange. JSON files are structured using attribute-value pairs, so its final appearance is similar to a common dictionary. In Kratos context, JSON files are used to collect the so called *ProjectParameters* generated by the interface. The *ProjectParameters*, which used to be a Python file in the past Kratos GUI versions, collect all the simulation settings such as the solving strategy specifications, the linear solver settings or the tolerances.

The main advantage of using a JSON string instead of a Python script to collect the *ProjectParameters* is the much larger information traceability that the JSON format has. It has to be said that the *ProjectParameters* information becomes in a dynamic object when Kratos simulation starts, so it changes during the problem resolution. In case of using a JSON string format, all the original information besides the modifications or additions keeps stored in a unique object. This particular feature is extremely useful for debugging purposes, since the updated JSON file can be entirely printed by console at any moment, something that cannot be done with a Python script.

Finally, it is interesting to point out that the boundary conditions and loads specifications have been also included in the `ProjectParameters.json` file. This has been possible due to the use of Kratos processes to impose any condition in the model. Hence, the `ProjectParameters.json` file also collects the processes settings that Kratos core takes as information to impose the model conditions.

5.2.2 Kratos Processes

The Kratos processes are a collection of subroutines which are introduced in the main scripts to perform any action at any moment of the program execution. Their main goal is to perform specific tasks of a particular simulation without the necessity of modifying the main scripts because of its standard syntax. Hence, its aim is to always keep the main file as it is and to do the user customization via his/her own library of processes.

In programming terms, Kratos processes are Python classes that contain a collection of methods that any Kratos process must have. Each one of these methods is intended to be always executed at a precise point of the simulation and is its content what distinguish one process from another. Besides, all processes must have the same construction function, called `process_factory`, to be consistent between them. The methods that any Kratos process must have are listed below.

- `__init__()`
- `ExecuteInitialize()`
- `ExecuteBeforeSolutionLoop()`
- `ExecuteInitializeSolutionStep()`
- `ExecuteFinalizeSolutionStep()`
- `ExecuteBeforeOutputStep()`
- `ExecuteAfterOutputStep()`

- `ExecuteFinalize()`
- `Clear()`

Despite the fact that processes have been created as a multi-purpose tool, so far they are mainly used in the loads and boundary conditions imposition. As pointed before, processes cannot work only on their own since they need the information contained in the previously described `ProjectParameters.json` file.

Once arrived to this point it has to be clearly stated that if it is possible, it is mandatory to use the general purpose processes already implemented in the Kratos core. If not, the newly developed ones must be always a derived class from them. In this manner the code is much more organised and its maintenance becomes easier. Under this assumption, the next general processes have been programmed

- `impose_scalar_value_process.py`
- `impose_vector_value_by_components.py`
- `impose_vector_value_by_direction_process.py`

which in combination with the already existent main processes

- `apply_constant_scalarvalue_process.h`
- `apply_constant_vectorvalue_process.h`

are enough to prescribe any of the boundary conditions and loads present in the new Kratos GUI. Furthermore, some extra processes have been developed to perform some specific task (e.g. the inlet or outlet imposition in the fluid dynamics application), but in any case these are somehow bridge processes towards the previously listed main ones.

Finally, it is interesting to spend a few words regarding the new concept of *submodelparts* that is supposed to substitute the meshes toolbox in the previous versions of Kratos. Thus, a Kratos *modelpart* is a class that contains all the information required for the simulation of a given problem, namely nodes, elements and conditions. Traditionally, it used to contain one or more *Mesh* instances, containing the nodes, elements and conditions that compose the problem domain. This *Mesh* concept have been substituted in the last release version by the *submodelparts*, which have the advantage of being derived from the main model part, meaning that they inherit its features by definition. Therefore, processes can be viewed as a tool to apply any modification in a specific *submodelpart*, which name is specified along with the process settings in the `ProcessPrarameters.json` file.

5.2.3 New solid and fluid solvers

As can be guessed, the previously commented new features have had implications in the existent Kratos solvers. Thus, it has been necessary to programme new fluid dynamics as well as solid mechanics solvers in accordance to the new `ProjectParameters.json` and model part (.mdpa) files.

Thus, two new Navier Stokes equations solvers have been created, one for the monolithic approach and another for the fractional step approach. For the solid mechanics case, a main implicit dynamic solver has been created. From this implicit solver, an explicit dynamic solver as well as a static one have been derived. Note that the concept of reducing the code maintenance has been taken into account again. The list of the newly developed solvers is related below.

- `navier_stokes_solver_fractionalstep.py`
- `navier_stokes_solver_vmsmonolithic.py`
- `solid_mechanics_implicit_dynamic_solver.py`
- `solid_mechanics_explicit_dynamic_solver.py`
- `solid_mechanics_static_solver.py`

It has to be said that all this solvers are in fact an adaptation of the old ones, which are still present in Kratos repository, to the newly developed JSON and submodelparts formats.

5.3 Kratos FSI interface draft

Before the description of the future FSI Kratos GUI it is interesting to point out which have been the main bottlenecks in the creation of an FSI simulation using the old Kratos problem type.

So far, the creation of an FSI model implies to generate two independent models, one for the solid mechanics domain and another one for the fluid dynamics one. Apart of the fact that this requires to work with two separated GiD projects at the same time, this approach has further disadvantages. The most important one is that the interface between subdomains cannot be identified in an automatic way. Therefore, one must do it by hand or in a tricky way (e.g. impose a load in those nodes belonging to the interface to highlight them in de .mdpa file and then suppress the load). It is obvious that this manner is neither comfortable nor intuitive for the user and may induce lots of errors in the model generation.

Moreover, if an ALE formulation is required in the fluid domain, the mesh boundary conditions imposition is neither automatic. Thus, it has to be done in a weird manner similar to the current interface definition.

Furthermore, it has to be said that the all the FSI solvers developed during this work are still programmed in the old format, meaning that they are to be modified to consider the JSON input format as well as the Kratos submodelparts.

Therefore, the main requirements of the FSI GUI that is to be developed are

- To use the newly developed `ProjectParameters.json` and the submodelparts technology. Besides, the main script must include the processes instructions.
- To be able to set the Kratos flag `INTERFACE` at the interface of both the solid and fluid domains.
- To join the fluid and solid GiD models in a unique model.

At the moment, the FSI interface is under development but so far it is clear that the solid mechanics as well as fluid dynamics applications interfaces that have been recently released will be included in the FSI one. Therefore, the FSI interface will have three main levels, the solid and fluid application ones besides a new one, which will include the FSI parameters such as the coupling scheme or the interface residual minimization strategy.

In addition, it is under study the implementation of a wizard utility. This wizard utility has the purpose of guiding the user in the creation of the FSI model. In general terms, it would have the next steps:

1. Creation of the structure: geometry and B.C. definition, meshing and interface identification.

2. Creation of the fluid domain: geometry and B.C. definition as well as ALE mesh boundary conditions (if proceeds). Meshing and interface identification.

3. FSI settings: selection of the FSI strategy and its settings.

As can be noted, the main idea of this wizard is that once the user has completed it, he/she gets a complete FSI model ready to be computed.

Chapter 6

Conclusions

The aim of this work was to develop an FSI coupling environment able to deal with strongly coupled large deformation problems. To do that, the optimized and widely tested fluid dynamics and solid mechanics solvers already implemented Kratos Multiphysics framework have been used as black boxes in the resolution of the FSI problem. As a consequence, only partitioned black-box FSI solvers have been considered. In this context, the next FSI coupling strategies have been implemented and tested

- Dirichlet-Neumann scheme with Jacobian Free Newton-Krylov minimization
- Dirichlet-Neumann scheme with Multivector Quasi-Newton minimization
- Dirichlet-Neumann scheme with Aitken relaxation
- Neumann-Neumann scheme with Jacobian Free Newton-Krylov minimization
- Neumann-Neumann scheme with Multivector Quasi-Newton minimization

Among all of the previous interface residual minimization strategies, the MVQN method has proved to be the most efficient one, leading to a reduction between 50 and 80% of the computational cost. Such overwhelming performance comes from its reduced computational cost per non-linear iteration, which requires only one residual evaluation, while the JFNK requires one per inner Krylov solver (GMRES) iterations. Comparing with the Aitken relaxation scheme, the MVQN method uses information from all the previous non-linear iterations while the Aitken relaxation uses only the two previous ones. Despite this, all the strategies have been successfully tested in strongly coupled problems, where the fluid exerts large displacements in the structure.

However, it has to be said that the MVQN method is slightly less robust than the JFNK one, because it does not ensure convergence by definition due to its Quasi-Newton nature. Besides, it has been observed to be more sensitive to the tolerance selection, meaning that, if the coupling and inner solver tolerances are not selected properly, the method might start to struggle around the convergence threshold without overcoming it. This has been also observed in the selection of the Krylov solver tolerance in the JFNK approach. Consequently, these strategies require more previous experience to be used.

Furthermore, the Aitken relaxation scheme deserves to be highlighted. Its well-behaviour in complex problems such as the Mok benchmark has been proved. Thus, the easiness in its implementation joined to its sufficiently good performance, makes the Aitken relaxation a good option to be considered as a reference in the implementation of other methods, as was done in this work, or in the resolution of simple enough problems.

Moreover, it is interesting to spend a few words about the implementation of each one of the minimization strategies considered. As pointed before, the implementation of the Aitken relaxation can be done straightforwardly. The main difficulty in the MVQN method implementation is its comprehension, which becomes a bit fuzzy at the initial stage. Once it has been fully understood, it only involves array updates and operations. Regarding the JFNK approach, its implementation has been the toughest one, firstly because the theoretical concepts required for its comprehension and secondly due to the fact that it requires the use of advanced programming tools such as the Python `LinearOperator`.

Regarding the coupling schemes, the D-N scheme has proved to be more efficient than the N-N one, specially at the very beginning of the problem where the N-N requires much more iterations to reach the equilibrium state. Despite this behaviour was completely expected, it is interesting to recall that the N-N coupling can be useful in those situations in where the D-N scheme does not manage to converge, since the N-N scheme imposes the interface equilibrium explicitly, while the D-N does it as a consequence of the velocity imposition. Besides, the presented N-N algorithm is able to work in parallel, something that has not been done in this work but would improve its performance so much.

On the other hand, the limitations of the ALE approach used in the fluid domain have been proved. Hence, when the structure displacements are extremely large (e.g. FSI2 Turek& Hron benchmark) the mesh movement might yield inverted elements. In this context, the newly developed embedded FSI techniques arise as a promising solution for this kind of problems. Furthermore, it has been also observed that pseudo-structural mesh updating techniques work better than the Laplacian ones, since a better interface tracking in such large displacement cases is obtained.

To sum up, it can be asseverated that the presented partitioned black-box coupling techniques are an extremely good alternative to the monolithic approaches if one wants to take profit from widely tested or even commercial fluid and solid solvers in the resolution of an FSI problem. Besides, it has been proved that the presented partitioned schemes can deal with strongly coupled large displacement problems as monolithic solvers do, but with the advantage of requiring much less time and human effort in its development and testing.

6.1 Achievements

In this work a black-box FSI solving environment has been developed in Kratos Multiphysics. The implementation of this FSI environment concerns three different strategies (Aitken relaxation, Jacobian Free Newton-Krylov and Multivector Quasi-Newton method) as well as two coupling schemes (Dirichlet-Neumann and Neumann-Neumann).

All the previous FSI strategies have been satisfactory tested by means of the well-known cavity and Mok benchmark problems, showing perfect correlation with the expected results. During this testing stage, special interest was put in the computational efficiency assessment.

After this initial study, the recently developed D-N MVQN approach became the reference strategy due to its much efficient behaviour. Then, the Turek& Hron FSI benchmark problems, which are widely known because of their complexity, were solved using the D-N MVQN technique as final 2D test. In all the cases, the obtained results have extremely good correlation with the ones present in the literature.

Besides, a preliminary hemodynamics 3D test has been also performed. Despite the fact that the results does not adjust perfectly to the ones in the literature, the black-box FSI resolution environment has proved to work also in 3D. Taking into account that the differences in the solution are associated with some issues in the fluid domain problem, the results can be considered as satisfactory.

Complementary, this work also involved a contribution in the development of the new solid mechanics and fluid dynamics Kratos GiD interfaces. These two interfaces are updated to the last Kratos released version and available to download. Thanks to this task, a knowledge concerning the GiD problem types programming have been acquired.

Finally, it is interesting to point out that all the developments of this work will remain in Kratos Multiphysics repository, which is open source and available to download.

6.2 Future work-lines

The most imminent work to be done is the creation of the FSI GiD interface. At the moment, the generation of a FSI simulation implies to create two separated fluid and solid problems and to carry out some extra tasks to define the coupling. Thus, to set up a FSI model becomes in a tedious and time consuming task that must be optimized to be much more user-friendly.

Moreover, the majority of code developments have been done at the Python level of Kratos. For the sake of efficiency, it is reasonable to move them to the C++ level which will perform faster.

Regarding the presented methodologies, it could be interesting to do a deeper assessment of the implications of the small perturbation ε used in the linearisation of the Jacobian approximation in the JFNK approach. It is understood that this parameter widely affects the Jacobian approximation and in consequence to the convergence of the method. The performance of the N-N JFNK has been surprisingly poor. Then, the parallelization of the residual evaluation of the N-N scheme remains as a further improvement to be done.

On the other hand, to implement a remeshing strategy along with the ALE mesh movement technique arises as a good option to avoid excessive distortion in the fluid elements.

Taking into account that the extension of the presented methodologies to the 3D case is more or less simple and that the resolution of a 3D case implies much more computational effort in the residual evaluation, the majority of tests in this work are two-dimensional. However, it is a must to do a further assessment of the capabilities and performance of the presented strategies in some extra 3D benchmarking problems such as the 3D version of the presented Turek & Hron benchmark.

Besides, it could be interesting to search for a benchmark problem in where the D-N scheme would not manage to converge to try the expected robustness of the N-N one.

Appendix A

Developed code

In this appendix the most important and representative Python and C++ files developed during this work are collected.

A.1 MAIN_FILE_FSI.py

```
# Import libraries
import numpy
import scipy
import scipy.sparse
import scipy.sparse.linalg
import time as timemodule
import json
# Import utilities
import connectivity_mapper
import residual_definitions
import mvqn_strategy
import jfnk_strategy
import relaxation_strategy
import KratosMultiphysics.FSIApplication
# Import solvers
import FluidProblemClass
import SolidProblemClass
# Import ProjectParameters
import ProjectParametersFluid
import ProjectParametersSolid

# Initial checks
if ProjectParametersFluid.domain_size != ProjectParametersSolid.domain_size:
    raise("ERROR: Different working dimensions among subdomains!")
if ProjectParametersFluid.Dt != ProjectParametersSolid.time_step:
    raise("ERROR: Different time step among subdomains!")
if ProjectParametersFluid.nsteps != ProjectParametersSolid.nsteps:
    raise("ERROR: Different number of time steps among subdomains!")
if ProjectParametersFluid.max_time != ProjectParametersSolid.end_time:
    raise("ERROR: Different final time among subdomains!")
if ProjectParametersFluid.output_time != ProjectParametersSolid.GiDWriteFrequency:
    raise("ERROR: Different output time among subdomains!")

# Stepping and time settings
domain_size = ProjectParametersFluid.domain_size
Dt = ProjectParametersFluid.Dt
```

```python
Nsteps = ProjectParametersFluid.nsteps
final_time = ProjectParametersFluid.max_time
output_time = ProjectParametersFluid.output_time
time = ProjectParametersFluid.Start_time

# Solid and fluid problem construction
D1_problem = FluidProblemClass.FluidProblem(ProjectParametersFluid)
D2_problem = SolidProblemClass.SolidProblem(ProjectParametersSolid)

# Solid and fluid problem initialization
print("Fluid and solid problems initialization...")
D1_problem.Initialize()
D2_problem.Initialize()
print("Fluid and solid problems initialization finished.")

# Conditions initialization
print("Fluid and solid problem dependent conditions set up starts...")
D1_problem.InitializeConditions()
D2_problem.InitializeConditions()
print("Fluid and solid problem dependent conditions finished.")

# Fluid interface is taken as reference interface
Interface_pb_size=D1_problem.interface_nodes

# Interface communicator construction
if domain_size == 2:
    # Fetch the solid and fluid interface nodes
    fluid_interface_nodes=D1_problem.interface_nodes_vec
    solid_interface_nodes=D2_problem.interface_nodes_vec

    # Check whether the interface nodes match or not
    if D1_problem.interface_nodes != D2_problem.interface_nodes:
        raise("ERROR: Different number of interface nodes among subdomains!.")

    # Construct the 2D conformant interface mapper
    print("2D interface communicator construction starts...")
    wet_interface_comm = connectivity_mapper.interface_communicator(fluid_interface_nodes,
                                                                    solid_interface_nodes)
    print("2D interface communicator successfully constructed.")

# Output initialization
print("Output initialization...")
D2_problem.InitializeOutput()
D1_problem.InitializeOutput()
print("Output initialization finished.")

# Interface residual construction
coupling_algorithm = "DirichletNeumann"

print("Interface residual construction starts...")
if coupling_algorithm == "DirichletNeumann":
    residual = residual_definitions.DirichletNeumannResidual(D1_problem,
                                                             D2_problem,
                                                             wet_interface_comm)
    print("Dirichlet-Neumann residual constructed.")

elif coupling_algorithm == "NeumannNeumann":
    residual = residual_definitions.NeumannNeumannResidual(D1_problem,
                                                           D2_problem,
                                                           wet_interface_comm)
    print("Neumann-Neumann residual constructed.")

# Interface strategy construction
coupling_strategy = "Relaxation"

print("Interface coupling strategy construction starts...")
if coupling_strategy == "MVQN":
    interface_strategy = mvqn_strategy.MultiVectorQuasiNewtonStrategy(Interface_pb_size,
                                                                      domain_size,
                                                                      D1_problem,
                                                                      D2_problem,
                                                                        wet_interface_comm,
                                                                        residual)
    print("MultiVector Quasi-Newton strategy constructed.")
elif coupling_strategy == "JFNK":
    interface_strategy = jfnk_strategy.JacobianFreeNewtonKrylovStrategy(Interface_pb_size,
                                                                          domain_size,
                                                                          D1_problem,
                                                                          D2_problem,
                                                                          wet_interface_comm,
                                                                          residual)
    print("Jacobian Free Newton-Krylov strategy constructed.")
elif coupling_strategy == "Relaxation":
    if coupling_algorithm == "DirichletNeumann":
        interface_strategy = relaxation_strategy.RelaxationStrategy(Interface_pb_size,
                                                                    domain_size,
                                                                    D1_problem,
                                                                    D2_problem,
                                                                    wet_interface_comm,
                                                                    residual)
        print("Relaxation strategy constructed.")
    elif coupling_algorithm == "NeumannNeumann":
        raise("ERROR: Relaxation strategy must be used with Dirichlet-Neumann algorithm.")
else:
    raise("ERROR: Interface strategy not implemented yet!")

# Output files
filename = "MAIN_FILE_FSI_"+coupling_algorithm+"_"+coupling_strategy+".log"

# .log File creation to store the iterations evolution
with open(filename, 'w') as file:
    file.write("Interface problem size: "+str(Interface_pb_size)+"\n")
    file.write("Interface residual size: "+str(Interface_pb_size*domain_size)+"\n"+"\n")
    file.close()

# NL solver parameters
max_nl_iterations = 50
nl_tol = 1e-5

guess_value = 0.0001*numpy.ones(Interface_pb_size*domain_size,dtype='float')

out = 0
step = 0

print("COUPLED PROBLEM RESOLUTION STARTS...")
print("Interface problem size: ",Interface_pb_size)

while(time <= final_time):

    time = time + Dt
    step = step + 1
    convergence = False

    D1_problem.ExecuteInitializeSolutionStep(time)
    D2_problem.ExecuteInitializeSolutionStep(time,step)

    interface_strategy.ExecuteInitializeSolutionStep()

    print("STEP = ", step)
    print("TIME = ", time)

    with open(filename, 'a') as file:
        file.write("STEP: "+str(step)+"\n")
        file.write("TIME: "+str(time)+"\n")
        file.close()

    for nl_it in range(1,max_nl_iterations+1):
        print("    NL-ITERATION ",nl_it,"STARTS.")

        print("     Residual computation starts...")
        vel_residual = residual.ComputeResidual(guess_value)
        nl_res_norm = scipy.linalg.norm(vel_residual)
        print("     Residual computation finished. |res|=",nl_res_norm)

        ### CONVERGENCE ACHIEVED ###
        if nl_res_norm < nl_tol:

            convergence = True

            print("    CONVERGENCE ACHIEVED")
            print("    Total non-linear iterations: ",nl_it," NL residual norm: ",nl_res_norm)

            with open(filename, 'a') as file:
                file.write("    Non-linear iteration summary:\n")
                file.write("        nl_it: "+str(nl_it)+"\n")
                file.write("        nl_res_norm: "+str(nl_res_norm)+"\n"+"\n")
                file.write("    CONVERGENCE ACHIEVED\n"+"\n")
                file.close()

            break

        ### CONVERGENCE NOT ACHIEVED ###
        else:

            iteration_corrected_value = interface_strategy.InterfaceSolutionUpdate(step,
                                                                                    nl_it,
                                                                                    guess_value,
                                                                                    vel_residual)
            guess_value = numpy.copy(iteration_corrected_value)

            with open(filename, 'a') as file:
                file.write("    Non-linear iteration summary:\n")
                file.write("        nl_it: "+str(nl_it)+"\n")
                file.write("        nl_res_norm: "+str(nl_res_norm)+"\n"+"\n")
                file.close()

    # Solve the mesh movement
    solid_interface_disp = D2_problem.GetInterfaceDisplacement()
    solid_interface_disp_comm = wet_interface_comm.StructureToFluid_VectorMap(solid_interface_disp)

    D1_problem.SolveMesh(solid_interface_disp_comm)

    # Print results
    if(output_time <= out):
        out = 0

    out = out + Dt

    D1_problem.ExecuteFinalizeSolutionStep(time,output_time,out)
    D2_problem.ExecuteFinalizeSolutionStep()

    interface_strategy.ExecuteFinalizeSolutionStep()

D1_problem.ExecuteFinalize()
D2_problem.ExecuteFinalize()

print("COUPLED PROBLEM SOLVED.")
```

A.2 Coupling strategies scripts

A.2.1 relaxation_strategy.py

```python
from __future__ import print_function, absolute_import, division

import numpy
import scipy

class RelaxationStrategy:

    def __init__(self,interface_problem_size,domain_size,problem1,problem2,interface_communicator,
     residual):

        # Common initialization for all strategies
        self.interface_problem_size = interface_problem_size
        self.domain_size = domain_size
        self.problem1 = problem1
        self.problem2 = problem2
        self.interface_communicator = interface_communicator
        self.residual_calculator = residual

        # Relaxation method initialization
        self.acceleration_type = "Aitken"   # Acceleration types available: "Aitken", "Fixed", "Full"
        self.w_0 = 0.25                     # Relaxation parameter initialization


    def ExecuteInitializeSolutionStep(self):
        pass


    def InterfaceSolutionUpdate(self,step,nl_it,iteration_guess_value,interface_residual):

        if self.acceleration_type == "Aitken":
            if nl_it==1:
                self.res_1 = numpy.copy(interface_residual)
                val_correct = iteration_guess_value + self.w_0*interface_residual

            elif nl_it>1:
                self.res_2 = numpy.array(interface_residual)

                aux1 = numpy.dot(self.res_1,self.res_2-self.res_1)
                aux2 = numpy.dot(self.res_2-self.res_1,self.res_2-self.res_1)
                w_1 = -self.w_0*(aux1/aux2)

                val_correct = iteration_guess_value + w_1*self.res_2

                # Update values
                self.res_1 = numpy.copy(self.res_2)
                self.w_0 = w_1

        elif self.acceleration_type == "Fixed":
            val_correct = iteration_guess_value + self.w_0*interface_residual

        elif self.acceleration_type == "Full":
            val_correct = iteration_guess_value + interface_residual

        return val_correct


    def ExecuteFinalizeSolutionStep(self):
        pass
```

A.2.2 jfnk_strategy.py

```python
from __future__ import print_function, absolute_import, division

import numpy
import scipy
import scipy.sparse.linalg

class gmres_counter(object): # Auxiliary class to be called within the GMRES solver
    def __init__(self, disp=True):
        self._disp = disp
        self.niter = 0
    def __call__(self, rk=None):
        self.niter += 1
        if self._disp:
            print("        GMRES iteration: ",self.niter,"rk = ",str(rk))


class JacobianEmulation:
    def __init__(self,iteration_guess,epsilon,problem1,problem2,
                 interface_communicator,residual,interface_residual):

        self.iteration_guess = iteration_guess.copy()
        self.epsilon = epsilon
        self.problem1 = problem1
        self.problem2 = problem2
        self.interface_communicator = interface_communicator
        self.base_residual = interface_residual
        self.residual_calculator = residual

    def ComputeDerivative(self, du):
        # This function approximates the Jacobian projection onto a vector using finite differences.
        # Jv = R(guess+epsilon*du)-R(guess)/epsilon
        du = du.reshape((du.shape[0],))

        v = self.iteration_guess + self.epsilon * du
        rv = self.residual_calculator.ComputeResidual(v)

        Jv = rv-self.base_residual
        Jv /= -(self.epsilon ) # Recall the minus sign from the Newton's method correction definition

        return Jv


class JacobianFreeNewtonKrylovStrategy:

    def __init__(self,interface_problem_size,domain_size,
                 problem1,problem2,interface_communicator,residual):

        # Common initialization for all strategies
        self.interface_problem_size = interface_problem_size
        self.domain_size = domain_size
        self.problem1 = problem1
        self.problem2 = problem2
        self.interface_communicator = interface_communicator
        self.residual = residual

        # GMRES parameters
        self.restart = 300
        self.maxiter = 300
        self.tolerance = 1e-6
        self.epsilon = 1e-6      #Step for the A derivate approximation


    def ExecuteInitializeSolutionStep(self):
        pass


    def InterfaceSolutionUpdate(self,step,nl_it,iteration_guess_value,interface_residual):

        emulator = JacobianEmulation(iteration_guess_value,
                                     self.epsilon,
                                     self.problem1,
                                     self.problem2,
                                     self.interface_communicator,
                                     self.residual,
                                     interface_residual)
        rhs = interface_residual

        Jemulator = scipy.sparse.linalg.LinearOperator((self.interface_problem_size*self.domain_size,
                                                       self.interface_problem_size*self.domain_size),
                                                       matvec = emulator.ComputeDerivative,
                                                       dtype=float)

        gmres_callback_counter = gmres_counter()
        print("    GMRES residual minimization starts ...")
        correction,info = scipy.sparse.linalg.gmres(Jemulator,
                                                    rhs,
                                                    restart=self.restart,
                                                    callback=gmres_callback_counter,
                                                    maxiter=self.maxiter,
                                                    tol=self.tolerance)
        print("    GMRES residual minimization finished.")

        val_correct = iteration_guess_value + correction

        return val_correct


    def ExecuteFinalizeSolutionStep(self):
        pass
```

A.2.3 mvqn_strategy.py

```python
from __future__ import print_function, absolute_import, division

import numpy
import scipy

class MultiVectorQuasiNewtonStrategy:

    def __init__(self,interface_problem_size,domain_size,
                 problem1,problem2,interface_communicator,residual):

        # Common initialization for all strategies
        self.interface_problem_size = interface_problem_size
        self.domain_size = domain_size
        self.problem1 = problem1
        self.problem2 = problem2
        self.interface_communicator = interface_communicator
        self.residual_calculator = residual

        # MVQN method initialization
        self.Jac_n = -(numpy.identity(self.interface_problem_size*self.domain_size))
        self. w_0 = 0.825


    def ExecuteInitializeSolutionStep(self):

        # Observation matrices initialization
        self.obs_matrix_v = numpy.zeros((self.interface_problem_size*self.domain_size,1),dtype="float
    ")
        self.obs_matrix_w = numpy.zeros((self.interface_problem_size*self.domain_size,1),dtype="float
    ")


    def InterfaceSolutionUpdate(self,step,nl_it,iteration_guess_value,interface_residual):

        val_k1 = numpy.array(iteration_guess_value)
        res_k1 = numpy.array(interface_residual)

        if nl_it == 1:
            if step == 1:
                # The very first correction of the problem is done with fixed relaxation
                print("First fixed point iteration")
                val_correct = val_k1 + self.w_0*res_k1

            elif step > 1:
                # First step correction is done with previous step Jacobian
                val_correct = iteration_guess_value - numpy.dot(self.Jac_n,res_k1)
                print("Iteration interface flux updated with previous Jacobian.")


        elif nl_it > 1:

            if nl_it == 2:
                # First observation matrices fill
                self.obs_matrix_v[:,0] = res_k1-self.res_k
                self.obs_matrix_w[:,0] = val_k1-self.val_k
                print("Observation matrices first fill.")


            elif nl_it > 2:
                if self.obs_matrix_v.shape[1]<self.interface_problem_size*self.domain_size:
                    new_obs_matrix_v = numpy.zeros((self.obs_matrix_v.shape[0],self.obs_matrix_v.
    shape[1]+1),dtype="float")
                    new_obs_matrix_w = numpy.zeros((self.obs_matrix_w.shape[0],self.obs_matrix_w.
    shape[1]+1),dtype="float")

                    new_obs_matrix_v[:,0] = res_k1-self.res_k
                    new_obs_matrix_v[:,1:] = self.obs_matrix_v

                    new_obs_matrix_w[:,0] = val_k1-self.val_k
                    new_obs_matrix_w[:,1:] = self.obs_matrix_w

                    self.obs_matrix_v = numpy.copy(new_obs_matrix_v)
                    self.obs_matrix_w = numpy.copy(new_obs_matrix_w)
                    print("Observation matrices updated.")

                else:
                    print("ALERT: Old columns are to be dropped.")
                    new_obs_matrix_v = numpy.zeros((self.obs_matrix_v.shape[0],self.obs_matrix_v.
    shape[1]),dtype="float")
                    new_obs_matrix_w = numpy.zeros((self.obs_matrix_w.shape[0],self.obs_matrix_w.
    shape[1]),dtype="float")

                    new_obs_matrix_v[:,0] = res_k1-self.res_k
                    new_obs_matrix_v[:,1:] = self.obs_matrix_v[:,:-2]

                    new_obs_matrix_w[:,0] = val_k1-self.val_k
                    new_obs_matrix_w[:,1:] = self.obs_matrix_w[:,:-2]

                    self.obs_matrix_v = numpy.copy(new_obs_matrix_v)
                    self.obs_matrix_w = numpy.copy(new_obs_matrix_w)
                    print("Observation matrices updated.")


            # Jacobian approximation
            aux_1 = numpy.array(self.obs_matrix_w - numpy.dot(self.Jac_n,self.obs_matrix_v))
            aux_2 = scipy.linalg.pinv2(numpy.dot(self.obs_matrix_v.T,self.obs_matrix_v))

            self.Jac_k1 = self.Jac_n + numpy.dot(numpy.dot(aux_1,aux_2),self.obs_matrix_v.T)
            print("Jacobian updated.")

            # Solution update
            val_correct = iteration_guess_value - numpy.dot(self.Jac_k1,res_k1)
            print("Iteration interface solution updated.")

        # Variables update
        self.val_k = numpy.copy(val_k1)
        self.res_k = numpy.copy(res_k1)

        return val_correct


    def ExecuteFinalizeSolutionStep(self):

            self.Jac_n = numpy.copy(self.Jac_k1)
```

A.3 residual_definitions.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import numpy

class NeumannNeumannResidual:
    def __init__(self,problem_D1,problem_D2,interface_communicator):
        self.problem_D1 = problem_D1
        self.problem_D2 = problem_D2
        self.interface_communicator = interface_communicator

    def ComputeResidual(self,interface_flux_D1):

        self.problem_D1.SolveNeumann(interface_flux_D1)
        result_int_D1 = self.problem_D1.GetInterfaceVelocity()

        interface_flux_D2=-(self.interface_communicator.FluidToStructure_VectorMap(interface_flux_D1)
    )

        self.problem_D2.SolveNeumann(interface_flux_D2)
        result_int_D2 = self.problem_D2.GetInterfaceVelocity()

        result_int_D2_comm = self.interface_communicator.StructureToFluid_VectorMap(result_int_D2)

        residual = result_int_D1 - result_int_D2_comm

        return numpy.array(residual)


class DirichletNeumannResidual:
    def __init__(self,problem_D1,problem_D2,interface_communicator):
        self.problem_D1 = problem_D1
        self.problem_D2 = problem_D2
        self.interface_communicator = interface_communicator

    def ComputeResidual(self,interface_velocity_D1):

        self.problem_D1.SolveDirichlet(interface_velocity_D1)
        result_int_D1 = self.problem_D1.GetInterfaceReaction()

        interface_flux_D2=-(self.interface_communicator.FluidToStructure_VectorMap(result_int_D1))

        self.problem_D2.SolveNeumann(interface_flux_D2)
        result_int_D2 = self.problem_D2.GetInterfaceVelocity()

        result_int_D2_comm = self.interface_communicator.StructureToFluid_VectorMap(result_int_D2)

        residual = result_int_D2_comm - interface_velocity_D1

        return numpy.array(residual)
```

A.4 Punctual fluid load

A.4.1 pointforce2Dfluid.h

```
#if !defined(KRATOS_PointForce2Dfluid_CONDITION_H_INCLUDED )
#define  KRATOS_PointForce2Dfluid_CONDITION_H_INCLUDED


// System includes


// External includes
#include "boost/smart_ptr.hpp"


// Project includes
#include "includes/define.h"
#include "includes/serializer.h"
#include "includes/condition.h"
#include "includes/ublas_interface.h"
#include "includes/variables.h"


namespace Kratos
{

///@name Kratos Globals
///@{

///@}
///@name Type Definitions
///@{

///@}
///@name  Enum's
///@{

///@}
///@name  Functions
///@{

///@}
///@name Kratos Classes
///@{

/// Short class definition.
/** Detail class definition.
*/
class PointForce2Dfluid
    : public Condition
{
public:
    ///@name Type Definitions
    ///@{

    /// Counted pointer of PointForce2Dfluid
    KRATOS_CLASS_POINTER_DEFINITION(PointForce2Dfluid);

    ///@}
    ///@name Life Cycle
    ///@{

    /// Default constructor.
    PointForce2Dfluid(IndexType NewId, GeometryType::Pointer pGeometry);
    PointForce2Dfluid(IndexType NewId, GeometryType::Pointer pGeometry,
                      PropertiesType::Pointer pProperties);

    /// Destructor.
    virtual ~PointForce2Dfluid();
```

```cpp


    ///@}
    ///@name Operators
    ///@{


    ///@}
    ///@name Operations
    ///@{

    Condition::Pointer Create(IndexType NewId, NodesArrayType const&
                              ThisNodes,  PropertiesType::Pointer pProperties) const;

    void CalculateLocalSystem(MatrixType& rLeftHandSideMatrix,
                              VectorType& rRightHandSideVector,
                              ProcessInfo& rCurrentProcessInfo);

    void CalculateRightHandSide(VectorType& rRightHandSideVector,
                                ProcessInfo& rCurrentProcessInfo);

    //virtual void CalculateLeftHandSide(MatrixType& rLeftHandSideMatrix,
    //                                   ProcessInfo& rCurrentProcessInfo);

    void EquationIdVector(EquationIdVectorType& rResult,
                          ProcessInfo& rCurrentProcessInfo);

    void GetDofList(DofsVectorType& ConditionalDofList,
                    ProcessInfo& CurrentProcessInfo);

    ///@}
    ///@name Access
    ///@{


    ///@}
    ///@name Inquiry
    ///@{


    ///@}
    ///@name Input and output
    ///@{

    /// Turn back information as a string.
//      virtual String Info() const;

    /// Print information about this object.
//      virtual void PrintInfo(std::ostream& rOStream) const;

    /// Print object's data.
//      virtual void PrintData(std::ostream& rOStream) const;


    ///@}
    ///@name Friends
    ///@{


    ///@}

protected:
    ///@name Protected static Member Variables
    ///@{


    ///@}
    ///@name Protected member Variables
    ///@{


```

```
    ///@}
    ///@name Protected Operators
    ///@{


    ///@}
    ///@name Protected Operations
    ///@{

    friend class Serializer;

    // A private default constructor necessary for serialization
    PointForce2Dfluid() {};

    virtual void save(Serializer& rSerializer) const
    {
        KRATOS_SERIALIZE_SAVE_BASE_CLASS(rSerializer, Condition );
    }

    virtual void load(Serializer& rSerializer)
    {
        KRATOS_SERIALIZE_LOAD_BASE_CLASS(rSerializer, Condition );
    }


    ///@}
    ///@name Protected  Access
    ///@{


    ///@}
    ///@name Protected Inquiry
    ///@{


    ///@}
    ///@name Protected LifeCycle
    ///@{


    ///@}

private:
    ///@name Static Member Variables
    ///@{


    ///@}
    ///@name Member Variables
    ///@{


    ///@}
    ///@name Private Operators
    ///@{

    ///@}
    ///@name Private Operations
    ///@{


    ///@}
    ///@name Private  Access
    ///@{


    ///@}
    ///@name Private Inquiry
    ///@{


```

```
    ///@}
    ///@name Un accessible methods
    ///@{

    /// Assignment operator.
    //PointForce2Dfluid& operator=(const PointForce2Dfluid& rOther);

    /// Copy constructor.
    //PointForce2Dfluid(const PointForce2Dfluid& rOther);


    ///@}

}; // Class PointForce2Dfluid

///@}

///@name Type Definitions
///@{


///@}
///@name Input and output
///@{


/// input stream function
/*  inline std::istream& operator >> (std::istream& rIStream,
            PointForce2Dfluid& rThis);
*/
/// output stream function
/*  inline std::ostream& operator << (std::ostream& rOStream,
            const PointForce2Dfluid& rThis)
    {
      rThis.PrintInfo(rOStream);
      rOStream << std::endl;
      rThis.PrintData(rOStream);

      return rOStream;
    }*/
///@}

}  // namespace Kratos.

#endif // KRATOS_PointForce2Dfluid_CONDITION_H_INCLUDED  defined
```

A.4.2 pointforce2Dfluid.cpp

```
// System includes


// External includes


// Project includes
#include "includes/define.h"
#include "custom_conditions/pointforce2Dfluid.h"
#include "incompressible_fluid_application.h"
#include "utilities/math_utils.h"

namespace Kratos
{
//************************************************************************************
//************************************************************************************
PointForce2Dfluid::PointForce2Dfluid(IndexType NewId,
                                    GeometryType::Pointer pGeometry)
                                    : Condition(NewId, pGeometry)
{
    //DO NOT ADD DOFS HERE!!!
}

//************************************************************************************
//************************************************************************************
PointForce2Dfluid::PointForce2Dfluid(IndexType NewId,
                                    GeometryType::Pointer pGeometry,
                                    PropertiesType::Pointer pProperties)
                                    : Condition(NewId, pGeometry, pProperties)
{
}

Condition::Pointer PointForce2Dfluid::Create(IndexType NewId,
                                            NodesArrayType const& ThisNodes,
                                            PropertiesType::Pointer pProperties) const
{
    return Condition::Pointer(new PointForce2Dfluid(NewId,
                              GetGeometry().Create(ThisNodes), pProperties));
}

PointForce2Dfluid::~PointForce2Dfluid()
{
}


//************************************************************************************
//************************************************************************************
void PointForce2Dfluid::CalculateRightHandSide(VectorType& rRightHandSideVector,
                                              ProcessInfo& rCurrentProcessInfo)
{
    KRATOS_TRY
    if(rRightHandSideVector.size() != 3)
        rRightHandSideVector.resize(3,false);

    array_1d<double,3>& force = GetGeometry()[0].FastGetSolutionStepValue(FORCE);
    rRightHandSideVector[0] = force[0];
    rRightHandSideVector[1] = force[1];
    rRightHandSideVector[2] = 0.0;

    KRATOS_CATCH("")
}

//************************************************************************************
//************************************************************************************
void PointForce2Dfluid::CalculateLocalSystem(MatrixType& rLeftHandSideMatrix,
                                            VectorType& rRightHandSideVector,
                                            ProcessInfo& rCurrentProcessInfo)
{
    KRATOS_TRY

    if(rLeftHandSideMatrix.size1() != 3)
        rLeftHandSideMatrix.resize(3,3,false);
    noalias(rLeftHandSideMatrix) = ZeroMatrix(3,3);

    if(rRightHandSideVector.size() != 3)
        rRightHandSideVector.resize(3,false);

    array_1d<double,3>& force = GetGeometry()[0].FastGetSolutionStepValue(FORCE);
    rRightHandSideVector[0] = force[0];
    rRightHandSideVector[1] = force[1];
    rRightHandSideVector[2] = 0.0;

    KRATOS_CATCH("")
}


//************************************************************************************
//************************************************************************************
void PointForce2Dfluid::EquationIdVector(EquationIdVectorType& rResult,
                                          ProcessInfo& CurrentProcessInfo)
{

    rResult.resize(3);
    rResult[0] = (GetGeometry()[0].GetDof(VELOCITY_X).EquationId());
    rResult[1] = (GetGeometry()[0].GetDof(VELOCITY_Y).EquationId());
    rResult[2] = (GetGeometry()[0].GetDof(PRESSURE).EquationId());

}

//************************************************************************************
//************************************************************************************
void PointForce2Dfluid::GetDofList(DofsVectorType& ConditionalDofList,
                                    ProcessInfo& CurrentProcessInfo)
{

    ConditionalDofList.resize(3);
    ConditionalDofList[0] = (GetGeometry()[0].pGetDof(VELOCITY_X));
    ConditionalDofList[1] = (GetGeometry()[0].pGetDof(VELOCITY_Y));
    ConditionalDofList[2] = (GetGeometry()[0].pGetDof(PRESSURE));

}
} // Namespace Kratos
```

Bibliography

[1] D. Baumgärtner, J. Wolf, R. Rossi, R. Wüchner, and P. Dadvand. *Contribution to the Fluid-Structure Interaction Analysis of Ultra-lightweight Structures using an Embedded Approach.* CIMNE, 2015.

[2] Y. Bazilevs, K. Takizawa, and T.E. Tezduyar. *Computational Fluid-Structure Interaction: Methods and Applications.* Wiley Series in Computational Mechanics. Wiley-Blackwell, 2013.

[3] T. Belytschko, W.K. Liu, and B. Moran. *Nonlinear Finite Elements for Continua and Structures.* John Wiley and Sons, 2000.

[4] A.E.J. Bogaers, S. Kok, B.D. Reddy, and T. Franz. Quasi-newton methods for implicit black-box fsi coupling. *Computer Methods in Applied Mechanics and Engineering*, 279:113–132, 2014.

[5] A.N. Brooks and T.J.R. Huges. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32:199–259, 1982.

[6] F.J. Calvo. *Simulación del flujo sanguíneo y su interacción con la pared arterial mediante modelos de elementos finitos.* PhD thesis: Universidad Politécnica de Madrid, 2006.

[7] J. Chung and G.M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized$-\alpha$ method. *Journal of Applied Mechanics*, 60:371–375, 1993.

[8] R. Codina. Pressure stability in fractional step finite element methods for incompressible flows. *Journal of Computational Physics*, 170:112–140, 2001.

[9] R. Codina. A stabilized finite element method for generalized stationary incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 190:2681–2706, 2001.

[10] R. Codina. Stabilized finite element approximation of transient incompressible flows using orthogonal subscales. *Computer Methods in Applied Mechanics and Engineering*, 191:4295–4321, 2002.

[11] R. Codina, J. Principe, and S. Badia. Dissipative structure and long term behavior of a finite element approximation of incompressible flows with numerical subgrid scale modeling. *Lecture Notes on Applied and Computational Mechanics*, 55:75–93, 2011.

[12] R. Codina and O. Soto. Approximation of the incompressible navier-stokes equations using orthogonal subscale stabilization and pressure segregation on anisotropic finite element meshes. *Computer Methods in Applied Mechanics and Engineering*, 193:1403–1419, 2004.

[13] M.A. Crisfield. *Non-Linear Finite Element Analysis of Solids and Structures*. Wiley, 1996.

[14] P. Dadvand. *A framework for developing finite element codes for multi-disciplinary applications*. PhD thesis: Universidad Politécnica de Cataluña, 2007.

[15] P. Dadvand, R. Rossi, A. Larese, M.A. Celigueta, and J.M. Carbonell. www.cimne.com/kratos/default.asp.com, 2012.

[16] P. Dadvand, R. Rossi, and E. Oñate. An object-oriented environment for developing finite element codes for multi-disciplinary applications. *Archives of Computational Methods in Engineering*, 17:253–297, 2010.

[17] J. Donea and A. Huerta. *Finite Elements Methods for Flow Problems*. Wiley, 2003.

[18] G.A. Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. Wiley, 2000.

[19] T.J.R. Hughes. Multiscale phenomena: Green's function, the dirichlet to neumann formulation, subgrid scale models, bubbles and the origins of stabilized formulations. *Computer Methods in Applied Mechanics and Engineering*, 127:387–401, 1995.

[20] T.J.R. Hughes, G.R. Feijóo, L. Mazzei, and J.B. Quincy. The variational multiscale method−a paradigm for computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 166:3–24, 1998.

[21] T.J.R. Hughes, L.P. Franca, and G.M. Hulbert. A new finite element formulation for computational fluid dynamics: VIII. the galerkin/least-squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73:173–189, 1989.

[22] T.J.R. Hughes, L. Mazzei, and K.E. Jansen. Large eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, 3:47–59, 2000.

[23] S.R. Idelsohn, F. del Pin, and R. Rossi. *Arbitrary Lagrangian-Eulerian and Fluid-Structure Interaction*, chapter 4. Wiley-ISTE, 2010.

[24] R.K. Jaiman, X. Jiao, P.H. Geubelle, and E. Loth. Assessment of conservative load transfer for fluid−solid interface with non-matching meshes. *Numerical Methods in Engineering*, 64:2014–2038, 2005.

[25] K.E. Jansen, C.H. Withing, and G.M. Hulbert. A generalized$-\alpha$ method for integrating the filtered navier-stokes equations with a stabilized finite element method. *Computer Methods in Applied Mechanics and Engineering*, 190:305–319, 2000.

[26] U. Küttler and W.A. Wall. Fixed−point fluid−structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43:61–72, 2008.

[27] V. Mataix. A contribution towards the development of a virtual wind tunnel (VWT). Master's thesis, École Politechnique Université Paris−Saclay, 2015.

[28] H. Matthies, R. Niekamp, and J. Steindorf. Algorithms for strong coupling procedures. *Computer Methods in Applied Mechanics and Engineering*, 195:2028–2049, 2004.

[29] A. Melendo, A. Coll, M. Pasenau, E. Escolano, and A. Monros. www.gidhome.com, 2015.

[30] S. Minami and S. Yoshimura. Performance evaluation of nonlinear algorithms with line-search for partitioned coupling techniques for fluid-structure interaction. *International Journal for Numerical Methods in Fluids*, 64:1129–1147, 2010.

[31] D.P. Mok. *Partitionierte Lösungsansätze in der Strukturdynamik und der Fluid−Struktur−Interaktion.* PhD thesis: Institut für Baustatik, Universität Stuttgart, 2001.

[32] F. Nobile. *Numerical Approximation of Fluid−Structure Interaction Problems with Application to Haemo−dynamics.* PhD thesis: Départment de Mathématiques, École Polytechnique Fédérale de Laussane, 2001.

[33] X. Oliver and C. Agelet de Saracibar. *Mecánica de Medios Contínuos para Ingenieros.* Edicions UPC, 2002.

[34] S. Oller. *Dinámica No-Lineal.* CIMNE, 2002.

[35] E. Oñate. A stabilized finite element method for incompressible viscous flows using a finite increment calculus formulation. *Computer Methods in Applied Mechanics and Engineering*, 182:355–370, 2000.

[36] E. Oñate. *Structural Analysis with the Finite Element Method. Linear Statics: Volume 1: Basis and Solids.* Springer, 2009.

[37] E. Oñate. *Structural Analysis with the Finite Element Method. Linear Statics: Volume 2: Beams, Plates and Shells.* Springer, 2014.

[38] A. Quarteroni. *Numerical Models for Differential Problems.* Springer, 2012.

[39] R. Rossi. *Light Weight Structures: Structural Analysis and Coupling Issues.* PhD thesis: Università degli Studi di Bologna, 2005.

[40] Y. Saad and M.H. Schultz. Algorithms for strong coupling procedures. *Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[41] M. Schäfer. *Arbitrary Lagrangian-Eulerian and Fluid-Structure Interaction*, chapter 3. Wiley-ISTE, 2010.

[42] T.E. Tezduyar. *Encyclopedia of Computational Mechanics, Volume 3: Fluids*, chapter 17. John Wiley and Sons, 2004.

[43] S. Turek and J. Hron. *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, chapter Proposal for Numerical Benchmarking of Fluid-Structure Interaction between an Elastic Object and Laminar Incompressible Flow, pages 371–385. Springer Berlin Heidelberg, 2006.

[44] G. Valdés. *Nonlinear Analysis of Orthotropic Membrane and Shell Structures Including Fluid-Structure Interaction.* PhD thesis: Universidad Politécnica de Cataluña, 2007.

[45] T. Wang, R. Wüchner, S. Sicklinger, and K.U. Bletzinger. Assessment and improvement of mapping algorithms for non-matching meshes and geometries in computational fsi. *Computational Mechanics*, 57:793–816, 2016.

[46] O.C. Zienkiewicz, R.L. Taylor, and D.D. Fox. *The Finite Element Method for Solid and Structural Mechanics (edition 7).* Butterworth-Heinemann, 2013.