Spring 1-1-2011

Modeling Fluid-Rigid Body Interaction Using the Arbitrary Lagrangian Eulerian Method

Sophie Capdevielle
University of Colorado at Boulder, sophie.capdevielle@colorado.edu

Modeling fluid-rigid body interaction using the Arbitrary Lagrangian Eulerian Method

by

Sophie Capdevielle

B.A., ENS Cachan and Universite Pierre et Marie Curie

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Civil, Environmental, and Architectural Engineering

2012

This thesis entitled:
Modeling fluid-rigid body interaction using the
Arbitrary Lagrangian Eulerian Method
written by Sophie Capdevielle
has been approved for the Department of Civil, Environmental, and Architectural Engineering

Mettupalayam V. Sivaselvan

Prof. Harihar Rajaram

Prof. Richard A. Regueiro

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Abstract

Capdevielle, Sophie (M.S., Civil Engineering)

Modeling fluid-rigid body interaction using the

Arbitrary Lagrangian Eulerian Method

Thesis directed by Prof. Mettupalayam V. Sivaselvan

The goal of this thesis is to build a clear understanding of the Arbitrary Lagrangian Eulerian (ALE) method and to develop a useful simple implementation. A review of the ALE method is presented with precise notations and detailed explanation of the combined kinematics. As an application, a complete model of fluid-rigid body interaction is developed. Starting from the Navier-Stokes equations, ALE governing equations for the fluid are derived. They are discretized in space using the Finite Element method. Coupled with the rigid body equation of motion via compatibility and equilibrium interface conditions, they lead to a nonlinear system of equations. The latter is solved using an approximated Newton's method. Details on the implementation are given to illustrate the numerical solution procedure. In particular, the prescription of the mesh motion, specific to the ALE method, is developed. The simple pure fluid problem of the Couette flow is used to test the implementation. The formulation is then used to simulate the free oscillations of a rigid circular cylinder embedded in a viscous fluid.

Acknowledgements

I would like to express my gratitude to Professor Siva, for enabling me to come to the University of Colorado at Boulder by funding my studies, and for letting me working with him. Although sometimes chaotic, research work with professor Siva has always been really interesting. I would like to thank Professors Rajaram and Regueiro who accepted to be my committee members.

I would like to thank the administrative team of the Civil Engineering department, in particular Pamela Williamson for her help.

I would like to thank my working mates Arnaud, Valentin, Scott, Umut, Reza, Louis, Xavi and Eduard for their serious and fun presence in the office. I would like to acknowledge Naree's writing advice. Thanks to Federico for his support while I have to work hard. Thank you to my family members for their support from far away. Finally, I would like to thank my friends, from France and from Boulder, for their support.

Contents

Tables

Figures

Chapter 1

Introduction

1.1 Need for an alternate description of motion

1.1.1 Practical problems of interest

Several interesting problems are encountered by engineers in their search for optimizing our resources and safety. Some examples are presented below, which all lead to continuum models with very large deformation and a changing boundary position. These combined two features make the classical modeling approaches hard to use, if not impossible. Thus, researchers resort to the Arbitrary Lagrangian Eulerian (ALE) method to address these problems.

Fluid-structure interaction

Behind the general name fluid-structure interaction, various practical applications are hidden. Civil engineers need to know how a bridge or a skyscraper behaves in interaction with the wind, to ensure structural safety. To help progress in medicine, biomechanicians study the dynamics of heart valves and blood flow in our cardiovascular system. In an attempt to find new durable sources of energy, recent research is developing small energy-harvesting devices based on flow-induced instabilities ([8]). For fluid-structure interaction applications of ALE, the reader is invited to refer to [21], [22], [17],[7] or [15].

Industrial problems: rolling contact and forming process

An apparently completely unrelated problem is the study of rolling contact. This is useful to make concerned components last longer, such as car tires or rollers in rotating machines. The goal is also to improve the performance of the products, such as printing machines ([13], [2]). Studying forming processes has also as a goal to improve the quality of the final product, as well as the manufacturing process, by predicting the load on the tools (see [18] and [12]).

Penetration mechanics

In the process of constructing a building, the foundation part is often the most uncertain, due to the difficulty in modeling the soil accurately. Penetration mechanics is useful to model the driving of a pile into the ground, serving as a structural support. It also models the intrusion of measurement tools in the soil ([14], [19], [20]).

1.1.2 Two classical approaches with limitations

To observe a continuum in motion, two perspectives are classically used. In a Lagrangian description, the motion is captured by observers moving with the material particles. This point of view enables precise tracking of moving boundaries. The Lagrangian approach is limited in its inability to deal with large deformations.[1] It is thus not suitable for studying fluid motion, but is usually used for the study of pure solid problems.

For a pure fluid problem, an Eulerian approach is generally used. The motion of the flow is then captured by observers fixed in space. This corresponds to the way most measurements are made in real flows. Large deformations of the material do not cause any numerical problem since the numerical discretization does not follow the material motion. Mobile boundaries can not be followed accurately unless a very fine mesh is used. This usually does not matter for fluid only problems, because the region of interest is usually fixed and fluid flows through. Difficulties arise when there is an interface to solve for, such as in a fluid-structure interaction problem.

[1] When the material encounters too large deformations, the deformation gradient becomes close to singular. The computation of its inverse, if not impossible, leads to large numerical errors.

1.1.3 Introduction of an alternate representation

As briefly stated above, the Lagrangian and Eulerian representations are not suitable for a problem with large deformations where the position of a mobile boundary needs to be known with accuracy. This arises in many applications, as shown in paragraph 1.1.1. To solve this issue, a combined description of motion is needed. Once such framework is the Arbitrary Lagrangian Eulerian (ALE) method, designed in the 1970's to address the fluid-structure interaction issue ([9], [21], [22], [17],[7], and [15]), and quickly extended to solid mechanics. The ALE method keeps the advantages of both Lagrangian and Eulerian descriptions, without suffering from their drawbacks. The ALE method is now used by commercial codes in solving multiphysics problems, such as COMSOL for example ([1]).

In solid mechanics, the ALE method is applied in the simulation of forming processes (see [18] and [12]), because the changing shape of the material needs to be precisely known, while really large deformations are encountered. For the same reasons, the ALE method is applied in modeling rolling contact (see [13]) or penetration mechanics (see [20]). Other applications can be found in nonlinear elastoplasticity ([16], [10]). The ALE approach is used to avoid excessive mesh distorsion, but leads to a new difficulty in the treatement of path-dependant constitutive relationships.

This study focuses on the ALE method because of its intriguing combined kinematic description and its wide range of applications.

1.2 Motivation and organization of this ALE study

The main purpose of this thesis is to build a deep understanding of the ALE formulation and express it in the clearest possible way. Literature is quite rich on the subject. But to lighten the notations and stay concise, most authors do not develop the ALE governing equations in a complete and indubitable way. Thus the first purpose of the work is to rigorously derive the ALE equations by untangling the notations. [9] and [5] are helpful references to understand the concept of the

ALE representation.

To support the study, the second aim of this work is to develop a simple implementation. As stated in paragraph 1.1.3, the oldest applications are found in fluid-structure interaction. [22] presents the numerical example of a rigid cylinder freely oscillating in a fluid. The problem is modeled in this thesis with the goal of getting results close to [22]. This numerical application builds up the essential implementation ingredients of the ALE method. It should found a starting point to explore further ALE applications, such as those cited in paragraph 1.1.1 and 1.1.3.

The presentation of the work is organized as follows. First, we explain how the ALE method combines the Lagrangian and Eulerian descriptions. We then introduce the notations corresponding to this combination, building all the bases to develop the governing equations for fluid-rigid body interaction. They are derived in chapter 2, starting with the ALE formulation of the fluid equations of motion. Solid governing equations are then presented, followed by the interface coupling conditions. Chapter 3 discretizes the fluid ALE equations by the Finite Element Method, to get a system of coupled nonlinear equations governing the problem. The numerical process to solve this system is detailed next. Chapter 4 explains its practical implementation. Selected parts of the code are highlighted to ease the reader's understanding. Chapter 5 presents the numerical examples. Finally, results of the work and concluding remarks are summarized in chapter 6.

1.3 From Lagrangian and Eulerian to ALE description of motion

1.3.1 A moving frame of reference

To combine the Lagrangian and Eulerian approaches, a third set of observers is introduced, used as a reference for describing motion. These ALE observers move as prescribed by the user of the ALE method, independently from the material (hence the name "arbitrary"). The corresponding mesh's motion can be chosen to keep track of mobile boundaries, while avoiding excessive mesh distortion where the material encounters large deformations. One way to prescribe the mesh motion is to set

it as Lagrangian on the mobile boundaries and as Eulerian far from these boundaries. Motion of the nodes located in the transition zone is treated using an interpolation technique, so that the mesh displacement and velocity are as smooth as possible. For example, they can be computed solving a Laplace equation (see [9] for more details or other interpolation techniques). [22] linearly interpolates the mesh displacement and velocity functions of the distance from the interface.

When ALE observers follow the material motion, the ALE method reduces to the Lagrangian formulation. Conversely, prescribing a fixed mesh motion reduces the ALE method to an Eulerian formulation. Defining the mesh motion is the tool used to combine Lagrangian and Eulerian descriptions.

The freedom of arbitrarily defining the mesh motion has another application : mesh adaptation. This enables mesh refinement in specific zones, such as zones of steep gradients. Since the number of elements and their connectivity remain the same, this mesh adaptation is computationally cheap. The reader is referred to [9] for more details on this technique and to [3] and [4] for application to failure localization.

Section 1.3.2 translates these concepts into mathematical notations. These are necessary for the derivation of the ALE description of motion.

1.3.2 Definition of the notations

1.3.2.1 Domains and mapping functions

Labels for the three groups of observers are collected into three different sets (see figure 1.1). The material domain Ω_0 corresponds to the initial (undeformed) configuration and contains the labels X of the Lagrangian observers. The spatial domain Ω corresponds to the current configuration and contains the labels χ of the Eulerian observers. These are fixed and attached to the global coordinate system $\mathbf{x}$. Labels $\mathcal{X}$ of the ALE observers are contained by the ALE domain $\tilde{\Omega}$, which is used as a reference for motion description and spatial discretization.

The mapping function between the ALE domain and the spatial domain $\Phi : \mathcal{X} \mapsto \chi$ represents the mesh motion. The mesh motion is the motion of ALE observers labeled by $\mathcal{X}$, seen from the point of view of Eulerian observers. The material ("true") motion is given by the map $\varphi : X \mapsto \chi$. Motion of the material observed by the ALE frame is given by the map $\phi : X \mapsto \mathcal{X}$. These functions are related by $\varphi = \Phi \circ \phi$.

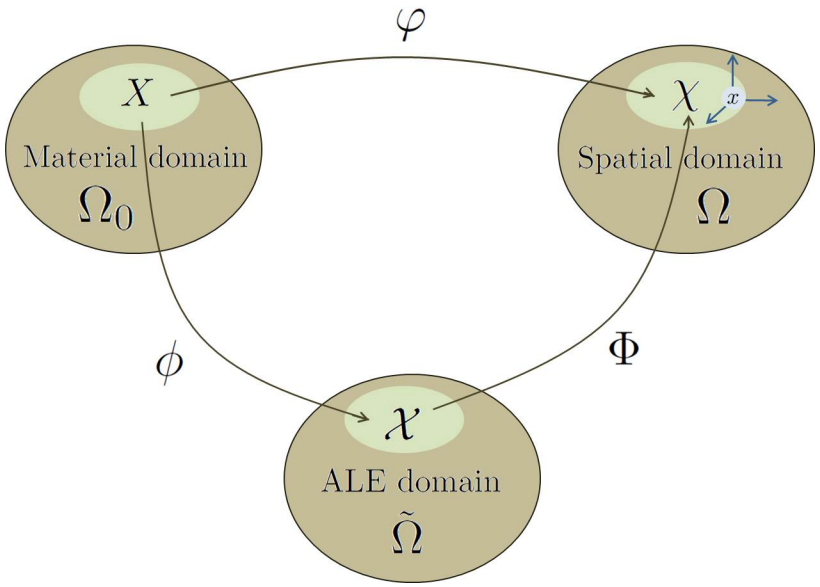

Figure 1.1: Notations for the ALE description of motion

1.3.2.2 Functions

A dependant variable f (such as velocity or pressure for example) can be function of any of the particle labels X, χ or $\mathcal{X}$. It keeps the same physical meaning but the function is different. Thus distinct notations are needed.

The dependant variables are denoted with a hat when they are function of the material labels X, with a tilde when they are functions of the ALE label $\mathcal{X}$ and without any additional symbol when they are function of the spatial labels χ (see figure 1.2).

Such a notational distinction is introduced in continuum mechanics textbooks (eg [5]), but quickly dropped. However the distinction is especially helpful in the ALE context, where there is a third set of observers.

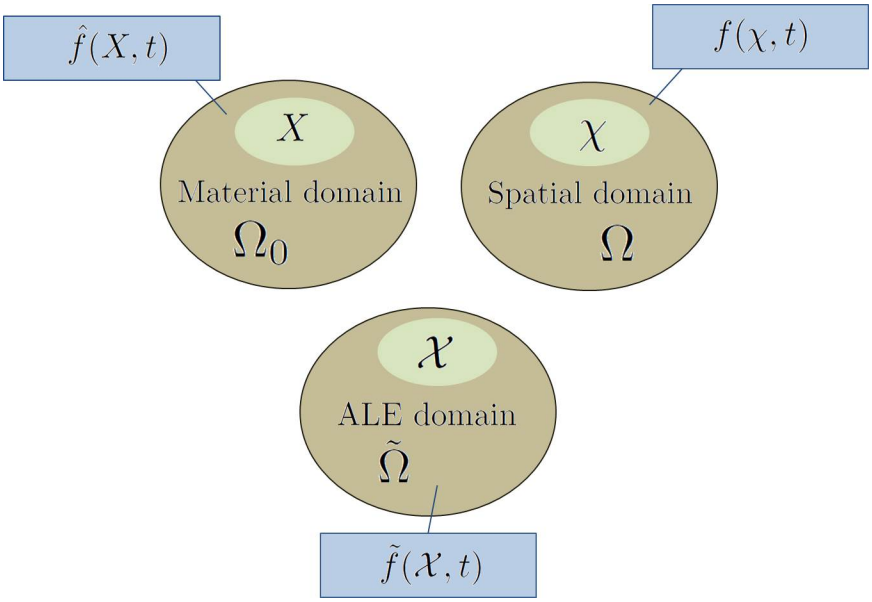

Figure 1.2: Notations for a dependant variable f, where $\tilde{f}(\mathcal{X},t) = \hat{f}(X,t) = f(\chi,t)$

1.3.2.3 Velocities

Because of the addition of the ALE domain and two corresponding mapping functions, different kinds of velocities are defined. Their definitions, notations and meanings require specific attention.

1.3.2.4 Material velocity

$$\hat{v}(X,t) = \frac{\partial \varphi}{\partial t}(X,t) \tag{1.1}$$

The material velocity $\hat{v}$ is the velocity of the material particles (labelled by X), observed by the spatial frame χ, and expressed in the global coordinate system $\mathbf{x}$.

1.3.2.5 Mesh velocity

$$\tilde{v}_{\text{mesh}}(\mathcal{X}, t) = \frac{\partial \Phi}{\partial t}(\mathcal{X}, t) \tag{1.2}$$

The mesh velocity corresponds to the motion of the ALE nodes (labeled by $\mathcal{X}$) as seen from the spatial perspective χ and expressed in the global coordinate system $\mathbf{x}$. This motion is prescribed by the user.

1.3.2.6 Convective velocity

Convective velocity is the difference between the material and mesh velocities. It represents the material velocity relative to the velocity of the ALE particles, as seen by the spatial observers χ. Its expression in terms of the ALE labels $\mathcal{X}$ is:

$$\tilde{c}\,(\mathcal{X}, t) = \hat{v}\left(\phi^{-1}(\mathcal{X}, t), t\right) - \tilde{v}_{\text{mesh}}(\mathcal{X}, t) \tag{1.3}$$

The convective velocity is used in the formulation of the conservation equations in the ALE framework, as detailed in chapter 2, section 2.1.

Chapter 2

ALE formulation of the governing equations for fluid-rigid body interaction

Introduction

The goal here is to set up the governing equations for a fluid-rigid body interaction problem. Fluid-rigid body interaction is studied as an idealization of fluid-structure interaction, when the structure deformations are small compared to its rigid-body motion. The problem is simpler, because the solid does not need a continuum description. Since a 2D problem is modeled here, the rigid body has only 3 degrees of freedom. But, because of the interface, an Arbitrary Lagrangian Eulerian (ALE) description of motion is needed for the fluid.

The equations for a 2D fluid-structure interaction problem are presented in [22]. They are derived here in more detail, with particular focus on obtaining the ALE form of the fluid equations. Section 2.1 explains the transformation of the Navier-Stokes equations from their Eulerian to their ALE expression. The solid equation of motion is presented in section 2.2. The fluid and solid problems interact through their common interface. Coupling conditions at this interface are detailed in section 2.3.

2.1 ALE expression of the fluid governing equations

Fluid governing equations come from the Navier-stokes equations, taking into account the assumptions listed in paragraph 2.1.1. They are expressed in Eulerian form in paragraph 2.1.2. These equations are then transformed to the ALE representation in paragraphs 2.1.3 and 2.1.4. In this

whole section, governing equations are written in index notation.

2.1.1 Assumptions

(1) 2D problem

(2) Homogeneous, Newtonian fluid

(3) Incompressible flow

(4) Effect of gravity neglected

2.1.2 Governing equations in the Eulerian representation

In Eulerian representation, the governing equations are completely evaluated at the Eulerian (spatial) observers locations χ.

Conservation of mass

Taking assumption 3 into account, the conservation of mass is written as :

$$\frac{\partial v_k}{\partial \chi_k} = 0 \tag{2.1}$$

Conservation of momentum

$$\rho \frac{dv_i}{dt} = \frac{\partial \sigma_{ij}}{\partial \chi_j} + \rho b_i \tag{2.2}$$

where $\frac{d}{dt}$ denotes the material derivative.

Constitutive relationship

For Newtonian fluids, the Cauchy's stress σ is related to the velocity gradient as follows:

$$\sigma_{ij} = -p\,\delta_{ij} + \mu \left(\frac{\partial v_i}{\partial \chi_j} + \frac{\partial v_j}{\partial \chi_i} \right) \tag{2.3}$$

where p is the pressure and μ the viscosity. δ_{ij} is the Kronecker delta.

2.1.3 Transformation to ALE

Spatial discretization leading to the finite element formulation of the conservation equations is based on the ALE frame of reference. Thus equations 2.1 and 2.2 need to be expressed in terms of the ALE labels $\mathcal{X}$.

But the most natural stress measure for fluid is the Cauchy's stress σ. Constitutive relationships are given in terms of σ for fluid. They are also often expressed in terms of σ in the solid case. Thus from a computational point of view, it is less expensive to express the weak form of conservation equations by integrating over the spatial domain Ω, according to [5]. Spatial derivatives are kept in terms of the spatial labels χ. The location of the nodes $\mathcal{X}$ will be taken into account by moving the mesh according to Φ.

To derive the ALE form of equations 2.1 and 2.2, the material time derivatives need to be expressed in the corresponding framework. Dependant variables are expressed in terms of $\mathcal{X}$ but spatial derivatives are kept in terms of χ. The convective term accounts for mesh motion.

2.1.3.1 Material derivative in the ALE description

With f being a dependent variable,

$$\begin{aligned}\frac{df}{dt}(\chi,t) &= \left.\frac{\partial \hat{f}}{\partial t}(X,t)\right|_{(\varphi^{-1}(\chi,t),t)} \\ &= \left[\frac{\partial \tilde{f}}{\partial t}(\mathcal{X},t) + \frac{\partial \tilde{f}}{\partial \mathcal{X}}(\mathcal{X},t)\,\frac{\partial \phi}{\partial t}\left(\phi^{-1}(\mathcal{X},t),t\right)\right]\Bigg|_{(\Phi^{-1}(\chi,t),t)}\end{aligned} \tag{2.4}$$

Since the spatial derivatives need to be expressed in terms of the spatial coordinates, equation 2.4 is further transformed to:

$$\frac{df}{dt}(\chi,t) = \left[\frac{\partial \tilde{f}}{\partial t}(\mathcal{X},t) + \frac{\partial f}{\partial \chi}(\Phi(\mathcal{X},t),t)\,\frac{\partial \Phi}{\partial \mathcal{X}}(\mathcal{X},t)\,\frac{\partial \phi}{\partial t}\left(\phi^{-1}(\mathcal{X},t),t\right)\right]\Bigg|_{(\Phi^{-1}(\chi,t),t)} \tag{2.5}$$

2.1.3.2 Expression of the convective velocity $\tilde{c}$ in terms of the mapping functions ϕ and Φ

Substituting the definitions of $\hat{v}$ (1.1) and $\tilde{v}_{\text{mesh}}$ (1.2) into (1.3) leads to:

$$\tilde{c}(\mathcal{X},t) = \frac{\partial \varphi}{\partial t}\left(\phi^{-1}(\mathcal{X},t),t\right) - \frac{\partial \Phi}{\partial t}(\mathcal{X},t) \tag{2.6}$$

Since $\varphi = \Phi \circ \phi$,

$$\tilde{c}(\mathcal{X},t) = \frac{\partial \Phi}{\partial \mathcal{X}}(\mathcal{X},t)\frac{\partial \phi}{\partial t}\left(\phi^{-1}(\mathcal{X},t)\right) + \frac{\partial \Phi}{\partial t}(\mathcal{X},t) - \frac{\partial \Phi}{\partial t}(\mathcal{X},t) \tag{2.7}$$

Thus

$$\tilde{c}(\mathcal{X},t) = \frac{\partial \Phi}{\partial \mathcal{X}}(\mathcal{X},t)\frac{\partial \phi}{\partial t}\left(\phi^{-1}(\mathcal{X},t),t\right) \tag{2.8}$$

2.1.3.3 Final expression of the material derivative

Substituting result 2.8 into 2.5 leads to:

$$\frac{df}{dt}(\chi,t) = \left[\frac{\partial \tilde{f}}{\partial t}(\mathcal{X},t) + \frac{\partial f}{\partial \chi}(\Phi(\mathcal{X},t),t)\,\tilde{c}(\mathcal{X},t)\right]\Bigg|_{(\Phi^{-1}(\chi,t),t)} \tag{2.9}$$

In index notations, equation 2.9 is expressed as:

$$\frac{df}{dt}(\chi,t) = \left[\frac{\partial \tilde{f}}{\partial t}(\mathcal{X},t) + \tilde{c}_k(\mathcal{X},t)\,\frac{\partial f}{\partial \chi_k}(\Phi(\mathcal{X},t),t)\right]\Bigg|_{(\Phi^{-1}(\chi,t),t)} \tag{2.10}$$

Equation 2.9 is referred to as the fundamental ALE equation by [9], because it serves as a basis to derive the ALE form of conservation equations.

2.1.4 ALE form of governing equations

Changing variable in the Eulerian form of the conservation equations 2.1 and 2.2, using the expression of the material derivative 2.9, leads to the conservation equation in the ALE description:

2.1.4.1 Conservation of mass

$$\frac{\partial v_k}{\partial \chi_k}\left(\Phi(\mathcal{X},t),t\right)=0 \tag{2.11}$$

2.1.4.2 Conservation of momentum

$$\rho\left[\frac{\partial \tilde{v}_i}{\partial t}\left(\mathcal{X},t\right)+\tilde{c}_k\left(\mathcal{X},t\right)\frac{\partial v_i}{\partial \chi_k}\left(\Phi(\mathcal{X},t),t\right)\right]=\frac{\partial \sigma_{ij}}{\partial \chi_j}\left(\Phi(\mathcal{X},t),t\right)+\rho\,\tilde{b}_i\left(\mathcal{X},t\right) \tag{2.12}$$

2.1.4.3 Constitutive relationship

$$\sigma_{ij}\left(\Phi(\mathcal{X},t),t\right)=-p\left(\Phi(\mathcal{X},t),t\right)\,\delta_{ij}+\mu\left(\frac{\partial v_i}{\partial \chi_j}\left(\Phi(\mathcal{X},t),t\right)+\frac{\partial v_j}{\partial \chi_i}\left(\Phi(\mathcal{X},t),t\right)\right) \tag{2.13}$$

2.2 Rigid body equation of motion

2.2.1 Assumptions

(1) 2D problem

(2) Motion of each degree of freedom uncoupled from the others

(3) Constant mass, damping and stiffness coefficients

2.2.2 Conservation of momentum of the rigid body

The rigid body motion is described by the motion of its center of mass G. Displacements δ_1 and δ_2 and rotation θ are collected in the vector δ.

$$\delta=\begin{bmatrix}\delta_1\\ \delta_2\\ \theta\end{bmatrix} \tag{2.14}$$

Solid velocities and accelerations are obtained by differentiation with respect to time:

$$\nu = \begin{bmatrix} \dot{\delta_1} \\ \dot{\delta_2} \\ \dot{\theta} \end{bmatrix} \tag{2.15}$$

$$\alpha = \begin{bmatrix} \ddot{\delta_1} \\ \ddot{\delta_2} \\ \ddot{\theta} \end{bmatrix} \tag{2.16}$$

The resulting external forces $F_{\text{ext}1}$ and $F_{\text{ext}2}$ and moment $\mathcal{M}$ acting on G are collected in the vector F_{ext}.

$$F_{\text{ext}} = \begin{bmatrix} F_{\text{ext}1} \\ F_{\text{ext}2} \\ \mathcal{M} \end{bmatrix} \tag{2.17}$$

Mass, damping, and stiffness coefficients are collected into the matrices m, c, and k respectively. According to assumptions 2 and 3, these matrices are diagonal with constant coefficients.

Conservation of momentum for the rigid body is then written as:

$$m\,\alpha + c\,\nu + k\,\delta = F_{\text{ext}} \tag{2.18}$$

2.3 Interface coupling conditions

The velocities and accelerations of the solid and fluid particles are constrained to be equal at the interface. This kinematic compatibility condition is described in section 2.3.1. The computation of the resultant of fluid forces on the solid at the interface is described in 2.3.2. This latter computation is related to compatibility by principle of virtual work.

2.3.1 Compatibility condition

Let A be a point belonging to the interface (see figure 2.1). Because A is part of the rigid body, its displacement d^A is related to δ through:

$$d^A = \underbrace{\begin{bmatrix}\delta_1\\ \delta_2\end{bmatrix}}_{\text{rigid body displacement}} + \underbrace{\begin{bmatrix}\cos(\theta)-1 & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)-1\end{bmatrix}\begin{bmatrix}x_0^A\\ y_0^A\end{bmatrix}}_{\text{rigid body rotation}} \tag{2.19}$$

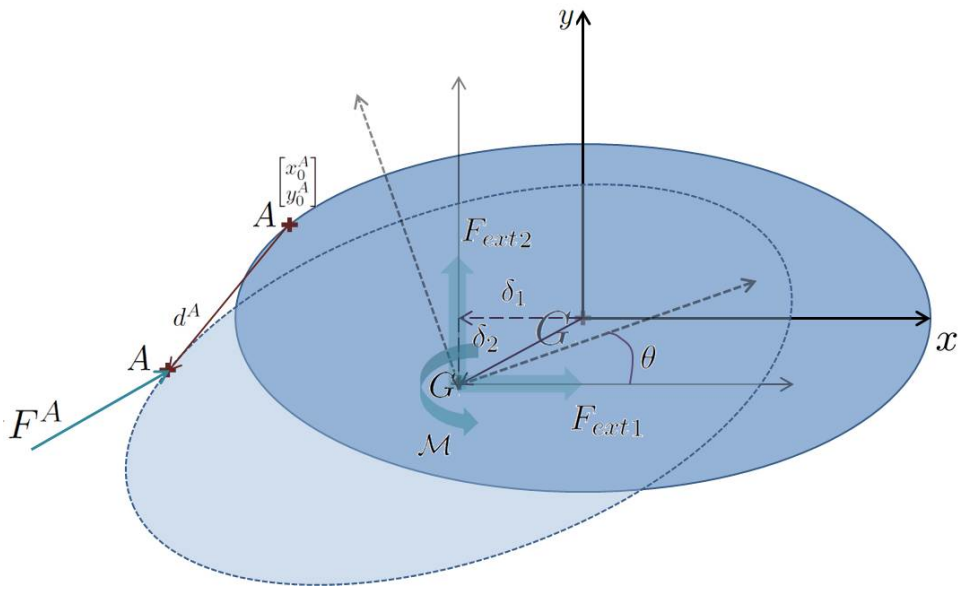

Figure 2.1: Initial and displaced configurations of the rigid body

x_0^A and y_0^A are the initial coordinates of A in the global coordinate system, which has the initial position of G as the origin.

Solid velocity at point A is calculated by differentiating equation 2.19. The first compatibility condition requires the fluid velocity to be equal to the solid velocity at point A. This leads to:

$$v^A = \begin{bmatrix} 1 & 0 & -\sin(\theta)\, x_0^A - \cos(\theta)\, y_0^A \\ 0 & 1 & \cos(\theta)\, x_0^A - \sin(\theta)\, y_0^A \end{bmatrix} \begin{bmatrix} \dot{\delta}_1 \\ \dot{\delta}_2 \\ \dot{\theta} \end{bmatrix} = {T^A}^T \nu \tag{2.20}$$

The second compatibility condition requires the fluid acceleration at point A to be equal to the solid acceleration. The latter is obtained by further differentiating equation 2.20:

$$a^A = {T^A}^T \begin{bmatrix} \ddot{\delta}_1 \\ \ddot{\delta}_2 \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -\cos(\theta)\, x_0^A + \sin(\theta)\, y_0^A \\ -\sin(\theta)\, x_0^A - \cos(\theta)\, y_0^A \end{bmatrix} \dot{\theta}^2 \tag{2.21}$$

2.3.2 Resultant force on the solid

F_{ext} is defined as the vector of resultant external forces and moments acting on G. With F^A defined as the external forces acting on the solid at point A,

$$F_{\text{ext}} = \sum_{A \in \text{interface}} T^A F^A \tag{2.22}$$

With f^A defined as the force acting on the fluid at point A, equation 2.23 is due to Newton's third law:

$$F^A = -f^A \tag{2.23}$$

Combining equations 2.22 and 2.23 leads to the equilibrium condition 2.24:

$$F_{\text{ext}} = \sum_{A \in \text{interface}} T^A \left(-f^A\right) \tag{2.24}$$

2.4 Summary of the fluid-rigid body interaction governing equations

Momentum conservation for the rigid body

$$m\,\alpha + c\,\nu + k\,\delta = F_{\text{ext}} \tag{2.25}$$

Conservation of mass for the fluid

$$\frac{\partial v_k}{\partial \chi_k}\left(\Phi(\mathcal{X},t),t\right) = 0 \tag{2.26}$$

Conservation of momentum for the fluid

$$\rho\left[\frac{\partial \tilde{v}_i}{\partial t}\left(\mathcal{X},t\right) + \tilde{c}_k\left(\mathcal{X},t\right)\frac{\partial v_i}{\partial \chi_k}\left(\Phi(\mathcal{X},t),t\right)\right] = \frac{\partial \sigma_{ij}}{\partial \chi_j}\left(\Phi(\mathcal{X},t),t\right) + \rho\,\tilde{b}_i\left(\mathcal{X},t\right) \tag{2.27}$$

Constitutive relationship for the fluid

$$\sigma_{ij}\left(\Phi(\mathcal{X},t),t\right) = -p\left(\Phi(\mathcal{X},t),t\right)\,\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial \chi_j}\left(\Phi(\mathcal{X},t),t\right) + \frac{\partial v_j}{\partial \chi_i}\left(\Phi(\mathcal{X},t),t\right)\right) \tag{2.28}$$

Interface compatibility conditions

$$v^A = \begin{bmatrix} 1 & 0 & -sin(\theta)\,x_0^A - \cos(\theta)\,y_0^A \\ 0 & 1 & \cos(\theta)\,x_0^A - sin(\theta)\,y_0^A \end{bmatrix} \begin{bmatrix} \dot{\delta}_1 \\ \dot{\delta}_2 \\ \dot{\theta} \end{bmatrix} = {T^A}^T \nu \tag{2.29}$$

$$a^A = T^{A^T} \begin{bmatrix} \ddot{\delta}_1 \\ \ddot{\delta}_2 \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -\cos(\theta)\,x_0^A + \sin(\theta)\,y_0^A \\ -\sin(\theta)\,x_0^A - \cos(\theta)\,y_0^A \end{bmatrix} \dot{\theta}^2 \tag{2.30}$$

Resultant force on solid

$$F_{\text{ext}} = \sum_{A \in \text{interface}} T^A\left(-f^A\right) \tag{2.31}$$

Chapter 3

Finite Element discretization and numerical solution

Introduction

The problem defined by the governing equations 2.25 to 2.31 can not be solved analytically. A numerical solution procedure is needed. Section 3.1 discretizes the governing equations in space and in time to obtain a set of discrete equations representing the problem. Section 3.2 develops the integration procedure to solve these discrete equations.

3.1 Setting up a numerically solvable problem

Whereas the solid part of the problem has only three degrees of freedom, the fluid part has an infinite number of degrees of freedom. The fluid domain is thus discretized to compute the solution at a finite number of locations. The spatially discrete equations are obtained using the Finite Element method, as detailed in paragraph 3.1.1. Using the solid governing equation and the coupling conditions, the final set of semi-dicrete equations is derived in paragraph 3.1.2. These equations are further discretized in time in paragraph 3.1.3.

3.1.1 From the continuous to the discrete flow problem

In this section, the partial differential equations 2.26 and 2.27 are discretized in space. First, they are expressed in weak form using the method of weighted residuals. Then, the fluid domain is spatially dicretized, to get the finite element form of the equations.

3.1.1.1 Weak form of the conservation equations

In this section, if a function is in spatial representation, dependency is not shown, but (χ, t) should be understood.

Mass conservation

Equation 2.26 is multiplied by the pressure weighting function δp and integrated over the spatial domain Ω.

$$\int_{\Omega} \frac{\partial v_k}{\partial \chi_k} \, \delta p \; d\Omega = 0 \tag{3.1}$$

Momentum conservation

Equation 2.27 (corresponding to direction i) is multiplied by the i^{th} component of the velocity weighting function δv_i and integrated over the spatial domain Ω.

$$\int_{\Omega} \left(\rho \left[\frac{\partial \tilde{v}_i}{\partial t} \left(\Phi^{-1}(\chi, t), t \right) + \tilde{c}_k \left(\Phi^{-1}(\chi, t), t \right) \frac{\partial v_i}{\partial \chi_k} \right] - \frac{\partial \sigma_{ij}}{\partial \chi_j} \right) \delta v_i \; d\Omega = 0 \tag{3.2}$$

To take the constitutive relationship and boundary conditions into account, term $\int_{\Omega} \frac{\partial \sigma_{ij}}{\partial \chi_j} \delta v_i \; d\Omega$ needs to be integrated by parts:

$$\int_{\Omega} \frac{\partial \sigma_{ij}}{\partial \chi_j} \delta v_i \; d\Omega = - \int_{\Omega} \sigma_{ij} \frac{\partial \delta v_i}{\partial \chi_j} \; d\Omega + \int_{\partial \Omega} \sigma_{ij} \, n_j \, \delta v_i \; d\partial\Omega \tag{3.3}$$

Introducing the constitutive relationship 2.28 into equation 3.3 leads to:

$$\int_{\Omega} \frac{\partial \sigma_{ij}}{\partial \chi_j} \delta v_i \; d\Omega = - \int_{\Omega} \left[-p \, \delta_{ij} + \mu \left(\frac{\partial v_i}{\partial \chi_j} + \frac{\partial v_j}{\partial \chi_i} \right) \right] \frac{\partial \delta v_i}{\partial \chi_j} \; d\Omega + \int_{\partial \Omega} \sigma_{ij} \, n_j \, \delta v_i \; d\partial\Omega \tag{3.4}$$

The boundary of the fluid domain $\partial\Omega$ is split into a prescribed i-direction traction part $\partial\Omega_{Ti}$ and prescribed i-direction velocity part $\partial\Omega_{vi}$. On $\partial\Omega_{Ti}$, $\sigma_{ij} \, n_j = T_i$. On $\partial\Omega_{vi}$, $\delta v_i = 0$. Substituting these results into equation 3.4, we get:

$$\int_\Omega \frac{\partial \sigma_{ij}}{\partial \chi_j} \delta v_i \; d\Omega = -\int_\Omega \left[-p\,\delta_{ij} + \mu \left(\frac{\partial v_i}{\partial \chi_j} + \frac{\partial v_j}{\partial \chi_i} \right) \right] \frac{\partial \delta v_i}{\partial \chi_j} \; d\Omega + \int_{\partial \Omega_{T\,i}} T_i \, \delta v_i \; d\partial\Omega \tag{3.5}$$

Introducing this result into equation 3.2, we get the final weak form of the momentum equation:

$$\int_\Omega \rho \frac{\partial \tilde{v}_i}{\partial t} \left(\Phi^{-1}(\chi, t), t \right) \delta v_i \; d\Omega + \int_\Omega \rho \, \tilde{c}_k \left(\Phi^{-1}(\chi, t), t \right) \frac{\partial v_i}{\partial \chi_k} \delta v_i \; d\Omega + \int_\Omega \mu \left(\frac{\partial v_i}{\partial \chi_j} + \frac{\partial v_j}{\partial \chi_i} \right) \frac{\partial \delta v_i}{\partial \chi_j} \; d\Omega$$
$$- \int_\Omega p \, \delta_{ij} \frac{\partial \delta v_i}{\partial \chi_j} \; d\Omega = \int_{\partial \Omega_{T\,i}} T_i \, \delta v_i \; d\partial\Omega \tag{3.6}$$

Equations 3.1 and 3.6 are the final weak forms that will serve as basis for spatial discretization.

3.1.1.2 Spatial discretization

Discretization of the fluid domain into finite elements

The current domain (2D) is discretized into n_{el} bilinear quadrilateral finite elements as:

$$\Omega^h = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \Omega^e \tag{3.7}$$

Ω^h refers to the discrete approximation of the spatial domain Ω. Ω^e represents the element domain.

Interpolations

The discrete solution is computed at ALE observer locations. Thus, the shape functions $\tilde{N}(\mathcal{X})$ used for the global coordinates and velocity interpolations are expressed in terms of the ALE labels. The acceleration term in the balance of momentum equation contains $\frac{\partial \tilde{v}(\mathcal{X},t)}{\partial t}$ (see equation 2.27). Using $\tilde{N}(\mathcal{X})$ as shape functions, time dependence only appears in the discrete velocity vector.

The pressure is treated separately as piecewise constant (constant in each element), for stability reasons.[1]

In practice, the fields and weighting functions are interpolated at the element level, using the parent coordinate system $\xi = \begin{bmatrix} \xi_1 & \xi_2 \end{bmatrix}^T$. Matrices and vectors necessary to build the discrete equations are then assembled. An isoparametric formulation is used, meaning that the same shape functions interpolate the ALE observers labels $\mathcal{X}$ and the fluid velocity. The mapping function from the parent coordinates to the ALE coordinates inside an element is given by equation 3.8. The mapping function relating the element coordinates to the spatial coordinates inside an element is given by equation 3.9. Nodal quantities are denoted with a bar.

$$\psi_{\mathcal{X}} : \xi \mapsto \mathcal{X} = \mathbf{N}(\xi)\bar{\mathcal{X}} \tag{3.8}$$

$$\psi_{\chi} : (\xi, t) \mapsto \chi(t) = \mathbf{N}(\xi)\bar{\chi}(t) \tag{3.9}$$

$\mathbf{N}$ is the matrix containing the shape functions, defined by equation 3.10. $\bar{\mathcal{X}}$ is the vector of the ALE labels of the nodes. $\bar{\chi}$ is the vector of spatial labels of the nodes.

$$\mathbf{N} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \tag{3.10}$$

The shape function N_a associated to node a is given by equation 3.11, where ξ_1^a and ξ_2^a are the parent coordinates of the node a (given on figure 3.1).

$$N_a = \frac{1}{4}\left(1 + \xi_1^a \xi_1\right)\left(1 + \xi_2^a \xi_2\right) \tag{3.11}$$

At the element level, the trial functions (velocity and pressure) are interpolated as :

$$\tilde{v}^e\left(\psi_{\mathcal{X}}(\xi), t\right) = \mathbf{N}(\xi)\, \bar{v}^e(t) \tag{3.12}$$

[1] Too many pressure degrees of freedom risk to prevent the uniqueness of the solution.

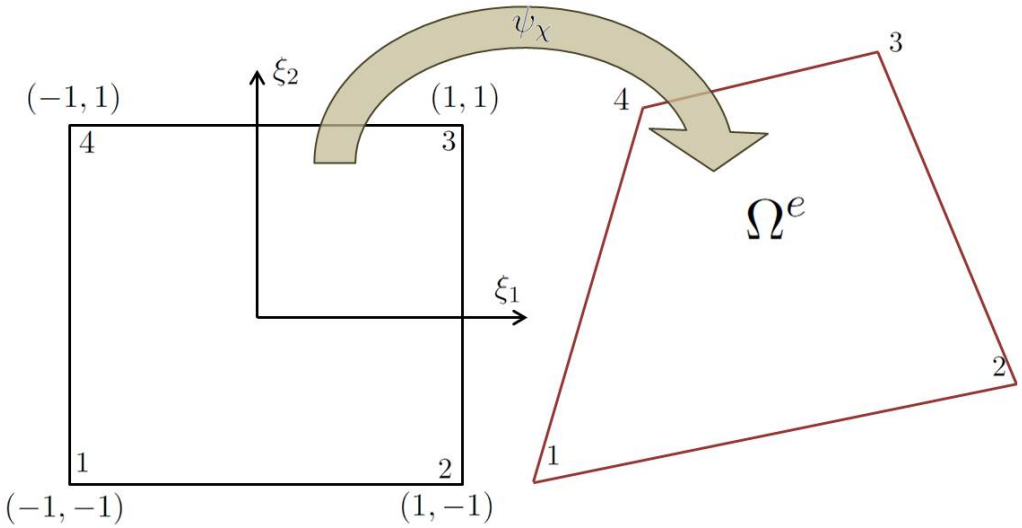

Figure 3.1: Representation of an element in the parent coordinate system and in the global coordinate system associated with spatial observers

$$\tilde{p}^e\left(\psi_{\mathcal{X}}(\xi),t\right) = \mathbf{P}(\xi)\,\bar{p}^e(t) \tag{3.13}$$

Since the pressure is interpolated as constant in each element, $\mathbf{P}(\xi) = 1$ and $\bar{p}^e$ is just the value of the pressure for the concerned element. Equation 3.13 becomes:

$$\tilde{p}^e\left(\psi_{\mathcal{X}}(\xi),t\right) = \bar{p}^e(t) \tag{3.14}$$

The interpolations for the test functions are:

$$\delta v^e\left(\psi_\chi(\xi,t)\right) = \mathbf{N}(\xi)\,\bar{\delta v}_e(t) \tag{3.15}$$

$$\delta p^e\left(\psi_\chi(\xi,t)\right) = \mathbf{P}(\xi)\,\bar{\delta p}_e(t) = \bar{\delta p}_e(t) \tag{3.16}$$

Note that here, the same shape functions are used to interpolate the trial and weighting (or test) functions. In [22] however, different functions are used, which are based on the SUPG formulation described in [6].

Discrete continuity equation

The discrete approximation of the weak form of the continuity equation (also called conservation of mass) 3.1 is written as:

$$\int_{\Omega^h} \frac{\partial v_k}{\partial \chi_k}\, \delta p \; d\Omega = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \int_{\Omega^e} \frac{\partial v^e{}_k}{\partial \chi_k}\, \delta p^e \; d\Omega^e = 0 \tag{3.17}$$

Equation 3.17 appears to be completely written in spatial representation. But the ALE representation implicitly comes in through $\psi_\chi = \Phi \circ \psi_\mathcal{X}$. $\psi_\mathcal{X}$ relates the element coordinates ξ to the ALE coordinates $\mathcal{X}$, taken as reference to compute the discrete solution. Each node's reference coordinate is related to its updated (spatial) coordinate through the mesh motion Φ, prescribed by the user.[2]

$\frac{\partial v^e{}_k}{\partial \chi_k}$ is expressed as:

$$\frac{\partial v^e{}_k}{\partial \chi_k} = \mathtt{d}\mathbf{N}\left(\psi_\chi^{-1}(\chi, t)\right) \bar{v}^e(t) \tag{3.18}$$

where $\mathtt{d}$ is the operator defined as:

$$\mathtt{d} = \begin{bmatrix} \frac{\partial}{\partial \chi_1} & \frac{\partial}{\partial \chi_2} \end{bmatrix} \tag{3.19}$$

Substituting the result 3.18 and the interpolation for δp^e defined by equation 3.16 into equation 3.17 leads to:

$$\int_{\Omega^h} \frac{\partial v_k}{\partial \chi_k}\, \delta p \; d\Omega = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\bar{\delta p}_e \int_{\Omega^e} \mathtt{d}\mathbf{N}\left(\psi_\chi^{-1}(\chi, t)\right) \; d\Omega^e \; \bar{v}^e \right] = 0 \tag{3.20}$$

After assembly of the global discrete test pressure vector $\bar{\delta p} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \bar{\delta p}_e$ and the global discrete velocity vector $\bar{v} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \bar{v}^e$, equation 3.20 becomes :

[2] Chapter 4, section 4.1 explains how the nodal coordinates are updated in practice

$$\bar{\delta p} \underset{e=1}{\overset{n_{\text{el}}}{\mathbf{A}}} \left[\int_{\Omega^e} \mathtt{dN} \left(\psi_\chi^{-1}(\chi, t) \right) \, d\Omega^e \right] \, \bar{v} = 0 \tag{3.21}$$

Since the components of the discrete weighting function vector can be varied independently, this leads to the discrete continuity equation 3.22:

$$\underset{e=1}{\overset{n_{\text{el}}}{\mathbf{A}}} \left[\int_{\Omega^e} \mathtt{dN} \left(\psi_\chi^{-1}(\chi, t) \right) \, d\Omega^e \right] \, \bar{v} = \mathbb{G}^T \, \bar{v} = 0 \tag{3.22}$$

Discrete momentum conservation equation

The discrete approximation of the weak form of the conservation of momentum 3.6 is written as:

$$\begin{aligned} & \int_{\Omega^h} \rho \, \tfrac{\partial \tilde{v}_i}{\partial t} \delta v_i \, d\Omega + \int_{\Omega^h} \rho \, \tilde{c}_k \, \tfrac{\partial v_i}{\partial \chi_k} \delta v_i \, d\Omega + \int_{\Omega^h} \mu \left(\tfrac{\partial v_i}{\partial \chi_j} + \tfrac{\partial v_j}{\partial \chi_i} \right) \tfrac{\partial \delta v_i}{\partial \chi_j} \, d\Omega - \int_{\Omega^h} p \, \delta_{ij} \tfrac{\partial \delta v_i}{\partial \chi_j} \, d\Omega \\ & \qquad - \int_{\partial \Omega_T{}^h} T_i \, \delta v_i \, d\partial\Omega \\ = \; & \underset{e=1}{\overset{n_{\text{el}}}{\mathbf{A}}} \left[\int_{\Omega^e} \rho \, \tfrac{\partial \tilde{v}_i^e}{\partial t} \delta v_i \, d\Omega^e + \int_{\Omega^e} \rho \, \tilde{c}_k^e \, \tfrac{\partial v^e{}_i}{\partial \chi_k} \delta v^e{}_i \, d\Omega^e + \int_{\Omega^e} \mu \left(\tfrac{\partial v_i}{\partial \chi_j} + \tfrac{\partial v_j}{\partial \chi_i} \right) \tfrac{\partial \delta v_i}{\partial \chi_j} \, d\Omega^e - \int_{\Omega^e} p \, \delta_{ij} \tfrac{\partial \delta v_i}{\partial \chi_j} \, d\Omega^e \right] \\ & \qquad - \int_{\partial \Omega_T{}^h} T_i \, \delta v_i \, d\partial\Omega \end{aligned} \tag{3.23}$$

To explain the derivation of the discrete equation as clearly as possible, each term of equation 3.23 is treated separately.

Mass term Using the interpolations 3.12 and 3.15 for the velocity trial and test functions respectively, the first term in equation 3.23 becomes:

$$\int_{\Omega^e} \rho \frac{\partial \tilde{v}_i^e}{\partial t} \delta v_i^e \, d\Omega^e = \bar{\delta v}_e^T \rho \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e \, \frac{d\bar{v}^e}{dt} = \rho \, \bar{\delta v}_e^T \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e \, \bar{a}_e \tag{3.24}$$

Convective term The same interpolation as 3.12 is used for the convective velocity vector, that is:

$$\tilde{c}^e \left(\psi_{\mathcal{X}}(\xi), t \right) = \mathbf{N}(\xi) \, \bar{c}_e \tag{3.25}$$

This interpolation and equations 3.12 and 3.15 for $\bar{v}^e$ and $\bar{\delta v}_e$ cause the second term of equations 3.23 to become:

$$\int_{\Omega^e} \rho\, \tilde{c}_k^e \frac{\partial {v^e}_i}{\partial \chi_k} \delta {v^e}_i \, d\Omega^e = \bar{\delta v}_e^T \rho \int_{\Omega^e} \mathbf{N}^T \sum_{k=1}^{2} (\mathbf{N}\bar{c}_e)_k \frac{\partial \mathbf{N}}{\partial \chi_k} \, d\Omega^e \, \bar{v}^e \tag{3.26}$$

Viscosity term Similarly, using the interpolations 3.12 and 3.15 for the velocity trial and test functions respectively, the third term in equation 3.23 becomes:

$$\mu \int_{\Omega^e} \left(\frac{\partial \left(\mathbf{N}\bar{v}^e\right)_i}{\partial \chi_j} + \frac{\partial \left(\mathbf{N}\bar{v}^e\right)_j}{\partial \chi_i} \right) \frac{\partial \left(\mathbf{N}\bar{\delta v}_e\right)_i}{\partial \chi_j} \, d\Omega^e \tag{3.27}$$

To express this equation in a friendlier form, the operators $\mathtt{D_1}$ and $\mathtt{D_2}$ need to be defined:

$$\mathtt{D_1} = \begin{bmatrix} \frac{\partial}{\partial \chi_1} & 0 \\ 0 & \frac{\partial}{\partial \chi_1} \\ \frac{\partial}{\partial \chi_2} & 0 \\ 0 & \frac{\partial}{\partial \chi_2} \end{bmatrix} \tag{3.28}$$

$$\mathtt{D_2} = \begin{bmatrix} \frac{\partial}{\partial \chi_1} & 0 \\ \frac{\partial}{\partial \chi_2} & 0 \\ 0 & \frac{\partial}{\partial \chi_1} \\ 0 & \frac{\partial}{\partial \chi_2} \end{bmatrix} \tag{3.29}$$

Note that:

$$\mathtt{D_1}\delta v = \begin{bmatrix} \frac{\partial}{\partial \chi_1} & 0 \\ 0 & \frac{\partial}{\partial \chi_1} \\ \frac{\partial}{\partial \chi_2} & 0 \\ 0 & \frac{\partial}{\partial \chi_2} \end{bmatrix} \begin{bmatrix} \delta v_1 \\ \delta v_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial \delta v_1}{\partial \chi_1} \\ \frac{\partial \delta v_2}{\partial \chi_1} \\ \frac{\partial \delta v_1}{\partial \chi_2} \\ \frac{\partial \delta v_2}{\partial \chi_2} \end{bmatrix} \tag{3.30}$$

$$\mathtt{D}_2 v = \begin{bmatrix} \frac{\partial}{\partial \chi_1} & 0 \\ \frac{\partial}{\partial \chi_2} & 0 \\ 0 & \frac{\partial}{\partial \chi_1} \\ 0 & \frac{\partial}{\partial \chi_2} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial v_1}{\partial \chi_1} \\ \frac{\partial v_1}{\partial \chi_2} \\ \frac{\partial v_2}{\partial \chi_1} \\ \frac{\partial v_2}{\partial \chi_2} \end{bmatrix} \tag{3.31}$$

Thus

$$\frac{\partial v_i}{\partial \chi_j}\frac{\partial \delta v_i}{\partial \chi_j} + \frac{\partial v_j}{\partial \chi_i}\frac{\partial \delta v_i}{\partial \chi_j} = \left(\mathtt{D}_1 \delta v\right)^T \left(\mathtt{D}_1 v\right) + \left(\mathtt{D}_1 \delta v\right)^T \left(\mathtt{D}_2 v\right) \tag{3.32}$$

Equation 3.27 becomes then:

$$\mu \int_{\Omega^e} \left(\frac{\partial \left(\mathbf{N}\bar{v}^e\right)_i}{\partial \chi_j} + \frac{\partial \left(\mathbf{N}\bar{v}^e\right)_j}{\partial \chi_i} \right) \frac{\partial \left(\mathbf{N}\bar{\delta v}_e\right)_i}{\partial \chi_j}\, d\Omega^e = \bar{\delta v}_e^T \mu \int_{\Omega^e} \left[\left(\mathtt{D}_1 \mathbf{N}\right)^T \left(\mathtt{D}_1 \mathbf{N}\right) + \left(\mathtt{D}_1 \mathbf{N}\right)^T \left(\mathtt{D}_2 \mathbf{N}\right) \right] d\Omega^e\ \bar{v}^e \tag{3.33}$$

Pressure term Using the interpolation 3.15 for δv, the result 3.18, and the interpolation 3.14 for p, the fourth term in equation 3.23 becomes:

$$\int_{\Omega^e} p\, \delta_{ij} \frac{\partial \delta v_i}{\partial \chi_j}\, d\Omega^e = \int_{\Omega^e} p\, \frac{\partial \delta v_i}{\partial \chi_i}\, d\Omega^e = \bar{\delta v}_e^T \int_{\Omega^e} \left(\mathtt{d}\mathbf{N}\right)^T\, d\Omega^e\, \bar{p}^e \tag{3.34}$$

External force term The remaining terms in equation 3.23 compose the discrete external force vector f.

Gathering the results 3.24, 3.26, 3.33 and 3.34 and substituting them into equation 3.23 leads to:

$$\begin{aligned} \mathop{\mathbf{A}}_{e=1}^{n_{\mathrm{el}}} \bar{\delta v}_e^T \Bigg[& \rho \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e\ \bar{a}_e + \rho \int_{\Omega^e} \mathbf{N}^T \sum_{k=1}^{2} \left(\mathbf{N}\bar{c}_e\right)_k \frac{\partial \mathbf{N}}{\partial \chi_k}\, d\Omega^e\, \bar{v}^e \\ & + \mu \int_{\Omega^e} \left[\left(\mathtt{D}_1 \mathbf{N}\right)^T \left(\mathtt{D}_1 \mathbf{N}\right) + \left(\mathtt{D}_1 \mathbf{N}\right)^T \left(\mathtt{D}_2 \mathbf{N}\right) \right] d\Omega^e\ \bar{v}^e - \int_{\Omega^e} \left(\mathtt{d}\mathbf{N}\right)^T\, d\Omega^e\, \bar{p}^e \Bigg] - f = 0 \end{aligned} \tag{3.35}$$

After assembling the global discrete weighting velocity vector $\bar{\delta v}$, the global discrete acceleration vector $\bar{a}$, velocity vector $\bar{v}$ and pressure vector $\bar{p}$, equation 3.35 becomes:

$$\bar{\delta v}^T \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\rho \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e \right] \bar{a} + \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\rho \int_{\Omega^e} \mathbf{N}^T \sum_{k=1}^{2} \left(\mathbf{N}\bar{c}_e\right)_k \frac{\partial \mathbf{N}}{\partial \chi_k} d\Omega^e \right.$$
$$\left. + \mu \int_{\Omega^e} \left[\left(\mathtt{D}_1\mathbf{N}\right)^T \left(\mathtt{D}_1\mathbf{N}\right) + \left(\mathtt{D}_1\mathbf{N}\right)^T \left(\mathtt{D}_2\mathbf{N}\right) \right] d\Omega^e \right] \bar{v} - \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\int_{\Omega^e} \left(\mathtt{d}\mathbf{N}\right)^T \, d\Omega^e \right] \bar{p} - f = 0 \quad (3.36)$$

The mass matrix $\mathbb{M}$ and the convection-viscosity matrix $\mathbb{N}$ are defined as:

$$\mathbb{M} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\rho \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e \right] \quad (3.37)$$

$$\mathbb{N} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \left[\rho \int_{\Omega^e} \mathbf{N}^T \sum_{k=1}^{2} \left(\mathbf{N}\bar{c}_e\right)_k \frac{\partial \mathbf{N}}{\partial \chi_k} d\Omega^e + \mu \int_{\Omega^e} \left[\left(\mathtt{D}_1\mathbf{N}\right)^T \left(\mathtt{D}_1\mathbf{N}\right) + \left(\mathtt{D}_1\mathbf{N}\right)^T \left(\mathtt{D}_2\mathbf{N}\right) \right] d\Omega^e \right] \quad (3.38)$$

Using these equations and the definition of $\mathbb{G}$ given by 3.22, equation 3.36 becomes:

$$\mathbb{M}\bar{a} + \mathbb{N}\bar{v} + \mathbb{G}\bar{p} = f \quad (3.39)$$

Section 3.1.1.3 details the expression of the elementary parts of the matrices $\mathbb{G}$, $\mathbb{M}$ and $\mathbb{N}$.

3.1.1.3 Expression of the element matrices

Matrices $\mathbb{G}$, $\mathbb{M}$ and $\mathbb{N}$ defined by equations 3.22, 3.37 and 3.38 are actually built element-wise, that is :

$$\mathbb{G} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \mathbb{G}^e \qquad \mathbb{M} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \mathbb{M}^e \qquad \mathbb{N} = \mathop{\mathbf{A}}_{e=1}^{n_{\text{el}}} \mathbb{N}^e \quad (3.40)$$

This section details the derivation of the element matrices $\mathbb{G}^e$, $\mathbb{M}^e$ and $\mathbb{N}^e$, necessary for a practical calculation. The actual implementation of the element matrices computation and of the assembly process are presented in chapter 4.

Element gradient matrix $\mathbb{G}^e$ It is an 8×1 matrix, defined by (see equation 3.22):

$$\mathbb{G}^e = \int_{\Omega^e} (\mathtt{d}\mathbf{N})^T \; d\Omega^e \tag{3.41}$$

For each node a of the element, the corresponding 2×1 matrix $\mathbb{G}^e_a$ is given by:

$$\mathbb{G}^e_a = \int_{\Omega^e} \begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} \\ \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} \; d\Omega^e \tag{3.42}$$

Transfer to the element coordinate system

Since the computations are made at the element level, the integral in equation 3.42 needs to be converted to the element coordinate system ξ.

Applying the chain rule to $\frac{\partial N_a}{\partial \chi_i}$ leads to:

$$\frac{\partial N_a}{\partial \chi_i} = \frac{\partial N_a}{\partial \xi_k} \frac{\partial \psi_{\chi k}^{-1}}{\partial \chi_i} \tag{3.43}$$

$$\begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} \\ \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \psi_{\chi 1}^{-1}}{\partial \chi_1} & \frac{\partial \psi_{\chi 2}^{-1}}{\partial \chi_1} \\ \frac{\partial \psi_{\chi 1}^{-1}}{\partial \chi_2} & \frac{\partial \psi_{\chi 2}^{-1}}{\partial \chi_2} \end{bmatrix} \begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} \\ \frac{\partial N_a}{\partial \xi_2} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial \psi_{\chi 1}}{\partial \xi_1} & \frac{\partial \psi_{\chi 2}}{\partial \xi_1} \\ \frac{\partial \psi_{\chi 1}}{\partial \xi_2} & \frac{\partial \psi_{\chi 2}}{\partial \xi_2} \end{bmatrix}^{-1}}_{E} \begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} \\ \frac{\partial N_a}{\partial \xi_2} \end{bmatrix} \tag{3.44}$$

E^{-1} needs to be calculated, because ψ_χ^{-1} is not known explicitly.

$$E^{-1} = \frac{1}{\det(E)} \left[\mathrm{Com}(E)\right]^T \tag{3.45}$$

Thus

$$E^{-1} = \frac{1}{J_\chi} \begin{bmatrix} \frac{\partial \psi_{\chi 2}}{\partial \xi_2} & -\frac{\partial \psi_{\chi 2}}{\partial \xi_1} \\ -\frac{\partial \psi_{\chi 1}}{\partial \xi_2} & \frac{\partial \psi_{\chi 1}}{\partial \xi_1} \end{bmatrix} \tag{3.46}$$

Where J_χ is the Jacobian of the transformation from the element coordinates to the global coordinates attached to the spatial observers. J_χ is given by:

$$J_\chi = \frac{\partial \psi_{\chi 1}}{\partial \xi_1}\frac{\partial \psi_{\chi 2}}{\partial \xi_2} - \frac{\partial \psi_{\chi 2}}{\partial \xi_1}\frac{\partial \psi_{\chi 1}}{\partial \xi_2} \tag{3.47}$$

The expression of the derivatives $\frac{\partial \psi_{\chi i}}{\partial \xi_j}$ ($i = 1, 2$ and $j = 1, 2$) are given in appendix A.

Inserting the result from equation 3.46 into equation 3.44 leads to

$$\begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} \\ \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} = \frac{1}{J_\chi} \begin{bmatrix} \frac{\partial \psi_{\chi 2}}{\partial \xi_2} & -\frac{\partial \psi_{\chi 2}}{\partial \xi_1} \\ -\frac{\partial \psi_{\chi 1}}{\partial \xi_2} & \frac{\partial \psi_{\chi 1}}{\partial \xi_1} \end{bmatrix} \begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} \\ \frac{\partial N_a}{\partial \xi_2} \end{bmatrix} \tag{3.48}$$

Now, changing variables in equation 3.42 gives

$$\mathbb{G}_a^e = \int_{-1}^{1}\int_{-1}^{1} \frac{1}{J_\chi} \begin{bmatrix} \frac{\partial \psi_{\chi 2}}{\partial \xi_2} & -\frac{\partial \psi_{\chi 2}}{\partial \xi_1} \\ -\frac{\partial \psi_{\chi 1}}{\partial \xi_2} & \frac{\partial \psi_{\chi 1}}{\partial \xi_1} \end{bmatrix} \begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} \\ \frac{\partial N_a}{\partial \xi_2} \end{bmatrix} J_\chi d\xi \tag{3.49}$$

That is to say

$$\mathbb{G}_a^e = \int_{-1}^{1}\int_{-1}^{1} \underbrace{\begin{bmatrix} \frac{\partial \psi_{\chi 2}}{\partial \xi_2} & -\frac{\partial \psi_{\chi 2}}{\partial \xi_1} \\ -\frac{\partial \psi_{\chi 1}}{\partial \xi_2} & \frac{\partial \psi_{\chi 1}}{\partial \xi_1} \end{bmatrix}}_{\mathcal{D}^T} \underbrace{\begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} \\ \frac{\partial N_a}{\partial \xi_2} \end{bmatrix}}_{d\mathcal{N}_a^T} d\xi \tag{3.50}$$

Element mass matrix $\mathbb{M}^e$ It is an 8×8 matrix defined by (see equation 3.37):

$$\mathbb{M}^e = \rho \int_{\Omega^e} \mathbf{N}\left(\psi_\chi^{-1}(\chi, t)\right)^T \mathbf{N}\left(\psi_\chi^{-1}(\chi, t)\right) d\Omega^e \tag{3.51}$$

Transfer to the element coordinate system

The process is simpler than for the element gradient matrix $\mathbb{G}^e$ since no spatial derivatives are implied here. With J_χ defined by equation 3.47, changing variable in equation 3.51 leads to:

$$\mathbb{M}^e = \rho \int_{\Omega^e} \mathbf{N}(\xi)^T \mathbf{N}(\xi)\, J_\chi(\xi) d\Omega^e \tag{3.52}$$

Element convection-viscosity matrix $\mathbb{N}^e$ It is an 8×8 matrix defined by (see equation 3.38):

$$\mathbb{N}^e = \rho \int_{\Omega^e} \mathbf{N}^T \sum_{k=1}^{2} (\mathbf{N}\bar{c}_e)_k \frac{\partial \mathbf{N}}{\partial \chi_k} \, d\Omega^e + \mu \int_{\Omega^e} \left[(\mathtt{D_1}\mathbf{N})^T (\mathtt{D_1}\mathbf{N}) + (\mathtt{D_1}\mathbf{N})^T (\mathtt{D_2}\mathbf{N}) \right] d\Omega^e \tag{3.53}$$

It consists of 16 2×2 matrices, each corresponding to a pair (a, b) of element nodes. Such a block $\mathbb{N}^e_{ab}$ is given by:

$$\begin{aligned} \mathbb{N}^e_{ab} = \rho \left(\int_{\Omega^e} N_a \sum_{k=1}^{2} (\mathbf{N}\bar{c}_e)_k \frac{\partial N_b}{\partial \chi_k} \, d\Omega^e \right) \mathbf{I} \\ + \mu \int_{\Omega^e} \left(\begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} & \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} \begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix}^T \mathbf{I} + \begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} & \frac{\partial N_a}{\partial \chi_2} \end{bmatrix}^T \begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix} \right) d\Omega^e \end{aligned} \tag{3.54}$$

This can be rewritten as:

$$\begin{aligned} \mathbb{N}^e_{ab} = \rho \left(\int_{\Omega^e} N_a \begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix} (\mathbf{N}\bar{c}_e) \, d\Omega^e \right) \mathbf{I} \\ + \mu \int_{\Omega^e} \left(\begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} & \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} \begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix}^T \mathbf{I} + \begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} & \frac{\partial N_a}{\partial \chi_2} \end{bmatrix}^T \begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix} \right) d\Omega^e \end{aligned} \tag{3.55}$$

Transfer to element coordinates

As seen in the derivation of the element gradient matrix $\mathbb{G}^e$ (see equations 3.42 to 3.50), the vector $\begin{bmatrix} \frac{\partial N_b}{\partial \chi_1} & \frac{\partial N_b}{\partial \chi_2} \end{bmatrix}$ can be expressed in terms of the element coordinates ξ as:

$$\begin{bmatrix} \frac{\partial N_a}{\partial \chi_1} & \frac{\partial N_a}{\partial \chi_2} \end{bmatrix} = \frac{1}{J_\chi} \begin{bmatrix} \frac{\partial N_a}{\partial \xi_1} & \frac{\partial N_a}{\partial \xi_2} \end{bmatrix} \begin{bmatrix} \frac{\partial \psi_{\chi 2}}{\partial \xi_2} & -\frac{\partial \psi_{\chi 1}}{\partial \xi_2} \\ -\frac{\partial \psi_{\chi 2}}{\partial \xi_1} & \frac{\partial \psi_{\chi 1}}{\partial \xi_1} \end{bmatrix} = \frac{1}{J_\chi} d\mathcal{N}_a \mathcal{D} \tag{3.56}$$

Now, changing variable in equation 3.55 leads to:

$$\begin{aligned} \mathbb{N}^e_{ab} = \rho \left(\int_{-1}^{1} \int_{-1}^{1} N_a \frac{1}{J_\chi} \left[d\mathcal{N}_b \mathcal{D} \right] (\mathbf{N}\bar{c}_e) \, J_\chi d\xi \right) \mathbf{I} \\ + \mu \int_{-1}^{1} \int_{-1}^{1} \left(\frac{1}{J_\chi} \left[d\mathcal{N}_a \mathcal{D} \right] \frac{1}{J_\chi} \left[d\mathcal{N}_b \mathcal{D} \right]^T \mathbf{I} + \frac{1}{J_\chi} \left[d\mathcal{N}_a \mathcal{D} \right]^T \frac{1}{J_\chi} \left[d\mathcal{N}_b \mathcal{D} \right] \right) J_\chi d\xi \end{aligned} \tag{3.57}$$

That is to say:

$$\mathbb{N}^e_{ab} = \rho \left(\int_{-1}^{1} \int_{-1}^{1} N_a \left[d\mathcal{N}_b \mathcal{D} \right] (\mathbf{N}\bar{c}_e) \, d\xi \right) \mathbf{I} + \mu \int_{-1}^{1} \int_{-1}^{1} \frac{1}{J_\chi} \left(\left[d\mathcal{N}_a \mathcal{D} \right] \left[d\mathcal{N}_b \mathcal{D} \right]^T \mathbf{I} + \left[d\mathcal{N}_a \mathcal{D} \right]^T \left[d\mathcal{N}_b \mathcal{D} \right] \right) d\xi \tag{3.58}$$

3.1.1.4 Numerical integration

The element matrices are expressed by the integral of functions over a 2D domain. In practice, these integrals are not calculated continuously, but approximated by numerical integration.This is performed by adding the values of the integrand evaluated at specific points.

The two-by-two Gausssian quadrature rule is used for numerical integration by [22], who follows the approach of [6]. The general formula for this rule is given by [11]:

$$\int_{-1}^{1}\int_{-1}^{1} g(\xi_1,\xi_2)\,d\xi = g\left(-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right) + g\left(\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right) + g\left(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right) + g\left(-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right) \quad (3.59)$$

3.1.1.5 Conclusion: summary of the fluid spatial discrete equations

This section has presented the derivation of the discrete fluid equations 3.60 and 3.61. Starting from the continuous Navier-Stokes equations, the weak forms have been expressed. Then the finite element approximation has been detailed. All the formulas to build the discrete matrices element-wise have been derived. The assembly process will be presented in chapter 4, completing the tools to build the discrete equations, summarized below.

Spatially discrete conservation of mass

$$\mathbb{G}^T\bar{v} = 0 \quad (3.60)$$

Spatially discrete conservation of momentum

$$\mathbb{M}\bar{a} + \mathbb{N}\bar{v} + \mathbb{G}\bar{p} = f \quad (3.61)$$

To set up the final semi-discrete problem, these equations need to be coupled with the solid equations through the conditions 2.29, 2.30 and 2.31. This is the subject of the next section.

3.1.2 Coupled semi-discrete equations

In this section, fluid semi-discrete equations 3.60 and 3.61 are coupled with the solid governing equation 2.25. To do so, these equations are first reorganized, to separate the interface from the rest of the fluid domain. Then, the interface conditions 2.29, 2.30 and 2.31 are expressed in a discrete form to enable the coupling.

3.1.2.1 Reorganization of the fluid semi-discrete equations

Fluid degrees of freedom coming from the finite element discretization are distributed into three sets. Interface degrees of freedom are marked by the superscript I. The rest of the fluid domain is further split into free degrees of freedom (marked by the superscript f) and prescribed degrees of freedom (marked by the superscript p).

Equations 3.60 and 3.61 are then reorganized as:

$$\begin{bmatrix} \mathbb{G}^{fT} & \mathbb{G}^{pT} & \mathbb{G}^{IT} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ \bar{v}^I \end{bmatrix} = 0 \tag{3.62}$$

$$\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \\ \mathbb{M}^{pf} & \mathbb{M}^{pp} & \mathbb{M}^{pI} \\ \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f \\ \bar{a}^p \\ \bar{a}^I \end{bmatrix} + \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \\ \mathbb{N}^{pf} & \mathbb{N}^{pp} & \mathbb{N}^{pI} \\ \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ \bar{v}^I \end{bmatrix} - \begin{bmatrix} \mathbb{G}^f \\ \mathbb{G}^p \\ \mathbb{G}^I \end{bmatrix} \bar{p} = \begin{bmatrix} f^f \\ f^p \\ f^I \end{bmatrix} \tag{3.63}$$

In these equations, $\bar{a}^f$, $\bar{v}^f$ and f^p are unknown. $\bar{a}^p$, $\bar{v}^p$ and f^f are prescribed. Since we are interested in solving for the fluid velocity and acceleration, and not for the reaction forces f^p, the second row of equation 3.63 can be ignored, leading to:

$$\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \\ \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f \\ \bar{a}^p \\ \bar{a}^I \end{bmatrix} + \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \\ \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ \bar{v}^I \end{bmatrix} - \begin{bmatrix} \mathbb{G}^f \\ \mathbb{G}^I \end{bmatrix} \bar{p} = \begin{bmatrix} f^f \\ f^I \end{bmatrix} \tag{3.64}$$

$\bar{a}^I$, $\bar{v}^I$ and f^I will be related to the solid degrees of freedom and forces through the discrete coupling conditions, developed in section 3.1.2.2.

3.1.2.2 Discrete expression of the coupling conditions

Because of the spatial discretization of the fluid domain, the interface comprises a finite number of nodes n_{n_I}.

Interface compatibility conditions At a node A, the compatibility conditions 2.29 and 2.30 are written as:

$$v^A = \begin{bmatrix} 1 & 0 & -\sin(\theta)\, x_0^A - \cos(\theta)\, y_0^A \\ 0 & 1 & \cos(\theta)\, x_0^A - \sin(\theta)\, y_0^A \end{bmatrix} \begin{bmatrix} \dot{\delta}_1 \\ \dot{\delta}_2 \\ \dot{\theta} \end{bmatrix} = {T^A}^T \nu \tag{3.65}$$

$$a^A = {T^A}^T \begin{bmatrix} \ddot{\delta}_1 \\ \ddot{\delta}_2 \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -\cos(\theta)\, x_0^A + \sin(\theta)\, y_0^A \\ -\sin(\theta)\, x_0^A - \cos(\theta)\, y_0^A \end{bmatrix} \dot{\theta}^2 = {T^A}^T \alpha + A^A \dot{\theta}^2 \tag{3.66}$$

Gathering all the matrices T^A in the transformation matrix T (size $3 \times 2\, n_{n_I}$), $\bar{v}^I$ is related to ν by:

$$\bar{v}^I = T^T \nu \tag{3.67}$$

Furthermore, gathering all the matrices A^A in a matrix A (size $2\, n_{n_I} \times 1$), $\bar{a}^I$ is related to α and $\dot{\theta}$ by:

$$\bar{a}^I = T^T \alpha + A\, \dot{\theta}^2 \tag{3.68}$$

Resultant force on the solid

$$F_{\text{ext}} = \sum_{A \in \text{interface}} T^A \left(-f^A\right) = T \left(-f^I\right) \tag{3.69}$$

3.1.2.3 Coupled semi-discrete equations

Substituting the interface equilibrium condition 3.69 into the solid momentum conservation equation 2.25 leads to:

$$m\,\alpha + c\,\nu + k\,\delta = T\left(-f^I\right) \tag{3.70}$$

Now, substituting the second row of equation 3.64 into equation 3.70 leads to:

$$m\,\alpha + c\,\nu + k\,\delta = -T\begin{bmatrix}\mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II}\end{bmatrix}\begin{bmatrix}\bar{a}^f\\ \bar{a}^p\\ \bar{a}^I\end{bmatrix} - T\begin{bmatrix}\mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II}\end{bmatrix}\begin{bmatrix}\bar{v}^f\\ \bar{v}^p\\ \bar{v}^I\end{bmatrix} + T\mathbb{G}^I\bar{p} \tag{3.71}$$

Substituting the compatibility conditions 3.67 and 3.68 into the first row of equation 3.64 and into equation 3.71 leads to:

$$\begin{bmatrix}\mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI}\end{bmatrix}\begin{bmatrix}\bar{a}^f\\ \bar{a}^p\\ T^T\alpha + A\,\dot{\theta}^2\end{bmatrix} + \begin{bmatrix}\mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI}\end{bmatrix}\begin{bmatrix}\bar{v}^f\\ \bar{v}^p\\ T^T\nu\end{bmatrix} - \mathbb{G}^f\bar{p} = f^f \tag{3.72}$$

$$m\,\alpha + c\,\nu + k\,\delta = -T\begin{bmatrix}\mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II}\end{bmatrix}\begin{bmatrix}\bar{a}^f\\ \bar{a}^p\\ T^T\alpha + A\,\dot{\theta}^2\end{bmatrix} - T\begin{bmatrix}\mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II}\end{bmatrix}\begin{bmatrix}\bar{v}^f\\ \bar{v}^p\\ T^T\nu\end{bmatrix} + T\mathbb{G}^I\bar{p} \tag{3.73}$$

Recalling the continuity equation 3.62 and rearranging equations 3.72 and 3.73 leads to the final set of coupled semi-discrete equations:

Coupled semi-discrete equations

$$\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \end{bmatrix} \begin{bmatrix} \bar{a}^f \\ \bar{a}^p \\ T^T\alpha + A\,\dot{\theta}^2 \end{bmatrix} + \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ T^T\nu \end{bmatrix} - \mathbb{G}^f\bar{p} = f^f \tag{3.74}$$

$$\left(m + T\mathbb{M}^{II}T^T\right)\alpha + c\nu + k\,\delta$$

$$= -T \begin{bmatrix} \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f \\ \bar{a}^p \\ A\,\dot{\theta}^2 \end{bmatrix} - T \begin{bmatrix} \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ T^T\nu \end{bmatrix} + T\mathbb{G}^I\bar{p} \tag{3.75}$$

$$\begin{bmatrix} \mathbb{G}^{fT} & \mathbb{G}^{pT} & \mathbb{G}^{IT} \end{bmatrix} \begin{bmatrix} \bar{v}^f \\ \bar{v}^p \\ T^T\nu \end{bmatrix} = 0 \tag{3.76}$$

3.1.3 Time discretization

Equations 3.74 to 3.76 are discrete in space. They need to be further discretized in time in order to be solved numerically.

3.1.3.1 Discretization of time derivatives

Fluid discrete acceleration vector is defined as (recall equation 3.24):

$$\bar{a} = \frac{d\bar{v}}{dt} \tag{3.77}$$

For the numerical solution procedure, continuous time is discretized in a sequence of intervals Δt. At time $t_n = n\,\Delta t$, the fluid acceleration and viscosity are respectively denoted by $\bar{a}_n$ and $\bar{v}_n$. Using these notations, the time derivative of the fluid velocity is approximated as:

$$\frac{d\bar{v}}{dt} \approx \frac{\bar{v}_{n+1} - \bar{v}_n}{\Delta t} \tag{3.78}$$

Using Newmark's method, equation 3.77 is then discretized in time as:

$$\frac{\bar{v}_{n+1} - \bar{v}_n}{\Delta t} = (1 - \gamma_v)\, \bar{a}_n + \gamma_v \bar{a}_{n+1} \tag{3.79}$$

Thus

$$\bar{v}_{n+1} = \bar{v}_n + \Delta t \left[(1 - \gamma_v)\, \bar{a}_n + \gamma_v \bar{a}_{n+1}\right] = \bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}_{n+1} \tag{3.80}$$

Where $\bar{v}^*_{n+1}$ collects the known part in $\bar{v}_{n+1}$, which is a combination of the previous solutions $\bar{a}_n$ and $\bar{v}_n$ at time t_n.

Following the same idea, solid velocity and displacement at time t_{n+1} are given by:

$$\nu_{n+1} = \nu_n + \Delta t \left[(1 - \gamma)\, \alpha_n + \gamma\, \alpha_{n+1}\right] = \nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1} \tag{3.81}$$

$$\delta_{n+1} = \delta_n + \Delta t\, \nu_n + \frac{\Delta t^2}{2} \left[(1 - 2\beta)\, \alpha_n + 2\beta\, \alpha_{n+1}\right] = \delta^*_{n+1} + \Delta t^2\, \beta\, \alpha_{n+1} \tag{3.82}$$

For stability reasons, values of the Newmark's method parameters are chosen as $\gamma_v = 1/2$, $\gamma = 1/2$ and $\beta = 1/4$.

3.1.3.2 Derivation of the discrete equations

The semi-discrete equations 3.74 to 3.76 are written at time t_{n+1} as:

$$\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ T^T \alpha_{n+1} + A\, \dot{\theta}^2_{n+1} \end{bmatrix} + \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \end{bmatrix} \begin{bmatrix} \bar{v}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \nu_{n+1} \end{bmatrix} - \mathbb{G}^f \bar{p}_{n+1} = f^f \tag{3.83}$$

$$\begin{aligned} &\left(m + T\mathbb{M}^{II} T^T\right) \alpha_{n+1} + c\nu_{n+1} + k\, \delta_{n+1} \\ &\quad = -T \begin{bmatrix} \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ A\, \dot{\theta}^2_{n+1} \end{bmatrix} - T \begin{bmatrix} \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \nu_{n+1} \end{bmatrix} + T\mathbb{G}^I \bar{p}_{n+1} \end{aligned} \tag{3.84}$$

$$\begin{bmatrix} \mathbb{G}^{f^T} & \mathbb{G}^{p^T} & \mathbb{G}^{I^T} \end{bmatrix} \begin{bmatrix} \bar{v}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \nu_{n+1} \end{bmatrix} = 0 \tag{3.85}$$

Note that all the matrices $\mathbb{M}$, $\mathbb{N}$, $\mathbb{G}$, T and A, as well as the force vector f, depend on time. The subscript $n+1$ is not written here because the matrices at time n are not needed.

Substituting time discretizations 3.80, 3.81 and 3.82 into equations 3.83 to 3.85 leads to[3] :

$$\begin{aligned} &\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ T^T \alpha_{n+1} + A \left(\dot{\theta}^*_{n+1} + \Delta t\, \gamma\, \ddot{\theta}_{n+1} \right)^2 \end{bmatrix} \\ &+ \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1} \right) \end{bmatrix} \\ &- \mathbb{G}^f \bar{p}_{n+1} = f^f \end{aligned} \tag{3.86}$$

$$\begin{aligned} &\left(m + T \mathbb{M}^{II} T^T \right) \alpha_{n+1} + c \left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1} \right) + k \left(\delta^*_{n+1} + \Delta t^2\, \beta\, \alpha_{n+1} \right) \\ &\qquad = - T \begin{bmatrix} \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ A \left(\dot{\theta}^*_{n+1} + \Delta t\, \gamma\, \ddot{\theta}_{n+1} \right)^2 \end{bmatrix} \\ &\qquad\quad - T \begin{bmatrix} \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1} \right) \end{bmatrix} \\ &\qquad\quad + T \mathbb{G}^I \bar{p}_{n+1} \end{aligned} \tag{3.87}$$

[3] Note that only the free part of the fluid velocity is substituted, since the prescribed part is known.

$$\begin{bmatrix} \mathbb{G}^{f^T} & \mathbb{G}^{p^T} & \mathbb{G}^{I^T} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1} \right) \end{bmatrix} = 0 \tag{3.88}$$

3.1.4 Concluding remarks

From the continuous governing equations summarized in 2.4, the discrete system of equations has been derived. Starting with the weak form of Navier-Stokes equations, the spatially discrete fluid governing equations have been set up. Coupled with the solid governing equation, the discrete fluid equations led to a coupled semi-discrete system of equations. The latter needed to be further discretized in time to enable the numerical solution process. The resulting nonlinear system of equations that remains to be solved is summarized below.

Residual equations

$$
\begin{aligned}
r_1 = &\begin{bmatrix} \mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ T^T \alpha_{n+1} + A \left(\dot{\theta}^*_{n+1} + \Delta t \, \gamma \, \ddot{\theta}_{n+1} \right)^2 \end{bmatrix} \\
&+ \begin{bmatrix} \mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t \, \gamma_v \, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t \, \gamma \, \alpha_{n+1} \right) \end{bmatrix} \\
&- \mathbb{G}^f \bar{p}_{n+1} - f^f = 0
\end{aligned}
\tag{3.89}
$$

$$
\begin{aligned}
r_2 = &\left(m + T \mathbb{M}^{II} T^T \right) \alpha_{n+1} + c \left(\nu^*_{n+1} + \Delta t \, \gamma \, \alpha_{n+1} \right) + k \left(\delta^*_{n+1} + \Delta t^2 \, \beta \, \alpha_{n+1} \right) \\
&+ T \begin{bmatrix} \mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II} \end{bmatrix} \begin{bmatrix} \bar{a}^f_{n+1} \\ \bar{a}^p_{n+1} \\ A \left(\dot{\theta}^*_{n+1} + \Delta t \, \gamma \, \ddot{\theta}_{n+1} \right)^2 \end{bmatrix} \\
&+ T \begin{bmatrix} \mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t \, \gamma_v \, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t \, \gamma \, \alpha_{n+1} \right) \end{bmatrix} - T \mathbb{G}^I \bar{p}_{n+1} = 0
\end{aligned}
\tag{3.90}
$$

$$
r_3 = \begin{bmatrix} \mathbb{G}^{f^T} & \mathbb{G}^{p^T} & \mathbb{G}^{I^T} \end{bmatrix} \begin{bmatrix} \bar{v}^*_{n+1} + \Delta t \, \gamma_v \, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T \left(\nu^*_{n+1} + \Delta t \, \gamma \, \alpha_{n+1} \right) \end{bmatrix} = 0 \tag{3.91}
$$

Unknown variables are $\bar{a}^f_{n+1}$, α_{n+1} and p_{n+1}. At each step, $\bar{v}^f_{n+1}$, ν_{n+1} and δ_{n+1} are calculated using equations 3.80 to 3.82. Matrices $\mathbb{M}$, $\mathbb{N}$, $\mathbb{G}$, T and A depend on the current solution. To solve this nonlinear system of equation, Newton's method is used. The process is developed in section 3.2.

3.2 Numerical solution

3.2.1 Introduction

In this section, the numerical procedure to solve the system of equations 3.89 to 3.91 is explained. [22] uses a predictor-multicorrector algorithm, based on [6] and [11]. This algorithm can be shown to come from Newton's method, with an approximated Jacobian matrix. This derivation is the subject of paragraph 3.2.2. The solution procedure is then summarized in paragraph 3.2.3.

3.2.2 Time integration by Newton's method

3.2.2.1 Exact Newton's method derivation

At each Newton iteration, linear system 3.92 needs to be solved.

$$\underbrace{\begin{bmatrix} \frac{\partial r_1}{\partial \bar{a}^f_{n+1}} & \frac{\partial r_1}{\partial \alpha_{n+1}} & \frac{\partial r_1}{\partial \bar{p}_{n+1}} \\ \frac{\partial r_2}{\partial \bar{a}^f_{n+1}} & \frac{\partial r_2}{\partial \alpha_{n+1}} & \frac{\partial r_2}{\partial \bar{p}_{n+1}} \\ \frac{\partial r_3}{\partial \bar{a}^f_{n+1}} & \frac{\partial r_3}{\partial \alpha_{n+1}} & \frac{\partial r_3}{\partial \bar{p}_{n+1}} \end{bmatrix}}_{B_{\text{full}}} \begin{bmatrix} \Delta a \\ \Delta \alpha \\ \Delta p \end{bmatrix} = \begin{bmatrix} -r_1 \\ -r_2 \\ -r_3 \end{bmatrix} \tag{3.92}$$

The variables Δa, $\Delta \alpha$ and Δp represent the fluid acceleration, solid acceleration, and pressure increments respectively.

To calculate the exact Jacobian matrix, we need to know which terms of equations r_1, r_2, and r_3 depend on which variable. Matrices m, c and k are constant. $\mathbb{M}$, $\mathbb{N}$ and $\mathbb{G}$ depend on α_{n+1} through δ_{n+1}. More precisely, δ_{n+1} is needed to calculate the new position of the nodes, which is necessary to compute the derivatives of the shape function (see section 3.1.1.3). $\mathbb{N}$ also depends on $\bar{a}^f_{n+1}$ through the convective velocity $\bar{c}$. T and A depend on α_{n+1} through θ_{n+1} (see paragraph 3.1.2.2). None of the matrices depend on the pressure $\bar{p}_{n+1}$.

Taking these into account, the derivatives appearing in equation 3.92 can now be computed. The subscripts $n+1$ are dropped to lighten the notations.

$$\frac{\partial r_1}{\partial \bar{a}^f} = \mathbb{M}^{ff} + \mathbb{N}^{ff}\Delta t\gamma_v + \frac{\partial}{\partial \bar{a}^f}\left(\begin{bmatrix}\mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} \tag{3.93}$$

To compute the derivatives of r_1 and r_2 with respect to α, we need to distinguish between α_1 or α_2 (corresponding to $\ddot{\delta}_1$ and $\ddot{\delta}_2$ respectively) and α_3 (corresponding to $\ddot{\theta}$).

$$\begin{aligned}\frac{\partial r_1}{\partial \alpha_{1-2}} = {} & \mathbb{M}^{fI}T^T + \mathbb{N}^{fI}T^T\Delta t\,\gamma + \mathbb{M}^{fI}\frac{\partial T^T}{\partial \alpha_{1-2}}\alpha_{1-2} \\ & + \frac{\partial}{\partial \alpha_{1-2}}\left(\begin{bmatrix}\mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI}\end{bmatrix}\right)\begin{bmatrix}\bar{a}^f \\ \bar{a}^p \\ T^T\alpha + A\left(\dot{\theta}^* + \Delta t\,\gamma\,\ddot{\theta}\right)^2\end{bmatrix} \\ & + \frac{\partial}{\partial \alpha_{1-2}}\left(\begin{bmatrix}\mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} + \frac{\partial \mathbb{G}^f}{\partial \alpha_{1-2}}\end{aligned} \tag{3.94}$$

$$\begin{aligned}\frac{\partial r_1}{\partial \alpha_3} = {} & \mathbb{M}^{fI}T^T + \mathbb{N}^{fI}T^T\Delta t\,\gamma + \mathbb{M}^{fI}\frac{\partial T^T}{\partial \alpha_3}\alpha_3 + \mathbb{M}^{fI}A2\Delta t\gamma\left(\dot{\theta}^* + \Delta t\,\gamma\,\alpha_3\right) \\ & + \frac{\partial}{\partial \alpha_3}\left(\begin{bmatrix}\mathbb{M}^{ff} & \mathbb{M}^{fp} & \mathbb{M}^{fI}\end{bmatrix}\right)\begin{bmatrix}\bar{a}^f \\ \bar{a}^p \\ T^T\alpha + A\left(\dot{\theta}^* + \Delta t\,\gamma\,\ddot{\theta}\right)^2\end{bmatrix} \\ & + \frac{\partial}{\partial \alpha_3}\left(\begin{bmatrix}\mathbb{N}^{ff} & \mathbb{N}^{fp} & \mathbb{N}^{fI}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} + \frac{\partial \mathbb{G}^f}{\partial \alpha_3}\end{aligned} \tag{3.95}$$

$$\frac{\partial r_1}{\partial \bar{p}} = -\mathbb{G}^f \tag{3.96}$$

$$\frac{\partial r_2}{\partial \bar{a}^f} = T\mathbb{M}^{If} + T\mathbb{N}^{If}\ \Delta t\,\gamma_v + T\frac{\partial}{\partial \bar{a}^f}\left(\begin{bmatrix}\mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} \tag{3.97}$$

$$\begin{aligned}\frac{\partial r_2}{\partial \alpha_{1-2}} &= m + T\mathbb{M}^{II}T^T + c\Delta t\gamma + k\Delta t^2\beta + \frac{\partial}{\partial \alpha_{1-2}}\left(T\mathbb{M}^{II}T^T\right)\alpha_{1-2} + T\mathbb{N}^{II}T^T\Delta t\,\gamma \\ &+ T\mathbb{N}^{II}\frac{\partial T^T}{\partial \alpha_{1-2}}\left(\nu^* + \Delta t\,\gamma\,\alpha\right) + \frac{\partial}{\partial \alpha_{1-2}}\left(T\begin{bmatrix}\mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II}\end{bmatrix}\right)\begin{bmatrix}\bar{a}^f \\ \bar{a}^p \\ A\left(\dot{\theta}^*_{n+1} + \Delta t\,\gamma\,\ddot{\theta}\right)^2\end{bmatrix} \\ &+ \frac{\partial}{\partial \alpha_{1-2}}\left(T\begin{bmatrix}\mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} - \frac{\partial\left(T\mathbb{G}^f\right)}{\partial \alpha_{1-2}}\bar{p}\end{aligned} \tag{3.98}$$

$$\begin{aligned}\frac{\partial r_2}{\partial \alpha_3} &= m + T\mathbb{M}^{II}T^T + c\Delta t\gamma + k\Delta t^2\beta + \frac{\partial}{\partial \alpha_3}\left(T\mathbb{M}^{II}T^T\right)\alpha_3 + T\mathbb{N}^{II}T^T\Delta t\,\gamma \\ &+ T\mathbb{M}^{II}2A\Delta t\gamma\left(\dot{\theta}^*_3 + \Delta t\,\gamma\,\ddot{\theta}\right) + T\mathbb{N}^{II}\frac{\partial T^T}{\partial \alpha_3}\left(\nu^* + \Delta t\,\gamma\,\alpha\right) \\ &+ \frac{\partial}{\partial \alpha_3}\left(T\begin{bmatrix}\mathbb{M}^{If} & \mathbb{M}^{Ip} & \mathbb{M}^{II}\end{bmatrix}\right)\begin{bmatrix}\bar{a}^f \\ \bar{a}^p \\ A\left(\dot{\theta}^*_3 + \Delta t\,\gamma\,\ddot{\theta}\right)^2\end{bmatrix} \\ &+ \frac{\partial}{\partial \alpha_3}\left(T\begin{bmatrix}\mathbb{N}^{If} & \mathbb{N}^{Ip} & \mathbb{N}^{II}\end{bmatrix}\right)\begin{bmatrix}\bar{v}^* + \Delta t\,\gamma_v\,\bar{a}^f \\ \bar{v}^p \\ T^T\left(\nu^* + \Delta t\,\gamma\,\alpha\right)\end{bmatrix} - \frac{\partial\left(T\mathbb{G}^f\right)}{\partial \alpha_3}\bar{p}\end{aligned} \tag{3.99}$$

$$\frac{\partial r_2}{\partial \bar{p}} = -T\mathbb{G}^I \tag{3.100}$$

$$\frac{\partial r_3}{\partial \bar{a}^f} = {\mathbb{G}^f}^T \Delta t\, \gamma_v + \frac{\partial}{\partial \bar{a}^f}\left(\begin{bmatrix}{\mathbb{G}^f}^T & {\mathbb{G}^p}^T & {\mathbb{G}^I}^T\end{bmatrix}\right)\begin{bmatrix}\bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T\left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1}\right)\end{bmatrix} \tag{3.101}$$

$$\begin{aligned}\frac{\partial r_3}{\partial \alpha} = {\mathbb{G}^I}^T T^T \Delta t\, \gamma + {\mathbb{G}^I}^T \frac{\partial T^T}{\partial \bar{a}^f}\left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1}\right) \\ + \frac{\partial}{\partial \alpha}\left(\begin{bmatrix}{\mathbb{G}^f}^T & {\mathbb{G}^p}^T & {\mathbb{G}^I}^T\end{bmatrix}\right)\begin{bmatrix}\bar{v}^*_{n+1} + \Delta t\, \gamma_v\, \bar{a}^f_{n+1} \\ \bar{v}^p_{n+1} \\ T^T\left(\nu^*_{n+1} + \Delta t\, \gamma\, \alpha_{n+1}\right)\end{bmatrix}\end{aligned} \tag{3.102}$$

$$\frac{\partial r_3}{\partial \bar{p}} = 0 \tag{3.103}$$

3.2.2.2 Method based on the approximated Jacobian

In practice, the linear system of equations solved at each iteration is not the complete one presented in paragraph 3.2.2.1. The matrix B_{full} in equation 3.92 is approximated by a matrix B, in which the derivatives of the matrices $\mathbb{M}$, $\mathbb{N}$, $\mathbb{G}$, T and A are ignored. The off-diagonal terms $\partial r_1/\partial\alpha$ and $\partial r_2/\partial \bar{a}^f$ are also set aside. In addition, the terms $T\mathbb{N}^{II}T^T\Delta t\,\gamma$ and $T\mathbb{M}^{II}2A\Delta t\gamma\left(\dot{\theta}^*_3 + \Delta t\,\gamma\,\ddot{\theta}\right)$ are ignored in $\partial r_2/\partial\alpha$.

These assumptions lead to the following linear system of equations:

$$\underbrace{\begin{bmatrix}\mathbb{M}^{ff} + \mathbb{N}^{ff}\Delta t\,\gamma_v & 0 & -\mathbb{G}^f \\ 0 & \left(m + T\mathbb{M}^{II}T^T\right) + c\Delta t\,\gamma + k\Delta t^2\beta & -T\mathbb{G}^I \\ {\mathbb{G}^f}^T\Delta t\,\gamma_v & {\mathbb{G}^I}^T T^T\Delta t\,\gamma & 0\end{bmatrix}}_{B}\begin{bmatrix}\Delta a \\ \Delta\alpha \\ \Delta p\end{bmatrix} = \begin{bmatrix}-r_1 \\ -r_2 \\ -r_3\end{bmatrix} \tag{3.104}$$

The equations of the solution phase presented in [22] are equivalent to solving this system. [22] does not solve it by directly inverting the matrix B but by using a procedure based on the Schur complement of the lower right block 0. Different ways to solve the linear system 3.104 are discussed in chapter 4.

3.2.3 Summary of the solution procedure

Section 3.2.2 presents the application of Newton's method to solve the nonlinear system of equations 3.89 to 3.91. The complete solution procedure in one time step is summarized here to clarify the structure of iterations.

From time t_n to time t_{n+1}

(1) Initial guess :

$$\begin{aligned} \bar{a}^f_{n+1} &= 0 \\ \bar{v}^f_{n+1} &= \bar{v}^*_{n+1} \\ \bar{p}_{n+1} &= \bar{p}_n \alpha_{n+1} = 0 \\ \nu_{n+1} &= \nu^*_{n+1} \\ \delta_{n+1} &= \delta^*_{n+1} \end{aligned} \tag{3.105}$$

Prescribed parts of the fluid acceleration, fluid velocity, solid acceleration, solid velocity, and solid displacement are assigned. They do not change during the Newton iterations. the interface part of the velocity is computed according to equation 3.67. It will be updated at each iteration, since it is needed to compute $\bar{c}$.

(2) Newton iterations [*while criterion > tolerance*]

(a) Compute T and A

(b) Compute the new coordinates of the nodes and $\bar{c}$

(c) Get element matrices $\mathbb{M}^e$, $\mathbb{N}^e$ and $\mathbb{G}^e$

(d) Assemble $\mathbb{M}$, $\mathbb{N}$ and $\mathbb{G}$

(e) Compute the residuals r_1, r_2 and r_3

(f) Solve equation 3.104

(g) Update $\bar{a}^f_{n+1}$, $\bar{v}^f_{n+1}$ and free parts of α_{n+1}, ν_{n+1} and δ_{n+1} as:

$$\begin{aligned}
\bar{a}^f_{n+1} &= \bar{a}^f_{n+1} + \Delta a \\
\bar{v}^f_{n+1} &= \bar{a}^f_{n+1} + \Delta t \gamma_v \Delta a \\
\alpha_{n+1}(\text{free}) &= \alpha_{n+1}(\text{free}) + \Delta\alpha \\
\nu_{n+1}(\text{free}) &= \nu_{n+1}(\text{free}) + \Delta t \gamma \Delta\alpha \\
\delta_{n+1}(\text{free}) &= \delta_{n+1}(\text{free}) + \Delta t^2 \beta \Delta\alpha
\end{aligned} \tag{3.106}$$

Update $\bar{p}_{n+1}$ as $\bar{p}_{n+1} = \bar{p}_{n+1} + \Delta p$

Then update $v^I_{n+1} = T^T \nu_{n+1}$

(3) Compute $\bar{a}^I_{n+1} = T^T \alpha_{n+1} + A\,\dot{\theta}^2_{n+1}$

3.3 Summary of the finite element discretization and numerical solution

In this chapter, the complete discrete system of nonlinear equations governing the fluid-rigid body interaction problem has been established. Starting from the continuous ALE form of the fluid governing equations (derived in chapter 2), the weak form has been written. Spatial discretization using finite elements has then been performed. The coupled semi-discrete equations have been obtained by coupling the semi-discrete fluid-equations to the solid equation of motion through the interface conditions. The semi-discrete equations are then discretized in time, leading to the final discrete equations.

Due to the convective velocity and the dependence of the matrices on the updated node coordinate, the discrete equations are nonlinear. A solution procedure based on an approximated Newton's method has been developed to solve the system of nonlinear discrete equations.

The practical implementation of the discrete equations and their solution is detailed in the next chapter.

Chapter 4

Implementation

Introduction

Chapter 3 has established the discrete governing equations of the fluid-rigid body problem and the numerical procedure to solve them. This chapter explains how these equations, as well as the procedure summarized in 3.2.3, are implemented in practice.

The global structure of the code is as follows:

(1) Input

(2) Numbering and reorganization of the DOF

(3) Global calculation of the matrix K needed to compute the mesh motion (see section 4.1)

(4) Allocation of memory for the matrices, mesh velocity and solution

(5) Application of the initial condition

(6) Time loop

  (a) Initial guess for Newton's method

  (b) Assign prescribed part of the solution

  (c) Newton iterations loop

    (i) Compute T, A

(ii) Compute the interface part of the fluid velocity

(iii) Compute the mesh velocity and update the node coordinates

(iv) Loop over the elements

(iv.a) Compute the convective velocity $\bar{c}$

(iv.b) Get element matrices $\mathbb{M}^e$, $\mathbb{N}^e$ and $\mathbb{G}^e$

(iv.c) Assign this element part to the global matrices $\mathbb{M}$, $\mathbb{N}$ and $\mathbb{G}$

(v) Complete the assembly of $\mathbb{M}$, $\mathbb{N}$ and $\mathbb{G}$

(vi) Compute the residuals r_1, r_2 and r_3

(vii) Solve for the accelerations and pressure increments

(viii) Update the free part of the solution

(d) Assign the converged free part of the solution

(e) Assign the interface part of the solution

The input needs to specify the fluid and solid parameters ρ, μ, m, c, and k. Newmark's method parameters γ_v, γ, and β are also given there. The mesh of the fluid domain is defined by the initial coordinates of the nodes and a "boundary condition" matrix that specifies whether each degree of freedom is free, fixed, prescribed (not fixed) or belongs to the interface. A connectivity matrix describes which node belongs to which element, as well as its local position in the element. A "link matrix" relates each node to its closest interface and opposite boundary node (more precision will be given in paragraph 4.1.2). The input code contains the prescribed acceleration, velocity, and force. It also specifies the initial condition for the solution. For time integration, the input code gives the total time over which the solution is computed, the time increment size, and the tolerance for Newton's method.

Selected parts of the code globally presented above are further detailed in the following sections. The purpose is to help the reader understand the implementation and to highlight some interesting efficiency considerations.

In section 4.1, the mesh and choice for the mesh motion are described. This point is specific to the application of the ALE method. It is thus interesting to have a concrete example to better understand the method. Knowing the mesh motion is necessary to update the node coordinates and compute the convective velocity. These are needed for the element matrices computation, detailed in section 4.2. This computation has shown to be time consuming, especially given the large number of elements needed in the numerical example (see chapter 5, section 5.2). Thus, the implementation's design presented in section 4.2 accounts for numerical efficiency. Section 4.3 details the matrices assembly process. First, we explain how the fluid degrees of freedom are sorted out between the free, prescribed and interface ones. Then, the actual assembly routine is briefly described. Section 4.4 discusses how the solution procedure is implemented. Finally, section 4.5 explains how the solution is updated, to be as efficient as possible. Parts of the Matlab code corresponding to each section are available in appendix B.

4.1 An ALE mesh in motion

As mentioned in chapter 3, section 3.1, the fluid domain is discretized in finite elements. The mesh is defined in input by the original coordinates of the nodes and by the connectivity matrix.[1] The main feature of the ALE method is that the nodes do not stay at their original positions, but instead follow a motion prescribed by the user. This section first describes the mesh motion chosen here, which comes from [22]. Then, mesh motion implementation is explained. Corresponding parts of the Matlab code can be seen in appendix B.1.

4.1.1 Choice of the ALE observer's motion

Motion of the interface between the fluid and the rigid body can be easily captured with good accuracy if a Lagrangian description is used. Further away from the interface, an Eulerian description seems more suitable, as it is usually used to study pure fluid problems. Thus, the motion of

[1] Generation of the mesh has not been automated here. It will be explained for each numerical example in chapter 5.

the ALE observers is prescribed as the Lagrangian ones on the interface and as the Eulerian ones far from the interface. In between, ALE observers motion is linearly interpolated, functions of the distance to the interface.

Concretely, let's consider a node A belonging to the fluid domain. If A belonged to the rigid body, its motion would be (see equations 2.19 and 2.20):

$$\begin{aligned} &\textbf{displacement} \quad d^A = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} + \begin{bmatrix} \cos(\theta) - 1 & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) - 1 \end{bmatrix} \begin{bmatrix} x_0^A \\ y_0^A \end{bmatrix} \\ &\textbf{velocity} \qquad\quad v^A = {T^A}^T \nu \end{aligned} \tag{4.1}$$

Defining the number κ^A as equal to 1 when A belongs to the interface, 0 if it belongs to the opposite boundary, and varying linearly in between, the following equation results:

$$\kappa^A = \frac{\text{domain width} - \text{distance of } A \text{ to interface}}{\text{domain width}} \tag{4.2}$$

Mesh motion of node A is prescribed as equation 4.3, with κ^A staying constant during the motion:

$$\begin{aligned} &\textbf{displacement} \quad d^A = \kappa^A \left(\begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} + \begin{bmatrix} \cos(\theta) - 1 & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) - 1 \end{bmatrix} \begin{bmatrix} x_0^A \\ y_0^A \end{bmatrix} \right) \\ &\textbf{velocity} \qquad\quad v^A = \kappa^A \left({T^A}^T \nu \right) \end{aligned} \tag{4.3}$$

Note that this particular prescription of the mesh motion works only for a certain type of mesh. Namely, the user needs to be able to define nodes as belonging to a line relating the interface to its opposite boundary. This is suitable to the numerical example presented in [22] reproduced in chapter 5, which is a cylinder oscillating in a cylindrical fluid domain. The idea would need to be adapted for other kinds of problems, as it has been for the first numerical example presented in chapter 5 that consisted in fluid moving in a rectangular domain.

The next section explains how the mesh motion described by equation 4.3 is implemented in practice.

4.1.2 Translation to implementation

Link matrix The input file defines a "link matrix" that relates each node to the corresponding interface node A_I and to the opposite boundary node A_B (see figure 4.1). This matrix is used to compute κ^A for each node.

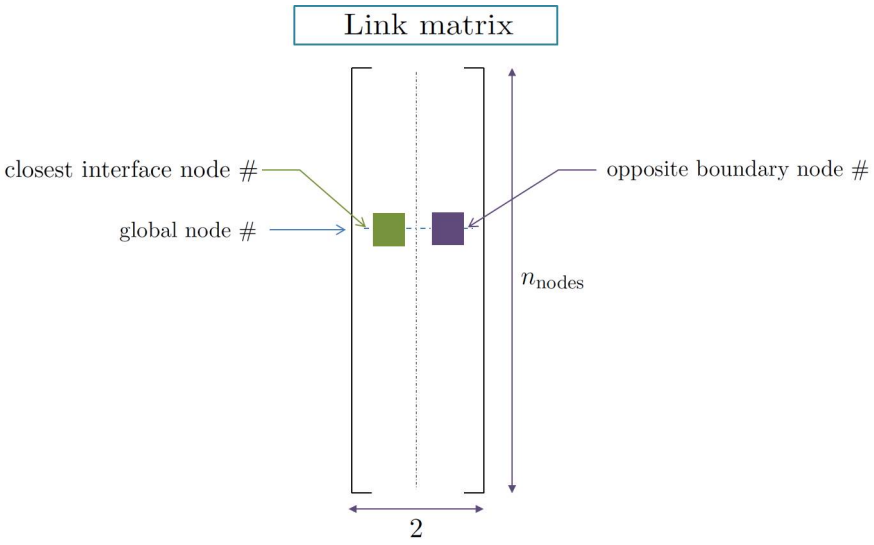

Figure 4.1: Structure of the link matrix

Matrix K This is a $1 \times n_{\text{nodes}}$ array, containing the κ^A. From equation 4.2, κ^A is computed as:

$$\kappa^A = 1 - \frac{\sqrt{\left(x_0^A - x_0^{A_I}\right)^2 + \left(y_0^A - y_0^{A_I}\right)^2}}{\sqrt{\left(x_0^{A_I} - x_0^{A_B}\right)^2 + \left(y_0^{A_I} - y_0^{A_B}\right)^2}} \tag{4.4}$$

Coordinates of the nodes are given by the input matrices X and Y. Thus, the coordinates x_0^A, y_0^A, $x_0^{A_I}$, $y_0^{A_I}$,$x_0^{A_B}$ and $y_0^{A_B}$ are obtained as follows:

$$x_0^A = X(A) \quad y_0^A = Y(A) \tag{4.5}$$

$$x_0^{A_I} = X\left(\text{link matrix}(A,1)\right) \quad y_0^{A_I} = Y\left((\text{link matrix}(A,1)\right) \tag{4.6}$$

$$x_0^{A_B} = X\,(\text{link matrix}(A,2)) \quad y_0^{A_B} = Y\,((\text{link matrix}(A,2)) \tag{4.7}$$

The computation of K is explained here node by node but is vectorized in practice (see code in appendix B.1).

Mesh displacement The $2 \times n_{\text{nodes}}$ array d-A-GLOB contains the mesh displacements in directions x and y for each node. It is computed globally following equation 4.3 (see code in appendix B.1). Node coordinates are then updated in a vectorized fashion.

Mesh velocity Mesh velocity is computed node by node, since the transformation matrix T (defined by equation 3.67) needs to be calculated node by node.

Element loop Updated nodal coordinates and mesh velocity are only needed at the element level, to compute the element matrices, but they are calculated outside the element loop (as seen in the global structure of the code presented in introduction). Parts of these global arrays are then accessed for each element. This way of computing the mesh motion is less time-consuming.[2] Further time-efficiency considerations are developed in the next section.

4.2 Obtaining element matrices

4.2.1 First implementation: an unexpectedly expensive operation

In the first version of the code, equations 3.50, 3.52, 3.58 (respectively defining the element matrices $\mathbb{G}^e$, $\mathbb{M}^e$ and $\mathbb{N}^e$) were directly translated to the implementation, together with the numerical integration equation 3.59. The corresponding Matlab code had the following structure:

For each element

[2] One Newton iteration takes $0.41\,s.$ if nodal position updates and mesh velocity are computed inside the element loop and $0.37\,s.$ as the code is set up.

(1) $\mathcal{D}$ is computed for each Gauss point. The corresponding 4 matrices (since there are 4 Gauss points) are stored in the same array. Parts of this array are accessed when a specific Gauss point is needed. J_χ is computed following the same concept.

(2) Memory for the element matrices is allocated.

(3) Loop over the element nodes a.

  (a) Loop over the Gauss points l

    (i) Product $d\mathcal{N}_a\mathcal{D}(l)$ is computed and stored

    (ii) Part a of the matrix $\mathbb{G}^e(l)$ is computed for each Gauss point, following equation 3.50 and added to the previous part.

  (b) Loop over the element nodes b

    (i) Loop over the Gauss points l

      (i.a) Product $d\mathcal{N}_b\mathcal{D}(l)$ is computed and stored

      (i.b) Part ab of the matrix $\mathbb{N}^e(l)$ is computed for each Gauss point, following equation 3.58 and added to the previous part.

(4) Loop over the Gauss points l

  (a) $\mathbb{M}^e(l)$ is computed according to equation 3.52 and added it to the previous part.

With this implementation, time needed for one iteration, for the numerical example of [22], was $1.600\,s.$. Time needed for one element loop was $3.687 \times 10^{-3}\,s.$ (which, multiplied by the number of elements $n_{\text{el}} = 432$ gave $1.593\,s.$ for all the element loops). Inside the element loop, time needed for the element matrices was $3.376 \times 10^{-3}\,s.$ (which, multiplied by $n_{\text{el}} = 432$ gave $1.458\,s.$). In summary, computing the element matrices took 91% of one iteration's time. In comparison, the solution procedure computed as described in [22] took 1.5% of an iteration.[3] These timing tests showed that the efficiency of the element matrices computation really needed to be improved.

[3] The solution procedure is further discussed in paragraph 4.4.

4.2.2 Improving the code efficiency

The first idea to improve the code efficiency was to avoid repeated access to the same part of an array. For example, instead of accessing the Gauss points coordinates as part of a vector, the coordinates are directly written where they are needed.

This idea also led us to look for a more direct way to compute the matrices. The calculation of $\mathbb{M}^e(l)$ was already vectorized. Most of the time consumption came from the computation of $\mathbb{N}^e(l)$. Thus, a function was built to directly calculate $\mathbb{N}^e(l)$ in terms of the Gauss points, avoiding the a- and b-loops over element nodes. The $a-$loop could be avoided when calculating $\mathbb{G}^e(l)$ too. The goal was to make the computation of the element matrices as explicit as possible to avoid accessing parts of arrays, plus as vectorized as possible to take advantage of Matlab matrix calculus features.

The principle of the new computation is simple: instead of using the following loops:

for $a = 1$ to 4

for $l = 1$ to 4

$$\mathbb{G}^e(2a-1:2a) = \mathbb{G}^e(2a-1:2a) + (d\mathcal{N}_a(l)\mathcal{D}(l))^T \tag{4.8}$$

The matrix $\mathbb{G}^e(l)$ is directly computed as follows:

$$\mathbb{G}^e = \begin{bmatrix} \left(\begin{bmatrix} \frac{\partial N_1}{\partial \xi_1} & \frac{\partial N_1}{\partial \xi_2} \end{bmatrix} \mathcal{D} \right)^T \\ \left(\begin{bmatrix} \frac{\partial N_2}{\partial \xi_1} & \frac{\partial N_2}{\partial \xi_2} \end{bmatrix} \mathcal{D} \right)^T \\ \left(\begin{bmatrix} \frac{\partial N_3}{\partial \xi_1} & \frac{\partial N_3}{\partial \xi_2} \end{bmatrix} \mathcal{D} \right)^T \\ \left(\begin{bmatrix} \frac{\partial N_4}{\partial \xi_1} & \frac{\partial N_4}{\partial \xi_2} \end{bmatrix} \mathcal{D} \right)^T \end{bmatrix} = \begin{bmatrix} (d\mathcal{N}_1\mathcal{D})^T \\ (d\mathcal{N}_2\mathcal{D})^T \\ (d\mathcal{N}_3\mathcal{D})^T \\ (d\mathcal{N}_4\mathcal{D})^T \end{bmatrix} \tag{4.9}$$

The same concept is applied to the calculation of $\mathbb{N}^e$. Furthermore, by looking at equation 3.58, one will realize that two of the terms are numbers that go on the diagonal of each ab block. They

can be calculated in a vectorized fashion, using a 4×4 matrix $\mathbb{N}^e_{\text{diag}}$:

$$\mathbb{N}^e_{\text{diag}} = \left(\rho \begin{bmatrix} d\mathcal{N}_1 \\ d\mathcal{N}_2 \\ d\mathcal{N}_3 \\ d\mathcal{N}_4 \end{bmatrix} \mathcal{D}\mathbf{N}\bar{c} \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \right)^T + \frac{\mu}{J_\chi} \begin{bmatrix} d\mathcal{N}_1 \\ d\mathcal{N}_2 \\ d\mathcal{N}_3 \\ d\mathcal{N}_4 \end{bmatrix} \mathcal{D} \left(\begin{bmatrix} d\mathcal{N}_1 \\ d\mathcal{N}_2 \\ d\mathcal{N}_3 \\ d\mathcal{N}_4 \end{bmatrix} \mathcal{D} \right)^T \tag{4.10}$$

Then, they need to be assigned to the corresponding location in $\mathbb{N}^e$. This can be done through a double loop as follows:

for $a = 1$ to 4

for $b = 1$ to 4

$$\mathbb{N}^e(2a-1:2a, 2b-1:2b) = \mathbb{N}^e_{\text{diag}}(a,b)\mathbf{I} \tag{4.11}$$

But this assignment is similar to the assembly process. So, $\mathbb{N}^e$ is left as only the term reproduced in equation 4.12. $\mathbb{N}^e_{\text{diag}}$ is directly added during the assembly part of the code.

$$\mu \int_{-1}^{1} \int_{-1}^{1} \frac{1}{J_\chi} \left([d\mathcal{N}_a\mathcal{D}] \, [d\mathcal{N}_b\mathcal{D}]^T \, \mathbf{I} + [d\mathcal{N}_a\mathcal{D}]^T \, [d\mathcal{N}_b\mathcal{D}] \right) d\xi \tag{4.12}$$

The calculation of $\mathbb{M}^e$ needs variables that are used for $\mathbb{G}^e$ and $\mathbb{N}^e$ (namely the shape functions $\mathbf{N}$ and the Jacobian determinant J_χ). Thus, all these matrices are gathered in the same function *element-matrices2* (see code in appendix B.2) to avoid repeated calculation.

After all the changes, time needed to create and assemble the element matrices for one element is 0.644×10^{-3}. It was 3.405×10^{-3} before, a factor of five times slower.

Now that the computation of the element matrices has been explained, the next section develops how the global matrices $\mathbb{M}$, $\mathbb{N}$ and $\mathbb{G}$ are assembled.

4.3 Assembling the matrices

4.3.1 Reorganizing the fluid degrees of freedom

Fluid degrees of freedom need to be sorted out between the free ones, prescribed ones, and the ones belonging to the interface, in order to build the discrete equations (recall chapter 3, section 3.1.2). In the input file, the boundary condition matrix BC specifies each DOF's category (see figure 4.2). It serves as condition to count and reorder the degrees of freedom as free, then prescribed, then belonging to the interface. The new global degrees of freedom numbers are stored in the $n_{\text{nodes}} \times 2$ matrix nodeDOFnum, which corresponds to the ID array in [11]. Figure 4.3 draws this matrix's structure. The code to build it can be seen in appendix B.3.1.

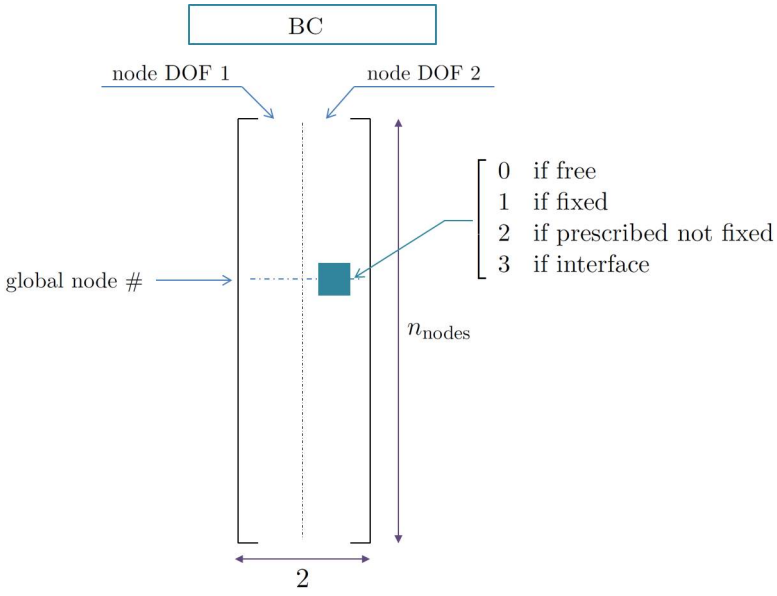

Figure 4.2: Structure of the link matrix

4.3.2 Assembly of matrices

Based on the nodeDOFnum matrix, the $n_{\text{el}} \times 8$ elemDOFnum matrix is built as represented on figure 4.4. It corresponds to the LM array in [11]. Each row of the elemDOFnum matrix corresponds to one element. Each column corresponds to one of these element's DOF. The global

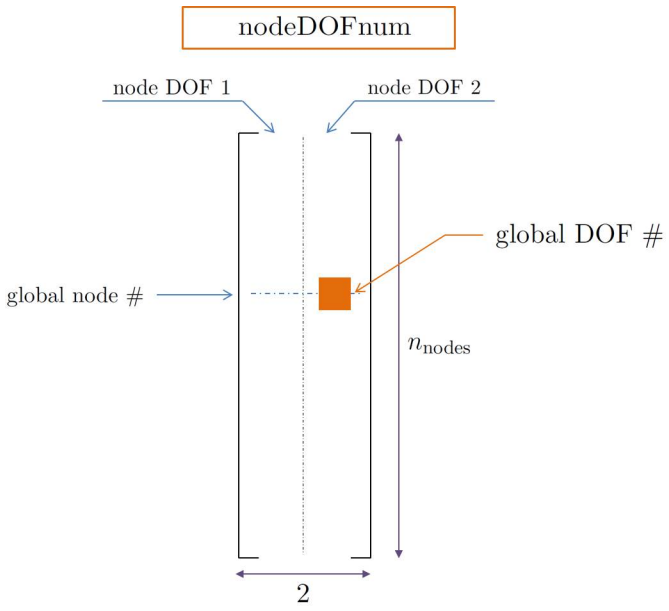

Figure 4.3: Structure of the nodeDOFnum matrix

DOF number is stored at the corresponding location. Corresponding Matlab code can be seen in appendix B.3.2.

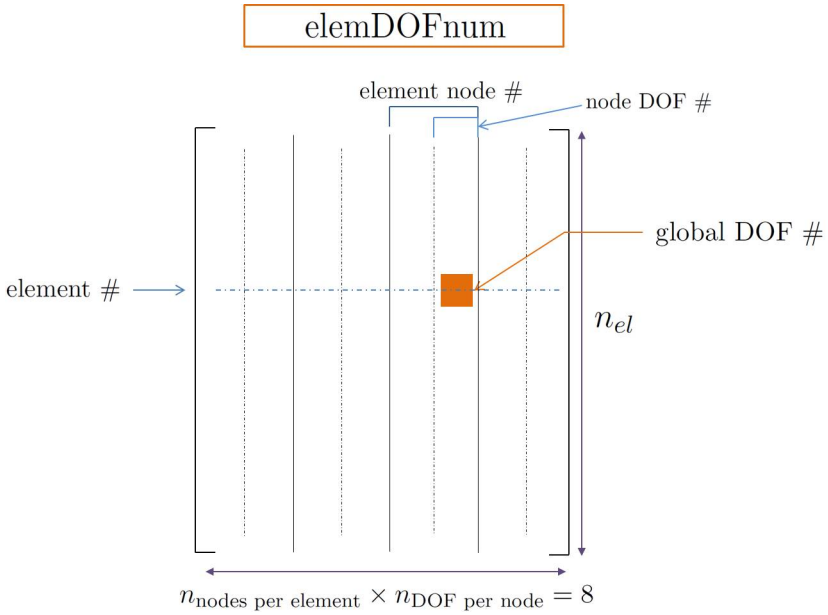

Figure 4.4: Structure of the elemDOFnum matrix

The principle of assembly process is as follows: parts of each element matrices are assigned to their global positions (given by the corresponding DOF number) in the global matrices. This assignment is done inside the element loop. Concretely, at a given element e, each element DOF n is reached by looping over rows and columns of the element matrices. The global DOF number is given by elemDOFnum(e, n).

Global matrices $\mathbb{M}$, $\mathbb{N}$, and $\mathbb{G}$ are respectively size $n_{\text{DOF}} \times n_{\text{DOF}}$, $n_{\text{DOF}} \times n_{\text{DOF}}$, and $n_{\text{DOF}} \times n_{\text{el}}$ (where n_{DOF} is the total number of DOF that are not fixed). But since a DOF belongs to at most four elements, the matrices are sparse. To take advantage of this feature, the assembly process is based on the sparse function in Matlab. Instead of directly assigning the element part to the global part, three arrays are built: one containing the row indices of the global location, one containing the column indices, and one containing the actual values. The sparse function is then called after the loops over all the elements are done.

The code corresponding to the matrix assembly can be seen in appendix B.3.3.

4.4 Discussion on the solution procedure

Once the matrices are assembled, the residuals can be computed. Newton linear equation 3.104, reproduced below, needs to be solved (see chapter 3, section 3.2.3).

$$\underbrace{\begin{bmatrix} \mathbb{M}^{ff} + \mathbb{N}^{ff}\Delta t\,\gamma_v & 0 & -\mathbb{G}^{f} \\ 0 & \left(m + T\mathbb{M}^{II}T^{T}\right) + c\Delta t\,\gamma + k\Delta t^2\beta & -T\mathbb{G}^{I} \\ \mathbb{G}^{f^T}\Delta t\,\gamma_v & \mathbb{G}^{I^T}T^{T}\Delta t\,\gamma & 0 \end{bmatrix}}_{B} \begin{bmatrix} \Delta a \\ \Delta\alpha \\ \Delta p \end{bmatrix} = \begin{bmatrix} -r_1 \\ -r_2 \\ -r_3 \end{bmatrix} \tag{4.13}$$

[22] has a solution process that solves for the pressure and acceleration increments separately. This can be shown as based on the Schur complement of the lower right block 0 of the matrix B. This method is presented first in section 4.4.1. Then, an alternate way to solve the system, simply using Matlab built-in linear solver, is tried (section 4.4.2). Timing for the two solution methods are compared.

4.4.1 Solution procedure based on the Schur complement

To reduce the notations, the matrices $\bar{\mathbb{M}}$ and $\bar{m}^*$ are defined as follows:

$$\bar{\mathbb{M}} = \mathbb{M}^{ff} + \mathbb{N}^{ff}\Delta t\,\gamma_v \tag{4.14}$$

$$\bar{m}^* = \left(m + T\mathbb{M}^{II}T^T\right) + c\Delta t\,\gamma + k\Delta t^2\beta \tag{4.15}$$

Equation 4.13 becomes:

$$\underbrace{\begin{bmatrix} \bar{\mathbb{M}} & 0 & -\mathbb{G}^f \\ 0 & \bar{m}^* & -T\mathbb{G}^I \\ \mathbb{G}^{f^T}\Delta t\,\gamma_v & \mathbb{G}^{I^T}T^T\Delta t\,\gamma & 0 \end{bmatrix}}_{B} \begin{bmatrix} \Delta a \\ \Delta\alpha \\ \Delta p \end{bmatrix} = \begin{bmatrix} -r_1 \\ -r_2 \\ -r_3 \end{bmatrix} \tag{4.16}$$

The Schur complement of the lower right block 0 is then:

$$\mathbf{K} = 0 - \begin{bmatrix} \mathbb{G}^{f^T}\Delta t\,\gamma_v & \mathbb{G}^{I^T}T^T\Delta t\,\gamma \end{bmatrix} \begin{bmatrix} \bar{\mathbb{M}} & 0 \\ 0 & \bar{m}^* \end{bmatrix} \begin{bmatrix} -\mathbb{G}^f \\ -T\mathbb{G}^I \end{bmatrix} \tag{4.17}$$

$$\mathbf{K} = \Delta t\,\gamma_v\mathbb{G}^{f^T}\bar{\mathbb{M}}^{-1}\mathbb{G}^f + \Delta t\,\gamma\mathbb{G}^{I^T}T^T\bar{m}^{*-1}T\mathbb{G}^I \tag{4.18}$$

Steps to solve equation 4.16 are then as follows:

(1) Solve for Δp

$$\mathbf{K}\Delta p = -r_3 + \Delta t\,\gamma_v\mathbb{G}^{f^T}\bar{\mathbb{M}}^{-1}r_1 + \Delta t\,\gamma\mathbb{G}^{I^T}T^T\bar{m}^{*-1}r_2 \tag{4.19}$$

(2) Solve for Δa and $\Delta\alpha$

$$\bar{\mathbb{M}}\Delta a = -r_1 + \mathbb{G}^f\Delta p \tag{4.20}$$

$$\bar{m}^*\Delta\alpha = -r_2 + T\mathbb{G}^I\Delta p \tag{4.21}$$

This solution procedure implemented in Matlab is reproduced in appendix B.4.1 and took $0.0249\,s.$ to complete.

4.4.2 Procedure based on the direct factorization of B

B is a bigger matrix than $\bar{\mathbb{M}}$, $\bar{m}^*$ or $\mathbf{K}$ but its inversion is needed only once. Using the Matlab command \, this solution method took $0.0186\,s.$ For the size of problems that are solved in this thesis (see chapter 5), the direct linear solve seems to be faster.[4] Thus, this simpler solution procedure is kept in the final version of the code. See appendix B.4.2 for Matlab implementation details.

4.5 Update of the dependent variables

The dependent variables that form the solution are $\bar{p}$, $\bar{a}$, $\bar{v}$, α, ν and δ. $\bar{a}$ and $\bar{v}$ are split into three parts. Only the free part needs to be solved for, and the prescribed part does not change throughout Newton iterations. Some of the solid degrees of freedom can be prescribed, in which case they do not need to be solved for. To efficiently update the variables, it is interesting to think of which parts are necessary to compute the residuals and which parts change during Newton iterations. These need to be updated in each iteration, while the remaining parts can be updated at each time step only.

A first look at the residual equations 3.89 to 3.91 shows that the free and prescribed parts of the fluid acceleration $\bar{a}^f$ and $\bar{a}^p$, the prescribed part of the fluid velocity $\bar{v}^f$, and the complete solid acceleration α are needed, as well as the pressure vector $\bar{p}$. But it should not be forgotten that the complete fluid velocity is necessary to compute the convective velocity $\bar{c}$, intervening in the matrix $\mathbb{N}$. To compute the interface part of $\bar{v}$, complete solid velocity is needed. Furhtermore, to update the nodal coordinates, complete solid displacement is needed. Table 4.1 summarizes the

[4] Note that a procedure based on the Schur complement my be chosen for other reasons, such as less numerical error propagation.

parts of the dependent variables needed to build the residual equations and the parts that need to be updated at each Newton iteration.

Dependent variables parts	needed for the residuals	needs to be updated at each iteration
$\bar{p}$	yes	yes
$\bar{a}^f$	yes	yes
$\bar{a}^p$	yes	no
$\bar{a}^I$	no	no
$\bar{v}^f$	yes	yes
$\bar{a}^p$	yes	yes
$\bar{a}^I$	yes	yes
α	yes, entirely	yes for the free part
ν	yes, entirely	yes for the free part
δ	yes, entirely	yes for the free part

Table 4.1: Summary of the necessary updates

At each time step, the new solution is stored in column $n+1$ of the corresponding array. To avoid repeated access to this column, a temporary vector denoted by the subscript "new" is defined for each dependent variable. Since only the free part of $\bar{a}$ needs to be updated throughout the iterations, $a_{\text{new}} = \bar{a}^f_{n+1}$. The structure of the updates can be seen in the appendix B.5, reproducing a whole time step of the code.

Summary of the implementation chapter

This chapter presented selected parts of the Matlab code written to solve a fluid-rigid body interaction problem, which has been summarized in introduction. The purpose was to lead the reader through interesting implementation issues, as well as to make a potential extension of the work easier. Mesh motions's choice and implementation, introduced first, are the main features of the ALE description. They are specific to the kind of problems this code is applied to, presented in chapter 5. For different kinds of mesh motion, the reader is invited to refer to [9]. Conceptual aspects of different mesh update methods are presented, without practical applications though. Building element matrices required some attention for computational efficiency. Resulting code

might not have been intuitive for the reader without the explanations of section 4.2. The assembly part explained how these element matrices are assembled, which is classical for any finite element method. Then, the solution procedure and the update of the dependent variables were discussed, taking efficiency into account.

The next chapter presents the numerical examples used to test the code described above. Once trustworthy results were obtained, a model problem of article [22] was implemented. The significant number of elements needed for this problem led to the computational efficiency concerns mentioned in this chapter.

Chapter 5

Numerical examples

Introduction

The previous chapter derived the complete discrete formulation and solution for the fluid-rigid body interaction, as well as its detailed implementation. This chapter shows the code application through two numerical examples.

The first example, presented in 5.1, is a fluid-only problem and simulates the steady-state Couette flow. One of the boundaries is registered as solid to cast this problem into the fluid-rigid body interaction framework. Due to the way the code is set up (see chapter 4), having a rigid body is necessary. This simple problem has the advantage of having a well-known analytical solution. Results given by the code can be easily checked. When these results are convincing enough, the second problem can be modeled as the rigid cylinder freely oscillating in a fluid presented by [22]. This problem is simulated in section 5.2. This section first details the geometry and parameters necessary to model the problem and summarizes its implementation as input for the previously presented code. Results obtained by the code are then compared to the ones presented in [22].

5.1 A model test problem: Simulation of 1D Couette flow

After a review of this problem and its analytical solution (paragraph 5.1.1), paragraph 5.1.2 presents its implementation as a test problem for the code built here. The main advantage of the 1D Couette flow problem is the existence of an analytical solution, to which answers given by the

code can be compared. The problem is implemented with only 4 elements, to keep the numbers of degrees of freedom very low. These two features ease the tracking of implementation errors while testing the solution.

5.1.1 Description of the problem and analytical solution

The Couette flow is driven by a prescribed velocity $v_x = U$ at $y = H$ (see figure 5.1). Prescribing the velocity at the top while the bottom is kept fixed is equivalent to applying a shear stress. There is no pressure gradient in the x direction.

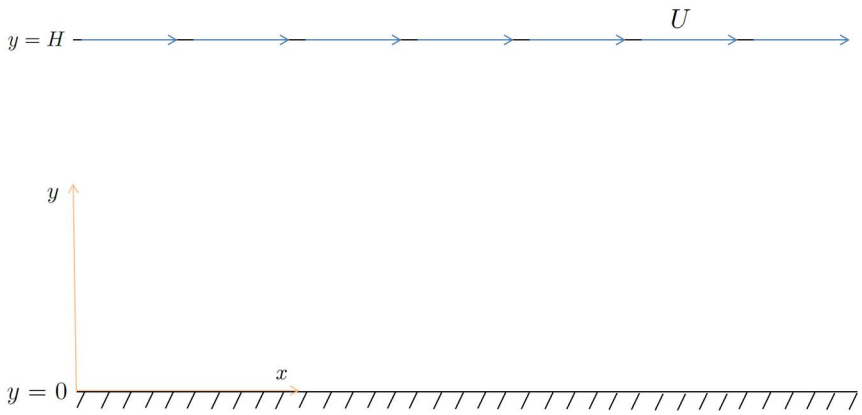

Figure 5.1: Geometry and boundary conditions of the Couette flow problem

5.1.1.1 Assumptions

- Newtonian fluid (constitutive relationship $\sigma = -p\mathbf{I} + 2\mu\frac{1}{2}\left(\mathbf{grad}(v) + \mathbf{grad}(v)^T\right)$, where σ is the Cauchy stress, p the pressure and v the velocity)
- Incompressible
- Steady state
- 1D flow

5.1.1.2 Simplified Navier-stokes equation

Taking the assumptions 5.1.1.1 into account, the Navier-Stokes equations are simplified as follows:

$$\frac{\partial v_x}{\partial x} = 0 \tag{5.1}$$

$$\frac{\partial p}{\partial x} = \mu \frac{\partial^2 v_x}{\partial y^2} = 0 \tag{5.2}$$

$$\frac{\partial p}{\partial y} = \rho g \tag{5.3}$$

5.1.1.3 Solution

Equation 5.3 shows that the vertical pressure gradient just balances gravity. In our case, all body forces are neglected. Equation 5.3 becomes:

$$\frac{\partial p}{\partial y} = 0 \tag{5.4}$$

Integrating equation 5.2 twice and using the boundary conditions $v_x(y = 0) = 0$ and $v_x(y = H) = U$ leads to:

$$v_x = \frac{U}{H}\, y \tag{5.5}$$

Figure 5.2 shows the resulting velocity profile.

5.1.2 Implementation and numerical results

The goal is to model the simplest possible problem to test the code. So, only four elements are used to discretize the fluid domain. Two cases are studied. The simplest case is to set the rigid body to the fixed boundary, shown in figure 5.3. None of the solid DOFs are moving. Thus the mesh stays fixed too. In the second studied case, the rigid body is set to the mobile upper boundary (figure 5.5). This enables the simulation of a very simple problem with a nonzero mesh motion.

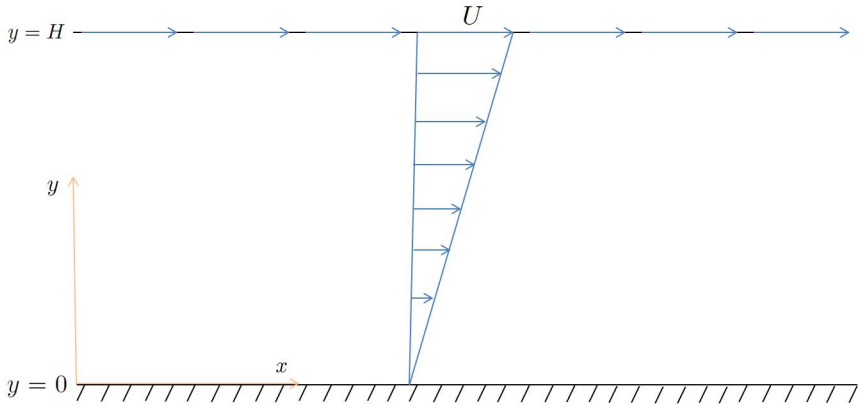

Figure 5.2: Velocity profile for the pure shear driven flow

5.1.2.1 First model problem: solid as the fixed bottom boundary

The first problem set the fixed bottom boundary as a rigid body. All the solid motions are prescribed as 0. The fluid has only seven free degrees of freedom. Figure 5.3 shows the geometry of the problem, with the interface nodes and prescribed velocities. Prescribed force vector is 0 for each free degree of freedom.

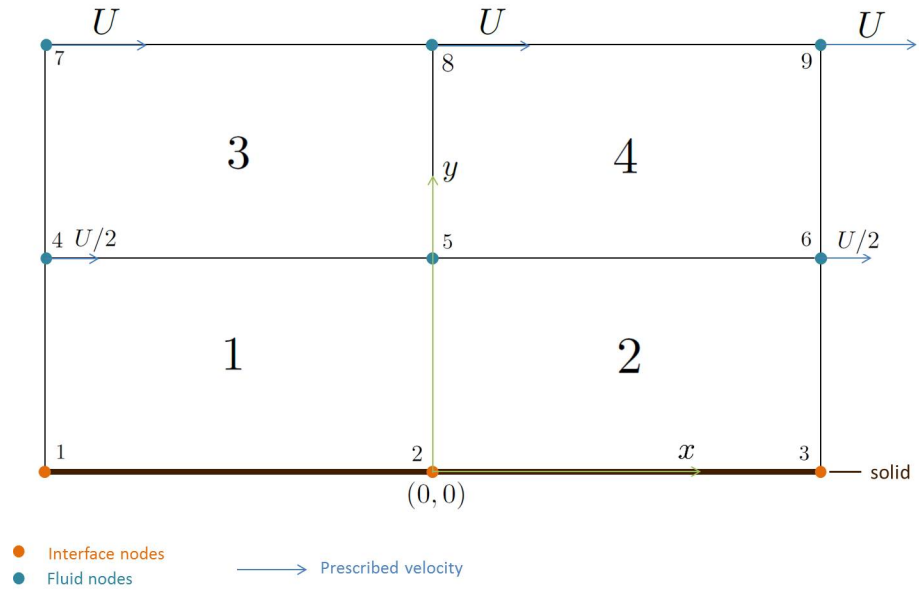

Figure 5.3: Representation of the finite element model for the Couette flow problem with the solid as fixed boundary

Since the effect of gravity is neglected, the pressure must be initially defined as constant within the domain. With the elements and nodes ordered as denoted on the figure 5.3, the initial conditions are as follows:

$$\bar{a} = 0 \qquad (18 \times 1 \text{ vector}) \tag{5.6}$$

$$\bar{v} = \bar{v} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad U/2 \quad 0 \quad U/2 \quad 0 \quad U/2 \quad 0 \quad U \quad 0 \quad U \quad 0 \quad U \quad 0]^T \tag{5.7}$$

$$\bar{p} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.8}$$

$$\alpha = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.9}$$

$$\nu = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.10}$$

$$\delta = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T . \tag{5.11}$$

Since it is theoretically a steady state problem, the solution is expected to stay as the initial condition. With the fluid acceleration and velocity vectors reordered as free degrees of freedom first, then prescribed and then belonging to the interface, the expected solution is:

$$\bar{a} = 0 \qquad (18 \times 1 \text{ vector}) \tag{5.12}$$

$$\bar{v} = [U/2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad U/2 \quad U/2 \quad U \quad U \quad U \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T \tag{5.13}$$

$$\bar{p} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.14}$$

$$\alpha = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.15}$$

$$\nu = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.16}$$

$$\delta = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.17}$$

Setting $U = 0.5\, m/s$, the solution given by the Matlab code is shown in figure 5.4. For this very simple problem, the numerical results match the theory.

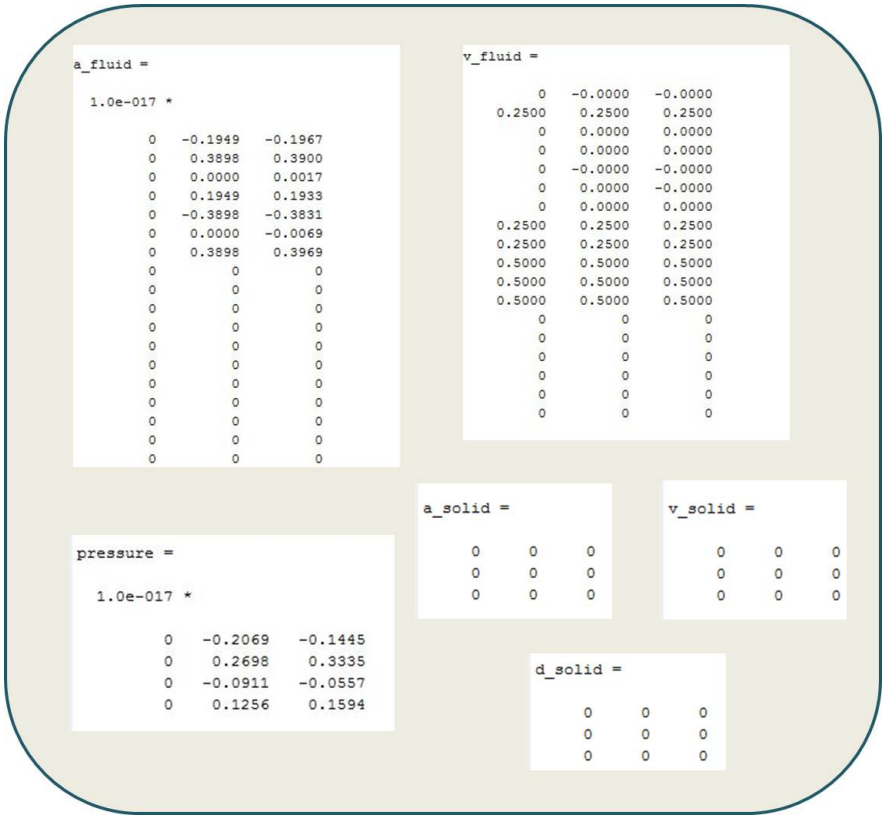

Figure 5.4: Matlab results for the Couette flow problem with the solid as fixed boundary

Note that if the pressure initial condition is not zero, the prescribed force vector can not be zero. Force boundary conditions need to be applied to model the effect of the surrounding fluid, because the domain has been artificially cut in the x direction. See code in appendix C.1 for more precision on how this is applied.

5.1.2.2 Second model problem: solid as the mobile upper boundary

The first problem set the moving top boundary as the rigid body. Solid velocity ν is thus prescribed as $\nu = \begin{bmatrix} U & 0 & 0 \end{bmatrix}$. At time t_n, solid displacement is prescribed as $\delta = \begin{bmatrix} (U\, n\Delta t) & 0 & 0 \end{bmatrix}^T$. Bottom fluid degrees of freedom are fixed in the x direction and left free in the y direction[1] . The fluid has again 7 free degrees of freedom. Figure 5.3 shows the geometry of the problem, with the

[1] If all the vertical bottom degrees of freedom are fixed for the fluid, we run into numerical issue because there are not enough conditions to determine the pressure.

interface nodes and prescribed velocities. Prescribed force vector 0 for each free degree of freedom. Code in appendix C.1 shows how to implement this problem as an input for the main code.

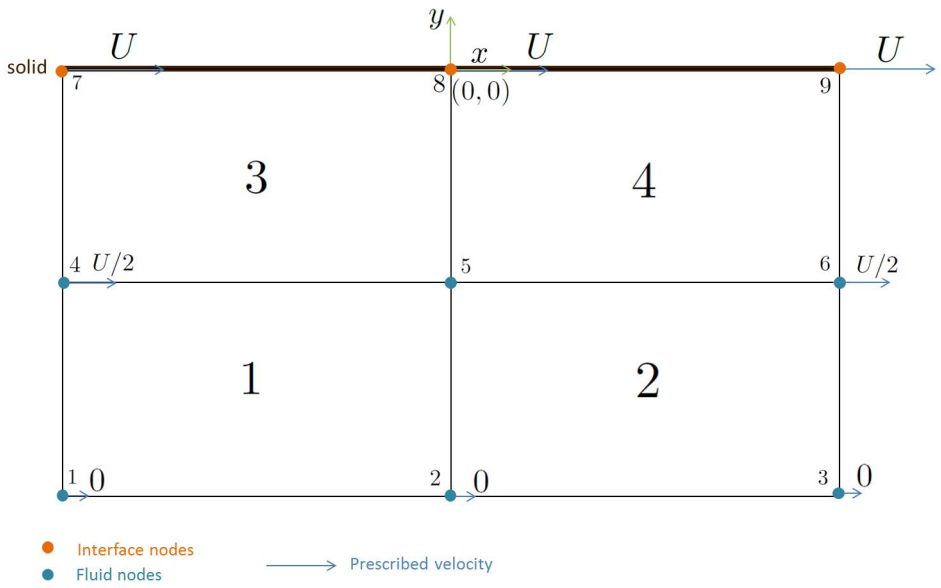

Figure 5.5: Representation of the finite element model for the Couette flow problem with the solid set as the mobile boundary

Since the number and order of elements and nodes has not changed compared to the previous problem, the fluid initial conditions are the same as equations 5.6 to 5.14. Solid initial conditions are:

$$\alpha = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.18}$$

$$\nu = \begin{bmatrix} U & 0 & 0 \end{bmatrix}^T \tag{5.19}$$

$$\delta = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T. \tag{5.20}$$

Again, the solution is expected to stay fixed as the initial condition. Since the three bottom x degrees of freedom are fixed, they are not solved for. Thus, the fluid acceleration and velocity

reordered solution vectors are size 15×1. The expected solution is:

$$\bar{a} = 0 \qquad (15 \times 1 \text{ vector}) \tag{5.21}$$

$$\bar{v} = [0 \quad 0 \quad 0 \quad 0 \quad U/2 \quad 0 \quad 0 \quad U/2 \quad U/2 \quad U \quad 0 \quad U \quad 0 \quad U \quad 0]^T \tag{5.22}$$

$$\bar{p} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.23}$$

$$\alpha = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{5.24}$$

$$\nu = \begin{bmatrix} U & 0 & 0 \end{bmatrix}^T \tag{5.25}$$

$$\delta = \begin{bmatrix} 0 & U\Delta t & 2U\Delta t \end{bmatrix}^T \tag{5.26}$$

Setting $U = 0.3\,m/s$, and $\Delta t = 0.5\,s$, the solution given by the Matlab code is shown in figure 5.6. For this less simple problem (complicated by the moving mesh), the numerical results once again match the theory. The deformed mesh is shown in figure 5.7. Nodes are moving as they are expected to, as explained in chapter 4, section 4.1.

Note, if the pressure initial condition is not zero, nonzero force boundary conditions need to be applied to model the effect of the surrounding fluid. The problem is that the artificially created side boundaries are tilting due to the mesh motion (see figure 5.7). Pressure applied by the surrounding fluid acts in the normal direction to the boundary. Thus, the applied force needs to change during the iterations.

5.1.3 Conclusion on the Couette flow problem simulation

Results of the simulations detailed in this section match the theoretical results. Some specific issues to the Couette flow problem arose, such as the expression of the force boundary conditions on the artificially created fluid side boundaries. When the fluid is contained in a finite domain like it is the case for the problem presented in section 5.2, this issue does not occur. The Couette flow problem was convenient to test the code, thanks to the existence of an analytical solution and the

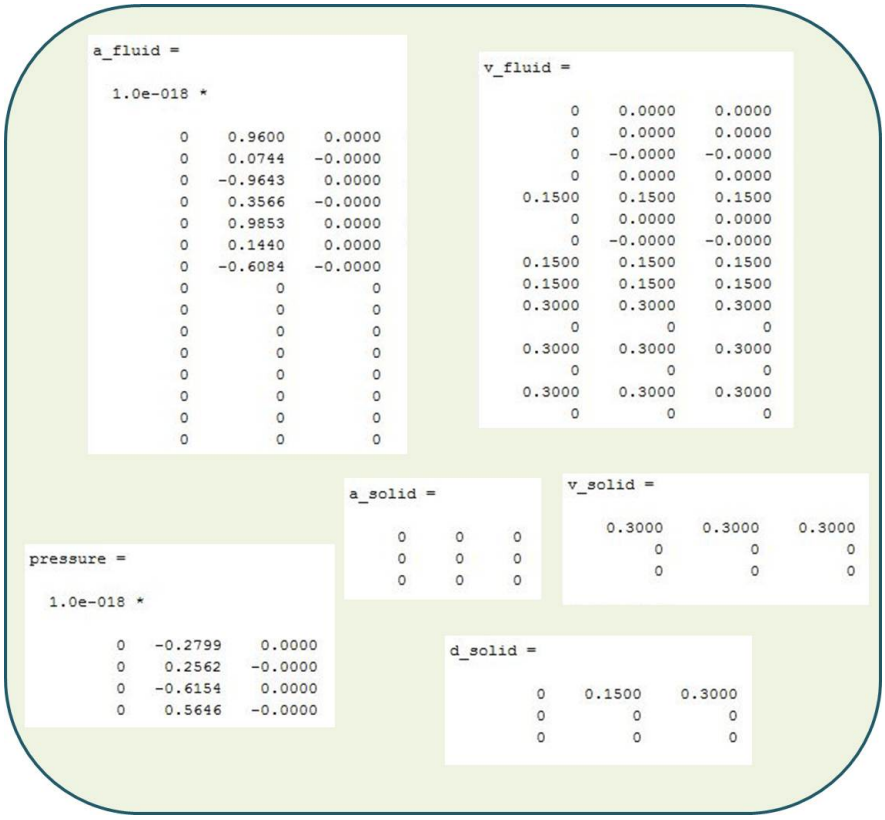

Figure 5.6: Matlab results for the Couette flow problem with the solid set as the mobile boundary

possibility to set up a model with very few degrees of freedom. But the ALE method is clearly not the best to address this pure flow problem. If the goal was to study the Couette flow itself, an Eulerian approach would have been much more suitable.

Results obtained in this section are convincing enough to enable us to step forward in the implementation by modeling our target problem of the oscillating cylinder. Compared to the Couette flow problems, the cylinder problem adds the complications of a free solid degree of freedom and a transient behavior.

5.2 Free oscillations of a rigid cylinder in a cylindrical fluid domain

The numerical example presented here is a rigid cylinder oscillating in a fluid, introduced by [22]. Maintained by a spring attached to the outer boundary, the cylinder is initially perturbed from its

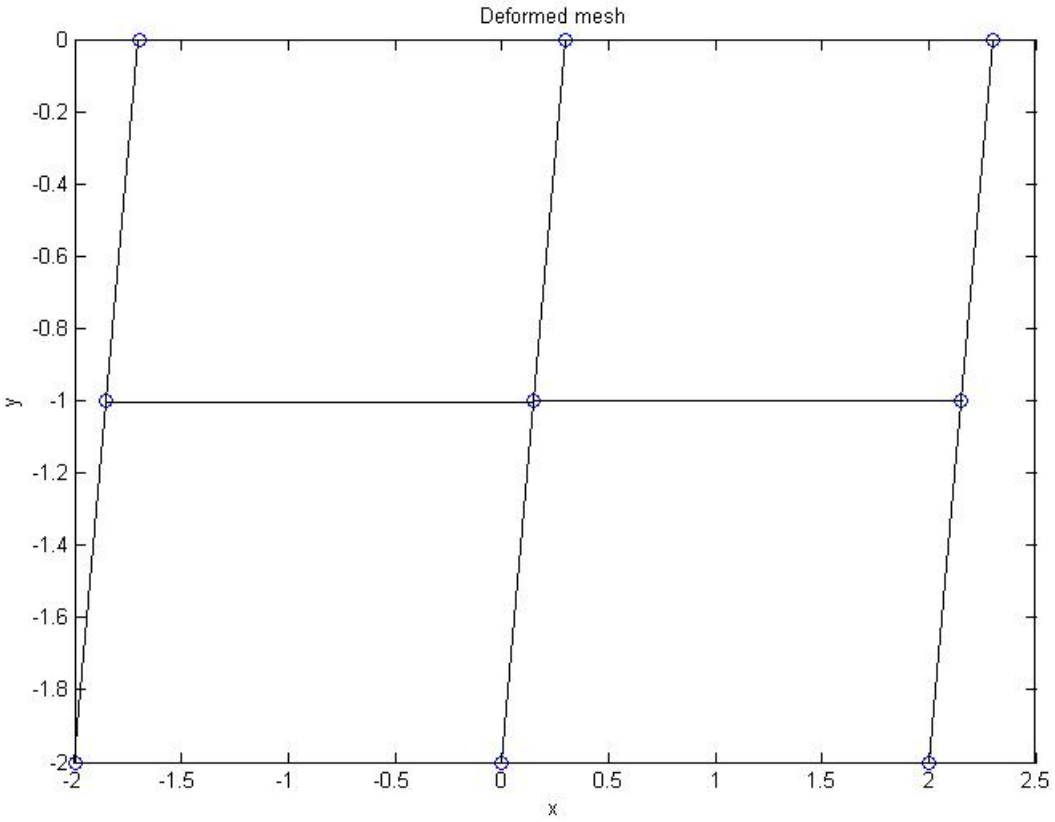

Figure 5.7: Deformed ALE mesh for the Couette flow problem

equilibrium position. It then freely oscillates in its fluid environment. Paragraph 5.2.1 presents the geometry and the parameters necessary to carry the study. The implementation of the problem is explained next in paragraph 5.2.2, with focus on the mesh generation. Paragraph 5.2.3 presents the results of the simulation.

5.2.1 Presentation of the problem

Figure 5.8 presents the geometry and notations for the studied problem. A circular domain of diameter D, with an impermeable fixed rigid boundary, is filled with a viscous fluid of density ρ and viscosity μ. A rigid circular cylinder of mass m_{11} is attached to the outer boundary by an elastic spring of stiffness k_{11}. At time $t = 0$, the cylinder's position is perturbed by the displacement

$\delta_1 = a$. The cylinder oscillates then freely in the x-direction, assuming that the spring does not tilt during motion.

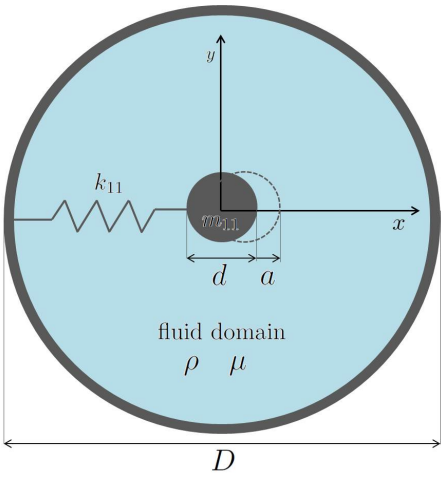

Figure 5.8: Geometry of the cylinder problem

The values of the parameters needed for the simulation are given in tables 5.1 to 5.3. They are taken form [22] and converted to the international unit system. They are specified in the input for the code (see appendix C.2).

symbol	d	D	a
value	$1.27 \times 10^{-2}\, m$	$5\, d$	$d/100$

Table 5.1: Geometrical parameters for the oscillating cylinder problem

matrix	m		c		k	
value	$\begin{bmatrix} 3.408 \times 10^{-3} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	kg	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 34.6113 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	N/m

Table 5.2: Solid parameters for the oscillating cylinder problem

symbol	ρ	μ
water	$1000\,kg/m^3$	$1.33 \times 10^{-3}\,Pas$
silicon oil	$936\,kg/m^3$	$0.145\,Pas$

Table 5.3: Fluid parameters for the oscillating cylinder problem

5.2.2 Mesh generation

The input file for the cylinder problem, in appendix C.2, is very similar to the one needed for the Couette flow problem. But generating the mesh for the cylinder is more challenging. Thus it is interesting to give some more details on this operation.

An example of coarse mesh with $n_{\text{arc}} = 4$ arcs and $n_{\text{rad}} = 8$ radii is shown in figure 5.9. Nodes and elements are numbered according to a spiral from the node adjacent to the solid on the right. The mesh is generated by advancing front. More precisely, the code loops over the arcs and radii as follows:

(1) Loop over the arcs

   (a) Get the arc radius R. To have a finer mesh closer to the solid, arcs are logarithmically distributed over the total width $(D - d)/2$.

   (b) Loop over the radii

      (i) Get the radius angle

      (ii) Compute the coordinates x^0 and y^0 of the corresponding node

      (iii) Assign corresponding parts of the link matrix and connectivity matrix

(2) Compute the boundary condition matrix: the inner arc is the interface, the outer arc is fixed, and all other nodes are free.

Note that the problem is symmetric with respect to the x-axis. Thus to save some computational expense, it would be good to model only half of the problem (for example the top half, as is done

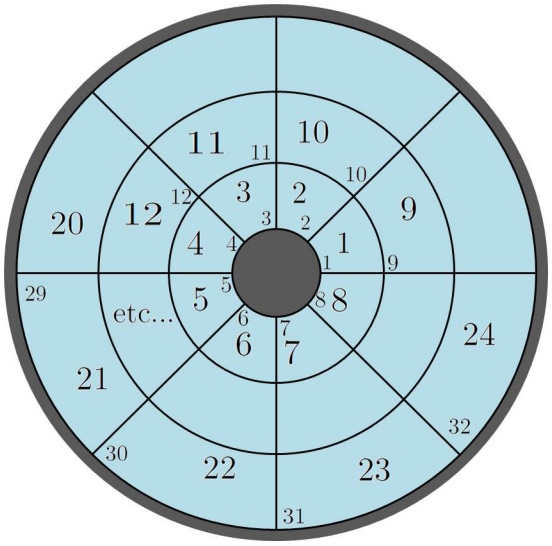

Figure 5.9: Coarse mesh example for the cylinder problem

in [22]). This requires to apply symmetry conditions on the bottom line, operation that is not straightforward. That is why the full problem is modelled here.

5.2.3 Results

5.2.3.1 Settings

To simulate the cylinder problem, $narc = 10$ and $n_{\text{rad}} = 48$ are used, so that we get the plot of the initial mesh in figure 5.10. Two fluids are tested : water and silicon oil. The time increment is set as $\Delta t = 10^{-3}$ for the water simulation and as $\Delta t = 5 \times 10^{-5}$ for the silicon oil simulation (specified by [22]). A tolerance of 10^{-12} is used for Newton's method.

5.2.3.2 Rigid body displacement

Rigid body displacement $\delta_1$in the x-direction is shown in figure 5.11 in the case of water and in figure 5.12 in the case of silicon oil.

Comparing figures 5.11 and 5.12, we can see that damping is higher in the case of silicon oil, which is more viscous than water (see table 5.3 for the values). Qualitatively, the results make

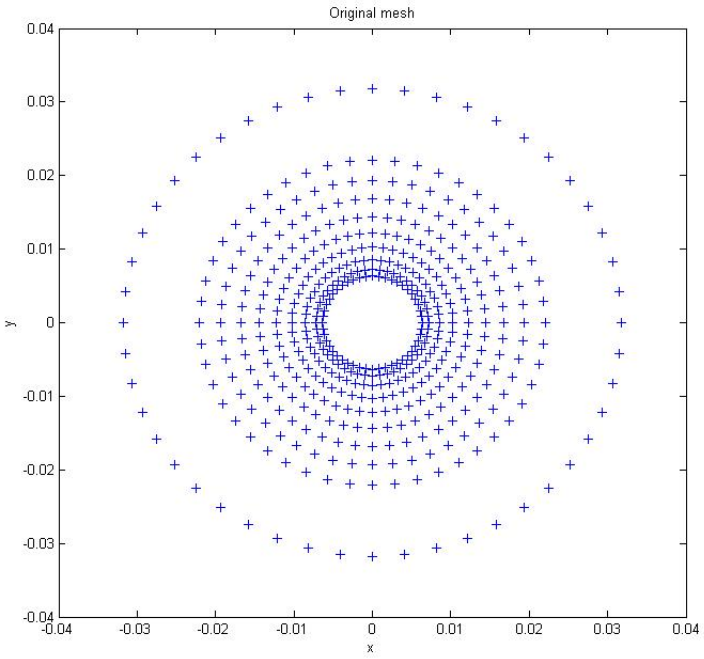

Figure 5.10: Matlab mesh at $t = 0$ for the simulated cylinder problem

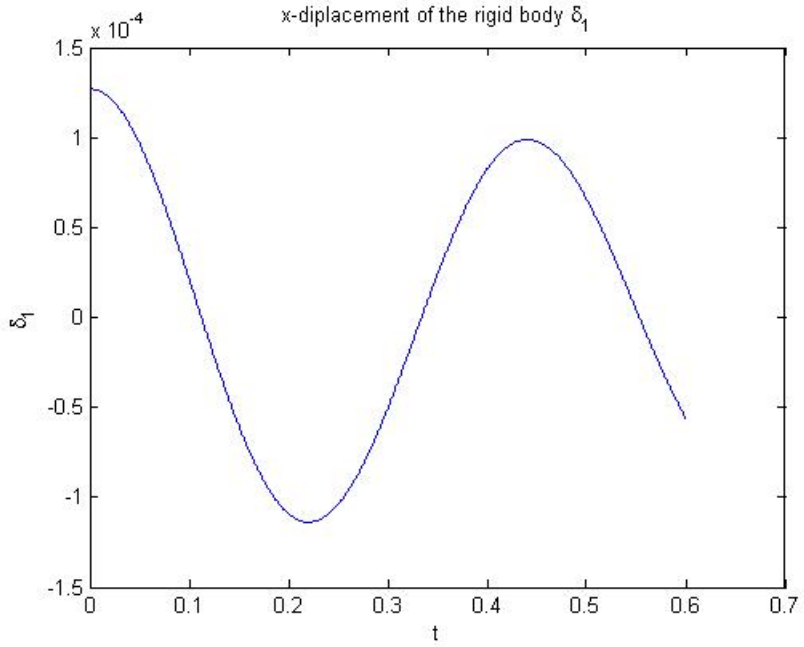

Figure 5.11: Resulting rigid body displacement δ_1 in the case of water

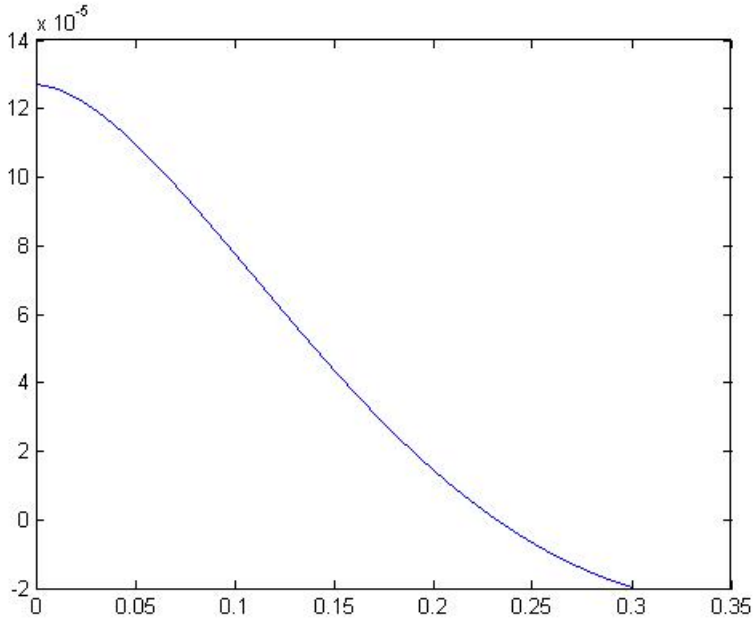

Figure 5.12: Resulting rigid body displacement δ_1 in the case of silicon oil

good intuitive sense.

Comparing the rigid-body displacement with the results of [22], figures 5.11 and 5.12 show more damping and a lower oscillations frequency. A possible explanation for this difference is the need of a refined mesh. Two clues are in favor of this explanation. When trying to model the same problem, but with a coarser mesh (namely 8 elements), the resulting δ_1 is further away from [22], that is to say that the frequency and damping are higher. The other reason is the difference in formulations between [22] and our problem. [22] uses a Streamline Upwind Petrov Galerkin formulation (SUPG), presented in [6], for the finite element discretization. The formualtion used in this thesis is a simple Galerkin formulation (see chapter 3, section 3.1.1). according to [6], spatial instabilities occur when using a Galerkin formulation for convection-dominated problems. One way to get around this issue is to severely refine the mesh. To avoid refining the mesh for computational expense reasons, the SUPG formulation has been developed. Our results might match the ones in [22] better if we heavily refine the mesh.

5.2.4 Concluding remarks on the oscillating cylinder problem

We have been able to model and implement the problem of the rigid cylinder oscillating in a fluid, introduced in [22]. This was one of the present work's objective. Results in terms of rigid body displacement make good intuitive sense. To further interpret the results, it would be interesting to compute the resultant force exerted by the fluid on the solid and to quantify the net effects in terms of added mass and added damping. Plotting and interpreting the pressure and velocity fields is still to be done, and would probably require another tool than Matlab.

To get closer results to [22], for the rigid body displacement δ_1, a severe mesh refinement or the use of the SUPG formulation are probably necessary.

5.3 Concluding remarks on the numerical examples

Two numerical examples have been presented here. First, the pure fluid Couette flow problem has been simulated. Its simplicity and the existence of an analytical solution were the useful ingredients that made us choose the Couette flow as a first test problem. It has been adapted to the fluid-rigid body interaction framework by assigning the rigid body to one of the boundaries. Depending on whether the solid where assigned to the fixed or the moving boundary, the ALE mesh was fixed or moving. This example has been helpful to confirm the code results by comparison with the theoretical solution. As an extension of the Couette problem implementation, Stokes' first and second problem could be modelled with the ALE code.

The second model problem was the freely oscillating cylinder, introduced in [22]. This example added to the first test problem a more complicated geometry (hence a less straightforward mesh generation), a free solid degree of freedom and an actual transient behavior. Results providing by our code were satisfying enough to be convinced that the method worked. Nevertheless, further refining the mesh, quantifying the added mass and added damping effects, and plotting the pressure and velocity fields would be interesting extensions of the work, to get further confirmation. Finally,

an SUPG formulation would be interesting to implement in the code that has been built in this thesis. Results of the code should then completely match the ones in [22].

Chapter 6

Concluding remarks

Along this thesis, we have built an understanding of the Arbitrary Lagrangian Eulerian method, concept and application. First, the motivation and a precise description of the ALE method have been presented. Clear and rigorous notations have been set up to derive ALE governing equations.

One of the work's goal was to develop a simple implementation. The application to fluid-rigid body interaction has been adopted. Within this context, ALE governing equations for the fluid have been derived. Fluid governing equations have been discretized in space using the Finite Element method. The derivation of their weak form using the method of weighted residuals, as well as the discrete approximation of the equations using the Galerkin formulation have been developed in detail. Spatially discrete fluid equations have been coupled with the solid motion governing equations through the interface coupling conditions. By discretizing the resulting set of equations in time, we have obtained the final set of discrete nonlinear equations governing the system. The solution method to solve them has been derived next, based on an approximated Newton's method.

Interesting implementation parts have been pointed out to support the reader's understanding and ease potential extension of the work. In particular, we have described the choice of a mesh motion, on which the ALE method is based. Two concrete examples have been implemented. The first one, Couette flow problem, has been chosen for its simplicity and the existence of an anlytical solution. With a simple 4-element model, good matching results have been obtained. Thus a more sophisticated problem has been simulated. The free oscillations of a rigid cylinder in a fluid medium

have been presented by [22]. Results of [22] have not been exactly reproduced here, but our code has obtained decent results. This simple fluid-rigid body interaction example should then be able to serve as basis for future work.

Fluid-structure interaction was the first motivation for the development of the ALE method. Fluid-rigid body interaction, application chosen for this work, is interesting as an idealization of fluid-structure interaction, when the solid deformations are small compared to its rigid-body motion. This is the case for offshore structure in the ocean for example. Considering the solid as a rigid body leads to a simpler problem, because only the fluid governing equations need to be expressed in the ALE framework and discretized in space using finite elements. But the ALE method has numerous other applications, as has been stated in introduction. If we further want to model forming processes, rolling contact, crack propagation, or pile driving using the ALE method, the framework developed in this thesis should make the work easier. Obviously, the governing equations of the problem would be different. But their transformation to the ALE representation would follow the one described in this thesis. Finite element discretization and associated implementation are not specific to the ALE method. The concept of the mesh motion would stay the same, but the practical choice of a mesh motion might need to be adapted to the requirements of the new problem.

Bibliography

[1] COMSOL Multiphysics website.

[2] 2010.

[3] H. Askes, A. Rodrguez-Ferran, and A. Huerta. Adaptive analysis of yield line patterns in plates with the arbitrary lagrangianeulerian method. Computers & Structures, 70(3):257 – 271, 1999.

[4] Harm Askes and Lambertus J. Sluys. Remeshing strategies for adaptive ale analysis of strain localisation. European Journal of Mechanics - A/Solids, 19(3):447 – 467, 2000.

[5] Ted Belytschko, W.K. Liu, and B. Moran. Nonlinear finite elements for continua and structures. Wiley, 2000.

[6] Alexander N. Brooks and Thomas J.R. Hughes. 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(13):199 – 259, 1982.

[7] W. Dettmer and D. Peri. A computational framework for fluidstructure interaction: Finite element formulation and applications. Computer Methods in Applied Mechanics and Engineering, 195(4143):5754 – 5779, 2006. ¡ce:title¿John H. Argyris Memorial Issue. Part II¡/ce:title¿.

[8] Olivier Doar and Sbastien Michelin. Piezoelectric coupling in energy-harvesting fluttering flexible plates: linear stability analysis and conversion efficiency. Journal of Fluids and Structures, 27(8):1357 – 1375, 2011.

[9] Jean Donea, Antonio Huerta, J.-Ph. Ponthot, and A. Rodrguez-Ferran. Arbitrary LagrangianEulerian Methods. John Wiley & Sons, Ltd, 2004.

[10] Somnath Ghosh and Noboru Kikuchi. An arbitrary lagrangian-eulerian finite element method for large deformation analysis of elastic-viscoplastic solids. Computer Methods in Applied Mechanics and Engineering, 86(2):127 – 188, 1991.

[11] T.J.R. Hughes. The finite element method: linear static and dynamic finite element analysis. Dover Civil and Mechanical Engineering Series. Dover Publications, 2000.

[12] J. Hutink, P. T. Vreede, and J. van der Lugt. Progress in mixed eulerian-lagrangian finite element simulation of forming processes. International Journal for Numerical Methods in Engineering, 30(8):1441–1457, 1990.

[13] U. Nackenhorst. The ale-formulation of bodies in rolling contact: Theoretical foundations and finite element approach. Computer Methods in Applied Mechanics and Engineering, 193(3941):4299 – 4322, 2004. ¡ce:title¿The Arbitrary Lagrangian-Eulerian Formulation¡/ce:title¿.

[14] Majidreza Nazem, Daichao Sheng, and John P. Carter. Stress integration and mesh refinement for large deformation in geomechanics. International Journal for Numerical Methods in Engineering, 65(7).

[15] Th. Richter and Th. Wick. Finite elements for fluidstructure interaction in ale and fully eulerian coordinates. Computer Methods in Applied Mechanics and Engineering, 199(4144):2633 – 2642, 2010.

[16] Antonio Rodrguez-Ferran, Agust Prez-Foguet, and Antonio Huerta. Arbitrary lagrangianeulerian (ale) formulation for hyperelastoplasticity. International Journal for Numerical Methods in Engineering, 53(8):1831–1851, 2002.

[17] Josep Sarrate, Antonio Huerta, and Jean Donea. Arbitrary lagrangianeulerian formulation for fluidrigid body interaction. Computer Methods in Applied Mechanics and Engineering, 190(2425):3171 – 3188, 2001. ¡ce:title¿Advances in Computational Methods for Fluid-Structure Interaction¡/ce:title¿.

[18] P.J.G. Schreurs, F.E. Veldpaus, and W.A.M. Brekelmans. Simulation of forming processes, using the arbitrary eulerian-lagrangian formulation. Computer Methods in Applied Mechanics and Engineering, 58(1):19 – 36, 1986.

[19] Daichao Sheng. Frictional contact for pile installation. In Peter Wriggers and Udo Nackenhorst, editors, IUTAM Symposium on Computational Methods in Contact Mechanics, volume 3 of IUTAM Bookseries (closed), pages 239–255. Springer Netherlands.

[20] Daichao Sheng, Majidreza Nazem, and John Carter. Some computational aspects for solving deep penetration problems in geomechanics. Computational Mechanics, 44:549–561, 2009. 10.1007/s00466-009-0391-6.

[21] Nomura Takashi. Ale finite element computations of fluid-structure interaction problems. Computer Methods in Applied Mechanics and Engineering, 112(14):291 – 308, 1994.

[22] Nomura Takashi and Thomas J.R. Hughes. An arbitrary lagrangian-eulerian finite element method for interaction of fluid and a rigid body. Computer Methods in Applied Mechanics and Engineering, 95(1):115 – 138, 1992.

Appendix A

Expression of ψ_χ spatial derivatives

$$\frac{\partial \psi_{\chi 1}}{\partial \xi_1} = \sum_{a=1}^{4} \frac{\partial N_a}{\partial \xi_1} \chi_{1a} \tag{A.1}$$

$$\frac{\partial \psi_{\chi 2}}{\partial \xi_1} = \sum_{a=1}^{4} \frac{\partial N_a}{\partial \xi_1} \chi_{2a} \tag{A.2}$$

$$\frac{\partial \psi_{\chi 1}}{\partial \xi_2} = \sum_{a=1}^{4} \frac{\partial N_a}{\partial \xi_2} \chi_{1a} \tag{A.3}$$

$$\frac{\partial \psi_{\chi 2}}{\partial \xi_2} = \sum_{a=1}^{4} \frac{\partial N_a}{\partial \xi_2} \chi_{2a} \tag{A.4}$$

Appendix B

Selected parts of Matlab code explained in chapter 4

B.1 Mesh motion

```
%------------Compute Kappa for the mesh motion--------------------
%kappa stays constant during the calculation
dist2interface=sqrt((X-X(link_matrix(:,1))).^2 ...
                 +(Y-Y(link_matrix(:,1))).^2);
domain_width=sqrt((X(link_matrix(:,1))-X(link_matrix(:,2))).^2 ...
                  +(Y(link_matrix(:,1))-Y(link_matrix(:,2))).^2);
kappa=1-dist2interface./domain_width;
%-------------Allocating memory for mesh velocity----------------------
v_mesh=zeros(NUM_DOF_PER_ELEM,1); %v_mesh only needed at the element level
VMESH=zeros(nnodes,NUM_DOF_PER_NODE);

%Inside Newton iteration loop
 theta=d_solid_new(3);
 R=[cos(theta), -sin(theta);
    sin(theta), cos(theta)];

T=zeros(3,interface_DOF_num);
A=zeros(interface_DOF_num,1);
%loop over interface nodes
X_interface=X(BC(:,1)==3);
Y_interface=Y(BC(:,2)==3);
for i=1:length(X_interface) %loop over nodes belonging to the interface
     T(:,2*i-1:2*i)=node_transformation_matrix...
                        (X_interface(i),Y_interface(i), theta);
     A(2*i-1:2*i)=-R*[X_interface(i);
                      Y_interface(i)];
end
v_fluid(fluid_DOF_num+1:end,nt+1)=T'*v_solid_new;
% node coordinate update and mesh motion
% order following the global numbering
d_A_GLOB=[d_solid_new(1)*one_vec;
         d_solid_new(2)*one_vec] + (R-eye2x2)*[X;
                                               Y];
XELEM=X+kappa.*d_A_GLOB(1,:);
YELEM=Y+kappa.*d_A_GLOB(2,:);
for i=1:nnodes
     VMESH(i,:)=(kappa(i)*...
```

```
     node_transformation_matrix(X(i),Y(i),theta)'*v_solid_new)';
end
    % inside the loop over all the elements for e = 1:nel
    %------ element nodes global coordinates -----------
    % note that x and y must be column vector to calculate D
    xelem = XELEM(connectivity(e,:))';
    yelem = YELEM(connectivity(e,:))';
    for i=1:NUM_DOF_PER_NODE
        v_mesh(i:NUM_DOF_PER_NODE:end)=VMESH(connectivity(e,:),i);
    end
```

B.2 Element matrices

```
function [Ge,NNe,NNe_diagparts,Me] = element_matrices2( xsi,eta,x,y,c,rho,mu )
%function that builds the element matrix N

N1=1/4*(1-xsi)*(1-eta);
N2=1/4*(1+xsi)*(1-eta);
N3=1/4*(1+xsi)*(1+eta);
N4=1/4*(1-xsi)*(1+eta);

N=[N1 0 N2 0 N3 0 N4 0;
   0 N1 0 N2 0 N3 0 N4 ];

dN1=-1/4*(1-eta); dN2=-1/4*(1-xsi);
dN3=1/4*(1-eta);  dN4=-1/4*(1+xsi);
dN5=1/4*(1+eta);  dN6=1/4*(1+xsi);
dN7=-1/4*(1+eta); dN8=1/4*(1-xsi);

dphi1_dxi=[dN1 dN3 dN5 dN7]*x; %sum(dNa/dxsi*xa);
dphi2_dxi=[dN1 dN3 dN5 dN7]*y; %sum(dNa/dxsi*ya);
dphi1_deta=[dN2 dN4 dN6 dN8]*x; %sum(dNa/deta*xa)
dphi2_deta=[dN2 dN4 dN6 dN8]*y; %sum(dNa/deta*ya)

D=[dphi2_deta -dphi1_deta;
   -dphi2_dxi dphi1_dxi];
Jx=dphi1_dxi*dphi2_deta-dphi1_deta*dphi2_dxi;



dND=[dN1 dN2; dN3 dN4; dN5 dN6; dN7 dN8]*D;

NNe_diagparts=(rho*dND*N*c*[N1 N2 N3 N4])'+mu/Jx*(dND*dND');
%NN21=mu/Jx*(dND*dND');

% Slightly faster
dNDv1=[dN1 dN2]*D;
dNDv2=[dN3 dN4]*D;
dNDv3=[dN5 dN6]*D;
dNDv4=[dN7 dN8]*D;

%additional line (compared to the program that ran only M and NN)
Ge=[dNDv1';dNDv2';dNDv3';dNDv4'];
```

```
NNe=mu/Jx*[dNDv1'*dNDv1, dNDv1'*dNDv2, dNDv1'*dNDv3, dNDv1'*dNDv4;
           dNDv2'*dNDv1, dNDv2'*dNDv2, dNDv2'*dNDv3, dNDv2'*dNDv4;
           dNDv3'*dNDv1, dNDv3'*dNDv2, dNDv3'*dNDv3, dNDv3'*dNDv4;
           dNDv4'*dNDv1, dNDv4'*dNDv2, dNDv4'*dNDv3, dNDv4'*dNDv4;];
Me=rho*Jx*(N'*N);

end
```

B.3 Assembly

B.3.1 Fluid DOF reorganization

```
%-------------Ordering the DOF-----------------------------------
free_DOF_num=0; % numbering the free DOF
prescribed_DOF_num=0; % numbering the prescribed non fixed  DOF
interface_DOF_num=0; % numbering the DOF belonging to the interface
%solid_prescribed_DOF_num=0; %number of prescibed solid DOF
nodeDOFnum = zeros(size(BC)); % ID array
%solid_ID=zeros(3,1);
f_prescribed=zeros(nnodes*NUM_DOF_PER_NODE,1);% at most
v_prescribed=zeros(nnodes*NUM_DOF_PER_NODE,1);% at most
a_prescribed=zeros(nnodes*NUM_DOF_PER_NODE,1);%at most

solid_DOF_num=size(solid_cond(solid_cond==0),2); %number of solid free DOF
for m = 1:nnodes
    for n = 1:NUM_DOF_PER_NODE
        if (BC(m,n)==0)
            free_DOF_num = free_DOF_num + 1;
            nodeDOFnum(m,n) = free_DOF_num;
            f_prescribed(free_DOF_num)=f_input(2*(m-1)+n);
        end
    end
end
f_prescribed(free_DOF_num+1:end)=[];
for m = 1:nnodes
    for n = 1:NUM_DOF_PER_NODE
        if (BC(m,n)==2)
            prescribed_DOF_num= prescribed_DOF_num + 1;
            nodeDOFnum(m,n) = free_DOF_num+prescribed_DOF_num;
            v_prescribed(prescribed_DOF_num)=v_input(2*(m-1)+n);
            a_prescribed(prescribed_DOF_num)=a_input(2*(m-1)+n);
        end
    end
end
%delete the extra part of v_prescribed and a_prescribed
v_prescribed(prescribed_DOF_num+1:end)=[];
a_prescribed(prescribed_DOF_num+1:end)=[];
for m = 1:nnodes
    for n = 1:NUM_DOF_PER_NODE
        if (BC(m,n)==3)
            interface_DOF_num= interface_DOF_num + 1;
```

```
            nodeDOFnum(m,n) = free_DOF_num+prescribed_DOF_num...
                              +interface_DOF_num;
        end
    end
end
a_solid_prescribed=a_solid_input(solid_cond==1);
v_solid_prescribed=v_solid_input(solid_cond==1);
d_solid_prescribed=d_solid_input(solid_cond==1,:);
equation_num=free_DOF_num+prescribed_DOF_num+interface_DOF_num;
fluid_DOF_num=free_DOF_num+prescribed_DOF_num;
```

B.3.2 ElemDOFnum matrix

```
%--elemDOFnum:global DOF number in terms of the element and the node DOF--
elemDOFnum = zeros(nel,NUM_DOF_PER_ELEM); % LM array
for m = 1:nel
    for n = 1:NUM_NODES_PER_ELEM
        elemDOFnum(m,(n-1)*NUM_DOF_PER_NODE+1:n*NUM_DOF_PER_NODE) = ...
            nodeDOFnum(connectivity(m,n),:);
    end
end
```

B.3.3 Assembly of $\mathbb{G}$, $\mathbb{M}$ and $\mathbb{N}$

```
%--elemDOFnum:global DOF number in terms of the element and the node DOF--
elemDOFnum = zeros(nel,NUM_DOF_PER_ELEM); % LM array
for m = 1:nel
    for n = 1:NUM_NODES_PER_ELEM
        elemDOFnum(m,(n-1)*NUM_DOF_PER_NODE+1:n*NUM_DOF_PER_NODE) = ...
            nodeDOFnum(connectivity(m,n),:);
    end
end
```

B.4 Solution procedures

B.4.1 Solution based on the Shur complement

```
 % COMPUTE f1, f2 AND f3 (residual equations)
     f1=M(1:free_DOF_num,:)*...
      [a_fluid_new;
       a_prescribed;
       T'*a_solid_new+A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
       + NN(1:free_DOF_num,:)*...
         [v_fluid_star+dt*gamma_v*a_fluid_new;
          v_prescribed;
          T'*(v_solid_star+dt*gamma*a_solid_new)]...
       -G(1:free_DOF_num,:)*pressure_new...
```

```
  -f_prescribed;

m_star=m_matrix_input + T*M(fluid_DOF_num+1:end,fluid_DOF_num+1:end)...
                        *T';

f2=m_star*a_solid_new...
    +c_matrix_input*(v_solid_star+dt*gamma*a_solid_new)...
    +k_matrix_input*(d_solid_star+beta*dt^2*a_solid_new)...
    +T*M(fluid_DOF_num+1:end,:)*...
     [a_fluid_new;
      a_prescribed;
      A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
    +T*NN(fluid_DOF_num+1:end,:)*...
     [v_fluid_star+dt*gamma_v*a_fluid_new;
      v_prescribed;
      T'*(v_solid_star+dt*gamma*a_solid_new)]...
    -T*G(fluid_DOF_num+1:end,:)*pressure_new;

f3=G'*[v_fluid_star+dt*gamma_v*a_fluid_new;
       v_prescribed;
       T'*(v_solid_star+dt*gamma*a_solid_new)];

% NORM OF THE RESIDUAL
residual_norm=norm([f1;f2(solid_cond==0);f3]);

% INTERMEDIATE STEPS
Mbar=M(1:free_DOF_num,1:free_DOF_num)...
    +NN(1:free_DOF_num,1:free_DOF_num)*dt*gamma_v;
mstarbar=m_star(solid_cond==0,solid_cond==0)+...
    c_matrix_input(solid_cond==0,solid_cond==0)*dt*gamma+...
    k_matrix_input(solid_cond==0,solid_cond==0)*beta*dt^2;
 B=[Mbar zeros(free_DOF_num,solid_DOF_num) -G(1:free_DOF_num,:);
   zeros(solid_DOF_num,free_DOF_num) mstarbar -T(solid_cond==0,:)...
   *G(fluid_DOF_num+1:end,:);
  dt*gamma_v*G(1:free_DOF_num,:)' dt*gamma*(T(solid_cond==0,:)...
  *G(fluid_DOF_num+1:end,:))' zeros(nel,nel)];

 % SOLVE FOR INCREMENTS IN ACCELERATION AND PRESSURE
temp_a1=Mbar\f1;
temp_a2=Mbar\G(1:free_DOF_num,:);
temp_alpha1=mstarbar\f2(1:solid_DOF_num);
temp_alpha2=mstarbar\(T(1:solid_DOF_num,:)*G(fluid_DOF_num+1:end,:));

K=gamma_v*dt*G(1:free_DOF_num,:)'*temp_a2...
  +gamma*dt*G(fluid_DOF_num+1:end,:)'*T(1:solid_DOF_num,:)'...
  *temp_alpha2;

% SOLVE FOR INCREMENTS IN ACCELERATION AND PRESSURE
Delta_p=K\(-f3+G(1:free_DOF_num,:)'*gamma_v*dt*temp_a1...
              +dt*gamma*G(fluid_DOF_num+1:end,:)'*T(1:solid_DOF_num,:)'...
              *temp_alpha1);
Delta_a=-temp_a1+temp_a2*Delta_p;
Delta_alpha=-temp_alpha1+temp_alpha2*Delta_p;
```

B.4.2 Solution based on direct inversion of B

```
% COMPUTE f1, f2 AND f3 (residual equations)
    f1=M(1:free_DOF_num,:)*...
     [a_fluid_new;
      a_prescribed;
      T'*a_solid_new+A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
      + NN(1:free_DOF_num,:)*...
        [v_fluid_star+dt*gamma_v*a_fluid_new;
         v_prescribed;
         T'*(v_solid_star+dt*gamma*a_solid_new)]...
      -G(1:free_DOF_num,:)*pressure_new...
      -f_prescribed;

    m_star=m_matrix_input + T*M(fluid_DOF_num+1:end,fluid_DOF_num+1:end)...
                                    *T';

    f2=m_star*a_solid_new...
        +c_matrix_input*(v_solid_star+dt*gamma*a_solid_new)...
        +k_matrix_input*(d_solid_star+beta*dt^2*a_solid_new)...
        +T*M(fluid_DOF_num+1:end,:)*...
         [a_fluid_new;
          a_prescribed;
          A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
        +T*NN(fluid_DOF_num+1:end,:)*...
         [v_fluid_star+dt*gamma_v*a_fluid_new;
          v_prescribed;
          T'*(v_solid_star+dt*gamma*a_solid_new)]...
        -T*G(fluid_DOF_num+1:end,:)*pressure_new;

    f3=G'*[v_fluid_star+dt*gamma_v*a_fluid_new;
           v_prescribed;
           T'*(v_solid_star+dt*gamma*a_solid_new)];

    % NORM OF THE RESIDUAL
    residual_norm=norm([f1;f2(solid_cond==0);f3]);

    % INTERMEDIATE STEPS
    Mbar=M(1:free_DOF_num,1:free_DOF_num)...
        +NN(1:free_DOF_num,1:free_DOF_num)*dt*gamma_v;
    mstarbar=m_star(solid_cond==0,solid_cond==0)+...
        c_matrix_input(solid_cond==0,solid_cond==0)*dt*gamma+...
        k_matrix_input(solid_cond==0,solid_cond==0)*beta*dt^2;
     B=[Mbar zeros(free_DOF_num,solid_DOF_num) -G(1:free_DOF_num,:);
       zeros(solid_DOF_num,free_DOF_num) mstarbar -T(solid_cond==0,:)...
       *G(fluid_DOF_num+1:end,:);
      dt*gamma_v*G(1:free_DOF_num,:)' dt*gamma*(T(solid_cond==0,:)...
      *G(fluid_DOF_num+1:end,:))' zeros(nel,nel)];

     % SOLVE FOR INCREMENTS IN ACCELERATION AND PRESSURE
     Delta=B\[-f1;-f2(solid_cond==0);-f3];


    Delta_a=Delta(1:free_DOF_num);
    Delta_alpha=Delta(free_DOF_num+1:free_DOF_num+solid_DOF_num);
```

```
    Delta_p=Delta(free_DOF_num+solid_DOF_num+1:end);
```

B.5 Update of the dependent variables

```
%--------------------Time integration---------------------------------
for nt=1:fix(tt/dt)
    %nt

    %Initialization of Newton's method
    % 1) Initial guess
    a_fluid_new=zeros(free_DOF_num,1);
    v_fluid_star=v_fluid(1:free_DOF_num,nt)...
                +dt*(1-gamma_v)*a_fluid(1:free_DOF_num,nt);
    pressure_new=pressure(:,nt);
    v_solid_star=v_solid(:,nt)+dt*(1-gamma)*a_solid(:,nt); %useful to compute residual
    d_solid_star=d_solid(:,nt)+dt*v_solid(:,nt)...
                                +dt^2/2*(1-2*beta)*a_solid(:,nt);
    %solid acceleration, velocity, displacement (needed entirely)
    a_solid(solid_cond==0,nt+1)=zeros(solid_DOF_num,1);
    a_solid(solid_cond==1,nt+1)=a_solid_prescribed;
    v_solid(solid_cond==0,nt+1)=v_solid_star(solid_cond==0);
    v_solid(solid_cond==1,nt+1)=v_solid_prescribed;
    d_solid(solid_cond==0,nt+1)=d_solid_star(solid_cond==0);
    d_solid(solid_cond==1,nt+1)=d_solid_prescribed(:,nt+1);
    a_solid_new=a_solid(:,nt+1);
    v_solid_new=v_solid(:,nt+1);
    d_solid_new=d_solid(:,nt+1);
    %initializing v_fluid (needed for convective velocity c)
    v_fluid(1:free_DOF_num,nt+1)=v_fluid_star;
    % Assigning prescribed part of the solution
    v_fluid(free_DOF_num+1:fluid_DOF_num,nt+1)=v_prescribed;
    a_fluid(free_DOF_num+1:fluid_DOF_num,nt+1)=a_prescribed;
%     v_fluid(fluid_DOF_num+1:end,nt+1)=T'*v_solid_new;
%     (needs to be updated every iteration)
    v_fluid_new=v_fluid(:,nt+1);

    % 2) initial value of the error and iteration number
    residual_norm=1;
    iteration_num=0;

    %Newton's method iterations
    while residual_norm>tol && iteration_num<=100
        % INITIALIZING ntriplets_G AND ntriplets_MNN
        ntriplets_G = 0;
        ntriplets_M = 0;
        ntriplets_NN = 0;
        % DISPLACEMENT OF INTERFACE NODES, T AND A MATRICES
        iteration_num=iteration_num+1;
%         if iteration_num==500
%             disp('number of iteration exceeds 500');
%             break
%         end
        % solid rotation
```

```
        theta=d_solid_new(3);
        R=[cos(theta), -sin(theta);
           sin(theta), cos(theta)];

        T=zeros(3,interface_DOF_num);
        A=zeros(interface_DOF_num,1);
        %loop over interface nodes
        X_interface=X(BC(:,1)==3);
        Y_interface=Y(BC(:,2)==3);
        for i=1:length(X_interface) %loop over nodes belonging to the interface
            T(:,2*i-1:2*i)=node_transformation_matrix...
                           (X_interface(i),Y_interface(i), theta);
            A(2*i-1:2*i)=-R*[X_interface(i);
                             Y_interface(i)];
        end
        v_fluid(fluid_DOF_num+1:end,nt+1)=T'*v_solid_new;
        % node coordinate update and mesh motion
        % order following the global numbering
        d_A_GLOB=[d_solid_new(1)*one_vec;
                  d_solid_new(2)*one_vec] + (R-eye2x2)*[X;
                                                        Y];
        XELEM=X+kappa.*d_A_GLOB(1,:);
        YELEM=Y+kappa.*d_A_GLOB(2,:);
        for i=1:nnodes
            VMESH(i,:)=(kappa(i)*...
            node_transformation_matrix(X(i),Y(i),theta)'*v_solid_new)';
        end

        % GET MATRICES M,NN,G
        for e = 1:nel % loop over all the elements
            %------ element nodes global coordinates -----------
            % note that x and y must be column vector to calculate D
            xelem = XELEM(connectivity(e,:))';
            yelem = YELEM(connectivity(e,:))';
            %---------Convective velocity c -----------------------
            %c=v_fluid-v_mesh;
            for i=1:NUM_DOF_PER_NODE
                v_mesh(i:NUM_DOF_PER_NODE:end)=VMESH(connectivity(e,:),i);
            end
%             if a DOF is fixed, elemDOFnum(e,i)=0
             c=zeros(NUM_NODES_PER_ELEM*NUM_DOF_PER_NODE,1);
             c(elemDOFnum(e,:)>0) = ...
                           v_fluid_new(elemDOFnum(e,elemDOFnum(e,:)>0))...
                                  -v_mesh(elemDOFnum(e,:)>0) ;
            %--------- Element matrices ---------------------------
            [Ge1,NNe1,NNe_diag1,Me1]=element_matrices2(-1/sqrt(3),...
                                       -1/sqrt(3),xelem,yelem,c,rho,mu);
            [Ge2,NNe2,NNe_diag2,Me2]=element_matrices2(1/sqrt(3),...
                                       -1/sqrt(3),xelem,yelem,c,rho,mu);
            [Ge3,NNe3,NNe_diag3,Me3]=element_matrices2(1/sqrt(3),...
                                       1/sqrt(3),xelem,yelem,c,rho,mu);
            [Ge4,NNe4,NNe_diag4,Me4]=element_matrices2(-1/sqrt(3),...
                                       1/sqrt(3),xelem,yelem,c,rho,mu);
            Ge=Ge1+Ge2+Ge3+Ge4;
            NNe=NNe1+NNe2+NNe3+NNe4;
```

```
        NNe_diag=NNe_diag1+NNe_diag2+NNe_diag3+NNe_diag4;
        Me=Me1+Me2+Me3+Me4;
        %-------------Assembling the matrices-----------------------
        for mm = 1:NUM_DOF_PER_ELEM % loop over rows of the elmnt mat
            if (elemDOFnum(e,mm) > 0) % this dof is not fixed
                ntriplets_G = ntriplets_G + 1;
                IG(ntriplets_G) = elemDOFnum(e,mm);
                JG(ntriplets_G) = e; %column index in the global G matrix
                XG(ntriplets_G) = Ge(mm,1);

                for nn=1:NUM_DOF_PER_ELEM % loop over the columns
                                          % of element matrices
                    if (elemDOFnum(e,nn) > 0) % this dof is not fixed
                        ntriplets_M= ntriplets_M + 1;
                        IM(ntriplets_M) = elemDOFnum(e,mm);
                        JM(ntriplets_M) = elemDOFnum(e,nn);
                        XM(ntriplets_M) = Me(mm,nn);
                    end
                end
            end
        end
        for mm=1:NUM_NODES_PER_ELEM
             for nn=1:NUM_NODES_PER_ELEM
                 if (elemDOFnum(e,2*mm-1)>0 && elemDOFnum(e,2*nn-1)>0)
                     ntriplets_NN= ntriplets_NN + 1;
                     INN(ntriplets_NN) = elemDOFnum(e,2*mm-1);
                     JNN(ntriplets_NN) = elemDOFnum(e,2*nn-1);
                     XNN(ntriplets_NN) = NNe(2*mm-1,2*nn-1)...
                                         +NNe_diag(mm,nn);
                 end
                 if (elemDOFnum(e,2*mm-1)>0 && elemDOFnum(e,2*nn)>0)
                     ntriplets_NN= ntriplets_NN + 1;
                     INN(ntriplets_NN) = elemDOFnum(e,2*mm-1);
                     JNN(ntriplets_NN) = elemDOFnum(e,2*nn);
                     XNN(ntriplets_NN) = NNe(2*mm-1,2*nn);
                 end
                 if (elemDOFnum(e,2*mm)>0 && elemDOFnum(e,2*nn-1)>0)
                     ntriplets_NN= ntriplets_NN + 1;
                     INN(ntriplets_NN) = elemDOFnum(e,2*mm);
                     JNN(ntriplets_NN) = elemDOFnum(e,2*nn-1);
                     XNN(ntriplets_NN) = NNe(2*mm,2*nn-1);
                 end
                 if (elemDOFnum(e,2*mm)>0 && elemDOFnum(e,2*nn)>0)
                     ntriplets_NN= ntriplets_NN + 1;
                     INN(ntriplets_NN) = elemDOFnum(e,2*mm);
                     JNN(ntriplets_NN) = elemDOFnum(e,2*nn);
                     XNN(ntriplets_NN) = NNe(2*mm,2*nn)+NNe_diag(mm,nn);
                 end
            end
        end
    end
    G = sparse(IG(1:ntriplets_G), JG(1:ntriplets_G), XG(1:ntriplets_G), ...
             equation_num, nel);
    M = sparse(IM(1:ntriplets_M), JM(1:ntriplets_M), ...
        XM(1:ntriplets_M), equation_num, equation_num);
```

```
   NN = sparse(INN(1:ntriplets_NN), JNN(1:ntriplets_NN), ...
        XNN(1:ntriplets_NN), equation_num, equation_num);
% COMPUTE f1, f2 AND f3 (residual equations)
f1=M(1:free_DOF_num,:)*...
 [a_fluid_new;
  a_prescribed;
  T'*a_solid_new+A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
  + NN(1:free_DOF_num,:)*...
    [v_fluid_star+dt*gamma_v*a_fluid_new;
     v_prescribed;
     T'*(v_solid_star+dt*gamma*a_solid_new)]...
  -G(1:free_DOF_num,:)*pressure_new...
  -f_prescribed;

m_star=m_matrix_input + T*M(fluid_DOF_num+1:end,fluid_DOF_num+1:end)...
                                    *T';

f2=m_star*a_solid_new...
    +c_matrix_input*(v_solid_star+dt*gamma*a_solid_new)...
    +k_matrix_input*(d_solid_star+beta*dt^2*a_solid_new)...
    +T*M(fluid_DOF_num+1:end,:)*...
     [a_fluid_new;
      a_prescribed;
      A*(v_solid_star(3)+dt*gamma*a_solid_new(3))^2]...
    +T*NN(fluid_DOF_num+1:end,:)*...
     [v_fluid_star+dt*gamma_v*a_fluid_new;
      v_prescribed;
      T'*(v_solid_star+dt*gamma*a_solid_new)]...
    -T*G(fluid_DOF_num+1:end,:)*pressure_new;

f3=G'*[v_fluid_star+dt*gamma_v*a_fluid_new;
       v_prescribed;
       T'*(v_solid_star+dt*gamma*a_solid_new)];

% NORM OF THE RESIDUAL
residual_norm=norm([f1;f2(solid_cond==0);f3]);

% INTERMEDIATE STEPS
Mbar=M(1:free_DOF_num,1:free_DOF_num)...
    +NN(1:free_DOF_num,1:free_DOF_num)*dt*gamma_v;
mstarbar=m_star(solid_cond==0,solid_cond==0)+...
    c_matrix_input(solid_cond==0,solid_cond==0)*dt*gamma+...
    k_matrix_input(solid_cond==0,solid_cond==0)*beta*dt^2;
 B=[Mbar zeros(free_DOF_num,solid_DOF_num) -G(1:free_DOF_num,:);
   zeros(solid_DOF_num,free_DOF_num) mstarbar -T(solid_cond==0,:)...
                        *G(fluid_DOF_num+1:end,:);
  dt*gamma_v*G(1:free_DOF_num,:)' dt*gamma*(T(solid_cond==0,:)...
                   *G(fluid_DOF_num+1:end,:))' zeros(nel,nel)];

 % SOLVE FOR INCREMENTS IN ACCELERATION AND PRESSURE
Delta=B\[-f1;-f2(solid_cond==0);-f3];
Delta_a=Delta(1:free_DOF_num);
Delta_alpha=Delta(free_DOF_num+1:free_DOF_num+solid_DOF_num);
Delta_p=Delta(free_DOF_num+solid_DOF_num+1:end);
```

```
      % UPDATE
      a_fluid_new=a_fluid_new+Delta_a;
      a_solid_new(solid_cond==0)=a_solid_new(solid_cond==0)...
                                  +Delta_alpha;
      pressure_new=pressure_new+Delta_p;

      v_fluid_new(1:free_DOF_num)=v_fluid_new(1:free_DOF_num)...
                                 +gamma_v*dt*Delta_a;
      v_solid_new(solid_cond==0)=v_solid_new(solid_cond==0)...
                                  +gamma*dt*Delta_alpha;
      d_solid_new(solid_cond==0)=d_solid_new(solid_cond==0)...
                                  +beta*dt^2*Delta_alpha;
    end
    % Assigning the free part of the solution
    a_fluid(1:free_DOF_num,nt+1)=a_fluid_new;
    v_fluid(1:free_DOF_num,nt+1)=v_fluid_new(1:free_DOF_num);
    a_solid(solid_cond==0,nt+1)=a_solid_new(solid_cond==0);
    v_solid(solid_cond==0,nt+1)=v_solid_new(solid_cond==0);
    d_solid(solid_cond==0,nt+1)=d_solid_new(solid_cond==0);
    pressure(:,nt+1)=pressure_new;
    a_fluid(fluid_DOF_num+1:end,nt+1)=T'*a_solid_new...
                                            +A*v_solid_new(3)^2;
    v_fluid(fluid_DOF_num+1:end,nt+1)=T'*v_solid_new;
end
```

Appendix C

Matlab code for the numerical examples input files

C.1 Couette flow problems

C.1.1 Problem with solid as the fixed boundary

```
clear all
%INPUT

g=9.81; %m.s^(-2)
%Flow parameters
%assumptions : homogeneous fluid and incompressible flow
%parameters for water at 20C
rho=1; % kg/m^3
mu=1e-3; % Pa.s

% Solid parameters
%number of solid DOF
% Mass matrix (3*3, diagonal)
m_matrix_input=0*diag(ones(3,1));
% Damping matrix
c_matrix_input=0*diag(ones(3,1));
% Stiffness matrix
k_matrix_input=0*diag(ones(3,1));

% Nodes initial coordinates
L=4;
H=2;
X = [-L/2, 0, L/2, -L/2, 0, L/2, -L/2, 0, L/2]; % global x coords of the nodes
Y = [0, 0, 0, H/2, H/2, H/2, H, H, H]; % global y coords of the nodes

% Time
tt=2; %total time

% Increments
dt=1; % Delta_t
n_inc=fix(tt/dt); % # of increments

% Tolerance
tol=10^(-6);
```

```
% Boundary conditions
%0 is free, 1 is fixed, 2 is a prescribed (nonzero) velocity, 3 is a node
%belonging to the interface fluid-solid.
BC = zeros(9,2);
BC(1:3,:)=3;
BC(4,1)=2;
BC(6,1)=2;
BC(7:9,1)=2*ones(3,1);

%solid condition: to know if the DOF are fixed or prescribed
% 0 is free, 1 is prescribed
solid_cond=[1 1 1];

% Prescribed_velocity and acceleration
U=0.5;
v_input=zeros(9*2,1);
v_input([7 11],:)=U/2*ones(2,1);
v_input([13 15 17],:)=U*ones(3,1);
a_input=zeros(9*2,1);
a_solid_input=[0 0 0];
v_solid_input=[0 0 0];
d_solid_input=zeros(3,n_inc+1);

% Prescribed force
f_input=zeros(9*2,1);
% If pressure is initially 15
% f_input([14 18],:)=-15;
% f_input(16,:)=-30;

% Initial conditions
a_fluid_init=zeros(9*2,1);
v_fluid_init=zeros(9*2,1);
v_fluid_init(9)=U/2;
v_fluid_init([7 11],:)=U/2*ones(2,1);
v_fluid_init([13 15 17],:)=U*ones(3,1);
p_init=[0;0;0;0];
%p_init=[15;15;15;15];
 a_solid_init=zeros(3,1);
 v_solid_init=zeros(3,1);
 d_solid_init=zeros(3,1);

% Connectivity matrix (nel*4 matrix for us)
connectivity = [1 2 5 4;
                2 3 6 5;
                4 5 8 7;
                5 6 9 8];

% Matrix that relates nodes to their closest interface and boundary node
% (2 columns: first one is the closest interface node,
% second one is the closest outer boundary node)
link_matrix=zeros(9,2);
link_matrix(:,1)=[1; 2; 3; 1; 2; 3; 1; 2; 3];
link_matrix(:,2)=[7; 8; 9; 7; 8; 9; 7; 8; 9];
```

```
% Newmark's method parameters
gamma_v=1/2;
gamma=1/2;
beta=0.25;


% END OF INPUT
```

C.1.2 Problem with solid as the mobile boundary

```
clear all
%INPUT

g=9.81; %m.s^(-2)
%Flow parameters
%assumptions : homogeneous fluid and incompressible flow
%parameters for water at 20C
rho=1; % kg/m^3
mu=1e-3; % Pa.s

% Solid parameters
%number of solid DOF
% Mass matrix (3*3, diagonal)
m_matrix_input=0*diag(ones(3,1));
% Damping matrix
c_matrix_input=0*diag(ones(3,1));
% Stiffness matrix
k_matrix_input=0*diag(ones(3,1));

% Nodes initial coordinates
L=4;
H=2;
X = [-L/2, 0, L/2, -L/2, 0, L/2, -L/2, 0, L/2]; % global x coords of the nodes
Y = [-H, -H, -H, -H/2, -H/2, -H/2, 0, 0, 0]; % global y coords of the nodes

% Time
tt=1; %total time

% Increments
dt=0.5; % Delta_t
n_inc=fix(tt/dt); % # of increments

% Tolerance
tol=10^(-12);

% Boundary conditions
%0 is free, 1 is fixed, 2 is a prescribed (nonzero) velocity, 3 is a node
%belonging to the interface fluid-solid.
BC = zeros(9,2);
BC(1:3,1)=1;
BC(4,1)=2;
BC(6,1)=2;
```

```
BC(7:9,:)=3;
%solid condition: to know if the DOF are fixed or prescribed
% 0 is free, 1 is prescribed
solid_cond=[1 1 1];

% Prescribed_velocity and acceleration
U=0.3;
v_input=zeros(9*2,1);
v_input([7 11],:)=U/2*ones(2,1);
a_input=zeros(9*2,1);
a_solid_input=[0 0 0];
v_solid_input=[U 0 0];
d_solid_input=zeros(3,n_inc+1);
for nt=1:n_inc
    d_solid_input(:,nt+1)=[nt*U*dt,0,0]';
end

% Prescribed force
f_input=zeros(9*2,1);

% Initial conditions
a_fluid_init=zeros(9*2,1);
v_fluid_init=zeros(9*2,1);
v_fluid_init(9)=U/2;
v_fluid_init([7 11],:)=U/2*ones(2,1);
v_fluid_init([13 15 17],:)=U*ones(3,1);
% pressure
p_init=[0;0;0;0];
 a_solid_init=zeros(3,1);
 v_solid_init=zeros(3,1);
 v_solid_init(1)=U;
 d_solid_init=zeros(3,1);

% Connectivity matrix (nel*4 matrix for us)
connectivity = [1 2 5 4;
                2 3 6 5;
                4 5 8 7;
                5 6 9 8];

% Matrix that relates nodes to their closest interface and boundary node
% (2 columns: first one is the closest interface node (actually closest
% Lagrangian boundary node), second one is the closest
% (Eulerian) boundary node)
link_matrix=zeros(9,2);
link_matrix(:,1)=[7; 8; 9; 7; 8; 9; 7; 8; 9];
link_matrix(:,2)=[1; 2; 3; 1; 2; 3; 1; 2; 3];


% Newmark's method parameters
gamma_v=1/2;
gamma=1/2;
beta=0.25;


% END OF INPUT
```

C.2 Oscillating cylinder problem

C.2.1 Input file

```
% INPUT FILE FOR THE PROBLEM OF FREE VIBRATIONS OF A CIRCULAR CYLINDER

%Flow parameters
%assumptions : homogeneous fluid and incompressible flow
%parameters for water from the article
rho=1; % kg/m^3
mu=1.33e-3; % Pa.s

% %parameters for silicon oil from the article
% rho=0.956; % kg/m^3
% mu=0.145; % Pa.s

% %parameters for mineral oil from the article
% rho=0.935; % kg/m^3
% mu=0.041; % Pa.s

% %parameters for air from the article
% rho=1.18e-3; % kg/m^3
% mu=1.82e-5; % Pa.s

% Solid parameters
%number of solid DOF
% Mass matrix (3*3, diagonal)
m_matrix_input=0*diag(ones(3,1));
m_matrix_input(1,1)=3.408e-3; %kg
% Damping matrix
c_matrix_input=0*diag(ones(3,1));
% Stiffness matrix
k_matrix_input=0*diag(ones(3,1));
k_matrix_input(1,1)=34.6113; %kg/s^2=N/m

% Geometrical parameters
d=1.27e-2; %m
D_domain=5*d;

% parameters for mesh generation
narc=10;
nrad=48;


% Time
tt=0.6; %total time (s)

% Increments
dt=1e-3; % Delta_t for water and air
% dt=5e-5; % Delta_t for silicon and mineral oil
n_inc=fix(tt/dt); % # of increments

% Tolerance
tol=10^(-4);
```

```
% Matrices associated with the mesh

[X,Y,link_matrix,connectivity,BC] = full_mesh_generation( d,D_domain,narc,nrad );

%solid condition: to know if the DOF are fixed or prescribed
% 0 is free, 1 is prescribed
solid_cond=[0 1 1];

% Prescribed_velocity and acceleration
v_input=zeros(narc*nrad*2,1);
a_input=zeros(narc*nrad*2,1);
a_solid_input=[0 0 0];
v_solid_input=[0 0 0];
d_solid_input=zeros(3,n_inc+1);

% Initial conditions
% pressure
p=0;
%p_init=p*ones((narc-1)*(nrad-1),1);
p_init=zeros((narc-1)*nrad,1);
a_fluid_init=zeros(narc*nrad*2,1);
v_fluid_init=zeros(narc*nrad*2,1);
a_solid_init=zeros(3,1);
v_solid_init=zeros(3,1);
d_solid_init=zeros(3,1);
d_solid_init(1)=d/100;

% Prescribed force
f_input=zeros(narc*nrad*2,1);

% free one of the boundary DOF
BC(433,1)=0;

% Newmark's method parameters
gamma_v=1/2;
gamma=1/2;
beta=0.25;

%END OF INPUT
```

C.2.1.1 Mesh generation code

```
function [X,Y,link,connect,BC] = full_mesh_generation( d,D,narc,nrad )
%Function to generate the axisymmetric mesh using advancing front

% nrad needs to be even

% Initialize X, Y, link (link_matrix), connect (connectivity)
X=zeros(1,narc*nrad); % X and Y inputs are row vectors
Y=zeros(1,narc*nrad);
link=zeros(narc*nrad,2);
connect=zeros((narc-1)*(nrad-1),4);
```

```
e=0;
%R=0;
R=d/2;
    for i=1:narc-1
        R=R+log10(i)*(D/2-d/2)/(narc-1);
       % R=R+r^(i-1)*d/2;
        for j=1:nrad-1
            theta=(j-1)*pi/(nrad/2);
            X((i-1)*nrad+j)=R*cos(theta);
            Y((i-1)*nrad+j)=R*sin(theta);
            link((i-1)*nrad+j,:)=[j,(narc-1)*nrad+j];
            e=e+1;
            connect(e,:)=[(i-1)*nrad+j, i*nrad+j, ...
                                    i*nrad+j+1, (i-1)*nrad+j+1];
        end
        % j=nrad
        theta=(nrad-1)*pi/(nrad/2);
        X(i*nrad)=R*cos(theta);
        Y(i*nrad)=R*sin(theta);
        link(i*nrad,:)=[nrad,narc*nrad];
        e=e+1;
        connect(e,:)=[i*nrad, (i+1)*nrad, i*nrad+1, (i-1)*nrad+1];
    end

    %i=narc; want to make sure R is exactly D/2
    R=D/2;
    for j=1:nrad
        theta=(j-1)*pi/(nrad/2);
        X((narc-1)*nrad+j)=R*cos(theta);
        Y((narc-1)*nrad+j)=R*sin(theta);
        link((narc-1)*nrad+j,:)=[j,(narc-1)*nrad+j];
    end
    % BC
    %0 is free, 1 is fixed, 2 is a prescribed (nonzero) velocity, 3 is a node
    %belonging to the interface fluid-solid.
    BC=zeros(narc*nrad,2);
    %interface
    BC(1:nrad,:)=3;
    %fixed boundary
    BC((narc-1)*nrad+1:narc*nrad,:)=1;

    % Symmetric part of the model

end
```