Numerical Simulation of Orbitally Shaken Reactors

THÈSE N^{O} 5477 (2012)

PRÉSENTÉE LE 20 SEPTEMBRE 2012

À LA FACULTÉ DES SCIENCES DE BASE
CHAIRE DE MODÉLISATION ET CALCUL SCIENTIFIQUE
PROGRAMME DOCTORAL EN MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Samuel QUINODOZ

acceptée sur proposition du jury:

Prof. T. Mountford, président du jury
Prof. A. Quarteroni, Dr M. Discacciati, directeurs de thèse
Prof. R. Codina, rapporteur
Prof. F. Nobile, rapporteur
Dr N. Parolini, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2012

Acknowledgements

First of all, I would like to thank Prof. Alfio Quarteroni for having shared with me a piece of his infinite mathematical knowledge. His contribution does not limit to the mathematical level: I appreciated his patience and wisdom as well as his capacity to take the best out of me. I would like also to thank a lot Dr. Marco Discacciati for his so precious pieces of advise that he gave me all along my thesis and for preconditioning all my manuscripts. It was a real pleasure to work with him.

I acknowledge the jury, namely Prof. Ramon Codina, Prof. Fabio Nobile and Dr. Nicola Parolini, for their careful reading of this thesis and for their pieces of advice. Prof. Thomas Mountford is also acknowledged for having presided the jury.

During my thesis, I had many colleagues and there is none of them I don't want to thank. I think that we had some great moments, especially during the three "sledge-days" and the two kart sessions. In particular, I thank Matteo Lombardi for the great moments we shared on mountain bikes. I also thank my successive roommates: Cristiano Malossi for having taught me the basics of Italian, Paolo Tricerri for the "advanced" matter and Dr. Vincent "Kouz" Keller for the deepest HPC topics. I am also grateful to Gwenol Grandperrin for coming every day as early as I do and sharing the French translations/corrections duty with me. I thank also the students that worked with me during their projects, Susanna Carcano, Claudia Colciago (who is now my most talkative colleague), Laura Cattaneo, Luis De Oliveira and Ivan Kuraj. I also do not forget those who stay in the "shadow", i.e. Léonard Gross for the precious computer support and the secretaries.

I would like also to thank all the LifeV community for the great discussions we had on the mailing list. In particular, I am very grateful to Dr. Simone Deparis, who was my "LifeV-mother", and much more.

The Swiss National Science Foundation is acknowledged for financing the Sinergia project, of which this thesis is part. I also thank all my Sinergia colleagues from the LBTC and the LMH, in particular Dr. Stéphanie Tissot, who introduced me to the bioreactors, Martino Reclari and Dominique Monteil. I also acknowledge the CSCS for offering me computational time on the Rosa supercomputer and the HPC-DIT group from EPFL for the access to Antares that I used for most of the simulations.

I don't forget all my friends outside the work, those from the EPFL and from Valais: Raphael Huser, Carol Kirchhofer, Sébastien Epiney, Stéphane Flotron and all the other ones. I thank

especially Sébastien Coupy and Laurent Carroz for the moments I spend with them on the singletrails.

I would like to thank my whole family, especially my mother, my father and my two sisters for their unconditional and essential support. I thank also all those who supported me with candies and chocolate. Finally, I thank my dear Aurélie, for having supported me during my work and especially the writing of this thesis. I hope that I will have my whole life span to give it back to her.

Abstract

Mammalian cell cultures have become a major topic of research in the biopharmaceutical industry. This kind of cells require specific conditions to grow. In this thesis, we study the hydrodynamics of orbitally shaken reactors (OSR), a recently introduced kind of bioreactors for mammalian cell cultures that represents a simple to operate and cheap alternative to commonly used reactors such as stirred tanks. OSRs can provide suitable conditions for small scale cell cultures, however a deeper understanding of the principles governing the OSRs is required to exploit their full potentiality and proceed with scaling up. This work aims at shedding light into the mechanisms of the OSRs through computational fluid dynamics.

OSRs are only partially filled with liquid medium, the remaining space is occupied by air. When an OSR is agitated, the interface between the two phases moves and creates different shapes. This interface is at the heart of the simulation of OSRs: not only its location is part of the problem, but it can also carry singularities. In particular, the pressure has usually a low regularity in the vicinity of the interface and numerical methods might underperform if the singularity is not treated in an appropriate manner.

This motivated the study of an elliptic problem in a medium with an internal interface carrying discontinuities. In this work, we devise a novel method called SESIC to solve this kind of problem. It uses the a priori knowledge to improve the numerical accuracy in the vicinity of the interface by removing the singularities. We prove that this method yields optimal orders of convergence in H^1 and L^2 norm. Numerical tests also show that optimal orders can be obtained in the L^∞ norm in some cases. Regularized integration is also investigated with the perspective of further simplifying the scheme. It is found that, if the regularization bandwidth is suitably defined, good approximations can still be obtained, even if the convergence rate is decreased.

We apply then the methodology of the SESIC method to the approximation of the two-phase Navier-Stokes equations, which amounts to correct the pressure. If adapted integration is used with it, the densitiy and viscosity can be kept discontinuous across the interface without creating spurious velocity, as shown by numerical experiments. The sharp treatment of the discontinuities improves the accuracy of the simulations by retaining the physical meaning of the phenomena independently of the mesh size. We also pay attention to the boundary conditions used, which must be suitably chosen to allow the interface motion but still reproduce the wall friction. We show that imposing the zero normal component of the velocity yields the best results for the no-penetration condition and that it must be employed with a correction term to avoid spurious velocities. Robin-type conditions are used for the

tangential components to recover the no-slip condition far from the contact line.
Specific tests are performed to assess the quality of the different components of the method. We also compare it with a regularized density/viscosity method and show that the sharp treatment of these physical quantities improves the quality of the simulation. The scalability properties of the method are also investigated and the bottlenecks pointed out.
Our method is then validated in various ways with experimental data. First of all, glycerine filled OSRs are simulated and we show that our method reproduces accurately the amplitude of the generated wave. The sensibility of the results with respect to the Robin condition is shown to be weak. We investigate then water filled OSRs. The different wave patterns, either breaking or non-breaking, single or multiple, observed experimentally are reproduced for various configurations. In particular, triple waves are obtained as well. We use laser Doppler velocimetry measures of the velocity field to further validate our simulations. The hydrodynamic stress and the mixing pattern of the different regimes are evaluated and put into relationship with the wave shape.
Finally, we investigate the modelling of the cell culture by devising a system non-linear ODEs which represents the evolution in time of the main nutriments and wastes and the growth of the cell population. The behaviour of the cell culture is well reproduced, but some phenomena remain unexplained. In particular, our model contains a toxic waste whose actual identity is discussed. Two alternative scenarios are proposed to improve the model.

Keywords: Navier-Stokes equations; Multi-phase free-surface flows; Finite elements method; Level set; Cell-growth models.

Résumé

Les cultures de cellules mammifères sont de nos jours un thème majeur de recherche dans l'industrie biopharmaceutique. Ces cellules ont besoin de conditions bien précises pour croître convenablement. Cette thèse se concentre sur l'étude de l'hydrodynamique dans les réacteurs à agitation orbitale (abrégé OSR pour l'anglais *orbitally shaken reactor*), un nouveau genre de bioréacteurs pour la culture des cellules mammifères. Les OSRs représentent une alternative peu coûteuse et simple d'utilisation aux réacteurs classiques tels que les *stirred tanks*. Il a été démontré que les OSRs peuvent fournir les conditions nécessaires aux cultures cellulaires à petite échelle mais de plus amples connaissances sont nécessaires pour exploiter tout leur potentiel et procéder à des cultures à grande échelle. Ce travail a pour but de mettre en lumière les principes régissant les OSRs au travers de la mécanique des fluides numérique.

Les OSRs ne sont remplis que partiellement par le milieu de culture liquide utilisé, l'espace restant étant occupé par de l'air. Lors de l'agitation, l'interface entre les deux phases bouge et prend différentes formes. Cette surface libre est au centre des simulations numériques d'OSRs: non seulement elle fait partie des inconnues puisque sa position doit être déterminée, mais elle est également porteuse de singularités. En particulier, la pression a une faible régularité dans le voisinage de l'interface ce qui réduit les performances des méthodes numériques si cette singularité n'est pas traitée de manière appropriée.

Ceci motive l'étude d'un problème elliptique muni d'une interface interne porteuse de discontinuités. Dans ce travail, nous concevons une méthode appelée SESIC pour résoudre ce genre de problèmes. Elle repose sur la connaissance des discontinuités a priori pour améliorer la précision dans le voisinage de l'interface en éliminant ces discontinuités. Nous démontrons que cette méthode permet d'obtenir des convergences optimales en norme H^1 et L^2. Les tests numériques montrent également qu'une convergence optimale en norm L^∞ peut être obtenue dans certains cas. L'utilisation d'intégrales régularisées est examinée avec comme perspective une simplification supplémentaire de la méthode. Si la largeur de la bande de régularisation est correctement définie, de bonnes approximations peuvent être obtenues, malgré une convergence plus faible.

Nous appliquons ensuite la méthodologie de la méthode SESIC à l'approximation des équations de Navier-Stokes à deux phases, ce qui revient à corriger la pression. Si une intégration adaptée à l'interface est utilisée, la densité et la viscosité peuvent être gardées discontinues à travers l'interface sans pour autant créer de vitesse parasite, comme démontré par nos tests numériques. Ce traitement précis des discontinuités permet de retenir toute la signification physique indépendamment de la taille du maillage. Nous portons également une

attention particulière aux conditions de bords qui doivent à la fois permettre le mouvement de l'interface et reproduire la friction des parois. Nous montrons qu'imposer une valeur nulle pour la composante normale de la vitesse donne les meilleurs résultats pour la condition de non-pénétration et que cette condition doit s'accompagner d'un terme correctif pour éviter l'apparition de courants parasites. Des conditions de type Robin sont utilisées pour les composantes tangentielles de la vitesse afin de se ramener à une vitesse nulle loin de la ligne de contact.
Nous effectuons des tests spécifiques pour déterminer la qualité de chacun des composants de notre méthode. Nous comparons, résultats à l'appui, notre méthode avec une méthode où la densité et la viscosité sont régularisées et montrons ainsi que le traitement précis de ces quantités physiques améliore la qualité des simulations. La scalabilité de la méthode est également examinée et les points problématiques sont mis en évidence.
Notre méthode est ensuite validée par rapport à des données expérimentales variées. Tout d'abord, des OSRs remplis de glycérine sont simulés et nous montrons que notre méthode reproduit précisément l'amplitude de la vague créée. Nous montrons que la sensibilité des résultats par rapport aux conditions de Robin est faible. Pour les OSRs remplis d'eau, nous reproduisons la forme de la vague pour différentes configurations, que ce soit une vague déferlante ou non, simple ou multiple. En particulier, une vague triple est reproduite dans les conditions expérimentale où elle est observée. Nous utilisons également des mesures effectuées par vélocimétrie laser pour valider plus fortement notre méthode. Les résultats obtenus sont ensuite utilisés pour évaluer les niveaux de contrainte hydrodynamique et les propriétés de mélange des différentes configurations, par rapport à la forme de l'interface.
Finalement, nous examinons la modélisation des cultures cellulaires par un système d'EDOs non-linéaires qui représentent l'évolution temporelle des principaux nutriments et déchets ainsi que de la population cellulaire. Le comportement de la culture est reproduit mais certains phénomènes restent inexpliqués. En particulier, notre modèle contient un déchet toxique dont la nature est discutée. Deux scénarios alternatifs sont proposés pour améliorer le modèle.

Mots-clés: Equations de Navier-Stokes; Ecoulement multi-phase à surface libre; Méthode des éléments finis; Level set; Modèle de croissance cellulaire.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Orbitally shaken reactor

Mammalian cell cultures are nowadays common in the biopharmaceutical industry for the production of therapeutic proteins. Indeed, the proteins produced by these cells are more suitable for usage by humans than those obtained from bacteria or yeasts. Therefore, efforts are being put to adapt existing culture technology to mammalian cells, which are fragile and require specific conditions to grow optimally.

Different systems to culture these cells are still in competition, evidencing that the ideal solution has not been found yet. Stirred tanks, that use an impeller to agitate the cells culture, are widely used but they present different disadvantages: the culture usually needs to be controled, air must be sparged into the tank and fragile cells can be damaged due to high stresses in the impeller area. Other systems, such as air lift reactors, use different principle to offer to the cells suitable conditions for growth. We refer to [95] for a comprehensive exposition of the different possibilities available.

In this context, orbitally shaken reactors (OSR) are a quite recent system which features a simple setup, see Fig. 1.1. Indeed, the only two pieces composing an OSR are the container and the shaker. The container, in general a plastic vessel or metallic structure holding a flexible bag, is partially filled with a liquid medium in which the cells are cultured, the remaining space being air. The shaker is the mechanical device lying below the container responsible for imparting an orbital motion to the container. The primary advantage of this technology is that it requires no invasive instrumentation such as impellers or air sparger. Thus, cultures can be performed in single-use bags, which simplifies the process of sterilization and reduces risks of contamination [95]. OSRs are frequently used for small to medium scales cultures, regimes for which they have been found to provide good conditions for cell growth [95, 104], despite their simplicity. The reason of theses performances is however not fully understood: the hydrodynamics can change drastically from one working configuration to the other and the way the moving surface between the liquid and the air phases influences the gas transfer is not clear.

Figure 1.1: An orbitally shaken reactor with both the container and the shaker displayed.

The current trend pushes the development of large scale OSRs, but, without more information on the principles governing the OSRs, optimal settings are difficult to find. Moreover, at large scale, tentative cultures are limited due to their prohibitive cost. Alternative investigation strategies have then to be setup.

The goal of this work is to show that computational fluid dynamics (CFD) can provide valuable informations about the principles governing OSRs. CFD simulations can describe in detail the different physical quantities involved, such as the velocity field of both fluids and their pressures, but they can also be used to compare qualitatively different configurations, e.g., with respect to the hydrodynamic stress or the mixing pattern. However, devising a method that can accurately predict the behavior of OSRs represents a difficult task.

1.2 Modeling of OSR

The study of OSRs yields a typical temporal multiscale problem. On one side, hydrodynamics phenomena, such as the formation of a wave, typically occur at a fast time scale, ranging from a few milliseconds to several minutes. On the other side, mammalian cell cultures take several days to fully develop and reach interesting states for the production of proteins. These two aspects are linked since the physical conditions of the medium influences the progress of the culture and, to a much lower extent, the content of the medium changes its viscosity.

1.2.1 The hydrodynamics

This work focuses mainly on the hydrodynamics aspects because they represent the specificity of OSRs, since the cells do not differ from those cultured in other types of reactors. We aim at better understanding the hydrodynamics by computing the velocity field within the container at different working configurations. The pressure will also be approximated in the whole reactor. Thanks to these results, we will be able to quantify the mixing behaviors, hydrodynamic stress levels, gas transfer rates and other quantities that would influence the culture of cells.

Mathematically, the fluid problem can be regarded as a two-phase problem where the gas and liquid phases are separated by a free surface whose position is an unknown of the problem. The computational domain Ω representing the container depends on time and is composed of two parts: Ω_{liquid} and Ω_{air} corresponding to the liquid and gas phases, respectively, separated by the free surface Γ. We assume that both phases are described as incompressible Newtonian

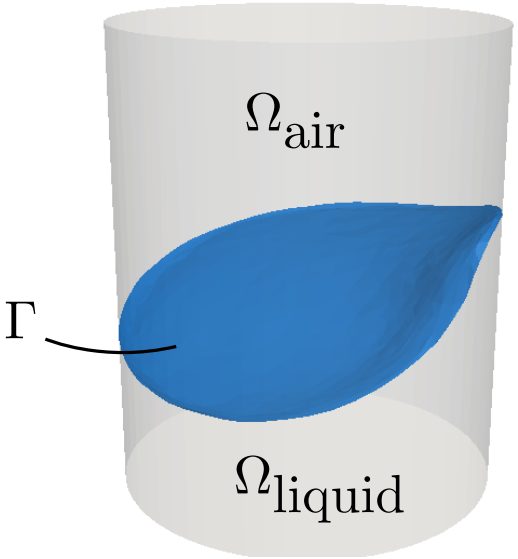

Figure 1.2: Illustration of the two subdomains Ω_{liquid} and Ω_{air} and of the interface Γ.

fluids. Thus, we can model them using the Navier-Stokes equations:

$$\begin{cases} \rho_a \left(\partial_t \mathbf{u}_a + (\mathbf{u}_a \cdot \nabla)\mathbf{u}_a\right) - \nabla \cdot \mathbf{T}(\mathbf{u}_a, p_a; \mu_a) &= \rho_a \mathbf{f} \\ \nabla \cdot \mathbf{u}_a &= 0 \end{cases} \quad \text{in } \Omega_{\text{air}} \tag{1.1}$$

$$\begin{cases} \rho_l \left(\partial_t \mathbf{u}_l + (\mathbf{u}_l \cdot \nabla)\mathbf{u}_l\right) - \nabla \cdot \mathbf{T}(\mathbf{u}_l, p_l; \mu_l) &= \rho_l \mathbf{f} \\ \nabla \cdot \mathbf{u}_l &= 0 \end{cases} \quad \text{in } \Omega_{\text{liquid}} \tag{1.2}$$

where $\mathbf{u}_a$ and $\mathbf{u}_l$ represent the velocity of the gas and the liquid, p_a and p_l their pressure, ρ_a

and ρ_l their density and μ_a and μ_l their viscosity. $\mathbf{T}(\mathbf{u}, p; \mu) = \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T) - pI$ is the Cauchy stress tensor and $\mathbf{f}_a$ and $\mathbf{f}_l$ represent the forces acting on the gas and the liquid. The interface $\Gamma = \Gamma(t)$ evolves according to the motion of the two phases.

The methods for modeling moving interfaces can be classified according to two categories:

- With *front tracking methods*, the mesh used for the approximation is kept conforming with the interface during the whole simulation. In this category, we find arbitrary Lagrangian-Eulerian methods (ALE, see, e.g., [92]) which deform the mesh according to a suitable map.
- On the contrary, *front capturing methods* keep the mesh fixed, without paying attention whether it conforms or not with the interface. Another quantity is in charge to retain the information of the location of the interface. The volume of fluid (VOF) method, introduced in [48], is an example of such techniques: A pseudo-concentration is added to the problem and its values, kept between 0 and 1, indicate the proportion of liquid present in a cell. Another widely used front capturing method is the level set method, that we describe in detail below.

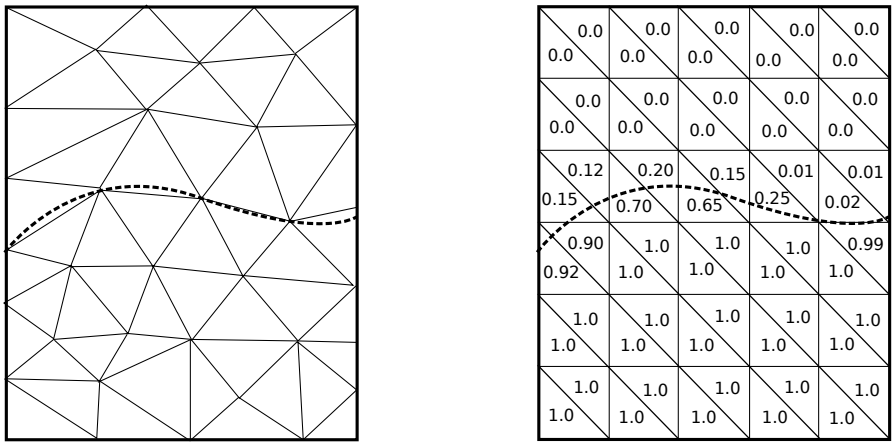

Figure 1.3: Representation of an interface (dashed curve) with a front tracking method (left) and the VOF method (right, numbers represent the values of the pseudo-concentration).

Besides this previous classification, we introduce another categorization of the different methods that have been proposed to solve the problem (1.1)-(1.2). The criterion here, is not the way the interface is handled, but the way the different physical properties are treated across the interface.

- The first option considers the equations (1.1) and (1.2) as a unique Navier-Stokes equation, where the density and the viscosity simply vary in space and time. Methodologies to approximate the solutions of single-phase Navier-Stokes are adapted. At the numerical level, the density and the viscosity are smoothed near the interface to obtain a stable

scheme and this is the reason why we call these methods *regularized methods*. The methods obtained are usually simple, easy to implement and benefit directly from the experience gathered with single-phase flows. Historically, they appeared first (see e.g. [94]).

- A second point of view, which has appeared recently, questions the well-foundedness of the regularized methods, arguing that they introduce unphysical fluid properties in the interface region that can create numerical aberrations [66]. The proposed solution is to treat the interface Γ sharply, i.e., without smoothing, thus maintaining the original physical meaning of the equations and, potentially, yielding better numerical accuracy. However, numerical schemes used for single phase flows cannot deal very well with sharp discontinuities, producing solutions heavily polluted by interface oscillations and unphysical velocities, see, e.g., [22, 28] and Sect. 4.2. Methods specially adapted to problems presenting an internal interface, that we call *sharp methods*, must be devised.

In this work we use a front capturing method since front tracking methods would lead to problems in some situations, e.g., when we have to represent a wave that breaks on the free surface. Moreover, it has been shown the ALE method might fail if no remeshing is considered, for OSR simulations even at low shaking velocities (see [40]).

We also use a sharp method to obtain the maximal possible accuracy. We need therefore a sharp representation of the interface and the most suitable front capturing method in such context is certainly the level set method. This method, first introduced in [73], consists in keeping trace of the interface Γ implicitly as the 0 level set of a continuous scalar function $\phi : \Omega \to \mathbb{R}$, i.e.

$$\Gamma = \left\{ \mathbf{x} \in \Omega \mid \phi(\mathbf{x}) = 0 \right\} . \tag{1.3}$$

The function ϕ is defined as the signed distance to the interface, the sign of ϕ retaining the information of the phase considered:

$$\Omega_{\text{liquid}} = \left\{ \mathbf{x} \in \Omega | \phi(\mathbf{x}) > 0 \right\} \qquad \Omega_{\text{air}} = \left\{ \mathbf{x} \in \Omega | \phi(\mathbf{x}) < 0 \right\} .$$

This method, like other front capturing schemes in general, can handle topological changes in a natural way. One of the advantages of the level set method with respect to the VOF is that it does not require any reconstruction of the interface to access the different geometrical quantities related to the interface itself, such as its normal and its curvature. However, it usually requires a redistancing procedure (see Sect. 3.5.2) and the conservation of the mass of the different phases is not ensured during the simulations (see Sect. 3.5.3).

Several sharp methods have been recently proposed. Front tracking is usually associated with sharp methods, since the mesh can deal with discontinuities thanks to its conformity. If front capturing methods are used, the background mesh does not suffice to capture the non-smoothness of the solution, thus one has to adapt the approximation scheme. In the

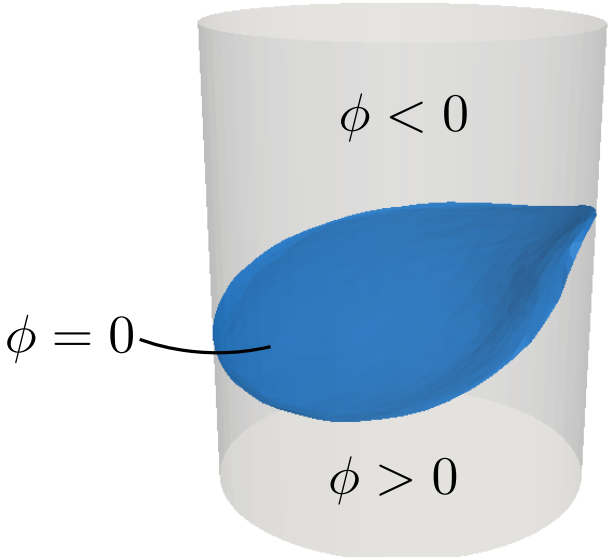

Figure 1.4: Definition of the different subdomains using the level set function ϕ.

finite element context, that we will consider in this work, the extended finite element method (XFEM) is the one that has probably been the most widely considered since its introduction for two-phase flows problem [17]. It consists in enriching the finite element spaces with basis functions that are adapted to the interface. It has also been proposed to modify the basis functions of the finite element space, e.g., in the immersed finite element method (IFEM) [62], also called intrinsic enrichment in the XFEM framework [37]. Sharp methods have also been applied to different problems with internal interfaces, such as problems with cracks in structures [8].

In this thesis, we will devise a numerical scheme of type "sharp" for free surface flows. To this aim, we will consider first a simpler problem, that we call elliptic internal discontinuity interface problem (IDIP): an open domain $\Omega \subset \mathbb{R}^N$ is divided in two disjoint subdomains Ω_1 and Ω_2, $\Omega_1 \cap \Omega_2 = \emptyset$, $\overline{\Omega_1} \cup \overline{\Omega_2} = \overline{\Omega}$, which are separated by the interface $\Gamma = \overline{\Omega_1} \cap \overline{\Omega_2}$. We want to find u solution of the following PDE

$$-\nabla \cdot (\beta \nabla u) = f \quad \text{in } \Omega \tag{1.4}$$

with homogeneous Dirichlet boundary conditions

$$u = 0 \quad \text{on } \partial\Omega \tag{1.5}$$

and interface conditions

$$[[u]] = g_d \quad \text{on } \Gamma \tag{1.6}$$

$$\left[\left[\beta \frac{\partial u}{\partial n}\right]\right] = g_n \quad \text{on } \Gamma \tag{1.7}$$

where we denote $[[g]] = g|_{\Omega_1} - g|_{\Omega_2}$. In chapter 2, we will study a suitable numerical method to solve this model problem, while in chapter 3 we will extend such method to represent correctly the free surface for the OSR problem.

1.2.2 The cell culture

The final aim of this work is to optimize the production of proteins by the cells cultured in the OSRs. The study of the hydrodynamics in the OSRs will allow us to select configurations that potentially lead to favorable conditions for the growth of the cells, e.g., with low shear stress or good mixing patterns. This is however not sufficient to fully determine the culture.

A first determinant factor for the culture is the cell line considered. Many types of cells are used for proteins production and all of them have different characteristics. The cells that we consider in this work are chinese hamster ovary (CHO) cells. These cells are of great interest for the production of therapeutic proteins, since they have a relatively simple genome for a mammalian. They also offer the advantage of producing proteins that are close to the ones produced by humans, so they are usually well accepted by the human organism. These are probably the reasons why this cell line was by far the line producing the highest number of biopharmaceuticals approved for therapies in 2010 [95].

These cells have specific needs to grow up to an interesting level for the production. Nutriments must be provided to keep the cells alive: glucose and glutamine are usually the main nutriments added to the medium for the production of energy, even if they are sometimes substitued by galactose [2] or glutamate [49], respectively. Next to that, sufficient oxygen must be provided, since CHO cells take most of their energy from aerobic respiration. Carbon dioxide, a by-product of the aerobic respiration must also be eliminated from the media. Other wastes, such as ammonia or lactate might slow down the culture when they accumulate [88, 59]. These substances also influence the pH of the medium, an important factor for the growth of the cells [99]. On top of the different species that must be present in the medium, other factors can greatly influence the cultures. The temperature is one of those, as a too low temperature slows down the process and high temperatures damage cells [35, 99]. The hydrodynamic stress, created by the bioreactor to enhance mixing, might also damage these cells, which are rather fragile [95].

The problem that we are interested in is the prediction of the evolution of the cell population in time. Since cells and nutriments are found only in the liquid phase, we are interested only in that subdomain and we consider the gas phase as having constantly the properties of the air. We reduce further the complexity of the problem by considering that the concentrations of the cells and of the different species in the liquid phase are uniformly distributed. Therefore,

we will only model the evolution in time of the different concentrations. In particular, we want to obtain the evolution of the culture during its four phases: the lag phase (a small phase starting the culture during which the culture seems to stagnate), the exponential phase, where the cells benefit from optimal conditions and grow quickly, the stationary phase, where the population reaches a maximum and stabilizes for a while, and the death phase, when the cell death rate exceeds the growth rate, because, e.g., of nutriments deplation or of toxic by-products accumulation.

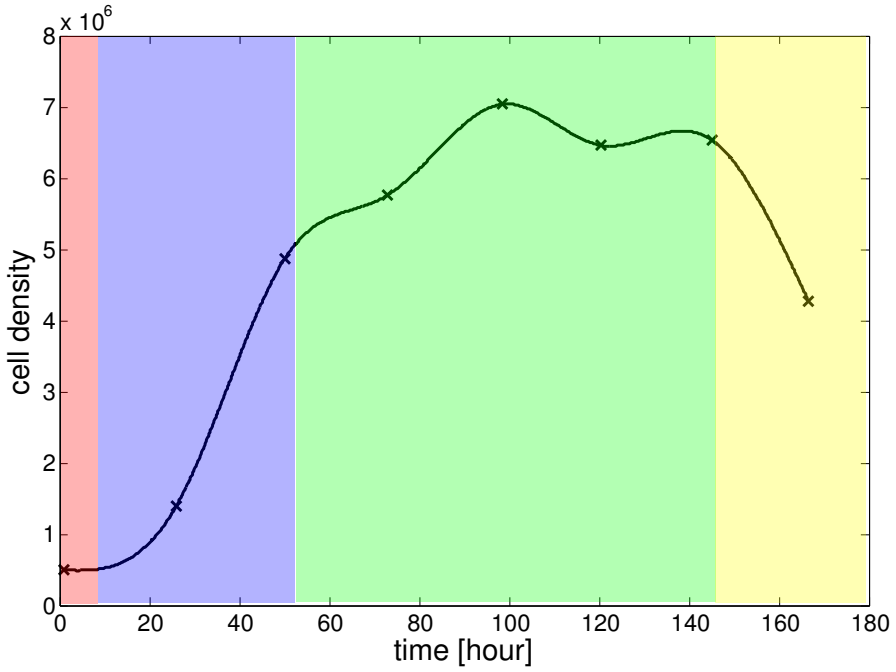

Figure 1.5: Typical evolution of a cell culture, featuring the 4 phases: in the order of appearance, the lag phase (red), the exponential growth (blue), the stationary phase (green) and the death (yellow). Crosses represent measures of a real CHO culture and the line a possible reconstruction using splines.

More precisely, we define $[X]$ the concentration of cells (in million of cells per liter), [Glc] the concentration of the glucose, [Gln] the concentration of glutamine, [Lac] the concentration of lactate (all in millimol per liter), etc. We want to find the evolution of these different components, $[X](t)$, $[\text{Glc}](t)$, $[\text{Gln}](t)$, $[\text{Lac}](t)$, etc. In a first step, we need to devise the model for the culture, describing the evolution of the different components with a set of possibly non-linear ODEs:

$$\begin{cases} \frac{d[X]}{dt} & = F_X([X],[\text{Glc}],[\text{Gln}],[\text{Lac}],\dots) \\ \frac{d[Glc]}{dt} & = F_{\text{Glc}}([X],[\text{Glc}],[\text{Gln}],[\text{Lac}],\dots) \\ \frac{d[Gln]}{dt} & = F_{\text{Gln}}([X],[\text{Glc}],[\text{Gln}],[\text{Lac}],\dots) \\ \frac{d[Lac]}{dt} & = F_{\text{Lac}}([X],[\text{Glc}],[\text{Gln}],[\text{Lac}],\dots) \\ \dots & \dots \end{cases} . \tag{1.8}$$

The different functions F_* contain all the informations needed to describe the evolution of a component, e.g., the rate of consumption of a certain nutriment by the cells or the inhibition of a reaction by a by-product. Informations about the hydrodynamics are also contained in the F_* functions. In particular, the $k_L a$, a quantity measuring the rate of gas transfer across the free surface, can be computed based on numerical simulations of the hydrodynamics. Moreover, we would like to link the system of ODEs (1.8) with the hydrodynamic stress. This issue will be discussed in Sect. 6.5.

1.3 Outline of the thesis

This paper is organized as follows: in chapter 2, we study the elliptic IDIP and we devise an efficient method for solving it. Theoretical analysis and numerical examples are also provided in that chapter. This method is then used in the context of free surface flows in chapter 3, where we detail the different numerical ingredients used for the approximation of the hydrodynamics in OSRs. Chapter 4 is dedicated to academic tests that investigate the reinitialization procedure, the apparition of spurious velocities, the boundary conditions and the parallelism. In chapter 5, we proceed with the comparison of results obtained numerically and experimentally. Several test cases are tackled and useful informations for the cell cultures are extracted for different regimes. Finally, chapter 6 focuses on the cell growth modeling: the model is developed on the basis of the metabolism of the cells and is tested on experimental data. The model is finally discussed on the basis of the results obtained.

2 The SESIC method for elliptic IDIP

2.1 Introduction

This section is focused on the numerical approximation of elliptic problems whose solution features discontinuities across interfaces internal to the computational domain. We consider a Poisson problem in two disjoint subdomains of the computational domain $\Omega \subset \mathbb{R}^N$ ($N = 1,2,3$) with jump conditions across the interface Γ separating the two subregions. The study of this problem will prove to be valuable for the approximation of the free surface flow problem, see Sect. 3.4.2. Indeed in the context of steady two phase flows, surface tension, which is a force localized on the interface [74], creates a jump in the pressure across the free surface [62]

$$[[p]] = \sigma\kappa$$

where σ is the surface tension coefficient and κ the curvature of the interface. Moreover, as investigated in Sect. 3.4, even when surface tension is not considered, jumps in the gradient of the pressure can appear due to the discontinuity of the external forces, e.g., gravitational effects, across the interface [62]:

$$\left[\left[\frac{\partial p}{\partial n}\right]\right] = [[\rho]]\mathbf{f}\cdot\mathbf{n}\,.$$

To obtain high accuracy simulations, these jumps must be captured, as evidenced in [22]. The same holds true for the Poisson problem with internal interface studied in this section: numerical methods taking into account the singularities at the interface are necessary for optimal error convergence. The link between the Poisson and the free surface flow problems can be even stronger in case splitting methods are used to discretize the Navier-Stokes equations, as for some of them, one solves a Poisson problem for the pressure [81].

We consider an open bounded domain Ω partitioned into the two non-overlapping subdomains $\Omega_1 = \{x \in \Omega |\ \phi(x) < 0\}$ and $\Omega_2 = \Omega \setminus \bar{\Omega}_1 = \{x \in \Omega |\ \phi(x) > 0\}$. See Fig. 2.1 for two possible instances. To approximate the interface (a point if $N = 1$, a line f $N = 2$ or a surface if $N = 3$), we use a level-set function $\phi : \Omega \to \mathbb{R}$. According to the classical level-set method (see, e.g., [72]

and [90]), ϕ is regarded as the signed distance function to the interface, whence Γ is defined by the equation $\phi = 0$. The mathematical formulation of our problem is as follows. We look for

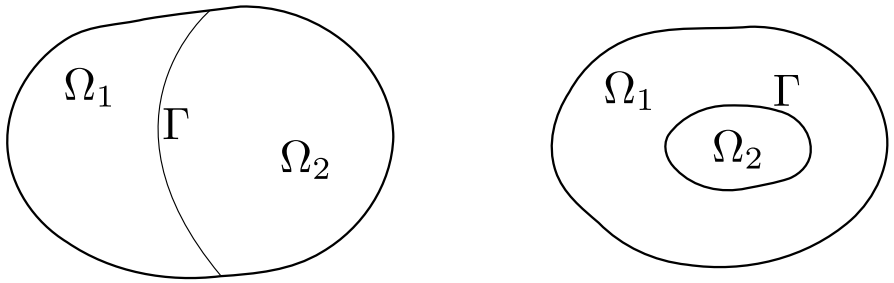

Figure 2.1: Two examples of partition of the domain Ω.

a function u in Ω that satisfies a Poisson problem in each subdomain:

$$-\nabla \cdot (\beta_i \nabla u_i) = f_i \quad \text{in } \Omega_i, \quad i = 1,2, \tag{2.1}$$

where $u_i = u_{|\Omega_i}$ and $\beta_i = 1$, with the following conditions on the jumps of the trace and of the normal derivatives across Γ:

$$[[u]]_\Gamma = u_{2|\Gamma} - u_{1|\Gamma} = g_d, \tag{2.2}$$

$$\left[\left[\beta \frac{\partial u}{\partial n}\right]\right]_\Gamma = \beta_2 \nabla u_2 \cdot n_2|_\Gamma + \beta_1 \nabla u_1 \cdot n_1|_\Gamma = g_n. \tag{2.3}$$

g_d and g_n are two assigned functions on Γ, while n_1 and n_2 are the unit normal vectors on Γ directed outwards of Ω_1 and Ω_2, respectively. Notice that $n_2 = -n_1$ on Γ. We assume $f_i \in L^2(\Omega_i)$, $g_d \in H_{00}^{1/2}(\Gamma)$ and $g_n \in L^2(\Gamma)$. For simplicity, we impose homogeneous Dirichlet boundary conditions on the boundary of the domain Ω: $u = 0$ on $\partial\Omega$. Remark that if $\partial\Omega \cap \Gamma \neq \emptyset$ (see Fig. 2.1, left), then the compatibility condition $g_d|_{\partial\Omega\cap\Gamma} = 0$ must hold.

Internal discontinuity interface problems (IDIPs) are of great interest since they are also encountered in other applications than free surface flows, such as crack modeling in structures [69, 3, 5], phase transitions [18, 54] and more in general any problem where several regions with different physical characteristics meet.

2.2 Review on existing methods for solving IDIPs

Applying a standard finite element discretization to problem (2.1)-(2.3) does not necessarily yield an optimal approximation. Indeed, the finite element method relies on regularities of the solution that cannot be expected for this problem even when $g_d = 0$. A standard finite element approximation would lead to suboptimal orders of convergence, with at most $h^{\frac{1}{2}}$ in the H^1 norm if $g_d = 0$, h being the size of the elements used for the discretization [60, 42]. This

suboptimal behavior is due to the inability of the approximation space to capture accurately the solution in the vicinity of the interface. Indeed, even the interpolation of the exact solution of the problem at hand yields suboptimal convergence, as illustrated on Fig. 2.2.

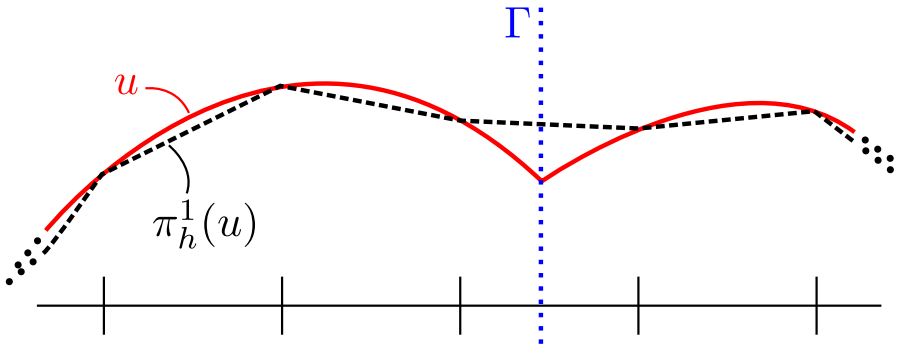

Figure 2.2: Schematic representation of the suboptimality of the Lagrange interpolation in case the mesh does not capture the interface.

Several methods have been proposed to treat the IDIP problem in the general case of $\beta_i \in \mathbb{R}$ for $i = 1,2$. We describe them briefly hereafter.

2.2.1 Methods with conforming meshes

A possible strategy to correctly approximate problem (2.1)-(2.3) is to build a mesh which captures the interface, in which case optimal orders of convergence can be obtained when the solution is regular enough and the mesh reproduces accurately the interface (see [60] for more details). This approach however may not be convenient. For instance, for a time dependent problem with a moving interface, conforming meshes have to be rebuilt at each time step, resulting in too expensive schemes in terms of computational cost. An alternative is to use front tracking methods (see Sect. 1.2.1).

The mesh could also be cut by the interface and only locally rebuilt as proposed in [70], but this could lead to highly deformed cells.

2.2.2 Methods for non-conforming meshes

Methods that do not require the reconstruction of a new mesh are in general preferable, in particular in terms of computational efficiency.

Instead of adapting the mesh to resolve the discontinuities of the solution, these methods rather adapt the approximation space to the interface. The usual Lagrangian spaces are enriched with suitable additional basis functions which can capture discontinuities or kinks. A recent extensive review of the different enrichment strategies and possible applications can be found in [37].

A first way to impose the interface conditions (2.2) and (2.3) is based on a variant of Nitsche's method, as proposed in [43]. In that work, the authors consider a finite element space composed by the restriction of the basis functions to either sides of the interface. In practice, the basis functions whose supports are crossed by the interface are doubled, each of the new functions being the restriction of the original basis function to one of the subdomains (see Fig.2.3 for an illustration in the $N = 1$ case).

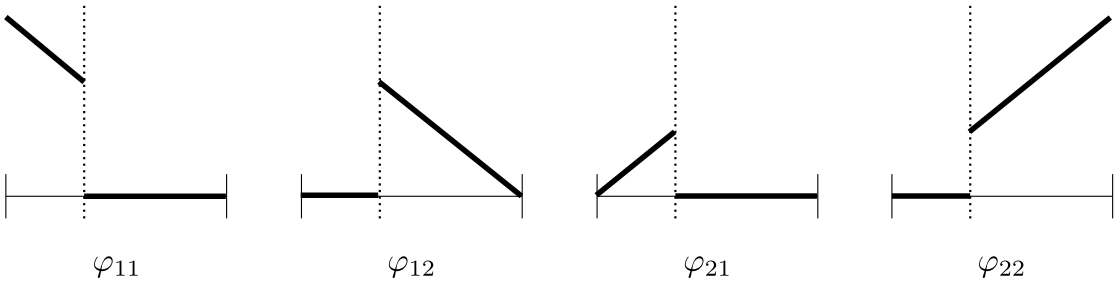

Figure 2.3: Representation of the basis functions used with Nitsche's method in the $N = 1$ case.

With the same enrichment, it was proposed to impose the interface condition (2.2) using Lagrange multipliers to be chosen in a suitable space [55]. In the $N = 1$ case, the solution is trivial, but it is more difficult for higher dimensions where oscillations can be observed in case the Lagrange multiplier space is too rich, in particular with the simplest choice, i.e. with piecewise linear functions on the interface mesh [54].

Finally, the immersed finite element method (IFEM) consists in modifying the basis functions so that they yield the right interface conditions (2.2) and (2.3), rather than adding new basis functions. However, in the 1-dimensional case, the IFEM method as formulated in [61] can be recovered by considering again the piecewise linear enrichment and imposing in a strong way the two interface conditions (2.2) and (2.3). In the case $\beta_1 = \beta_2 = 1$, that is of interest in this work, the IFEM simplifies. Indeed, in that case, the basis functions are not changed but the right hand side of the system is modified to take into account the jumps across the interface. The difference with the method developed hereafter is that the IFEM is rather difficult to generalize to the multidimensional case $N > 1$: the piecewise linear approximation must be given up [61] or the interface conditions have to be satisfied in a weaker sense [45]. Our method, on the contrary, extends naturally to any space dimension.

2.3 Weak formulation for the internal discontinuity interface problem

To derive a weak formulation of (2.1) - (2.3), we introduce two suitable liftings (also called extensions) $R_i g_d$ $(i = 1,2)$ of g_d in Ω_i so that the jump of $R_i g_d$ is g_d on Γ: $R_i g_d \in H^1_{\partial\Omega_i \setminus \Gamma}(\Omega_i) = \{v \in H^1(\Omega_i) | v = 0 \text{ on } \partial\Omega_i \setminus \Gamma\}$ $(i = 1,2)$ such that $[[R_i g_d]]_\Gamma = g_d$. The trace theorem (see [63])

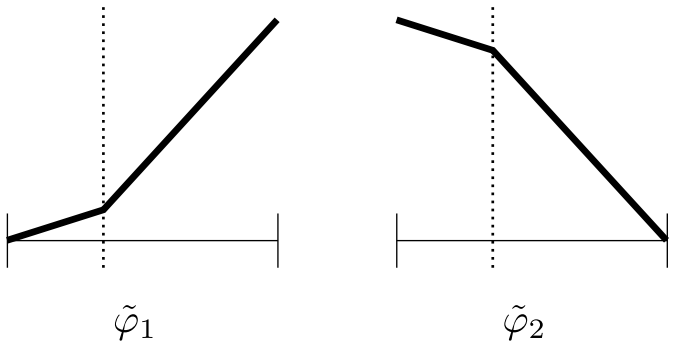

Figure 2.4: Schematic representation of the modified basis functions as defined by the IFEM.

guarantees that such a lifting operator exists. Then, we consider the splitting

$$u_i = \bar{u}_i + R_i g_d \quad \text{in } \Omega_i. \tag{2.4}$$

We denote $\bar{u} : \Omega \to \mathbb{R}$ such that $\bar{u}_i = \bar{u}|_{\Omega_i}$. The function $\bar{u}$ belongs to $H_0^1(\Omega)$. We consider a global test function $v \in H_0^1(\Omega)$ and its restrictions v_i on Ω_i. Then, on each domain, starting from (2.1), integrating by parts and exploiting the homogeneous Dirichlet boundary conditions on $\partial\Omega_i \cap \partial\Omega$, we obtain

$$\int_{\Omega_i} \nabla u_i \cdot \nabla v_i - \int_\Gamma \frac{\partial u_i}{\partial n_i} v_i = \int_{\Omega_i} f_i v_i, \quad i = 1,2. \tag{2.5}$$

Summing up the contributions of each subdomain and imposing the jump condition on the normal derivative (2.3) in a natural way, we obtain

$$\sum_{i=1}^{2} \int_{\Omega_i} \nabla u_i \cdot \nabla v_i - \int_\Gamma g_n v_i = \sum_{i=1}^{2} \int_{\Omega_i} f_i v_i \,. \tag{2.6}$$

Finally, using the decomposition (2.4), we obtain the weak form of problem (2.1)-(2.3): find $\bar{u} \in H_0^1(\Omega)$ such that $\forall\, v \in H_0^1(\Omega)$:

$$\int_\Omega \nabla \bar{u} \cdot \nabla v = \sum_{i=1}^{2} \int_{\Omega_i} f_i v_i - \sum_{i=1}^{2} \int_{\Omega_i} \nabla R_i g_d \cdot \nabla v_i + \int_\Gamma g_n v. \tag{2.7}$$

An alternative weak formulation of (2.1)-(2.3) was proposed by Huh and Sethian [52], by considering an additional lifting for the function g_n. More precisely, they define a function $S_i g_n \in H^1_{\partial\Omega_i \setminus \Gamma}(\Omega_i), i = 1,2$ such that $\left[\!\left[\frac{\partial S_i g_n}{\partial n} \right]\!\right]_\Gamma = g_n$ and, instead of (2.4), they consider the three-term splitting

$$u_i = \hat{u}_i + R_i g_d + S_i g_n \quad \text{in } \Omega_i. \tag{2.8}$$

Since we have assumed that $g_n \in L^2(\Gamma)$, the existence of $S_i g_n \in H^1_{\partial\Omega_i \setminus \Gamma}(\Omega_i)$ is ensured by the solution of the Neumann problem for the Laplace equation. The splitting (2.8) is similar to

the one proposed in [6] in a finite difference context. The two liftings $R_i g_d$ and $S_i g_n$ should ideally satisfy the following constraints:

$$[[R_i g_d]]_\Gamma = g_d \qquad \left[\left[\frac{\partial R_i g_d}{\partial n} \right]\right]_\Gamma = 0 \,, \tag{2.9}$$

$$[[S_i g_n]]_\Gamma = 0 \qquad \left[\left[\frac{\partial S_i g_n}{\partial n} \right]\right]_\Gamma = g_n \,, \tag{2.10}$$

in which case they would take into account the jump of the functions and that of the fluxes independently.

Using these lifting operators, the following weak form of problem (2.1)-(2.3) can be derived: find $\hat{u} \in H_0^1(\Omega)$ such that, $\forall v \in H_0^1(\Omega)$,

$$\int_\Omega \nabla \hat{u} \cdot \nabla v = \sum_{i=1}^{2} \int_{\Omega_i} f_i v_i - \sum_{i=1}^{2} \int_{\Omega_i} \nabla (R_i g_d + S_i g_n) \cdot \nabla v_i + \int_\Gamma g_n v \,. \tag{2.11}$$

Note that $\hat{u}$ is such that $\hat{u}|_{\Omega_i} = \hat{u}_i$ for $i = 1,2$.

We can remark that the bilinear form associated to both methods (2.7) and (2.11) is the classical Dirichlet formulation of the Poisson problem in $H_0^1(\Omega)$ (without internal discontinuity interface). This allows proving the well-posedness of the problem in a direct way by the Lax-Milgram lemma [79].

Both formulations (2.11) and (2.7) are equivalent from the mathematical point of view. Note that $\bar{u}$ and $\hat{u}$ have different regularity. In fact, the jump of normal derivatives of $\hat{u}$ is 0 across Γ whereas that of $\bar{u}$ isn't.

Obviously, their numerical approximation would yield different numerical solutions. We will discuss this issue in Sect. 2.6.1, while we focus now on the construction of the lifting operators R_i and S_i.

2.3.1 The continuous lifting operators

Formulations (2.7) and (2.11) require the knowledge of liftings of the jump conditions. If such liftings are already provided with the definition of the problem, they can be used without modifications and this section might be skipped. However, since in general these liftings are not known, we propose an approach to construct the liftings, which is based on the assumption that there exist two regular-enough scalar functions $\bar{g}_d$ and $\bar{g}_n$ in Ω such that $g_d = \bar{g}_d|_\Gamma$ and $g_n = \bar{g}_n|_\Gamma$. We will discuss this point in Sect. 2.3.2.

Thanks to the extensions $\bar{g}_d$ and $\bar{g}_n$, we are able to define liftings that satisfy conditions (2.9) and (2.10) exactly at the continuous level. For the sake of simplicity, we start with the lifting

for g_n which accounts for the jump in the normal derivative. From now on we assume that the level set function be such that $\phi \in C^1(\bar{\Omega})$. Consider the function

$$Sg_n = H(\phi)\phi\bar{g_n} \quad \text{in } \Omega, \tag{2.12}$$

where $H(\phi)$ is the Heaviside function of the domain $\Omega \setminus \Omega_1$:

$$H(\phi)(x) = \begin{cases} 1 & \text{if } \phi(x) \geq 0 \\ 0 & \text{if } \phi(x) < 0\,, \end{cases}$$

whence

$$Sg_n(x) = \begin{cases} \phi(x)\bar{g_n}(x) & \text{if } \phi(x) \geq 0 \\ 0 & \text{if } \phi(x) < 0\,. \end{cases} \tag{2.13}$$

Note that Sg_n is continuous across Γ (the latter being the 0-level set of ϕ), that is $[\![Sg_n]\!]_\Gamma = 0$. On the other hand,

$$\left[\!\left[\frac{\partial Sg_n}{\partial n}\right]\!\right]_\Gamma = \frac{\partial(\phi\bar{g_n})}{\partial n}|_\Gamma = \frac{\partial\phi}{\partial n}|_\Gamma\ \bar{g_n}|_\Gamma + \frac{\partial\bar{g_n}}{\partial n}|_\Gamma\ \phi|_\Gamma = g_n\,, \tag{2.14}$$

thus Sg_n is a lifting of g_n which satisfies both conditions (2.10). We denote $(Sg_n)|_{\Omega_i}$ by $S_i g_n$.

Now we need a lifting Rg_d for the function g_d which is discontinuous across Γ although featuring no jump in the normal derivative. We set

$$Rg_d = H(\phi)(\bar{g_d} - \phi\nabla\bar{g_d}\cdot\nabla\phi) \tag{2.15}$$

which can be expressed explicitly as

$$Rg_d(x) = \begin{cases} \bar{g_d}(x) - \phi(x)\nabla\bar{g_d}(x)\cdot\nabla\phi(x) & \text{if } \phi(x) \geq 0 \\ 0 & \text{if } \phi(x) < 0. \end{cases} \tag{2.16}$$

We can see that $[\![Rg_d]\!]_\Gamma = g_d$, whereas by a direct computation

$$\begin{aligned} \left[\!\left[\frac{\partial Rg_d}{\partial n}\right]\!\right]_\Gamma &= \frac{\partial(\bar{g_d} - \phi\nabla\bar{g_d}\cdot\nabla\phi)}{\partial n}|_\Gamma \\ &= \frac{\partial\bar{g_d}}{\partial n}|_\Gamma - \frac{\partial(\nabla\bar{g_d}\cdot\nabla\phi)}{\partial n}|_\Gamma\ \phi|_\Gamma - \frac{\partial\phi}{\partial n}|_\Gamma\ (\nabla\bar{g_d}\cdot\nabla\phi)|_\Gamma \\ &= \frac{\partial\bar{g_d}}{\partial n}|_\Gamma - \frac{\partial(\nabla\bar{g_d}\cdot\nabla\phi)}{\partial n}|_\Gamma\ 0 - 1\ (\nabla\bar{g_d}\cdot n)|_\Gamma \\ &= \frac{\partial\bar{g_d}}{\partial n}|_\Gamma - \frac{\partial\bar{g_d}}{\partial n}|_\Gamma = 0. \end{aligned} \tag{2.17}$$

Rg_d is therefore a lifting that satisfies conditions (2.9) (as before, we denote $R_i g_n = (Rg_n)|_{\Omega_i}$, $i = 1,2$).

Remark 2.3.1. *An alternative approach to that adopted before would consist in solving a suitable PDE in each triangle crossed by the interface to play the role of extension operators. To control both the trace and the normal derivative at the same time on the interface, we can solve a fourth order biharmonic problem in $K_i = K \cap \Omega_i$ for each K such that $K \cap \Gamma \neq \emptyset$. Precisely, the problem reads: find $R \in H^2(K_i)$ such that:*

$$\Delta^2 R = 0 \quad in\ K_i \tag{2.18}$$

$$R = g_d \quad on\ K_\Gamma \tag{2.19}$$

$$\frac{\partial R}{\partial n} = g_n \quad on\ K_\Gamma. \tag{2.20}$$

Suitable Dirichlet and Neumann boundary conditions are then prescribed on $\partial_i K_i$ to close the problem and to keep the support of R restricted to Ω_Γ (K_Γ and $\partial_i K_i$ are defined as in figure 2.5).

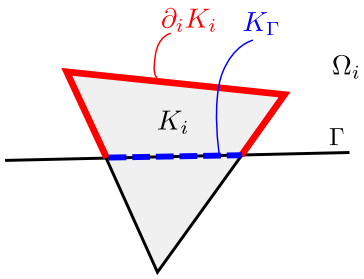

Figure 2.5: Illustration of the geometry of a triangle K cut by the interface Γ.

Example 2.3.1. *We consider a $1D$ example for the sake of clarity. The domain is $\Omega = (0,1)$ and the interface Γ is composed of the two points $x_1 = \pi^{-1}$ and $x_2 = 1 - \pi^{-1}$. The level set function is defined accordingly. We consider $\bar{g}_d(x) = \exp(2x)$ and $\bar{g}_n(x) = \sin(3x)$ so that $g_d(x_1) = \exp(2\pi^{-1})$, $g_d(x_2) = \exp(2(1-\pi^{-1}))$, $g_n(x_1) = \sin(3\pi^{-1})$ and $g_n(x_2) = \sin(3(1-\pi^{-1}))$. The explicit formula in this example are given by:*

$$Rg_d = \begin{cases} \exp(2x) + 2\exp(2x)(\pi^{-1} - x) & if\ x < \pi^{-1} \\ 0 & if\ \pi^{-1} < x < 1 - \pi^{-1} \\ \exp(2x) - 2\exp(2x)(x - 1 + \pi^{-1}) & if\ 1 - \pi^{-1} < x \end{cases}$$

$$Sg_n = \begin{cases} \sin(3x)(\pi^{-1} - x) & if\ x < \pi^{-1} \\ 0 & if\ \pi^{-1} < x < 1 - \pi^{-1} \\ \sin(3x)(x - 1 + \pi^{-1}) & if\ 1 - \pi^{-1} < x \end{cases}$$

The continuous liftings Rg_d and Sg_n are shown in figure 2.6.

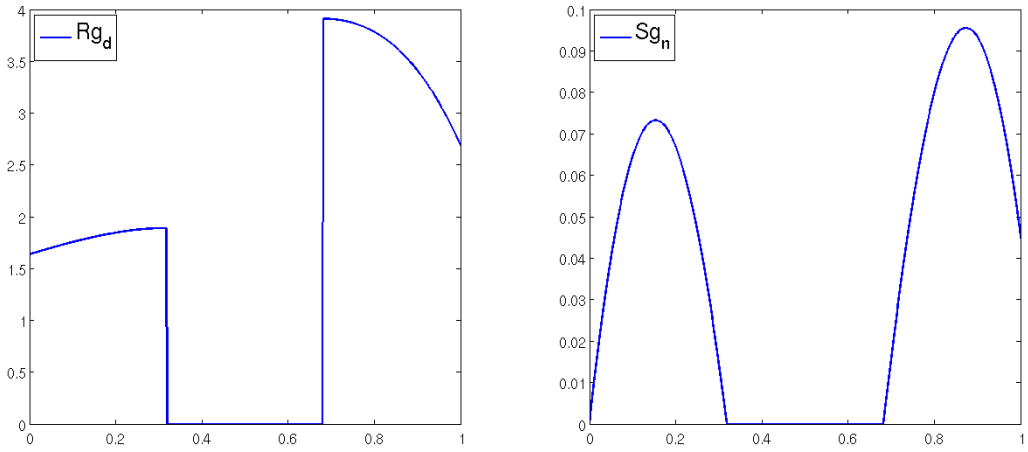

Figure 2.6: Continuous liftings obtained for the example 2.3.1: Rg_d (left) and Sg_n (right).

Example 2.3.2. *On Fig.2.7, liftings for a 2D example (detailed in Sect. 2.6.2) are represented. In this example, the level set function is given by*

$$\phi(x,y) = \begin{cases} \sqrt{x^2+(y-0.5)^2}-0.2 & \text{if } y > 0.5 \\ |x|-0.2 & \text{if } |y| \le 0.5 \\ \sqrt{x^2+(y+0.5)^2}-0.2 & \text{if } y < -0.5 \end{cases}$$

and the exact solution (from which the jumps can be computed) by

$$u(x,y) = \begin{cases} 1-\log(2\sqrt{x^2+y^2}) & \text{if } \phi(x,y) \ge 0 \\ 0 & \text{if } \phi(x,y) < 0\,. \end{cases} \tag{2.21}$$

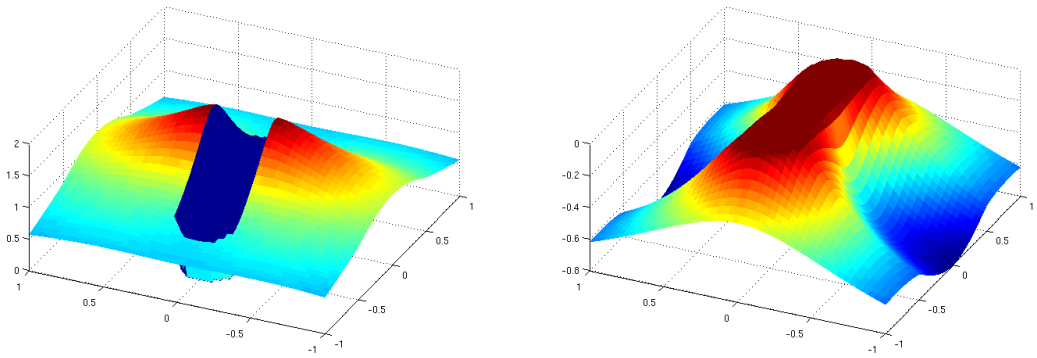

Figure 2.7: Continuous liftings Rg_d (left) and Sg_n (right) for a 2D problem.

2.3.2 Extending interface data

In section 2.3.1, we supposed that extensions of the interface data g_d and g_n are already given and that they enjoy suitable regularity. This assumption is more or less strong depending on the way the data g_d and g_n are provided. If they are given as functions on the whole domain Ω or as a finite element function, this assumption is fulfilled. In other cases, this can be regarded as a strong limitation, then we propose here solutions to overcome it.

A first solution is based on the resolution of a PDE of Hamilton-Jacobi type. To extend the interface data g, we solve to steady state the following hyperbolic equation [75]:

$$\partial_t \bar{g} + S(\phi)\nabla\phi \cdot \nabla \bar{g} = 0 \tag{2.22}$$

where $S(\phi)$ represents the signature function, that is

$$S(\phi) = \begin{cases} 1 & \text{if } \phi > 0 \\ 0 & \text{if } \phi = 0 \\ -1 & \text{if } \phi < 0 \end{cases}$$

We remark that at steady state, $\bar{g}$ represents the extension of g by a function that is constant in the normal direction, i.e. $\nabla\phi \cdot \nabla\bar{g} = 0$. If this method is used to extend g_d from the interface, the formula (2.15) simplifies then to

$$Rg_d = H(\phi)\bar{g}_d \,. \tag{2.23}$$

In 1D cases, this method leads to constant extensions in the vicinity of each interface point. In practice, the extension $\bar{g}_d$ and $\bar{g}_n$ are only required in those elements that are cut by the interface (see Sect. 2.4.1) and therefore, the equation (2.22) can be solved only in the region close to the interface. However, the numerical resolution of equation (2.22) is rather complicated [90] and we cannot ensure high regularity for the solution.

Another idea would be to consider a biharmonic problem so as to define $\bar{g}_d$ by $\bar{g}_d|_{\Omega_i} = \bar{g}_d^i$ where $\bar{g}_d^i$ solves the following PDE (supposing now $g_d \in H^{3/2}(\Gamma)$):

$$\begin{array}{ll} \Delta^2 \bar{g}_d^i = 0 & \text{in } \Omega_i \\ \bar{g}_d^i = g_d & \text{on } \Gamma \\ \bar{g}_d^i = 0 & \text{on } \partial\Omega_i \setminus \Gamma \\ \frac{\partial \bar{g}_d^i}{\partial n} = 0 & \text{on } \partial\Omega_i \,. \end{array}$$

Similarily, we can consider the following harmonic problems (supposing $g_n \in H^{1/2}(\Gamma)$)

$$\begin{array}{ll} \Delta \bar{g}_n^i = 0 & \text{in } \Omega_i \\ \bar{g}_n^i = g_n & \text{on } \Gamma \\ \bar{g}_n^i = 0 & \text{on } \partial\Omega_i \setminus \Gamma \end{array}$$

and we set $\bar{g}_n|_{\Omega_i} = \bar{g}_n^i$. The drawback of this approach is that, to ensure sufficient regularities on the solution of the biharmonic problem, we need to consider conforming finite elements, i.e. C^1, since a mixed formulation would yield only an H^1 approximation. Another possibility would be to resort to isogeometric analysis [50].

As we will see in Sect. 3.4.2, for our application we will not need such extention since our interface data are expressed in terms of the level set function and its derivative and therefore they can naturally be defined on the whole experimental domain.

2.4 Finite element approximation

In this section, we will address the numerical approximation of the problems introduced thus far, together with the introduction of the approximate lifting operators.

We consider a uniform triangulation τ_h of Ω made of elements K (intervals if $N = 1$, triangles if $N = 2$ or tetrahedra if $N = 3$). The interface Γ may intersect the elements K arbitrarily. As finite element space, we use the continuous $\mathbb{P}_1$ elements:

$$V_h = \{v_h \in H_0^1(\Omega) \cap C^0(\bar{\Omega}) : v_h|_K \in \mathbb{P}_1 \ \forall K \in \tau_h\}, \tag{2.24}$$

and we denote by $\{\Psi_j\}$ the basis functions of V_h.

The finite element approximation of (2.7) reads: find $\bar{u}_h \in V_h$ such that

$$\int_\Omega \nabla \bar{u}_h \cdot \nabla v_h = \int_\Gamma g_n v_h + \sum_{i=1}^{2} \int_{\Omega_i} f_i v_{hi} - \sum_{i=1}^{2} \int_{\Omega_i} \nabla R_i^h g_d \cdot \nabla v_{hi} \quad \forall v_h \in V_h, \tag{2.25}$$

while that of (2.11) becomes: find $\hat{u}_h \in V_h$ such that, for all $v_h \in V_h$,

$$\int_\Omega \nabla \hat{u}_h \cdot \nabla v_h = \int_\Gamma g_n v_h + \sum_{i=1}^{2} \int_{\Omega_i} f_i v_{hi} - \sum_{i=1}^{2} \int_{\Omega_i} \nabla (R_{hi} g_d + S_{hi} g_n) \cdot \nabla v_{hi} \, . \tag{2.26}$$

2.4.1 Discrete lifting operators

We introduce now suitable finite element approximations of the continuous liftings Rg_d and Sg_n. At the discrete level, we would like to have liftings with minimal support around the interface. Ideally, only the cells crossed by the interface ought be used in order to keep the computational cost of the finite element approximation as low as possible. This is why the knowledge of the extensions $\bar{g}_d$ and $\bar{g}_n$ will be required only in those neighbooring cells.

Let $\pi_h^1 : C^0(\bar{\Omega}) \to V_h$ be the classical finite element interpolant operator

$$\pi_h^1(v) = \sum_j v(x_j)\Psi_j \, , \tag{2.27}$$

i.e. $\pi_h^1(v)$ is the unique function in V_h which takes the same values of v at all finite element nodes x_j, Ψ_j being the characteristic basis function associated with x_j, that is $\Psi_j \in V_h$: $\Psi_j(x_i) = \delta_{ij}\ \forall i, j$ (see [82]).

Remark that both liftings Rg_d and Sg_n that we have defined at the continuous level are the product of the Heaviside function by a continuous function. For all $t \in C^0(\bar{\Omega})$, define $T = H(\phi)t$ and then

$$\Pi_h^1(T)(x) = \begin{cases} \pi_h^1(t)(x) & \text{if } \phi(x) \geq 0 \\ 0 & \text{if } \phi(x) < 0. \end{cases} \tag{2.28}$$

Note that $\Pi_h^1(T) = \pi_h^1(t)H(\phi)$. We define then the discrete liftings $R_h^{glo}g_d = \Pi_h^1(Rg_d)$ and $S_h^{glo}g_n = \Pi_h^1(Sg_n)$. The index glo stands for *global* and it indicates that these functions are defined on the global domain Ω.

To guarantee that $Rg_d \in C^0(\bar{\Omega})$, we have to require a higher regularity for $\bar{g}_d$ than in Sect. 2.3.2. Indeed, if now $\bar{g}_d \in C^1(\bar{\Omega}) \cap H^2(\Omega)$, then $Rg_d \in C^0(\bar{\Omega}) \cap H^1(\Omega)$ so that $R_h^{glo}g_d$ is well defined and $R_i g_d,\ R_{ih} g_d \in H^1_{\partial\Omega_i \setminus \Gamma}(\Omega_i)$. Moreover, to ensure that $Sg_n \in C^0(\bar{\Omega})$, we now consider $\bar{g}_n \in C^0(\bar{\Omega}) \cap H^1(\Omega)$ so that $S_h^{glo}g_n$ is well-defined. Under these assumptions both formulations (2.11) and (2.26) are well-posed.

The regularity required on $\bar{g}_d$ is very strong. In case $\bar{g}_d$ does not satisfy it, one can resort to the procedure described in Sect. 2.3.2, which will lower the regularity requirement to $\bar{g}_d \in C^0(\bar{\Omega}) \cap H^1(\Omega)$. The above regularity assumptions might be further weakened if other types of interpolations (e.g., Clément interpolation [82] or Scott-Zhang interpolation [11]) were used instead of the Lagrangian one. Suitable projections (e.g. with respect to the L^2 or H^1 scalar product) might also be used in case of lower regularity.

Example 2.4.1. *With the same settings of Example 2.3.1, we perform the interpolation on a mesh with* 5 *intervals using* $\mathbb{P}_1$ *finite elements. The resulting liftings are shown in figure 2.8.*

To reduce the computational cost induced by the fact that our liftings have global support, we introduce a region Ω_Γ of width h around the interface Γ (see Fig. 2.9) and we modify $R_h^{glo}g_d$ and $S_h^{glo}g_n$ so that the support of the modified functions be reduced to Ω_Γ. Notice that Ω_Γ corresponds to the strip of width h formed by those triangles that intersect the interface.

We use the abstract notation T_h^{glo} to identify either $R_h^{glo}g_d$ or $S_h^{glo}g_n$. We can express T_h^{glo} on each side of the interface using the finite element basis:

$$T_h^{glo}|_K(x) = \begin{cases} \sum_i \alpha_i \Psi_i(x) & \text{if } \phi(x) \geq 0 \\ 0 & \text{if } \phi(x) < 0. \end{cases} \tag{2.29}$$

Adding any arbitrary function from the finite element space on both sides of the interface does not change the jump of this lifting. Using the notation $\phi_i = \phi(x_i)$ and $\sum_{\phi_i \geq 0}$ to indicate

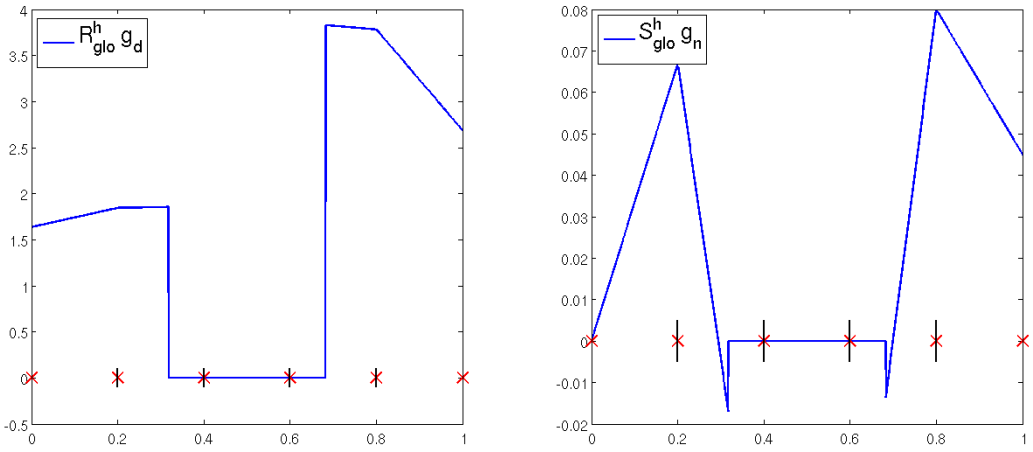

Figure 2.8: Discrete global liftings $R_h^{glo} g_d$ (left) and $S_h^{glo} g_n$ (right). The crosses show the location of the degrees of freedom.

$\sum_{\{i:\phi_i \geq 0\}}$ (analogously, we denote $\sum_{\phi_i<0}$), we can define a new lifting T_h as

$$T_h|_K(x) = \begin{cases} \sum_{\phi_i<0} \alpha_i \Psi_i(x) & \text{if } \phi(x) \geq 0 \\ -\sum_{\phi_i \geq 0} \alpha_i \Psi_i(x) & \text{if } \phi(x) < 0, \end{cases} . \tag{2.30}$$

By definition, the support of T_h is Ω_Γ and it is actually 0 on $\partial\Omega_\Gamma$ and it is then extended by zero outside Ω_Γ. Applying this procedure to $R_h^{glo} g_d$ and $S_h^{glo} g_n$ we obtain the liftings $R_h g_d$ and $S_h g_n$ that fulfill all our requirements.

We can now give the explicit expression of the two liftings:

$$R_h g_d|_K(x) = \begin{cases} \sum_{\phi_i<0} (\bar{g}_d(x_i) - \nabla \bar{g}_d(x_i) \cdot \nabla\phi(x_i)\, \phi(x_i)) \Psi_i(x) & \text{if } \phi(x) \geq 0 \\ -\sum_{\phi_i \geq 0} (\bar{g}_d(x_i) - \nabla \bar{g}_d(x_i) \cdot \nabla\phi(x_i)\, \phi(x_i)) \Psi_i(x) & \text{if } \phi(x) < 0, \end{cases} \tag{2.31}$$

$$S_h g_n|_K(x) = \begin{cases} \sum_{\phi_i<0} (\bar{g}_n(x_i)\, \phi(x_i)) \Psi_i(x) & \text{if } \phi(x) \geq 0 \\ -\sum_{\phi_i \geq 0} (\bar{g}_n(x_i)\, \phi(x_i)) \Psi_i(x) & \text{if } \phi(x) < 0. \end{cases} \tag{2.32}$$

In case the interface coincides with the boundary of an element, the formulas (2.31) and (2.32) remain valid, in the sense that (2.9) and (2.10) are still approximated in a suitable way. In such a case, the correction will be taken into account only on those elements belonging to the subdomain characterized by negative values of the level set function.

Example 2.4.2. *In figure 2.10 we show the new liftings $R_h g_g$ and $S_h g_n$ of reduced support corresponding to the functions of the example 2.3.1 using $\mathbb{P}_1$ polynomials.*

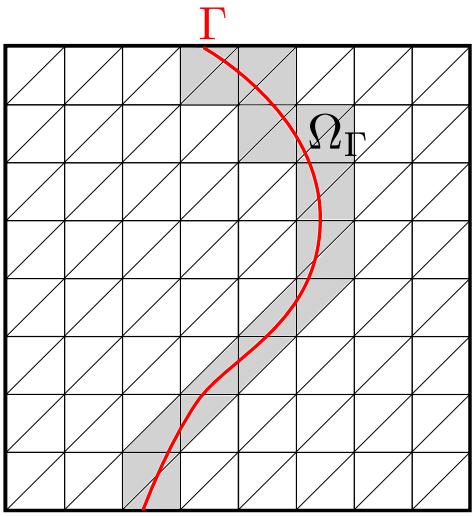

Figure 2.9: Illustration of Ω_Γ in a 2D case.

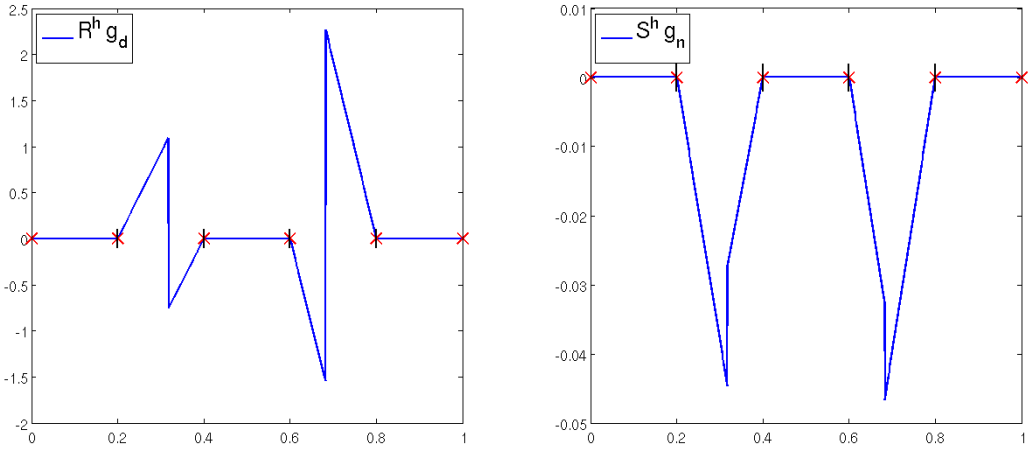

Figure 2.10: Liftings after the support reduction: $R_h g_d$ (left) and $S_h g_n$ (right). The crosses show the location of the degrees of freedom. These liftings should be compared to those in Fig. 2.8

Remark 2.4.1. *Remark that, thanks to the reduction of the support that we have performed, the extensions $\bar{g}_d$ and $\bar{g}_n$ of the interface data g_d and g_n have to be known only on Ω_Γ. This potentially reduces the cost of an extension procedure, e.g. the one described in Sect. 2.3.2.*

2.4.2 The SESIC method

The SESIC (Simplified Exact Subgrid Interface Correction) method that we propose is obtained by using the discrete lifting operators (2.31) and (2.32) in the context of the weak formulation (2.26). The only ingredient that remains to be detailed is the numerical integration formula that will be used to compute the new terms in the weak formulation (2.26). More precisely, we have to perform one integral on the interface Γ of a continuous function and two integrals over Ω of possibly discontinuous functions (indeed, both $\nabla R_h g_d$ and $\nabla S_h g_n$ might be discontinuous across Γ at the discrete level). We propose two different methods for the integration.

Two-side integration

The first method consists in building quadrature rules that take into account the interface. A possible way to integrate singular functions of type

$$\int_\Gamma f = \int_\Omega \delta_\Gamma f$$

is to reconstruct the interface Γ explicitly and then to use on it a $(N-1)$-dimensional quadrature rule. If $N = 1$, the interface reduces to a point and the integration requires only to evaluate f at a given point. If $N = 2$, the elements are triangles and then the interface in a single triangle is a segment in the case of a piecewise linear approximation. To apply a suitable integration rule on this segment, we need to compute the intersections of Γ with the edges of the triangle.

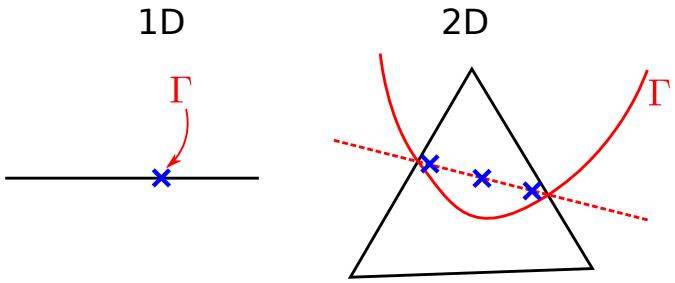

Figure 2.11: Illustration of the methods used for the computation of the line integral in 1D and 2D.

On the other hand, to integrate discontinuous functions like

$$\int_\Omega f_1 + H(\phi) f_2,$$

we define a quadrature rule for an element crossed by Γ considering a quadrature rule on the polygons on each side of the interface. More precisely, if $N = 1$, we compute the location of the interface then we combine a quadrature rule for segments on each side of the interface. When $N = 2$, the triangles crossed by Γ are split into a triangle and a quadrilateral. To integrate

discontinuous functions, we combine then a quadrature rule for triangles and a quadrature rule for quadrilaterals.

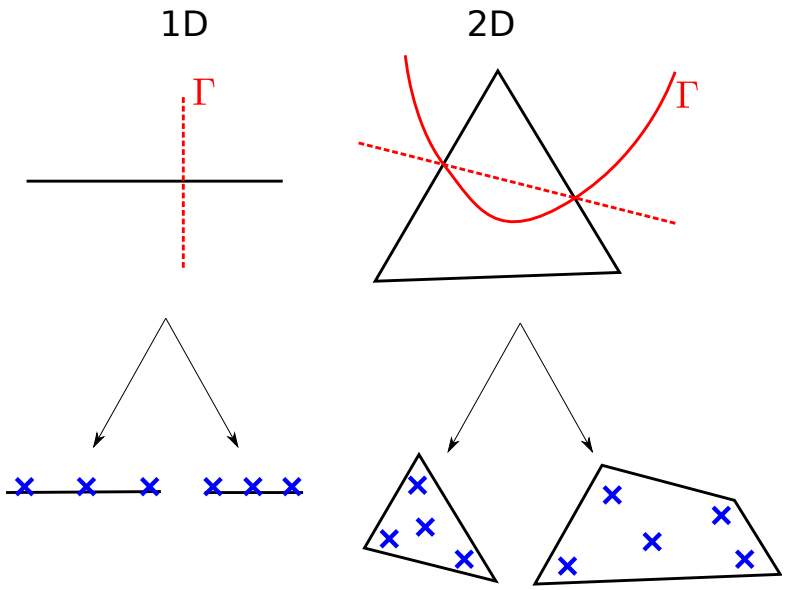

Figure 2.12: Illustration of the methods used for the computation of the discontinuous functions in 1D and 2D.

Similar methods are available for 3D simulations, however they lead to complicated schemes where many different cases have to be distinguished depending on the way the interface cuts the tetrahedra. Thus we propose an alternative method that is simpler and more suitable for higher space dimensions or for higher polynomial degrees.

Integration of regularized functions

The idea, that can be adopted in any space dimension, is to approximate singular or discontinuous integrands by smooth functions. For example, we make the following approximation:

$$\int_\Gamma g_n v = \int_\Omega \bar{g}_n v \delta_\Gamma \cong \int_\Omega \bar{g}_n v \delta_w$$

where δ_w is an approximation of δ_Γ whose support is limited to a band of width w around Γ. This method is quite widely used, even if, often, there is no real control on the error produced. We refer to [97] for the error analysis of the regularization step, in which two errors are highlighted:

- the error (called “analytical error” in [97]) produced by the introduction of the regulariz-

ing function:

$$\left| \int_\Gamma g_n v - \int_\Omega \bar{g}_n v \delta_w \right| ;$$

- the quadrature error coming from the inexact integration of the regularized integrand.

The usual procedure is to take w as a given quantity of cells, meaning that w is proportional to h. However, the approach that we adopt in this paper is to use *a width w that is proportional to* $\sqrt{h}$ (a similar width has been suggested in [62] in another context). Our choice is motivated by the following considerations. First of all, our function δ_w must have the form $\delta_w(d) = \frac{1}{w}\hat{\delta}(d/w)$ (prolongated by 0 outside the band of width w) where $\hat{\delta}$ is a function that does not depend on w, the factor $\frac{1}{w}$ making the weight of δ_w constant with respect to w, and d is the distance to the interface, that is $d(x) = \phi(x)$.

- The analytical error is then proportional to w^β, where β can be computed using the properties (vanishing moments) of $\hat{\delta}$ [97].

- If the integrand $\bar{g}_n v \delta_w$ is a function of $C^m(\Omega)$ and the quadrature rule (based on the finite element mesh, with typical size h) has a degree of exactness $m-1$ (see [80]), the quadrature error is proportional to $h^m ||(g_n v \delta_w)^{(m)}||_{L^\infty(\Omega)}$. However, $\delta_w^{(m)}$ scales like $w^{-(m+1)}$. Therefore, the quadrature error is dominated by $h^m w^{-(m+1)}$.

Based on these arguments, we derive that, by choosing $w = h$ we cannot ensure that the quadrature error will vanish when $h \to 0$. Indeed, with $w = h$, the number of quadrature points in the band of width w is constant while the function δ_w is becoming steeper to conserve the mass. Our choice of $w = c\sqrt{hL}$ (where $L = \text{diam}(\Omega)$, which we use to obtain correct scaling and c is a dimensionless constant to be chosen) leads to control the analytical error by $\mathcal{O}(h^{\beta/2})$ and the quadrature error by $\mathcal{O}(h^{(m-1)/2})$. We can then fully control the decay rate of the overall error by choosing the appropriate $\hat{\delta}$ function.

If we look for second order accuracy, building $\hat{\delta}$ with 3 vanishing moments (then $\beta = 4$, see [97]) and 5 continuous derivatives would be sufficient. By looking for the polynomial function with the smallest degree featuring these properties, we end up with:

$$\hat{\delta}(d) = \frac{6435}{8192}(3 - 35d^2 + 147d^4 - 315d^6 + 385d^8 - 273d^{10} + 105d^{12} - 17d^{14}) \tag{2.33}$$

A representation of this function is given in Fig. 2.13.

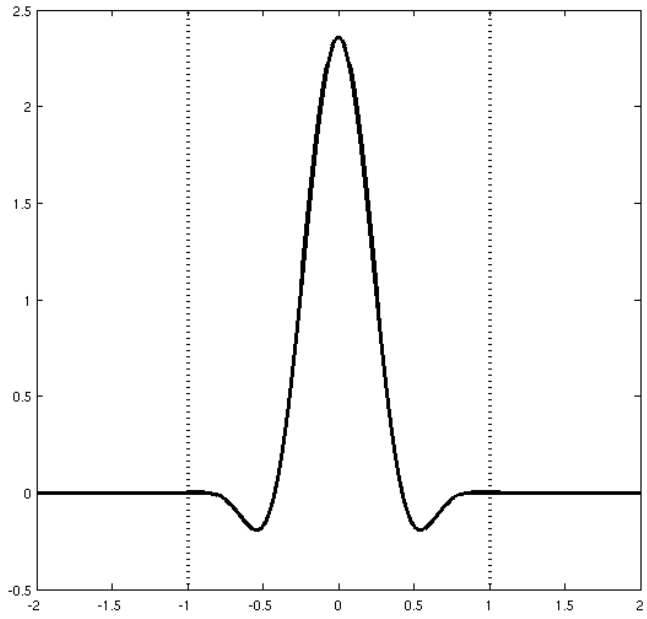

Figure 2.13: Plot of the function $\hat{\delta}(d)$

The same approach can be applied for the Heaviside function to integrate discontinuous integrand across the interface. We used for our tests the regularized Heaviside function:

$$\hat{H}(d) = \int_{-1}^{d} \hat{\delta}(\xi)\, d\xi.$$

To illustrate our choice for the integration, we consider 3 examples in 2D, where the domain is the square $\Omega = (-1,1)^2$ and the triangulation is made by triangles whose typical size is denoted by h. All the integrations are performed using a Dunavant quadrature rule of degree 4 in all the triangles of the mesh [29]. For each example, we will use three different methods to evaluate a line integral on the 0 level set of the function ϕ:

- **Method A:** The bandwith is set to $w = 2h$ and we take $\hat{\delta}(d) = (1+\cos(\pi d))/2$.
- **Method B:** The bandwith is set to $w = 2h$ and $\hat{\delta}$ is defined as in (2.33).
- **Method C:** The bandwith is set to $w = \sqrt{h}/2$ and $\hat{\delta}$ is again as in (2.33).

Example 2.4.3. *The first example consists in simply calculating the length of a circle. The level set function is defined as* $\phi(x,y) = \sqrt{x^2+y^2} - 0.5$. *In this example, the error is due only to the quadrature error. Figure 2.14 shows the magnitude of the error as function of the mesh size* h. *We can remark that with methods A and B the convergence quickly slows down to order of at most* 1, *while method C yields a convergence rate of order* 4.

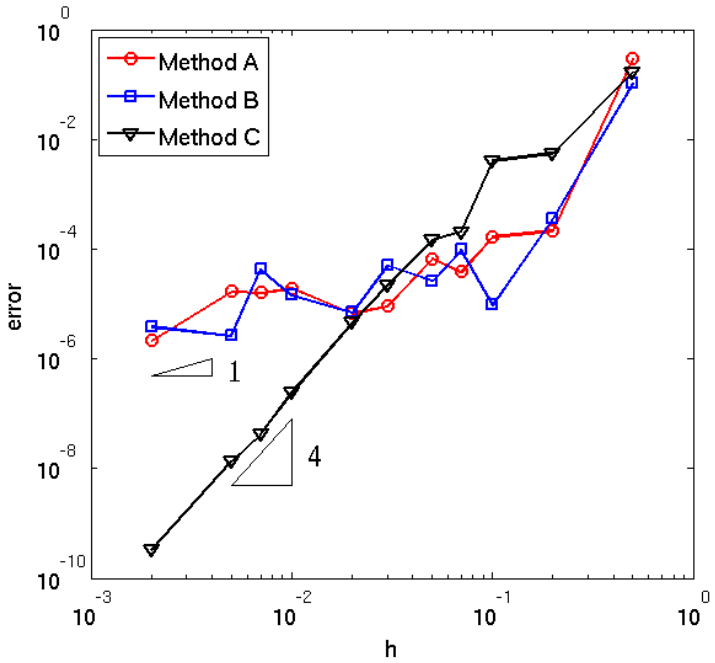

Figure 2.14: Results of example 2.4.3.

Example 2.4.4. *The second example consists in integrating on a quarter of circle the function* $f(x,y) = (y+1)\exp(x+1)$. *The level set function is* $\phi(x,y) = \sqrt{(x+1)^2+(y+1)^2} - 1.5$. *Figure 2.15 shows the results obtained. We can see that the methods A and B give rise to a good convergence for coarse meshes, but the convergence slows down at a level comparable to the one observed in example 2.4.3. Method C is more robust as it yields again a convergence rate close to* 2 *for the whole range of meshes tested.*

Example 2.4.5. *The last example consists again in computing the length of a curve. However, the curve that we consider has just a* C^0 *regularity (see Fig. 2.16 left). As shown in Fig. 2.16 (right), the convergence rates for the methods A, B and C are lower than in the previous examples. We can also remark that method C yields a slower convergence in this example.*

From these examples, we can see that in case the level set function has a lower regularity than $C^1(\bar{\Omega})$, method C might underperform with respect to methods A and B. Indeed, when the regularity is low, the integration error seems to be bounded by a term of the form w^α where $\alpha > 0$ is relatively small (around 0.5 in example 2.4.5). In such a case, choosing w proportional to $\sqrt{h}$ might give worse results (convergence in $h^{0.25}$) than if w is proportional to h (convergence in $h^{0.5}$). Similar remarks hold for 3D cases.

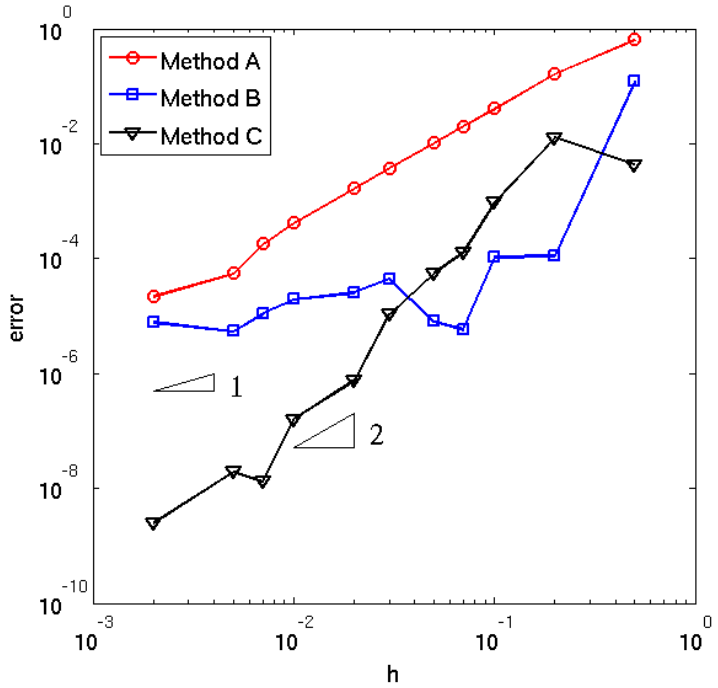

Figure 2.15: Results of example 2.4.4.

2.4.3 On the choice of the discrete lifting operator

In the previous section 2.4.1, we proposed a particular construction of the lifting operators that is interesting as it does not require to reconstruct the interface. We shall demonstrate here that the solution u_h obtained using the SESIC method is actually independent of this construction to a certain extent.

Suppose that we consider a suitable discrete lifting $\tilde{T}_h = \tilde{T}_h(g_d, g_n)$ such that $R_h g_d + S_h g_n - \tilde{T}_h \in V_h$. As a consequence

- $\tilde{T}_h$ is piecewise linear is each subdomain Ω_i, $i = 1,2$;
- the interface conditions are satisfied in the same way as with our construction in the sense that they satisfy the same jumps across the interface at the discrete level.

Using the discrete formulation of the SESIC method (2.26), we denote by $\hat{w}_h \in V_h$ the solution obtained considering the lifting $\tilde{T}_h$, i.e., for all $v_h \in V_h$,

$$\int_\Omega \nabla \hat{w}_h \cdot \nabla v_h = \int_\Gamma g_n v_h + \sum_{i=1}^{2} \int_{\Omega_i} f_i v_{hi} - \sum_{i=1}^{2} \int_{\Omega_i} \nabla \tilde{T}_h \cdot \nabla v_{hi} \,. \tag{2.34}$$

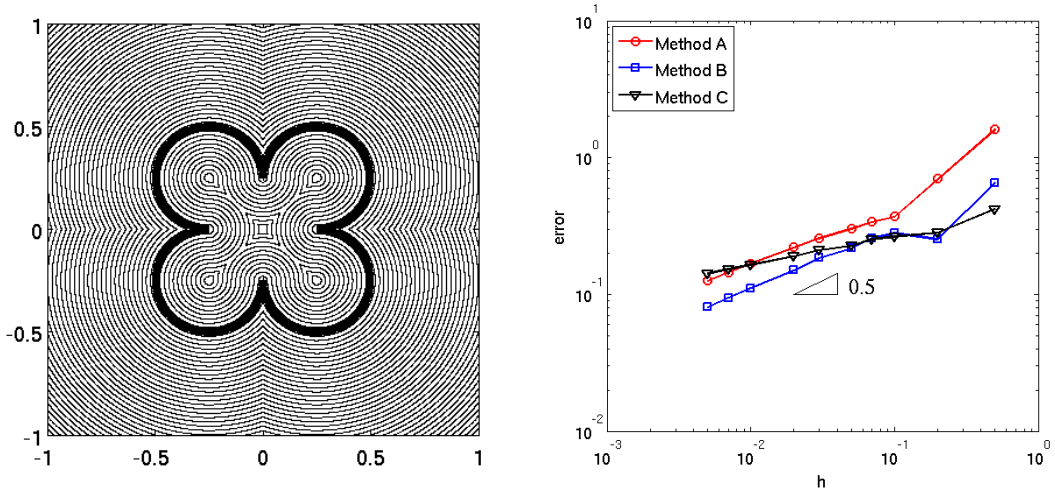

Figure 2.16: Representation of the isocurves of the level set function (left, Γ in bold) and magnitude of the error on the computation of the length of the curve.

We now prove that $w_h = \hat{w}_h + \tilde{T}_h$ coincides with u_h. Substracting (2.34) from (2.26), we obtain, for all $v_h \in V_h$,

$$\int_\Omega \nabla(u_h - w_h) \cdot \nabla v_h = 0\,. \tag{2.35}$$

We have $u_h - w_h = (\hat{u}_h + Rg_d + Sg_n) - (\hat{w}_h + \tilde{T}_h) \in V_h$ as by definition $\hat{u}_h \in V_h$, $\hat{w}_h \in V_h$ and $R_h g_d + S_h g_n - \tilde{T}_h \in V_h$ by assumption. Since the Galerkin approximation of the Laplace problem is well-posed on V_h, from (2.35) it follows that $u_h - w_h = 0$.

In view of this result, we can now comment on the case of 1D problems. As already remarked, using the extension of the interface data proposed in Sect. 2.3.2, $\bar{g}_d$ and $\bar{g}_n$ are constant in the vicinity of the interface (that might be composed of several points). Thanks to the definitions (2.12) and (2.23) and to the fact that the level set function ϕ is piecewise linear in 1D, Rg_d and Sg_n are piecewise linear in the neighborhood of the interface. Therefore, the interpolation step (2.28) leaves Rg_d and Sg_n unchanged and the jump conditions (2.9) and (2.10) are satisfied exactly. This implies that in 1D, any construction of the liftings using piecewise linear functions and satisfying exactly the interface conditions yields the same solution as the SESIC method.

2.4.4 The ESIC method

As stated before, the SESIC was inspired by the exact subgrid interface correction (ESIC) method first proposed in [52]. For the sake of comparison, let us recall the principle of the

ESIC method and emphasize the differences with the new SESIC method.

The two methods are built on different weak formulations of the given problem (2.1)-(2.3). SESIC stems from the weak form (2.26) whereas in order to get rid of the line integral in (2.26), Huh and Sethian perform a counter integration by parts yielding:

$$\int_\Gamma g_n v_h - \sum_{i=1}^{2} \int_{\Omega_i} \nabla S_i g_n \cdot \nabla v_{hi} = \sum_{i=1}^{2} \sum_K \int_{K\cap\Omega_i} \Delta S_i g_n v_{hi} - \sum_{i=1}^{2} \sum_K \int_{\partial K} \frac{\partial S_i g_n}{\partial n} v_{hi}\,. \tag{2.36}$$

However, this formulation introduces a new error source, as it makes use of the equality $g_n = \left[\!\left[\frac{\partial S g_n}{\partial n} \right]\!\right]$, which might be wrong at the discrete level. This is in fact documented by the numerical tests that we will present in section 2.6.

The second major difference relies on the construction of the lifting operators. According to [52], the extension $\tilde{g_*}$ on Ω_Γ of a generic function g_* defined only on Γ is defined as follows

$$\tilde{g_*}(x) = g_*(x_\Gamma) \quad \text{for all } x \in \Omega_\Gamma, \tag{2.37}$$

where x_Γ is the point of Γ that minimizes the distance to x. As a consequence, $\tilde{g_*}$ is constant along any normal direction issuing from Γ.

Let now $\tilde{g_d}$ be the extension in Ω_Γ of g_d according to (2.37). Then, considering the triangulation of Ω and the basis $\{\Psi_j\}$ of V_h (2.24), in [52] the lifting is defined as follows

$$\tilde{R}g_d = \begin{cases} -\sum_{\phi_j \geq 0} \Psi_j \tilde{g_d} & \text{in } \Omega_1 \\ \sum_{\phi_j < 0} \Psi_j \tilde{g_d} & \text{in } \Omega_2. \end{cases} \tag{2.38}$$

By construction, $\tilde{R}g_d$ has the prescribed jump $[[\tilde{R}g_d]]_\Gamma = g_d$ and it has continuous normal derivative across Γ: $\left[\!\left[\frac{\partial \tilde{R}g_d}{\partial n} \right]\!\right]_\Gamma = 0$. At the discrete level, the lifting becomes:

$$\tilde{R}_h g_d = \begin{cases} -\sum_{\phi_j \geq 0} \Psi_j \tilde{g}_{dj} & \text{in } \Omega_1 \\ \sum_{\phi_j < 0} \Psi_j \tilde{g}_{dj} & \text{in } \Omega_2, \end{cases} \tag{2.39}$$

where $\tilde{g}_{dj}$ denotes the value of the function $\tilde{g_d}$ at the node x_j. In this case, the jumps through the interface are satisfied in an interpolation sense: indeed it is $[[\tilde{R}_h g_d]]_\Gamma = \sum_j \tilde{g}_{dj} \Psi_j$ and $\left[\!\left[\frac{\partial \tilde{R}_h g_d}{\partial n} \right]\!\right]_\Gamma = \sum_j \tilde{g}_{dj} \frac{\partial \Psi_j}{\partial n}$.

The potential disadvantage of this methodology with respect to the approach that we have developed in the previous section 2.4.1 is that the interface has to be reconstructed since in (2.37) the closest point x_Γ is requested and, according to [52], this operation has to be performed using the interface explicitly.

A similar approach is proposed in [52] to construct the lifting of g_n. Note that after multiplying a function built as in (2.39) by the level-set function ϕ, it becomes continuous across the interface (since ϕ is equal to 0 on the interface), while its normal derivative exhibits a jump

of the desired magnitude. So, after constructing $\tilde{g_n}$ as in (2.37), the discrete lifting for g_n becomes:

$$\tilde{S}_h g_n = \begin{cases} -\sum_{\phi_j \geq 0} \Psi_j \tilde{g_n}_j \phi & \text{in } \Omega_1 \\ \sum_{\phi_j < 0} \Psi_j \tilde{g_n}_j \phi & \text{in } \Omega_2. \end{cases} \tag{2.40}$$

The potential drawback of this construction is that the multiplication by ϕ increases the polynomial order of the lifting function, thus requiring a polynomial refinement in the neighborhood of the interface. Moreover, the closest point extension requires again to rebuild the interface explicitly.

Remark 2.4.2. *The construction of the liftings of Sect. 2.4.1 can be seen as a generalization of those presented in this section and used in the ESIC method. Indeed, if instead of the extensions of g_* introduced in Sect. 2.3.1, we took the closest point extension (2.37), then the liftings $R_h g_d$ and $S_h g_n$ would coincide with (2.39) and (2.40). Notice again that in Sect. 2.4.1 we do not need at all to reconstruct* Γ.

2.4.5 Higher order approximations

Our construction concerns only linear finite elements. However, it can be generalized to higher order polynomial approximation. Indeed, only the finite element space (2.24) and the interpolation operator (2.28) have to be adapted.

The reason for treating only the linear case resides in the regularity of $\hat{u}$. Indeed, thanks to the construction of the liftings, we know that $\hat{u} \in H^2(\Omega)$, but it is not possible to ensure more regularity, e.g. $H^3(\Omega)$, as the liftings introduced so far do not provide control on second order derivatives across the interface. The use of higher order polynomial for the approximation would not be necessarily rewarded by an higher convergence rate, in particular close to the interface.

In order to achieve higher convergence rate, one would need to build liftings that correct the second (and possibly higher) derivatives. This can be done by considering functions of the type [51]

$$L_k g = H(\phi)\phi^k g \tag{2.41}$$

which have $k-1$ continuous derivatives across Γ and yield a jump in the k^{th} derivative. The solution would then be further decomposed, following (2.4) and (2.8):

$$u_i = \bar{u}_i + R_i g_d + S_i g_n + \sum_{j=2}^{l} L_i^j g \tag{2.42}$$

and the weak formulation would be changed accordingly.

2.5 Error analysis

In this section, we carry out the error analysis of the SESIC method, using the weak formulation (2.26). To perform the analysis, we will use exclusively the liftings $R_h^{glo} g_d$ and $S_h^{glo} g_n$, as they have continuous counterparts to which they can be compared. However, in practice, one would rather use the liftings $R_h g_d$ and $S_h g_n$. The following argument shows that our analysis also stands for this latter couple of liftings.

Let us denote by $\hat{u}_h$ the solution of the problem (2.26) using $R_h g_d$ and $S_h g_n$, $u_h = \hat{u}_h + R_h g_d + S_h g_n$, $\hat{u}_h^{glo}$ the solution of problem (2.26) using $R_h^{glo} g_d$ and $S_h^{glo} g_n$ and $u_h^{glo} = \hat{u}_h^{glo} + R_h^{glo} g_d + S_h^{glo} g_n$. First of all, we remark that $u_h - u_h^{glo} \in V_h$, as $\hat{u}_h, \hat{u}_h^{glo} \in V_h$ by definition and $R_h g_d - R_h^{glo} g_d \in V_h$, $S_h g_n - S_h^{glo} g_n \in V_h$ because of the way the support reduction has been performed in Sect. 2.4.1. Because both $\hat{u}_h$ and $\hat{u}_h^{glo}$ satisfy (2.26), we have:

$$\int_\Omega \nabla(u_h - u_h^{glo}) \cdot \nabla v_h = 0 \qquad \forall v_h \in V_h \,. \tag{2.43}$$

Since the Galerkin approximation of the Laplace problem is well-posed on V_h, $u_h - u_h^{glo}$ is the unique solution. As 0 is also a solution of that problem, these two solutions must coincide: $u_h - u_h^{glo} = 0$. Therefore, the two lifting sets yield the same final solution and so the convergence analysis carried out for u_h^{glo} also applies to u_h.

For the sake of analysis, besides the regularity assumptions made in the previous sections, we also suppose that the restrictions of the continuous liftings Rg_d and Sg_n to Ω_i belong to $H^2(\Omega_i)$, $i = 1,2$. Throughout this section we will denote by C a suitable constant which may change in the different inequalities, but that will always be independent of h. In this analysis, we neglect errors coming from the computation of the integrals (see Sect. 2.4.2).

In order to obtain *a-priori* error estimates, we split the error into three parts:

$$u - u_h^{glo} = (\hat{u} + Rg_d + Sg_n) - (\hat{u}_h^{glo} + R_h^{glo} g_d + S_h^{glo} g_n) \tag{2.44}$$

whence

$$||u - u_h^{glo}|| \leq ||\hat{u} - \hat{u}_h^{glo}|| + ||Rg_d - R_h^{glo} g_d|| + ||Sg_n - S_h^{glo} g_n|| \tag{2.45}$$

where $||\cdot||$ represents a suitable norm. Using the construction of the lifting operators and the classical interpolation error estimates for the operator π_h^1 (see, e.g., [82]), the last two terms may be bounded as follows: if T (respectively, T_h^{glo}) denotes either Rg_d or Sg_n (respectively, $R_h^{glo} g_d$ or $S_h^{glo} g_n$), we have

$$\sum_{i=1}^{2} ||T - T_h^{glo}||_{L^2(\Omega_i)} = ||T - T_h^{glo}||_{L^2(\Omega_2)}$$

thanks to (2.28) and to the fact that $T = H(\phi)\, t$. Therefore, we also have

$$||T - T_h^{glo}||_{L^2(\Omega_2)} = ||t - \pi_h^1(t)||_{L^2(\Omega_2)} \leq ||t - \pi_h^1(t)||_{L^2(\Omega)} \,.$$

Using standard interpolation results, we obtain

$$\sum_{i=1}^{2} ||T - T_h^{glo}||_{L^2(\Omega_i)} \leq Ch^2 |t|_{H^2(\Omega)} \,. \tag{2.46}$$

A similar reasoning gives

$$\sum_{i=1}^{2} ||T - T_h^{glo}||_{H^1(\Omega_i)} \leq Ch |t|_{H^2(\Omega)} \,. \tag{2.47}$$

Thus, we need to estimate the first term in (2.45). The analysis that we will carry out in this section does not rely on a specific construction of the lifting operators and could be applied directly to other constructions.

2.5.1 Convergence in the H^1 norm

To quantify the convergence in the energy norm, we use the first Strang lemma [82] (in our case, the bilinear forms of the continuous and discrete weak formulations coincide):

$$\begin{aligned} ||\hat{u} - \hat{u}_h^{glo}||_{H^1(\Omega)} \leq & \left(1 + \frac{\gamma}{\alpha}\right) \inf_{w_h \in V_h} ||\hat{u} - w_h||_{H^1(\Omega)} \\ & + \frac{1}{\alpha} \sup_{v_h \in V_h} \frac{1}{||v_h||_{H^1(\Omega)}} \left| \sum_{i=1}^{2} \int_{\Omega_i} \nabla(Rg_d + Sg_n) \cdot \nabla v_h \right. \\ & \left. - \sum_{i=1}^{2} \int_{\Omega_i} \nabla(R_h^{glo} g_d + S_h^{glo} g_n) \cdot \nabla v_h \right| \end{aligned} \tag{2.48}$$

where γ is the continuity constant of the bilinear form $a(u,v) = \int_\Omega \nabla u \cdot \nabla v$ and α its coercivity constant. The first term on the right hand side can be bounded thanks to the properties of the finite element space used, i.e., linear Lagrangian functions:

$$\inf_{w_h \in V_h} ||\hat{u} - w_h||_{H^1(\Omega)} \leq Ch |\hat{u}|_{H^2(\Omega)} \,. \tag{2.49}$$

For the second term, using the Cauchy-Schwarz inequality, we obtain:

$$\begin{aligned} & |\sum_{i=1}^{2}\int_{\Omega_i}\nabla(Rg_d+Sg_n)\cdot\nabla v_h-\int_{\Omega_i}\nabla(R_h^{glo}g_d+S_h^{glo}g_n)\cdot\nabla v_h| \\ = \; & |\sum_{i=1}^{2}\int_{\Omega_i}\nabla(Rg_d-R_{glo}^{h}g_d+Sg_n-S_{glo}^{h}g_n)\cdot\nabla v_h| \\ \leq \; & \left(\sum_{i=1}^{2}|(Rg_d-R_{glo}^{h}g_d)+(Sg_n-S_{glo}^{h}g_n)|_{H^1(\Omega_i)}\right)||v_h||_{H^1(\Omega)}\,. \end{aligned}$$

Therefore, thanks to the triangular inequality and the error estimate (2.47), we have

$$\begin{aligned} \sup_{v_h\in V_h} \quad & \frac{1}{||v_h||_{H^1(\Omega)}}\left|\sum_{i=1}^{2}\int_{\Omega_i}\nabla(Rg_d+Sg_n)\cdot\nabla v_h\right. && (2.50)\\ & \left.-\sum_{i=1}^{2}\int_{\Omega_i}\nabla(R_h^{glo}g_d+S_h^{glo}g_n)\cdot\nabla v_h\right| && (2.51)\\ & \leq\sum_{i=1}^{2}\left(|(Rg_d-R_h^{glo}g_d)+(Sg_n-S_h^{glo}g_n)|_{H^1(\Omega_i)}\right) && (2.52)\\ & \leq Ch(|\bar{R}g_d|_{H^2(\Omega)}+|\bar{S}g_n|_{H^2(\Omega)})\,. && (2.53) \end{aligned}$$

where we denoted $\bar{R}g_d=\bar{g}_d-\phi\,\nabla\bar{g}_d\cdot\nabla\bar{g}_n$ and $\bar{S}g_n=\phi\bar{g}_n$. To sum up, we have:

$$||\hat{u}-\hat{u}_h^{glo}||_{H^1(\Omega)}=Ch|\hat{u}|_{H^2(\Omega)}+Ch(|\bar{R}g_d|_{H^2(\Omega)}+|\bar{S}g_n|_{H^2(\Omega)}) \tag{2.54}$$

so that $\hat{u}_h^{glo}$ converges towards $\hat{u}$ with order 1 in H^1 norm, and thanks to (2.45), we have also optimal convergence of u_h^{glo} towards u:

$$\sum_{i=1}^{2}||u-u_h^{glo}||_{H^1(\Omega_i)}\leq Ch(|\hat{u}|_{H^2(\Omega)}+|\bar{R}g_d|_{H^2(\Omega)}+|\bar{S}g_n|_{H^2(\Omega)}). \tag{2.55}$$

2.5.2 Convergence in the L^2 norm

We define $e_h=u-u_h^{glo}\in L^2(\Omega)$ the total error done by our scheme, and $\hat{e}_h=\hat{u}-\hat{u}_h^{glo}\in H_0^1(\Omega)$. Then, for a fixed $h>0$, we define $\psi\in H_0^1(\Omega)$ the solution of the following dual problem:

$$\int_\Omega\nabla v\cdot\nabla\psi=\int_\Omega\hat{e}_h v\qquad\forall v\in H_0^1(\Omega)\,. \tag{2.56}$$

Using $v=\hat{e}_h$ in the previous relation, we have

$$||\hat{e}_h||_{L^2(\Omega)}^2=\int_\Omega\nabla\hat{e}_h\cdot\nabla\psi \tag{2.57}$$

which is equivalent to

$$||\hat{e}_h||^2_{L^2(\Omega)} = \sum_{i=1}^{2} \int_{\Omega_i} \nabla e_h \cdot \nabla \psi + \sum_{i=1}^{2} \int_{\Omega_i} \nabla(\hat{e}_h - e_h) \cdot \nabla \psi \tag{2.58}$$

We shall now analyse the two terms on the right hand side separately. For the first term we follow the usual *Aubin-Nitsche argument*. Using Galerkin's othogonality, we have for any $w_h \in V_h$:

$$\begin{aligned} \sum_{i=1}^{2} \int_{\Omega_i} \nabla e_h \cdot \nabla \psi &= \sum_{i=1}^{2} \int_{\Omega_i} \nabla e_h \cdot \nabla(\psi - w_h) \\ &\leq \sum_{i=1}^{2} ||e_h||_{H^1(\Omega_i)} ||\psi - w_h||_{H^1(\Omega_i)} \\ &\leq 2||\psi - w_h||_{H^1(\Omega)} \sum_{i=1}^{2} ||e_h||_{H^1(\Omega_i)} \,. \end{aligned} \tag{2.59}$$

Therefore,

$$\begin{aligned} \sum_{i=1}^{2} \int_{\Omega_i} \nabla e_h \cdot \nabla \psi &\leq 2 \inf_{w_h \in V_h} ||\psi - w_h||_{H^1(\Omega)} \sum_{i=1}^{2} ||e_h||_{H^1(\Omega_i)} \\ &\leq Ch||\psi||_{H^2(\Omega)} \sum_{i=1}^{2} ||e_h||_{H^1(\Omega_i)} \,. \end{aligned} \tag{2.60}$$

Regarding the second term in (2.58), we have, as the liftings are defined to be 0 in Ω_1,

$$\sum_{i=1}^{2} \int_{\Omega_i} \nabla(\hat{e}_h - e_h) \cdot \nabla \psi = \int_{\Omega_2} \nabla(Sg_n + Rg_d - S_h^{glo} g_n - R_h^{glo} g_d) \cdot \nabla \psi \,. \tag{2.61}$$

By a counter integration by parts, we obtain:

$$\begin{aligned} &\sum_{i=1}^{2} \int_{\Omega_i} \nabla(\hat{e}_h - e_h) \cdot \nabla \psi \\ &= -\int_{\Omega_2} (Sg_n + Rg_d - S_h^{glo} g_n - R_h^{glo} g_d)\, \Delta \psi \\ &\quad + \int_{\partial\Omega_2} \frac{\partial \psi}{\partial n} (Sg_n + Rg_d - S_h^{glo} g_n - R_h^{glo} g_d) \\ &= -\int_{\Omega_2} (Sg_n + Rg_d - S_h^{glo} g_n - R_h^{glo} g_d)\, \Delta \psi \\ &\quad + \int_{\Gamma} \frac{\partial \psi}{\partial n} (Sg_n + Rg_d - S_h^{glo} g_n - R_h^{glo} g_d) \end{aligned} \tag{2.62}$$

as the liftings have null trace on $\partial\Omega$. Using Cauchy-Schwarz inequality and a trace inequality, we have:

$$\begin{aligned}
&\sum_{i=1}^{2}\int_{\Omega_i}\nabla(\hat{e}_h-e_h)\cdot\nabla\psi\\
&\leq ||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Omega_2)}||\Delta\psi||_{L^2(\Omega_2)}\\
&\quad+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Gamma)}||\frac{\partial\psi}{\partial n}||_{L^2(\Omega)}\\
&\leq C\Big(||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Omega)}\\
&\quad+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Gamma)}\Big)\,||\psi||_{H^2(\Omega)}\,.
\end{aligned}\tag{2.63}$$

All together, (2.58) gives

$$\begin{aligned}
||\hat{e}_h||^2_{L^2(\Omega)}\leq&\Big(Ch\sum_{i=1}^{2}||e_h||_{H^1(\Omega_i)}+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Omega)}\\
&+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Gamma)}\Big)||\psi||_{H^2(\Omega)}
\end{aligned}\tag{2.64}$$

Thanks to the elliptic regularity, we have $||\psi||_{H^2(\Omega)}\leq C||\hat{e}_h||_{L^2(\Omega)}$, so

$$\begin{aligned}
||\hat{e}_h||_{L^2(\Omega)}\leq&\, Ch||e_h||_{H^1(\Omega)}+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Omega)}\\
&+||Sg_n+Rg_d-S_h^{glo}g_n-R_h^{glo}g_d||_{L^2(\Gamma)}
\end{aligned}\tag{2.65}$$

Using the H^1 error estimate that we have already derived and the interpolation errors for the liftings (2.46) and [89], we have:

$$||\hat{e}_h||_{L^2(\Omega)}\leq Ch^2\left(|\hat{u}|_{H^2(\Omega)}+|\bar{R}g_d|_{H^2(\Omega)}+|\bar{S}g_n|_{H^2(\Omega)}+|\bar{R}g_d|_{H^2(\Gamma)}+|\bar{S}g_n|_{H^2(\Gamma)}\right),\tag{2.66}$$

where we have supposed that Rg_d and Sg_n are regular enough so that the norms on the right hand side are well defined.

2.6 Numerical results

In this section, we present numerical results obtained using the methodologies described in the previous sections for different geometric dimensions.

2.6.1 1D test case

First of all, we consider a 1D Poisson problem, as this allows us to make complete error measurements and visualizations. We consider the unit interval $\Omega=(0,1)$ with an interface located in $\Gamma=\{\pi^{-1}\}$ so that the uniform meshes that we will use will not conform with the interface. The level set function is defined as $\phi(x)=\pi^{-1}-x$. The Poisson problem consists in

finding $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} &-u''(x) = -e^x \quad \text{in } \Omega \\ &u(0) = 1 \\ &u(1) = e + 2, \end{aligned} \tag{2.67}$$

with the jump conditions

$$[[u]]_\Gamma = -2\pi^{-1} \qquad \left[\left[\frac{\partial u}{\partial n}\right]\right]_\Gamma = 2. \tag{2.68}$$

The exact solution reads

$$u = \begin{cases} e^x & \text{if } x \le \pi^{-1} \\ e^x + 2x & \text{if } x > \pi^{-1}. \end{cases} \tag{2.69}$$

As both jump conditions are non-homogeneous, we need to extend them in the whole domain Ω. To this aim, we define two possible sets of extension to highlight the role of the choice of the extensions for the convergence of the method. The first set is made of arbitrary functions:

$$\bar{g}_d(x) = -(2\pi^{-1} + \sin(x - \pi^{-1})) \qquad \bar{g}_n(x) = 1 + e^{(x-\pi^{-1})} \tag{2.70}$$

while the second set, called simplified extensions, is made of constant functions:

$$\bar{\bar{g}}_d(x) = -2\pi^{-1} \qquad \bar{\bar{g}}_n(x) = 2 \tag{2.71}$$

For the simplified extensions (2.71), thanks to the definitions (2.13) and (2.16), it is easy to see that the interpolation does not introduce any error while if we take the extensions in (2.70), the interpolation will produce some error on the jumps and the conditions (2.68) will not be satisfied exactly. In the latter case, we have measured the error due to the liftings on the jump conditions (2.68) for $\mathbb{P}_1$ and $\mathbb{P}_2$ finite elements. The following table shows the order of convergence of these errors for $h \to 0$:

	Convergence rates for errors on:			
Elements	$[[Rg_d]]$	$\left[\left[\frac{\partial Rg_d}{\partial n}\right]\right]$	$[[Sg_n]]$	$\left[\left[\frac{\partial Sg_n}{\partial n}\right]\right]$
$\mathbb{P}_1$	3	2	2	1
$\mathbb{P}_2$	3	2	3	2

We can see that the orders are optimal for all the quantities and that we have a superconvergence for Rg_d with $\mathbb{P}_1$ elements.

We apply now the SESIC method to solve the 1D problem. To measure the associated error, we use three error measures:

- the H^1 norm of the error in the domain $\Omega^* = (0, \pi^{-1} - 0.1) \cup (\pi^{-1} + 0.1, 1)$,

- the L^2 norm of the error in the domain Ω^*,
- the L^∞ norm of the error in the entire domain Ω.

We typically get a quite smooth error pattern on the whole domain, as shown in Fig. 2.17 (left), what provides an evidence that all the components of the error are balanced.

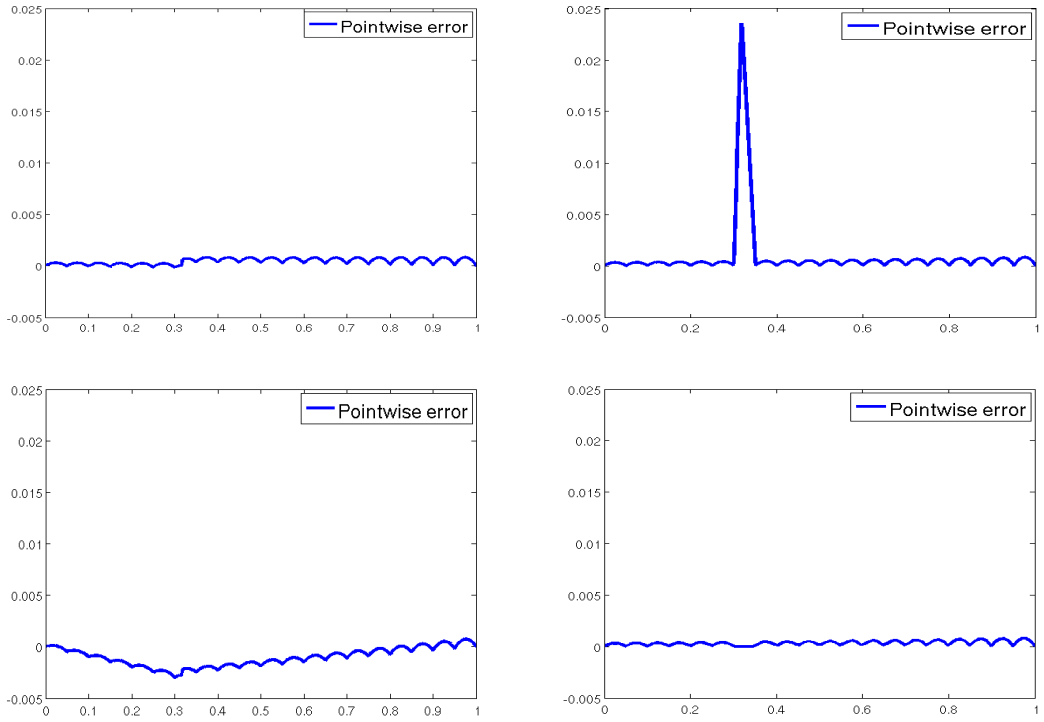

Figure 2.17: Pointwise error in the solution for the 1D test using $\mathbb{P}_1$ elements and a grid of 20 intervals: top left, using the SESIC method; top right, using the SESIC method without lifting for the normal derivative; bottom left, using the modification (2.36); bottom right, using the ESIC method.

However, as shown in the next table, we do not get optimal orders for the maximum error with $\mathbb{P}_2$ elements, while the errors in Ω_* and the errors for the $\mathbb{P}_1$ elements exhibit optimal convergence rates:

Norm	$\mathbb{P}_1$ elements	$\mathbb{P}_2$ elements
$L^2(\Omega^*)$	2	3
$H^1(\Omega^*)$	1	2
$L^\infty(\Omega)$	2	*2*

Optimal orders also in the L^∞ norm can be recovered if we use the simplified extensions (2.71). The suboptimality remarked for $\mathbb{P}_2$ might be due to a lack of regularity of $\hat{u}$: we can ensure that $\hat{u} \in H^2(\Omega)$ but we would need to build liftings taking into account also the second derivatives to provide more regularity (see Sect. 2.4.5).

Using this test case, we can also provide a justification of the use of the lifting for the jump in the normal derivative. If we do not take into account the lifting Sg_n, we have to use the weak formulation (2.25). We keep the same definition for Rg_d (with the extension given in (2.70)).

This produces results that are different from our method mainly near the interface: figure 2.17 (right) shows a large error peak in the element crossed by the interface. The error located in that element is far larger than the interpolation error visible in the other elements.

This additional error comes from the fact that the underlying finite element space cannot reproduce jumps inside the elements. It is then impossible to reduce this error without providing the finite element space with the ability to capture jumps. In the SESIC method, this is the role of the lifting, that carry the jumps but does not belong to the finite element space. We can also see this behavior in the following table that shows the convergence orders for the method without Sg_n: even if the errors computed in the domain Ω_* show optimal convergence orders, the high error near the interface reduces the convergence rates in the maximum norm.

Norm	$\mathbb{P}_1$ elements	$\mathbb{P}_2$ elements
$L^2(\Omega^*)$	2	3
$H^1(\Omega^*)$	1	2
$L^\infty(\Omega)$	*1*	*1*

The role of the lifting Sg_n for the normal derivative is then clear: it helps to reduce the magnitude of the error in the neighborhood of the interface.

For the sake of comparing our method with the ESIC method described in section 2.4.4, we have also tested the weak formulation modified with (2.36) and with the liftings described in section 2.4.1. This allows us to bring to light the consequences of using (2.36) (we will use the two-side integration (See Sect. 2.4.2) to keep the error coming exclusively from method and not from the integration scheme). The following table shows the convergence rates for this test case: we can clearly see that the convergences are slower than with the weak formulation (2.26).

Norm	$\mathbb{P}_1$ elements	$\mathbb{P}_2$ elements
$L^2(\Omega^*)$	*1*	*2*
$H^1(\Omega^*)$	1	2
$L^\infty(\Omega)$	*1*	*2*

The reason for this reduction of the convergence rates is found in the derivation of the weak formulation for the ESIC method in Sect. 2.4.4: it is assumed that $g_n = \left[\!\left[\frac{\partial S g_n}{\partial n}\right]\!\right]$, which is true at the continuous level but not at the discrete level. An additional error accounting for this mismatch appears. Due to the slow convergence of this error (proportional to h), it dominates the whole error for fine grids.

Figure 2.18 shows the typical pattern that we get using the modified weak formulation. The solution looks like if the force applied on the interface (by the term $\int_\Gamma g_n v_h$ in (2.25) and (2.26)) was badly estimated, leading to the trend of the error to be greater near the interface, while producing no peak there.

The origin of the error is also emphasized in the next figure, that shows that there is a big correlation between the error $\left|\left[\!\left[\frac{\partial S g_n}{\partial n}\right]\!\right] - g_n\right|$ and the L^2 error.

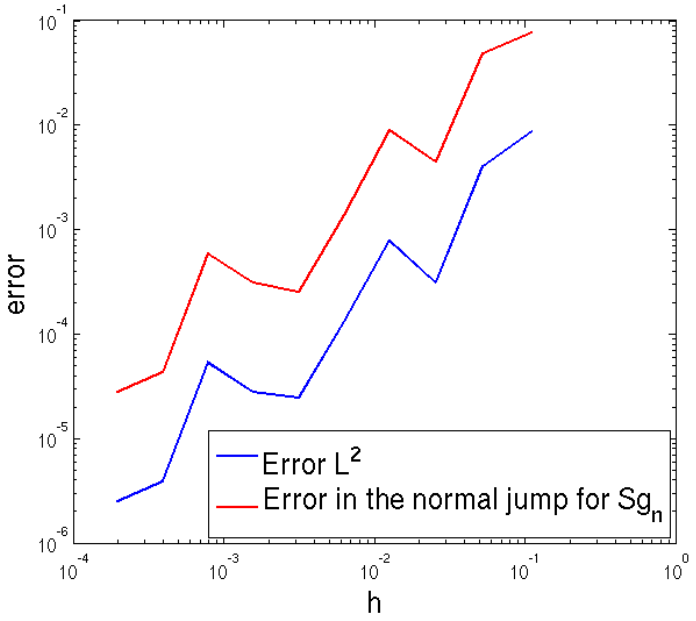

Figure 2.18: Comparison between the error on the normal jump of Sg_n and the global L^2 error.

This error does not show up in the original ESIC method as shown in the next table.

Norm	Convergence rate
$L^2(\Omega^*)$	2
$H^1(\Omega^*)$	1
$L^\infty(\Omega)$	2

The reason is that in the latter method, polynomial refinement is performed near the interface. In this example, $\mathbb{P}_1$ elements have been used except for the elements containing the interface

where a $\mathbb{P}_2$ basis was defined. As shown at the beginning of this section, when $\mathbb{P}_2$ are used, the error $|\left[\left[\frac{\partial S g_n}{\partial n}\right]\right] - g_n|$ has a second order convergence and then has the same behavior as the interpolation error. However this approach requires an additional programming effort as well as unnecessary addition of degrees of freedom: Fig. 2.19 shows the pointwise error for the ESIC method and we can observe that the error in the element containing Γ is smaller than in the rest of the domain.

Finally, we investigate the effects of computing integrals using the regularized integrands introduced in section 2.4.2. We test both widths $w = \sqrt{\frac{h}{2}}$ and $w = h$. The effects of the thickness of the regularization band is clearly visible in Fig. 2.19 (left) where we show the behavior of the $L^2(\Omega^*)$ norm of the error.

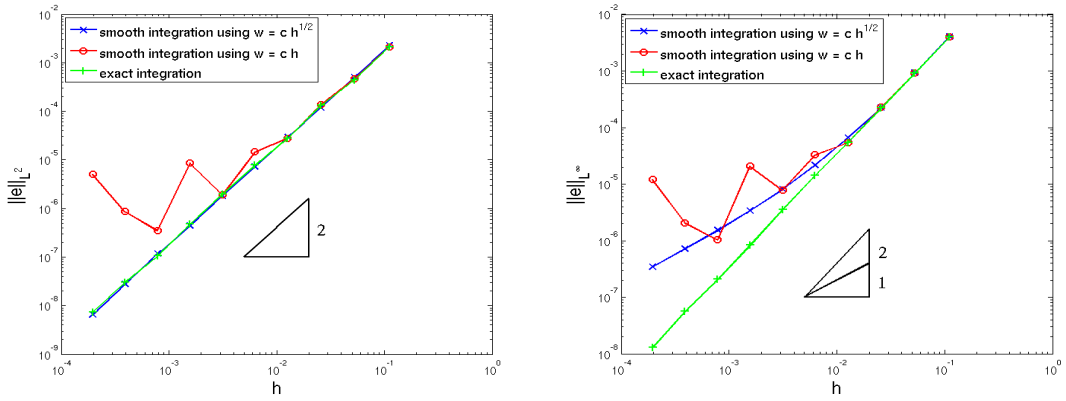

Figure 2.19: $L^2(\Omega^*)$ error (left) and $L^\infty(\Omega)$ error (right) associated with the different integration methods.

As stated previously in this section, optimal order of convergence is achieved with the exact integration. Using the proposed smooth integration, i.e., with w proportional to $\sqrt{h}$, we also get the optimal order of convergence in this norm. On the contrary, using a regularization band with thickness $w = h$, we obtain an unpredictable behavior when h becomes small. The convergence rate in that case is difficult to assess. We can observe the same kind of behavior in the $L^\infty(\Omega)$ norm as shown in Fig. 2.19 (right). In this case we can see that using w proportional to $\sqrt{h}$ leads to a convergence slower than the optimal one: if for coarse meshes the convergence rate seems to be close to 2, it then slows down to 1 for finer meshes.

All these results correspond quite well to the remarks that we made in Sect. 2.4.2. The smooth integration using w proportional to $\sqrt{h}$ permits to control the error leading to regular convergence, even optimal in the $L^2(\Omega^*)$ norm. This means that the error generated near the interface, reported in the $L^\infty(\Omega)$ norm, is confined in that area and does not pollute the solution in the whole domain. On the contrary, with $w = h$, we lose the control on the quadrature error causing a large error in the interface area that eventually spreads in the whole

domain.

2.6.2 2D test case

We first test our method on the two dimensional test case defined in [52]. This test is quite simple as the exact solution is continuous, so that the jump is only in the normal derivative. The domain is defined as the square $\Omega = (-1,1)^2$ and the interface is the circle with radius 0.5 centered at the origin.

The exact solution reads:

$$u(x,y) = \begin{cases} 1 & \text{if } x^2+y^2 \leq 0.25 \\ 1-\log(2\sqrt{x^2+y^2}) & \text{if } x^2+y^2 > 0.25. \end{cases}$$

Dirichlet boundary conditions are set to ensure this exact solution and the jump in the normal derivative to be

$$\left[\left[\frac{\partial u}{\partial n}\right]\right]_\Gamma = -2.$$

Cartesian meshes with n cells on each side were used. The results that we obtain are listed in the following table:

n	Maximal error on Γ	rate	Maximal error in Ω	rate
9	7.13×10^{-3}		1.84×10^{-2}	
19	2.85×10^{-3}	1.23	4.61×10^{-3}	1.85
39	7.10×10^{-4}	1.93	1.13×10^{-3}	1.96
79	1.71×10^{-4}	2.02	2.71×10^{-4}	2.02

We can see that the SESIC method gives optimal orders of convergence both at the interface and in the entire domain. This means that the error decreases with the same rate everywhere in the domain, included near and on the interface. Moreover, the magnitude of the error is lower than for the methods (ESIC, XFEM and IBM) compared in [52] (even if the orders of magnitude are similar), while being easier to implement and cheaper to compute (These good performances may be due to the simple test problem that we adopted, where g_n is constant).

We want then to investigate the influence of the regularity of the interface. The next test case consists in a domain cut by a C^1 curve: the level set function is defined as (see Fig. 2.20 for a representation):

$$\phi(x,y) = \begin{cases} \sqrt{x^2+(y-0.5)^2}-0.2 & \text{if } y > 0.5 \\ |x|-0.2 & \text{if } |y| \leq 0.5 \\ \sqrt{x^2+(y+0.5)^2}-0.2 & \text{if } y < -0.5 \end{cases}$$

and the exact solution is defined as

$$u(x,y) = \begin{cases} 1 - \log(2\sqrt{x^2+y^2}) & \text{if } \phi(x,y) \geq 0 \\ 0 & \text{if } \phi(x,y) < 0\,. \end{cases} \tag{2.72}$$

The Dirichlet boundary conditions on $\partial\Omega$ and the jump conditions across Γ are computed using this exact solution. Using the SESIC method, we obtain the following errors:

n	$H^1(\Omega^*)$ error	rate	$L^2(\Omega^*)$ error	rate	Maximal error in Ω	rate
9	6.63×10^{-1}		4.55×10^{-2}		9.07×10^{-2}	
19	3.23×10^{-1}	0.96	1.19×10^{-2}	1.79	3.50×10^{-2}	1.27
29	2.11×10^{-1}	1.01	5.45×10^{-3}	1.85	1.78×10^{-2}	1.60
39	1.56×10^{-1}	1.01	3.22×10^{-3}	1.78	1.09×10^{-2}	1.64
69	8.85×10^{-2}	1.00	1.12×10^{-3}	1.84	4.07×10^{-3}	1.73
99	6.17×10^{-2}	1.00	6.04×10^{-4}	1.72	2.23×10^{-3}	1.67
149	4.11×10^{-2}	1.00	2.98×10^{-4}	1.73	1.14×10^{-3}	1.65
199	3.08×10^{-2}	1.00	1.85×10^{-4}	1.65	7.13×10^{-4}	1.61

We can remark that the error estimate in H^1 is optimal whereas the errors in the L^2 and L^∞ norms have convergence rates between 1.5 and 2.

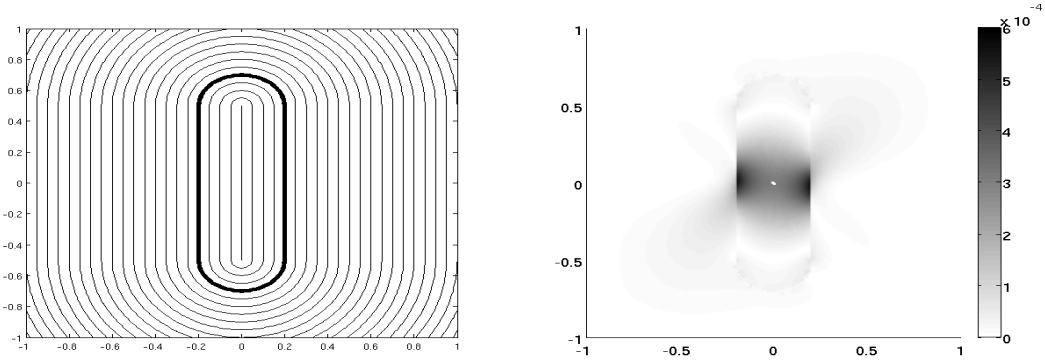

Figure 2.20: Representation of the level set function (left, Γ bold) and pointwise error produced by the SESIC method with $n = 99$ (right).

Finally, we used the SESIC method to approximate the solution of a problem where the curve

Γ is only C^0. We define the level set function as:

$$\phi(x,y) = \begin{cases} \sqrt{(x-\frac{\sqrt{2}}{4})^2+y^2}-0.5 & \text{if } x \geq 0 \\ \sqrt{(x+\frac{\sqrt{2}}{4})^2+y^2}-0.5 & \text{if } x < 0 \end{cases}$$

and the exact solution is defined as in (2.72). The next table reports the errors obtained in this test case.

n	$H^1(\Omega^*)$ error	rate	$L^2(\Omega^*)$ error	rate	Maximal error in Ω	rate
9	3.01×10^{-1}		5.26×10^{-2}		7.74×10^{-2}	
19	1.57×10^{-1}	0.87	2.74×10^{-2}	0.88	5.13×10^{-2}	0.55
29	1.04×10^{-1}	0.97	1.86×10^{-2}	0.91	3.48×10^{-2}	0.91
39	7.90×10^{-2}	0.93	1.41×10^{-2}	0.94	2.87×10^{-2}	0.66
69	4.73×10^{-2}	0.90	8.18×10^{-3}	0.96	1.91×10^{-2}	0.71
99	3.40×10^{-2}	0.91	5.68×10^{-3}	1.01	1.47×10^{-2}	0.73
149	2.32×10^{-2}	0.94	3.80×10^{-3}	0.98	1.06×10^{-2}	0.79
199	1.73×10^{-2}	1.01	2.87×10^{-3}	0.98	8.04×10^{-3}	0.97

The lack of regularity of Γ is directly reflected in the convergence orders: the H^1 error as well as those in L^2 and L^∞ norm do not exceed the first order convergence. This can also be seen when looking at the pointwise error (Fig. 2.21) where one can remark that the largest error are created near the two points of low regularity of Γ.

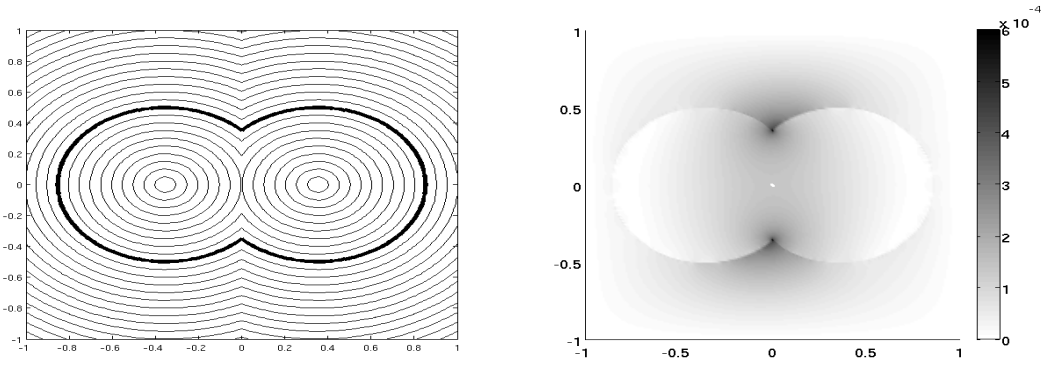

Figure 2.21: Representation of the level set function (left, Γ bold) and pointwise error produced by the SESIC method with $n = 99$ (right).

2.6.3 3D test case

We finally consider a 3D problem. This example was implemented in the parallel version of the finite element library LifeV (www.lifev.org). We consider the domain $\Omega = (-1,1)^3$. The level set function is $\phi(x,y,z) = \left(x^2+y^2+z^2\right)^{1/2} - 0.5$ so that the interface Γ is a sphere centered in the origin with a radius 0.5. In Ω, we want to find the solution $u : \Omega \to \mathbb{R}$ of the problem $-\Delta u = 0$ with jumps conditions through Γ:

$$[[u]]_\Gamma = 2 - e^{x+z}\sin(\sqrt{2}y)$$

$$\left[\left[\frac{\partial u}{\partial n}\right]\right]_\Gamma = 4 + 2e^{x+z}\left((x+z)\sin(\sqrt{2}y) + \sqrt{2}y\cos(\sqrt{2}y)\right).$$

Boundary conditions are such that the exact solution is

$$u(x,y,z) = \begin{cases} (x^2+y^2+z^2)^{-1/2} & \text{if } \phi(x,y,z) \geq 0 \\ e^{x+z}\sin(\sqrt{2}y) & \text{if } \phi(x,y,z) < 0. \end{cases}$$

We solved this problem using $\mathbb{P}_1$ finite elements. To measure the error, we computed both the L^2 error in the domain $\Omega^* = \{\mathbf{x} \in \Omega \mid |\phi(\mathbf{x})| > 0.1\}$ for regularity and maximal error in all the finite element nodes (denoted hereafter l^∞). We did not compute errors in the H^1 norm nor in L^∞ norm because these functionalities were not provided by the library used. We used Cartesian meshes with n representing the number of nodes in each direction. The computed errors and convergence rates are given in the next table.

n	degrees of freedom	error $L^2(\Omega^*)$	rate	error l^∞	rate
5	125	6.26×10^{-1}		3.52×10^{-1}	
10	1000	2.11×10^{-1}	1.34	8.39×10^{-2}	1.79
20	8000	1.72×10^{-2}	3.36	2.09×10^{-2}	1.86
40	64000	3.07×10^{-3}	2.40	5.63×10^{-3}	1.82
60	216000	1.34×10^{-3}	2.00	2.63×10^{-3}	1.84
80	512000	7.50×10^{-4}	1.99	1.67×10^{-3}	1.56
100	1000000	4.78×10^{-4}	2.00	1.17×10^{-3}	1.58

The order of convergence in the $L_2(\Omega^*)$ norm is close to the optimal rate 2 when n increases. This is due to the fact that the error produced by the smooth integration is well controlled and it is reflected only in the l^∞ norm as it is confined to the interface area (as shown in Fig. 2.22). The error in the l^∞ norm is expected to behave like the $L^\infty(\Omega)$ error in the 1D test case, i.e., to decrease slowly for finer meshes until it reaches the convergence rate 1.

Finally, with this test case we can emphasize the need for a good integration scheme for the singular functions. Indeed, if instead of smoothing the integrands on a width proportional to

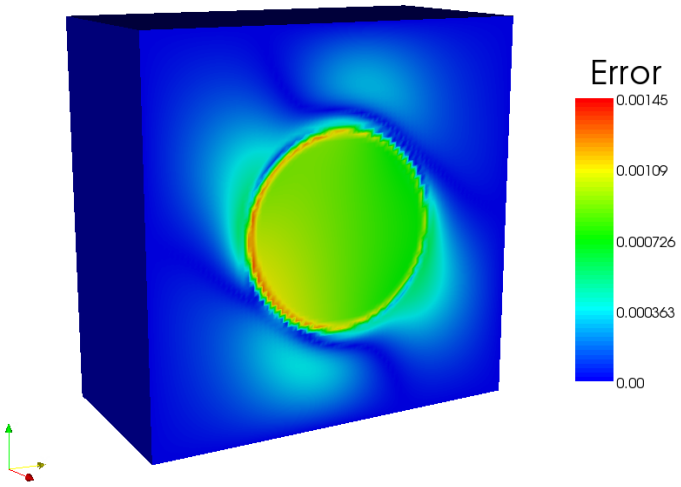

Figure 2.22: Representation of the error on the surface $x = 0$ for $n = 80$. We can remark that the error near the interface is of the same order of magnitude as far from it.

$\sqrt{h}$, we use a width of h, we get the following results:

n	degrees of freedom	error $L^2(\Omega^*)$	rate	error l^∞	rate
5	125	6.33×10^{-1}		3.18×10^{-1}	
10	1000	2.16×10^{-1}	1.33	7.23×10^{-2}	1.83
20	8000	1.71×10^{-2}	3.39	1.86×10^{-2}	1.82
40	64000	3.92×10^{-3}	2.05	1.27×10^{-2}	0.53
60	216000	1.38×10^{-3}	2.52	5.47×10^{-3}	2.03
80	512000	1.17×10^{-3}	0.57	7.57×10^{-3}	−1.11
100	1000000	1.01×10^{-3}	0.65	5.82×10^{-3}	1.16

We can observe that the convergence is slower when the mesh gets finer because the quadrature error is dominating.

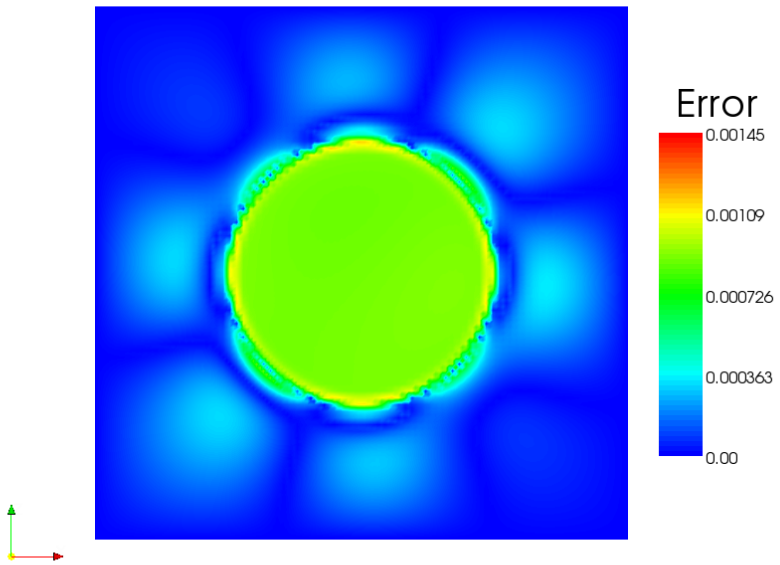

Figure 2.23: Error for the 3D test case on the plane $y = 0$ when the interface width is $w = 0.0796$ $(= \sqrt{h/2})$.

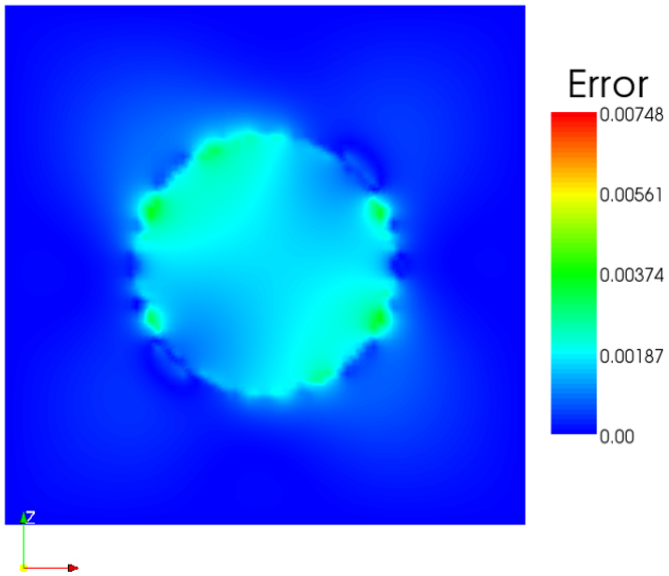

Figure 2.24: Error for the 3D test case on the plane $y = 0$ when the interface width is $w = 0.0127$ $(= h)$.

3 Modeling free surface flows in OSRs

In this chapter, we study the hydrodynamics of the OSRs that we introduced in Sect. 1.2.1. We describe the numerical tools required for simulations and, in particular, how the SESIC approach can be efficiently adapted to this context. We will characterize a solver with a simple design that yields a high accuracy without sacrifying scalability properties and computational efficiency. Indeed, the computational cost for the simulation of an OSR is very high due to the small time step used, the long time interval considered and the fine mesh needed for accurate results. To obtain reasonable computation times, the simulations are run in a parallel environment. The scalability, i.e. the ability to take advantage of the parallel computations efficiently, is an important criterion for the choice of the components of the method that we propose.

Before discussing the numerical approximation, we model the OSR fluid dynamics. First of all, the motion prescribed by the shaker can be represented as the sum of two rotations with opposite angular velocities: one around the axis of the vessel and the other one around the center of the shaker, see Fig. 3.1 (the vessel is then always translated from its original position). The shaking radius R_s is defined as the distance between the two centers of rotations, thus controling the amplitude of the motion, see Fig. 3.1. Instead of considering a moving domain, we change our point of view and model the OSR by a fixed cylinder, with height H and radius R (see Fig. 3.2):

$$\Omega = \{(x,y,z) = (r\cos(\theta), r\sin(\theta), z) \in \mathbb{R}^3 \ : \ r \in [0,R), \theta \in [0,2\pi), z \in (0,H)\} \, .$$

The motion is then represented by fictitious forces (see also [25]) that we can decompose into two components: a radial one to account for the centrifugal effects:

$$\mathbf{f}_r = \rho R_s (\dot{\theta})^2 \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{pmatrix}$$

and a tangential one, describing the possible change in the agitation rate with a Coriolis type

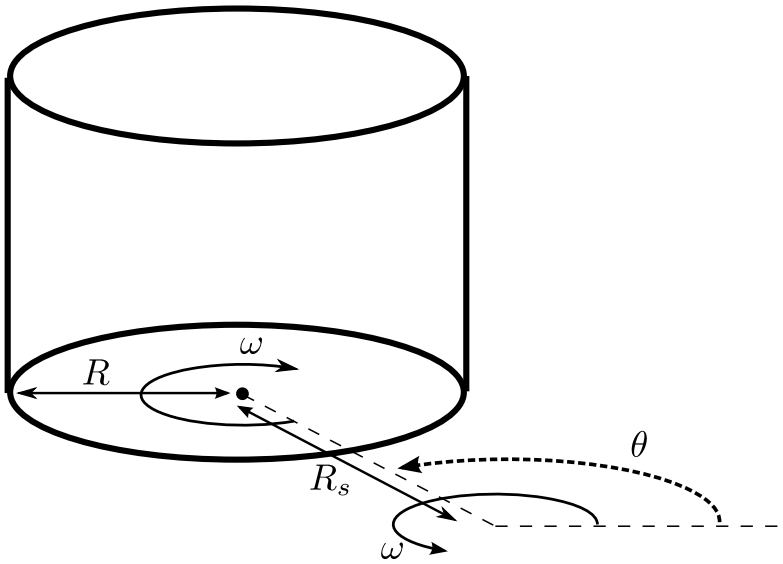

Figure 3.1: Schematic representation of the motion of an OSR. Remark that, for the sake of clarity, the illustration shows a configuration where $R < R_s$, while real configurations feature usually $R > R_s$.

force

$$\mathbf{f}_t = \rho R_s \ddot{\theta} \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \\ 0 \end{pmatrix}$$

where $\theta = \theta(t)$ denotes the angle between the position of the bioreactor and the horizontal axis (see Fig. 3.1). The total external force acting on the fluid (fictitious forces and gravity) can therefore be described as

$$\rho \mathbf{f} = \rho\, \mathbf{g} + \mathbf{f}_r + \mathbf{f}_t \tag{3.1}$$

where $\mathbf{g} = \left(0, 0, -g\right)^T$ is the gravitational acceleration. We underline here that, due to the orbital motion of the container, the reference frame adopted here is always translated from its original position, but does not rotate. Therefore, only the fictitious forces appear and no Coriolis term involving the velocity (like that reported in [21]) applies.

The regime in which we are interested is a constant angular velocity $\dot{\theta} = \omega$. However, as we will begin our simulations using a flat interface and a null velocity, we avoid starting directly with the targeted agitation rate, since this would create large perturbations that would probably take a long time to disappear. Therefore, we mimic a progressive acceleration of the OSR using

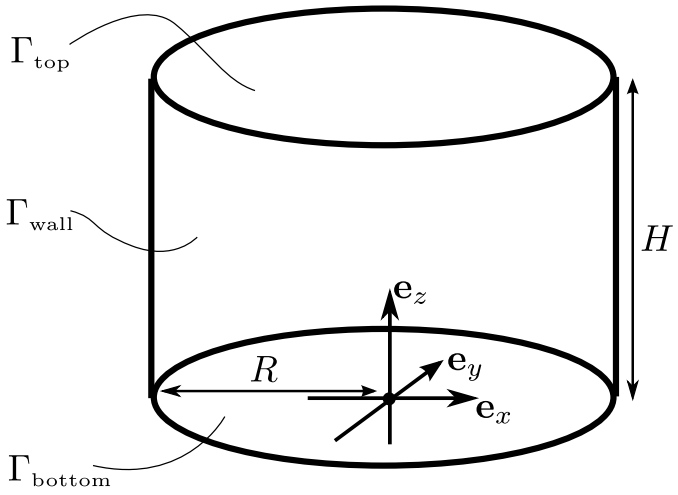

Figure 3.2: Geometry of an OSR.

the following law:

$$\theta(t) = \begin{cases} \omega \frac{1+\sin(\frac{-\pi}{2}+\frac{\pi t}{T_{\text{acc}}})}{2} & \text{if } t < T_{\text{acc}} \\ \omega(t - \frac{T_{\text{acc}}}{2}) & \text{if } t \geq T_{\text{acc}} \end{cases} \tag{3.2}$$

where $(0, T_{\text{acc}})$ is the time interval during which the OSR is accelerated. Remark that with this law, we have $\dot{\theta}(t) = \omega$ for all $t \geq T_{\text{acc}}$.

Remark 3.0.1. *In this work, we will assume that surface tension has a negligible effect on the flow. This assumption can be justified by considering the Eötvös number:*

$$Eo = \frac{4(\rho_l - \rho_a)gR^2}{\sigma}$$

which is used to describe the relative importance of surface tension with respect to gravity, where σ is the surface tension coefficient. In our case, the Eötvös number is of the order of 1000*, indicating that surface tension can be neglected.*

3.1 The coupled problem

Using the level set method that we presented in Sect. 1.2.1, we can write the coupled problem, consisting of the Navier-Stokes equations for the fluids dynamics and of the interface motion represented by the transport of the level set function. Remark that we write now the Navier-Stokes equations in the whole domain Ω: we want to find $(\mathbf{u}, p, \phi)$, all depending on space and

time, such that

$$\begin{cases} \rho\left(\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) - \nabla\cdot\mathbf{T}(\mathbf{u},p;\mu) &= \rho\mathbf{f} \\ \nabla\cdot\mathbf{u} &= 0 \qquad \text{in } \Omega, \forall\, t\in(0,T) \\ \partial_t\phi + \mathbf{u}\cdot\nabla\phi &= 0 \end{cases} \tag{3.3}$$

where ρ and μ depend on the position of the interface through the following relations:

$$\begin{cases} \rho = & \rho_a + H(\phi)(\rho_l - \rho_a) \\ \mu = & \mu_a + H(\phi)(\mu_l - \mu_a) \qquad \text{in } \Omega, \forall\, t\in(0,T) \end{cases} \tag{3.4}$$

with $H(\cdot)$ the Heaviside function. The system of equations (3.3) is highly non-linear due to the non-linear convection inherent to the Navier-Stokes equations, to the transport term of the level-set function and to the coupling between the equations through ρ and μ.

In this work, we do not consider turbulence modeling. Indeed, the question of whether the flow is actually turbulent is still open and probably depends on the configuration considered. We will see in chapter 5 (see in particular Sect. 5.3) that we can reproduce the velocity of the different fluids without taking into account turbulence. We do not observe variations in time of the velocity that could not be reproduced numerically, evidencing that, at least at that particular regime, turbulence can be neglected. Turbulence could however play a critical role for the mixing, by enhancing it at the small scales. This would have an impact in particular on the results on mixing particles patterns that we illustrate in Sect. 5.5.

Remark 3.1.1. *We did not rescale the momentum equation by ρ, as performed by some authors, e.g. [66]. If this scaling helps in removing some singularities across the interface $\left(\left[\left[\frac{\partial p}{\partial n}\right]\right] = 0$ as the right hand side of the momentum equation is now continuous$\right)$, it actually leads to a wrong interface condition. Indeed, the natural condition is the continuity of the normal stress*

$$[[\mathbf{T}(\mathbf{u},p;\mu)\mathbf{n}]] = \mathbf{0}$$

but with the scaling, the condition would become

$$[[\frac{1}{\rho}\mathbf{T}(\mathbf{u},p;\mu)\mathbf{n}]] = \mathbf{0}$$

which does not imply the continuity of the normal stress since ρ is discontinuous across Γ.

3.2 Initial and boundary conditions

To ensure that problem (3.3) is well-posed, we need to provide suitable initial and boundary conditions for the different quantities.

3.2.1 Initial conditions

Initial conditions must be provided for $\mathbf{u}$ and ϕ:

$$\begin{cases} \mathbf{u}(\mathbf{x}, t=0) = & \mathbf{u}_0(\mathbf{x}) \equiv \mathbf{0} \\ \phi(\mathbf{x}, t=0) = & \phi_0(\mathbf{x}) \end{cases} . \tag{3.5}$$

We suppose that $\phi_0 \in C^0(\bar{\Omega})$ so that the interface Γ is well-defined.

3.2.2 Boundary conditions for the fluid

If initial conditions are quite easy to setup, boundary conditions are more complicated to define. We investigate first of all the conditions applying to the velocity and the pressure. To this aim, we split the boundary of the domain $\partial\Omega$ in 3 parts (see Fig. 3.2):

- $\Gamma_{\text{top}} = \{(x, y, z) \in \overline{\Omega} \,:\, z = H\}$ represents the disk at the top of the domain Ω;
- $\Gamma_{\text{bottom}} = \{(x, y, z) \in \overline{\Omega} \,:\, z = 0\}$ represents the lower part of Ω;
- $\Gamma_{\text{wall}} = \{(x, y, z) \in \overline{\Omega} \,:\, x^2 + y^2 = R^2\}$ is the curved vertical wall (the lateral surface of the cylinder).

In the regimes that we will investigate, the surface Γ never reaches the top nor the bottom of the container, i.e. $\Gamma \cap \Gamma_{\text{top}} = \emptyset$ and $\Gamma \cap \Gamma_{\text{bottom}} = \emptyset$ for all $t \in [0, T]$.

We start by devising the boundary conditions for the pair $(\mathbf{u}, p)$. The classical no-slip condition $\mathbf{u} = \mathbf{0}$ imposed on the walls (see [4]) cannot be applied on Γ_{wall}. Indeed, because of the zero velocity, the level set function ϕ would not be advected, keeping the interface fixed on Γ_{wall} thus resulting in an unphysical situation. Therefore, we apply the no-slip condition only on $\Gamma_{\text{top}} \cup \Gamma_{\text{bottom}}$, but alternative conditions must be considered for Γ_{wall}. A review of some possibilities in a 2D framework can be found in [84]. The alternatives that we consider are made of two components, reflecting the two roles of the no-slip condition:

1. a "no-penetration" condition, whose role is to avoid leaks of fluid across the boundary;
2. a "friction" condition, which controls the way the fluid can slip on the lateral surface.

3.2.3 No penetration conditions

Two different ways of forcing the fluid to stay within the container can be devised. A first possibility would be to impose the horizontal velocity to be null:

$$\begin{cases} \mathbf{u} \cdot \mathbf{e}_x = 0 \\ \mathbf{u} \cdot \mathbf{e}_y = 0 \end{cases} \quad \text{on } \Gamma_{\text{wall}} . \tag{3.6}$$

This first condition is very simple to set up, but it has the disadvantage that it cannot be applied to other geometries, e.g., a sphere. Moreover, even in the cylindrical case, waves cannot slip on the wall with this condition, see the test in Sect. 4.3. For curved geometries, it is also possible to impose the normal component of the velocity as an essential condition:

$$\mathbf{u}\cdot\mathbf{n} = 0 \qquad \text{on } \Gamma_{\text{wall}}\,. \tag{3.7}$$

The only difficulty is how to define the normal at the discrete level. We will discuss this issue in Sect. 3.4.4 and test these two possibilities in Sect. 4.3.

3.2.4 Friction conditions

Once the no-penetration condition is set, we still have to choose the conditions for the remaining velocity components. In the following, we denote by $\boldsymbol{\tau}$ one of the remaining directions: for example, $\boldsymbol{\tau} = \mathbf{e}_z$ in case the horizontal condition (3.6) is used, or $\boldsymbol{\tau}$ is any of the tangent directions $\boldsymbol{\tau}\cdot\mathbf{n} = 0$ on Γ_{wall} in the case the normal condition (3.7) is applied.

The simplest strategy is to impose a free stress condition in the $\boldsymbol{\tau}$ direction:

$$(T(\mathbf{u},p;\mu)\mathbf{n})\cdot\boldsymbol{\tau} = \big(\mu\nabla\mathbf{u}\,\mathbf{n} - p\mathbf{n}\big)\cdot\boldsymbol{\tau} = 0 \qquad \text{on } \Gamma_{\text{wall}}\,. \tag{3.8}$$

This yields the so-called free-slip conditions, that is often used for two-phase flows [94]. However, as pointed out in [84], the results obtained using the free-slip condition lack vorticity near the boundary, producing a too weak shear stress layer. We will therefore consider this option as a reference to be improved.

A way to modulate between the free-slip and the no-slip conditions is to introduce a Robin condition, which reads:

$$\alpha\chi\mathbf{u}\cdot\boldsymbol{\tau} + (1-\alpha)\big(\mu\nabla u\,\mathbf{n} - p\mathbf{n}\big)\cdot\boldsymbol{\tau} = 0 \qquad \text{on } \Gamma_{\text{wall}} \tag{3.9}$$

where α is a dimensionless parameter to be estimated and χ is a dimensional number whose unit is Pa s/m. We let α vary in the domain: a small value $0 \le \alpha \ll 1$ near the interface creates a slip zone, while a large value $1 \ge \alpha \gg 0$ on the remaining part of Γ_{wall} mimics a no-slip condition. To allow this effect, we make α depend on the distance to the interface ϕ. Several transition functions can be employed to define $\alpha = \alpha(\phi(\mathbf{x},t))$. We choose a simple one, since the tests that we performed did not show a strong dependence on that function:

$$\alpha(d) = \begin{cases} 0 & \text{if } |d| \le l_s \\ \frac{K}{K+1} & \text{if } |d| > l_s \end{cases} \tag{3.10}$$

where K is taken large enough and l_s represents the slip length, i.e., the size of the band on each side of the interface within which the fluid is allowed to slip, see Fig. 3.3. The choice for the value $\frac{K}{K+1}$ yields a factor $\frac{\alpha}{1-\alpha} = K$ in the weak formulation (3.16), which allows to interpret

K as a penalty-like factor. Using this kind of boundary conditions adds a non-linearity to the problem considered.

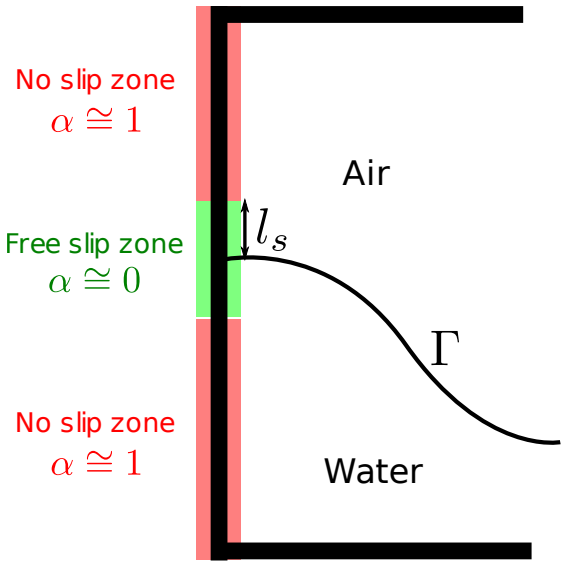

Figure 3.3: Illustration of the principle of the Robin-type boundary conditions.

3.2.5 Considered strategies

We will finally consider three different strategies of boundary conditions for Γ_{wall}, the no-slip condition $\mathbf{u} = \mathbf{0}$ being always imposed on $\Gamma_{\text{top}} \cup \Gamma_{\text{bottom}}$:

Strategy A: Horizontal condition (3.6) and free slip (3.8) in the direction $\boldsymbol{\tau} = \mathbf{e}_z$.

Strategy B: Normal condition (3.7) and free slip (3.8) in the two tangential directions $\boldsymbol{\tau}$ perpendicular to $\mathbf{n}$.

Strategy C: Normal condition (3.7) and Robin condition (3.9) in the two tangential directions $\boldsymbol{\tau}$ perpendicular to $\mathbf{n}$.

Combinations **A** and **B** are essentially devised for testing purposes and for comparison with results previously obtained in [28], while strategy **C** will represent our prefered choice for physically relevant simulations, as shown in chapter 5.

3.2.6 Boundary conditions for the level set

The level set might also require boundary conditions, depending on the fluid conditions used. Indeed, the PDE governing the evolution of the level set function is an hyperbolic transport equation, therefore an essential condition is required where the advection field points inwards,

i.e. on $\{\mathbf{x} \in \partial\Omega : \mathbf{u}(\mathbf{x}) \cdot \mathbf{n} < 0\}$ where $\mathbf{n}$ is the outward normal to $\partial\Omega$. However, with the choices proposed for the fluid boundary conditions (**A**, **B** and **C**), no such boundary condition is required.

3.3 Time discretization

To approximate the solution of problem (3.3), we introduce time steps $0 = t_0 < t_1 < \cdots < t_N = T$ with a uniform spacing $\Delta t = t_{n+1} - t_n$. This discretization is used for two purposes: to approximate the time derivatives appearing in (3.3) and to decouple the fluid dynamics from the interface advection.

We choose the backward Euler method for both the momentum and the level set advection equations for its simple setup and its stability properties. We use an explicit treatment of the density ρ and μ, which are evaluated at a previous time step and treat the non-linear convective term in the momentum equation semi-implicitly. If Robin type boundary conditions are used (strategy **C**), the values of the level set to compute α in (3.9) are also treated explicitly. This leads us to the following problem: for each $n > 0$, find $(\mathbf{u}^{(n)}, p^{(n)}, \phi^{(n)})$ such that

$$\left\{ \begin{array}{rl} \rho^{(n-1)}\left(\frac{1}{\Delta t}\mathbf{u}^{(n)} + (\mathbf{u}^{(n-1)} \cdot \nabla)\mathbf{u}^{(n)}\right) - \nabla \cdot \mathbf{T}(\mathbf{u}^{(n)}, p^{(n)}; \mu^{(n-1)}) & = \rho^{(n-1)}\mathbf{f}^{(n)} + \rho^{(n-1)}\frac{1}{\Delta t}\mathbf{u}^{(n-1)} \\ \nabla \cdot \mathbf{u}^{(n)} & = 0 \qquad \text{in } \Omega \\ \frac{1}{\Delta t}\phi^{(n)} + \mathbf{u}^{(n)} \cdot \nabla\phi^{(n)} & = \frac{1}{\Delta t}\phi^{(n-1)} \end{array} \right. \tag{3.11}$$

where $\rho^{(n-1)}$ and $\mu^{(n-1)}$ indicate that the density and the viscosity are evaluated using $\phi^{(n-1)}$ in the relations (3.4) and $\mathbf{f}^{(n)}(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t_n)$. Thanks to this discretization, two blocks have appeared, since the two first equations in (3.11) do not depend on $\phi^{(n)}$, but only on $\phi^{(n-1)}$. Therefore, we can solve first

$$\left\{ \begin{array}{rl} \rho^{(n-1)}\left(\frac{1}{\Delta t}\mathbf{u}^{(n)} + (\mathbf{u}^{(n-1)} \cdot \nabla)\mathbf{u}^{(n)}\right) - \nabla \cdot \mathbf{T}(\mathbf{u}^{(n)}, p^{(n)}; \mu^{(n-1)}) & = \rho^{(n-1)}\mathbf{f}^{(n)} + \rho^{(n-1)}\frac{1}{\Delta t}\mathbf{u}^{(n-1)} \\ \nabla \cdot \mathbf{u}^{(n)} & = 0 \qquad \text{in } \Omega \end{array} \right. \tag{3.12}$$

and use the newly computed $\mathbf{u}^{(n)}$ to solve then

$$\frac{1}{\Delta t}\phi^{(n)} + \mathbf{u}^{(n)} \cdot \nabla\phi^{(n)} = \frac{1}{\Delta t}\phi^{(n-1)} \quad \text{in } \Omega\,. \tag{3.13}$$

The rest of this section is dedicated to the approximation of the solution to these two linear problems.

3.4 Two-fluid solver

We need an accurate solver for the linearized Navier-Stokes equations (3.12) with discontinuous density and viscosity. Advances in that direction have recently been made, by taking into account the discontinuities naturally appearing in the velocity $\mathbf{u}$ and pressure p across the interface. Indeed, according to [62], we have (in case we do not consider surface tension):

$$\begin{aligned} [[\mathbf{u}]] &= \mathbf{0} \qquad \left[\left[\mu \frac{\partial \mathbf{u}}{\partial n}\right]\right] \cong \mathbf{0} \\ [[p]] &\cong 0 \quad \left[\left[\frac{\partial p}{\partial n}\right]\right] \cong [[\rho \mathbf{f}]] \cdot \mathbf{n} \end{aligned} \tag{3.14}$$

if the viscous effects do not dominate (the viscosity is sufficiently small). Sharp methods try somehow to capture these jumps. With the XFEM method, functions with jumps across the interface are added to the finite element basis, thus allowing the finite element space to capture the jumps (3.14) and thus yielding a better approximation (see [42] for an analysis of this method). The IFEM method modifies the basis functions of the finite element space so that the jumps (3.14) are directly satisfied when this modified space is used.

The drawbacks of the above mentioned methods rely on their high computational cost, since the approximation spaces are rebuilt each time the interface moves, and the additional efforts required to adapt existing codes to these new methods.

We identified the scalability of the method as a key aspect for our application. The scalability measures how well parallelization is used, by computing, e.g., the speed up obtained when doubling the number of processors involved (see Sect. 4.4 for more precise definitions). The load balancing aims at giving to each processor the same amount of work to perform, so that none remains idle while waiting for the others to complete their tasks. If the load balancing is suboptimal, the scalability is affected. In the case of the XFEM methods, the scalability is difficult to reach: since the interface moves, the different subdomains (distributed among the processors) receive different computational work at different time steps. A process for which many elements are crossed by the interface can have significantly more computations to do than processors whose subdomain is not crossed by the interface (see Fig. 3.4). The same considerations apply for adaptatively refined mesh or locally conforming mesh. Finally, the conditioning of the resulting linear system might suffer if the interface cuts elements close to their boundaries [109]. In this section, we will present an approach, that we described in [28], providing an enriched space for the pressure but still using the non-enriched finite element space for the discrete system, thus keeping the load balancing and the conditioning unaffected.

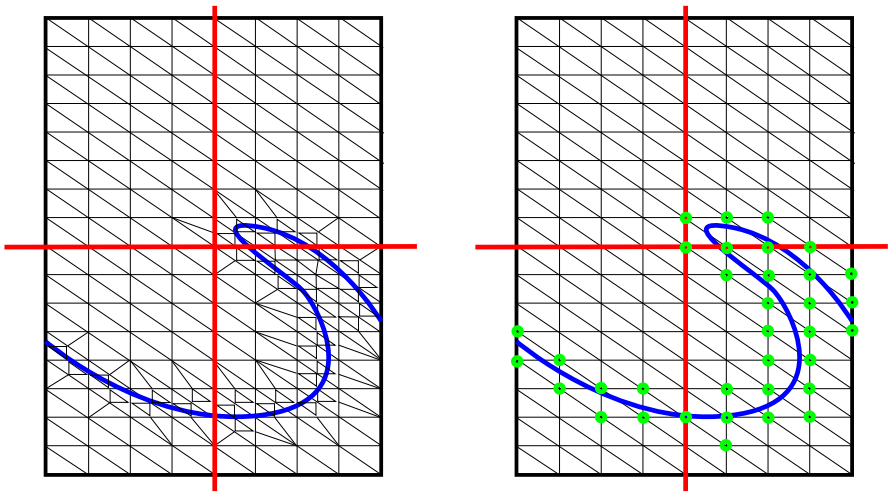

Figure 3.4: Illustration of the bad load balancing that can arise with different methods. A 1-level adapted mesh is presented on the left, while the XFEM is illustrated on the right (green circles represent additional degrees of freedom in case of a $\mathbb{P}_1$ approximation). The interface is the thick blue curvy line and the 4 subdomains are separated by the thick straight red lines.

3.4.1 Space discretization

We proceed with the space discretization of (3.12). Let us proceed formally and integrate (3.12) against suitable test functions

$$\left\{\begin{array}{rl} \int_\Omega \rho\left(\frac{\mathbf{u}^{(n)}}{\Delta t}+(\mathbf{u}^{(n-1)}\cdot\nabla)\mathbf{u}^{(n)}\right)\cdot\mathbf{v}+\int_\Omega(\mathbf{T}(\mathbf{u}^{(n)},p^{(n)};\mu)):\nabla\mathbf{v} & \\ -\int_{\partial\Omega}(\mathbf{T}(\mathbf{u}^{(n)},p^{(n)};\mu)\,\mathbf{n})\cdot\mathbf{v} & =\int_\Omega\rho\mathbf{f}^{(n)}\cdot\mathbf{v}+\int_\Omega\rho\frac{\mathbf{u}^{(n-1)}}{\Delta t}\cdot\mathbf{v} \\ \int_\Omega\nabla\cdot\mathbf{u}^{(n)}q & =0 \end{array}\right. \tag{3.15}$$

$\forall\mathbf{v}\in V, q\in Q$ where V and Q are suitably defined spaces as discussed afterwards. We can now use the boundary conditions for the fluid that we described in Sect. 3.2.2. On Γ_{top} and Γ_{bottom}, the boundary term vanishes because, as $\mathbf{u}=\mathbf{0}$, we take $\mathbf{v}=\mathbf{0}$ on $\Gamma_{\text{top}}\cup\Gamma_{\text{bottom}}$. On Γ_{wall}, the boundary term transforms differently depending on the boundary conditions chosen.

- With the strategy **A**, since $\mathbf{u}\cdot\mathbf{e}_x=0$ and $\mathbf{u}\cdot\mathbf{e}_y=0$ on Γ_{wall} and so the same holds for $\mathbf{v}$, this term yields zero contribution in the x and y directions. On the z direction, we have the condition (3.8) so the boundary term in (3.15) vanishes completely. We define then the spaces $V=\{\mathbf{v}\in(H^1(\Omega))^3\ :\mathbf{v}=\mathbf{0}$ on $\Gamma_{\text{top}}\cup\Gamma_{\text{bottom}}$ and $\mathbf{v}\cdot\mathbf{e}_x=0\,,\mathbf{v}\cdot\mathbf{e}_y=0$ on $\Gamma_{\text{wall}}\}$ and $Q=\{q\in L^2(\Omega)\ :\ \int_\Omega q=0\}$.
- With the strategy **B**, we have $\mathbf{u}\cdot\mathbf{n}=0$ on Γ_{wall}, so we take a test function $\mathbf{v}$ such that $\mathbf{v}\cdot\mathbf{n}=0$ on Γ_{wall}. With the condition (3.8) in the two tangent directions, the boundary term in (3.15) vanishes again but $V=\{\mathbf{v}\in(H^1(\Omega))^3\ :\mathbf{v}=\mathbf{0}$ on $\Gamma_{\text{top}}\cup\Gamma_{\text{bottom}}$ and $\mathbf{v}\cdot\mathbf{n}=0$ on $\Gamma_{\text{wall}}\}$ and $Q=\{q\in L^2(\Omega)\ :\ \int_\Omega q=0\}$.
- Finally, with the strategy **C**, we take $\mathbf{v}\cdot\mathbf{n}=0$ on Γ_{wall} as normal conditions are strongly

applied on $\mathbf{u}$. The spaces are $V = \{\mathbf{v} \in (H^1(\Omega))^3 : \mathbf{v} = \mathbf{0} \text{ on } \Gamma_{\text{top}} \cup \Gamma_{\text{bottom}} \text{ and } \mathbf{v}\cdot\mathbf{n} = 0 \text{ on } \Gamma_{\text{wall}}\}$ and $Q = \{q \in L^2(\Omega) : \int_\Omega q = 0\}$.

The problem reads then: find $\mathbf{u}^{(n)} \in V$ and $p^{(n)} \in Q$ such that, for all $\mathbf{v} \in V$ and $q \in Q$, we have

$$\begin{cases} a(\mathbf{u}^{(n)}, \mathbf{v}; \mathbf{u}^{(n-1)}) - b(\mathbf{v}, p^{(n)}) + c(\mathbf{u}^{(n)}, \mathbf{v}) &= F(\mathbf{v}; \mathbf{u}^{(n-1)}) \\ b(\mathbf{u}^{(n)}, q) &= 0 \end{cases} \tag{3.16}$$

where

$$a(\mathbf{u}, \mathbf{v}; \mathbf{w}) = \int_\Omega \rho \left(\frac{\mathbf{u}}{\Delta t} + (\mathbf{w}\cdot\nabla)\mathbf{u} \right) \cdot \mathbf{v} + \mu \nabla\mathbf{u} : \nabla\mathbf{v}$$

$$b(\mathbf{u}, p) = \int_\Omega (\nabla\cdot\mathbf{u}) p$$

$$F(\mathbf{v}; \mathbf{w}) = \int_\Omega \rho \mathbf{f}\cdot\mathbf{v} + \rho \frac{\mathbf{w}}{\Delta t}\cdot\mathbf{v}$$

$$c(\mathbf{u}, \mathbf{v}) = \begin{cases} 0 & \text{for strategies } \mathbf{A} \text{ and } \mathbf{B} \\ \sum_{i=1}^2 \int_{\Gamma_{\text{wall}}} \frac{\alpha}{1-\alpha} (\mathbf{u}\cdot\boldsymbol{\tau}_i)(\mathbf{v}\cdot\boldsymbol{\tau}_i) & \text{for strategy } \mathbf{C} \end{cases}$$

with $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$ two arbitrary orthogonal tangential directions.

For the space discretization, a mesh $\boldsymbol{\tau}_h$, made of tetrahedra $\boldsymbol{\tau}_h = \{K_i\}_i$, is built on the domain Ω. We define then the finite element spaces for the approximation of $\mathbf{u}$ and p. In [28], we used a finite element space with bubble stabilization for the velocity to fulfill the inf-sup condition (see [79]), but this approach turned out to be too computationally expensive, because of the extra load of 3D degrees of freedom at the barycenter of each tetrahedra that are necessary to ensure the fulfilment of the inf-sup condition. Indeed, on the mesh presented in [28] (which corresponds to the XS mesh hereafter), the velocity space contains around 180′000 degrees of freedom and among them, 150′000 those associated to the bubbles. We decided to adopt full linear elements, with a stabilization that serves also the purpose of stabilizing the convective term in case of convection-dominated problems. Thus, we look for $\mathbf{u}_h^{(n)}$ approximating $\mathbf{u}^{(n)}$ in the space

$$V_h = \left\{ \mathbf{v} \in V \cap \left(C^0(\overline{\Omega}) \right)^3 : \mathbf{v}|_{K_i} \in (\mathbb{P}_1(K_i))^3 \ \forall K_i \in \boldsymbol{\tau}_h \right\} \tag{3.17}$$

and $p_h^{(n)}$ approximating $p^{(n)}$ in the space

$$Q_h = \left\{ \{q \in Q \cap C^0(\overline{\Omega}) : q|_{K_i} \in \mathbb{P}_1(K_i) \ \forall K_i \in \boldsymbol{\tau}_h \right\} . \tag{3.18}$$

Several stabilizations exist for the Navier-Stokes equations, see e.g. [10] for a recent review. The SUPG/PSPG stabilization developed in [12] is used in this work. With this stabilization, the discrete problem reads: find $\mathbf{u}_h^{(n)} \in V_h$ and $p_h^{(n)} \in Q_h$ such that for all $\mathbf{v}_h \in V_h$ and $q_h \in Q_h$,

we have

$$\begin{cases} a_h(\mathbf{u}_h^{(n)},\mathbf{v}_h;\mathbf{u}_h^{(n-1)}) - b_h^u(\mathbf{v}_h,p_h^{(n)};\mathbf{u}_h^{(n-1)}) + c(\mathbf{u}_h^{(n)},\mathbf{v}_h) &= F_h(\mathbf{v}_h;\mathbf{u}_h^{(n-1)}) \\ b_h^p(\mathbf{u}_h^{(n)},q_h;\mathbf{u}_h^{(n-1)}) + j_h(p_h^{(n)},q_h) &= l_h(q_h) \end{cases} \tag{3.19}$$

where the different forms are defined by:

$$\begin{aligned} a_h(\mathbf{u},\mathbf{v};\mathbf{w}) =& a(\mathbf{u},\mathbf{v};\mathbf{w}) + \sum_K \int_K \delta_{\text{div}}(\nabla\cdot\mathbf{u})(\nabla\cdot\mathbf{v}) \\ &+ \sum_K \int_K \delta_{\text{SUPG}}\rho\left(\frac{\mathbf{u}}{\Delta t} + (\mathbf{w}\cdot\nabla)\mathbf{u}\right)\cdot((\mathbf{w}\cdot\nabla)\mathbf{v}) \end{aligned}$$

$$b_h^u(\mathbf{v},p;\mathbf{w}) = b(\mathbf{v},p) - \sum_K \int_K \delta_{\text{SUPG}}\nabla p\cdot((\mathbf{w}\cdot\nabla)\mathbf{v})$$

$$b_h^p(\mathbf{u},q;\mathbf{w}) = b(\mathbf{u},q) + \sum_K \int_K \delta_{\text{PSPG}}\rho\left(\frac{\mathbf{u}}{\Delta t} + (\mathbf{w}\cdot\nabla)\mathbf{u}\right)\cdot\nabla q$$

$$j_h(p,q) = \sum_K \int_K \delta_{\text{PSPG}}\nabla p\cdot\nabla q$$

$$F_h(\mathbf{v};\mathbf{w}) = F(\mathbf{v}) - \sum_K \int_K \delta_{\text{SUPG}}\rho\left(\mathbf{f} + \frac{\mathbf{w}}{\Delta t}\right)\cdot((\mathbf{w}\cdot\nabla)\mathbf{v})$$

$$l_h(q;\mathbf{w}) = \sum_K \int_K \delta_{\text{PSPG}}\rho\left(\mathbf{f} + \frac{\mathbf{w}}{\Delta t}\right)\cdot\nabla q$$

where δ_{div}, δ_{SUPG} and δ_{PSPG} are suitable coefficients. In this formulation, we can distinguish three components of the stabilizing term that have been added with respect to (3.19):

- The term in $(\nabla\cdot\mathbf{u})(\nabla\cdot\mathbf{v})$ helps enforcing the divergence free constraint in the high Reynolds regimes;
- The terms containing the coefficient δ_{SUPG} stabilize the convection dominated cases, i.e., when the Reynolds is high;
- The terms containing the coefficient δ_{PSPG} ensures that the inf-sup condition is satisfied for our choice of finite element spaces V_h and Q_h.

All these new terms are consistent since they contain the residual of one of the equations (3.12) (the second derivative yields no contribution with the piecewise linear elements). Many criteria have been proposed in the literature on how to choose the coefficients δ_{div}, δ_{SUPG} and δ_{PSPG} [10, 1]. In this work we use the following simple formulas:

$$\begin{aligned} \delta_{\text{div}} &= \alpha_{\text{div}} h_K \\ \delta_{\text{SUPG}} &= \alpha_{\text{SUPG}} \frac{h_K}{|\mathbf{w}|} \\ \delta_{\text{PSPG}} &= \alpha_{\text{PSPG}} h_K \end{aligned} \tag{3.20}$$

where α_{div}, α_{SUPG} and α_{PSPG} are three real constants to be chosen. Remark that there is no problem with using a small velocity ($|\mathbf{w}| \sim 0$) since the coefficient δ_{SUPG} always appears with the term $(\mathbf{w}\cdot\nabla)\mathbf{v}$; the term $\delta_{\text{SUPG}}(\mathbf{w}\cdot\nabla)\mathbf{v}$ behaves well for any value of $\mathbf{w}$, excepted for $\mathbf{w}=\mathbf{0}$, in which case we set $\delta_{\text{SUPG}}=0$. We also remark that we used different values for δ_{SUPG} and δ_{PSPG}, although they should be taken equal for the theoretical proof of stability (see [10]): in practice, for our two-phase flow, several values for α_{div}, α_{SUPG} and α_{PSPG} were tried until a stable scheme was found.

Remark 3.4.1. *The coefficients α_{div}, α_{SUPG} and α_{PSPG} are not adimensional. This calls for a better tuning of these parameters depending on the viscosity, density and time step.*

Remark 3.4.2. *At the discrete level, we use the formulation $\int_\Omega \mu\nabla\mathbf{u}:\nabla\mathbf{v}$ instead of the more natural $\int_\Omega \mu(\nabla\mathbf{u}+\nabla\mathbf{u}^T):\nabla\mathbf{v}$. Indeed, the only difference between the two formulations is the way natural conditions and interface conditions are retrieved from our weak formulation. However, we did not observe any significant difference in our simulations associated to the two formulations, because of the low viscosities used. We preferred the first formulation which yields a block diagonal structure for the momentum equation, reducing therefore the computational cost of our algorithm.*

3.4.2 Pressure correction via the SESIC method

Since we do not deal with surface tension, the only non-null interface jump condition in (3.14) is on the gradient of the pressure. It has been highlighted in [22] that capturing the jump of the gradient of the pressure can significantly improve the quality of the simulation by suppressing spurious velocities in the vicinity of the interface. The method that we propose stems from the SESIC method devised in chapter 2. It consists in exploiting the fact that the jump in the pressure gradient is known [62]:

$$\left[\!\left[\frac{\partial p}{\partial n}\right]\!\right] = (\rho_l-\rho_a)\mathbf{f}\cdot\mathbf{n} = g_n\,, \tag{3.21}$$

to build a lifting L_p that captures this singularity. This jump is due to the discontinuity of the right hand side $\rho\mathbf{f}$ in (3.3). We remark that if only the gravity is taken into account and the fluid does not move, we recover the well-known jump in the gradient of the hydrostatic pressure, and if no external force applies to the fluid, the gradient of the pressure is continuous across the interface Γ.

We first note that the function g_n can be extended naturally to a function $\bar{g}_n = [\![\rho]\!]\mathbf{f}\cdot\nabla\phi$ defined on Ω by remarking that $\mathbf{n}=\nabla\phi$, ϕ being the level set function. We define then the auxiliary discrete function

$$L_p(\mathbf{x}) = \begin{cases} \sum_{\phi_i<0}(\bar{g}_n(\mathbf{x}_i)\phi(\mathbf{x}_i))\varphi_i(\mathbf{x}) & \text{if } \phi(\mathbf{x})\geq 0 \\ -\sum_{\phi_i\geq 0}(\bar{g}_n(\mathbf{x}_i)\phi(\mathbf{x}_i))\varphi_i(\mathbf{x}) & \text{if } \phi(\mathbf{x})<0 \end{cases} \tag{3.22}$$

where $\mathbf{x}_i$ is the position of the node associated to the finite element basis function φ_i of the

discrete pressure space P_h and $\phi_i = \phi(\mathbf{x}_i)$. The idea behind this formula is that the jump of L_p approximates the ones of the pressure p:

$$[[L_p]] = \sum_i (\bar{g}_n(\mathbf{x}_i)\phi(\mathbf{x}_i))\varphi_i(\mathbf{x}) \cong 0 \tag{3.23}$$

as $(\sum_i \phi(\mathbf{x}_i)\varphi_i(\mathbf{x}))|_\Gamma = \phi|_\Gamma = 0$ and

$$\left[\left[\frac{\partial L_p}{\partial n}\right]\right] = \frac{\partial}{\partial n}\sum_i (\bar{g}_n(\mathbf{x}_i)\phi(\mathbf{x}_i))\varphi_i(\mathbf{x}) \cong g_n. \tag{3.24}$$

This lifting construction is valid if $\phi \in C^1(\Omega)$, because we need to evaluate its gradient in the finite element nodes to get $\bar{g}_n(\mathbf{x}_i)$ in (3.22). However, with our choice of space L_h, this does not hold. To overcome this issue, we replace $\nabla\phi$ by a $C^0(\Omega)$ reconstruction $Gr_h(\phi)$, as proposed in [108]. The extended function $\bar{g}_n$ that has been used for the simulations is then defined as

$$\bar{g}_n = [[\rho\mathbf{f}]] \cdot \frac{Gr_h(\phi)}{|Gr_h(\phi)|} \tag{3.25}$$

and L_p defined with this new choice.

After being constructed, the lifting is subtracted from the pressure in order to identify a pressure $\hat{p} = p - L_p$, which satisfies

$$[[\hat{p}]] = [[p]] - [[L_p]] \cong 0 \tag{3.26}$$

and

$$\left[\left[\frac{\partial \hat{p}}{\partial n}\right]\right] = \left[\left[\frac{\partial p}{\partial n}\right]\right] - \left[\left[\frac{\partial L_p}{\partial n}\right]\right] \cong 0. \tag{3.27}$$

thanks to (3.23) and (3.24). We can now rewrite the discrete problem (3.19) in terms of the unknowns $(\mathbf{u}_h^{(n)}, \hat{p}_h^{(n)})$:

$$\left\{\begin{array}{lll} & a_h(\mathbf{u}_h^{(n)}, \mathbf{v}_h; \mathbf{u}_h^{(n-1)}) - b_h^u(\mathbf{v}_h, \hat{p}_h^{(n)}; \mathbf{u}_h^{(n-1)}) + c(\mathbf{u}_h^{(n)}, \mathbf{v}_h) \\ = & F_h(\mathbf{v}_h; \mathbf{u}_h^{(n-1)}) + b_h^u(\mathbf{v}_h, L_p; \mathbf{u}_h^{(n-1)}) \\ & b_h^p(\mathbf{u}_h^{(n)}, q_h; \mathbf{u}_h^{(n-1)}) + j_h(p_h^{(n)}, q_h) \\ = & l_h(q_h) - j_h(L_p, q_h) \end{array}\right. . \tag{3.28}$$

Remark that the lifting L_p must be recomputed at each time step, since the interface changes and so does ϕ. We can observe that the discrete formulation (3.28) yields exactly the same left hand side as the non-corrected formulation (3.19). Therefore, the matrix of the system is not changed with respect to the scheme without pressure correction.

This method shares some similarities with the one proposed in [22], in the sense that the final linear system is solved only for the unknowns from V_h and Q_h and not in an enriched space, as would be the case for XFEM based methods. Indeed, in [22], bubbles are added in the pressure

space near the interface and condensed a priori. The condensed pressure is then put to the right hand side and it can therefore be seen as the analogue of our lifting. The difference lies in the fact that our approach allows cheaper computations (as the lifting is explicitly known) and it is easier to incorporate in standard finite element codes.

3.4.3 Numerical integration

Another important ingredient for accurate simulations of two phase flows with different physical characteristics is the numerical integration of the singular integrals. Indeed, several integrals with discontinuous integrand appear in the discrete formulation (3.28), because the density ρ and the viscosity μ are discontinuous across Γ, but also because L_p is not ensured to be continuous across Γ and has a jump in the gradient, i.e., a kink. Using standard Gaussian quadratures can lead to substantial errors in the integration of these terms and can deteriorate the whole simulation, see [22, 28] but also the test in Sect. 4.2.

This problem can be overcome by resorting to quadrature rules that are adapted to the interface: in each element crossed by the interface, the position of the interface is computed by finding its intersection with the edges of the tetrahedron and a quadrature rule is devised on each side by combining a few local quadrature rules (see Fig. 3.5).

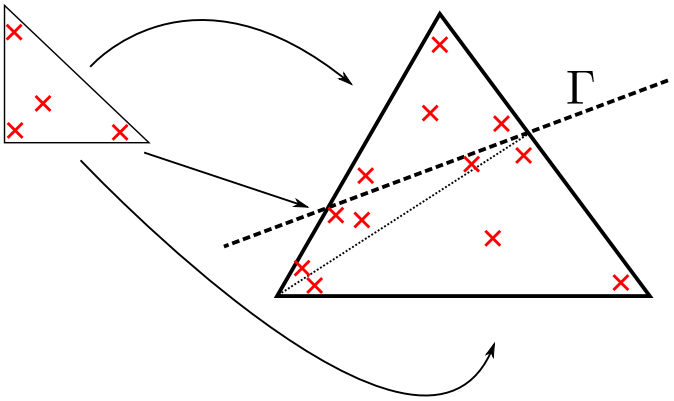

Figure 3.5: The quadrature rule in a triangle (bold lines) cut by the interface Γ is built by triangulating (dotted line) both sides of the triangle, associating a quadrature rule to each of the triangles (arrows) and putting all the resulting quadrature points (crosses) together.

Other strategies are possible, see Sect. 2.4.2.

3.4.4 Discrete normal boundary condition and correction

At the discrete level, the imposition of the normal condition (3.7) is not straightforward. Indeed, the normal $\mathbf{n}$ must be defined nodally which is not trivial as the normal is not uniquely defined on the vertices of the mesh. Three approaches can be adopted to compute the normals:

1. Use the normal that is defined by the exact geometry, which is easy to compute in the case of a cylindrical domain, but might be more demanding for other geometries.

2. Compute the normals of the faces neighboring the vertex and average them (e.g., with weights corresponding to the relative face area).

3. Compute the normal in a way consistent with the divergence free constraint, as proposed in [32].

In our tests, we could not see significant differences between these three possibilities, probably because of the simple geometry of the domain and the structured nature of the meshes employed. Another difficulty is the way to actually impose the condition (3.7). We adopt the method using rotations of the reference frame as proposed in [32].

Independently of this choice, the strong imposition of the normal velocity might create spurious velocities due to the fact that the normal imposed does not coincide with the normal to the faces, as first observed in [7]. To overcome this issue, it was proposed in [7] to assemble a corrective term on the left hand side, which does not make sense at the continuous level but cancels the spurious velocity at the discrete level. We consider however the correction proposed in [23] which involves the correction only in the normal direction. The correction term to be assembled on the left hand side reads

$$-\int_{\Gamma_{\text{wall}}} \left((\mu \nabla \mathbf{u}_h^{(n)}\, \mathbf{n}) \cdot \mathbf{n} - \hat{p}_h^{(n)}\right)(\mathbf{v}_h \cdot \mathbf{n}) \tag{3.29}$$

and a similar term is put on the right hand side for the pressure correction

$$-\int_{\Gamma_{\text{wall}}} L_p(\mathbf{v}_h \cdot \mathbf{n})\ . \tag{3.30}$$

Remark that after assembling this term, the normal condition is still imposed strongly. This correction can be used for both strategies **B** and **C** concerning the choice of the boundary conditions. If the boundary is straight, in which case the normal computed at the nodes and the normal of the faces coincide, the correction term is completely removed by the essential imposition of the normal.

3.5 Level set solver

From the coupled equation (3.3), the level set block can be extracted directly since at the time it must be computed, the velocity of the fluid is already known. The discretization of this PDE is the subject of this section. Moreover, at the numerical level, our scheme contains two additional operations that are performed on the level set function in order to obtain physically relevant results, which we will also describe here: the reinitialization of the level set function and a correction to improve the mass conservation.

We will avoid Narrow Band techniques [19, 90], which can be used to fasten the computations by considering the level set evolution only in a narrow subset of the domain Ω around the interface Γ. Indeed, we need the information of the distance to the interface not only close to it to devise the wall boundary conditions **C** for the fluid problem (see Sect. 3.2.2).

3.5.1 Interface evolution

At each time step, after the velocity and the pressure have been determined from the Navier-Stokes equations (3.12), the interface must be moved according to the state of the fluid. This is achieved through the advection of the level set function with the velocity of the fluid at that time, as in (3.13).

Space discretization

As for the space discretization, we use the same mesh τ_h used to solve the fluid dynamics in order to avoid interpolation or projection of the different quantities. Continuous linear finite element are used for the space discretization, i.e., the discrete space L_h in which we look for the approximation $\phi_h^{(n)}$ of $\phi^{(n)}$ is defined as

$$L_h = \left\{ \psi_h \in H^1(\Omega) \cap C^0(\bar{\Omega}) \ : \ \psi_h|_K \in \mathbb{P}_1 \ \forall K \in \tau_h \right\} . \tag{3.31}$$

The PDE governing the evolution of the level set function (3.13) is an hyperbolic advection equation and it is well known that numerical instabilities can appear (see, e.g., [82]). A SUPG stabilization is then added, leading to the following weak formulation: find $\phi_h^{(n)} \in L_h$

$$a_\phi(\phi_h^{(n)}, \psi_h; \mathbf{u}^{(n)}) + j_\phi(\phi_h^{(n)}, \psi_h; \mathbf{u}^{(n)}) = f_\phi(\psi_h; \phi_h^{(n-1)}) + k_\phi(\psi_h; \mathbf{u}^{(n)}, \phi_h^{(n-1)}) \tag{3.32}$$

$\forall \psi_h \in L_h$, with $a_\phi(\cdot,\cdot)$ and $f_\phi(\cdot)$ representing the bilinear and linear forms from the Galerkin method, $j_\phi(\cdot,\cdot)$ and $k_\phi(\cdot)$ being the stabilization terms:

$$a_\phi(\phi, \psi; \mathbf{u}) = \int_\Omega \left(\frac{1}{\Delta t}\phi + \mathbf{u} \cdot \nabla \phi \right) \psi \tag{3.33}$$

$$f_\phi(\psi; \phi) = \int_\Omega \frac{1}{\Delta t} \phi \psi \tag{3.34}$$

$$j_\phi(\phi, \psi; \mathbf{u}) = \sum_K \int_K \gamma_{\text{SUPG}}^\phi \left(\frac{1}{\Delta t}\phi + \mathbf{u} \cdot \nabla \phi \right) \left(\mathbf{u} \cdot \nabla \psi \right) \tag{3.35}$$

$$f_\phi(\psi; \mathbf{u}, \phi) = \sum_K \int_K \gamma_{\text{SUPG}}^\phi \left(\frac{1}{\Delta t}\phi \right) \left(\mathbf{u} \cdot \nabla \psi \right) \tag{3.36}$$

where $\gamma^{\phi}_{\text{SUPG}}$ is a coefficient, depending on $\mathbf{u}$, that is used to weight the stabilization. Several choices are available in the literature, see e.g. [57] for an overview and recent developments. We have opted for the classical definition:

$$\gamma^{\phi}_{\text{SUPG}} = \frac{h_K}{2||\mathbf{u}||_2} \tag{3.37}$$

where h_K represents the diameter of the tetrahedron K. With this choice of parameter, the scheme was analyzed in [15] where it was proven that under the condition

$$\frac{h_K}{||\mathbf{u}||_2} > \Delta t > \frac{h_K^2}{4||\mathbf{u}||_2} \tag{3.38}$$

the scheme is stable and converges with quasi-optimal rates.

Remark 3.5.1. *Another possibility is to use the interior penalty stabilization as in [28], which yields better mass conservation properties [26]. However, the larger stencil associated with the interior penalty term makes preconditioners computationally more expensive and the assembly less suitable to parallel computations.*

3.5.2 Reinitialization

One issue with the level set method is that while it is being advected, the level set function ϕ loses its property of being a distance function, i.e. $|\nabla\phi| = 1$. This is preferable to avoid since too large or too small gradients indicate a steep or flat function, whose zero is less accurately tracked. We also need an estimate of the distance to the interface for the boundary condition (3.9), so a gradient close to 1 is necessary to obtain a reliable estimate.

A reinitialization procedure (also called redistancing or reparametrization procedure) must be devised to recover the distance property. An ideal reinitialization procedure should possess to following properties:

- The mass of each phase should not change with the reinitialization;
- The interface should not be moved from its position;
- It should yield a good scalability;
- The reinitialized functions should be close to a distance function.

The two first points are rather related to the behavior of the procedure in the set of those tetrahedra crossed by the interface Γ (called Ω_Γ) while the two last are more linked to $\Omega \setminus \Omega_\Gamma$. This is reflected in the most recent reinitialization algorithm, which marry a precise method for Ω_Γ with a fast "far field" method.

Far field methods usually consist in methods that were originally designed as a whole reinitializations and are usually split in two categories, non-PDE-based and PDE-based. The most

common non PDE-based method is probably the Fast Marching Method (FMM, [90]), which consists in computing an approximated distance to the interface using the geometric structure of the mesh. Its complexity is optimal but the parallel implementations that have been proposed [47, 100] do not scale to a very high number of processors. It seems that the time for the information to travel from one side of the domain to another one in the worst case scenario limits the scalability: they miss what would be the analoguous to the coarse grid approximation for Schwarz type preconditioners [83]. A parallel implementation of another non PDE-based method has been proposed in [36], but its precision and speed depends also on a parameter to be chosen and the scability is not clear. PDE-based methods rely on the resolution for φ, a function evolving from ϕ to a reinitialized function, of the following Hamilton-Jacobi equation

$$\partial_\tau \varphi + S(\phi)(|\nabla\varphi| - 1) = 0 \qquad \text{in } \Omega \tag{3.39}$$

where τ is a pseudo-time, S denotes the sign function and ϕ is the function to reinitialize. The initial condition is given by $\varphi(\tau = 0) = \phi$ and the reinitialized function $\tilde{\phi}$ by $\tilde{\phi} = \varphi(\tau \to \infty)$. At the numerical level, the equation (3.39) is linearized and solved for a sufficiently long time $\mathcal{T}$ (to be determined). The linearized PDE reads

$$\partial_\tau \varphi + \beta \cdot \nabla\varphi = S(\phi) \tag{3.40}$$

where β is a suitable approximation of $S(\phi)\varphi/|\nabla\varphi|$. Equation (3.40) is an hyperbolic transport equation, which needs a particular care at the numerical level. First of all, the sign function must be smoothed across the interface Γ [90]. Then, a numerical stabilization must be added if one uses finite elements, e.g. SUPG terms to deal with the convection dominated cases and additonal diffusion for the interface (where $\beta = 0$) [98]. The final scheme is difficult to set up correctly, as remarked in [41]: sufficient diffusion must be added to obtain a stable scheme, but the more diffusion is added, the worse is the mass conservation. There is also a quite large number of parameters to tune. In spite of these drawbacks, this method has good scalability properties as it is inherited from the numerical scheme for solving the linearized PDEs.

Methods for reinitializing in Ω_Γ have recently been introduced. The interpolation using cubic polynomials introduced in [20] was probably the first method of this kind. After the level set function has been interpolated with C^1 cubic functions, a Newton solver is used to compute the distance to the zero level set of the interpolant. It has been used in conjunction with FMM for the far field, e.g., in [87]. Another method for redistancing in the domain Ω_Γ in a finite element framework is the interface local projection proposed in [74] and used e.g. in [106]. It consists in a $L^2(\Omega_\Gamma)$ projection of the level set function normalized by the norm of its gradient:

$$\int_{\Omega_\Gamma} \varphi\psi_h = \int_{\Omega_\Gamma} \frac{\phi}{|\nabla\phi|}\psi_h \quad \forall \psi_h \in L_h \,. \tag{3.41}$$

Numerical scheme

In terms of possible scalability and ease of implementation, the most suitable method is the interface local projection combined with a Hamilton-Jacobi-based far field reinitialization. There are however two drawbacks to this approach. First of all, the local projection step adds a PDE with respect to purely Hamilton-Jacobi-based reinitalization, which makes it more expensive. Secondly, the PDEs to be solved are not defined on Ω, but on subsets of it. To get good scalability, one needs then to either rebalance the partitioning of the mesh or deal with a suboptimal load balancing. To overcome these two issues, the method that we propose merges the two steps (interface local projection and Hamilton-Jacobi equation) into a single step.

The idea to realize the merge is to consider the interaction of the two steps. The Hamilton-Jacobi equation takes as essential boundary condition on $\partial\Omega_\Gamma$ (Ω_Γ denoting the elements crossed by Γ) the solution of the interface local projection. Instead of imposing these values nodally, one could use volume penalization techniques, such as proposed in [68]. To merge the two steps, one can penalize the solution against the level set function normalized by its gradient instead of penalizing it against the projected function. More in details, the scheme that we propose consists in solving a modified Hamilton-Jacobi equation incorporating the interface local projection. We consider a (pseudo-) time discretization $0 = \tau_0 < \tau_1 < \cdots < \tau_M = \mathcal{T}$ with a fixed pseudo time step $\Delta\tau = \tau_m - \tau_{m-1}$. The discrete formulation of this modified Hamilton-Jacobi equation reads: find $\varphi_h^{(m)} \in L_h$ such that for all $\psi_h \in L_h$,

$$r(\varphi_h^{(m)}, \psi_h; \beta) + r_\Gamma(\varphi_h^{(m)}, \psi_h) = s(\psi_h; \varphi_h^{(m-1)}) + s_\Gamma(\psi_h) \tag{3.42}$$

where $\beta = S(\phi)\frac{\varphi_h^{(m-1)}}{|\nabla\varphi_h^{(m-1)}|}$ and the different bilinear and linear forms are defined as follows. The forms $r(\cdot,\cdot)$ and $s(\cdot)$ corresponds to the Hamilton-Jacobi equation, discretized in time with a backward Euler scheme and supplemented with suitable stabilization terms $j(\cdot,\cdot)$ and $k(\cdot)$:

$$r(\varphi,\psi) = \int_{\Omega\setminus\Omega_\Gamma} \frac{1}{\Delta\tau}\varphi\psi + (\beta\cdot\nabla\varphi)\psi + j(\varphi,\psi) \tag{3.43}$$

$$s(\psi;\varphi) = \int_{\Omega\setminus\Omega_\Gamma} \frac{1}{\Delta\tau}\varphi\psi + S(\phi)\psi + k(\psi)\,. \tag{3.44}$$

Remark that in this bilinear form, the integrals are performed on $\Omega\setminus\Omega_\Gamma$, the sign function $S(\cdot)$ is constant in each connected component of $\Omega\setminus\Omega_\Gamma$ and there is no need to smooth it. The forms $r_\Gamma(\cdot,\cdot)$ and $s_\Gamma(\cdot)$ define the problem in Ω_Γ as the L^2-projection of the function to reinitialized divided by its gradient:

$$r_\Gamma(\varphi,\psi) = K_{\text{penal}}\int_{\Omega_\Gamma}\varphi\psi \tag{3.45}$$

$$s_\Gamma(\psi) = K_{\text{penal}} \int_{\Omega_\Gamma} \frac{\phi}{|\nabla\phi|}\psi \tag{3.46}$$

and K_{penal} is a penalization parameter, that must be chosen large enough to keep the interface fixed and allow a good reinitialization near the interface.

Concerning the stabilization terms $j(\cdot,\cdot)$ and $k(\cdot)$ that we introduced in (3.43) for the Hamilton-Jacobi equation, a natural choice would be to reuse a SUPG stabilization, but this turns out to be a bad choice. Indeed, the role of the Hamilton-Jacobi equation is to reach a solution where $|\nabla\phi| = 1$ with $\Gamma = 0$, but for a given interface Γ, there might exist several such functions (see Fig. 3.6). The correct solution is given by the *viscosity solution* [90] of the Hamiton-Jacobi

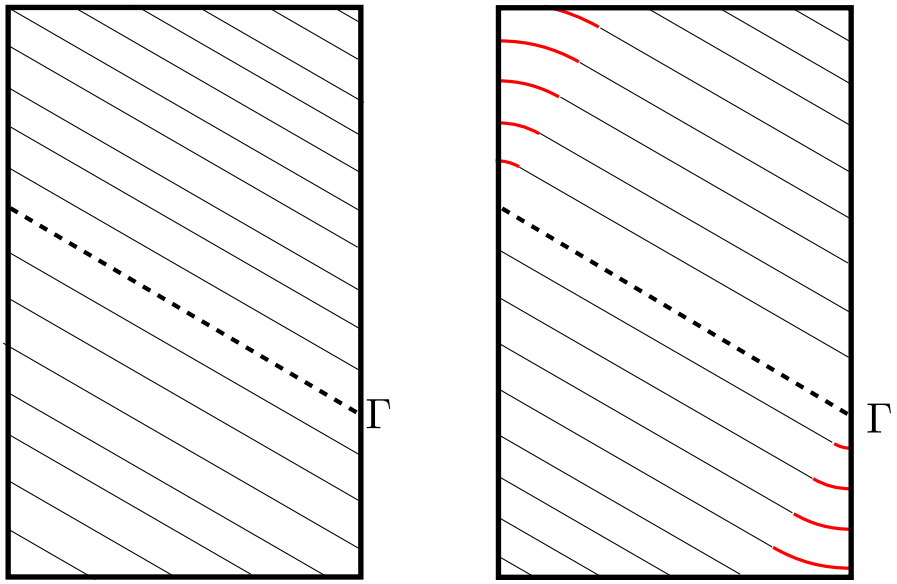

Figure 3.6: Schematic representation of two functions satisfying $|\nabla\phi| = 1$ and $\phi(\mathbf{x}) = 0\ \forall \mathbf{x} \in \Gamma$. Several level sets of the two functions are displayed to represent them; the differences are in the upper left and lower right corners (in red). The solution on the right represents the actual distance to the interface Γ (and the viscosity solution).

equation. Unfortunately, the SUPG stabilization does not help in chosing that particular solution (since it is residual based). From the properties of the viscosity solutions, it is possible to understand that a better stabilization is represented by the artificial viscosity. However, it is well-known that the artificial diffusion can lead to solutions that are far from the exact one when the mesh is not fine enough (see tests in Sect. 4.1). We propose therefore to mix the artificial diffusion and the SUPG scheme, so that both have distinct roles: SUPG stabilizes the convective dominated equation while the artificial diffusion selects the right solution. The form $j(\cdot,\cdot)$ and $k(\cdot)$ are then defined as

$$j(\varphi,\psi) = \sum_{K\subset\Omega\setminus\Omega_\Gamma} \int_K \gamma_{\text{HJ,Diff}} h_K \nabla\varphi\cdot\nabla\psi + \gamma_{\text{HJ,SUPG}} h_k \left(\frac{1}{\Delta\tau}\varphi + \beta\cdot\nabla\varphi\right)(\beta\cdot\nabla\psi) \tag{3.47}$$

$$k(\psi) = \sum_{K \subset \Omega \setminus \Omega_\Gamma} \int_K \gamma_{\text{HJ,SUPG}} h_k \left(\frac{1}{\Delta\tau}\varphi + S(\phi)\right)(\beta \cdot \nabla\psi) \tag{3.48}$$

where $\gamma_{\text{HJ,Diff}}$ and $\gamma_{\text{HJ,SUPG}}$ are stabilization parameters.

Finally, this reinitalization scheme constains 5 parameters:

- $\mathcal{T}$ the final time can be chosen large enough (but not too much to avoid making too many time steps);
- $\Delta\tau$ must be small enough to yield a stable scheme but still quite large to improve the range of the reinitialization;
- K_{penal} has to be chosen quite large to penalize the system. The only thing that would prevent the usage of an extremly large value for K_{penal} is the conditioning of the resulting matrix.
- $\gamma_{\text{HJ,Diff}}$ settles the amount of artificial diffusion added in the far field. A value larger than the optimal value does not harm too much in the sense that this artificial diffusion is not added near the interface, but it should still be kept as low as possible to maintain a better accuracy.
- $\gamma_{\text{HJ,SUPG}}$ is used for the stabilization of the convective term. We use here the same value as for the level set advection, i.e. $\gamma_{\text{HJ,SUPG}} = \frac{1}{2||\beta||_2}$.

Remark that no parameter for smoothing the interface was introduced. Tests dedicated to this reinitialization procedure are performed in Sect. 4.1 to show the influence of the different parameters.

3.5.3 Volume correction

Another concern when one uses only the level set method (instead of coupling it with volume of fluid [93] or particle methods [33]) is volume conservation. As a matter of fact, since the advection process does not preserve the volume at the discrete level and that the reinitalization might slightly move the interface position, the volume occupied by the two phases might change. This is a critical issue for long time simulations, as with time the differences in the volume accumulate and therefore, the final volumes might be far from the initial ones. To alleviate this problem, we use the level set correction from [28], which is a variant of the method proposed in [91]. A more sophisticated method is proposed in [41]. When the exact volumes of the phases are known, the method consists in updating after each time step the level set function with the correction

$$\phi(\mathbf{x}) \leftarrow \phi(\mathbf{x}) + \frac{\alpha_\phi}{|\Gamma|}\left(V_{\text{exact}} - V(\phi)\right) \tag{3.49}$$

where V_{exact} is the exact volume of the fluid represented by positive signs of the level set function, $V(\phi)$ the volume of the fluid represented by positive signs of the level set function using the current values of the level set function ϕ, $|\Gamma|$ is the measure of the interface and α_ϕ is a relaxation parameter, that we set to 0.5, as suggested in [28].

At the numerical level, we found that it is important to compute accurately $V(\phi)$ in order to get the volume increase or decrease (or remain the same) in the right occasions. We use then the exact integration scheme devised in Sect. 3.4.3 to compute the volume as

$$V(\phi) = \int_\Omega H(\phi) \, .$$

On the contrary, an estimate for $|\Gamma|$ suffices, since this quantity influences only the amount of correction and not its direction. We use a smoothed integrand method to estimate it:

$$|\Gamma| = \int_\Omega \delta(\phi) \cong \int_\Omega \delta_\epsilon(\phi)$$

where $\delta_\epsilon(\phi)$ is an approximation of the Dirac delta. We choose the following expression for δ_ϵ:

$$\delta_\epsilon(\phi) = \begin{cases} 0 & \text{if } |\phi| > \epsilon \\ \frac{1+\cos(\pi\phi/\epsilon)}{2\epsilon} & \text{if } |\phi| \leq \epsilon \end{cases}$$

We emphasize the fact this interface width is introduced only for computing $|\Gamma|$ and is not used for any other purpose, and in particular it does not add diffusion to the numerical scheme.

3.6 Overall scheme

The scheme that we proposed can be summarized as follows: at each (real) time step,

1. the lifting L_p for the pressure is computed using the method from Sect. 3.4.2;
2. the stabilized linearized Navier-Stokes equations (3.28) are solved;
3. the level set function is advected using (3.32) and the newly computed velocity;
4. the level set function can be redistanced by solving the discretized Hamilton-Jacobi (3.42) with a pseudo time;
5. the volume is corrected using the procedure described in Sect. (3.5.3);
6. post processing is performed if needed.

The level set function is not reinitialized at each time step since this is not necessary, potentially displaces the interface and is computationally expensive. Therefore, we reinitialize only each N_{reinit} time steps. Adaptative criteria based on the gradient of the level set function exist [106],

but they were not used since we expect to encounter situations were the gradient of the level set function might degenerate due to corners (e.g. with breaking waves). Even though no adaptative criterion is used, we still reinitialize with a sufficiently high frequency to ensure that the level set is close to a distance function. Indeed, the Robin conditions need this information to work correctly. The dependancy of the pressure correction scheme on the reinitialization is weaker, since in equation (3.25), we normalize the gradient of the level set function.

3.7 Meshes

Four different meshes were used for the simulation, each of them representing a cylinder that was then stretched in the vertical and radial directions to accomodate for the size of the cylinder to represent, since different containers have been used for the experiments. The different characteristics of the meshes are described in table 3.1.

Code	Number of vertices	Number of tetrahedra	Number of vertical subdivisions
XS	9′568	50′550	25
S	32′923	181′920	40
M	80′835	456′000	50
L	249′318	1′433′760	80

Table 3.1: Characteristics of the different meshes.

All these meshes have been built by generating an unstructured mesh for the bottom of the container and extruding it in the vertical direction, using the number of vertical subdivisions indicated in table 3.1, using the free software Gmsh [39]. This ensures that the mesh is structured in the vertical direction.

To better capture phenomena close to the wall such as boundary layers, we have the possibility to change the position of its vertices according to the following update (the initial cylinder has a radius of 1):

$$\mathbf{x} \leftarrow \frac{\mathbf{x}}{r(\mathbf{x})} \frac{1 - e^{-\alpha_{BL}\, r(\mathbf{x})}}{1 - e^{-\alpha_{BL}}} \tag{3.50}$$

where $\mathbf{x} = [x, y, z]^T$, $r(\mathbf{x}) = \sqrt{x^2 + y^2}$ and $\alpha_{BL} > 0$ is a parameter that we are free to chose. By convention, if $\alpha_{BL} = 0$, the mesh is kept untouched. Then, the larger is α_{BL}, the more the vertices of the mesh move towards the wall of the cylinder, which might improve the accuracy in this region. Remark that after this transformation, the mesh is still structured in the vertical direction.

To give an idea of how α_{BL} can change the mesh, we estimate the dimensionless wall distance for the mesh M for the application presented in Sect. 5.3, which features a quite gentle flow: we obtain $y^+ \cong 16$ with $\alpha_{BL} = 0$ while we get $y^+ \cong 4$ with $\alpha_{BL} = 3$.

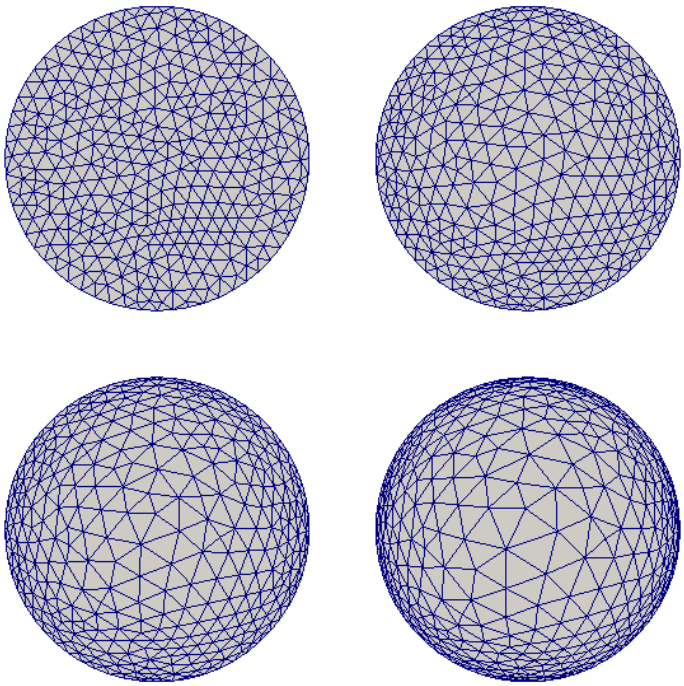

Figure 3.7: Bottom section of the XS mesh, transformed with $\alpha_{BL} = 0$ (top left), $\alpha_{BL} = 1$ (top right), $\alpha_{BL} = 2$ (bottom left) and $\alpha_{BL} = 3$ (bottom right).

4 Test cases

The different problems that we propose in this section are meant to assess the quality of the different components of our method. First of all, three different tests are devised in Sect. 4.1 for the reinitialization procedure to appreciate its quality and to better understand the effect of the stabilization terms introduced in Sect. 3.5.2. Then, in Sect. 4.2 we test the two-phase flow solver on an OSR at rest to verify the performance of the numerical integration, the pressure correction and the correction for the normal boundary condition. The test in Sect. 4.3 aims at comparing the different no-penetration boundary conditions by creating a large wave slipping along the curved wall. Finally, we assess the parallel performances of our solver in Sect. 4.4.

4.1 Reinitialization procedure: numerical assessment

Before testing the free-surface solver, we devise three tests to assess the quality of the different components of the reinitialization procedure.

For these tests, we use a simple cubic domain $\Omega = (-1,1)^3$ and structured tetrahedral meshes with N_{elem} denoting the number of elements in each direction. The goal is only to reinitialize a given level set function ϕ_0, so no flow or real time is considered. Remark that the level set function ϕ_0 is first of all interpolated on the mesh before being reinitialized. The final pseudo-time is set to $\mathscr{T} = 10$ for all the tests and the pseudo-time step to $\Delta\tau = 1$ in order to speed up the tests.

4.1.1 Grape shape surface

The domain $\Omega_1 \subset \Omega$ consists in the union of 3 spheres whose centers are $\mathbf{C}_1 = (0.4, 0, -0.3)^T$, $\mathbf{C}_2 = (-0.4, 0, -0.3)^T$ and $\mathbf{C}_3 = (0, 0, 0.4)^T$ and with radius $r = 0.45$ (see Fig. 4.1). The resulting surface is non-convex and features a thin structure: we can observe that, for sufficiently fine meshes, there is a small "tunnel" in the middle of the 3 spheres. This small feature is a good indicator of the quality of the reinitialization near the interface.

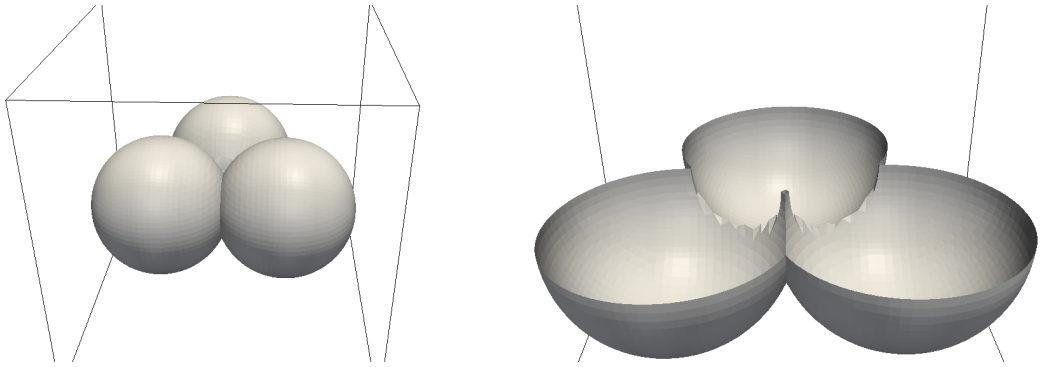

Figure 4.1: Shape of the surface for this test and a cut along the $y = 0$ plan. Remark that the internal edge on the foreground is sharply captured because it is alined with the mesh, while the two other internal edges are not.

The non-smoothness of the surface reflects situations that we might face with breaking waves or complicated wave patterns in OSRs. The initial level set function is defined as twice the signed distance to the interface:

$$\phi_0(\mathbf{x}) = 2\min(||\mathbf{x}-\mathbf{C}_1||_2 - r; ||\mathbf{x}-\mathbf{C}_2||_2 - r; ||\mathbf{x}-\mathbf{C}_3||_2 - r) \ \forall \mathbf{x} \in \Omega. \tag{4.1}$$

We compare a reference volume with the volumes obtained after the differents steps of interpolation and reinitialization. The reference volume is computed by interpolating the level set function on a very fine grid ($N_{\text{elem}} = 120$) and computing the volume delimited by the interpolated level set function. The value obtained is $\text{Vol}_{\text{ref}} = 1.12569$.

We report in table 4.1 the relative change in volume after interpolation and after reinitialization:

$$E = \frac{|\text{Vol}_{\text{ref}} - \text{Vol}_{\text{approx}}|}{\text{Vol}_{\text{ref}}}.$$

We can observe that the volume before and after reinitialization does not change much and that the error on the volume is second order with respect to $\frac{1}{N_{\text{elem}}}$. This indicates that the local interface projection works well, even with non-smooth surfaces, keeping the interface nearly unchanged. With this test, we could observe that the Hamilton-Jacobi solver (in particular the stabilization used) has nearly no influence on the results obtained for the mass conservation, highlighting the fact that the reinitialization in the elements crossed by the interface (Ω_Γ) is well isolated from the far field.

With the meshes corresponding to $N_{\text{elem}} = 30$, $N_{\text{elem}} = 50$ and $N_{\text{elem}} = 60$, the "tunnel" is captured by the interpolated level set. In these cases, we can observe that the tunnel is also present after the reinitialization, which demonstrates the ability of the scheme to keep small features.

N_{elem}	After interpolation	After reinitialization
10	$6.49\ 10^{-2}$	$6.79\ 10^{-2}$
20	$1.50\ 10^{-2}$	$1.56\ 10^{-2}$
30	$6.04\ 10^{-3}$	$5.99\ 10^{-3}$
40	$3.00\ 10^{-3}$	$3.04\ 10^{-3}$
50	$1.78\ 10^{-3}$	$1.83\ 10^{-3}$
60	$1.03\ 10^{-3}$	$1.03\ 10^{-3}$

Table 4.1: Relative error on the volume for the first reinitialization test.

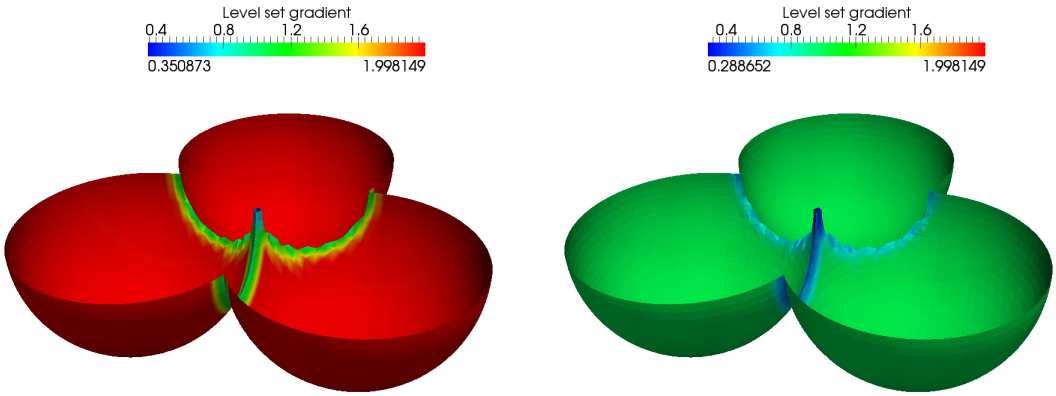

Figure 4.2: Surface before (left) and after (right) reinitialization, colored by $||\nabla\phi||_2$ ($N_{\text{elem}} = 60$ for both pictures).

4.1.2 Single sphere

We proceed now with two tests focusing on the far field reinitalization. In the first test, the surface Γ describes a sphere centered in the origin with a radius of 0.5. The initial level set function is set to

$$\phi_0(\mathbf{x}) = 2(||\mathbf{x}||_2 - 0.5) \tag{4.2}$$

which is twice the signed distance to the interface. For this test, we keep the mesh fixed ($N_{\text{elem}} = 30$) and we vary the stabilization parameters. We test 4 combinations for the two parameters appearing in Sect. 3.5.2:

- Only artificial diffusion: $\gamma_{\text{HJ,Diff}} = 0.5$ and $\gamma_{\text{HJ,SUPG}} = 0$;
- SUPG stabilization with some artificial diffusion: $\gamma_{\text{HJ,Diff}} = 0.1$ and $\gamma_{\text{HJ,SUPG}} = 0.5$;
- SUPG stabilization with less artificial diffusion: $\gamma_{\text{HJ,Diff}} = 0.01$ and $\gamma_{\text{HJ,SUPG}} = 0.5$;
- Only SUPG stabilization: $\gamma_{\text{HJ,Diff}} = 0.0$ and $\gamma_{\text{HJ,SUPG}} = 0.5$.

At the final time $\tau = \mathcal{T}$, the magnitude of the gradient of the level set function $||\nabla\phi||$ is computed on the plane $x = 0$ and the results are displayed in Fig. 4.3. We can see that the gradients are very different depending on which stabilization is chosen. Indeed, with a large artificial diffusion, the level set function becomes flatter when approaching $\partial\Omega$. With smaller artificial diffusion, the effect vanishes, thus giving a better approximation of the distance to the interface.

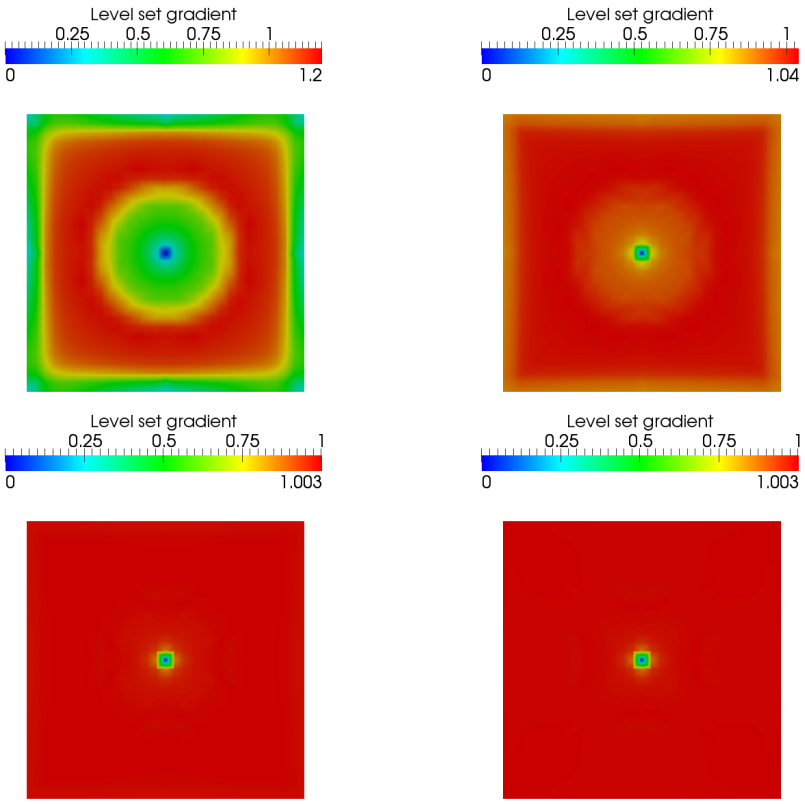

Figure 4.3: Magnitude of the gradient of the level set function after reinitialization for the different choices of stabilization: only artificial diffusion (top left), SUPG with large artificial diffusion (top right), SUPG with small artificial diffusion (bottom left) and SUPG only (bottom right).

Remark that the gradient at the interface itself is largely unaffected by this choice, as shown in Fig. 4.4, thanks to the interface local projection.

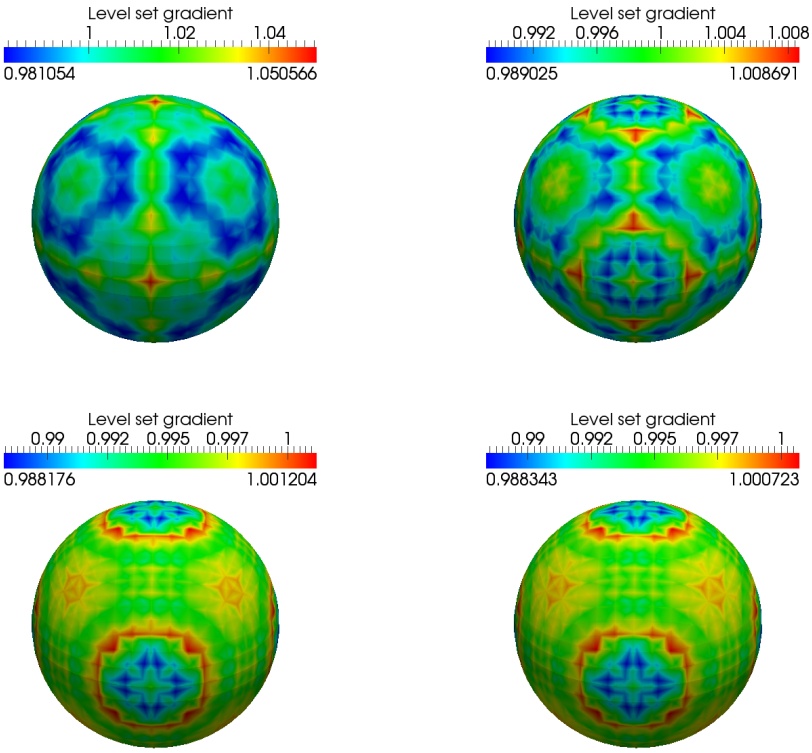

Figure 4.4: Magnitude of the gradient of the level set function on the interface after reinitialization for the different choices of stabilization: only artificial diffusion (top left), SUPG with large artificial diffusion (top right), SUPG with small artificial diffusion (bottom left) and SUPG only (bottom right).

4.1.3 Planar interface

The last test that we perform for the reinitialization simply considers the interface Γ as the plane $x+2z=0$. This simple case aims at testing the direction of the gradient $\nabla\phi$ (as discussed in Sect. 3.5.2). Morevoer, this interface has non-null intersection with $\partial\Omega$, which was not the case for the previous tests. We define the initial level set as

$$\phi_0(\mathbf{x}=(x,y,z)) = \frac{1}{\sqrt{5}}(x+2z) \tag{4.3}$$

so that it does not coincide with the distance to the interface, even if $||\nabla\phi_0|| = 1$, see Fig. 3.6. The quantity of interest here is not the magnitude of $\nabla\phi$, but its direction. The direction of $\nabla\phi$ is important because it shows from where the information is taken for the distance to the interface. On Fig. 4.5, several streamlines computed from $\nabla\phi$ after reinitialization are

displayed.

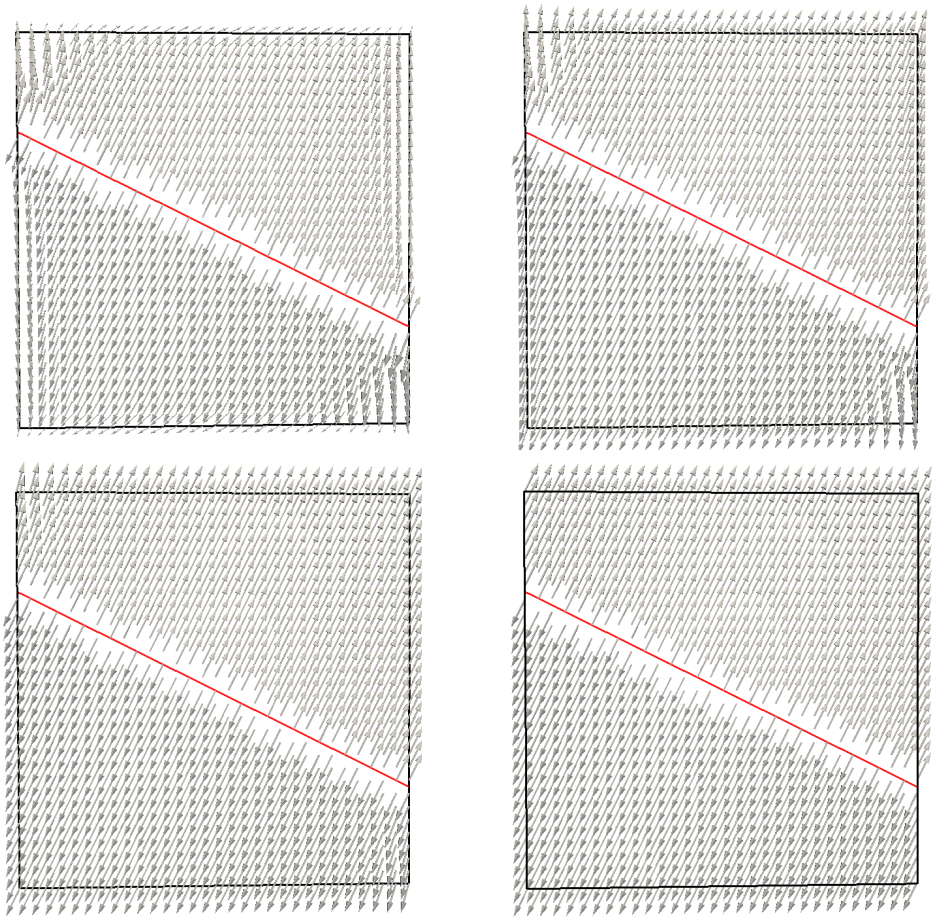

Figure 4.5: The vector field $S(\phi)\nabla\phi$ computed after reinitialization for the different choices of stabilization: only artificial diffusion (top left), SUPG with large artificial diffusion (top right), SUPG with small artificial diffusion (bottom left) and SUPG only (bottom right). The red line indicates the interface Γ. Remark the differences on the top of the left vertical side. For large artificial diffusion (top left and right), the vector field is nearly parallel to the vertical boundary while with lower artificial diffusion (bottom left and right), it is rather oblique.

We can observe that with only artificial diffusion, the information is correctly originated from the interface. Combining the SUPG stabilization with a quite high artificial diffusion also gives the correct gradient direction, however, with lower artificial diffusion, we can see that some information is coming from $\partial\Omega$ and not from Γ. This yields a dangerous situation at the numerical level: indeed, we linearized the Hamilton-Jacobi equation in a transport equation (see equation (3.40)) and we get here an inlet boundary, where no Dirichlet condition is enforced. Numerical aberrations might then occur at these inlets. This is even more evident when no artificial diffusion is added with the SUPG stabilization as the solution does not

change with respect to the initial solution ϕ_0.

We can draw two conclusions from these three reinitialization tests:

1. The interface local projection yields a very good mass conservation, it is accurate enough to track small features and it gives a very good estimate of the distance to the interface in Ω_Γ. Moreover, it is only very weakly influenced by the Hamilton-Jacobi solver adopted for the far field $\Omega \setminus \Omega_\Gamma$.

2. The Hamilton-Jacobi solver must be carefully stabilized. A balance between the SUPG stabilization, which in some cases cannot distinguish the right solution, and the artificial diffusion, which yields a poor approximation of the gradient of ϕ, must be adopted. This leads us to the choice for the stabilization parameters $\gamma_{\text{HJ,Diff}} = 0.1$ and $\gamma_{\text{HJ,SUPG}} = 0.5$.

4.2 Standing still cylinder

We consider an OSR which is not shaken ($\omega = 0$), a situation for which we can expect to get results very close to the exact solution, i.e. a flat interface and zero velocity in Ω. This test shares some similarities with the tests proposed in [22] and with a test that we performed in [28], but here stabilization is added and curved boundaries are considered.

The setup of this test is very simple: the domain Ω is a cylinder with height $H = 0.4$m and radius $R = 0.15$m. The liquid is initially at rest, i.e. $\mathbf{u}_0(\mathbf{x}) = \mathbf{0}$ and the interface is flat, at an height of $H_0 = 0.123$m, so the initial level set function is defined as $\phi_0(\mathbf{x} = (x, y, z)) = H_0 - z$. The only external force acting on the fluid is the gravity, therefore $\mathbf{f} = (0, 0, -9.806)^T$.

The exact solution corresponds to the interface Γ defined by $\phi(\mathbf{x}, t) = \phi_0(\mathbf{x})$ and to a null velocity $\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_0(\mathbf{x}) = \mathbf{0}$. This exact solution is not easy to obtain numerically, for several reasons. First of all, the misalignment of the interface with the mesh makes it unable to capture jumps across the interface (3.14), the effect of the pressure correction should then be visible here. Secondly, the integration of the discontinuous quantities across the interface must be carefully performed. Finally, we will see that, depending on the boundary conditions chosen, the curved geometry might prevent us from getting correct results. We consider here only the boundary conditions **A** (vertical free slip) and **B** (tangential free slip) for the fluid (see Sect. 3.2.2). For this test, we disabled the reinitialization to avoid effects that could come from it. Since the interface is not supposed to move, it should not be required.

We start with the boundary conditions **A**, i.e., we impose strongly $\mathbf{u} \cdot \mathbf{e}_x = 0$ and $\mathbf{u} \cdot \mathbf{e}_y = 0$ on Γ_{wall} and we let the fluid slip freely in the vertical direction. We perform simulations with a time step of $\Delta t = 0.005$s and a total time $T = 0.1$s, with four configurations which differ on whether the integration is adapted to the interface and the pressure correction is enabled.

First of all, if neither the adapted integration nor the pressure correction are used, we can observe that treating the density and the viscosity in a sharp way introduces a spurious velocity

which deforms the interface (see Fig. 4.6). The magnitude of the spurious velocity is slightly reduced if the integration is adapted to the interface or if the pressure correction is used. However, one needs to use both the adapted integration and the pressure correction to obtain a velocity with a negligible norm. The table 4.2 shows the maximum velocity magnitude at the time $t = T = 0.1$s. Remark that for this particular test case, splitting the pressure as we did in Sect. 3.4.2 for the pressure correction yields the same effect as substracting the hydrostatic pressure.

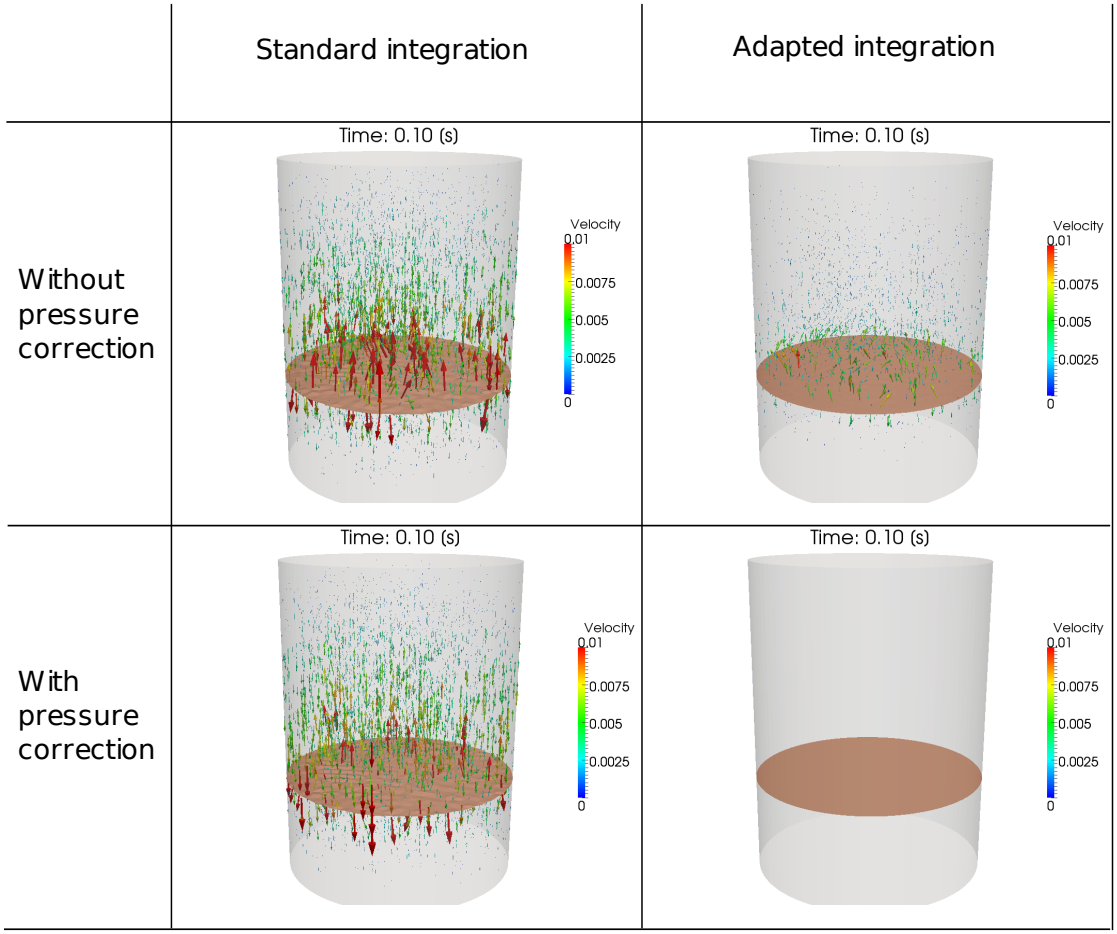

Figure 4.6: Representation of the position of the interface Γ and of the velocity field $\mathbf{u}$ at the time $t = T = 0.1$s, for the 4 different configurations considered. Remark that the arrows representing the velocity field have the same scale for all the four cases.

	Standard integration	Adapted integration
No pressure correction	$\max_\Omega \|\|\mathbf{u}\|\|_2 = 1.85\ 10^{-2}$	$\max_\Omega \|\|\mathbf{u}\|\|_2 = 1.01\ 10^{-2}$
With pressure correction	$\max_\Omega \|\|\mathbf{u}\|\|_2 = 1.68\ 10^{-2}$	$\max_\Omega \|\|\mathbf{u}\|\|_2 = 4.19\ 10^{-7}$

Table 4.2: Velocity magnitude for different combinations of integration and pressure correction

These results are of the same quality as those presented in [28] for another configuration.

We change now the boundary conditions and use the strategy **B** instead of **A**, i.e., only the normal component of the velocity is imposed essentially $\mathbf{u}\cdot\mathbf{n} = 0$ on Γ_{wall}. All the other parameters do not change, neither does the exact solution. Our interest now is to test the effect of the correction devised in Sect. 3.4.4 on the solution, so adapted integration and the pressure correction are used (without them, results are similar to those with conditions **A**). In Fig. 4.7, we can observe that without correction, a spurious velocity appears, which looks different from the ones observed in Fig. 4.6: here the spurious velocity is horizontal while it was rather vertical when no adapted integration or no pressure correction was used, evidencing that the sources of these spurious velocities are different. We can also observe that with the correction term the spurious velocity completely vanishes: without correction, we have at time $t = T = 0.1$s $\max_\Omega ||\mathbf{u}||_2 = 1.42\ 10^{-2}$ while with the correction, this value drops to $\max_\Omega ||\mathbf{u}||_2 = 1.42\ 10^{-7}$. This demonstrates that normal boundary conditions imposed strongly should always be used in conjunction with the correction term from Sect. 3.4.4.

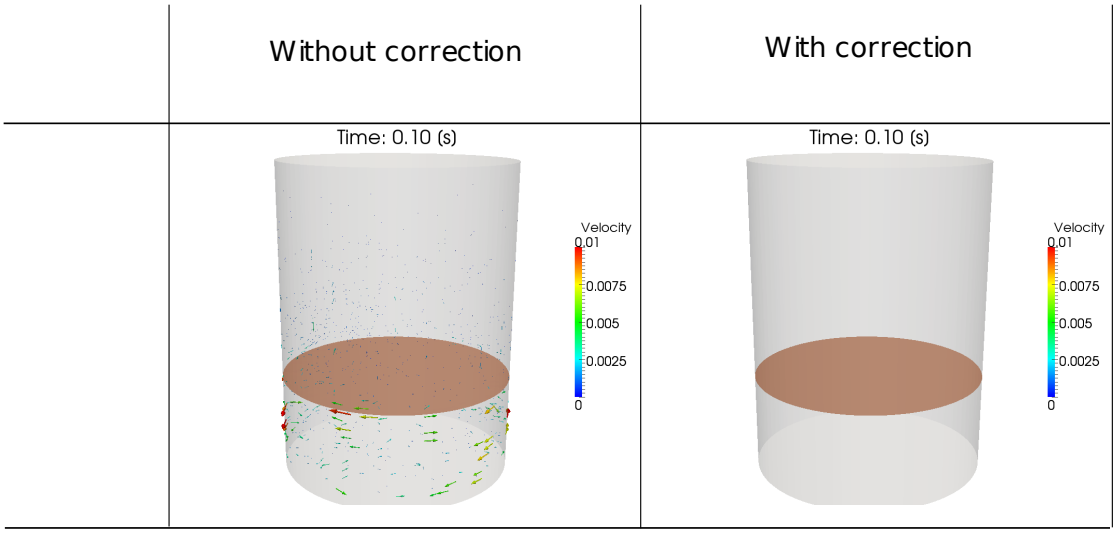

Figure 4.7: Representation of the position of the interface Γ and of the velocity field $\mathbf{u}$ at the time $t = T = 0.1$s, for configurations with and without correction for the essential normal boundary condition.

4.3 Breaking wave in a cylinder

To investigate the effects of the different no-penetration conditions presented in section 3.2.3, we devise a test case in a cylindrical domain with a radius $R = 0.144$m and height $H = 0.4$m. We consider that

- the fluid is at rest at the initial time $\mathbf{u}_0 = \mathbf{0}$ and the surface flat with an height of $H_0 =$

0.15m, $\phi_0(\mathbf{x} = (x, y, z)) = z - H_0$;

- no-slip conditions are applied on the top and bottom of the cylinder;

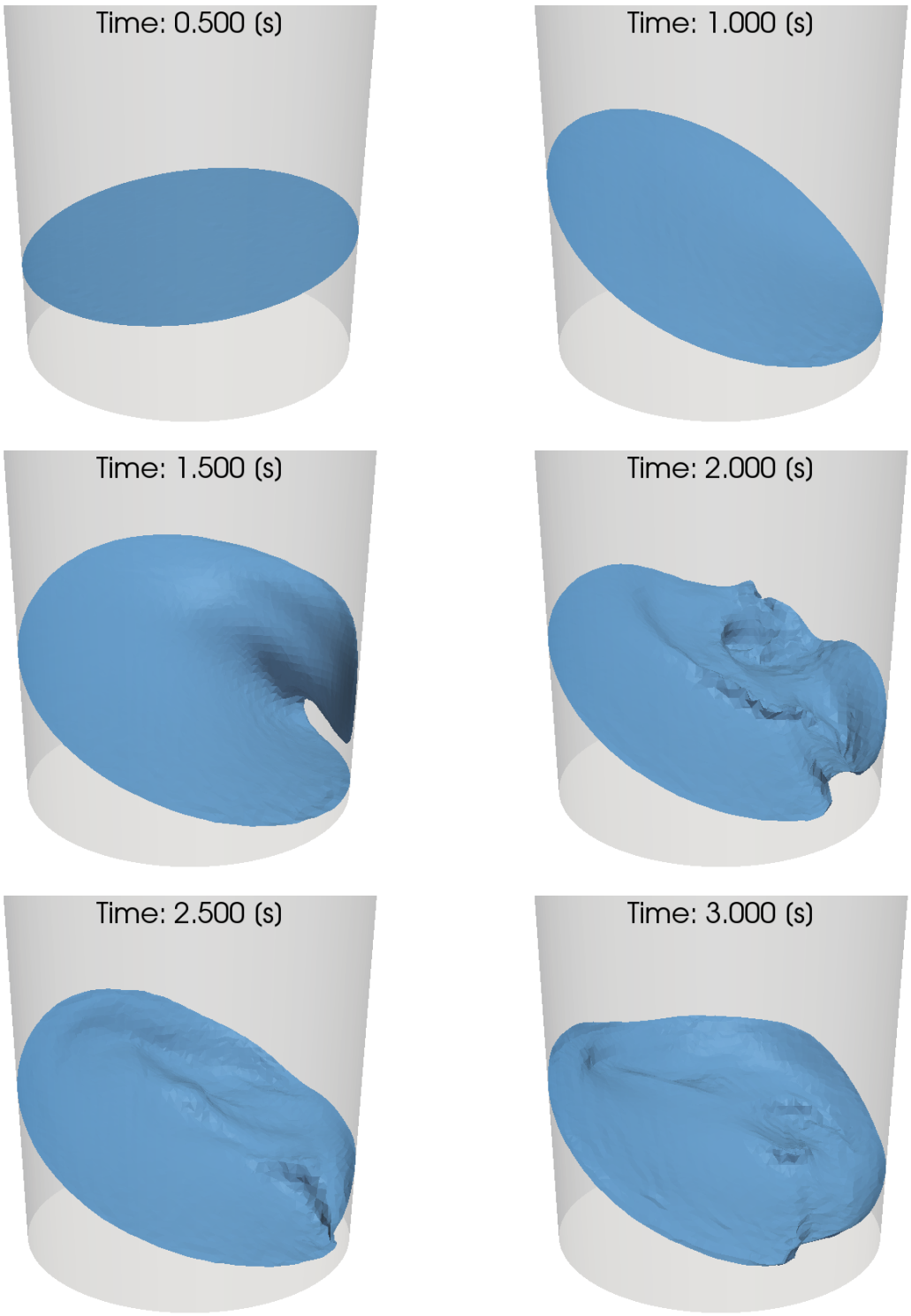

Figure 4.8: Illustration of the general pattern of the wave at different times. The simulation was carried out with normal boundary conditions (with correction) and the M mesh. Note that the angle of the camera changes between the pictures.

- the different no-penetration conditions are imposed on the curved wall, supplemented by a free-stress condition on the remaining components, i.e., strategy **A** and **B** are used.

To test the difference between horizontal and normal boundary conditions, we create a large wave rolling on the curved wall by agitating quickly the cylinder: it is accelerated from 0 to 60 RPM in 1 second.

At the numerical level, we simulate the creation of the wave during $T = 3$ seconds, with a time step $\Delta t = 0.005$ s. The different meshes, listed in Sect. 3.4.4 are used. In all the simulations that are conducted, the same general behavior is observed (see Fig. 4.8):

- Until $t = 1$s, a large wave develops on the surface but it remains quite planar, with an increasing slope.
- Between $t = 1$s and $t = 1.5$, the wave becomes asymmetric and it starts breaking, eventually crashing on the surface around $t = 1.5$s (depending on the boundary condition used and on the mesh size).
- After the wave has broken, secondary waves are created from the shock and the surface stays perturbed for the rest of the simulation, showing sometimes waves going backwards, vortices or locally breaking waves. In most of the cases, at the end of the simulation, a small continuously breaking wave is present near the trough.

However, besides this general trend, the results of the simulations were very different one from each other, and especially between the two types of boundary conditions.

4.3.1 Boundary conditions comparison

The first comparison that we perform is between the two types of no-penetration conditions. We compute the solution of this test case on the mesh S for both the horizontal condition and the normal condition. The differences between the two configurations are considerable.

During the first second of the simulation, no notable difference on the free surface can be observed. Then, when the wave starts rolling on the wall with the normal conditions, jets of liquid are projected vertically in the case of the conditions **A** (see Fig. 4.9). These jets are neither physical nor due to instabilities, they just reflect the constraint on the fluid to slip vertically on the walls instead of being free to roll tangentially.

Thanks to this test, we can conclude that condition **A** is not suitable for the simulation of free surface flows in cylindrical vessels where large tangential velocities appear. The only acceptable no-penetration condition is therefore the normal condition, with the correction term, as shown by the test in Sect. 4.2.

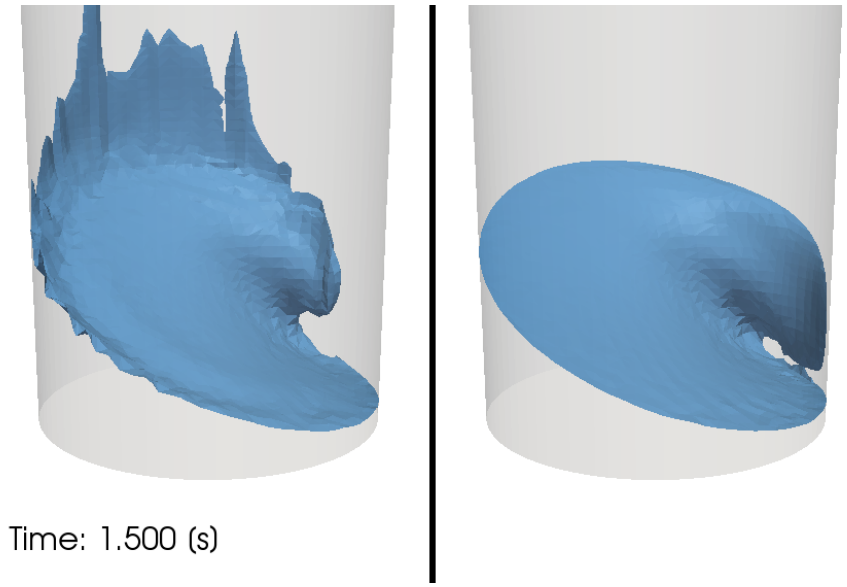

Figure 4.9: Difference of the free surface position between a simulation with horizontal boundary condition (strategy **A**,left) and normal boundary condition (strategy **B**,right). We can observe the jets of liquid on the left side. We can also see that the wave on the right has already broken and is about the meet the surface, while the wave on the left is late.

Mesh	XS	S	M	L
Merge time [s]	-	1.505	1.515	1.515

Table 4.3: Time observed for the topological change depending on the mesh used.

4.3.2 Mesh comparison

This test case is a good benchmark to check the numerical accuracy on complicated flows and interface structures, since many details of the flow from this test case are directly visible. Several small features are only captured when the mesh is fine enough and numerical diffusion does not smooth them out. We use the different meshes that we have at hand, from the XS to the L meshes (see Sect. 3.7 for a detailed description). The numbers of degrees of freedom for the fluid (velocity and pressure) are, from the smaller to the finer mesh, around 38′000, 132′000, 323′000 and 997′000. The degrees of freedom for the level set function represent one fourth of the degrees of freedom for the fluid.

A first benchmark quantity is the time at which the breaking wave merges with the surface, i.e., a topological change occurs. The table 4.3 reports these times for the different meshes. We recall that the time step was $\Delta t = 0.005$ s for all the meshes. The time step used influences the results obtained, but this will be investigated in Chap.5.

We observe that for the coarsest mesh, the wave does not break, but rather collapses. For the three other meshes, the breaking time is almost the same, 1.515s for the two finest meshes.

We could also identify other specific features of the flow that can be used to estimate the precision of the simulations. At $t = 2.0$s, several interesting phenomena occur. First of all, the remainder of the initial wave splashes against the wall on the foreground of Fig. 4.10. This is well captured by all three finest meshes. A secondary wave, propagating on the surface since the merge of the initial wave can be seen in the center of the surface on Fig. 4.10. The resolution of the finest mesh allows us to observe that this wave is actually breaking. The wave is also distinguishable on the mesh M and it is also present, to some extent, on the mesh S. It is however completely absent on the coarsest mesh. Finally, a large vortex is created at the back of the initial wave, as shown on the mesh L. The mesh M shows a smaller vortex, while the mesh S has only a small depression.

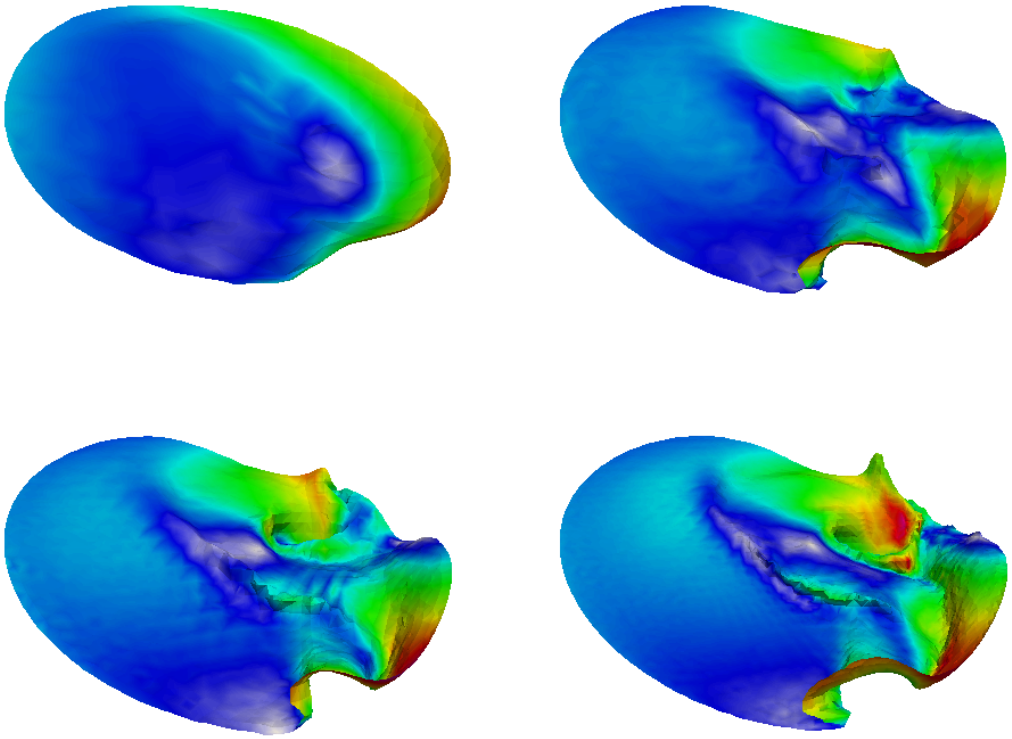

Figure 4.10: Interface shapes obtained with the meshes XS (top left), S (top right), M (bottom left) and L (bottom right) at $t = 2.0$s. Interfaces are colored by velocity magnitude.

At $t = 2.75$, we can count 3 small breaking waves on the surface: one continuously breaking wave at the base of the wave, a second one just on top of it and a third smaller one near the top. They can be well identified on both the M and L meshes. It is also possible to identify their locations with the mesh S, but the fact that these waves are breaking is not evident. A last benchmark, interesting for the following parts of this work, is the shape of the wave on the

back of the cylinder. With the mesh L, we can observe a well-formed secondary peek, which has a lower amplitude with the mesh M. With the mesh S, this peak reduces to a flatter area while the profile with the mesh XS is simply regular.

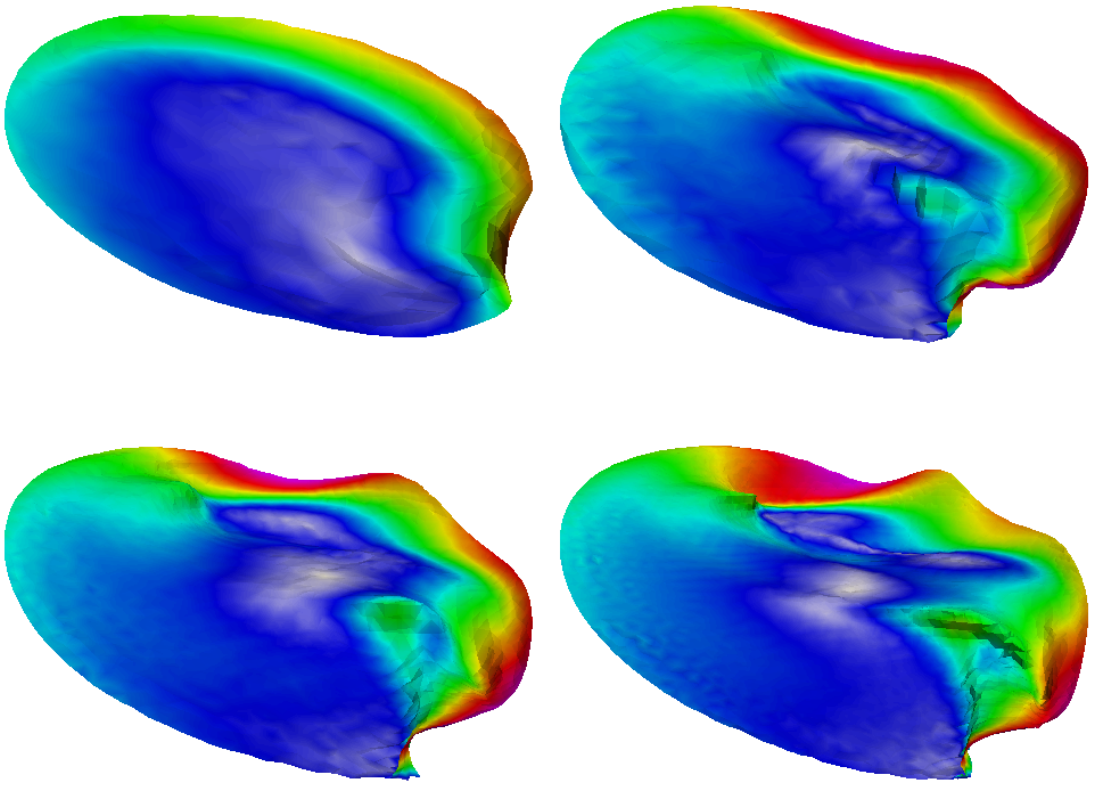

Figure 4.11: Interface shapes obtained with the meshes XS (top left), S (top right), M (bottom left) and L (bottom right) at $t = 2.75$s. Interfaces are colored by velocity magnitude.

We can conclude that the mesh L provides the best accuracy, capturing the complicated structures of the surface. The mesh M, however, yields a fairly good approximation of all the phenomena appearing, even if they do not appear as clearly as with the mesh L. The mesh S can be useful to describe the general behaviour of the flow, but it does not fully capture its complicated nature. Finally, the coarsest mesh XS does not provide a good enough approximation.

4.3.3 Comparison with a regularized method

We present now a comparison between the method that we developed in chapter. 3 and a simple regularized method. For the sake of comparison, we kept almost the same settings between the two techniques. The only differences are that with the regularized method,

- the viscosity μ and the density ρ are smoothed across the interface. We use the same smoothing as in [103], namely $\mu = \mu_a + H_\delta(\phi)(\mu_l - \mu_a)$ and $\rho = \rho_a + H_\delta(\phi)(\rho_a - \rho_l)$ with

$$H_\delta(d) = \begin{cases} 0 & \text{if } d < -\delta \\ \frac{d+\delta}{2\delta} & \text{if } -\delta \leq d \leq \delta \\ 1 & \text{if } d > \delta \end{cases}$$

  where δ is a constant real parameter;

- the integration adapted to the interface Γ as described in Sect. 3.4.3 and the pressure correction scheme devised in Sect. 3.4.2 are not used.

The parameter δ introduced for the smoothing of the viscosity and density across the interface plays a crucial role in the regularization method: a too large value of δ might induce non-physical phenomena due to the large interface width with intermediate density and viscosity while a too small value of δ might lead to inaccurate solutions, as observed in the test 4.2.

To compare the two methods, we go on with the same test already used in this section. Fig. 4.12 illustrates the fact that small features are inaccurately captured by the smoothed method, especially when δ is large: after the wave met the surface, some water is projected against the wall, forming a thin film which is well captured by our sharp method. On the contrary, with the regularized method, the film is much smaller than it should be. Fig. 4.13 shows the opposite effect: the regularized method tends to favour the formation of irrelevant bubbles in the liquid. These small structures are actually created normally but, due to their mixed density, they do not rise to the surface as quickly as they should, as illustrated on Fig. 4.14.

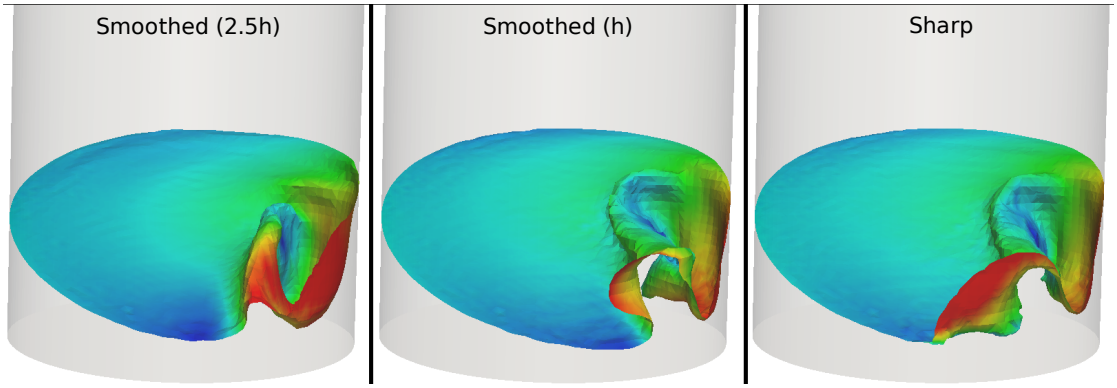

Figure 4.12: Comparison of the shape of the free surface with the regularized method with $\delta = 0.02 \cong 2.5h$ (left) and $\delta = 0.007 \cong h$ (center) and with the sharp method (right), at $t = 1.75$s.

We can conclude from this test that the method proposed in this work performs better than the regularized method implemented here. Indeed, the sharp method captures much better the small details of the flow. On top of that, the physics is kept unchanged between the different meshes, which is reflected by the fact that refining the mesh only adds details, but it does

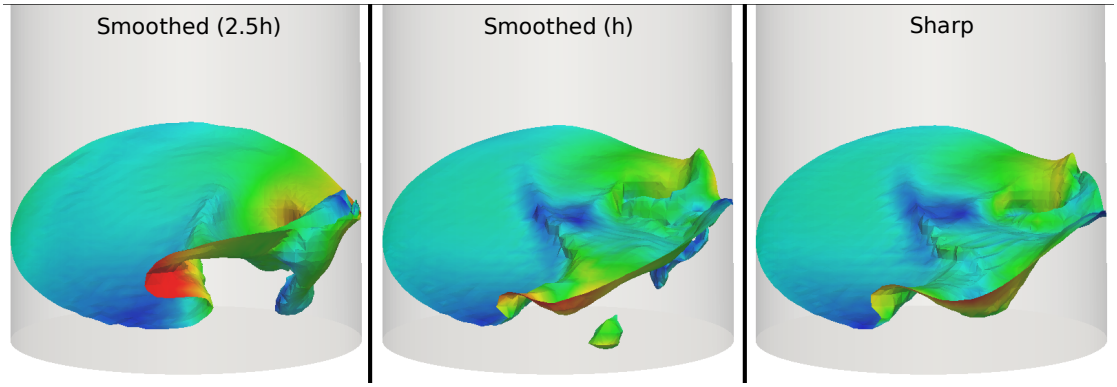

Figure 4.13: Comparison of the shape of the free surface with the regularized method with $\delta = 0.02 \cong 2.5h$ (left) and $\delta = 0.007 \cong h$ (center) and with the sharp method (right), at $t = 2$s.

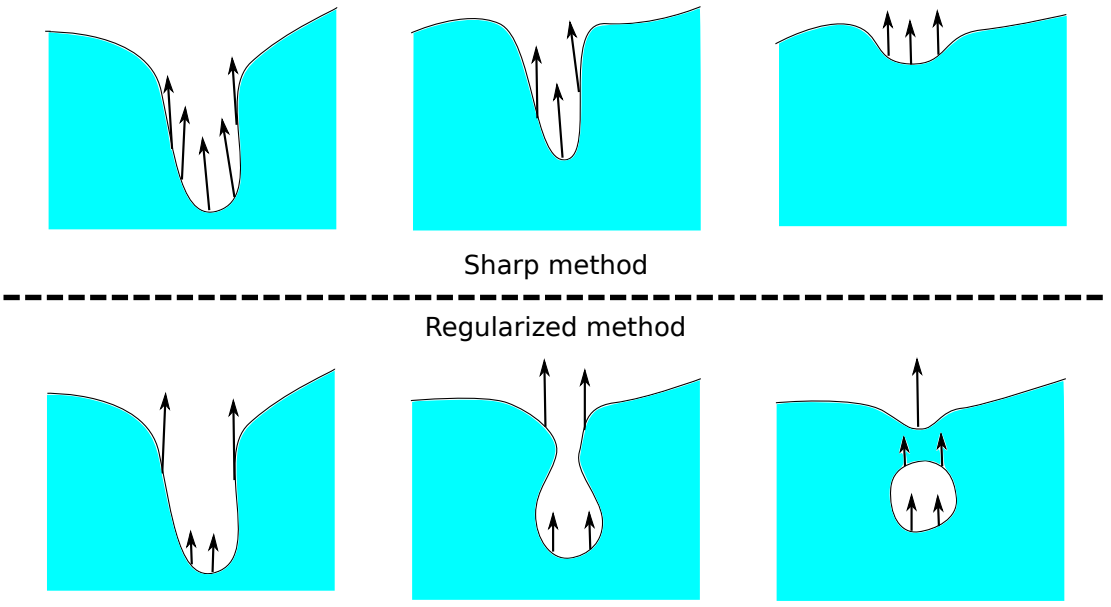

Figure 4.14: Schematic explanation of the inaccurate bubble creation by regularized methods: in the case of sharp methods, buoyancy forces act as physically expected. In the oppposite, with regularized methods, the air in the depression has a higher density than nominally, so the buoyancy forces are reduced, enhancing the creation of bubbles or other structures.

not change the global behavior of the flow. This is extremely important to trust the results obtained.

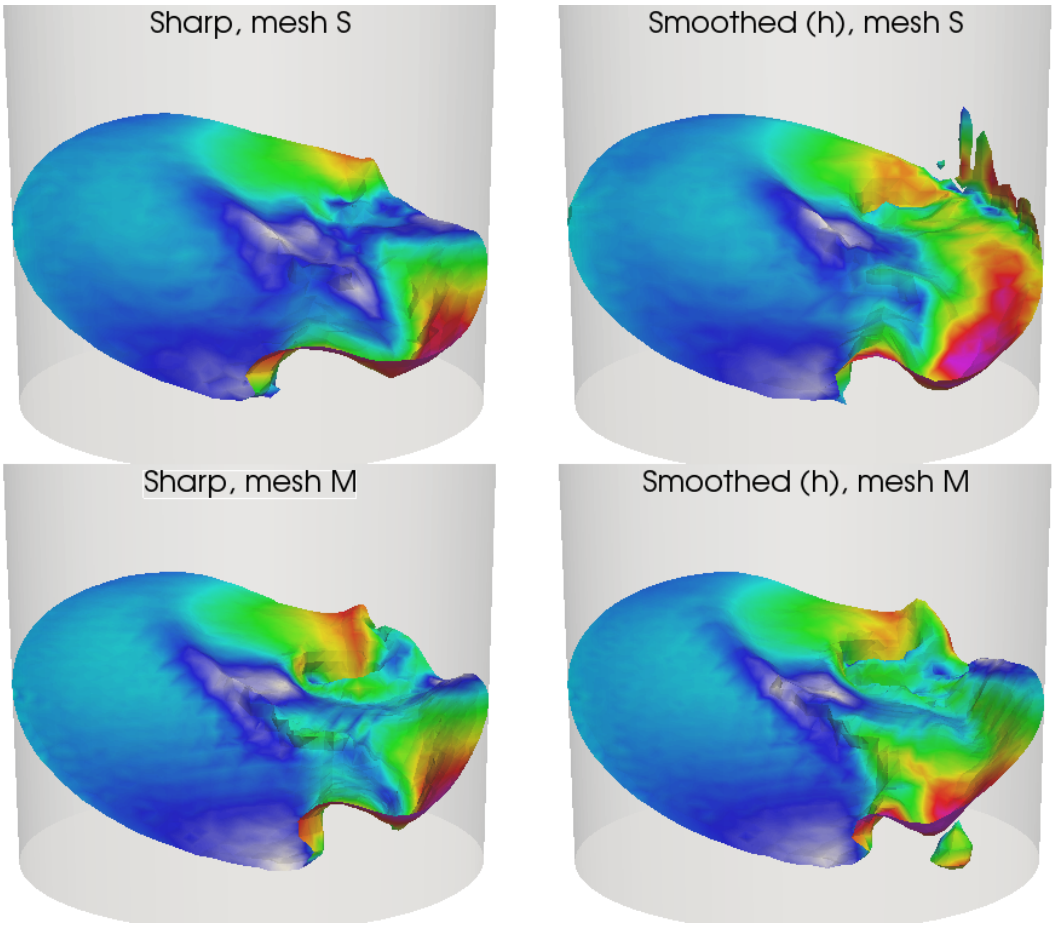

Figure 4.15: Comparison of the shape of the free surface for two meshes (S and M) for both sharp and regularized (with $\delta = h$) methods.

4.4 Parallel performances

Finally, we come to the assessment of the parallel performances of our solver. In this work, we do not intend to improve the parallelism of fluid solvers with respect to state-of-the-art solvers for single phase flows. However, we still comment on how the machinery that we set up affects the parallel properties of the scheme. For all the simulations that we performed, most of the time was spent to assemble and solve the linearized Navier-Stokes equations. We focus on that part of the simulation for the parallel performances, knowing that the remarks and comments also apply to the other parts of the method, the level set advection and the Hamilton-Jacobi resolution.

The parallel abilities of a code are quantified by the strong and weak scalabilities:

- The *strong scalability* deals with the speed up obtained by increasing the number of processors used to solve a given problem. In case of perfect strong scalability, multiplying

the number of processors by two yields a two times faster execution. This is interesting if we simply want to speed up the simulations. In the figures that we produce to assess the parallel properties of our method (Fig. 4.16 for example), a perfect strong scalability is represented by a slope of 1 and the greater is the slope, the more strongly scalable is the method.

- The *weak scalability* deals with the difference in the timings between the resolution of a given problem and a similar problem twice larger solved with twice the number of processes. This is the quantity to observe if we want to pass from a coarse to a finer mesh and increase the number of CPUs to keep a similar computational time. On the scalability figures (e.g., Fig. 4.16), a perfect weak scalability is materialized by the perfect overlap of the curves representing the different meshes. The closer are the curves, the more weakly scalable is the method.

We implemented our method within the LifeV library, using the generic programming paradigm described in Appendix A for the assembly of the linear systems. LifeV is a general purpose C++ library for scientific computing, with special emphasis on finite elements and parallel computations.

Domain decomposition methods are used to design parallel preconditioners, composed of an overlapping Schwarz provided by the Ifpack package of Trilinos [85] with full *LU* factorization for the subdomains, performed by the UMFPACK library [24]. All the preconditioned systems, i.e., the linearized Navier-Stokes, the level set advection and the linearized Hamilton-Jacobi equation, are solved iteratively by a GMRES solver provided by the AztecOO package of Trilinos [46]. The preconditioners were not reused between the different time steps since their properties quickly degraded, e.g. due to the rapid change of density and viscosity in the linearized Navier-Stokes equations.

The first expensive computation is the assembly of the linear system, which must be redone at each time step, since the density and viscosity vary due to the motion of the interface. Incremental updates can be done, see e.g. [106], but they were not used here for implementation reasons. Usually, the assembly is a part of the code that scales very well, unless XFEM, adaptative mesh refinement or specific finite element space are used. In these cases, elements crossed by the interface might require more time to get their local contributions computed, unbalancing the computational load. In Tab. 4.4, we report the times required to assemble the linearized Navier-Stokes (left and right hand side, with application of the boundary conditions).

Number of CPU	8	16	32	64
Assembly time (mesh S) [s]	13.54	6.95	4.01	3.00
Assembly time (mesh M) [s]	49.04	19.66	9.77	5.91
Assembly time (mesh L) [s]	–	–	41.99	17.33

Table 4.4: Assembly time for different meshes and different CPU counts.

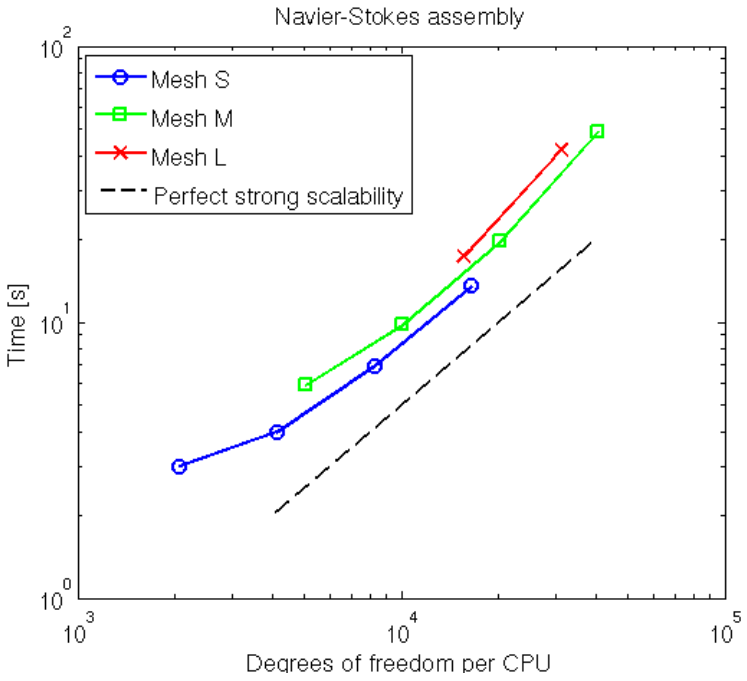

Figure 4.16: Timings for the assembly of the linearized Navier-Stokes system for different meshes and numbers of CPUs.

We represent these timings on Fig. 4.16 for a more comprehensive analysis. We remark first of all that we could not perform simulations with more than 50000 degrees of freedom per CPU because of the limitation in memory. Near this limit, the Fig. 4.16 shows that the strong scalability is good, since the different curves have a slope close to 1. The scalability slows down as the number of degrees of freedom decreases: with the coarsest mesh (mesh S), doubling the number of CPUs from 32 to 64 yields only gain 25% instead of the 50% expected. Several factors might explain this phenomena:

- First of all, communications are necessary to complete the assembly. Indeed, the degrees of freedom of the problem are distributed among the different processors and contributions related to a particular degree of freedom can come from elements belonging to different subdomains. However, this is an issue common to every assembly and good scalability can still be expected.

- The test used for computing the timings was carried out with boundary conditions **B**, that is with strongly imposed normal boundary conditions. Imposing this condition on a particular degree of freedom takes some time, due to the implementation chosen (see Sect. 3.4.4). Moreover, the corrective term must be assembled on the same boundary. The partitioning of the mesh does not take into account the different boundaries and some subdomains might contain significantly more degrees of freedom located on Γ_{wall} than others, resulting in a non-optimal balancing of the computational load.

- Finally, if no enrichment of the space is used, we still need a special operation for the elements crossed by the interface: in the test 4.2, we stated that the integration must be adapted to the interface to get correct solutions. This represents an additional work that is needed only in those elements crossed by the interface and therefore, a subdomain containing many of these elements can have more computations to perform.

On Fig. 4.17, we report the proportion of time spent for the different components of the assembly of the system, namely the assembly of the contributions local to each processor for the left (LHS) and right hand sides (RHS), the finalization of the LHS and RHS, including interprocessor exchanges of contributions, and finally the application of the boundary conditions to the whole system. We observe that the part dedicated to the application of the boundary conditions increases with the number of CPUs, indicating that most of the strong scalability is lost due to this operation. To remove this bottleneck, it would be necessary to find a better implementation of the imposition of the normal condition than the one proposed in Sect. 3.4.4. Load balancing taking into account the boundary could also be devised in order to improve the strong scability.

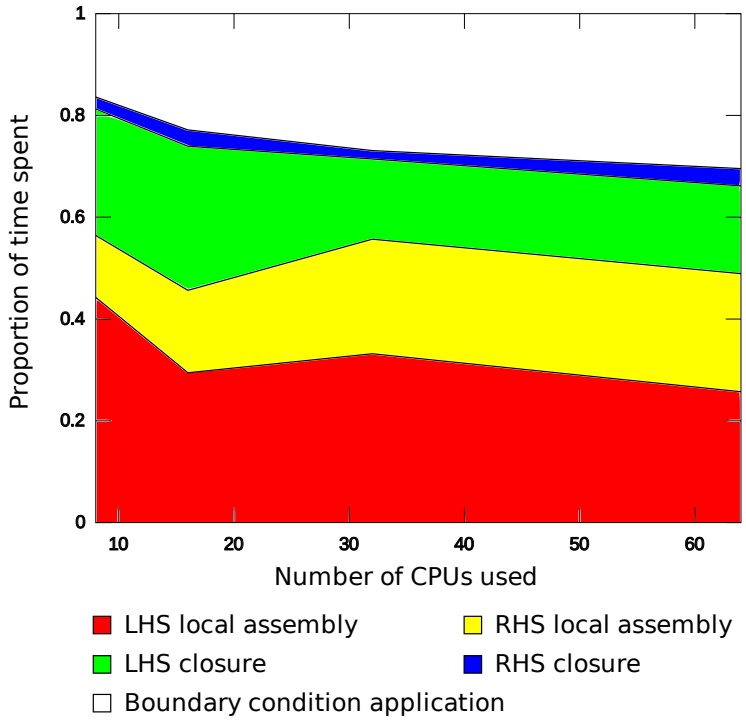

Figure 4.17: Proportion of time taken by the different components of the assembly with respect to the number of CPUs used.

On the other side, we can remark on Fig. 4.16 that the weak scalability is very good independently of the problem size since the different curves are close one to each other. This means that increasing the size of the problem, i.e. use a finer mesh, is not a problem since we can use more CPUs and keep a similar efficiency for the assembly.

The computation of the preconditioner is another item where heavy computations are required. We used here an additive Schwarz preconditioner, with a complete LU factorization for each subdomain. The timings for its computation are reported in table 4.5 for several meshes and numbers of CPUs and they are also represented on Fig. 4.18.

Number of CPU	8	16	32	64
Preconditioner time (mesh S) [s]	12.38	4.50	2.42	1.76
Preconditioner time (mesh M) [s]	70.58	22.18	8.55	3.77
Preconditioner time (mesh L) [s]			60.10	23.26

Table 4.5: Timings corresponding to the computation of the preconditioner for different meshes and CPU counts.

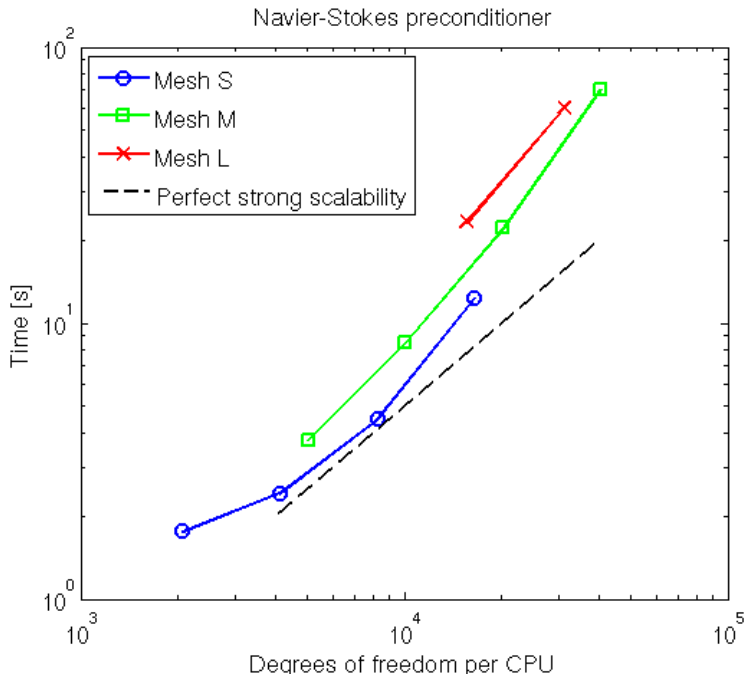

Figure 4.18: Timings for the computation of the preconditioner for the linearized Navier-Stokes system for different meshes and numbers of CPUs.

We can observe that the computation of the preconditioner enjoys a strong superscalability, a phenomenon due to the superlinear complexity of the LU factorization. The weak scalability is also quite good, but not as good as for the assembly. The good scalability properties of the preconditioner are however compensated by the rather bad scalabilities of the linear solver, as reported in table 4.6. Indeed, the additive Schwarz preconditioner is known to be suboptimal in the sense that the conditioning of the preconditioned system increases with an increasing number of subdomains [83]. This is reflected in the number of GMRES iterations required to reach convergence, as reported in table 4.7. We can clearly see on Fig. 4.19 that both the strong and weak scalabilities are affected, even if with a large number of degrees of freedom per CPU, the strong scalability is quite close to the optimal one.

Number of CPU	8	16	32	64
Linear solve time (mesh S) [s]	11.18	6.25	3.88	3.30
Linear solve time (mesh M) [s]	47.06	24.13	13.61	8.01
Linear solve time (mesh L) [s]			61.86	35.36

Table 4.6: Timings corresponding to the resolution of the linear system for different meshes and CPU counts.

Number of CPU	8	16	32	64
Number of GMRES iterations (mesh S)	37	45	46	60
Number of GMRES iterations (mesh M)	44	48	62	69
Number of GMRES iterations (mesh L)			66	76

Table 4.7: GMRES iteration counts for the resolution of the linear system for different meshes and CPU counts.

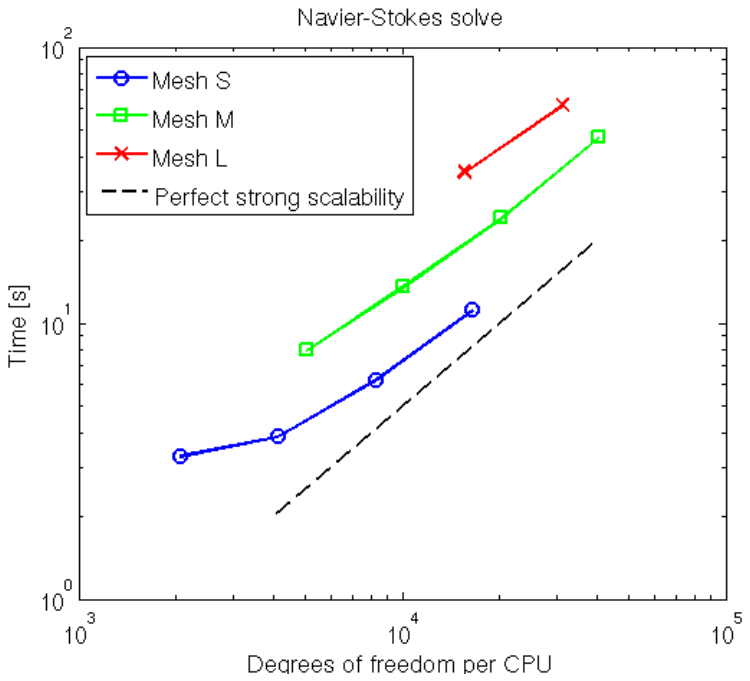

Figure 4.19: Timings for the computation of the solution of the linearized Navier-Stokes system for different meshes and numbers of CPUs.

This issue could be overcome by using multilevel Schwarz preconditioners, which are known to yield better scalability [83]. More specific preconditioners for the Navier-Stokes equations, such as the PCD (see, e.g., [31] for a recent overview), might also be adapted to two phase flows.

5 OSR simulations

After having considered academic tests in chapter 4, we compare our method with real experiments. All the different experimental results were obtained by the Laboratory for Hydraulic Machines of the EPFL.

Before coming to the tests themselves, we comment on the different combinations of parameters. The hydrodynamics in an OSR is fully determined by four parameters (see Fig. 3.1 for a representation):

- the radius of the cylinder R,
- the initial liquid height H_0,
- the agitation rate ω
- and the agitation radius R_s.

In practice, only three adimensional numbers are necessary to determine the hydrodynamic regime of a configuration: the Froude number, defined by

$$\mathrm{Fr} = \sqrt{\frac{2\omega^2 R}{g}}$$

quantifies the relative strength of the inertial forces and the gravity. The diameters ratio

$$\Pi_2 = \frac{R_s}{R}$$

and the height to diameter ratio

$$\Pi_3 = \frac{H_0}{2R}$$

take into account the scale changes in the geometry. For two configurations with the same adimentional numbers $(\mathrm{Fr}, \Pi_2, \Pi_3)$, the resulting flow structure will be similar [96].

Surface shape representation

To appreciate the shape of the free surface, which is an important output of our simulations, we display its "trace" on the boundary Γ_{wall}. To do so, we define 90 points $\mathbf{P}_i = (R^\star \cos(\theta_i), R^\star \sin(\theta_i), 0)$ where $R^\star$ is a radius close to R but slightly smaller (to avoid points that would lie outside the discretized domain) and the angles are $\theta_i = i\frac{\pi}{45}$ for $i = 0, \ldots 89$. The height h_i of the free surface above the points $\mathbf{P}_i$ is measured and reported in correspondence of θ_i. For more clarity, each point on the figure is reported twice, as (θ_i, h_i) and as $(\theta_i + 2\pi, h_i)$, see Fig. 5.1.

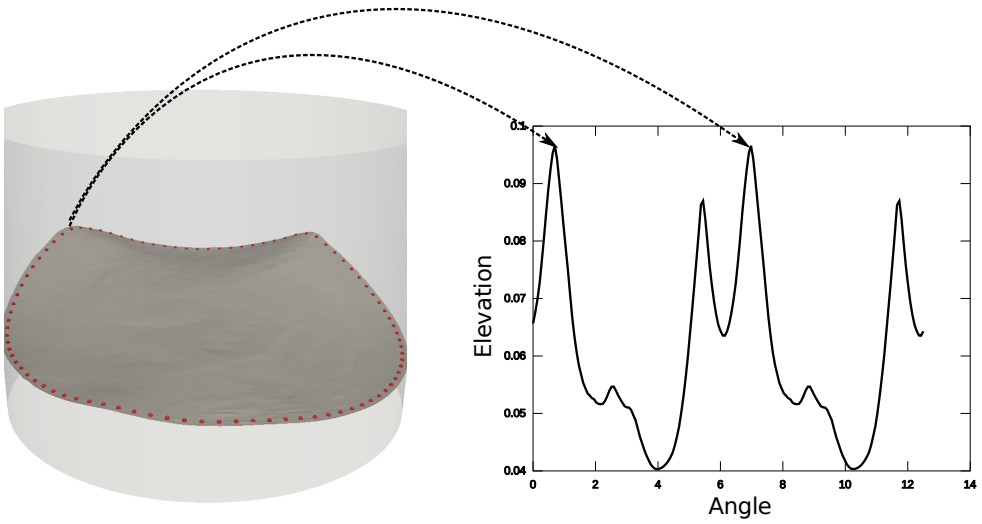

Figure 5.1: Illustration of the way the trace of the free surface is displayed. Red spheres on the left indicate the points $(R^\star \cos(\theta_i), R^\star \sin(\theta_i), h_i)$.

5.1 High-viscosity orbitally shaken reactor

In [28], we performed numerical simulations of OSRs filled with glycerine instead of water. The goal was to circumvent the necessity of stabilizing the high Reynolds case appearing with water filled OSRs. Boundary conditions **A** and **B** (see Sect. 3.2.2) were used to compute the amplitude of the wave, with good agreement with experimental measures in the case of conditions **B**.

However, it was remarked that conditions **B** cannot yield correct stress measurements, due to the lack of vorticity close to the curved wall. Strategy **C** was then proposed but only a preliminary test was performed. We present here further results using strategy **C**. Moreover, thanks to the gain in computational efficiency, we could also afford finer discretizations in both time and space. Indeed, we use here the mesh M, while the XS was used in [28] and a time step of $\Delta t = 5 \cdot 10^{-3}$ while it was 10 times larger in [28].

5.1.1 Comparison with previously obtained results

We compare here the results obtained with the method presented in this work with the results obtained in [28], both using strategy **B** for the boundary conditions, to assess the differences due to the change in discretizations. We recall that the viscosity of both fluids is increased by 1000 with respect to those used for the other simulations, i.e. $\mu_l = 1$ and $\mu_a = 2\ 10^{-2}$. We concentrate on the case with an agitation rate of $\omega = 125$ RPM, since the largest differences between the boundary conditions appeared at that regime. On Fig. 5.2, we compare the wave newly computed with the one obtained in [28] and the experimental one (We do not take into account the wetting of the wall because of the glycerine, since this phenomena does not appear in water-filled OSRs). From that angle, the newest computations seem to better agree with the experimental shape of the surface. On Fig. 5.3, we can observe that the shapes of the two numerical interfaces are quite different, especially with respect to the symmetry of the wave.

Concerning the wave amplitude, the quantity that we use for the comparison with the experiments, we observe that it has not evolved by a significant amount with the new discretization. It is then still in the range of the experimental measures.

5.1.2 Robin boundary conditions and wave shapes

We push now further the tests with Robin-type boundary conditions. The agitation of 125 RPM is simulated again but using conditions **C**. Three different slip lengths l_s have been tested, with values 0.02, 0.03 and 0.05.

First of all, we can remark that the amplitude of the wave is only mildly affected by the change from condition **B** to condition **C** and by the slip length l_s (see Fig. 5.5). This differs from the coarser discretization used in [28] where an increase of the wave amplitude was recorded with condition **C** and a slip length of 0.04. Therefore, the simulations with the finer discretization are more in agreement with the experimental measures. A possible reason is that the finer mesh might allow a better transition between the no-slip and the free slip conditions in strategy **C** thanks to the greater number of elements in the vertical direction.

We also remark that the shape of the free surface is quite similar between the conditions **B** and **C** with the different slip lengths. We can however observe a phase shift between the different waves: the wave with condition **B** is the earliest wave and then, the smaller is the slip length,

Figure 5.2: Comparison of the wave shape computed with the method developed in this work (left) and with the method from [28](right) with the experimental picture (center, courtesy of M. Reclari, LMH-EPFL). Remark that the two computed waves are observed from the same angle.

the later is the wave. For example, the phase shift between condition **B** and condition **C** with $l_s = 0.03$ is of about 15° (see Fig. 5.4).

With smaller slip length than 0.02, the surface starts sticking to the wall, creating films close to it. These small films are unphysical and are actually consequences of the too weak slip allowed. Indeed, when looking, e.g., at Fig. 5.2, we can observe that the film that forms on the wall is always above the surface, while the one obtained with $l_s = 0.01$ (which represents around 5 elements in height for the whole band) and depicted on Fig. 5.6 is alternatively above and below the interface.

On Fig. 5.7, we compare the velocity magnitude on the wall for both the conditions **B** and **C** with $l_s = 0.03$. We remark that with condition **C**, the velocity outside the slip band drops down to zero while it is non-negligible with conditions **B**. This shows that the Robin condition (3.9) works as expected, giving a solution close to $\mathbf{u} = \mathbf{0}$ on Γ_{wall}.

Thanks to the no-slip zone, we can expect to better capture the boundary layer in the case of boundary conditions **C**, as shown on Fig. 5.8.

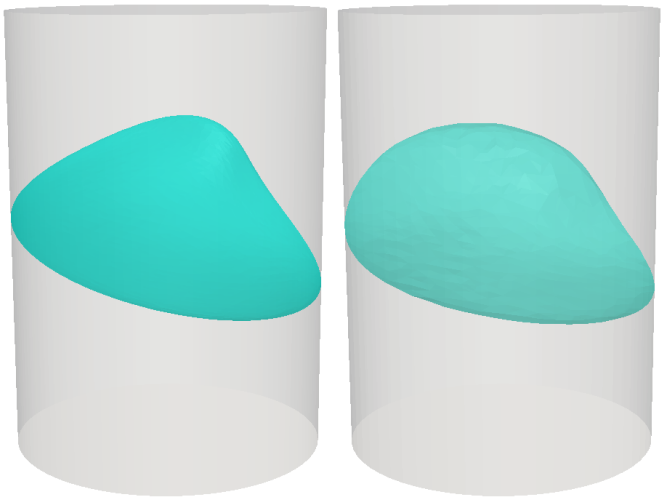

Figure 5.3: Comparison between the two interfaces numerically obtained (from this work on the left, from [28] on the right). Remark that an image of the experiment similar to Fig. 5.2 could not be used for the comparison: with glycerine, the liquid sticks to the walls, which made the wave undistinctive from this angle.

Figure 5.4: Difference of wave shape between conditions **B** (transparent red) and **C** with $l_s = 0.03$(solid color).

5.1.3 Robin boundary conditions and hydrodynamic stress

One of the reasons for devising Robin boundary conditions **C**, was to obtain better values for of the hydrodynamic stress. To show that, we compare the computed strain magnitude $\frac{1}{2}\|\nabla\mathbf{u} + \nabla\mathbf{u}^T\|_2$ with the different conditions.

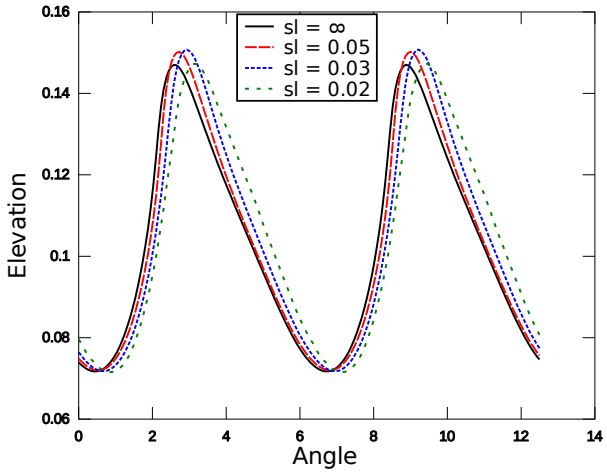

Figure 5.5: Shape of the waves produces by the different slip length (condition **B** can be seen as **C** with infinite l_s, it is therefore represented by $l_s = \infty$).

Figure 5.6: Shape of the surface obtained with condition **C** and a slip length of $l_s = 0.01$

On the boundary of the cylinder, we can observe on Fig. 5.9 large differences between all the conditions. The highest strain magnitude while using condition **C** is located in the zone where the transition between slip and no-slip happens. This peak in the strain is then certainly more due to the numerical treatment of the wall boundary condition than reflecting a physical reality.

In the bulk of the reactor however, we can see on Fig. 5.10 that with $l_s = 0.02$ and $l_s = 0.03$ (the two smallest slip lengths), the strains are similar, with only differences near the contact line. The strain in these cases is higher than that with conditions **B** and than with greater slip, so it

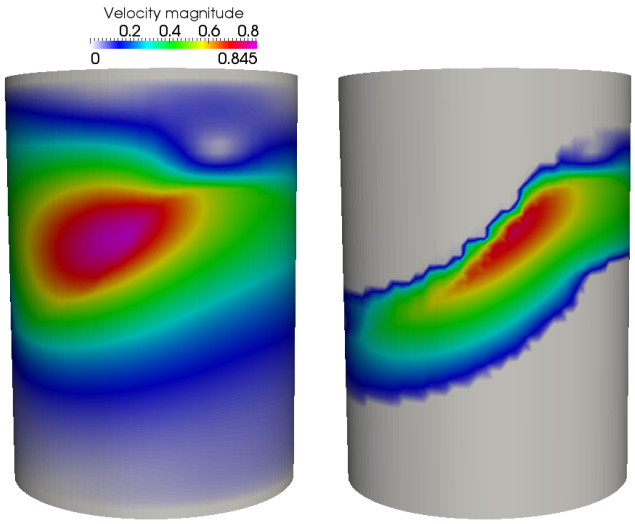

Figure 5.7: Velocity magnitude on $\partial\Omega$ for boundary conditions **B** (left) and **C** with $l_s = 0.03$ (right).

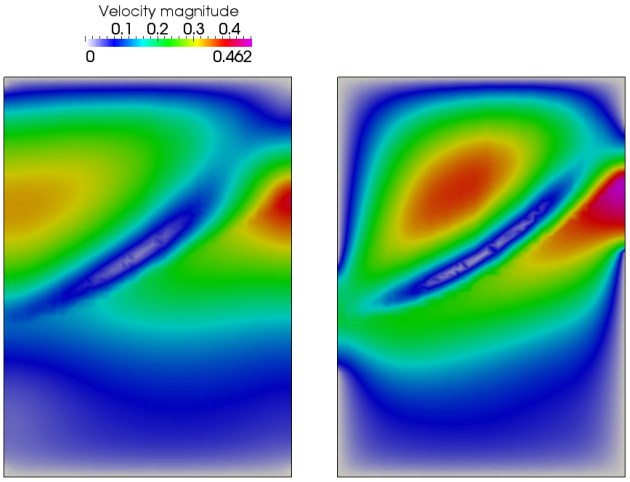

Figure 5.8: Velocity magnitude on the plane $y = 0$ at $t = 8$s, with boundary conditions **B** (left) and **C** with $l_s = 0.03$ (right). The boundary layer is visible outside the slip band on the right.

is probably captured in a better way.

We also investigate the influence of the refinement of the mesh near the wall. Comparing the results obtained previously (where $\alpha_{BL} = 2$) with the same mesh M and no refinement $\alpha_{BL} = 0$, we observe no significant difference on the surface shape and the strain values, both on Γ_{wall} and in the bulk of the fluids.

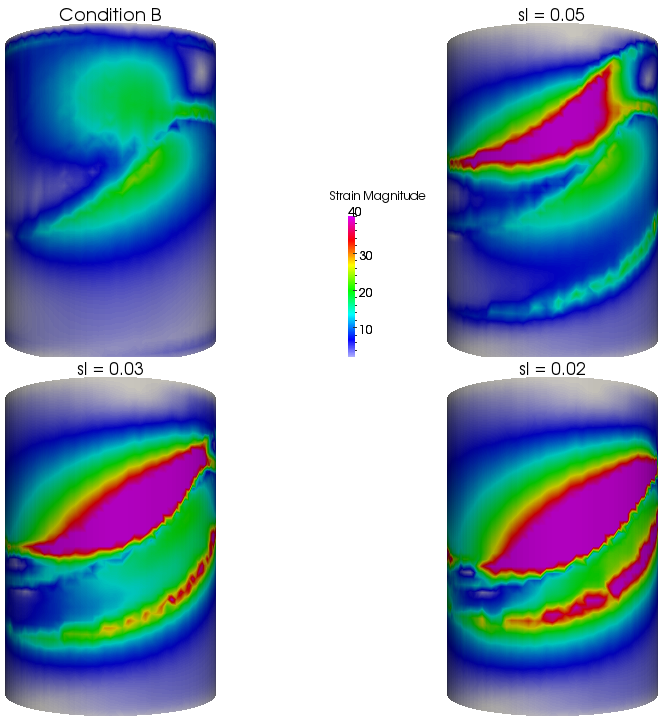

Figure 5.9: Strain magnitude on the surface of the reactor for condition **B** (top left) and condition **C** with $l_s = 0.05$ (top right), $l_s = 0.03$ (bottom left) and $l_s = 0.02$ (bottom right).

5.2 Wave pattern

A first step towards the validation of our method and our code for the simulation of water filled OSRs is to correctly capture the different regimes that the free surface can show, depending on the parameters (R, H_0, ω, R_s), or more precisely on $(\mathrm{Fr}, \Pi_2, \Pi_3)$. Indeed, these three adimensional parameters are sufficient to determine which regime is met and therefore which wave pattern is found.

Different wave shapes can be observed in OSRs and they can be classified according to two criteria.

1. The first criterion is the number of peaks that appear on the wave. Most of the waves have one peak and one trough, usually opposed in the container. However, for regimes with rather small Π_2, i.e. a small amount of liquid, waves can feature several peaks. Double waves, triple waves and even quadruple waves have been observed experimentally, with lower Fr and Π_2 numbers producing the waves with the highest number of peaks.

2. The second criterion for the classification of the waves is whether they are breaking or not. Breaking waves feature a peak that has become too large and collapses under its weight. They appear with sufficiently high agitation rates, so high Fr number. Most of the breaking waves are single wave, i.e. they feature only one peak, but breaking waves

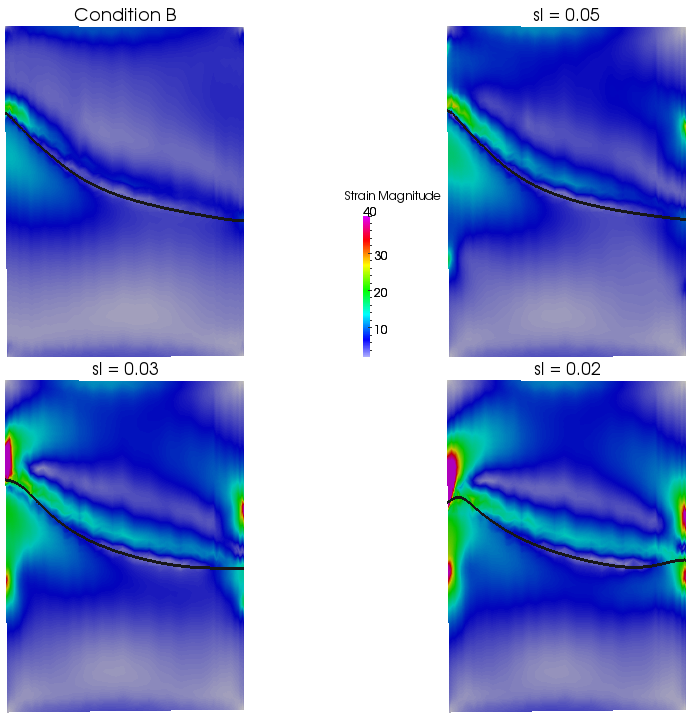

Figure 5.10: Strain magnitude on a cut through the reactor for condition **B** (top left) and condition **C** with $l_s = 0.05$ (top right), $l_s = 0.03$ (bottom left) and $l_s = 0.02$ (bottom right). The black line indicates for each situation the position of the interface.

with multiple peaks have been observed experimentally in specific regimes.

These different wave patterns have not been observed with glycerine filled OSRs because of the high viscosity, which seem to force the wave to keep a unique non-breaking peak. For water filled OSRs, we select several configurations for which the type of wave produced is known and compare it with numerical simulations. Some types of waves can be rather easily obtained while others require more accuracy to appear.

This gives also another motivation for this work: the type of wave produced by an OSR represents a very efficient test case for free surface flow solvers. Indeed, the experimental setup is extremely simple to reproduce and the numerical simulations can exactly match it, which is not obvious for classical test cases such as the dam break [67, 66, 58] or the rising bubble [53]. Moreover, the geometry of this test offers also the possibility to test the boundary conditions which are essential to get physically relevant results.

The different wave patterns are obtained by keeping the adimensional number Π_2 fixed to 0.1736 and varying Fr and Π_3. The table in Fig. 5.11 reports several experiments carried out and the wave pattern observed for the corresponding regime.

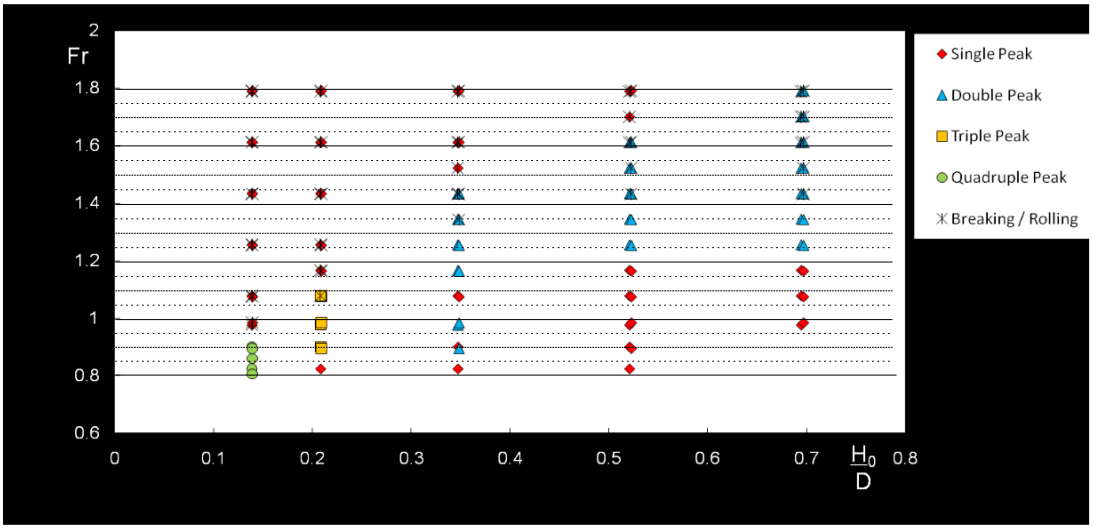

Figure 5.11: Table showing the type of wave observed experimentally for different regimes when $\Pi_2 = 0.1736$ (courtesy of M. Reclari, LMH-EPFL).

5.2.1 From single to breaking wave

The first test that we conducted is to examine the wave produced for a certain Π_3 value and different agitation rates, hence with different values of Fr. We choose the following parameters:

$$R = 0.144m \qquad H_0 = 0.15m \qquad R_s = 0.025m$$

and vary the agitation rate ω, from 50 to 100 RPM. The value of the adimensional number Π_3 is then 0.5208. According to the table in Fig. 5.11, the waves observed experimentally are:

- from 50 to 65 RPM, a single wave,
- from 70 to 75 RPM, a double wave,
- from 80 to 95 RPM, a breaking double wave,
- from 100 RPM on, a breaking single wave.

The simulations that we perform use ω from 50 to 100 RPM, with a difference of 10 RPM between the configurations, so that every type of wave is represented at least once.

50 **RPM**

At 50 RPM, the Froude number is Fr = 0.8973 and the wave is a very simple single wave, with a low amplitude (see Fig. 5.12). The trough and the crest of the wave are in opposed positions in the container, as it can be usually observed at low regimes.

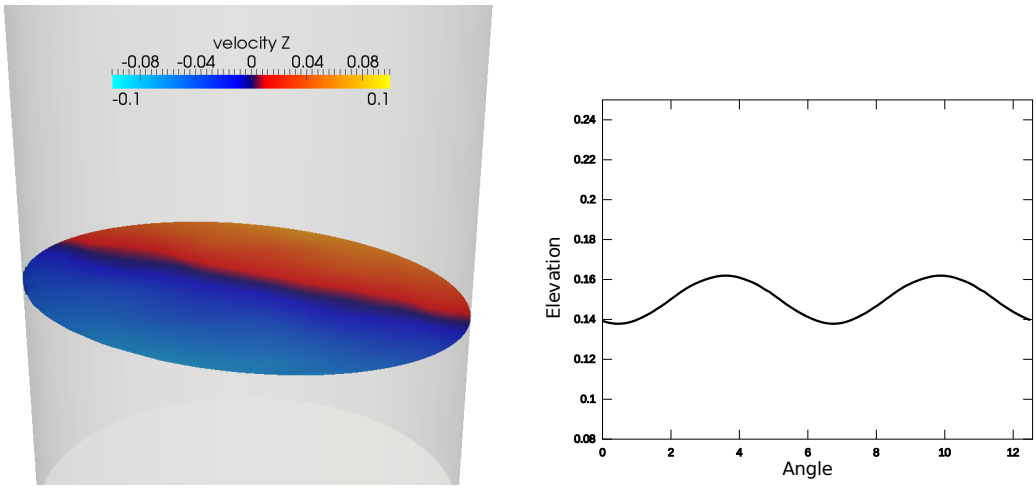

Figure 5.12: Shape of the free surface at 50 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

60 **RPM**

At 60 RPM, i.e., Fr = 1.0768, the wave is still a single non-breaking wave, but its amplitude has increased with respect to the one observed at 50 RPM. Moreover, we can see on Fig. 5.13 that the wave has lost its symmetry.

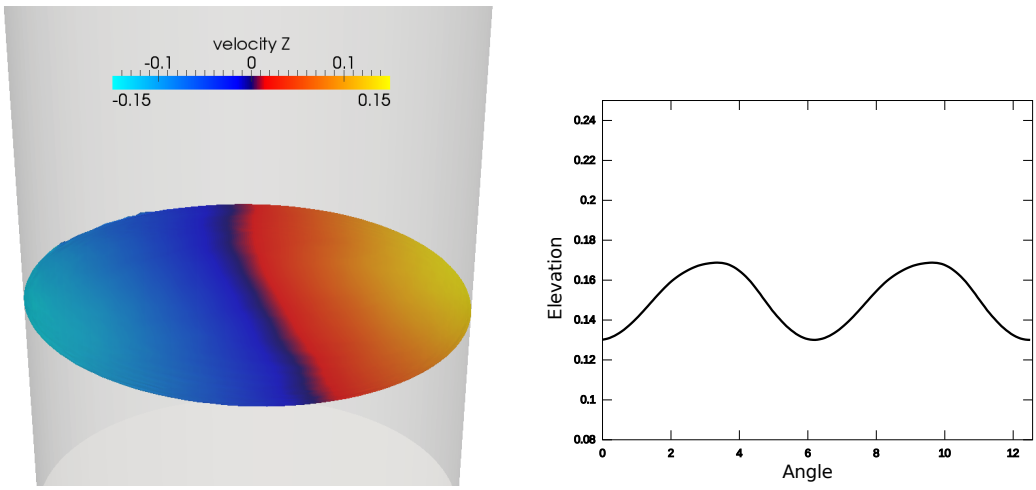

Figure 5.13: Shape of the free surface at 60 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

70 **RPM**

At 70 RPM, Fr = 1.2563, we can observe on Fig. 5.14 that the wave has now two well-formed peaks. The first peak does not seem to break, even if it is very steep and breaks immediately after the end of the acceleration phase.

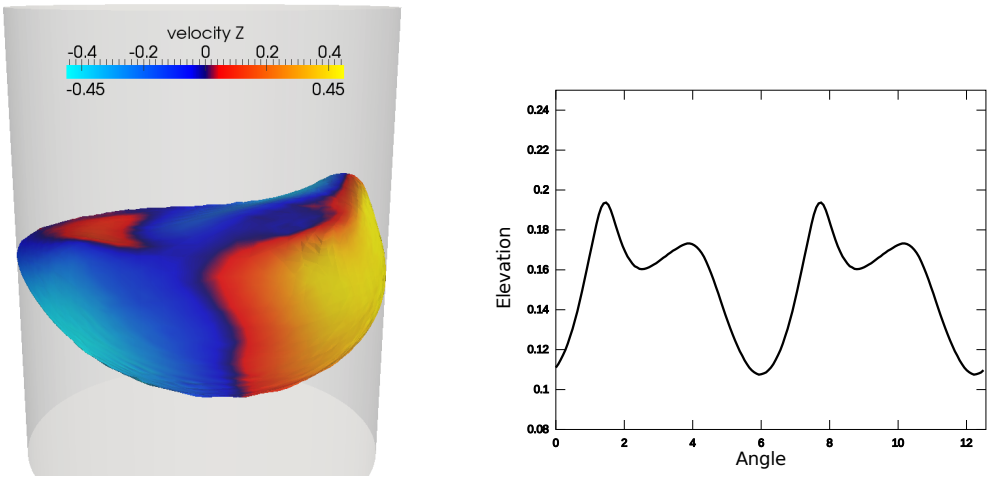

Figure 5.14: Shape of the free surface at 70 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

80 **RPM**

At 80 RPM, the first peak of the double wave starts breaking, as it can be seen of Fig. 5.15. The Froude number is Fr = 1.4357 at that regime.

90 **RPM**

At 90 RPM, the Froude number is Fr = 1.6152 and the wave has again a double peak with the first one breaking (see Fig. 5.16). This shape is not coming directly like the other waves, in the sense that after the acceleration phase, a single breaking wave is observed. More than 10 seconds are required for the second peak to appear on the wave.

100 **RPM**

At 100 RPM, a single peak wave is observed and it continuously breaks (see Fig. 5.17). The Froude number, Fr = 1.7946, is the largest we consider in this test.

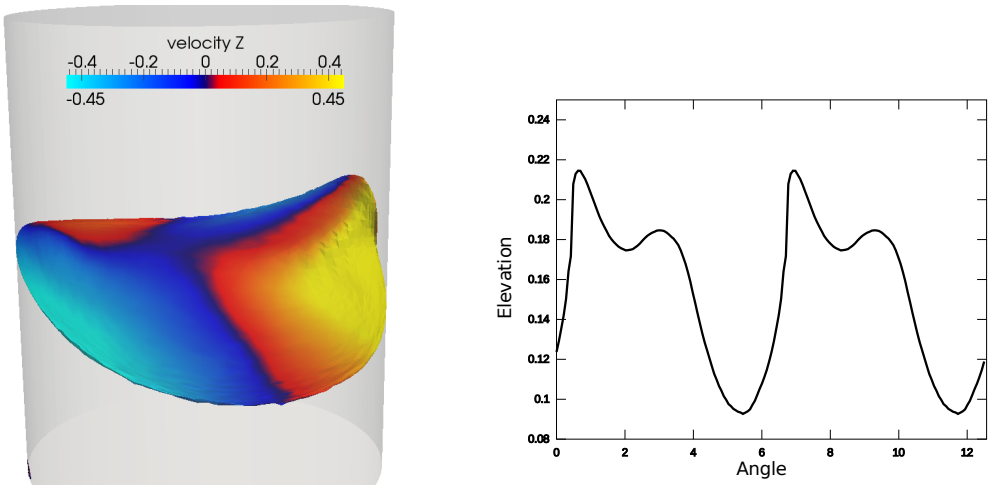

Figure 5.15: Shape of the free surface at 80 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

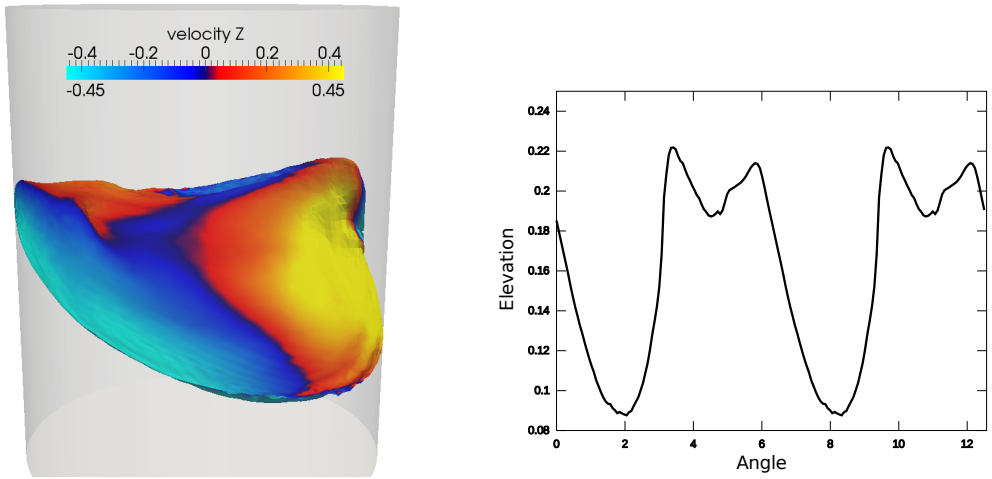

Figure 5.16: Shape of the free surface at 90 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

5.2.2 The triple wave

Among the considered wave patterns, the triple wave was the most difficult to reproduce numerically. Experimentally, triple waves are obtained near the configuration $(\text{Fr}, \Pi_2, \Pi_3) = (0.987, 0.1736, 0.208)$. We used the following values for the real parameters:

$$R = 0.144 \text{ m} \qquad H_0 = 0.06 \text{ m} \qquad \omega = 55 \text{ RPM} \qquad R_s = 0.025 \text{ m}$$

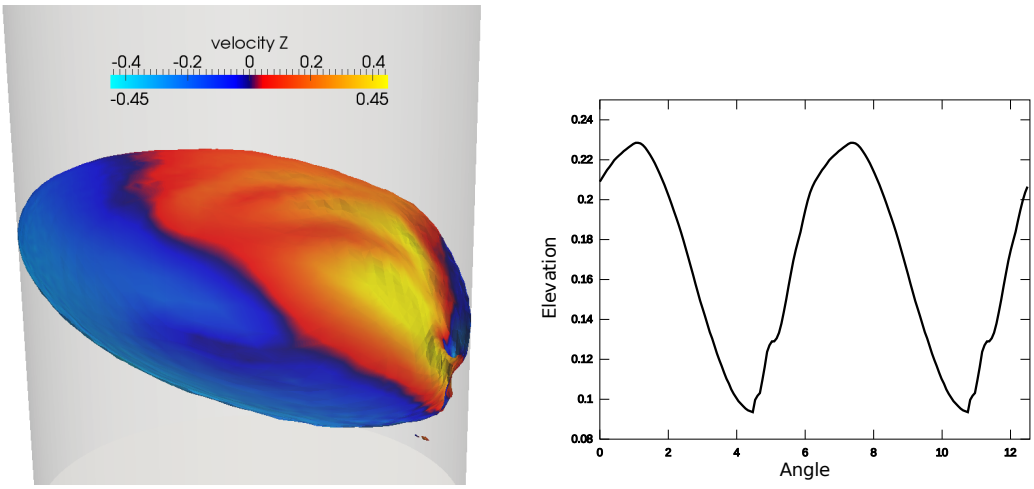

Figure 5.17: Shape of the free surface at 100 RPM (colored by vertical velocity) and its trace on Γ_{wall}.

The window in which triple waves can exist is very tiny, as shown on Fig. 5.11. The scheme must then be very accurate to keep the solution in that window. In particular, a too diffusive scheme would drive the solution out of the window very quickly.

Using the same discretization as for the previous cases, i.e. the mesh M and a time step of 0.002 s, the result of the simulation shows rapidly changing shapes in the sense that the wave starts as a single peak wave, then a second and a third peak appear, the first peak breaks after that and finally the third peak disappears and the wave remains a double peak, with all the phenomena appearing in 12 seconds.

As this is not satisfactory, we investigated the influence of the different approximations and we found that using the mesh M with $\alpha_{BL} = 3$ was sufficiently accurate for the space discretization and that the critical parameter was the time discretization. Indeed, we use a shorter time step of $\Delta t = 10^{-3}$s to get a stable triple wave, as shown on Fig. 5.18. This wave is still quite unsymmetric, the peak appearing last being smaller than the other two peaks, but it seems that it actually gains in importance as the simulation goes on, as depicted on Fig. 5.19.

This represents a great achievement for two reasons. First of all, we are able to capture complicated wave patterns accurately, since only small deviations were allowed here. Secondly, the fact that the critical parameter, for the mesh M, is the time step indicates that the time discretization is the most limiting element of the scheme in that case. This means that the integration, the pressure correction, the different stabilizations and the reinitialization introduce less error and affect less the results.

We also investigated the sensitivity of the results with respect to the slip length used. Using three different lengths, $l_s = 0.02$, 0.03 (used for the results presented before), 0.05 and 0.1, we

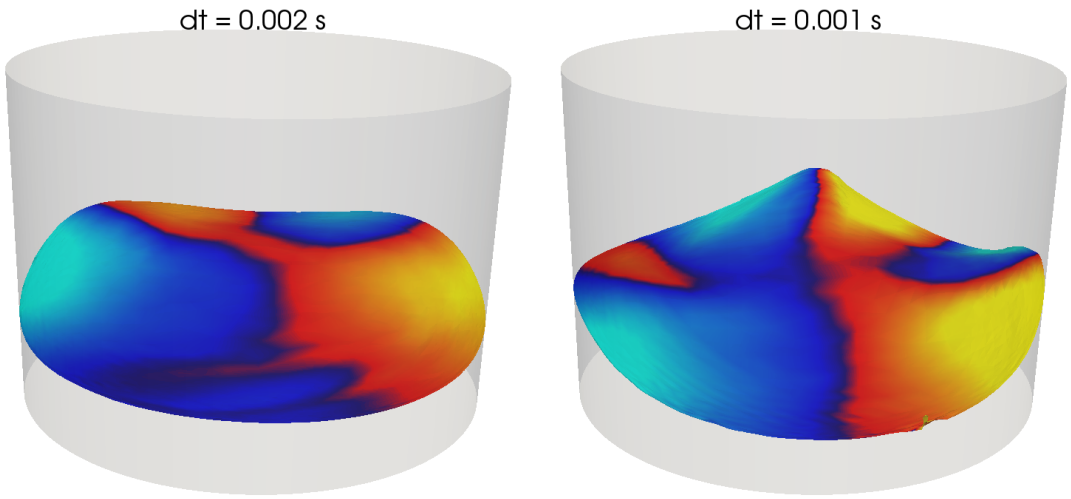

Figure 5.18: Comparison between the shape obtained after $t = 17$s with time steps $\Delta t = 0.002$s (left), which is clearly a double wave, and with $\Delta t = 0.001$s, where a triple wave is visible (right). The two free surfaces are colored by vertical velocity, a yellow-red color indicates a positive value while blue colors indicate negative values.

find that all of them lead to triple waves, with however small differences (see Fig. 5.20):

- When a too small slip length is used (e.g., 0.02), we observe that the wave has a phase shift with respect to the others, a phenomena already observed with glycerine filled OSRs. Moreover, the third peak appears smaller, while the two first ones still have comparable sizes.
- A too large slip length also makes the third peak smaller, while the two first peaks have correct sizes.
- In between these two extremes, the slip lengths can lead to nicely developed triple waves, as we saw with $l_s = 0.03$.

We conclude that the slip length does not need to be extremely precisely set, but must still be in a range where sufficient elements are used for the transition between no-slip and slip condition (i.e., l_s sufficiently large), without being too large.

5.3 Validation with laser Doppler velocimetry measures

To obtain a stronger validation of our method, we need experimental measures that would allow to quantify how close our results are to the physical truth. In this work, laser Doppler

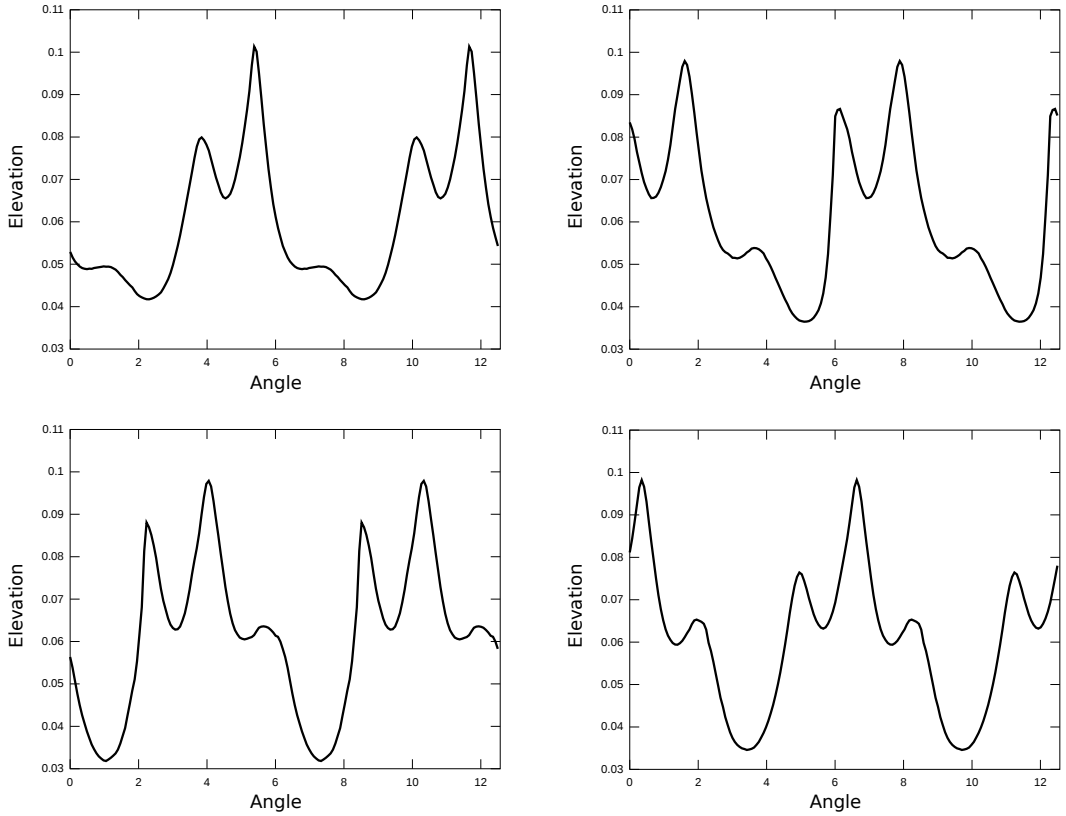

Figure 5.19: Wave shape for $\Delta t = 0.001$ s at $t = 5$s (top left), $t = 10$s (top right), $t = 15$s (bottom left) and $t = 20$s (bottom right).

velocimetry (LDV) is used. The LDV technique uses two coherent laser beams to measure the velocity of small dies floating in the liquid: the two beams interfere and create fringes, see Fig. 5.21. When a die crosses this set of fringes, it reflects them and the velocity perpendicular to the fringes can be extracted from the frequency at which the reflected light fluctuates.

In this work, we compare the radial and tangential components of the velocity with experimental measurements. The configuration consists in an OSR with radius $R = 0.15$m, a liquid height $H_0 = 0.2$m, an agitation rate of $\omega = 60$ RPM and a shaking radius of 0.025m. The velocities have been measured in different points of the reactor. We focus on 4 different radii for the 4 heights available: the radii are $\{0.01, 0.05, 0.09, 0.13\}$ and the heights $\{0.05, 0.08, 0.10, 0.14\}$. We remark that, due to limitations inherent to the measurement method, it is not possible to obtain measures of the velocity in the boundary layer nor closer to the interface Γ. During the experiments, after the initial acceleration phase, one has to wait some time, several tenths of seconds, before starting the measures, otherwise they strongly fluctuate because the fluid has not reached a periodic state yet.

The same holds true for the numerical simulations: after the acceleration phase, the fluid

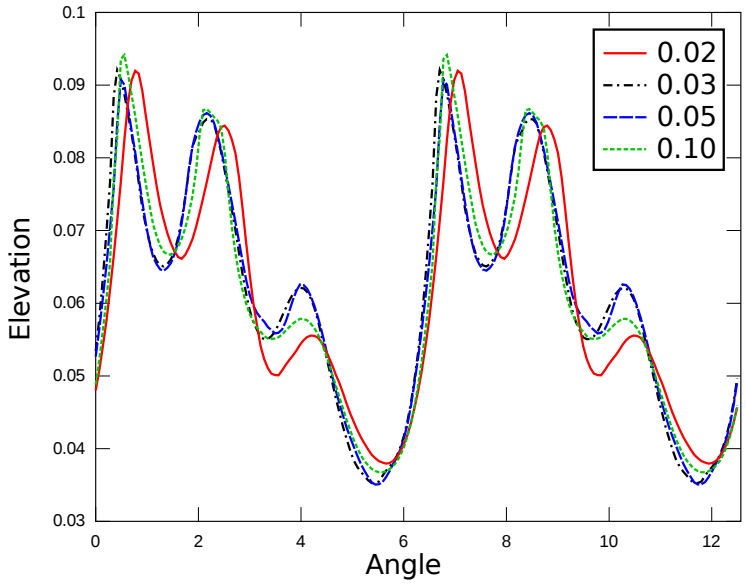

Figure 5.20: Wave shape at $t = 9$s for different slip length l_s.

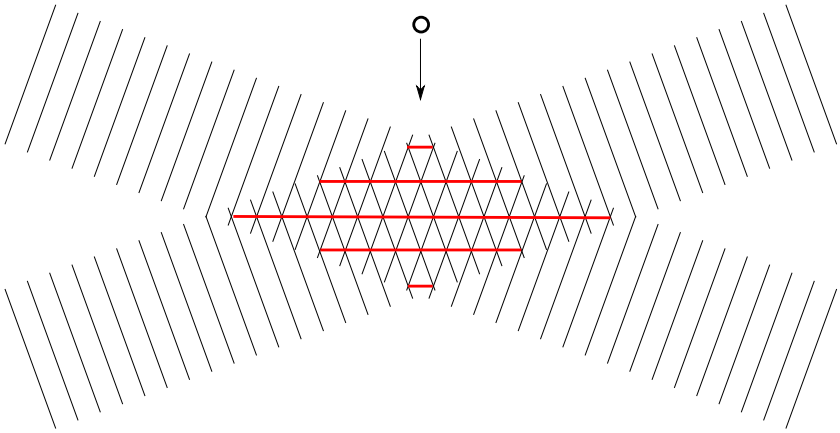

Figure 5.21: Illustration of the principle of the LDV technique.

remains in a transient state for a very long while. This phenomenon is more important with small amounts of numerical viscosity, since the more damping is present, the quicker the fluid reaches a periodic state, but also the worse are the results. Fig. 5.22 shows the radial velocity measured at the point $(0.09, 0, 0.10)$ in the OSR over a long period and how it fluctuates in time.

To compare the numerical results with the experimental ones, we compute the radial and tangential velocities over a given time interval and plot the average velocities found. We also show the standard deviation to better appreciate the size of the fluctuations. We used the mesh M, a time step $\Delta t = 0.002$s and an acceleration interval of length $T_{\text{acc}} = 5$s. The slip length was set to $l_s = 0.05$m. We compute the average velocities in the interval between 30s and 40s

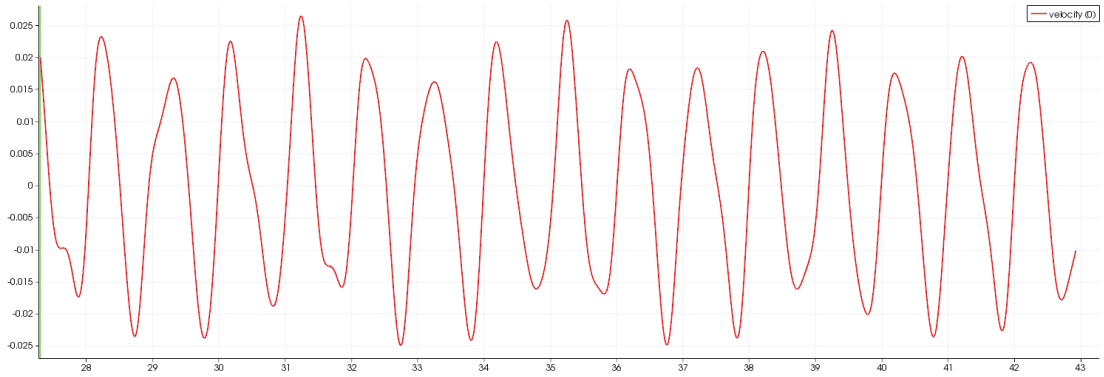

Figure 5.22: Radial velocity measured at the point $(0.09, 0, 0.10)$ as a function of time.

and we use 501 measures (one each 0.02s). We compare the experimental velocities and the ones numerically obtained in all the 16 locations and we present here 4 of them which are representative of the general behaviour observed. In the figures comparing the experimental and the numerical velocities (e.g., Fig. 5.23), we display both the average velocities and the standard deviations observed and computed.

First of all, in the bulk of the fluid, we obtain results that are in good agreement with the experimental ones. This is illustrated by the results obtained for a radius $r = 0.09$m and height of $z = 0.1$m in Fig. 5.23 and 5.24. We can observe that both velocities are well-captured.

Apart from the bulk, the interface area is of special interest since it can give strong indications of whether our treatment yields a good accuracy at that location. We used the radius $r = 0.01$m and the greatest height $z = 0.14$m and observe the differences in Fig. 5.25 and 5.26. Again, both velocities are equally well-captured, the only noticeable difference lying when the largest velocities are reached in the experiments. Even if these measures are not taken in the vicinity of the interface, we can see that the approximation does not worsen when moving towards the direction of the interface.

Surprisingly, the results obtained at the bottom of the container are worse than in the bulk. Looking at the measure with radius $r = 0.09$m and with an height of $z = 0.05$m (the lowest one), we observe that the radial velocity matches very closely the experimental one (even the standard deviations look similar), see Fig. 5.27 and 5.28. Even if the general trends are comparable, the numerical tangential velocity is shifted from the experimental one, by around 0.007m/s. This might indicate that, in the time interval considered, the periodic state that we are looking for was not yet reached close to the bottom.

Finally, we investigate the measures for the radius $r = 0.13$ (the largest) and the height $z = 0.14$ (the highest) which is the measure point located the closest to the contact line. The amplitude of the radial velocity is larger than the one measured experimentally, but the shape looks quite similar, with a slow increase and an abrupt decrease, as depicted in Fig. 5.29 and 5.30. The

tangential has also the right trend, even if it looks again shifted with respect to the experimental one. However, by considering that we are simulating the physics of the moving contact line with a simple Robin condition, these results are rather satisfactory.

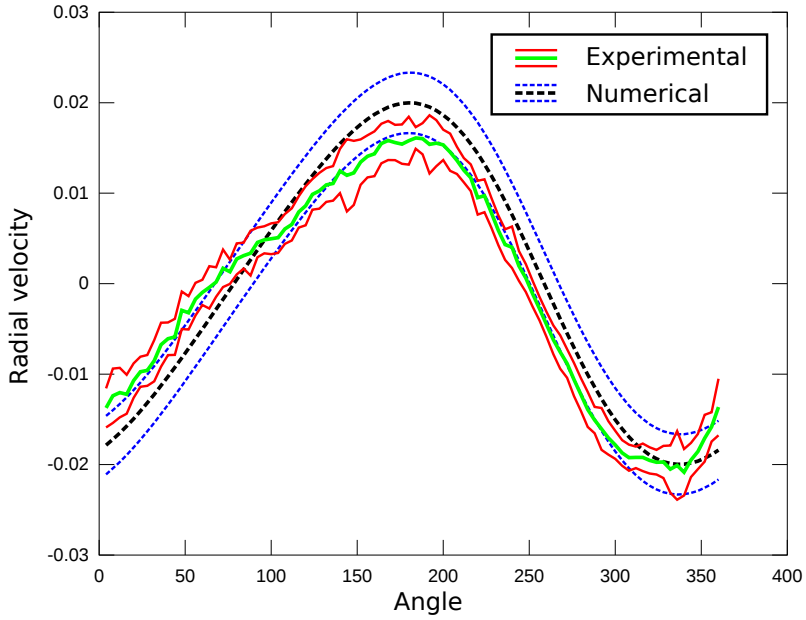

Figure 5.23: Comparison of the radial velocity at $r = 0.09$m and $z = 0.1$m. The plain green line represents the average velocity experimentally observed and the two plain red lines the average velocity plus and minus the standard deviation. The black dotted line is the average velocity computed and the two blue dotted lines represent the average velocity plus and minus the standard deviation computed.

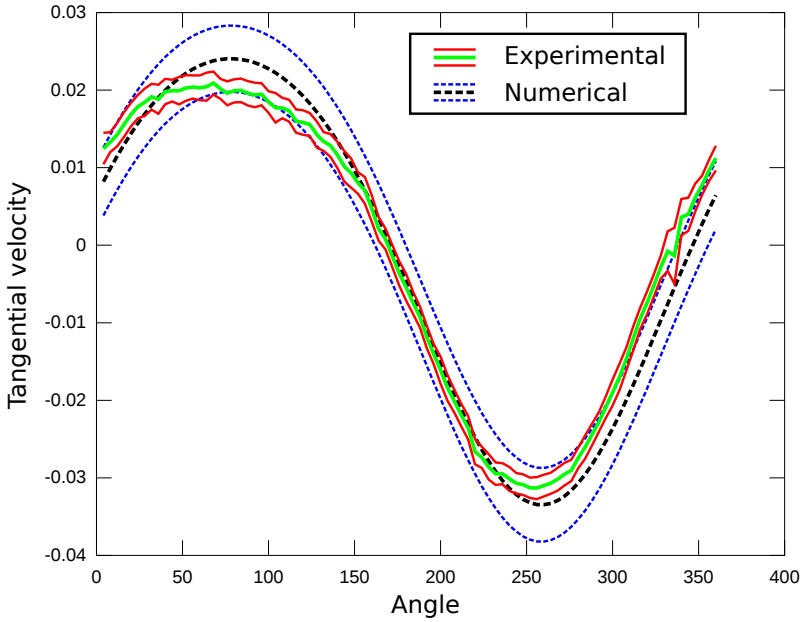

Figure 5.24: Comparison of the tangential velocity at $r = 0.09$m and $z = 0.1$m.

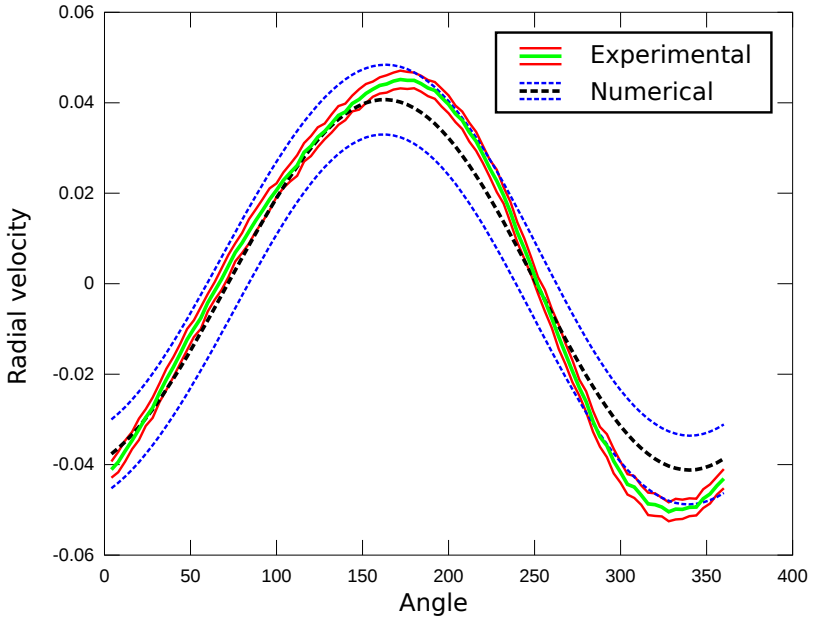

Figure 5.25: Comparison of the radial velocity at $r = 0.01$m and $z = 0.14$m.

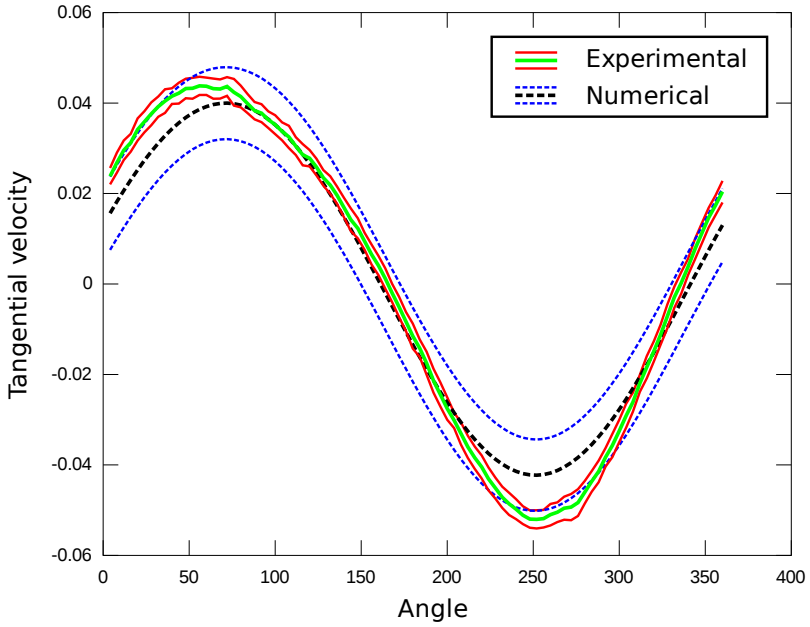

Figure 5.26: Comparison of the tangential velocity at $r = 0.01$m and $z = 0.14$m.

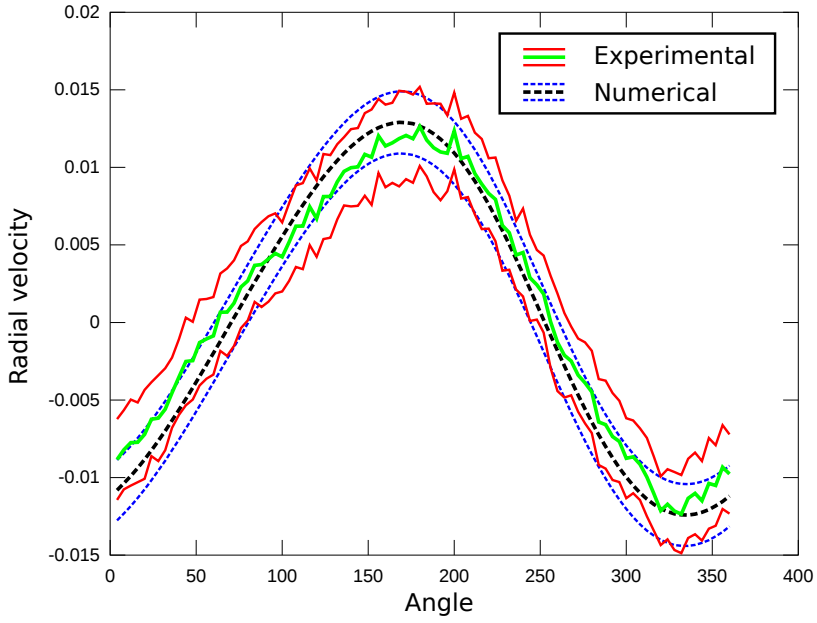

Figure 5.27: Comparison of the radial velocity at $r = 0.09$m and $z = 0.05$m.

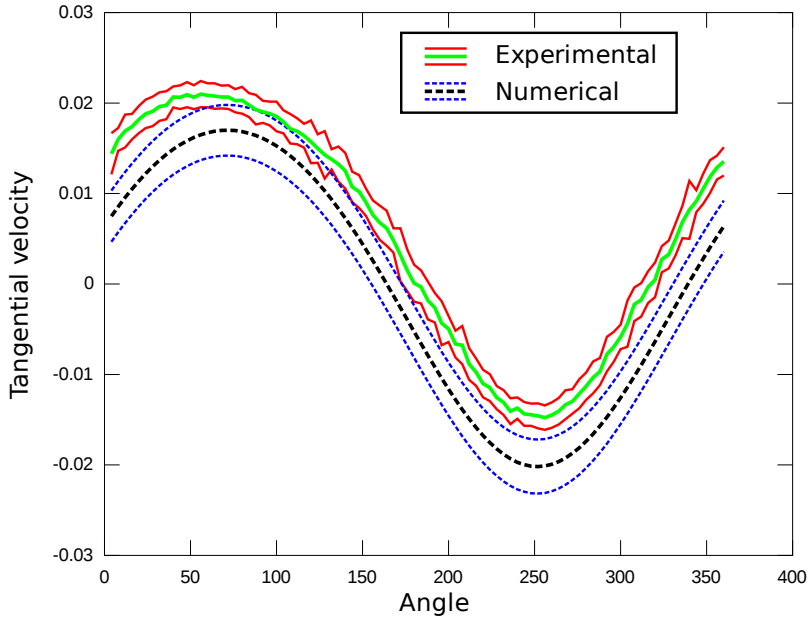

Figure 5.28: Comparison of the tangential velocity at $r = 0.09$m and $z = 0.05$m.

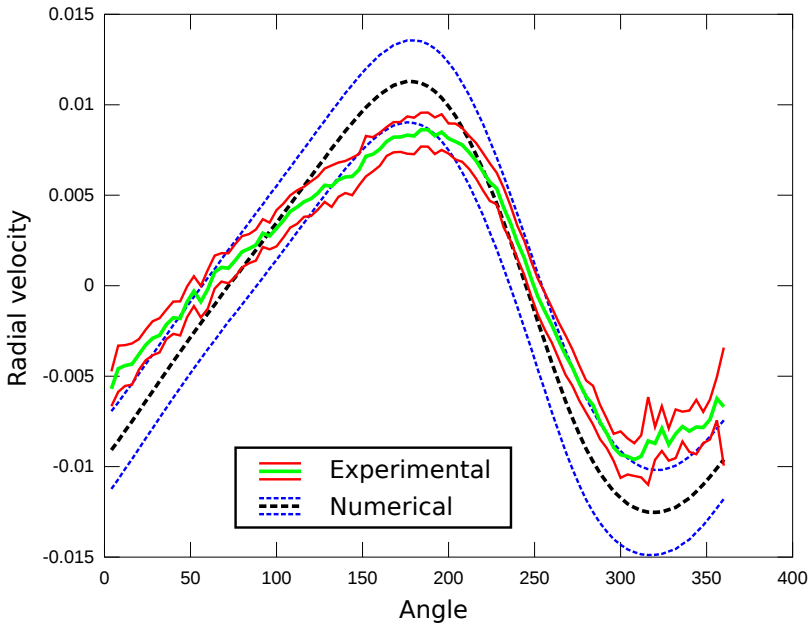

Figure 5.29: Comparison of the radial velocity at $r = 0.13$m and $z = 0.14$m.

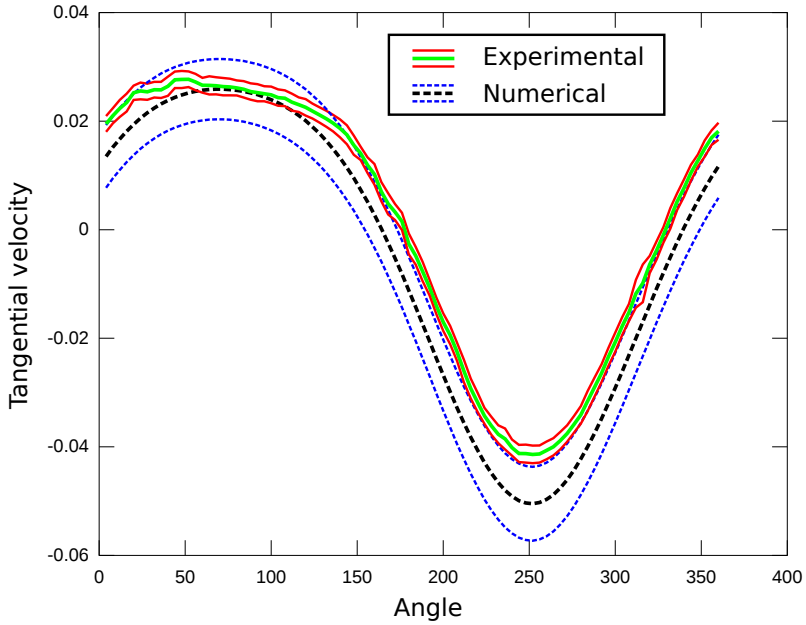

Figure 5.30: Comparison of the tangential velocity at $r = 0.13$m and $z = 0.14$m.

5.3.1 Influence of the correction for normal boundary conditions

In Sect. 4.2, we already showed that when the boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$ is imposed strongly, a correction term must be added to prevent the rise of spurious velocities. This is actually not only useful to obtain the exact solution in case the reactor is standing still: if the correction is not added, we found that the magnitude of the spurious current increases in time. For a simulation needed on a long time interval, this spurious current can reach an intensity greater than the physically relevant one, thus making the results useless. Fig. 5.31 shows the velocity obtained with the case under study but $\omega = 0$ RPM. We can see that the velocity magnitude can reach 0.1 m/s, which is of the order of the experimental measures. The velocities measured on simulations performed without the correction were systematically not approaching a periodic state because their tangential components increased during the whole simulations.

5.3.2 Influence of the slip length

A parameter that could play an important role, at least for the velocity near the interface, is the slip length l_s used for the Robin-type boundary condition. Indeed, we could expect that, with a larger slip length, the velocity can have a greater magnitude near the contact line. We simulated the bioreactor in the time interval $(0, T)$ with $T = 30$s and the size of the acceleration interval was set to $T_{\text{acc}} = 3$s. We used a larger time step $\Delta t = 0.005$s than before to lower the computational effort. Two slip lengths of $l_s = 0.05$m and $l_s = 0.02$m, representing respectively 25% and 10% of the liquid height H_0, are tested.

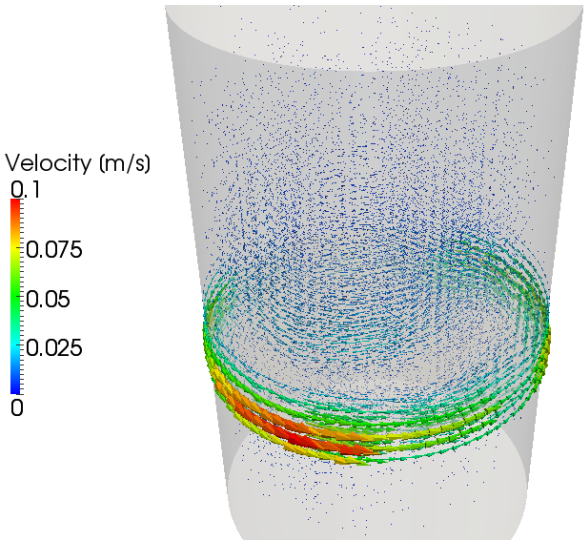

Figure 5.31: Velocity obtained for the case $\omega = 0$ RPM at $t = 60$s.

First of all, we can observe the difference on the shape of the surface. Since the wave generated in the OSR is a simple non-breaking wave, we expect that the shape of the interface will not depend strongly on the slip length, if it is chosen reasonably, i.e., neither too small nor very large. Indeed, we observed that the interfaces at time $t = 30$s were very similar between the simulations with the two slip lengths, the only small difference observed being the larger slip length giving rise to a slightly taller wave, see Fig. 5.32.

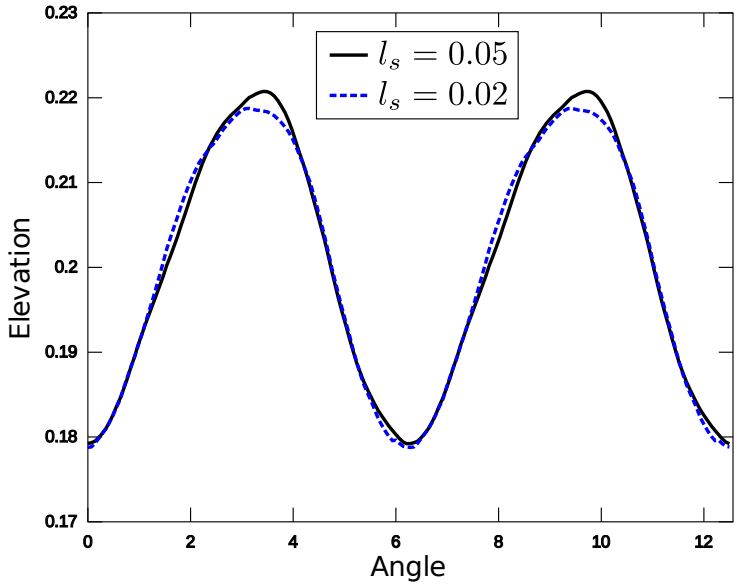

Figure 5.32: Free surface shape for the LDV measure test with slip length $l_s = 0.05$ and $l_s = 0.02$.

We then inspect the velocity at different locations of the bioreactor. The closest point to the contact line, where the largest differences are expected, is located at $r = 0.13$m from the center of the container, i.e. 0.02m from the lateral wall, and at $z = 0.14$m from the bottom, approximatively 0.035m below the trough of the wave. Therefore, it is located at the height of the slip band if $l_s = 0.05$m, but not when $l_s = 0.02$. Fig. 5.33 and Fig. 5.34 show the radial and tangential velocities for both slip lengths. We can observe that the two computed velocities are very close one to each other. Only the tangential velocity is larger when the trough passes close to the measure location for $l_s = 0.05$m.

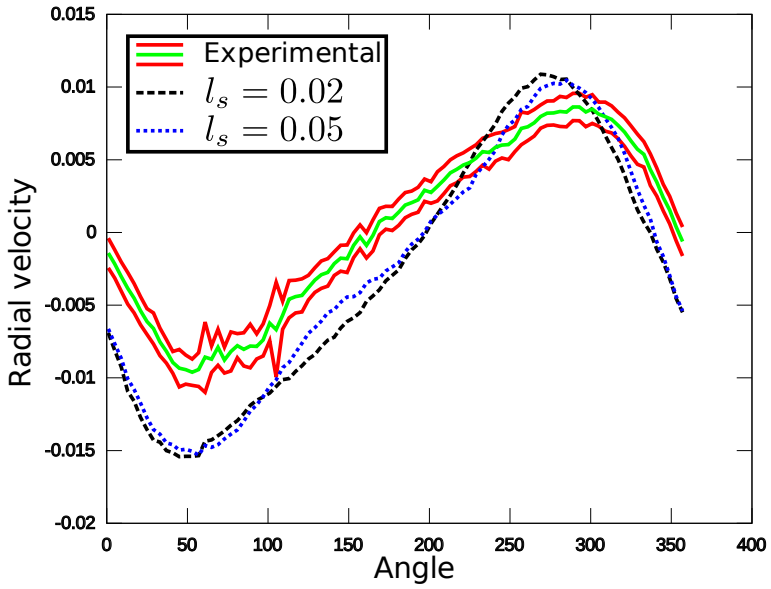

Figure 5.33: Comparison of the radial velocities computed with slip length $l_s = 0.05$m and $l_s = 0.02$m with the experimental measures. The distance to the center was $r = 0.13$m and the height $z = 0.14$m.

At all the other points where measures were taken, the velocities share the same similarities, see, e.g., Fig. 5.35 and Fig. 5.36. Fig. 5.37 shows the difference between the velocities computed at time $t = 30$s considering the smallest ($l_s = 0.02$) and the largest ($l_s = 0.05$) slip lengths.

5.4 Hydrodynamic stress

Mammalian cells are more fragile than other cells and so more care must be adopted when cultivating them. In particular, the hydrodynamic stress should be kept as low as possible to avoid tearing up their membranes. We investigate here how the different regimes influence the stress. To this aim, we reuse the configurations from Sect. 5.2, so that we can observe the correlation with the interface shape.

For the different regimes, from 50 RPM to 100 RPM, we compute the strain $\frac{1}{2}||\nabla\mathbf{u} + \nabla\mathbf{u}^T||_2$,

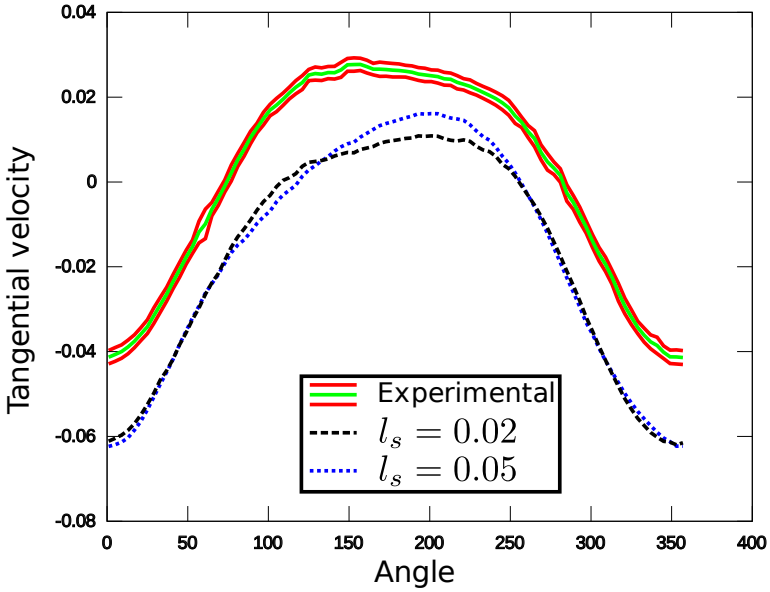

Figure 5.34: Comparison of the tangential velocities computed with slip length $l_s = 0.05$m and $l_s = 0.02$m with the experimental measures. The large mismatch between numerical and experimental velocities is due to suboptimal choices of parameters used to speed up the simulations. The distance to the center was $r = 0.13$m and the height $z = 0.14$m.

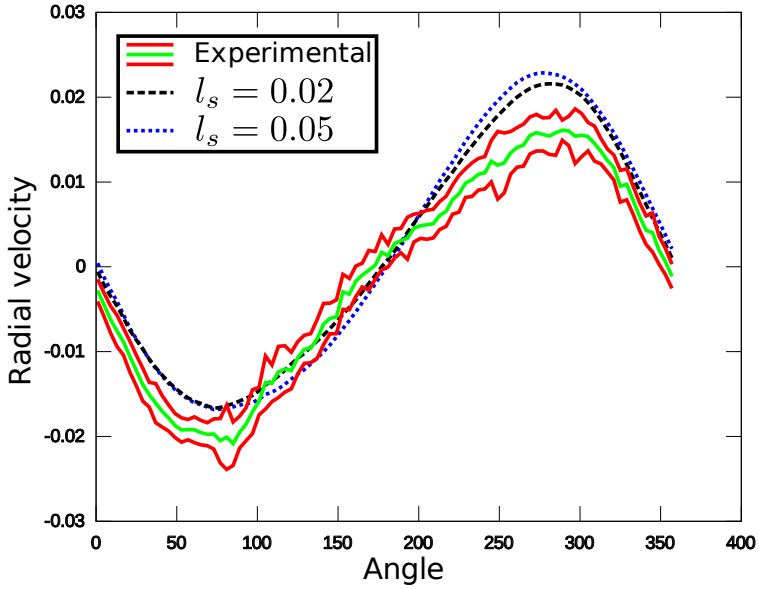

Figure 5.35: Comparison of the radial velocities computed with slip length $l_s = 0.05$m and $l_s = 0.02$m with the experimental measures. The distance to the center was $r = 0.09$m and the height $z = 0.10$m.

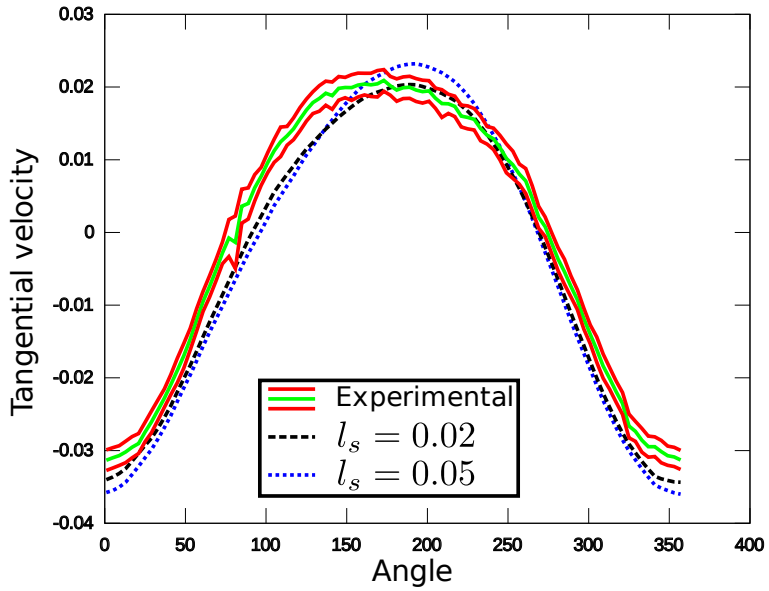

Figure 5.36: Comparison of the tangential velocities computed with slip length $l_s = 0.05$m and $l_s = 0.02$m with the experimental measures. The distance to the center was $r = 0.09$m and the height $z = 0.10$m.

knowing that it might be not accurate close to the contact line, as this was the case in Sect. 5.1.3.

At low RPM and up to 60 RPM, when the wave is a single non-breaking wave, we find very low levels of strain, as highlighted by Fig. 5.38. The highest strain magnitudes are found near the boundary and nearly no strain is found in the bulk of the liquid.

At 70 RPM, a small region of higher strain is created near the front of the first peak, see Fig. 5.39. However, the double wave does not seem to affect the rest of the liquid, where the strain remains quite low. At 80 RPM, the breaking wave creates a larger area of strain. Large strains start also to appear elsewhere in the container, e.g. near the bottom. This phenomena is even more visible at 90 RPM, where the double breaking wave is also present.

Finally, at 100 RPM, the strain does not seem to be much higher than at 90 RPM. The large strains are located near the breaking wave and close to the boundaries, e.g., at the bottom.

From the point of view of the strain, the lowest agitation rate is of course the best, since it yields the lowest stress possible. However, other mecanisms triggered by the agitation are important and we will see that these considerations will require higher agitations. The configuration yielding the double wave might then be the most interesting of them, since the higher strains are confined in a rather small region of the liquid phase.

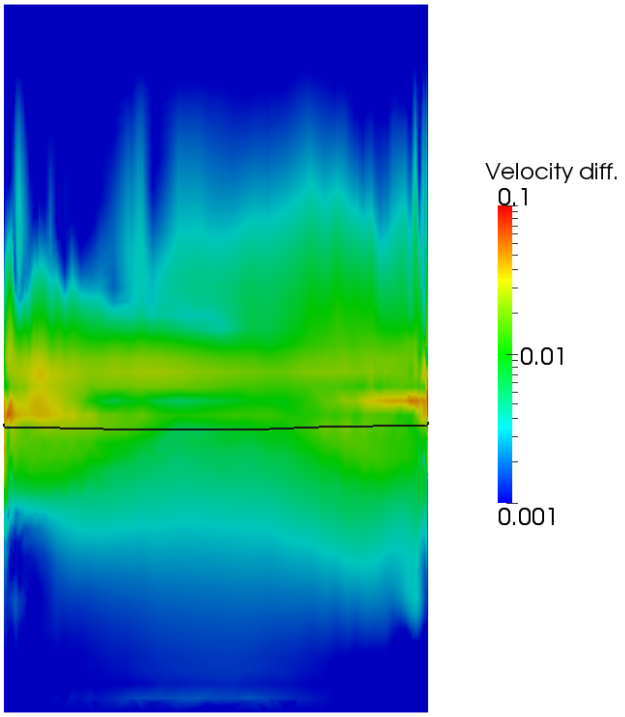

Figure 5.37: Magnitude of the difference of velocity for $l_s = 0.05$ and $l_s = 0.02$ after $t = 30$s. The black line represents the position of the interface Γ.

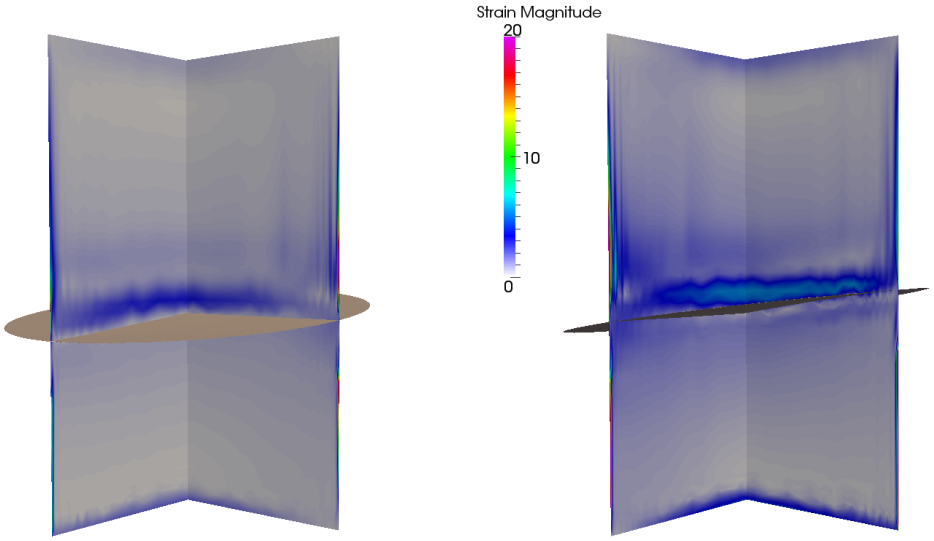

Figure 5.38: Strain magnitude at 50 and 60 RPM on two cuts and at $t = 15$s.

5.5 Mixing patterns

The shape and size of the free surface cannot explain alone the good gas transfer properties of the OSRs. The flow in the bulk of the liquid and the air phase also play an important role. For

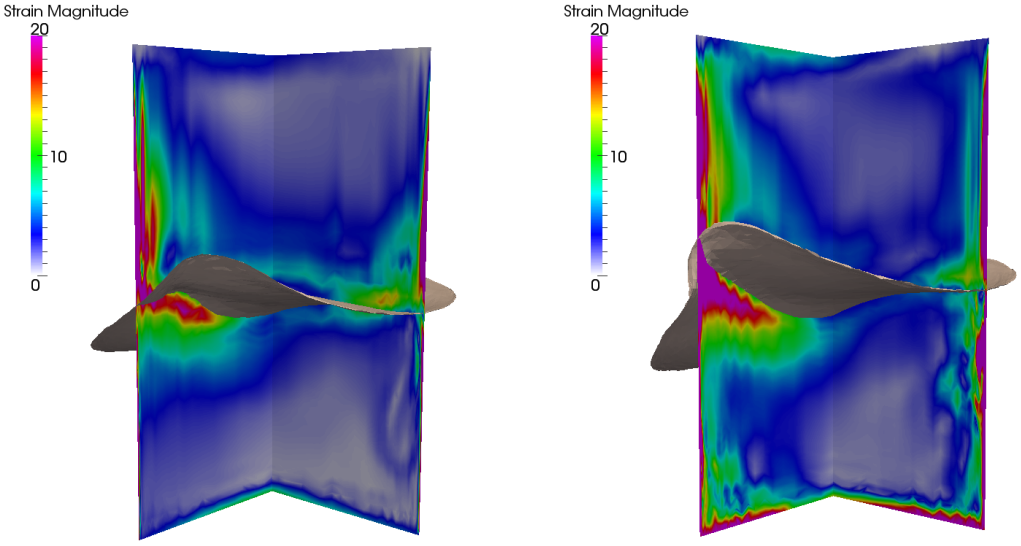

Figure 5.39: Strain magnitude at 70 and 80 RPM on two cuts and at $t = 15$s.

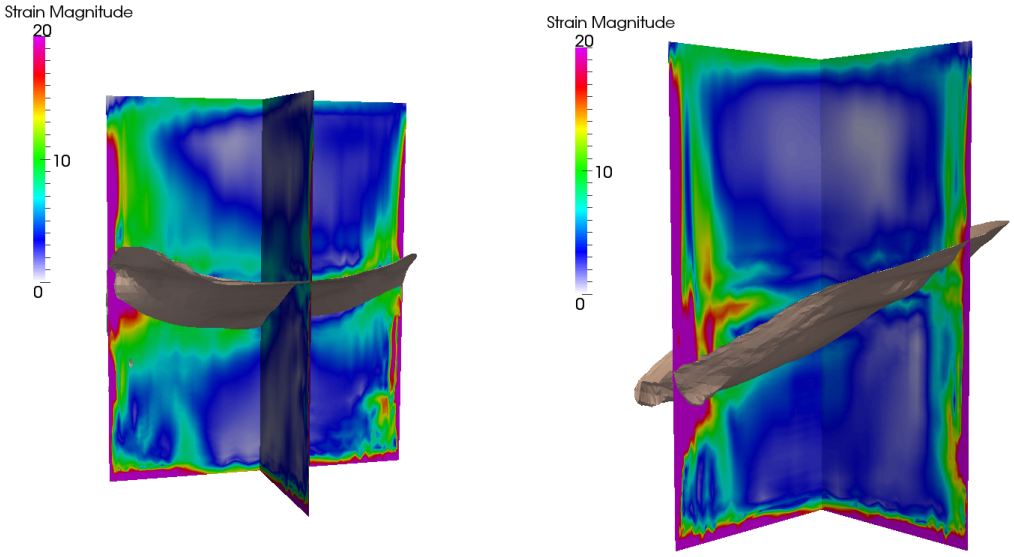

Figure 5.40: Strain magnitude at 90 and 100 RPM on two cuts and at $t = 15$s.

example, if we consider the transfer of oxygen from the air phase to the liquid phase, a flow that would keep the different vertical layers separated would lead to a very slow propagation of the oxygen in the liquid phase since only the diffusion acts. On the opposite, if the flow drives the layers initally far from the interface towards it, the mixing is enhanced. The mixing properties of the flow are also useful the keep the culture homogeneous, in both terms of

nutriments, wastes and cells.

To characterize the mixing pattern, we use massless particles, which are added in the flow after the whole simulation has been run. Different injection sites are set and particles are injected from these sites at regular time intervals. They are then transported passively with the fluid; thus, we vizualize the streaklines of the flow. The particles injected from a same site are represented with the same color which is useful to track the different trajectories. If the different colors are well spread, the mixing is good, otherwise, if particles stay clustered, the mixing is rather poor. This characterization is rather qualitative, but no quantitative measure of the whole mixing pattern is available for the moment.

We study the mixing patterns on the same cases that we used to generate the different waves. We will therefore have a good idea of the relationship between the wave shape and the mixing behaviour of the configurations.

Mixing from single to breaking wave

We start by investigating the mixing with the configurations studied in Sect. 5.2, i.e., with an agitation rate ranging from 50 to 100 RPM. Particles are injected from 9 different sites, defined by $x \in \{0, 0.05, 0.10\}$, $y = 0$ and $z \in \{0.02, 0.07, 0.12\}$ and they are transported by the fluid velocity during 1 second.

At 50 and 60 RPM, where we found a single wave, the particle trajectory is circular, irrespectively of the injection site (see Fig. 5.41). This indicates that the motion of the fluid particles is essentially local and the mixing quite poor.

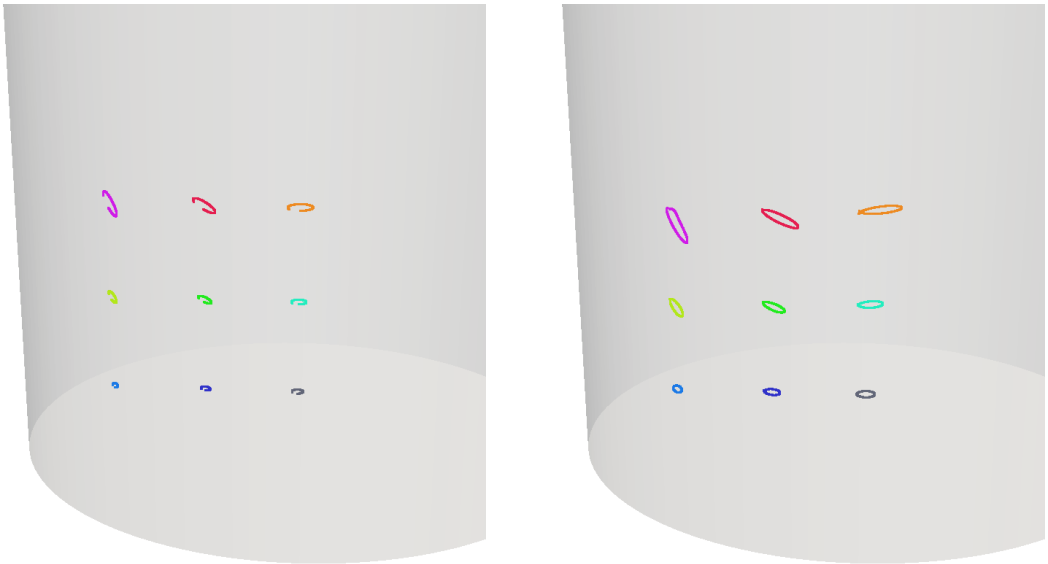

Figure 5.41: Trajectory of the particles for agitations rates of 50 (left) and 60 (right) RPM.

If we increase the agitation rate to 70 RPM, we observe that in the upper layer, the motion gains in amplitude, while the lower layers still keep their local motion, especially close to the center of the OSR. At 80 RPM, the particles near the boundary have also a large motion, but there remains a zone in the center which is only poorly mixed.

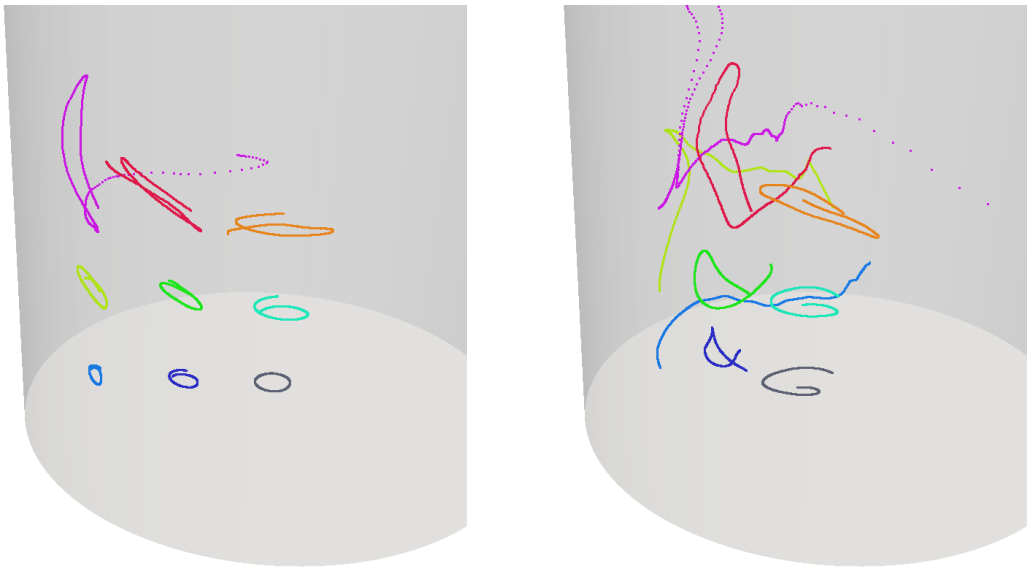

Figure 5.42: Trajectory of the particles for agitations rates of 70 (left) and 80 (right) RPM.

At 90 and 100 RPM, the particles have a large motion in the OSR. The mixing is therefore certainly very good at these regimes.

In the configuration proposed, it seems that the wave shape gives a clear hint on the mixing pattern: with a single wave, the particle motion is only local and the mixing is poor. When multiple waves are observed, the upper layer of the liquid undergoes a large motion while the rest of the liquid retains its local motion. Finally, when breaking waves are observed, the whole liquid phase enjoys a global motion, thus improving the mixing behaviour of the regime.

Mixing with the triple wave

We also investigated the mixing in the case of the triple wave. First of all, if we look at the velocity field in the OSR, we can identify three vortices between the peaks, as shown in Fig. 5.44. At first sight, this might be a sign of good mixing, since vortices might create interesting vertical motion at the local scale.

However, the analysis using the particles reveals that the periphery of the bioreactor enjoys a very good mixing, the center is not mixed at all since the particles describe circular paths: there is a phenomenon of solid rotation, i.e., particle have no relative velocity so that this part of the liquid phase behaves like a solid (see Fig. 5.45).

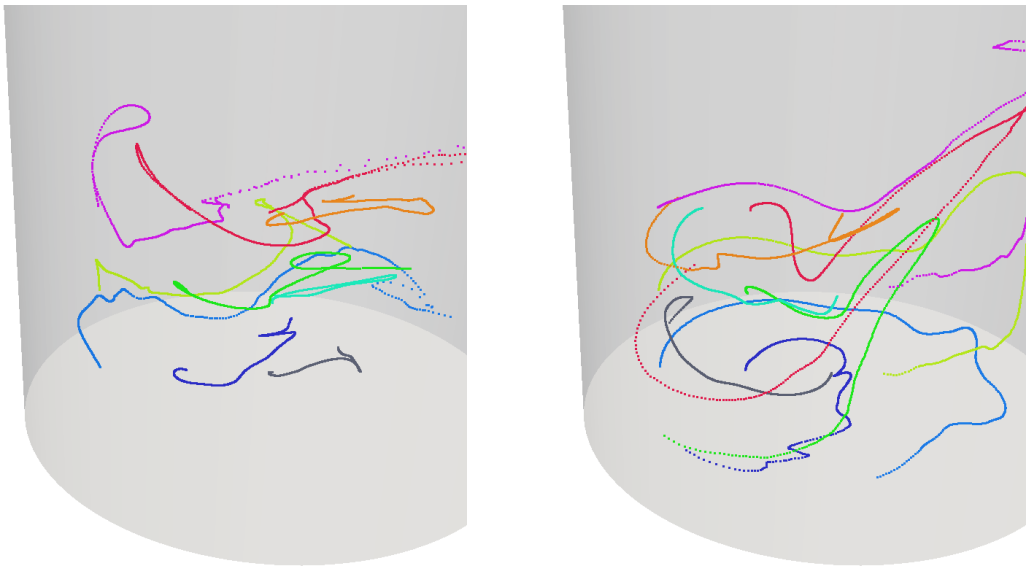

Figure 5.43: Trajectory of the particles for agitations rates of 90 (left) and 100 (right) RPM.

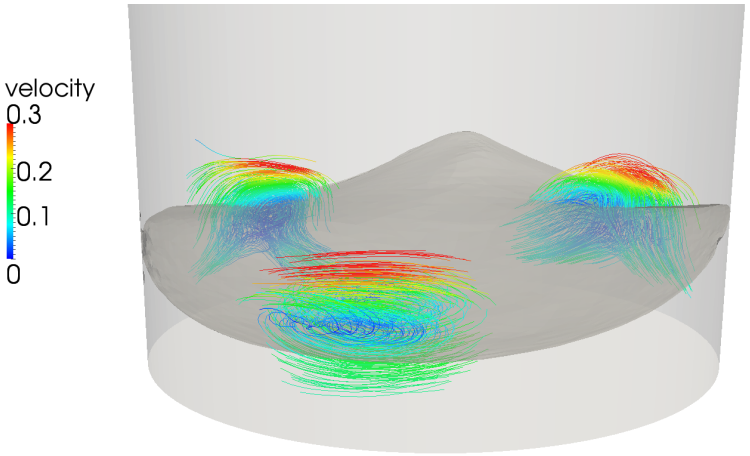

Figure 5.44: Streamlines showing the three small vortices.

5.6 Conclusion

This chapter allowed us to compare the numerical results obtained by our solver with experimental measures. Both the different wave shapes from Sect. 5.2 and the velocity of the LDV measures could be reproduced accurately. We could also observe that the strategy **C** for the boundary conditions was the most valuable option. Indeed, the hydrodynamic stress was better reproduced with them and the LDV measures could be used for the validation. Another good point for strategy **C** was that the different simulations that we carried out were not too sensible to the slip length l_s. A rather wide range of values for l_s could be used without

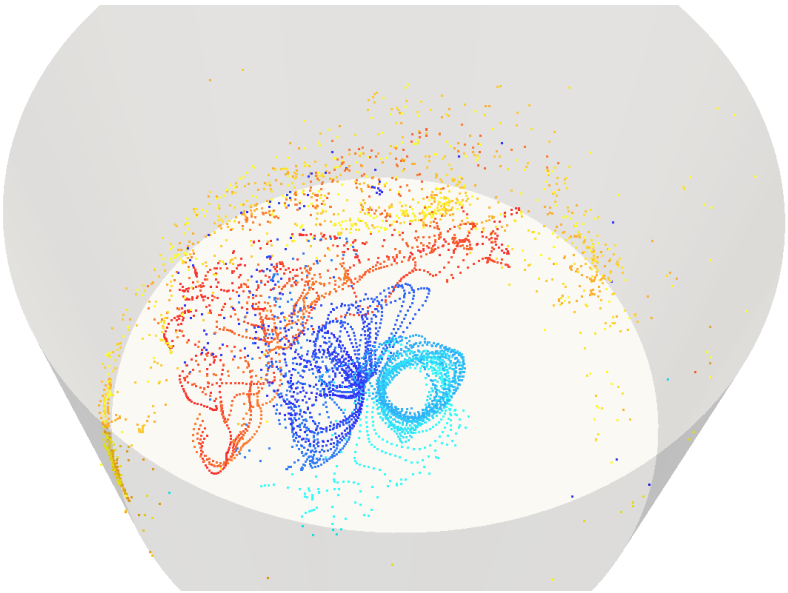

Figure 5.45: Trajectory of the particles for the triple wave configuration (colored by injection site, duration 5s).

affecting too significantly the results.

The different configurations tested were also useful to determine which regimes are the most suitable for the culture of mammalian cells. From the simulations that we carried out, we saw that breaking waves generate high hydrodynamic stress. Configurations leading to such a regime should be avoided. The triple wave gave rise to a poor mixing, probably due to its low ratio $\Pi_3 = \frac{H_0}{2R}$. Moreover, configurations with large radius R are more complicated to build and operate. Too high ratios $\Pi_3 = \frac{H_0}{2R}$ lead probably to poorly oxygenated zones at the bottom of the reactor, since the interface is too far from it. Configurations with a double wave might represent a good trade-off.

6 Cell growth modelling

6.1 Introduction

The aim of the previous chapters was to better understand how the different configurations of an OSR influence the hydrodynamics and create more or less favorable conditions for the cell cultures. The shear stress, the free surface shape and the mixing pattern are examples of quantities that can influence a culture. This section aims at presenting a further step towards the prediction of whole cell culture behaviour, i.e., the evolution of both the different nutriments and by-products and the cell population.

This task is difficult due to the complexity of the metabolism of mammalian cells. Indeed, many pathways have been identified and strong links between them ensure the optimal activity of cells, via enzyme activation or inhibition. Identifying which are the relevant reactions and molecules to determine the evolution of the culture is essential to yield an efficient model. The experimental data on which a model is built contain measures of the concentrations of some of the main species involved in the culture, but not all of them are measured. Usually, around 10 probings are performed during the whole culture, so no continuous data are available.

Several authors have tried to model and simulate cell cultures. Historically, the first models were dealing only with the concentration of cells in the culture and explicit expressions linking it with time were used. Among the models used, we find the logistic growth and the Gompertz law (see, e.g., [86] for different models and [13] for a comparison). Other experimental parameters, such as the temperature, were also incorporated in the model [110]. However, the numerous parameters (most of them without a precise physical meaning) to estimate was a big drawback.

These models have been nowadays replaced by systems of non-linear ordinary differential equations (ODEs), which have more flexibility but also yield better understanding of the processes underlying the growth. Indeed, in these models, one ODE is devised for each component of the medium. Thus, it is possible to understand which are the different active pathways and how to optimize a culture. Due to the complexity of the metabolism of the cells,

a very large number of models have been proposed, each based on different reactions (in [71], 34 reactions are considered).

The more complex the model, the larger the number of phenomena that it can potentially capture, but also the more parameter are introduced. All these parameters must be estimated to provide useful informations. The most successful approach for parameters estimation is probably the metabolic flux analysis (MFA) (see, e.g., [105]). This method considers a metabolic network of possible reactions and selects, using the measures available, the most likely used pathways. The reduced network is then used to define a system of ODEs. However, the model is valid only for one phase of the culture and must be recalibrated for each of them [77]. Moreover, the intracellular pools are supposed be constant in time (a strong hypothesis when considering entire cultures) [105] and the growth of the population is not provided.

To avoid expensive parameter estimations, we aim at creating a simple model representing the energy metabolism of CHO cells, in the spirit of [16]. We define only a few substances of interest and link them together through non-linear ODEs: given initial concentrations of the species of interest, as glucose $[\mathrm{Glc}]^{(0)}$, glutamine $[\mathrm{Gln}]^{(0)}$ or lactate $[\mathrm{Lac}]^{(0)}$, and the initial cell concentration $X^{(0)}$, we want to find the evolutions of the concentrations $X(t)$, $[\mathrm{Glc}](t)$, $[\mathrm{Gln}](t)$, $[\mathrm{Glc}](t)$ ... which satisfy the system of ODEs

$$\begin{cases} \frac{d[X]}{dt} & = F_X([X],[\mathrm{Glc}],[\mathrm{Gln}],[\mathrm{Lac}],\dots) \\ \frac{d[Glc]}{dt} & = F_{\mathrm{Glc}}([X],[\mathrm{Glc}],[\mathrm{Gln}],[\mathrm{Lac}],\dots) \\ \frac{d[Gln]}{dt} & = F_{\mathrm{Gln}}([X],[\mathrm{Glc}],[\mathrm{Gln}],[\mathrm{Lac}],\dots) \\ \frac{d[Lac]}{dt} & = F_{\mathrm{Lac}}([X],[\mathrm{Glc}],[\mathrm{Gln}],[\mathrm{Lac}],\dots) \\ \dots & \dots \end{cases} \tag{6.1}$$

for all $t \in (0,T)$, T being the period of culture.

First of all, we will describe the metabolism of CHO cells. We devise then our model by expliciting the list of all the species modeled and the actual expressions for the F_* functions. A numerical method is then employed to approximate the solution of the system of ODE (6.1). We finally present and comment the results obtained.

6.2 CHO cell metabolism

In this section, we present a summary of the metabolism of CHO cells. Even if this cell line has been extensively used in the biopharmaceutical industry, it is still nowadays not clear which are the active pathways in CHO cells and different scenarios are still conjectured [64, 76]. We present then the pathways which are generally present in mammalian cells and highlight some possible specificities of CHO cells. We refer to [9] for a complete exposition of the metabolism of mammalian cell.

Mammalian cells have an extremely complicated metabolism, involving many pathways with

different roles interacting together. In this work, we concentrate only on the energy related pathways, since they are supposed to represent the limiting factor in the growth of a culture. In this respect, one of the most important molecules is the adenosine triphosphate (ATP) which acts as "currency" for the energy. ATP works together with adenosine diphosphate (ADP) to transport energy within the cell: ATP is transformed into ADP and phosphate when energy is needed and, conversely, when energy is produced, it is stored by recycling ADP into ATP. ATP and ADP are present in very low concentrations in the cells and are used as temporary energy storage: it is estimated that, in the human body, an ATP molecule is transformed to ADP and back to ATP several hundred times a day [9]. The energy is however not only produced as ATP, but most of it as nicotinamide adenine dinucleotide (NAD or NADH). NAD is transformed into NADH when energy is released but it must be transformed into ATP to be used by the cell. This is the role of the electron transport chain, which uses oxygen to transfer energy from NADH to ATP and restore the NAD pool. This is the reason why oxygen does not appear in the pathway below, but it is necessary as soon as NADH is produced. Flavin adenine dinucleotide ($FADH_2$) has a similar role as NADH while guanosine triphosphate (GTP) can directly transfer its energy to ATP.

We distinguish three main pathways for the production of energy: the glycolysis, glutamine related metabolisms and the TCA cycle. We describe them in detail hereafter and Fig. 6.1 helps in visualizing the different reactions while Fig. 6.2 shows also all the different products of the reactions. We denote by a single arrow $\rightarrow$ irreversible reactions and by a double arrow $\leftrightarrow$ reversible reactions. We omit molecules, such as water, that do not effectively take part in the metabolism (and that have therefore no incidence on our model).

6.2.1 Glycolysis

Glycolysis is the pathway that starts the consumption of glucose to yield energy. It consists in a chain of reactions which transforms 1 mole of glucose into 2 moles of pyruvate, producing energy in the form of 2 moles of ATP and 2 moles of NADH:

$$1\,\mathrm{Glc} + 2\,\mathrm{ADP} + 2\,\mathrm{NAD} \rightarrow 2\,\mathrm{Pyr} + 2\,\mathrm{ATP} + 2\,\mathrm{NADH}\,. \tag{6.2}$$

As this reaction is not revertible in mammalian cells, it is strictly controlled in order not to waste glucose, a precious nutriment. Indeed, glucose can be used for other purposes such as the biosynthesis of serine and glycine, two amido-acides [9]. The glycolysis is inhibited by a high concentration of ATP, which indicates that enough energy is produced, by alanine, a by-product of pyruvate, and by citrate, which also indicates a sufficient energy charge [9]. Low pH also slows down the glycolysis.

On the side of glycolysis, we find the fermentation reaction. This revertible reaction transforms pyruvate into lactate to restore the pool of NAD:

$$1\,\mathrm{Pyr} + 1\mathrm{NADH} \leftrightarrow 1\,\mathrm{Lac} + 1\mathrm{NAD}\,. \tag{6.3}$$

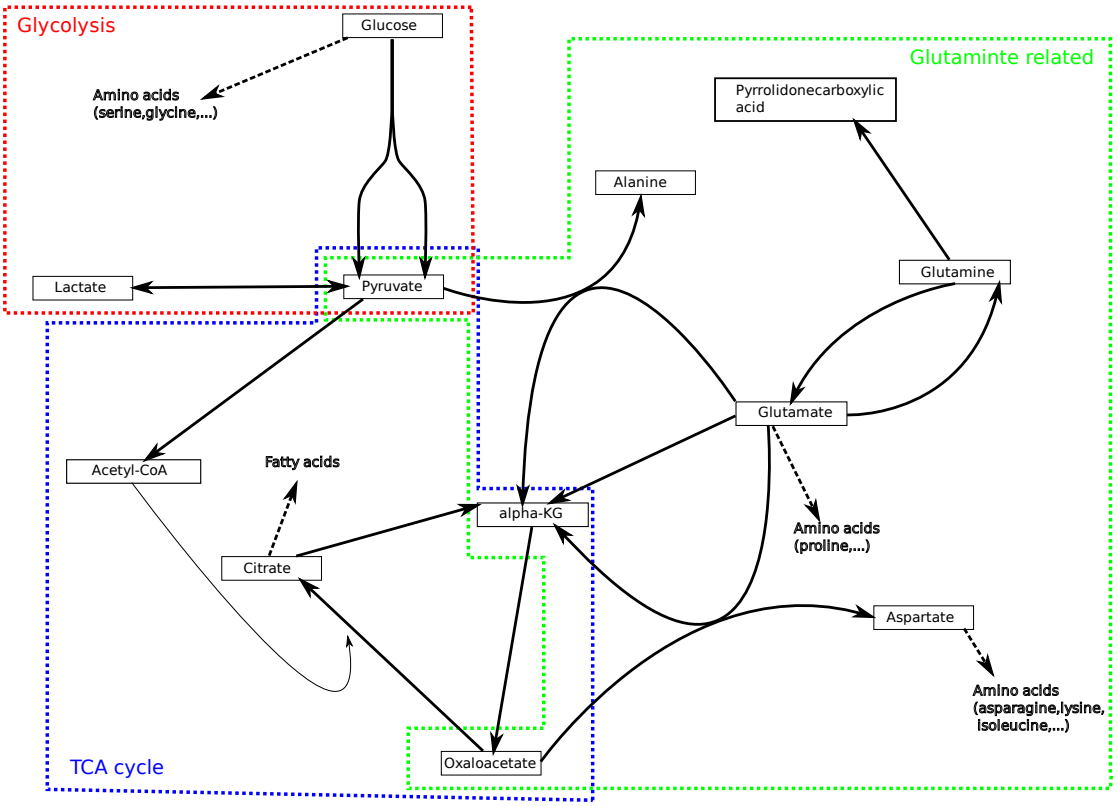

Figure 6.1: Partial metabolism of a CHO cell that we consider and the three different "energy" pathways.

Together with the glycolysis, this reaction yields the anaerobic respiration:

$$1\,\mathrm{Glc} + 2\mathrm{ADP} \rightarrow 2\,\mathrm{Lac} + 2\mathrm{ATP} \tag{6.4}$$

which is used by cells when oxygen is short or when high concentrations of glucose are available (an effect called *Crabtree effect*).

6.2.2 Glutamine related pathways

Glutamine is another important source of energy for mammalian cells, but it has many other roles: it is itself an amido acid and it is an important precursor of other amino acids, such as proline. Like glucose, glutamine is usually provided in the medium of the culture.

The first step in its catabolism is the decomposition of glutamine into glutamate, which releases ammonia

$$\mathrm{Gln} \rightarrow \mathrm{Glu} + NH_3\,. \tag{6.5}$$

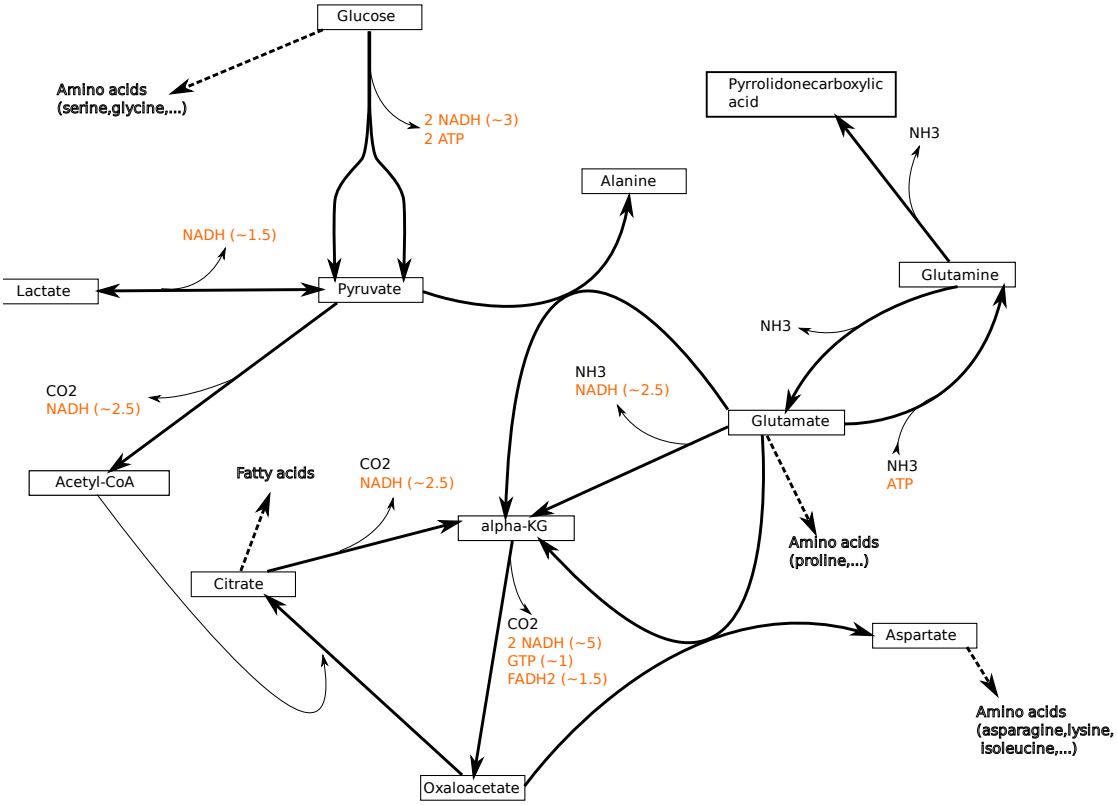

Figure 6.2: Metabolism reported in Fig. 6.1 with all the by-products explicited. The number between brackets indicates the equivalent in ATP of the different energy resources. Note that NADH yields a different quantity of ATP depending on the location of the reaction (inside or ouside the mitochondria).

Glutamate is then further decomposed to yield α-ketoglutarate. Three different reactions are used for that: a further ammonia molecule can be separated from glutamate, producing energy under the form of NADH

$$\text{Glu} + NAD \leftrightarrow \alpha\text{KG} + NH_3 + NADH\,. \tag{6.6}$$

Alternatively, the amino group can be transferred to pyruvate to yield alanine or to oxaloacetate to yield aspartate.

$$\begin{aligned} \text{Glu} + \text{Pyr} &\leftrightarrow \alpha\text{KG} + \text{Ala} \\ \text{Glu} + \text{Oxo} &\leftrightarrow \alpha\text{KG} + \text{Asp}\,. \end{aligned} \tag{6.7}$$

These three different reactions are inhibited by high ammonia concentrations and are reversible. On the contrary, the decomposition of glutamine into glutamate is not fully reversible: the synthesis of glutamine from glutamate requires the energy of an ATP molecule for each

molecule of glutamine produced.

We also consider another reaction occurring during the cultures. Glutamine is not a stable substance when it is free in water based media: it slowly decomposes into pyrrolidonecarboxylic acid and ammonia.

$$\mathrm{Gln} \rightarrow \mathrm{APy} + \mathrm{NH_3}\ . \tag{6.8}$$

6.2.3 TCA cycle

The TCA cycle (also called Krebs cycle or citric acid cycle) is the most important source of energy for mammalian cells. Starting from the α-ketoglutarate, 5 successive reactions lead to oxaloacetate, producing both carbon dioxide and energy, under different forms (NADH, GTP and $FADH_2$). Oxaloacetate is then associated to Acetyl-CoA, a substance derived from pyruvate, to yield citrate. Citrate is then retransformed into α-ketoglutarate by two reactions that produce another carbon dioxide molecule and energy (NADH). Summing the different contributions of the cycle, we end up with

$$\mathrm{Pyruvate} + 4\mathrm{NAD} + \mathrm{FAD} + \mathrm{GDP} \rightarrow 3\ \mathrm{CO_2} + 4\mathrm{NADH} + \mathrm{FADH_2} + \mathrm{GTP}\ . \tag{6.9}$$

This chain of reactions is inhibited by high energy charge in the cell, i.e., high concentrations of ATP and NADH. An accumulation of an intermediate product would also block the whole cycle.

Several exits to this cycle have been reported. As already stated in (6.7), oxaloacetate can be substracted from the cycle to produce aspartate, which is an important precursor for amino-acids such as asparagine. Citrate can also be extracted from the cycle to participate to the fabrication of fatty acids, for the fabrication of the cell membrane for example.

Due to these entries, the cycle must be constantly replenished in both acetyl-CoA, which serves as fuel for the cycle, and carbon chains, to compensate for the different exits. Acetyl-CoA is produced exclusively from pyruvate. Carbon chains can enter the TCA cycle via α-ketoglutarate, produced from glutamine. Another possibility for mammalian cells is to produce oxaloacetate from pyruvate, but it has been reported in [35, 56] that the enzyme catalizing this reaction, the pyruvate carboxylase, has a very activity, thus making the contribution of this pathway negligible for CHO cells.

Together with the glycolysis, the TCA cycle yields the aerobic respiration:

$$\mathrm{Glc} + 2\mathrm{ADP} + 10\mathrm{NAD} + 2\mathrm{FAD} + 2\mathrm{GDP} \rightarrow 6\mathrm{CO_2} + 2\mathrm{ATP} + 10\mathrm{NADH} + 2\mathrm{FADH_2} + 2\mathrm{GTP}\ . \tag{6.10}$$

Replacing the NADH, $FADH_2$ and GTP by their correspondant values in ATP, we find that from 1 mole of glucose, around 30 moles of ATP can be created, i.e., far more than with anaerobic respiration.

6.3 Cell growth models

In this section, we present the model that has been devised to simulate the cell cultures. This model focuses on the nutriments and their by-products rather than on gases, since measures are avalaible for them. We will however still model gases and pH, but in a second phase. The modeling comprises the three pathways described in Sect. 6.2.

From the experimental measures available, it appears that only these three pathways will not be sufficient to reproduce the whole culture. Indeed, when the cells enter the decline phase, there is apparently no reason why cells would not continue growing: nutriments are still present with sufficient concentrations and none of the by-products has reached a toxic concentration. pH is also not responsible for the death of the cells since its value remains suitable for the cells. We introduce then in our model an "anonymous" waste whose role is to signal the end of the growth. Potential identities for this waste are discussed in Sect. 6.5.2.

Table 6.1 references the quantities that we model and their significations.

Symbol	Signification
[Glc]	Glucose concentration
[Lac]	Lactate concentration
[Gln]	Glutamine concentration
[Glu]	Glutamate concentration
[Amm]	Ammonia concentration
[Was]	"Anonymous" waste concentration
$[O_2]$	Oxygen concentration
$[CO_2]$	Carbon dioxide concentration

Table 6.1: List of the different species

6.3.1 Pathways modeling

We start by the modeling of the three pathways presented in Sect. 6.2. For each of them, an activity factor $\alpha \in [0,1]$ is expressed as a function of the concentrations of the different substances considered. It represents the rate at which the pathway operates with respect to its maximal rate.

Glucolysis

Instead of modeling the glycolysis and the fermentation in separate reactions, we suppose that glycolysis is immediately followed by the fermentation, i.e., that glucose is always consumed by anaerobic respiration, therefore yielding lactate instead of pyruvate. This is justified by the fact that, experimentally, it has been observed that the reaction linking lactate and pyruvate is kept in equilibrium and pyruvate has a very low concentration [107]. So, as soon as pyruvate is being consumed, the concentration of lactate decreases, and vice versa.

The activity of the glycolysis pathway is not affected by the lactate concentration, unless it is out of the range of the cases that we consider here [59]. However, the lactate concentration still influences the glycolysis through its acidity, which lowers the pH and therefore slows down the process. Using the Michaelis-Menten kinetics, the activity of the glycolysis is modeled as:

$$\alpha_{\text{glycolysis}} = \left(\frac{[\text{Glu}]}{[\text{Glu}] + K^{\text{Glc}}_{\text{glycolysis}}} \right) \left(\frac{K^{\text{Lac}}_{\text{glycolysis}}}{[\text{Lac}] + K^{\text{Lac}}_{\text{glycolysis}}} \right) . \tag{6.11}$$

Glutamine related pathways

On the side of the glutamine, we model only its consumption and its spontaneous decomposition. The consumption of glutamine is regulated by the concentration of ammonia in the media:

$$\alpha_{\text{glutaminolysis}} = \left(\frac{[\text{Gln}]}{[\text{Gln}] + K^{\text{Gln}}_{\text{glutaminolysis}}} \right) \left(\frac{K^{\text{Amm}}_{\text{glutaminolysis}}}{[\text{Amm}] + K^{\text{Amm}}_{\text{glutaminolysis}}} \right) . \tag{6.12}$$

TCA cycle

Finally, the TCA cycle is modeled by the simple lactate consumption, which, in our model, is equivalent to a pyruvate consumption. We assume that the activity of the cycle is not affected by other substances than its substrate, in particular that the carbon chains are recycled:

$$\alpha_{\text{TCA}} = \left(\frac{[\text{Lac}]}{[\text{Lac}] + K^{\text{Lac}}_{\text{TCA}}} \right) . \tag{6.13}$$

6.3.2 Cell

The modeling of the cells is one of the key aspects of the model. The ODE for the cell concentration in our model reads:

$$\frac{dX}{dt} = \left(\gamma_{\text{glycolysis}} \alpha_{\text{glycolysis}} + \gamma_{\text{TCA}} \alpha_{\text{TCA}} + \gamma_{\text{glutaminolysis}} \alpha_{\text{glutaminolysis}} - \frac{[\text{Was}]}{[\text{Was}] + K^{\text{Was}}_{\text{cell}}} \right) X . \tag{6.14}$$

This ODE means that the growth of the cells is proportional to the activity of the different pathways and that too much waste can inhibit the growth or even reduce the cell population if it becomes more important than the other pathways.

6.3.3 Metabolites evolution

We describe now the evolution of the different substances present in the medium, one after each other.

Glucose

Glucose is mainly consumed to produce pyruvate, supposed to transform immediately to lactate. The ODE governing the glucose concentration [Glc] is

$$\frac{d[\text{Glc}]}{dt} = \left(-\mu_{\text{glycolysis}}\alpha_{\text{glycolysis}}\right) X$$

where $\mu_{\text{glycolysis}}$ represents the maximal yield of the glycolysis.

Lactate

Lactate is produced by the glycolysis and is consumed, through pyruvate, by the TCA cycles. We define then the following ODE for the concentration of lactate [Lac]

$$\frac{d[\text{Lac}]}{dt} = \left(2\mu_{\text{glycolysis}}\alpha_{\text{glycolysis}} - \mu_{\text{TCA}}\alpha_{\text{TCA}}\right) X \tag{6.15}$$

where μ_{TCA} is the maximal yield of the TCA cycle.

Glutamine

Glutamine is essentially consumed by the cells. Even if it can also be synthetized by them, we will not consider this process as important for the cultures at hand. As already remarked, glutamine can also decompose spontaneously. The ODE summarizing the possible fates of glutamine is

$$\frac{d[\text{Gln}]}{dt} = -\mu_{\text{glutaminolysis}}\alpha_{\text{glutaminolysis}} X - \lambda_{\text{Gln}}[\text{Gln}] \tag{6.16}$$

where [Gln] is the glutamine concentration, $\mu_{\text{glutaminolysis}}$ is the maximal yield of the glutaminolysis, $\alpha_{\text{glutaminolysis}}$ its activity and λ_{Gln} is the decay constant of the glutamine whose value, according to [30], is $\lambda_{\text{Gln}} = 4.1\ 10^{-3}$.

Ammonia

In our model, ammonia is produced by the consumption of glutamine to glutamate and by the spontaneous decomposition of glutamine. We neglect then the potential production of ammonia through glutamate consumption. However, this path has been observed to be inhibited by high ammonia concentration [59]. The ODE is then

$$\frac{d[\text{Amm}]}{dt} = \mu_{\text{glutaminolysis}}\alpha_{\text{glutaminolysis}} X + \lambda_{\text{Gln}}[\text{Gln}]\ . \tag{6.17}$$

Waste

For the toxic waste that we introduced in the model, we suppose that cells simply produce it constantly:

$$\frac{d[\text{Was}]}{dt} = \mu_{\text{waste}} X \tag{6.18}$$

6.3.4 pH, oxygen and carbon dioxide

Besides the nutriments and their by-products, we would like also to model the pH and the gases in the bioreactor.

pH

The pH is fully determined by the concentration of the substances in the bioreactor. However, determining the pH for a complicated solution whose complete composition is unknown is a complex task. We used here a more heuristic model which consists in tracking its variation with respect to the activity of the different pathways:

$$\frac{d10^{-pH}}{dt} = (\gamma^{\text{pH}}_{\text{glycolysis}} \alpha_{\text{glycolysis}} - \gamma^{\text{pH}}_{\text{TCA}} \alpha_{\text{TCA}} - \gamma^{\text{pH}}_{\text{glutaminolysis}} \alpha_{\text{glutaminolysis}}) X \,. \tag{6.19}$$

Oxygen and carbon dioxide

Oxygen and carbon dioxide are the most important gases for the cell cultures. However, the data for their concentrations were missing and they represented no limitation for the culture at hand. Modeling their interactions with the other pathways was therefore impossible. We still developed equations for their concentrations, even if they are decoupled from the other substances.

Oxygen is used in the electron transport chain to help the production of energy while carbon dioxide is directly rejected by the TCA cycle. We must also consider gas exchanges with the air phase of the bioreactor, which we consider having a constant composition. Exchanges between the two phases occur to requilibrate the concentrations on both sides. The ODEs that we consider then are:

$$\frac{d[\text{O}_2]}{dt} = -3\mu_{\text{TCA}} \alpha_{\text{TCA}} X + k_L a([\text{O}_2]^* - [\text{O}_2]) \tag{6.20}$$

$$\frac{d[\text{CO}_2]}{dt} = 3\mu_{\text{TCA}} \alpha_{\text{TCA}} X + k_L a([\text{CO}_2]^* - [\text{CO}_2]) \tag{6.21}$$

where $k_L a$ is a coefficient depending on the configuration of the bioreactor, in particular of the area of the free surface, $[O_2]^*$ and $[CO_2]^*$ are the concentrations of the gases in the air.

6.4 Numerical approximation

The model that we designed in Sect. 6.3 is a set of coupled non-linear ODEs. Our numerical method is based on the forward Euler method: the explicit nature of this scheme avoids the resolution of a non-linear equation. To obtain a stable scheme, we need to use small enough time steps, but this is not an issue since the dynamics of the culture is slow and every step of the algorithm is very cheap computationally.

More precisely, we define a time step Δt (the same symbol has already been used in chapters 3, but there is no confusion possible) and we define discrete times $t^n = n\Delta t$ at which we approximate the different quantities of interest. We assume that the initial concentrations are known:

- The initial concentrations of glucose $[\text{Glc}]^{(0)}$, of lactate $[\text{Lac}]^{(0)}$, of glutamine $[\text{Gln}]^{(0)}$ and cell $X^{(0)}$ are known from experimental data;
- $pH^{(0)}$ is also supposed to be known from the experimental data.
- The initial concentrations of ammonia $[\text{Amm}]^{(0)}$ and of waste $[\text{Was}]^{(0)}$ are supposed to be null, i.e., that the media is clean at the beginning of the culture;
- Oxygen $[O_2]^{(0)}$ and carbon dioxide $[CO_2]^{(0)}$ concentrations are assumed to coincide with the air concentrations.

At each time step t_n, the pathways activities are computed from the concentrations at the time step t_{n-1}:

$$\begin{cases} \alpha^{(n)}_{\text{glycolysis}} &= \left(\dfrac{[\text{Glu}]^{(n-1)}}{[\text{Glu}]^{(n-1)}+K^{\text{Glc}}_{\text{glycolysis}}}\right)\left(\dfrac{K^{\text{Lac}}_{\text{glycolysis}}}{[\text{Lac}]^{(n-1)}+K^{\text{Lac}}_{\text{glycolysis}}}\right) \\ \alpha^{(n)}_{\text{glutaminolysis}} &= \left(\dfrac{[\text{Gln}]^{(n-1)}}{[\text{Gln}]^{(n-1)}+K^{\text{Gln}}_{\text{glutaminolysis}}}\right)\left(\dfrac{K^{\text{Amm}}_{\text{glutaminolysis}}}{[\text{Amm}]^{(n-1)}+K^{\text{Lac}}_{\text{glutaminolysis}}}\right) \\ \alpha^{(n)}_{\text{glutaminolysis}} &= \left(\dfrac{[\text{Lac}]^{(n-1)}}{[\text{Lac}]^{(n-1)}+K^{\text{Lac}}_{\text{TCA}}}\right) \end{cases} . \tag{6.22}$$

The concentrations at the time step t_n are then computed:

$$\begin{cases} X^{(n)} & = X^{(n-1)} + \Delta t(\gamma_{\text{glycolysis}}\alpha^{(n)}_{\text{glycolysis}} + \gamma_{\text{TCA}}\alpha^{(n)}_{\text{TCA}} + \gamma_{\text{glutaminolysis}}\alpha^{(n)}_{\text{glutaminolysis}} - \frac{[\text{Was}]^{(n-1)}}{[\text{Was}]^{(n-1)}+K^{\text{Was}}_{\text{cell}}})X^{(n-1)} \\ [\text{Glc}]^{(n)} & = [\text{Glc}]^{(n-1)} - \Delta t\left(\mu_{\text{glycolysis}}\alpha^{(n)}_{\text{glycolysis}}\right)X^{(n-1)} \\ [\text{Lac}]^{(n)} & = [\text{Lac}]^{(n-1)} + \Delta t\left(2\mu_{\text{glycolysis}}\alpha^{(n)}_{\text{glycolysis}} - \mu_{\text{TCA}}\alpha^{(n)}_{\text{TCA}}\right)X^{(n-1)} \\ [\text{Gln}]^{(n)} & = [\text{Gln}]^{(n-1)} + \Delta t\left(-\mu_{\text{glutaminolysis}}\alpha^{(n)}_{\text{glutaminolysis}}X^{(n-1)} - \lambda_{\text{Gln}}Gln^{(n-1)}\right) \\ [\text{Amm}]^{(n)} & = [\text{Amm}]^{(n-1)} + \Delta t\left(\mu_{\text{glutaminolysis}}\alpha^{(n)}_{\text{glutaminolysis}}X^{(n-1)} + \lambda_{\text{Gln}}Gln^{(n-1)}\right) \\ [\text{Was}]^{(n)} & = [\text{Was}]^{(n-1)} + \Delta t\mu_{\text{waste}}X^{(n-1)} \\ [\text{O}_2]^{(n)} & = [\text{O}_2]^{(n-1)} - \Delta t\left(3\mu_{\text{TCA}}\alpha^{(n)}_{\text{TCA}}X^{(n-1)} + k_La([\text{O}_2]^* - [\text{O}_2]^{(n-1)})\right) \\ [\text{CO}_2]^{(n)} & = [\text{CO}_2]^{(n-1)} + \Delta t\left(3\mu_{\text{TCA}}\alpha^{(n)}_{\text{TCA}}X^{(n-1)} + k_La([\text{CO}_2]^* - [\text{CO}_2]^{(n-1)})\right) \\ pH^{(n)} = & \log_{10}\left(-10^{-pH^{(n-1)}} - \Delta t(\gamma^{\text{pH}}_{\text{glycolysis}}\alpha^{(n)}_{\text{glycolysis}} - \gamma^{\text{pH}}_{\text{TCA}}\alpha^{(n)}_{\text{TCA}} - \gamma^{\text{pH}}_{\text{glutaminolysis}}\alpha^{(n)}_{\text{glutaminolysis}})X^{(n-1)}\right) \end{cases} \tag{6.23}$$

The model that we designed comprises 16 parameters to determine, if we take the value of λ_{Gln} experimentally measured in [30]. To estimate their values, we started by approximating all the measured evolutions by splines. Then, we removed the spline approximation for the glucose and estimated the coefficients $\mu_{\text{glycolysis}}$, $K^{\text{Glc}}_{\text{glycolysis}}$ and $K^{\text{Lac}}_{\text{glycolysis}}$ which are needed for the glucose evolution using the spline approximation for all the other quantities. Once this was achieved, we went on with lactate, glutamine and so on (with the cell concentration being the last considered), removing the spline approximations one after each other.

6.5 Results and discussion

We use the numerical scheme devised in Sect. 6.4 to simulate data from a real experiment, conducted in an OSR by the Laboratory of Cellular Biotechnology of the EPFL. The experimental measures comprise glucose concentration, glutamine concentration, lactate concentration, glutamate concentration, cell density and pH measures. We discard the glutamate measure because it is known that the glutamate concentration in the medium do not coincide with the intracellular concentration [65]. The different measures have been performed after 1, 26, 50, 73, 98.5, 120.5, 145 and 166.5 hours. At the numerical level, the span of the culture, 156.5 hours, is split into 1000 time interval, so that $\Delta t = 0.1565$ hours (around 10 minutes).

6.5.1 Comparison with the experiments and values of the parameters

The estimation of the different parameters in our model leads us to the values given in table 6.2.

Some of the parameters were quite difficult to estimate. An example is $K^{\text{Glc}}_{\text{glycolysis}}$ which modifies the rate of the glycolysis in case glucose is short. However, in the data that we used here, glucose

Parameter	Value
$K^{\text{Glc}}_{\text{glycolysis}}$	4
$K^{\text{Lac}}_{\text{glycolysis}}$	1.5
$K^{\text{Gln}}_{\text{glutaminolysis}}$	1
$K^{\text{Amm}}_{\text{glutaminolysis}}$	3.2
$K^{\text{Lac}}_{\text{TCA}}$	1.25
$\gamma_{\text{glycolysis}}$	$2.2e-3$
$\gamma_{\text{glutaminolysis}}$	$5e-2$
γ_{TCA}	$3.2e-2$
$K^{\text{Was}}_{\text{cell}}$	$1.8e2$
$\mu_{\text{glycolysis}}$	$7e-8$
$\mu_{\text{glutaminolysis}}$	$5e-8$
μ_{TCA}	$2.2e-8$
μ_{waste}	$9e-9$
$\gamma^{\text{pH}}_{\text{glycolysis}}$	$6.2e-15$
$\gamma^{\text{pH}}_{\text{glutaminolysis}}$	0
$\gamma^{\text{pH}}_{\text{TCA}}$	$2.2e-15$

Table 6.2: List of the parameters and their approximated values.

had not reached a limiting rate. The parameters related to the waste were also determined up to a constant (which cannot be found since no reference is available).

From the values of the parameters, one can see that the glutamine related pathways is the most important pathway for the growth of the cells, since $\gamma_{\text{glutaminolysis}}$ is larger than γ_{TCA} and $\gamma_{\text{glycolysis}}$. The TCA cycle is also important, much more than the glycolysis, which relfects the fact that CHO cells are obligate aerobes. The activity of the different pathways is depicted on Fig. 6.3.

On Fig. 6.4, we can see that the evolution of the glucose concentration is globally well reproduced, even though it should be consumed slightly more quickly during the first days and slower in the last days. The same effect is then found on the lactate, whose trend is well approximated, excepted that it should be produced in slightly larger quantities in the first days.

The other nutriment, glutamine, is entirely consumed both in the experimental measures and in the numerical approximation. However, in the experiement, glutamine is produced near the end of the culture. This was not part of the model that we proposed and there is apparently no reason why cells would produce glutamine. Ammonia, produced by the catabolism and the decomposition of the glutamine accumulates during time, as could be expected. The same happens with the toxic waste, which heavily accumulates in the medium, thus slowing down the cell growth and provoking the decline of the cell population from 100h of culture. The evolution of the cell population is well captured by our model. The fact that the population should grow a bit faster during the exponential phase might come from the too low activity of

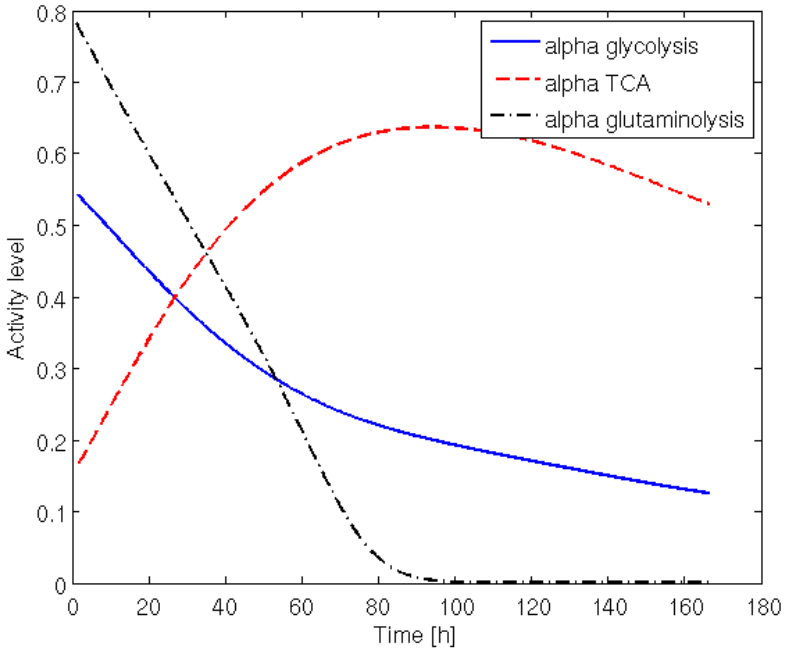

Figure 6.3: Level of activity of the different pathways modeled.

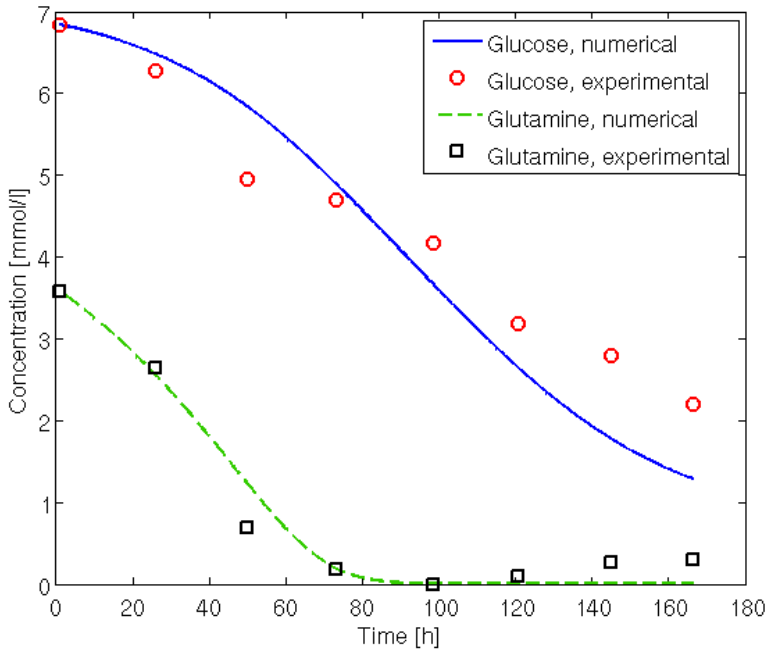

Figure 6.4: Evolution of the concentrations of glucose and glutamine during the culture.

the glucolysis at that time.

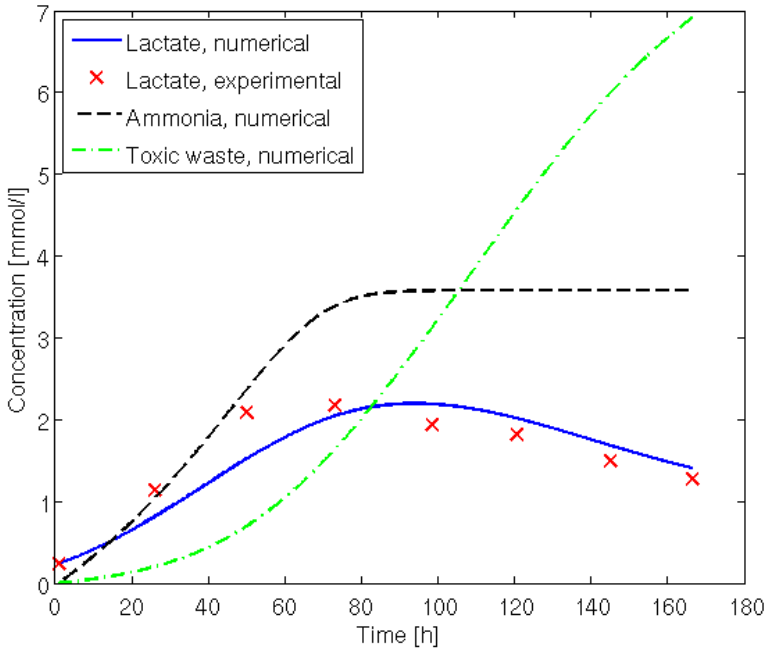

Figure 6.5: Evolution of the concentrations of glucose and glutamine during the culture.

Finally, the pH was computed and its trend was quite well captured even if our modelization of the pH was rather heuristic. Oxygen and carbon dioxide behave as expected: oxygen decreases when cell activity is at maximum and carbon dioxide increases in the same time.

6.5.2 Discussion

The proposed model reflects quite well the culture during the different phases. However, some phenomena cannot be explained. First of all, we observed that glutamine is produced in the end of the culture. This sounds rather illogical, since the production of glutamine costs energy. The most important problem with this model is the "anonymous" waste. Indeed, it would be extremely useful to find which is the substance limiting the growth.

- This cannot be lactate, since CHO cells can grow with larger concentrations [59].
- Ammonia must be added in larger quantities to inhib the growth [59].
- Pyrrolidoncarboxylic acid has no effect on the growth [30, 88].
- The pH observed in the end is about the same as in the initial phase of the growth.

Moreover, at the end of the culture, glucose is still at sufficient concentration [65] and glutamine is being produced. Several species have been proposed to play the role of the toxic

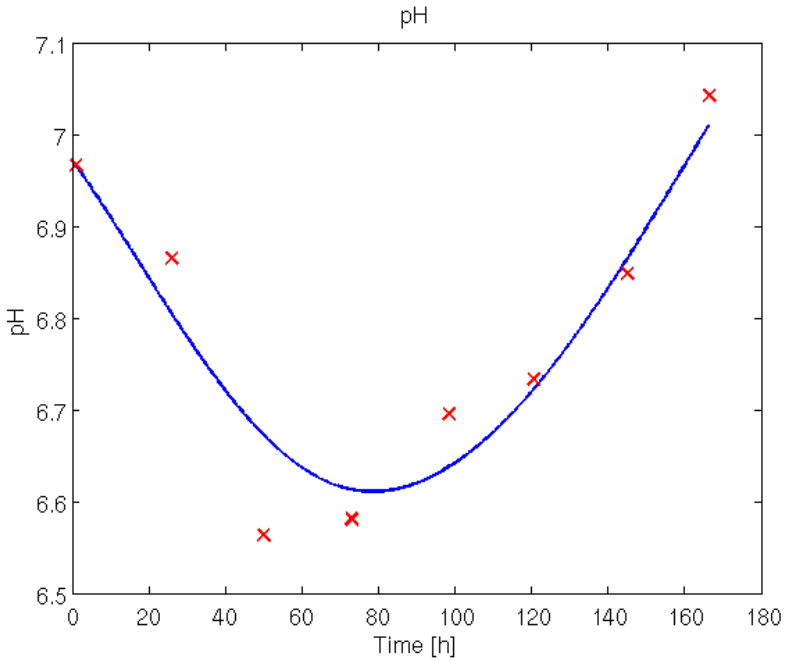

Figure 6.6: Evolution of the pH during the culture. Crosses represent the experiemental measures while the curve represents the result of the numerical approximation.

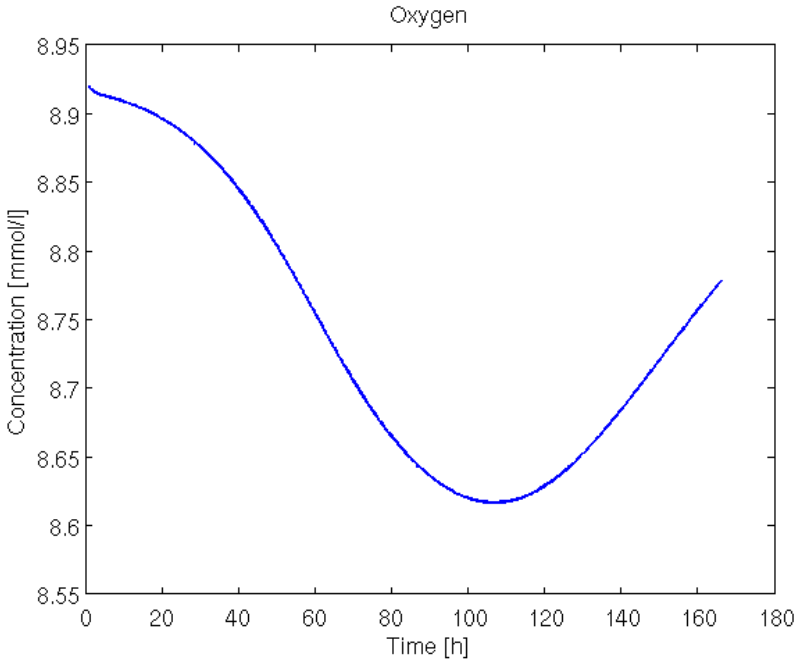

Figure 6.7: Evolution of the oxygen in the culture.

waste, e.g., acetoin [2], but none of these hypotheses has been confirmed experimentally. Re-

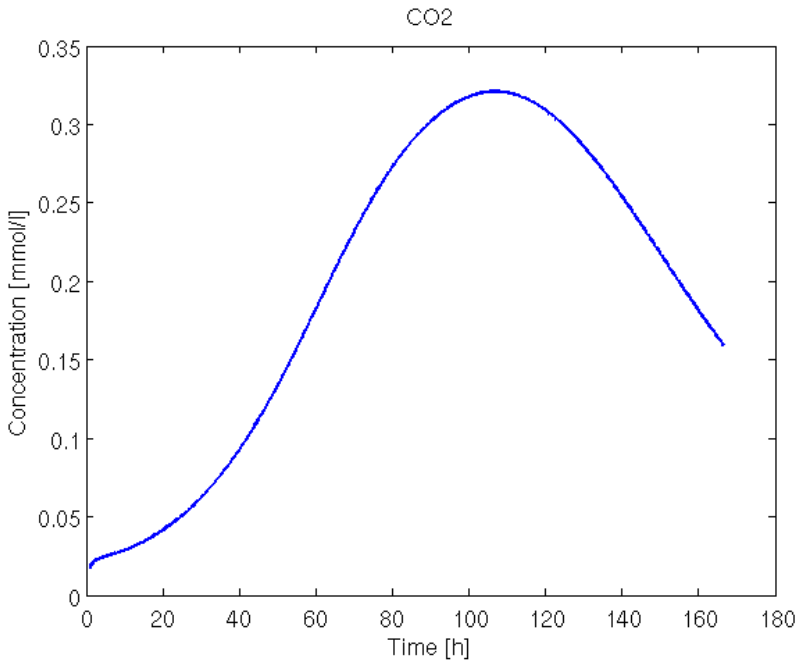

Figure 6.8: Evolution of the concentrations of carbon dioxide during the culture.

cently, it has been proposed that the TCA cycle might be actually truncated and two scenarios have been proposed.

In the first alternative scenario, the TCA cycle stops at the level of the oxaloacetate and, instead of producing citrate, takes the path towards aspartate [76], see Fig. 6.9. With this alternative, in the end of the culture, the cells are running out of energy because glutamate cannot be consumed any more to yield α-ketoglutarate due to the concentrations of ammonia [59] and therefore, the partial TCA cycle is empty. The production of glutamine could also be explained by the reaction of two glutamate molecules forming one molecule of α-ketoglutarate, to be used for energy production, and one molecule of glutamine. This reaction costs 1 ATP molecule, but with the newly created α-ketoglutarate molecule, more than 7 can be recovered. However, with this scenario, other phenomena cannot be explained. We observed from the data that lactate is consumed, which was attributed to the TCA cycle. With the TCA cycle truncated at the level of oxaloacetate, no acetyl-CoA is needed any more and lactate should not be consumed. Moreover, no evidence of large aspartate production was found experimentally. On the contrary, aspartate is usually consumed during the cultures [44].

A second scenario considers also the TCA cycle as truncated, but at the level of the citrate, see Fig. 6.10. The citrate might be used for other purposes, e.g., for the production of fatty acids, and quit the cycle. This alternative explains the end of the cutlure by the lack of carbon chains in the TCA cycle, but it still leads to the consumption of lactate, as observed. Moreover, recent studies of intracellular components [64] tend to confirm this scenario.

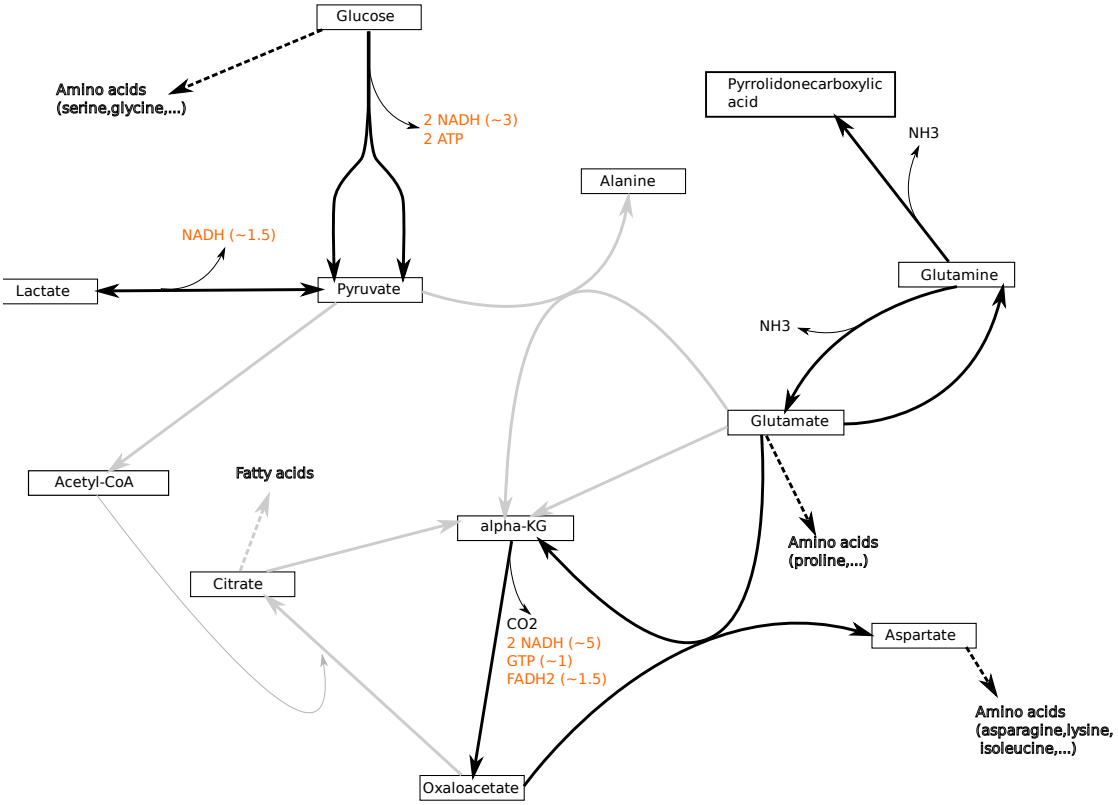

Figure 6.9: Metabolism proposed in the first scenario. Grey arrows indicate missing reactions.

These two scenarios are plausible, even if the second one is more likely. It would then be interesting to design models based on these two alternatives and compare the results with those obtained in Sect. 6.5.1. Experimentally, one could also try to verify which scenario is the most likely by making precise measures of oxygen and carbon dioxide concentrations:

- if the TCA is not truncated, 1 mol of oxygen is consumed per mol of carbon dioxide produced;
- if the TCA cycle is truncated at the level of the oxaloacetate (scenario 1), 1.5 mol of oxygen is consumed per mol of carbon dioxide produced;
- if the TCA cycle is truncated at the level of the citrate (scenario 2), 1.5 mol of oxygen is also consumed per mol of carbon dioxide produced (independently of the reaction used to transform glutamate into α-ketoglutarate). However, in this case, twice more carbon dioxide is produced (per glutamate molecule consumed) than with the first alternative scenario.

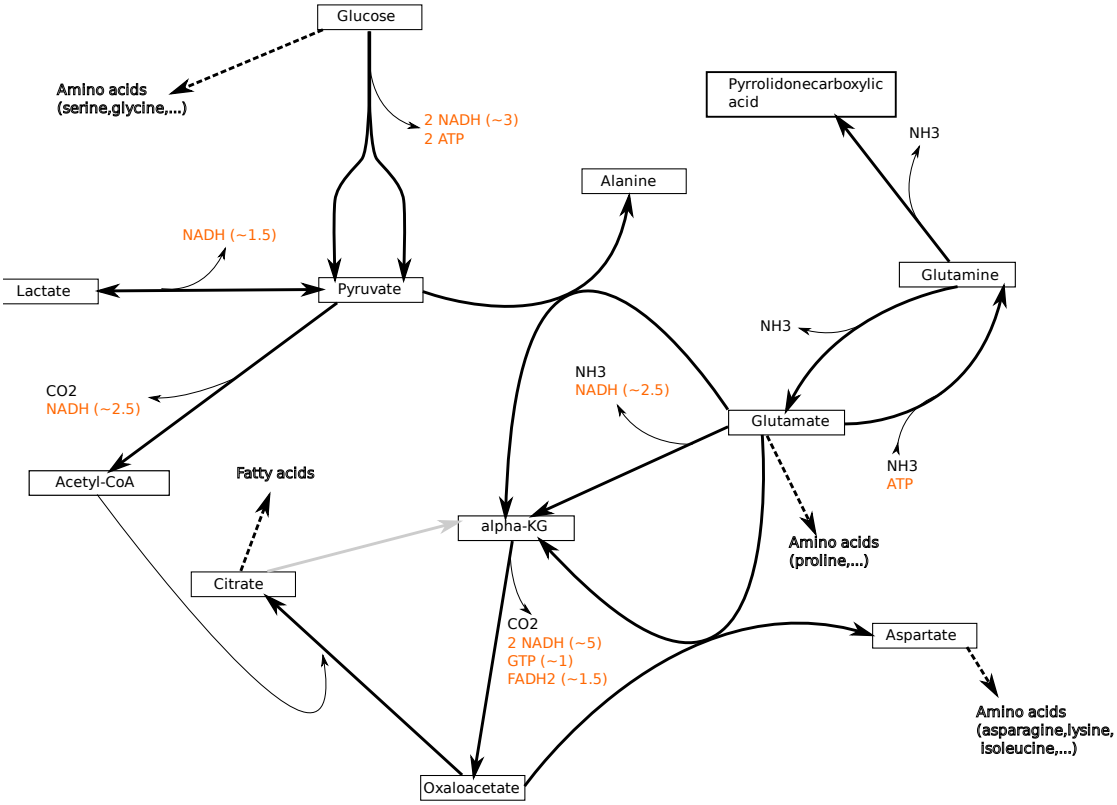

Figure 6.10: Metabolism proposed in the second scenario. Grey arrows indicate missing reactions.

6.5.3 Linking cell growth to the hydrodynamics

The final stage for a complete cell growth model would be to take into account for the specificities of the bioreactor used, OSRs in this work. A first step in that direction would be to compute representative values of 3D simulations of the hydrodynamics and incorporate them in our cell growth model. The $k_L a$ is a first example of such value and it is readily incorporated in our model.

Another factor that influences the cultures is the level of hydrodynamic stress created during the culture. Indeed, mammalian cells are more fragile than other cells [88] and high stresses might destroy their membrane. To incorporate this effect into our model, two questions must be addressed. First of all, one needs to determine which is the best value to represent a given OSR configuration. This could be, e.g., the maximal stress or the average stress. Then, this value must be incorporated in the model. The simplest way is probably to introduce an additional term in the ODE of the evolution of the cell population (6.14) which decreases the cell population as soon as the stress is too high.

With the second alternative scenario developed in 6.5.2, there would be another possibility:

fatty acids, derived from the TCA cycle, are important precursors of the membrane components. If we suppose that cells can repair their membrane to some extent, they would consume more fatty acids in case of high stress. We could therefore incorporate the stress effects in the consumption rate of fatty acids by cells. This would have the advantage to nicely reproduce the higher sensibility of cells to stress in the end of the culture observed in [95]. Indeed, when fatty acid has been consumed, cells would be running out of molecules to repair their membrane, thus increasing the death rate.

A further step towards the simulation of a complete culture would be to make 3D simulations of all the different components of the culture, i.e. nutriments, wastes, gases and cells. This would allow us to track potential inhomogeneities in the bioreactor: indeed, CHO cells are obligate aerobes, i.e., they cannot live without oxygen, and so cells that would be more often located close to the interface would benefit from better growth conditions. The PDEs governing the evolution of the different components would take into account, in addition to the couplings found with the ODE system, the diffusion of the components and their transport by the fluid, as done in [16]. For the cells, a sedimentation model, such as the one presented in [14], could be used as cells are slightly heavier than water (their density is estimated to 1100 kg/m^3). One would however have to deal with the different time scales of the culture and the hydrodynamics (which would act through the convective term).

7 Conclusion

In this thesis, several aspects of the cell culture in OSRs have been adressed, from the hydrodynamics and the related interface problems to the evolution of the cell culture.

Elliptic internal discontinuity interface problem

We started by studying the elliptic internal discontinuity interface problem in chapter 2. The SESIC method allowed us to approximate at low computational cost the solution to this problem. Indeed, the matrix part of the linear system obtained does not change with respect to the one coming from the same problem without interface. We demonstrated both numerically and analytically that optimal orders of convergence can be obtained, provided sufficient regularities on the interface data and the level set function used to describe the interface.

The replacement of singular integrals in the formulation associated with the SESIC method by regularized ones was also investigated. We found that higher regularities were required on the data and that the approximation slightly degraded close to the interface. However, this provided an additional level of simplicity to the method that would alleviate the need to rebuild the interface geometrically.

It would be interesting for future work to extend this methodology to the case where the diffusion coefficient is discontinuous across the interface, i.e., $\beta_1 \neq \beta_2$ in equation (2.1), without losing the point of view adopted in this work that made it simple and easily extensible to high dimensions.

Free surface flow in OSRs

The methodology developed for the SESIC method was reused to devise a free surface solver in chapter 3. In particular, the pressure was corrected to take into account for the gradient jump across the interface. This improved the accuracy so that we could treat the density and the viscosity in a sharp way, if adapted integration was used. The different test cases in chapter

4 highlighted the performances of our method with respect to regularized methods.

Different wall boundary conditions suitable for free surface flows were tested in replacement of the no-slip condition. It was found that imposing the normal condition in an essential way improved significantly the results over the imposition of both horizontal components, in the case of a cylindrical container. A correction term was then added to account for the mismatch between the normal to the faces and the normal imposed on the vertices. This avoided the presence of a spurious current that could deeply disturb the solutions of problems on long time intervals. Robin-type boundary conditions improved the phyiscal relevance with respect to the free-slip condition, by providing a better approximation of the boundary layer. We found that the solutions are not too sensitive to the size of the slip length which represents another good point for this type of boundary conditions.

Our solver was compared with real experiments in chapter 5.

- First of all, glycerine flows in OSRs were simulated and the amplitude of the wave was compared with experimental measures. Normal boundary conditions yielded results close to the experimental ones. Adding Robin-type boundary conditions improved the approximation of the stress on the boundary while keeping accurate wave amplitudes, when the discretization was fine enough.
- Water-air OSRs were then simulated with the aim of reproducing different wave patterns, including breaking and multiple waves. These different modes could be reproduced by our solver, in particular the triple wave, which required particular care to show up. The time step was found to be a critical issue to obtain the triple wave, indicating that most of the error was coming from that part of the method.
- We also compared our results with LDV measures of the velocity in the OSR. Even if the regime at which the measures were made was still transient, they generally agreed with the experimental measures. Only close to the bottom and the contact line, the tangential velocity was observed to be slightly shifted.

Thanks to the simulations, we could bring to light the hydrodynamics governing the different regimes of the OSRs. In particular, we characterized the mixing pattern and the strain associated to the different wave patterns.

We also showed that our method yields good parallel performances, especially when a large number of degrees of freedom was used per CPU. The bottleneck on the assembly side was identified as the application of the normal condition, which was not well balanced between the processes. The scalability of the resolution of the linear system was not perfect as well, but multi-level preconditioners should be able to improve it.

It would also be interesting to compare our results with those obtained with the XFEM, as this method is becoming increasingly popular and can provide the highly accurate solutions for free

surface flows. Different time discretizations, such as a Crank-Nicolson based scheme, could also be tested to improve the numerical accuracy. Better weights for the SUPG stabilization for the linear Navier-Stokes equations should also be used, to account for the different viscosities and densities in the two phases and for the time step used.

Cell culture

Finally, in chapter 6, we designed a simple model of non-linear ODEs to describe the evolution in time of the main nutrients and wastes in the cell cultures, as well as the concentration of cells. We calibrated our model with experimental data. It could describe the different phases of the culture quite accurately depending on the species. However, we could not explain all the phenomena observed and we had to introduce a generic toxic waste to explain the death phase. Two alternative scenarios for the culture were proposed and they would deserve both experimental and numerical investigations. Linking the ODEs with the hydrodynamics, from the points of view of the gas transfer, the mixing pattern and the hydrodynamic stress would help in identifying more precisely the optimal regimes for the cultures.

A A C++ DSEL for finite element assembly

A.1 Introduction

Many finite element codes, open source or proprietary, can be found which all have their own characteristics. We can however separate them in two main categories.

- Specialized libraries: The main goal of such a code is to provide a specific but highly optimized access to procedures for solving a particular problem. Examples are the structural codes FEAP (www.ce.berkeley.edu/projects/feap) and GetFEM++ (home.gna.org/getfem), the CFD codes FEATFLOW (www.featflow.de) and SU2 (su2.stanford.edu) and the two phase flow code DROPS (www.igpm.rwth-aachen.de/forschung/drops). Besides serial and parallel performances, the quality of such a code can be measured by the know-how contained in the code.
- General libraries: They aim at providing a general framework for solving PDEs using finite elements. Among the existing free softwares, we can cite DealII (www.dealii.org), Fenics (fenicsproject.org), Dune (www.dune-project.org), FreeFem++ (www.freefem.org/ff++) ... In this case, the most imporant quality, excepted the performances, is the user interface. This interface should be as clear and general as possible, to yield the smallest time spent for coding and debugging as possible.

The free open source library LifeV (www.lifev.org) used to be a specific code for fluid structure interaction and geometric multiscale with cardiovascular applications. It has been recently used for other applications, like free surface flows, porous media simulations,... It targets therefore a broader audience, but it user interface is not general and can be redefined. This appendix deals with the implementation of a domain specific embedded language (DSEL) for the assembly within the LifeV library, which would hide all the technicalities from the user and possibly generates efficient code.

Concerning the finite element assembly, some codes let the user free to write the whole assembly procedure by explicitly writing the loops, e.g. , DealII. This provides the highest

flexibility and probably the most optimized code. However, for complicated problems or formulations, this can yield long and error prone codes.

A better user interface is presented in libraries that allow the user to write directly the variational formulation to be assembled. This can save a lot of a coding time and also provide an easier code to debug if this is needed. The variational formulation is sometimes written in a different language than the language in which the calculations are performed. This is the case for example for FreeFem++ and, to a lower extend, of Fenics.

Finally, some libraries allow the user the write the variational formulation directly in the language in which the computations are performed, e.g. , Feel++ (www.feelpp.org), MFEM (code.google.com/p/mfem) or Sundance (www.math.ttu.edu/ klong/sundance/html). This seems to be the ideal solution since it yields a suitable user interface and makes also an eventual interface with another code possible.

This chapter aims at presenting a possible full C++ implementation of the assembly via the variational formulation. It will share some similarities with the implementation proposed in [78, 27] but with important differencies that make the implementation easier and the generalization to mixed finite element more straight forward.

A.2 The finite element assembly

In this first section, we provide a quick overview of the way the finite element method leads to an algebraic linear system from a given PDE. We will emphase the details that are the most relevant for the implementation exposed in the following sections. We refer to a general book [82] for a more complete introduction to this method.

A.2.1 Continuous and discrete formulations

To illustrate the assembly procedure, we study a Poisson problem: Find $u : \Omega \to \mathbb{R}$ such that

$$-k\Delta u = f \qquad \text{in } \Omega\,,$$

$$u = 0 \qquad \text{on } \partial\Omega\,,$$

where $\Omega \subset \mathbb{R}^N$ is an open domain (usually, $N \in \{1,2,3\}$), $f \in L^2(\Omega)$ and $k \in \mathbb{R}$ is a real constant. The variational formulation of this problem is given by: Find $u \in H_0^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$, it holds

$$\int_\Omega k\nabla u \cdot \nabla v = \int_\Omega f v\,.$$

A finite element space, where the approximation u_h of u is seeked, is defined based on a mesh τ_h made of different elements $\{K_i\}_{i<n_K}$:

$$V_h = \{v_h \in H_0^1(\Omega) \cap C^0(\Omega) \mid v_h|_{K_i} \in \mathbb{P}_k(K_i)\ \forall i < n_K\}\ ,$$

where k is the polynomial degree of the approximation. We denote $\{\phi_i\}$ a finite element basis of V_h. The discrete formulation reads then: find $u_h \in V_h$ such that for all $v_h \in V_h$,

$$\int_\Omega k\nabla u_h \cdot \nabla v_h = \int_\Omega f v_h\ .$$

This discrete problem is equivalent to a linear system defined by

$$A\mathbf{U} = \mathbf{F}$$

where A is a matrix such that $A_{ij} = \int_\Omega k\nabla\phi_j \cdot \nabla\phi_i$, U is a vector such that $u_h(\mathbf{x}) = \sum_i U_i\phi_i(\mathbf{x})$ and F is a vector such that $F_i = \int_\Omega f\phi_i$.

A first important remark here is that the algebraic system is defined from the discrete formulation, not from the continuous weak formulation or the strong formulation. Discrete formulations might also contain stabilization term which only have a meaning at that level. An interface aiming at representing a general formulation should therefore be based on the discrete formulation.

A.2.2 Decomposition in elements and numerical quadrature

A finite element basis $\{\phi_i\}$ is usually composed of functions whose support is restricted to a small number of elements of the mesh. The integration is then performed by looping over the elements of the mesh, computing the local contribution for each element and summing them into the matrix A:

$$A_{ij} = \sum_K \int_K k\nabla\phi_j \cdot \nabla\phi_i\ .$$

The loop over the element can also be split into several pieces to get an efficient parallelization scheme. The evaluation of the integral in each element is handled by quadrature rules at the numerical level. A quadrature rule is defined for each element by a set of nodes $\{\mathbf{x}_q^K\}_{1\le q\le n_q}$, $\mathbf{x}_q^K \in \mathbb{R}^N$ and the corresponding weights $\{w_q^K\}_{1\le q\le n_q}$. An element of the matrix A_{ij} is then evaluated as:

$$A_{ij} = \sum_K \sum_{q=1}^{n_q} k\ \nabla\phi_j(\mathbf{x}_q^K) \cdot \nabla\phi_i(\mathbf{x}_q^K)\ w_q^K\ . \tag{A.1}$$

Let us now reformulate the last expression in a slightly different way. We define 3 functions:

$$\begin{aligned} \text{dphi\_i}(i,j,q;K) &= \nabla\phi_i(\mathbf{x}_q^K) \quad \in \mathbb{R}^N \\ \text{dphi\_j}(i,j,q;K) &= \nabla\phi_j(\mathbf{x}_q^K) \quad \in \mathbb{R}^N \\ \text{cst}(i,j,q;K) &= k \quad \in \mathbb{R}\,. \end{aligned}$$

The relation (A.1) becomes, using these newly defined functions,

$$A_{ij} = \sum_K \sum_{q=1}^{n_q} \text{cst}(i,j,q;K)\ \text{dphi\_j}(i,j,q;K) \cdot \text{dphi\_i}(i,j,q;K)\ w_q^K\,.$$

This simple rewriting shows that the common property of the entities that are used in the assembly is the ability to return a value, for a given test function, trial function, quadrature node and element. This is not limited to the objects used here since any object used in a finite element assembly procedure has this property and can written in a same way, e.g. basis functions, finite element functions, XFEM functions, FE-HMM coefficients are comprised in this category. This is one of the key points for the flexibilty of the implementation that we will develop in the next sections.

We shall here comment finally on the computation of the values of the basis function, its derivatives and the position of the quadrature nodes. All the quantites are computed from a reference element and a reference quadrature and then mapped to the current element, as described in [34, 27]. At the implementation level, all these computations can be delegated to a class that we call `CurrentFE` and whose implementation is not investigated here.

A.3 General design

In this section, we will motivate and explain the workflow of the proposed implementation for the matrix assembly. This implementation easily extends to the assembly of the right hand side or to the integration of a quantity over the computational domain.

A.3.1 User interface

We start from the user interface to ensure that it is as clear as possible. To assemble the matrix associated to the Poisson problem, the sole code line to write is:

Listing A.1: User interface

```
integrate(
    elements(myMesh), // Where to integrate
    myQR,             // The quadrature rule
    testSpace,        // The test FE space
    trialSpace,       // The trial FE space
    value(3.0)* dot( dphi_i() , dphi_j() )
    )
```

```
>> myMatrix;                    // The assembled matrix
```

Let us discribe the different structures used in the interface, starting from the arguments of the `integrate` function. The first argument, `elements(myMesh)` indicates that the loop as to be performed on the elements of the mesh, and not on the boundary or on selected elements only. This elegant trick, proposed in [78], allows to keep the same interface for different integration procedures. For simplicity, we will assume that the loop is always done on all the elements of the mesh.

The structure `myQR` represents the quadrature rule to be used. It simply stores the location of the quadrature nodes and the associated weights in the reference element. The third and fourth arguments `testSpace` and `trialSpace` are structures storing the informations about the respective finite element spaces, e.g. the reference element and the degrees of freedom numbering.

The last argument represents the discrete formulation of the Poisson problem introduced previously.

Finally, the structure `myMatrix` represents the matrix to be assembled, which in our implementation is a wrapper around a matrix from the Epetra package of the Trilinos library [46].

The function `integrate` itself returns a structure, whose type is `IntegrateMatrixElement`, that contains the procedure to perform the assembly, using the informations given by the arguments. The assembly is initiated by the operator `>>` defined in the `IntegrateMatrixElement` structure (in similar way as the operator `=` usually triggers the computations in linear algebra codes using Expression Template techinques, [102, 101]). The assembly procedure consists in looping over the elements of the mesh and for each element

1. Update the internal structures with the data of the element on which the integration is currently performed.

2. Loop over the quadrature nodes, local test basis functions and local trial basis functions, get the value corresponding to the expression for these functions and this quadrature node and add it to the local matrix.

3. Add the local contributions to the global matrix.

The point is now how to get the value from the expression given as fifth argument of the `integrate` function for the local matrix and what is the meaning of the update of the internal structures.

A.3.2 `Expressions` and `Evaluations`

To represent an expression to be integrated, a tree is a suitable structure. In the proposed implementation, we will distinguish two trees:

- The `Expression` tree whose component represent the user input as given in argument of the `integrate` function.
- The `Evaluation` tree which contains structures that can be used within the assembly procedure, from which the values can be obtained.

Two reasons motivated the coexistence of these two trees:

1. First of all, it is in general a good practice to separate the user interface from the actual implementation. This can be used to add functionalities, for example, automatic differentiation [78].
2. A deeper reason is that the expression, as given in argument of the integrate function, does not suffice to perform the integration: some information coming from the finite element space are required at compile-time. For example, the dimension of the finite element space (i.e. , if the finite element space is scalar or vectorial) influences the expression by changing the type of the value returned. Here, for our example with the Poisson equation, the term $\nabla\phi_i$ is vector-valued , but it could be tensor-valued if the finite element space would be vectorial. The information about the dimension of the different finite element spaces is already present in the finite element space structure, so the user should not be asked to specify it again to avoid consistency checks.

The convertion between the two trees is done within the `IntegrateMatrixElement` structure (see Sect. A.4.3), which receives an `Expression` tree and stores its `Evaluation` equivalent.

Another consequence is that different `Evaluations` can return different type of values. This exactly reproduces what happen at the mathematical level: in the previous section, we stated that every expression able to give a value for a given combinaison of local basis functions and quadrature node could be integrated, but we never specified which kind of mathematical object is returned and it can be indeed different: $\nabla\phi_i$ returns a vectorial quantity whereas k is a scalar quantity.

This simple statement restricts also the possible implementations of the two trees: the composite design pattern based on dynamic polymorphism (as present in [38]) cannot be used, since it would require that all the `Evaluation` instantiations return the same type (as required by overloaded C++ methods). The best choice is then to rely on the static version of this pattern based on templates and usually refered to as Expression Template. This template metaprogramming technique [101] was first used in [102] to compute algebraic operations

without temporary objects. The main issue with template metaprogramming is the complexity of the code and its portability, although simple implementations might still be accepted by a large variety of compilers. Moreover, expression template trees might yield significantly better performances than their dynamic counterparts by avoiding calls to virtual functions (which are known to be slower) and by providing to the compiler more information, thus giving it a better chance to optimize the code.

A.4 Detailed mechanism

In this section, we shall explain more in detail the mechanisms introduced in the previous section. We describe the `Expression` tree, the `Evaluation` tree and the mechanism to translate one tree into another at compile time. Finally, we will look at the `IntegrateMatrixElement` class.

A.4.1 `Expression` tree

The `Expression` classes represent mathematical symbols, to which the `Evaluation` classes give then a sense. The `Expression` classes do not have a particular interface: they are simply classes containing the data necessary for the `Evaluation` class, e.g. the `ExpressionScalar` class only contains the value of the scalar constant it represents.

The `Expression` can be combined in a tree structure to represent the whole expression entered by the user, in the way it is usually done with the expression template technique. This means that the tree is build using the different template arguments of the classes that can represent nodes of the tree. For example, for the expression of the code A.1, we first of all make the dot product between the gradient of the basis functions, so the aggregrate expression for that part is of type `ExpressionDot<ExpressionDphiI,ExpressionDphiJ>`. The result of the dot product is then multiplied by a scalar constant, such that the type of the whole expression is

```
ExpressionProduct< ExpressionScalar , ExpressionDot<ExpressionDphiI,ExpressionDphiJ> >
```

It is represented graphically in Fig. A.1.

The end user should however not be asked to enter the whole tree structure, since this would be a very long and error-prone task, especially for complicated expressions. We used two tricks to allow a fast and easy definition of the `Expression`.

- To represent a leaf `Expression`, an helper function is defined to build the corresponding `Expression`. For example, an `ExpressionScalar` can be built using the following function:

Listing A.2: Helper function

```
ExpressionScalar value(const double& val)
{
   return ExpressionScalar(val);
}
```

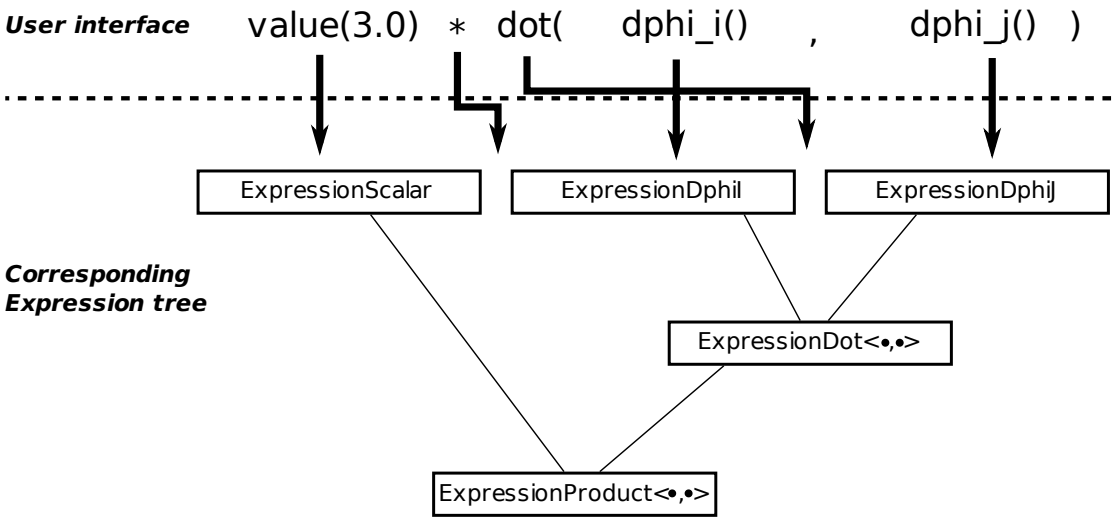

Figure A.1: `Expression` tree corresponding to the entry in A.1. A link between two classes means that the type above is a template argument of the type below.

This helps simplifying the interface, since the user does not need to know the type associated to a scalar constant.

- To combine the different `Expression` classes, we will use the possibility to overload the operators in C++ to define the `Expression` tree implicitly, i.e. without actually knowing the types or the tree structure. If we take as example the product, we overload the operator `*` to make it constructing the tree, in a similar way as what is done in expression template techniques:

Listing A.3: First attempt for implementing the product operation (see A.7)

```
template< typename LExpressionType, typename RExpressionType >
ExpressionProduct<LExpressionType,RExpressionType>
operator*(const LExpressionType& l, const RExpressionType& r)
{
    return ExpressionProduct<LExpressionType,RExpressionType>(l,r);
};
```

This last point can be unsatisfactory in the sense that the definition of the operator `*` has now been generalized to any two types. This might create undesired side effects and prevents two such frameworks to be emplemented in the same time.

To circumvent this problem, we use the CRTP (*Curiously recursive template pattern*) to mark the `Expression` classes, so that the overloaded operators work only for the marked types. First of all, we define a base class, `ExpressionBase`, which acts like a pure virtual class for the dynamic polymorphism:

Listing A.4: CRTP base class for the Expression classes

```
template <typename DerivedType>
class ExpressionBase
{
public:
    ExpressionBase(){}

    virtual ~ExpressionBase(){}

    const DerivedType& cast() const { return static_cast<const
      DerivedType&>(*this); }
};
```

All the `Expression` classes inherit then from this same templated base class. For example, the `ExpressionScalar` class is declared as

Listing A.5: The implementation of the class ExpressionScalar.

```
class ExpressionScalar : public ExpressionBase<ExpressionScalar>
{
public:
    typedef ExpressionBase<ExpressionScalar> base_Type;

    ExpressionScalar(const double& myValue)
         : base_Type(), M_value(myValue) {}

    ExpressionScalar(const ExpressionScalar& expr)
              : base_Type(), M_value(expr.M_value) {}

    const double& value() const { return M_value; }
private:
    double M_value;
};
```

and the `ExpressionProduct` as

Listing A.6: The implementation of the class ExpressionProduct.

```
template <typename LExpressionType, typename RExpressionType>
class ExpressionProduct : public ExpressionBase<ExpressionProduct<
  LExpressionType , RExpressionType> >
{
public:
    typedef ExpressionBase<ExpressionProduct<LExpressionType ,
      RExpressionType> > base_Type;

    ExpressionProduct(const LExpressionType& l, const RExpressionType& r)
        : base_Type(), M_l(l), M_r(r) {}

    ExpressionProduct(const ExpressionProduct<LExpressionType,
      RExpressionType>& expression)
```

```
        : base_Type(), M_l(expression.M_l), M_r(expression.M_r) {}

    const LExpressionType& left() const { return M_l; }

    const RExpressionType& right() const { return M_r; }
private:
    LExpressionType M_l;
    RExpressionType M_r;
};
```

The operator * can now be implemented so that it acts only on types inheriting from the `ExpressionBase` class:

Listing A.7: Final implementation of the product operation.

```
template< typename LExpressionType, typename RExpressionType >
ExpressionProduct<LExpressionType,RExpressionType>
operator*(const ExpressionBase<LExpressionType>& l, const ExpressionBase<
  RExpressionType>& r)
{
    return ExpressionProduct<LExpressionType,RExpressionType>(l.cast(),r.
      cast());
};
```

Remark that the type `ExpressionBase` does not appear in the `Expression` tree, since it is implicitly treated by the overloaded operators.

A.4.2 Evaluation tree

The `Evaluation` tree is the structure that provides the values for the integration within the `IntegrateMatrixElement`. It has a similar structure as the `Expression` tree, with the difference that the leafs of the tree contain also informations coming from the finite element spaces. A difference in the implementation is that the `Evaluation` classes do not derive from a common base (unlike the `Expression` classes). Indeed, no operator overloading is required here, since these classes are not present in the user interface.

All the `Evaluation` classes present the same interface to allow illimited combinaisons between them through static polymorphism. The minimal interface require for the implementation presented here is:

- `return_Type` is the type of the value returned by the `Evaluation`.
- A constructor taking in argument the corresponding `Expression` is used when the `Expression` tree is translated into the `Evaluation` tree.
- For the matrix assembly, a getter called `value_qij`, which returns for the basis function indices i and j and the quadrature index q the associated value.

- Finally, a setters for the test and trial `CurrentFE` structure is needed for the `Evaluation` corresponding to the values or derivatives of the basis functions.

This interface can be extended at will, depending on the specific needs of other `Evaluation` classes. We present now 3 examples corresponding to classes used for the Poisson problem (omitted methods are empty).

Scalar constant

The `EvaluationScalar` is the class corresponding to the scalar constants, like k in the Poisson problem. Each time a value is requested to this class, the value k is returned. The only data stored is the value of the constant. The main constructor of this class takes as argument an `ExpressionScalar`, the interface class containing the constant. Remark that the type returned by the this Evaluation class is always scalar.

Listing A.8: Implementation of the EvaluationScalar class

```
class EvaluationScalar
{
public:
    typedef double return_Type;

    explicit EvaluationScalar(const ExpressionScalar& expression)
     : M_value(expression.value()) {}

    return_Type value_qij(const UInt& /*q*/,
                          const UInt& /*i*/,
                          const UInt& /*j*/) const
    {
        return M_value;
    }

    // (...)
private:
    double M_value;
};
```

Gradient of the test function

The `EvaluationDphiI` aims at providing the value for the gradient of the test function. It needs therefore to access somehow the updated values of the gradient of the test functions, depending on the current element. There are two options for implementing it:

1. Store within this class the test finite element space structure and a CurrentFE. Then, for each element, this class updates the internal CurrentFE.

2. The other option is to store the CurrentFE in the `IntegrateMatrixElement` and only store a pointer to the values of interest.

We opted the second option: in terms of performance, this is advantageous since the values of the gradient of the basis functions are computed only once, even if this term is used several times in the same expression. Another issue with this `Evaluation` is the type of the value returned. If the test space is scalar, the gradient is a vector and if the test space is vectorial, the gradient is a tensor. Therefore, the `EvaluationDphiI` class is templated on the dimension of the test space to define different return types, either vector-values if the test space dimension is 1 or tensor values in the other cases. We show only the implementation of the scalar case, the vectorial case being similar.

Listing A.9: Implementation of the EvaluationDphiI class.

```
template<UInt testDim, UInt spaceDim>
class EvaluationDphiI
{
    // (...)
};

template <UInt spaceDim>
class EvaluationDphiI<1,spaceDim>
// Specialization for testDim=1
{
public:
    typedef Vector<spaceDim> return_Type;

    explicit EvaluationDphiI(const ExpressionDphiI& /*exp*/) {}

    template< typename CFEType > // CFE=Current Finite Element
    void setTestCFE(const CFEType* testCFE)
    {
        M_valuesPtr = &(testCFE->M_dphi);
    }

    return_Type value_qij(const UInt& q,
                          const UInt& i,
                              const UInt& /*j*/) const
    {
        return (*M_valuesPtr)[q][i];
    }

    // [ ... ]
private:
    std::vector< std::vector < Vector<spaceDim> > >
         const * M_valuesPtr;
};
```

Product operation

The two first `Evaluation` represent leafs in the `Evaluation` tree. We shall now describe the product operation, which represents a node in this tree. The `EvaluationProduct` class mainly fowards the constructors,the setters and the getters to the two `Evaluation` classes located above in the tree. The interesting point is to determine the type of the value returned. Indeed, this type depends on the types returned by the two classes above in the tree. There are two options: the first option is to let the compiler guess the type to be returned. This can be achieved by using the keyword *decltype* newly defined by the C++11 standard. However, most of the compilers do not support this feature yet. Therefore, an additional trick is needed to let the compiler know which type to return. This is achieved by delegating the definition of the returned type to another template class, called `ProductResult`.

Listing A.10: Implementation of the class ProductResult

```
template <typename L, typename R>
class ProductResult
{
  // Empty template class. We have to work
  // with its specializations
};

template <>
class ProductResult<double,double>
{
public:
    typedef double type;
};

template <UInt Size>
class ProductResult<double,Vector<Size> >
{
public:
    typedef Vector<Size> type;
};
```

The template class `ProductResult` defines only a type, depending on the template arguments corresponding to the type to be returned by the product operation. Now, the `EvaluationProduct` can simply query the type to return from the class `ProductResult`.

Listing A.11: Implementation of the class EvaluationProduct

```
template <typename EvaluationLType, typename EvaluationRType>
class EvaluationProduct
{
public:
    typedef typename EvaluationLType::return_Type Lreturn_Type;
    typedef typename EvaluationRType::return_Type Rreturn_Type;

    typedef typename ProductResult<Lreturn_Type,Rreturn_Type>::type
```

```
     return_Type;

    template <typename L, typename R>
    explicit EvaluationProduct
    (const ExpressionProduct<L,R>& expression)
        : M_evaluationL(expression.left()),
          M_evaluationR(expression.right()) {}

    return_Type value_qij(const UInt& q,
              const UInt& i,
              const UInt& j) const
    {
        return M_evaluationL.value_qij(q,i,j)
         * M_evaluationR.value_qij(q,i,j);
    }

    template< typename CFEType >
    void setTestCFE(const CFEType* testCFE)
    {
        M_evaluationL.setTestCFE(testCFE);
        M_evaluationR.setTestCFE(testCFE);
    }

    // [...]
private:
    EvaluationLType M_evaluationL;
    EvaluationRType M_evaluationR;
};
```

A.4.3 ExpressionToEvaluation class

The last class that we will comment in detail makes the link between the `Expression` and the `Evaluation` trees. This class, named `ExpressionToEvaluation`, consists in defining the type of the `Evaluation` corresponding to the `Expression` and the additional informations given as template arguments.

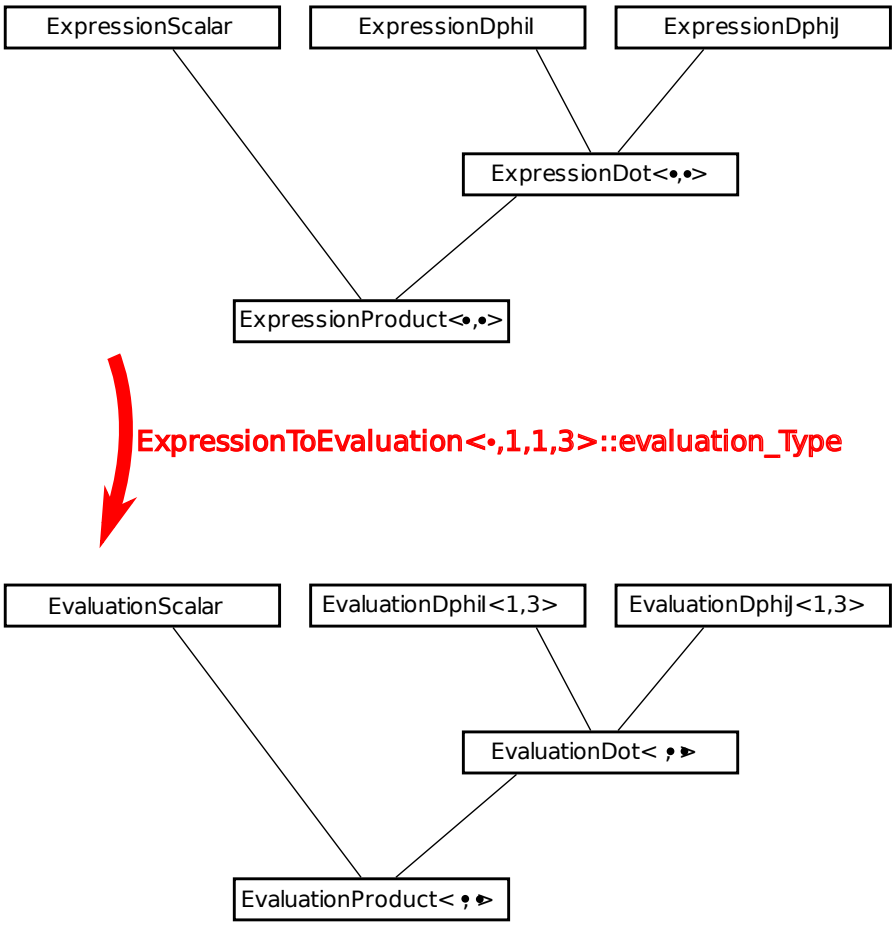

Figure A.2: Translation of the `Expression` tree into an `Evaluation` tree for the example considered.

The general definition of the templated class is empty:

Listing A.12: Generic implementation of the ExpressionToEvaluation class

```
template<typename Expression, UInt testDim, UInt solutionDim, UInt
  spaceDim>
class ExpressionToEvaluation
{};
```

The template arguments have the following meaning:

- **Expression** is the type of the Expression to be translated.
- **testDim** is the dimension of the test finite element space (1 for scalar quantites, more for vectorial quantities).
- **solutionDim** is the dimension of the trial finite element space.

- **spaceDim** is the dimension of the domain (3 for a 3D problem).

The actual implementation is based on a partial specialization with respect to the Expression type, available for each Expression, which defines the type `evaluation_Type`. For example, considering the scalar constant, we have:

Listing A.13: Specialization of the ExpressionToEvaluation class for scalar constants.

```
template<UInt testDim, UInt solutionDim, UInt spaceDim>
class ExpressionToEvaluation<ExpressionScalar,testDim,solutionDim,
  spaceDim>
{
public:
    typedef EvaluationScalar evaluation_Type;
};
```

as, independantly of the dimensions, the Evaluation corresponding to the ExpressionScalar is EvaluationScalar. For the gradient of the test function, we have

Listing A.14: Specialization of the ExpressionToEvaluation class for gradient of the basis functions.

```
template<UInt testDim, UInt solutionDim, UInt spaceDim>
class ExpressionToEvaluation<ExpressionDphiI,testDim,solutionDim,spaceDim
  >
{
public:
    typedef EvaluationDphiI<testDim,spaceDim> evaluation_Type;
};
```

since the type of the Evaluation depends on the dimensions. For the operations, the same holds, even though the implementation is a little bit more technical:

Listing A.15: Specialization of the ExpressionToEvaluation class for product operation.

```
template<typename ExpressionL, typename ExpressionR, UInt testDim, UInt
  solutionDim, UInt spaceDim>
class ExpressionToEvaluation<ExpressionProduct<ExpressionL,ExpressionR>,
  testDim,solutionDim,spaceDim>
{
public:
    typedef EvaluationProduct<
        typename ExpressionToEvaluation<ExpressionL,testDim,solutionDim,
          spaceDim>::evaluation_Type
        ,typename ExpressionToEvaluation<ExpressionR,testDim,solutionDim,
          spaceDim>::evaluation_Type
    > evaluation_Type;
};
```

Remark that this last definition makes a recursive use of the `ExpressionToEvaluation` class, making the `Evaluation` tree grow.

A.4.4 Integration

Now that the `Expression` and `Evaluation` classes are defined, we can describe and implement the integration loop. The role of the class `IntegrateMatrixElement` is to provide the general algorithm for the assembly on a matrix, by looping over elements. It stores as member the Evaluation tree (among others), even if it receives the `Expression` tree as input for the constructor: the `Expression` tree is translated into the `Evaluation` tree in the constructor of this class.

Once the `IntegrateMatrixElement` is constructed, it is used to perform the assembly, which is contained in the `operator>>`. The assembly consists in a loop over the elements of the mesh, and for each element:

1. Updates the evaluation tree with the data of the current element.
2. Fills the local matrix by summing, for each test function and trial function, the contributions corresponding to the quadrature nodes.
3. Gets the global numbering of the local basis functions and sum the values of the local matrix into the global matrix.

Listing A.16: Implementation of the class IntegrateMatrixElement

```
template < typename MeshType, typename TestSpaceType, typename
  SolutionSpaceType, typename ExpressionType>
class IntegrateMatrixElement
{
public:
     typedef typename ExpressionToEvaluation
        < ExpressionType, TestSpaceType::S_fieldDim,
        SolutionSpaceType::S_fieldDim, 3>::evaluation_Type
        evaluation_Type;

     IntegrateMatrixElement
        (const boost::shared_ptr<MeshType>& mesh,
        const QuadratureRule& quadrature,
        const boost::shared_ptr<TestSpaceType>& testSpace,
        const boost::shared_ptr<SolutionSpaceType>& solutionSpace,
        const ExpressionType& expression)
    : M_evaluation(expression),
    // (...)
    {};

    template <typename MatrixType>
    inline void operator>>(MatrixType& mat)
    {
    // Perform the assembly by calling
    // M_evalution.value_qij(...)
```

```
    // (...)
    }

    // (...)

private:
    evaluation_Type M_evaluation;

    // (...)
};
```

A.5 Extension to mixed problems

To solve problems with multiple unknown quantities, e.g. Stokes, Navier-Stokes and many multiphysics problem, in a monolithic way, one needs to assemble the monolithic system, which consists in several blocks. Our implementation considers blocks as the paradigm for the assembly.

The first step consists in defining block matrices, which derive from the original matrix, but contain also the sizes and coordinates of the different blocks. Then, the blocks are defined as views on the monolithic matrix. The only roles of the blocks is to shift automatically the contributions to the right place when added from the local matrix.

Thanks to these definitions, the assembly of the matrix corresponding to the Stokes problem can be performed using the following code

Listing A.17: Example of user interface for the assembly for the Stokes problem

```
integrate( elements(myMesh), myQR, uSpace, uSpace,
    dot( dphi_i() , dphi_j() ) )
>> myMatrix.block(0,0);
integrate( elements(myMesh), myQR, uSpace, pSpace,
    div_i() * phi_j() )
>> myMatrix.block(0,1);
integrate( elements(myMesh), myQR, pSpace, uSpace,
    div_j() * phi_i() )
>> myMatrix.block(1,0);
```

The advantage of this approach is that no consistency checks is needed and it extendeds to an arbitrary number of unknowns while keeping a very simple implementation. This implementation might however be slower than the one proposed in [78], due to the update of the CurrentFE structures which has to be done for every block needing it. On an other side, one might implement a copy method based on the blocks which might accelerate the process in some cases (see Sect. A.6).

A.6 Comparison and performances

In this section, we expose some examples showing the advantages and disadvantages of the assembly via expressions with respect to the usual "loop by hand" approach.

Before coming to performance issues, we shall comment on the other differences between the two approaches. On one hand, the portability of the "loop by hand" approach is greater, since it does not require the compiler to deal with metaprogramming. However, we did not encounter problems while compiling the Expression Template Assembly with recent versions of both GCC and Intel compilers. On the other hand, the Expression Template Assembly makes the code cleaner and easier to understand while the "loop by hand" approach is very time consuming and error-prone.

A.6.1 Assembly of an advection-diffusion-reaction matrix

Our first test consists in assembling the different parts of the matrix associated with an advection-diffusion-reaction problem (ADR):

$$A_{ij} = \int_\Omega \underbrace{\nabla\phi_i \cdot \nabla\phi_j}_{\text{Diffusion}} + \underbrace{\beta \cdot \nabla\phi_j \, \phi_i}_{\text{Advection}} + \underbrace{2\phi_i\phi_j}_{\text{Reaction}} \tag{A.2}$$

where β is a velocity field that as been interpolated on the mesh τ_h. The `Expression` and the `Evalutation` classes corresponding to the integration of an interpolated field have not been shown before, but follow the same ideas and implementations. The "loop by hand" approach is implemented as a class, that can assemble each of the diffusion, advection or reaction contributions in a given matrix, but not all of them in one shot.

The different tests consists in adding selectively only some of the terms to be assembled. The results are given in the next table.

Problem	"loop by hand"	Expression Template
Diffusion	0.58s	0.62s
Advection	1.01s	0.84s
AD	1.78s	1.01s
ADR	2.53s	1.07s

From the assembly of the single terms A and D, we can observe that the Expression Template assembly yields comparable performances as the "loop by hand" approach for simple expressions. For more than one term, the "loop by hand" strategy adds sequentially the different parts and the total timings are therefore close to the sum of the timings for the assembly of each single term. This is not the case for the Expression Template assembly, which assembles

all the term in one shot, thus saving time with the update of the `CurrentFE` and with the addition of the local contribution to the global matrix, which are both done only once for the whole expression (instead of being performed two or three times). The difference is specially large for the assembly of the whole ADR, where the Expression Template assembly is more than twice faster.

A.6.2 Assembly of a Stokes matrix

The second test that we designed to assess the performance of the assembly procedures described before consists in a Stokes problem. This problem is *a priori* not favorable for the Expression Template Assembly, since only single terms are present: the matrix M to be assembled has the following block structure:

$$M = \left(\begin{array}{c|c} A & B \\ \hline B^T & 0 \end{array} \right). \tag{A.3}$$

The different blocks are defined by

$$A_{ij} = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \tag{A.4}$$

$$B_{ij} = \int_\Omega (\nabla \cdot \phi_i) \psi_j \tag{A.5}$$

where $\{\phi_i\}$ and $\{\psi_i\}$ represent respectively the basis of the velocity space and the pressure space. In this example, we use the $\mathbb{P}_2 - \mathbb{P}_1$ Taylor-Hood elements which fulfill the inf-sup compatibility condition [82]. We considered again a structured cubic mesh, which yields around $400'000$ degrees of freedom for this problem.

"Loop by hand"

The "loop by hand" strategy assembles the whole system in 3.21s: it does not exploit the fact that B and its transpose are needed, but assembles both of them separately. However, the block diagonal structure of the matrix A is used, only one diagonal block is assembled and copied on the other diagonal blocks.

"Naive" Expression Template implementation

The straight forward approach with Expression Templates is to define the velocity as a vectorial quantity and assembles the three blocks as defined by (A.4) and (A.5). The time to assemble the whole matrix is then 6.74s, which exeeds by far the time needed by the "loop by hand strategy". The main reason for this overhead is the inner structure of the block A, which is actually block diagonal. This structure is exploited by the "loop by hand strategy", but cannot be detected within the Expression Template assembly.

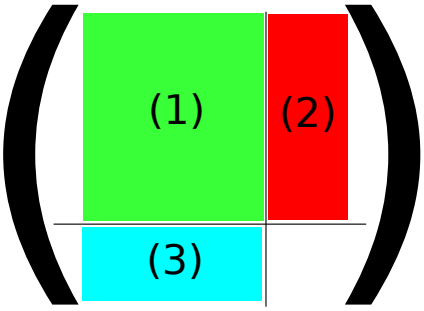

Figure A.3: Illustration of the first implementation of the assembly using Expression Template

Component-wise Expression Template implementation

To exploit the block diagonal structure of the matrix A, we can split the different components of the velocity into different blocks and assemble a 4×4 block matrix. This implementation turns out to be faster than the previous one with 4.21s, but still not as fast as the "loop by hand" strategy.

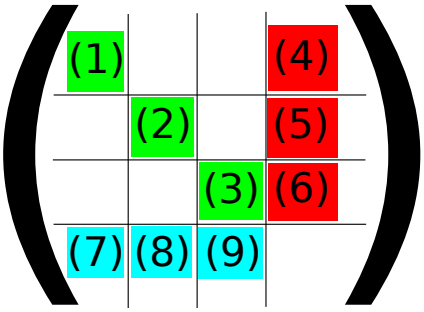

Figure A.4: Illustration of the component-wise implementation of the assembly using Expression Template

Block copy Expression Template implementation

The "loop by hand" not only exploits the fact that the matrix A is block diagonal, but also that its blocks are equal. We can reproduce this by assembling a small matrix and then copy it on all the diagonal blocks. This yields a faster assembly with 3.69s.

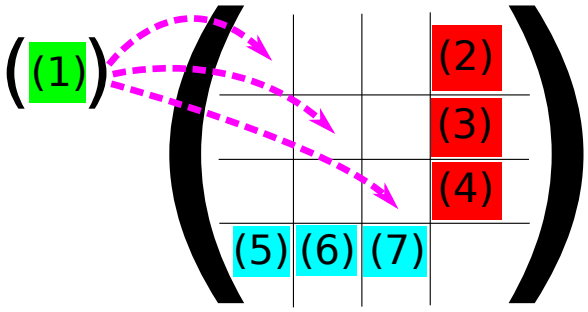

Figure A.5: Illustration of the implementation of the assembly using Expression Template and copy of blocks

Block copy Expression Template implementation with matrix restructuring

The fastest implementation that we could achieve with the Expression Template consists in using a different structure for the matrix A and B: the matrix A is assembled by considering a 4×4 block matrix and using the block copy. The structure of the matrix is then changed on the fly (this is a cheap operation) and each block B or B^T is assembled in one shot. This last implementation yield a timing of 3.00s, thus competing with the "loop by hand" strategy.

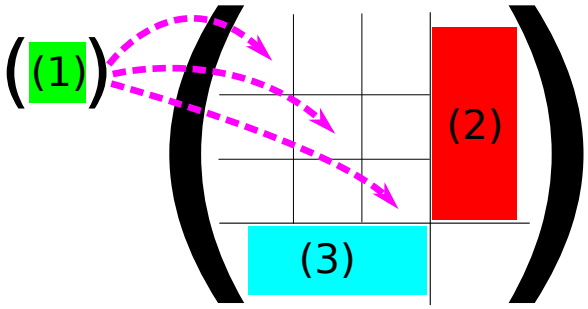

Figure A.6: Illustration of the implementation of the assembly using Expression Template, copy of blocks and restructuration of the matrix.

Remark that even if this last implementation seems complicated, it is performed using only 16 C++ code lines, hence remaining easily accessible for the end user.

A.7 Conclusion

The implementation proposed here for the assembly of finite element systems has demonstrated its ability to compete with classical implementation for simple discrete formulation, even surpassing it for more complicated formulations. It is moreover very flexible, thanks to the block structures that can be set up. The user interface, which was the starting point of this work, allows a fast and clear development of finite element approximations. Moreover, the proposed implementation does not require heavy metaprogramming and is therefore well

accepted by modern compilers.

The possibilities to broaden the abilities of the implementation are nearly boundless. Some extensions are already available in the library LifeV, including:

- additional expressions such as the cell size, interpolated values, interpolated gradient and functors;
- a neater interface for the `Expression` to be integrated, which accepts the code `3.0*dot( grad(phi_j),grad(phi_i));`
- assembly of the right hand side of the system and computation of domain integral, which might be useful for error computations for example;

Computation of boundary integrals is currently under development, but no difficulty with respect to the implementation is forseen.

Bibliography

[1] S. Aliabadi, K. Shujaee, and T. Tezduyar. Parallel simulation of two-phase flow problems using the finite element method. In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pages 113–120. IEEE, 1999.

[2] C. Altamirano, A. Illanes, S. Becerra, J.J. Cairó, and F. Gòdia. Considerations on the lactate consumption by CHO cells in the presence of galactose. *Journal of Biotechnology*, 125(4):547–556, 2006.

[3] P. M. A. Areias and T. Belytschko. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(5):760–788, 2005.

[4] Y. Bazilevs and T.J.R. Hughes. Weak imposition of Dirichlet boundary conditions in fluid mechanics. *Computers & fluids*, 36(1):12–26, 2007.

[5] E. Béchet, H. Minnebo, N. Moës, and B. Burgardt. Improved implementation and robustness study of the X-FEM for stress analysis around cracks. *International Journal for Numerical Methods in Engineering*, 64(8):1033–1056, 2005.

[6] J. Bedrossian, J.H. von Brecht, S. Zhu, E. Sifakis, and J. M. Teran. A second order virtual node method for elliptic problems with interfaces and irregular domains. *Journal of Computational Physics*, 229(18):6405–6426, 2010.

[7] M. Behr. On the application of slip boundary condition on curved boundaries. *International Journal for Numerical Methods in Fluids*, 45(1):43–51, 2004.

[8] T. Belytschko, N. Moes, S. Usui, and C. Parimi. Arbitrary discontinuities in finite elements. *International Journal for Numerical Methods in Engineering*, 50:993–1013, 2001.

[9] J.M. Berg, J.L. Tymoczko, and L. Stryer. Biochemistry, 2006.

[10] M. Braack, E. Burman, V. John, and G. Lube. Stabilized finite element methods for the generalized Oseen problem. *Computer Methods in Applied Mechanics and Engineering*, 196(4):853–866, 2007.

[11] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2008.

[12] A.N. Brooks and T.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(1):199–259, 1982.

[13] R.L. Buchanan, R.C. Whiting, and W.C. Damert. When is simple good enough: a comparison of the Gompertz, Baranyi, and three-phase linear models for fitting bacterial growth curves. *Food Microbiology*, 14(4):313–326, 1997.

[14] R. Bürger, W.L. Wendland, and F. Concha. Model equations for gravitational sedimentation-consolidation processes. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 80(2):79–92, 2000.

[15] E. Burman. Consistent SUPG-method for transient transport problems: Stability and convergence. *Computer Methods in Applied Mechanics and Engineering*, 199(17–20):1114–1123, 2010.

[16] Susanna Carcano. A model for cell growth in batch bioreactors. Master's thesis, Politecnico di Milano, 2010.

[17] J. Chessa and T. Belytschko. An enriched finite element method and level sets for axisymmetric two-phase flow with surface tension. *International Journal for Numerical Methods in Engineering*, 58(13):2041–2064, 2003.

[18] J. Chessa, P. Smolinski, and T. Belytschko. The extended finite element method (XFEM) for solidification problems. *International Journal for Numerical Methods in Engineering*, 53:1959–1977, 2002.

[19] D.L. Chopp. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106:77–91, 1993.

[20] D.L. Chopp. Some improvements of the fast marching method. *SIAM Journal on Scientific Computing*, 23:230–244, 2001.

[21] R. Codina. Numerical solution of the incompressible Navier–Stokes equations with Coriolis forces based on the discretization of the total time derivative. *Journal of Computational Physics*, 148(2):467–496, 1999.

[22] A.H. Coppola-Owen and R. Codina. Improving Eulerian two-phase flow finite element approximation with discontinuous gradient pressure shape functions. *International Journal for Numerical Methods in Fluids*, 49:1287–1304, 2005.

[23] H. Coppola-Owen and R. Codina. A free surface finite element model for low Froude number mould filling problems on fixed meshes. *International Journal for Numerical Methods in Fluids*, 66(7):833–851, 2011.

[24] T.A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):165–195, 2004.

[25] D. De Sanctis and M. Perrone. Numerical Analysis of Fluid Dynamics and Oxygen Diffusion in Shaking Bioreactors. Master's thesis, Universita degli studi di Roma "Tor Vergata", 2007.

[26] D.A. Di Pietro, S. Lo Forte, and N. Parolini. Mass preserving finite element implementations of the level set method. *Applied Numerical Mathematics*, 56(9):1179–1195, 2006.

[27] D.A. Di Pietro and A. Veneziani. Expression templates implementation of continuous and discontinuous galerkin methods. *Computing and visualization in science*, 12(8):421–436, 2009.

[28] M. Discacciati, D. Hacker, A. Quarteroni, S. Quinodoz, S. Tissot, and F. M. Wurm. Numerical simulation of orbitally shaken viscous fluids with free surface. *International Journal for Numerical Methods in Fluids*, 2012. Accepted.

[29] D.A. Dunavant. High degree efficient symmetrical gaussian quadrature rules for the triangle. *International Journal for Numerical Methods in Engineering*, 21:1129–1148, 1985.

[30] C. Dyring, HA Hansen, and C. Emborg. Observations on the influence of glutamine, asparagine and peptone on growth and t-PA production of Chinese hamster ovary (CHO) cells. *Cytotechnology*, 16(1):37–42, 1994.

[31] H. Elman, V.E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 227(3):1790–1808, 2008.

[32] M. S. Engelman, R. L. Sani, and P. M. Gresho. The implementation of normal and/or tangential boundary conditions in finite element codes for incompressible fluid flow. *International Journal for Numerical Methods in Fluids*, 2(3):225–238, 1982.

[33] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[34] A. Ern and J.L. Guermond. *Theory and practice of finite elements*, volume 159. Springer Verlag, 2004.

[35] M.B. Fogolin, R. Wagner, M. Etcheverrigaray, and R. Kratje. Impact of temperature reduction and expression of yeast pyruvate carboxylase on hGM-CSF-producing CHO cells. *Journal of Biotechnology*, 109(1):179–191, 2004.

[36] O. Fortmeier and H.M. Bucker. Parallel re-initialization of level set functions on distributed unstructured tetrahedral grids. *Journal of Computational Physics*, 230(12):4437–4453, 2011.

[37] T.-P. Fries and T. Belytschko. The extended/generalized finite element method: An overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(3):253–304, 2010.

[38] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[39] C. Geuzaine and J.F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[40] G. Grandperrin. Numerical Approximation of Incompressible Free-Surface Flows. Master's thesis, EPFL, 2010.

[41] S. Gross, V. Reichelt, and A. Reusken. A finite element based level set method for two-phase incompressible flows. *Computing and Visualization in Science*, 9(4):239–257, 2006.

[42] S. Gross and A. Reusken. An extended pressure finite element space for two-phase incompressible flows with surface tension. *Journal of Computational Physics*, 224(1):40–58, 2007.

[43] A. Hansbo and P. Hansbo. An unfitted finite element method, based on Nitsche's method, for elliptic interface problems. *Computer Methods in Applied Mechanics and Engineering*, 191:5537–5552, 2002.

[44] P. M. Hayter, E. M. A. Curling, A. J. Baines, N. Jenkins, I. Salmon, P. G. Strange, and A. T. Bull. Chinese hamster ovary cell growth and interferon production kinetics in stirred batch culture. *Applied Microbiology and Biotechnology*, 34(5):559–564, 1991.

[45] X. He, T. Lin, and Y. Lin. Immersed finite element methods for elliptic interface problems with non-homogeneous jump conditions. *International Journal of Numerical Analysis and Modeling*, 8(2):284–301, 2010.

[46] M. Heroux, R. Bartlett, V.H.R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[47] M. Herrmann. A domain decomposition parallelization of the fast marching method. *Annual Research Briefs-2003*, pages 213–225, 2003.

[48] C.W. Hirt and B.D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39(1):201–225, 1981.

[49] J.K. Hong, S.M. Cho, and S.K. Yoon. Substitution of glutamine by glutamate enhances production and galactosylation of recombinant IgG in Chinese hamster ovary cells. *Applied Microbiology and Biotechnology*, 88(4):869–876, 2010.

[50] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39):4135–4195, 2005.

[51] J.-S. Huh and J.A. Sethian. Private communications.

[52] J.-S. Huh and J.A. Sethian. Exact subgrid interface correction schemes for elliptic interface problems. *Proceedings of the National Academy of Sciences of the United States of America*, 105(29):9874–9879, 2008.

[53] S. Hysing, S. Turek, D. Kuzmin, N. Parolini, E. Burman, S. Ganesan, and L. Tobiska. Quantitative benchmark computations of two-dimensional bubble dynamics. *International Journal for Numerical Methods in Fluids*, 60(11):1259–1288, 2009.

[54] H. Ji and J. E. Dolbow. On strategies for enforcing interfacial constraints and evaluating jump conditions with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 61(14):2508–2535, 2004.

[55] B.L. Vaughan Jr, B.G. Smith, and D.L. Chopp. A comparison of the extended finite element method with the immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *Communications in Applied Mathematics and Computational Science*, 1:207–228, 2006.

[56] S.H. Kim and G.M. Lee. Functional expression of human pyruvate carboxylase for reduced lactic acid formation of Chinese hamster ovary cells (DG44). *Applied Microbiology and Biotechnology*, 76(3):659–665, 2007.

[57] P. Knobloch. On the definition of the SUPG parameter. *Electronic Transactions on Numerical Analysis*, 32:76–89, 2008.

[58] S. Kurioka and D.R. Dowling. Numerical simulation of free surface flows with the level set method using an extremely high-order accuracy WENO advection scheme. *International Journal of Computational Fluid Dynamics*, 23(3):233–243, 2009.

[59] M.S. Lao and D. Toth. Effects of ammonium and lactate on growth and metabolism of a recombinant Chinese hamster ovary cell culture. *Biotechnology progress*, 13(5):688–691, 1997.

[60] J. Li, J. M. Melenk, B. Wohlmuth, and J. Zou. Optimal a priori estimates for higher order finite elements for elliptic interface problems. *Applied Numerical Mathematics*, 60(1–2):19–37, 2010.

[61] Z. Li. The immersed interface method using a finite element formulation. *Applied Numerical Mathematics*, 27(3):253–267, 1998.

[62] Z. Li and M.-C. Lai. The immersed interface method for the Navier–Stokes equations with singular forces. *Journal of Computational Physics*, 171(2):822–842, 2001.

[63] J.L. Lions and E. Magenes. *Non-Homogeneous Boundary Value Problems and Applications*. Springer, 1972.

[64] S. Lu, X. Sun, C. Shi, and Y. Zhang. Determination of tricarboxylic acid cycle acids and other related substances in cultured mammalian cells by gradient ion-exchange chromatography with suppressed conductivity detection. *Journal of Chromatography A*, 1012(2):161–168, 2003.

[65] S. Lu, X. Sun, and Y. Zhang. Insight into metabolism of CHO cells at low glucose concentration on the basis of the determination of intracellular metabolites. *Process Biochemistry*, 40(5):1917–1921, 2005.

[66] E. Marchandise and J.-F. Remacle. A stabilized finite element method using a discontinuous level set approach for solving two phase incompressible flows. *Journal of Computational Physics*, 219(2):780–800, 2006.

[67] J.C. Martin and W.J. Moyce. Part iv. an experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 244(882):312–324, 1952.

[68] B. Maury. Numerical analysis of a finite element/volume penalty method. *SIAM Journal on Numerical Analysis*, 47(2):1126–1148, 2009.

[69] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999.

[70] D. R. Noble, E. P. Newren, and J. B. Lechman. A conformal decomposition finite element method for modeling stationary fluid interface problems. *International Journal for Numerical Methods in Fluids*, 63(6):725–742, 2010.

[71] R.P. Nolan and K. Lee. Dynamic model of CHO cell metabolism. *Metabolic Engineering*, 13(1):108–124, 2011.

[72] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[73] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[74] N. Parolini. *Computational Fluid Dynamics for Naval Engineering Problems*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2004.

[75] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-based fast local level set method. *Journal of Computational Physics*, 115:410–438, 1999.

[76] T.J. Piva and E. Mcevoy-Bowe. Oxidation of glutamine in HeLa cells: role and control of truncated TCA cycles in tumour mitochondria. *Journal of Cellular Biochemistry*, 68(2):213–225, 1998.

[77] A. Provost, G. Bastin, S. Agathos, and Y. Schneider. Metabolic design of macroscopic bioreaction models: application to Chinese hamster ovary cells. *Bioprocess and Biosystems Engineering*, 29:349–366, 2006.

[78] C. Prud'Homme. A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations. *Scientific Programming*, 14(2):81–110, 2006.

[79] A. Quarteroni. *Numerical Models for Differential Problems*. Springer Verlag, 2009.

[80] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2007.

[81] A. Quarteroni, F. Saleri, and A. Veneziani. Factorization methods for the numerical approximation of Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 188(1):505–526, 2000.

[82] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer, Berlin, 1994.

[83] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Clarendon Press, 1999.

[84] I. Robertson, S. J. Sherwin, and J. M. R. Graham. Comparison of wall boundary conditions for numerical viscous free surface flow simulation. *Journal of Fluids and Structures*, 19(4):525–542, 2004.

[85] M. Sala and M. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005.

[86] M.A. Savageau. Growth equations: A general equation and a survey of special cases. *Mathematical Biosciences*, 48(3-4):267–278, 1980.

[87] R. I. Saye and J.A. Sethian. Analysis and applications of the Voronoi implicit interface method. *Journal of Computational Physics*, 2012. Accepted.

[88] M. Schneider, I.W. Marison, and U. Von Stockar. The importance of ammonia in mammalian cell culture. *Journal of Biotechnology*, 46(3):161–185, 1996.

[89] R. Scott. Interpolated boundary conditions in the finite element method. *SIAM Journal on Numerical Analysis*, 12(3):404–427, 1975.

[90] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, second edition, 1999.

[91] A. Smolianski. Finite-element/level-set/operator-splitting (FELSOS) approach for computing two-fluid unsteady flows with free moving interfaces. *International Journal for Numerical Methods in Fluids*, 48(3):231–269, 2005.

[92] M. Souli and J. P. Zolesio. Arbitrary Lagrangian–Eulerian and free surface methods in fluid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 191(3):451–466, 2001.

[93] M. Sussman and E.G. Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162(2):301–337, 2000.

[94] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, 1994.

[95] S. Tissot. *OrbShake Bioreactors for Mammalian Cell Cultures : Engineering and Scale-up*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2011.

[96] S. Tissot, M. Farhat, D.L. Hacker, T. Anderlei, M. Kühner, C. Comninellis, and F. Wurm. Determination of a scale-up factor from mixing time studies in orbitally shaken bioreactors. *Biochemical Engineering Journal*, 52(2-3):181–186, 2010.

[97] A.K. Tornberg. Multi-dimensional quadrature of singular and discontinuous functions. *BIT Numerical Mathematics*, 42(3):644–669, 2002.

[98] A.K. Tornberg and B. Engquist. A finite element based level-set method for multiphase flow applications. *Computing and Visualization in Science*, 3(1):93–101, 2000.

[99] E. Trummer, K. Fauland, S. Seidinger, K. Schriebl, C. Lattenmayer, R. Kunert, K. Vorauer-Uhl, R. Weik, N. Borth, H. Katinger, et al. Process parameter shifting: Part i. effect of DOT, pH, and temperature on the performance of Epo-Fc expressing CHO cells cultivated in controlled batch bioreactors. *Biotechnology and Bioengineering*, 94(6):1033–1044, 2006.

[100] M.C. Tugurlan. *Fast Marching Methods-Parallel Implementation and Analysis*. PhD thesis, Louisiana State University, 2008.

[101] D. Vandevoorde and N.M. Josuttis. *C++ Templates*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[102] T. Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, 1995.

[103] L. Ville, L. Silva, and T. Coupez. Convected level set method for the numerical simulation of fluid buckling. *International Journal for Numerical Methods in Fluids*, 66(3):324–344, 2011.

[104] S. Werner, R. Eibl, C. Lettenbauer, M. Roll, D. Eibl, M. De Jesus, X. Zhang, M. Stettler, S. Tissot, C. Burkie, et al. Innovative, non-stirred bioreactors in scales from milliliters up to 1000 liters for suspension cultures of cells using disposable bags and containers a Swiss contribution. *CHIMIA International Journal for Chemistry*, 64(11):819–823, 2010.

[105] W. Wiechert. 13c metabolic flux analysis. *Metabolic engineering*, 3(3):195–206, 2001.

[106] C. Winkelmann. *Interior Penalty Finite Element Approximation of Navier-Stokes Equations and Application to Free Surface Flows*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2007.

[107] F. Zhang, X. Sun, X. Yi, and Y. Zhang. Metabolic characteristics of recombinant Chinese hamster ovary cells expressing glutamine synthetase in presence and absence of glutamine. *Cytotechnology*, 51(1):21–28, 2006.

[108] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery (SPR) and adaptive finite element refinement. *Computer Methods in Applied Mechanics and Engineering*, 101(1-3):207–224, 1992.

[109] P. Zunino, L. Cattaneo, and C. M. Colciago. An unfitted interface penalty method for the numerical approximation of contrast problems. *Applied Numerical Mathematics*, 61(10):1059–1076, 2011.

[110] M.H. Zwietering, J.T. De Koos, B.E. Hasenack, J.C. De Witt, and K. Van't Riet. Modeling of bacterial growth as a function of temperature. *Applied and Environmental Microbiology*, 57(4):1094–1101, 1991.

Curriculum vitæ

Samuel Quinodoz

Personal details

Date of birth	December 18th, 1984
Place of birth	Sion (VS), Switzerland

Academic education

2009 - 2012	Ph.D. student at the Ecole Polytechnique Fédérale de Lausanne (EPFL). Thesis title: Numerical simulation of orbitally shaken reactors. Advisors: Prof. A. Quarteroni and Dr. M. Discacciati.
2007 - 2009	Master degree in mathematical engineering at EPFL. Prize for the best GPA for the complete cycle at EPFL (2009). Prize for the best GPA for the Master cycle at EPFL (2009). Prize for the best GPA in mathematical engineering (2009). Thesis title: Volume penalty and Nitsche type methods for convective heat transfer problems. Advisors: Prof. A. Quarteroni and Prof. E. Burman
2004 - 2007	Bachelor degree in mathematics at EPFL. Prize for the best GPA for the propedeutic year at EPFL (2005).

Scientific publications

- M. Discacciati, A. Quarteroni, and S. Quinodoz. Numerical approximation of internal discontinuity interface problems. *Submitted*, 2012.
- M. Discacciati, D. Hacker, A. Quarteroni, S. Quinodoz, S. Tissot and F.M. Wurm (2012), Numerical simulation of orbitally shaken viscous fluids with free surface. International Journal for Numerical Methods in Fluids. doi: 10.1002/fld.3658

Conferences

- RMMM 2011, Reliable Methods of Mathematical Modeling. Contributed talk in plenary session: *Free surface flow simulations for Orbitally Shaken Reactors.* EPFL, Lausanne (Switzerland), July 3rd, 2011.
- FEF 2011, 16^{th} International Conference on Finite Elements in Flow Problems. Contributed talk in plenary session: *Two fluids free surface flow simulation with large density ratio.* Munich (Germany), 25th March, 2011.
- Immersed boundaries and fictitious domain methods: theory and applications. Contributed talk in plenary session: *An efficient method for solving elliptic problems with interior discontinuities.* Marseille (France), 31th August 2010.