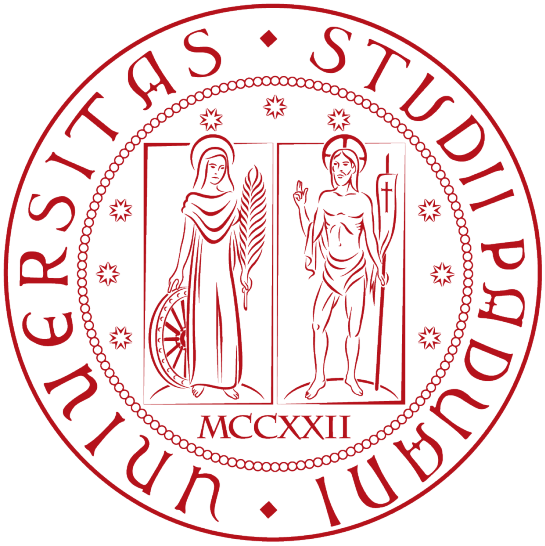
UNIVERSITÁ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria Civile, Edile e Ambientale
Corso di Laurea Magistrale in Mathematical Engineering



TESI DI LAUREA

A novel Shifted Boundary Method (SBM) for embedded domains based on Multi-Point Constraints.

Relatori:
Prof. Rubén Zorrilla
Prof. Antonia Larese
Prof. Riccardo Rossi

Nicoló Antonelli
n. 2016824

Anno Accademico 2022-23

Contents

1 Introduction **1**
1.1 Why immersed objects . . . 1
1.2 Body-fitted approach . . . 2
1.3 Unfitted methods . . . 4
1.4 Objectives . . . 7

2 Preliminaries for the SBM **8**
2.1 Mathematical notation . . . 8
2.2 Surrogate boundary . . . 9
2.3 Dirichlet surrogate boundary conditions . . . 11

3 Poisson problem **14**
3.1 Nitsche variational forms of the Poisson problem . . . 14
3.2 Nitsche variational form for the SB method . . . 15
3.3 Nitsche symmetric variational form for the SB method . . . 17
3.4 Stability of the symmetric formulation . . . 18
3.4.1 Coercivity of $a_s^h(\cdot,\cdot)$ . . . 19
3.4.2 Continuity of $a_s^h(\cdot,\cdot)$ . . . 20
3.4.3 Well-posedness . . . 22
3.4.4 Consistency . . . 22
3.4.5 Convergence of the error . . . 23

4 Convection-Diffusion problem **26**
4.1 Nitsche variational form for the convection-diffusion problem . . . 26
4.2 Nitsche variational form for the SB method . . . 27
4.3 Conservation properties . . . 29
4.4 Stability of the formulation . . . 30
4.4.1 Coercivity of $a^h(\cdot,\cdot)$ . . . 30
4.4.2 Consistency . . . 32
4.4.3 Convergence of the error . . . 33

5 Implementation details **36**
5.1 Kratos Multi-physics . . . 36
5.2 General strategy . . . 37
5.3 Shifted boundary . . . 37
5.3.1 Import the skin boundary . . . 38
5.3.2 Level set function . . . 39
5.3.3 Find the surrogate boundary . . . 41
5.4 Imposition of the Dirichlet BC . . . 43
5.4.1 Taylor expansion . . . 44
5.4.2 Computation of the closest-point . . . 45
5.4.3 Computation of the gradient term . . . 48
5.5 Multi-point constraints in SB method . . . 48
5.5.1 MPCs . . . 49
5.5.2 MPCs in Kratos . . . 52
5.5.3 MPCs for SB method . . . 52
5.5.4 Additional problems . . . 54
5.6 Flux through the surrogate boundary . . . 59
5.7 SB Solvers . . . 63

5.7.1 Stationary SBM solver 63
5.7.2 Transient SBM solver 63
5.7.3 Moving objects solver 65

6 Results **66**
6.1 Poisson problem with linear exact solution 66
6.2 Poisson equation with uniform boundary conditions 68
6.3 Poisson equation with non-uniform boundary conditions 75
6.4 Complicated domains with corners 78
6.5 Transient Poisson problem 89
6.6 Stationary convection-diffusion problem 93
6.7 Transient convection-diffusion problem 100
6.8 Moving objects 103

7 Conclusions **112**

8 Appendix **115**
8.1 Appendix 1 115
8.2 Appendix 2 115

Abstract

This project presents a recent finite element approach for embedded domains that belongs to the category of approximate boundary methods: the Shifted Boundary Method (SBM). This is an unfitted approach used to represent embedded boundaries. SBM suggests shifting the location of boundary conditions from the true boundary to a surrogate boundary and adjusting the Dirichlet boundary conditions appropriately to preserve the optimal convergence rate. The theory and implementation details of the developed formulations in the open-source Kratos Multiphysics software have been comprehensively covered for Poisson and convection-diffusion problems. In addition, we introduced a novel approach based on Multi-Point constraints. The new technique allows the method to impose the new Dirichlet boundary conditions, which depend on the gradient of the solution, without introducing any nonlinear loop. The new shifted boundary method was validated by performing convergence studies on stationary and transient convection-diffusion problems, also testing embedded complex inner and outer boundaries.

1 Introduction

1.1 Why immersed objects

The simulation of immersed objects in a fluid domain is a common problem in many engineering applications, such as in aerodynamics and hydrodynamics, as well as in industrial processes, such as heat transfer and fluid flow.
In these kinds of situations, running a simulation of the flow of fluid and the transfer of heat around an object that is immersed is absolutely necessary if one wants to gain knowledge of the performance of the system and enhance the design process [24, 30]. In the aerospace business, the investigation of aerodynamics around aeroplanes and spacecraft is an important part of their design and performance [23]. One common application of this kind of simulation is in the aerospace industry. The immersed object in these simulations is often the aeroplane or spaceship, and the fluid domain is the surrounding air. In order for the simulation to be accurate, it is necessary to take into consideration not only the shape of the object but also its motion in the fluid.
The study of hydrodynamics, which seeks to understand the flow of fluids in complex settings such as rivers, lakes, and oceans, represents an alternative use of this type of simulation [13]. The purpose of this study is to understand the flow of fluids. In these simulations, the object that is immersed can be anything from a simple geometric shape like a cylinder or a sphere to a more complex object like a ship or a submarine. It can even be a combination of these two extremes.

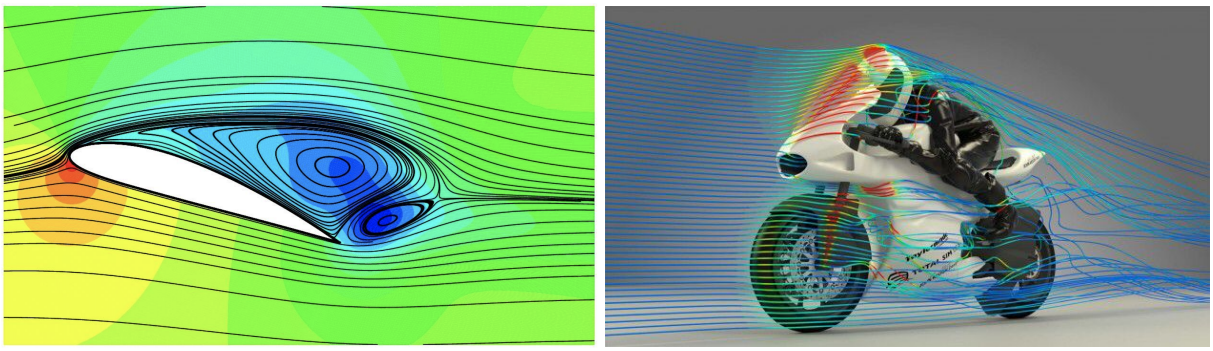


Figure 1: Two examples of CFD simulations: the pressure vector field around an airfoil in 2D on the left and around a motorcycle in 3D on the right.

Simulation of fluid flow and heat transfer around an immersed object is essential for the design of industrial processes. These operations include the cooling of electronic components, the heating of liquids in pipelines, and the optimization of energy transfer. The immersed body in these simulations is often a component or a device, and the fluid domain is the air or liquid that is being utilised for heating or cooling. These simulations are carried out using software on computers. These kinds of numerical simulations are particularly delicate because of the presence of immersed objects, the main difficulty is how to correctly represent it and its effects on the rest of the domain; how to obtain accurate results without spending too much time in the representation of the object geometry and the construction of the mesh.
Concluding this brief introduction, the modelling of immersed objects in a fluid domain is still a critical challenge in many engineering applications. Furthermore, it is essential for understanding the performance of systems immersed in fluid domains and enhancing their design.

1.2 Body-fitted approach

In all the previously mentioned real applications, engineers usually simulate their physical phenomena using commercial software that use a conforming mesh to deal with objects which are immersed. A conforming mesh is a mesh that is constructed to fit closely around an immersed object without gaps or overlaps. Because it ensures that the fluid flow and heat transfer around the immersed object is accurately represented, this type of mesh is required for precise simulation results.
In CFD simulations, the construction of conforming meshes for immersed objects, can be accomplished in several ways. The use of a body-fitted mesh, which is a mesh that is generated specifically for a particular object and is designed to conform closely to its shape, is one of the most common methods. Advancing front methods [25] and Delaunay triangulation method [36, 40] are some examples of algorithms that can be used to generate this kind of mesh. Other possible algorithms include the use of mesh adaptation [14, 40] which is another common technique for the construction of conforming meshes.
This technique entails making adjustments to the mesh after it has been generated to improve its conformity to the object in question. This can be accomplished through the utilisation of many different methods, including mesh smoothing, mesh optimization, and mesh refinement.
In a CFD simulation, the utilisation of conforming meshes is advised for the accurate representation of the fluid flow or heat transfer around an immersed object. By ensuring that the mesh closely encloses the object and accurately represents its shape, simulation results are more precise and provide a deeper understanding of the system's behaviour.

Therefore, the body-fitted technique helps, in a CFD context, to explicitly define the interface between the fluid phase and the solid body when a finite element method (FEM) is applied to simulate the physical phenomena. The interface fluid object is made up of a specific number of conforming mesh nodes and elements.
This suggests that, in a two-dimensional problem, the interface of an object that is immersed is represented as a polygon composed of one-dimensional elements in the space. In a CFD scenario, because the finite element method attempts to compute the unknown quantities on the nodes, we may simply impose a zero velocity constraint in all of the interface nodes when we need to enforce a no-slip requirement, for example.

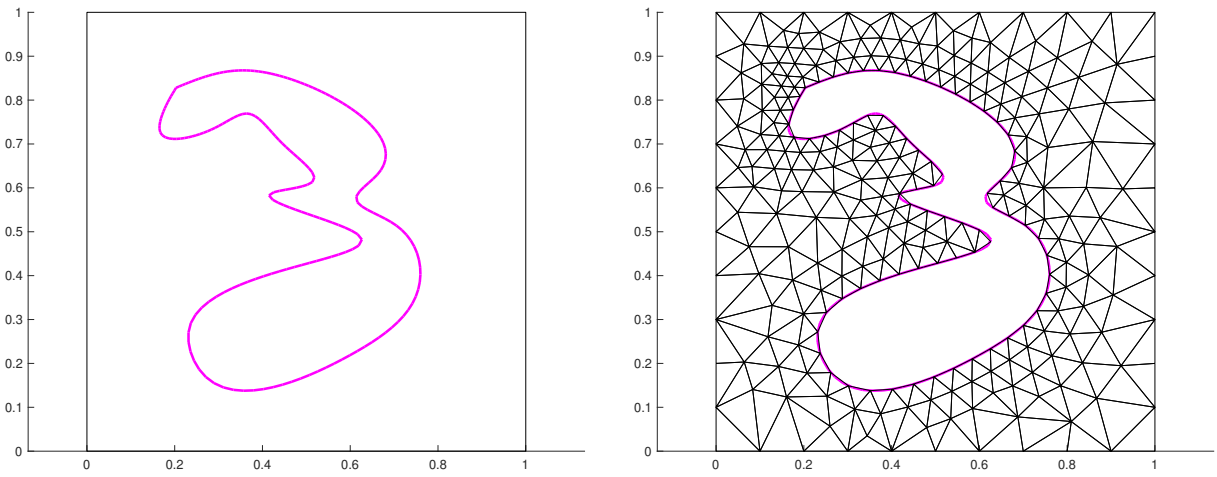


Figure 2: In the left panel, an immersed object is represented in a square computational domain $[0, 0] \times [1, 1]$ and in the right panel a mesh using the body-fitted approach has been constructed.

Therefore, in the case of a fixed immersed solid body, once the mesh has been constructed, we

could eliminate the solid region and consider the fluid-solid interface as a boundary, substituting the equations pertaining to those nodes with the Dirichlet boundary conditions.

While body-fitted mesh techniques are widely used in CFD simulations to represent immersed objects, they have several limitations. One of the main problems is that making and changing the mesh takes a lot of computing power. Body-fitted mesh generation algorithms can take a lot of processing power, and adapting the mesh can take even more processing power. This can make it hard to model larger or more complicated objects with this method.
Another problem is that it is hard to deal with changes in the shape of an object that is immersed. Imagine, for instance, that we want to slightly modify the object shape to see if the cooling process may be faster, then at least a part of the mesh needs to be deleted and re-meshed. Or it may happen that, during the simulation, the shape of the object changes, for example, because it deforms or moves, the mesh must be changed to account for the boundary movements. This can be a complicated and time-consuming process that can make it hard to model systems that change over time. Lastly, the accuracy of the mesh representation is often what limits body-fitted mesh techniques. The simulation results may be affected if the mesh is not generated with sufficient precision or if the adaptation techniques fail to improve its conformity to the object.
Even with these problems, body-fitted mesh techniques are often used to represent immersed objects in CFD commercial software because, if the mesh is built correctly, it can simulate fluid flow and heat transfer very accurately. This leads to what is called body-fitted discretization. Even though these methods are appealing, they have clear drawbacks that become clear when simulating bodies that are moving.

In reality, the elements surrounding the immersed object and the nodes defining the solid object in the body-fitted formulation are both adherent to the representation. This means that if the solid object moves, the elements may be excessively stretched or even flipped, resulting in potential losses.

Numerous techniques, including the re-meshing technique [40], exist to solve the problem of large boundary movement when conforming meshes are used. Re-meshing at each time step or each non-linear iteration allows us to circumvent the problem of stretched elements; indeed, with re-meshing, the elements surrounding a solid object are not always identical during boundary movements, but new elements substitute the stretched ones. In this case, it is essential to divide the motion into small steps, with each step being sufficient to keep elements from becoming distorted or inverted. However, as expected, the re-meshing method required a significant amount of computational time, especially when boundary movements are significant. It is possible that is required to re-mesh hundreds of times during a simulation. Additionally, the meshing process is not always straightforward in many applications, such as when tiny details are essential. In these cases, an automatic re-meshing procedure may be dangerous since bad quality meshes that lead to non-accurate results can be obtained.

Re-meshing aims to maintain an accurate and conformal representation of the objects being modelled while minimising the computational cost of regenerating the mesh. There are numerous re-meshing methods, each with its own advantages and disadvantages. Using an adaptive mesh refinement technique [9] that adds or removes mesh elements based on the solution and gradients is a common method. Another possibility is to use a moving mesh technique [21], in which the mesh is permitted to move with the objects being modelled. This implementation is more difficult, but it can provide a more accurate representation of the objects and their interactions with the fluid. However, it can be computationally intensive and may necessitate additional algorithms for mesh stabilisation and re-meshing.

The impossibility to transfer the precise CAD geometry of the solid body is another drawback of adopting a body-fitted formulation. The body form used by the finite element solver will undoubtedly have some inaccuracies because the mesh is often made of triangles or quadrilateral pieces of finite size. A curved shape or extremely microscopic details, for example, cannot be completely taken into account. In actual applications, an engineer is always needed to fix any issues with mesh generation. This is especially true for minor details or 3D cases, which results in high simulation costs because there is currently no reliable algorithm to do it precisely.

1.3 Unfitted methods

The reader should now be able to comprehend the justification behind the development of unfitted methods. Unfitted methods offer an alternative for representing immersed objects in a computational domain. In particular, the main idea of all unfitted methods is to use the same computational mesh regardless of the immersed object. Thus, they were born to simplify the procedure of meshing immersed objects and make it completely independent of the computational mesh of the fluid domain.
When compared to their body-fitting counterparts, unfitted or non-conforming methods have a more complicated implementation and, in general, exhibit lower levels of accuracy when the same mesh size is used. This is because unfitted or non-conforming methods do not conform to the shape of the body. On the other hand, they offer significant advantages in terms of the management of significant border movements and ease of the simulation process.
The most significant distinction between the body-fitted and the unfitted formulation is that, in the latter one, the discretization of the fluid and the discretization of the solid body are carried out in a manner that is entirely separate from one another. In other words, in unfitted formulations first, the mesh of the fluid is made, then the one of the immersed solid, and finally, we overlap the two. This enables us to completely eliminate the time cost of adjusting the mesh around the immersed solid object to accurately represent it without any stretched elements; all that is required is to refine the grid around the object as we deem appropriate because some approximations will be made to impose on the fluid the effects of the presence of the immersed body.

The ability to simply simulate moving objects or large displacements without introducing an additional and time-consuming re-meshing procedure is the key advantage that has led to the development of so many different unfitted approaches in recent times. Therefore, the primary distinction between the unfitted and body-fitted formulations with re-meshing is that, even with large boundary movements, we always can reuse the same background mesh.

Unfitted methods distinguish from each other in how they take into account the presence of an immersed object in the computational domain. Since the object skin overlaps the background or computational mesh, unfitted methods need to find a way to first locate where the immersed object is and, secondly, impose in the stiffness matrix the effect of the immersed object on the fluid domain. Between all the unfitted methods we may mention the family of Embedded Boundary methods (EBM) and the Shifted Boundary method (SBM).

The first family of methods, where for instance we may mention the X-FEM method [20] or the Cut-FEM method [41, 42, 43] are based on the idea of finding the intersected elements. The intersected elements are the ones cut by the fluid-object interface. The family of EBMs tries to reconstruct the embedded boundary within the cut elements creating, for instance, new smaller

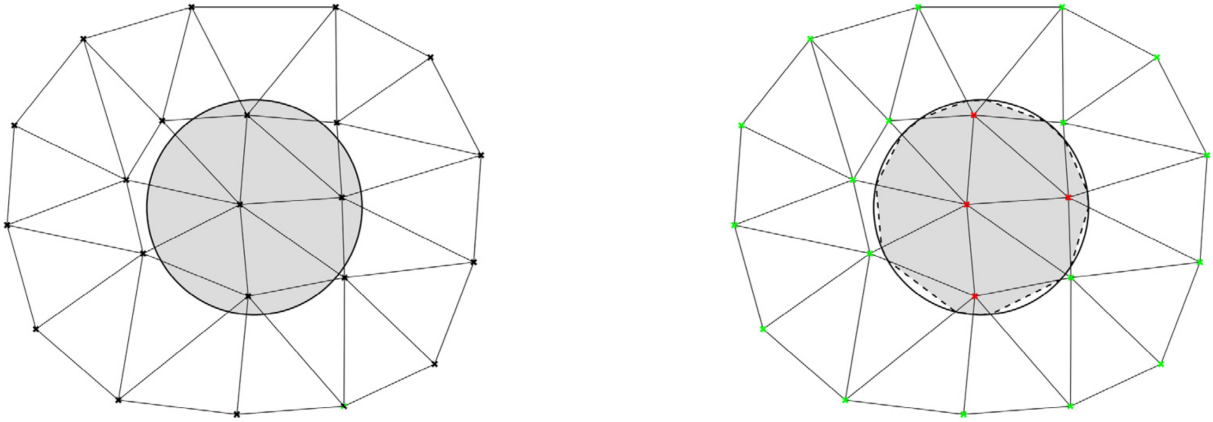


Figure 3: Images were taken by [41] where the Cut-FEM method is considered. A continuous distance function has been used in order to have the representation of the embedded object as its zero isosurface (dashed line).

elements. In this way, the background mesh is able to represent, with a good approximation, the object shape in the intersected elements and then accurately impose the boundary conditions. In Figure 3, an example from [41] shows the intersections between the background elements and the object skin boundary; in this case, the Cut-FEM approach identifies the dashed line which is the location where the interface boundary conditions are imposed.
The main problem with these kinds of methods is the so-called small cut-cell problem. It frequently happened that arbitrary small intersections form between the background mesh and the immersed skin object, which leads to instabilities due to the formation of microelements. In [31] they try to introduce a bubble function on each of the cut elements for stabilizing them and in [2, 3] other approaches have been implemented to overcome the problem of small cut-cell.

On the other hand, the SBM method [26, 27] can be considered an unfitted method, but it also belongs to the family of approximate boundary methods. Actually, the concept is to approximate the object interface instead of trying to represent it by cutting or using special elements. One of the earliest approximate domain approaches for inviscid multiphase compressible flow was proposed in [16]. Later on, other studies have been done improving this new field; we may mention the Ghost Fluid Method (GFM) that has been applied to compressible fluid flows in [1, 15], to fluid-structure-interaction in [37] and to multiphase flow in [38]. The GFM does not have any theoretical foundations, but it has been proven to be very effective for simulating practical problems.

In general, the shifted boundary method can be considered a different version of the rest of unfitted methods. Moreover, since it does not cut the background elements, the SBM method does not suffer from small cut-cell problems which is the main issue of classic embedded methods. The method is called "shifted" because it actually considers a shifted fluid-object interface instead of the correct one. The true interface between the fluid and the object is described by the embedded object shape, but SBM practically uses an approximate one. The approximate boundary is the closest shape which the background mesh can provide (see Figure 4). This kind of approximation would lead to a 1st order approximation of the geometry leading to a first-order convergence of the error in the L^2-norm if modified Dirichlet boundary conditions are not considered.

As previously stated, the body-fitted formulation imposes boundary conditions differently than

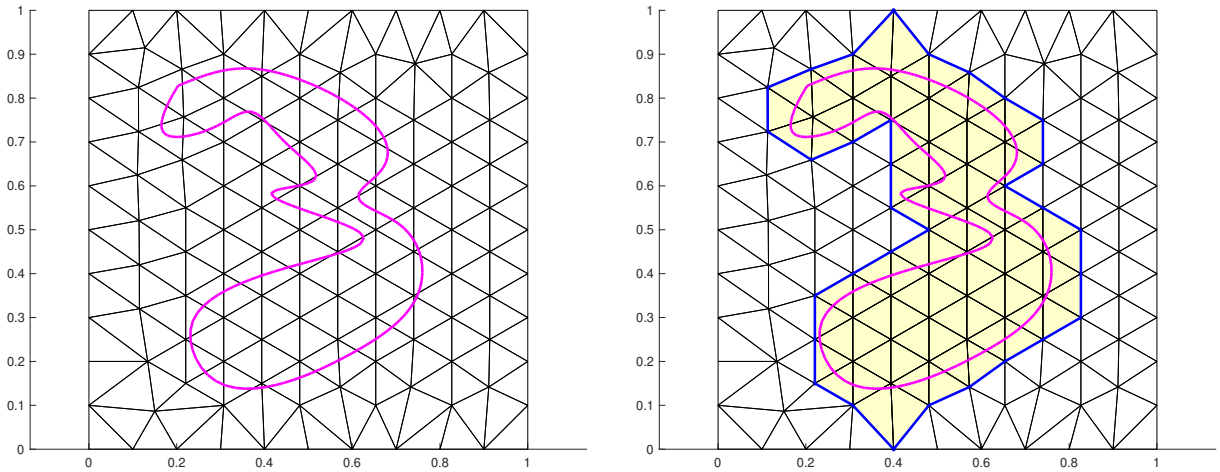


Figure 4: Embedded immersed object (pink line) in a square computational domain $[0, 0] \times [1, 1]$ in the left panel and the right one the SBM algorithm has been applied to finding the shifted boundary (in blue).

non-conforming formulations like EBM and SBM. Unfitted approaches do, in fact, seek to simulate problems with embedded objects, but doing so necessitates imposing boundary conditions outside of the exact interface between fluid and solid.

Although it appears at this time that unfitted techniques add complexity to the algorithm and that it is unclear how they maintain convergence and the appropriate convergence velocity. Since we are imposing the interface conditions in some other points, the creation of a robust solution will require the introduction of numerous extra steps, which will increase the computational cost of imposing the boundary conditions. Additionally, this may introduce an additional inaccuracy in the region of the domain that is close to the skin boundary.

On the other hand, there is a significant reduction in the computational cost associated with the initial meshing process since the background mesh is usually a structured mesh of a fluid domain. It's no anymore required to spend time adjusting the mesh around the immersed object. The only advisable thing for getting accurate results is a mesh refinement in the region where the embedded interface is.
Moreover, a re-meshing algorithm is not needed anymore because large boundary movements or large deformations only change the embedded boundary but not the background mesh. Since the background mesh and the embedded skin boundary are built independently, the movements or deformations of the object shape would simply lead to a change of the background cut elements that take into account the presence of the immersed object. Then, the particular unfitted method finds these new cut elements and imposes the embedded interface conditions differently. The family of unfitted methods is designed mainly for applications where complex immersed objects are considered, or simulations of large object displacements or deformations are required.

1.4 Objectives

The primary purpose of this master's thesis is to create an algorithm that implements the shifted boundary method (SBM) within the Kratos Multiphysics software. Kratos is an open-source software that has been developed since 2005 and is maintained also by the International Center for Numerical Methods in Engineering (CIMNE). CIMNE is a research organization created in 1987 and located at the Technical University of Catalonia (UPC).
The long-term goal is to create an SBM code for computational fluid dynamics (CFD) problems.

Particular attention has been paid to the convection-diffusion application of Kratos Multiphysics; the structure and tools of Kratos will be explained in Chapter 5.1. The application and validation of the technique within a convection-diffusion problem can be viewed as the initial major step toward the application of the SBM to CFD problems. The convection-diffusion problem has been approached in a series of smaller steps during the thesis which will be detailed in depth, both theoretically and computationally. The first part, which is shared by all applications such as convection-diffusion and CFD applications, is the creation of an algorithm for identifying the cut elements, the internal elements, and the shifted boundaries.
Then, the specifics of the modified Dirichlet boundary conditions required to maintain a second-order convergence in the L^2-norm will be clarified. Particularly at this phase of the development of the shifted boundary conditions, the details described by Professor Guglielmo Scovazzi in [6, 26, 27] will be applied.

Furthermore, a novel technique will be taken into account for the imposition of the modified Dirichlet shifted boundary conditions using the so-called Multi-Point Constraints (MPCs). The SBM method developed in Kratos Multiphysics utilising the MPCs must be progressively validated. First, it will be applied to a Poisson problem, i.e. a pure diffusion stationary problem, and then the transient case will be examined. Eventually, the convection term is considered, resulting in a stationary convection-diffusion problem and then in a transient convection-diffusion problem.

During the validations that will be presented in the results chapter, Chapter 6, the properties of the SB method, including those related to dealing with immersed objects with complex corners or external embedded borders, will be examined.
To demonstrate the true potential of an unfitted approach, the ability to simulate a moving immersed object is demonstrated. In fact, it should deal easily with large boundary movements without the need for re-meshing and without leading to stretched or distorted elements. In the conclusions, Chapter 7, a comprehensive and critical examination will be conducted to determine which objectives have been attained and which will require further development.

2 Preliminaries for the SBM

To demonstrate the well-posedness of the shifted boundary method is necessary to present the mathematical tools and ideas that are going to be used. Later, the proofs of convergence and well-posedness of Professor Scovazzi in [6, 26, 27] will be reported and explained in detail, concerning the Poisson and the convection-diffusion problem.

2.1 Mathematical notation

Consider n_d to be the number indicating the space dimension of the problem, then we may define the domain Ω as an open set in $\mathbb{R}^{n_d}$ with a Lipschitz boundary $\Gamma = \partial\Omega$. Let T_h be a shape-regular triangulation formed by a set of non-overlapping partitions or elements Ω_e of dimension n_d. We can think of triangles or quadrilaterals for $n_d = 2$ or tetrahedra or hexahedra for $n_d = 3$. The partition is defined such that:

$$\overline{\Omega} = \overline{\bigcup_{e=1}^{n_e l} \Omega_e} \tag{1}$$

Where, n_{el} is the total number of elements. Let us define the diameter of element e which is denoted by $h_e = h_e(\Omega_e)$. Where we also define the maximum elemental diameter h_e of the triangulation:

$$h = \max_{\Omega_e \in T_h} h_e \tag{2}$$

We now introduce some operations between functions that will be used later. The L^2 and the $[L^2]^{n_d}$ inner products on the interior of a portion of Ω is define as:

$$\begin{aligned} (v, w)_\omega &= \int_\omega v\ w \\ (\boldsymbol{v}, \boldsymbol{w})_\omega &= \int_\omega \boldsymbol{v} \cdot \boldsymbol{w} \end{aligned} \tag{3}$$

Where $\omega \subset \Omega$. We also define boundary functionals on portions of the boundary Γ. Given $\gamma \subset \Gamma$:

$$\begin{aligned} (v, w)_\gamma &= \int_\gamma v\ w \\ (\boldsymbol{v}, \boldsymbol{w})_\gamma &= \int_\gamma \boldsymbol{v} \cdot \boldsymbol{w} \end{aligned} \tag{4}$$

We consider the space of functions $L^2(\Psi)$ that are square integrable on the interior or exterior boundary set Ψ.

$$\|f\|_{L^2}^2 = \int_\Psi |f(x)|^2 < \infty \tag{5}$$

For simplicity, we will use the following notation for the L^2-norm:

$$||v||_{L^2(\omega)}^2 = (v, v)_\omega \tag{6}$$

We also have to introduce the Sobolev space with the usual norm:

$$||v||_{W_j^k(\Omega)} = \left(\sum_{\alpha \leq k} ||\boldsymbol{D}_x^\alpha v||_{L^j(\Omega)}^j \right)^{1/j} \tag{7}$$

Where α is a multi-index, $|\alpha|$ is the sum of the exponents in α, and D^α is the partial derivative operator of order α, $1 \leq j \leq \infty$. The Sobolev space is a mathematical space that generalizes the

concept of square-integrable functions to functions that have limited smoothness. It is a vector space of functions defined on a measurable set equipped with a norm that measures not only the size of the function but also its derivatives.
We will use the Sobolev space in the case j = 2, such that we have an extension of the L^2 space. We have that $W_2^k(\Omega) = H^k(\Omega)$ and we consider the following notation:

$$||v||_{W_2^k(\Omega)} = ||v||_{k;\Omega} \tag{8}$$

Following the same reason is possible to extend the Sobolev space in the case $j = 2$ and $k = 0$, $W_2^0(\Omega) = H^0(\Omega) = L^2(\Omega)$, i.e. using the previous notation, we can write:

$$||v||_{0;\Omega} = ||v||_{L^2(\Omega)} \tag{9}$$

2.2 Surrogate boundary

Move on to unfitted methods and investigate unfitted discretizations and their characteristics. In other words, let's consider a simple instance in which a structured or unstructured mesh is used as the computational grid and a portion of the boundary is embedded. We are going to call this mesh the background mesh. We may think of this mesh as the entire computational domain of interest. On the other hand, we have the embedded interface, which is only an interface that, for instance, represents an object immersed in the computational domain.
Unfitted approaches are different because the background mesh can be fully constructed independently from the immersed interface; this is also the method's primary advantage. Let's refer to the immersed interface as the "true boundary" Γ, which is the boundary between the immersed object and the fluid domain. Γ is also the name of the boundary of the entire domain i.e. $\partial\Omega$, actually, the boundary Γ may be fully or partially embedded. In most cases, the domain's boundary Γ is only considered embedded on a portion. For instance, the most common scenario is when the immersed boundary is embedded and the external boundary of the fluid domain is treated with a body-fitted approach.
Consider the latter scenario. Then, if the immersed object is inside the computational domain, the embedded interface Γ really crosses the background mesh elements cutting their edges.

The objective of the method is to impose the conditions concerning the interface fluid-object Γ, not on Γ but on other surrogate points while maintaining the same results as the constraints were imposed at the exact locations. Consequently, it is logical to generate a surrogate boundary using the mesh's edges and faces which are most similar to the original boundary. This approximate boundary is denoted as $\tilde{\Gamma}$. The concept of a surrogate boundary $\tilde{\Gamma}$, as opposed to the true boundary Γ, is central for the SB method. The method proposed in [26, 27] simply proposes to shift the interface boundary from the true one Γ to a surrogate one $\tilde{\Gamma}$ that is dependent on the background mesh we are considering. This fact enables us, as with all unfitted approaches, to paste the immersed object skin on top of a computational mesh without worrying about meshing the interface geometry conformingly.

Given the logic behind the shifted boundary approach, we must now provide a method for selecting the edges that form the shifted boundary $\tilde{\Gamma}$. A way to build the surrogate boundary is by finding the closest edges of the background mesh to the true boundary Γ. For instance, one can use the nearest-point projection method once the intersected edges of the background mesh with the true boundary have been found. In the implementation section, Chapter 5, we will discuss in detail our technique, which is based on the design of a level-set function that can determine which elements are cut by the true boundary and if a node is inside or outside the immersed object.

For the moment, let's consider even an easier case where Ω is the computational domain with as boundary only the external boundary. Γ is therefore the true boundary, and we would like to consider this boundary as an embedded one. Then, we consider the best internal (or external depending on the application) approximation $\tilde{\Gamma}$ formed by the background mesh that contains the surrogate domain $\tilde{\Omega}$. The left panel of Figure 5 represents the case we are considering.

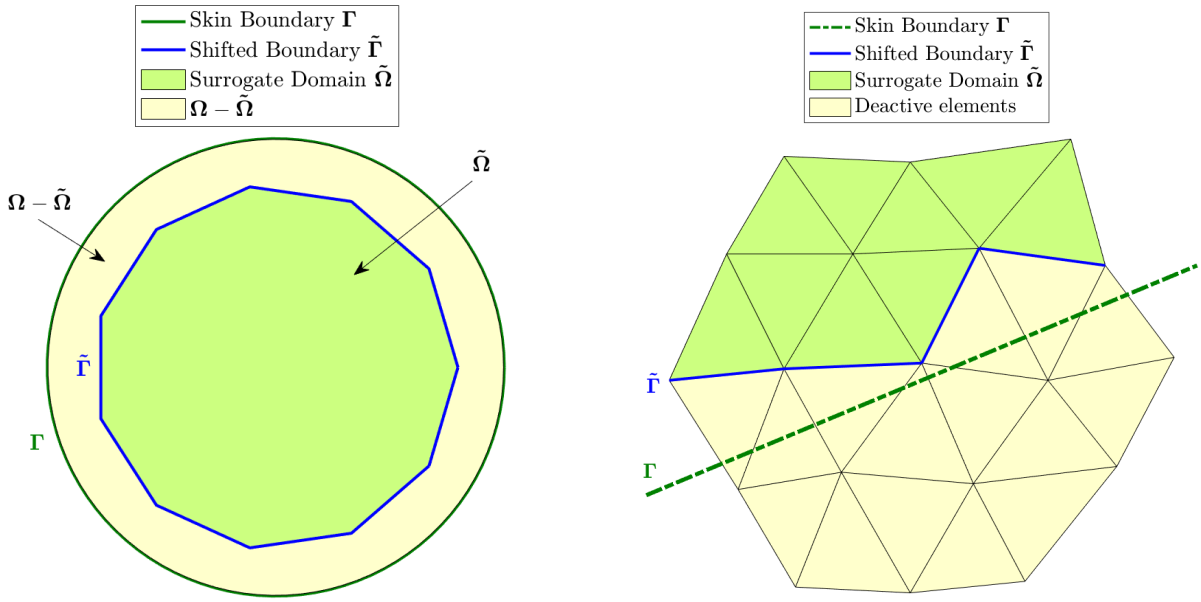


Figure 5: In the left panel, the border of Ω is considered as embedded, in green it is represented by the surrogate domain $\tilde{\Omega}$ enclosed by the surrogate boundary $\tilde{\Gamma}$. In the right panel, a zoom of a similar example is reported where the background mesh is visible.

The situation is different, but the algorithm can be generalized for two common cases: external embedded boundaries and embedded objects. Here, instead of having an immersed object, the embedded boundary is the border of the computational domain Ω.
During the theoretical analysis of the method, we need to define other entities such as the unit normal $\tilde{\boldsymbol{n}}$ that points outward to the surrogate boundary $\tilde{\Gamma}$ (see Figure 6) and the unit normal $\boldsymbol{n}$ of the true boundary Γ. We also need to define the map $\boldsymbol{M}$ that takes values $\tilde{\boldsymbol{x}} \in \tilde{\Gamma}$ and it associates the "closest-point" on the true boundary Γ:

$$\boldsymbol{M} : \tilde{\Gamma} \to \Gamma \tag{10}$$

Thus, for each point $\tilde{\boldsymbol{x}}$ on the surrogate boundary returns a point $\boldsymbol{x} \in \Gamma$ that possibly is as close as possible to $\tilde{\boldsymbol{x}}$. The choice in the implementation chapter, Chapter 5, is to use as map $\boldsymbol{M}$ the closest-point projection onto the true boundary.

Given the definition of $\boldsymbol{M}$, for instance, using the closest-point projection, we can define another useful operator that is crucial for the imposition of the boundary conditions on the shifted boundary $\tilde{\Gamma}$.

$$\boldsymbol{d}_{\boldsymbol{M}}(\tilde{\boldsymbol{x}}) = \boldsymbol{x} - \tilde{\boldsymbol{x}} = [\boldsymbol{M} - \boldsymbol{I}](\tilde{\boldsymbol{x}}) \tag{11}$$

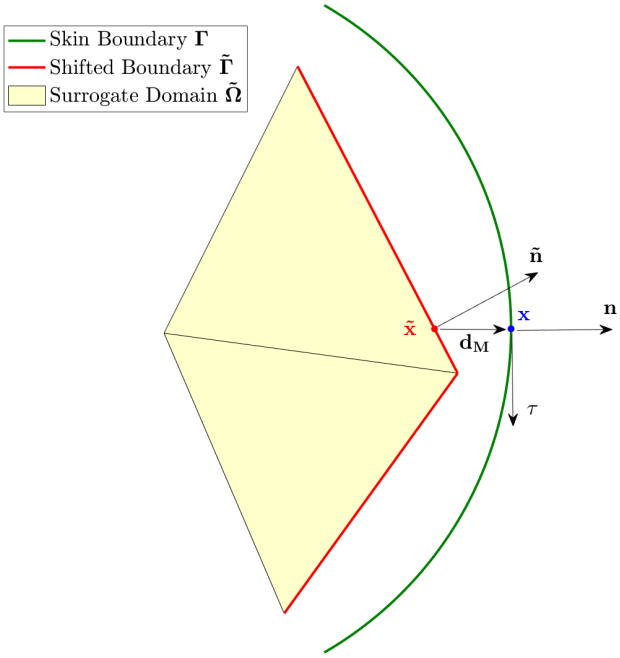


Figure 6: Graphical interpretation of the map $\boldsymbol{d_M}(\tilde{\boldsymbol{x}})$.

This map $\boldsymbol{d_M}(\tilde{\boldsymbol{x}})$ is called the "distance function" and, as $\boldsymbol{M}$, it takes as input a position vector $\tilde{\boldsymbol{x}} \in \tilde{\Gamma}$ and, differently, it returns the vector that connects $\tilde{\boldsymbol{x}}$ and its "closest-point" $\boldsymbol{x}$. This operator is used to quantify how much the condition we want to impose on the true boundary Γ is modified on $\tilde{\Gamma}$.
It is important to underline that the choice of the map $\boldsymbol{M} : \tilde{\Gamma} \to \Gamma$ does not affect the discussion we are going to do, but as said the easiest and the most robust choice is the closet-point projection. One may notice that in the case one chooses this latter option we have that the distance function $\boldsymbol{d_M}$ is parallel and aligned with the outer unit normal to the true boundary $\boldsymbol{n}$ as one can see in Figure 6.

2.3 Dirichlet surrogate boundary conditions

The key point of the imposition of the Dirichlet boundary conditions in the SB method is that we want to apply them on the shifted or surrogate boundary. Naturally, it is supposed that the Dirichlet values on the true boundary are known and correspond to the boundary conditions of the differential problem.
Following the proofs in [26, 27] for the well-posedness of the SB method for the Poisson and convection-diffusion problem in [26, 27], it is necessary to assume the following inequality:

$$\boldsymbol{n} \cdot \tilde{\boldsymbol{n}} \geq 0 \tag{12}$$

That is a condition that is always verified in practical cases, as will be shown in Chapter 6. In other words, we are asking that $\tilde{\boldsymbol{n}}$ lies on the half-plane identified by the unit normal of the true boundary $\boldsymbol{n}$ (see Figure 6).

It is possible now to define an extension of a generic function $\psi(\boldsymbol{x})$ defined on the true boundary Γ. In fact, given the definition of a map $\boldsymbol{M}$ of Equation 10, let's say the closest-point projection, the extension of $\psi(\boldsymbol{x})$ on the shifted boundary $\tilde{\Gamma}$ is the following:

$$\tilde{\psi}(\tilde{\boldsymbol{x}}) = \psi(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \tag{13}$$

For instance, we can consider as a function defined on the true boundary the normal unit vector $\boldsymbol{n}$ and the tangential one τ. Thus, we can write the following:

$$\begin{aligned} \tilde{\boldsymbol{n}}(\tilde{\boldsymbol{x}}) &= \boldsymbol{n}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \\ \tilde{\boldsymbol{\tau}}(\tilde{\boldsymbol{x}}) &= \boldsymbol{\tau}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \end{aligned} \tag{14}$$

From now on we omit the tilde over the functions defined on the surrogate boundary, i.e. instead of $\tilde{\boldsymbol{n}}(\tilde{\boldsymbol{x}})$ we will simply write $\boldsymbol{n}(\tilde{\boldsymbol{x}})$.
The same reasoning allows us to define the derivative of a generic function $\psi(\tilde{\boldsymbol{x}})$ for all $\tilde{\boldsymbol{x}} \in \tilde{\Gamma}$, defined on the shifted boundary, along the orthogonal ($\tilde{\boldsymbol{n}}$) and tangential direction ($\tilde{\boldsymbol{\tau}}$) to the shifted boundary itself $\tilde{\Gamma}$.

$$\begin{aligned} \psi_{\tilde{\boldsymbol{n}}}(\tilde{\boldsymbol{x}}) &= \nabla\psi(\tilde{\boldsymbol{x}}) \cdot \tilde{\boldsymbol{n}}(\tilde{\boldsymbol{x}}) = \nabla\psi(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{n}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \\ \psi_{\tilde{\boldsymbol{\tau}}}(\tilde{\boldsymbol{x}}) &= \nabla\psi(\tilde{\boldsymbol{x}}) \cdot \tilde{\boldsymbol{\tau}}(\tilde{\boldsymbol{x}}) = \nabla\psi(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{\tau}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \end{aligned} \tag{15}$$

After introducing the SB method's terminology, it is simple to describe its essential characteristics. Suppose we have an embedded boundary where Dirichlet boundary conditions are required. The simplest scenario that one can consider is a CFD simulation in which the fluid flow hits an immersed body. In this case, the true boundary Γ is the real interface between fluid and solid. In addition, imagine that the physical condition that must be enforced is a no-slip condition, i.e. it is required that the unknowns x- and y-components of the fluid velocity are equal to zero in the nodes along the true boundary.
Consider now that we want to use a shifted boundary approach; thus, we obtain a surrogate boundary that approximates the actual boundary by using the edges of the background fluid mesh which are closest to it. In this instance, we are aware of the velocity value at the real boundary and would want to extend this function to the shifted boundary $\tilde{\Gamma}$.
This procedure can be extended, in general terms, for any problem with scalar unknown $u(\boldsymbol{x})$; suppose as in the example to know the value of u on a portion of the true boundary, $\Gamma_D \subseteq \Gamma$, from Equation 13 we can build its extension on the corresponding portion of surrogate boundary $\tilde{\Gamma}_D$:

$$\tilde{u}_D(\tilde{\boldsymbol{x}}) = u_D(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \tag{16}$$

And we can also write the derivatives of $\tilde{u}_D(\tilde{\boldsymbol{x}})$ along the directions $\tilde{\boldsymbol{n}}$ and $\tilde{\boldsymbol{\tau}}$ as done before:

$$\begin{aligned} \tilde{u}_{D,\tilde{\boldsymbol{n}}}(\tilde{\boldsymbol{x}}) &= \nabla\tilde{u}_D(\tilde{\boldsymbol{x}}) \cdot \tilde{\boldsymbol{n}}(\tilde{\boldsymbol{x}}) = \nabla\tilde{u}_D(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{n}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \\ \tilde{u}_{D,\tilde{\boldsymbol{\tau}}}(\tilde{\boldsymbol{x}}) &= \nabla\tilde{u}_D(\tilde{\boldsymbol{x}}) \cdot \tilde{\boldsymbol{\tau}}(\tilde{\boldsymbol{x}}) = \nabla\tilde{u}_D(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{\tau}(\boldsymbol{M}(\tilde{\boldsymbol{x}})) \end{aligned} \tag{17}$$

In particular, it may be helpful to Taylor expand the function $u(\boldsymbol{x})$ around the point $\tilde{\boldsymbol{x}}$ along the normal direction to the true boundary $\boldsymbol{n}$, that is also along the direction of the vector $\boldsymbol{d_M}(\tilde{\boldsymbol{x}})$ defined in Equation 11.

$$\begin{aligned} u(\tilde{\boldsymbol{x}}) &= u(\boldsymbol{x}) + \nabla u(\tilde{\boldsymbol{x}}) \cdot (\tilde{\boldsymbol{x}} - \boldsymbol{x}) + O\Big(||\tilde{\boldsymbol{x}} - \boldsymbol{x}||^2\Big) \\ &= u_D(\boldsymbol{x}) + \nabla u(\tilde{\boldsymbol{x}}) \cdot (\tilde{\boldsymbol{x}} - \boldsymbol{M}(\tilde{\boldsymbol{x}})) + O\Big(||\tilde{\boldsymbol{x}} - \boldsymbol{M}(\tilde{\boldsymbol{x}})||^2\Big) \\ &= u_D(\boldsymbol{x}) - \nabla u(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{d_M}(\tilde{\boldsymbol{x}}) + O\Big(||\boldsymbol{d_M}(\tilde{\boldsymbol{x}})||^2\Big) \end{aligned} \tag{18}$$

Where we have substitute the known Dirichlet value at $\boldsymbol{x} \in \Gamma$, i.e. that $u(\boldsymbol{x}) = u_D(\boldsymbol{x})$ and we have used the definitions in Equation 10 and 11. Equation 18 contains the key idea behind the shifted boundary method. If we can introduce in some ways the information of Equation 18 in our FEM linear system, we are passing the information that near $\tilde{\Gamma}$ there is an embedded boundary

where some Dirichlet boundary conditions are present. But, actually, Equation 18 allows us to provide equivalent information to the system imposing condition not on the true boundary Γ, but on the shifted boundary $\tilde{\Gamma}$. In the finite element context, we neglect the second-order term $O(||\boldsymbol{d_M}(\tilde{\boldsymbol{x}})||^2)$ and we consider a second-order accurate imposition of the boundary conditions at the shifted boundary:

$$u(\tilde{\boldsymbol{x}}) \simeq u_D(\boldsymbol{x}) - \nabla u(\tilde{\boldsymbol{x}}) \cdot \boldsymbol{d_M}(\tilde{\boldsymbol{x}}) \tag{19}$$

The imposition of the Dirichlet boundary conditions using Equation 19 is suitable in the case of finite element methods with linear basis functions since the convergence rate is second-order and, therefore, once we introduce these "surrogate" or modified Dirichlet boundary conditions we still expect a second-order convergence of the error in the L^2-norm. Throughout Chapter 5, the specifics of how to compute the terms of Equation 19 will be covered in depth.

Chapter 3 and Chapter 4 are dedicated to the mathematical foundations of the SB method for the Poisson and convection-diffusion problems, respectively.

3 Poisson problem

3.1 Nitsche variational forms of the Poisson problem

The Poisson problem with non-homogeneous Dirichlet boundary conditions is presented in this chapter. In particular, the well-posedness of the SBM is shown. Later, in this context, we will also handle the convection-diffusion problem with the same purpose. The system of equation for a Poisson problem with only Dirichlet boundary condition on $\Gamma = \partial\Omega$ reads as follows:

Find $u \in C^2(\Omega)$ such that:

$$\begin{aligned} -\Delta u &= f \qquad \text{on } \Omega \\ u &= u_D \qquad \text{on } \Gamma_D = \partial\Omega \end{aligned} \tag{20}$$

At this point, we follow the Nitsche weak formulation [17]. We first introduce the discrete space $V^h(\Omega) \subset H^1(\Omega)$ for the approximate solution u^h and the discrete space $V^h_{0;D}(\Omega) \subset H^1_{0;D}(\Omega)$ for the space of test function w^h. Later on, the discrete spaces V^h and $V^h_{0;D}$ will be simply the space of continuous piecewise polynomial functions.

In the case of a standard body-fitted approach, i.e. using a conformal grid, the boundary conditions can be imposed using the Nitsche consistent weak formulation. Let's first integrate over Ω on both sides of the equation and consider the space of test functions $w \in H^1_{0;D}(\Omega)$; doing it, we get the weak formulation that requires less regularity on u and allows us to deeply analyse the FEM discretization that we are going to introduce. Using the notation defined in Chapter 2.1, the weak formulation read as follows:

Find $u \in H^1(\Omega)$ such that $\forall w \in H^1_{0;D}(\Omega)$:

$$(\nabla u, \nabla w)_\Omega = (f, w)_\Omega \tag{21}$$

Where $H^1_{0;D}(\Omega)$ is the space of functions in $H^1(\Omega)$. The subscript "0;D" indicates that the space consists of functions having a zero trace on the boundary Γ_D of the domain Ω. The trace of a function is the restriction of the function to the boundary of the domain. By requiring the trace to be zero, we are imposing Dirichlet boundary conditions on the functions in the space. In summary, the space $H^1_{0;D}(\Omega)$ consists of functions in $H^1(\Omega)$ that have zero values on the boundary Γ_D, and it is helpful using test functions that do not affect the Dirichlet boundary conditions that u has to satisfy.
At this point, we follow the Nitsche weak formulation [17]. We first introduce the discrete space $V^h(\Omega) \subset H^1(\Omega)$ for the approximate solution u^h and the discrete space $V^h_{0;D}(\Omega) \subset H^1_{0;D}(\Omega)$ for the space of test functions w^h's. Later on, the discrete spaces V^h and $V^h_{0;D}$ will be simply the space of continuous piecewise polynomial functions.

In addition, Nitsche presented a consistent weak formulation in which Dirichlet boundary requirements are enforced weakly. The formulation is the following:

Find $u^h \in V^h(\Omega)$ such that $\forall w^h \in V^h_{0;D}(\Omega)$:

$$(\nabla u^h, \nabla w^h)_\Omega - \langle w^h, \nabla u^h \cdot n\rangle_{\Gamma_D} - \langle \nabla w^h \cdot n, u^h - u_D\rangle_{\Gamma_D} + \langle \alpha/h^T w^h, u^h - u_D\rangle_{\Gamma_D} = (f, w^h)_\Omega \tag{22}$$

Where α represents a penalty parameter and h^T represents the characteristic length of the discretization. h^T commonly is selected as the characteristic length of the elements in the direction orthogonal to the boundary in calculations. In the limit as $h^T \to 0$ Nitsche, in his famous paper [33], demonstrated that the solution of the weak form of equation (22) converges to the solution of equation (21).

Consider the Euler-Lagrange equations associated with equation (22), which can be obtained using integration by parts if u^h is substituted with the exact solution $u \in H^2(\Omega)$. It clarifies the nature of the Nitsche approach. Substituting and applying integration by parts backwards to the first two terms, we get:

$$-(w^h, \Delta u^h)_\Omega - \langle \nabla w^h \cdot n - \alpha/h^T w^h, u^h - u_D \rangle_{\Gamma_D} = (f, w^h)_\Omega$$

And rearranging the terms:

$$-(w^h, \Delta u + f)_\Omega - \langle \nabla w^h \cdot n - \alpha/h^T w^h, u^h - u_D \rangle_{\Gamma_D} = 0 \qquad \forall w^h \in V^h(\Omega) \tag{23}$$

Hence, the Nitsche method enforces weakly both the partial differential equation on Ω indeed, the first term of the previous equation is zero when u satisfies the differential equation in 20, and the boundary condition $u = u_D$ on Γ_D due to the second term. The Nitsche weak formulation we have stated is a numerical methodology for enforcing Dirichlet boundary conditions in finite element methods. It is a reformulation of the standard weak formulation that permits the introduction of boundary conditions more naturally and robustly. The conventional weak formulation necessitates the insertion of boundary conditions using Lagrange multipliers. On the other hand, the Nitsche weak formulation imposes the boundary conditions directly within the formulation, eliminating the requirement for Lagrange multipliers.
The essential concept underlying the Nitsche weak formulation is the addition of penalty terms that indicate the boundary conditions. These penalty terms are usually proportionate to the solution's deviation from the specified boundary values and are weighted by a penalty parameter α. The penalty parameter sets the severity of the penalty term and ensures that the boundary requirements are precisely satisfied.

The weak formulation in Equation 22 is classical for a standard body-fitted approach, that is, when the Dirichlet boundary conditions are imposed on the true boundary Γ_D. In the next chapter, the SB extension of the Posson problem is considered.

3.2 Nitsche variational form for the SB method

However, while working with an unfitted formulation, we are confronted with two major challenges that are not present when working with a mathematical model resulting from a body-fitted approach:

- If there are any tiny cut components in the system, Nitsche's approach may become numerically unstable, and the resulting algebraic system may have low condition numbers.
- It is computationally costly and complex to integrate the Nitsche variational form on cut components.

In the context of unfitted methods, where the Γ boundary is not well-defined, the Nitsche variational formulation is not recommended. Instead, the shifted boundary method is a potential strategy that preserves all of the general advantages of unfitted methods, is unaffected by small cut elements, and is simple to implement.

The SB technique is a novel unfitted finite element formulation in which the boundary conditions are applied to the surrogate boundary $\tilde{\Gamma}$ instead of the true boundary Γ. The map $\boldsymbol{M}$ established in Equation 10 and the Taylor expansion described in Equation 18 are important in imposing shifted boundary conditions on $\tilde{\Gamma}$ that, as seen before, they are second-order accurate in imposing the boundary conditions.

Let us introduce the usual discrete space of piecewise-linear continuous functions on the surrogate domain $\tilde{\Omega}$:

$$V^h(\tilde{\Omega}) = \left\{ w^h : w^h \in C^0(\tilde{\Omega}) \cap P^1\left(\Omega_e\right) \right\} \tag{24}$$

Then, we can state the Nitsche variational form for the SB method. Where to keep everything as simple as possible, we consider that all boundaries of Ω are embedded and that the Dirichlet boundary conditions are known. Notice that the following formulations can be easily generalized also for problems with mixed boundaries (embedded and body-fitted).

Find $u^h \in V^h(\tilde{\Omega})$ such that $\forall w^h \in V^h_{0;D}(\tilde{\Omega})$:

$$\begin{aligned}(\nabla w^h, \nabla u^h)_{\tilde{\Omega}} - \langle w^h, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d} - \bar{u}_D \rangle_{\tilde{\Gamma}_D} \\ + \langle \alpha / h^{\perp} (w^h + \nabla w^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d} - \bar{u}_D \rangle_{\tilde{\Gamma}_D} + \langle \beta h^{\perp} w^h_{,\bar{\tau}_i}, u^h_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i} \rangle_{\tilde{\Gamma}_D} = (w^h, f)_{\tilde{\Omega}}\end{aligned} \tag{25}$$

Note that in the SB method, the computational domain and its boundary are not anymore the true ones (Ω and Γ). All the integrations are made on the surrogate domain $\tilde{\Omega}$ and on the surrogate boundary $\tilde{\Gamma}$ that, in this case, is all a Dirichlet boundary $\tilde{\Gamma}_D$. In addition, also the normal $\boldsymbol{n}$ to Γ becomes now $\tilde{\boldsymbol{n}}$, that is the normal of the surrogate boundary.
Moreover, one may also notice that the term u^h is replaced by $u^h + \nabla u^h \cdot \boldsymbol{d}$, this is because the integration of the boundary terms are on $\tilde{\Gamma}_D$, where we don't want to impose simply $u^h - \bar{u}_D = 0$. But, for having a second-order convergent scheme, we also consider the first term of the Taylor expansion. Same reason for w^h.
Finally, it has been introduced an additional term that was not present in the body-fitted Nitsche formulation of Equation 22 is the following:

$$\langle \beta h^{\perp} w^h_{,\bar{\tau}_i}, u^h_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i} \rangle_{\tilde{\Gamma}_D} \tag{26}$$

This boundary term, as shown in [6], which described the second generation of SBM, is not fundamental for proving that the problem is well-posed, but in this context, it helps. This additional term is essentially enforcing the matching condition between the tangential derivative of u^h and $\bar{u}_D$ on the surrogate boundary $\tilde{\Gamma}_D$. i.e. it enforces $u^h_{,\bar{\tau}_i} = \bar{u}_{D,\bar{\tau}_i}$ on $\tilde{\Gamma}_D$. It introduces only a second-order consistency error of the same order as the method we are considering since we have taken the space V^h of piecewise-linear continuous functions.

To better understand what the Nitsche variational form of Equation 25 is imposing weakly, we can substitute the exact solution u, instead of u^h, and apply integration by parts backwards, obtaining:

$$-(w^h, \Delta u + f)_{\tilde{\Omega}} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}} - \alpha / h^{\perp} (w^h + \nabla w^h \cdot \boldsymbol{d}), u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D \rangle_{\tilde{\Gamma}_D} + \langle \beta h^{\perp} w^h_{,\bar{\tau}_i}, u_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i} \rangle_{\tilde{\Gamma}_D} = 0 \tag{27}$$

The first term enforces the differential equation on the surrogate domain $\tilde{\Omega}$, the second term weakly imposes the surrogate Dirichlet boundary conditions at the shifted boundary $\tilde{\Gamma}$, and the third one enforces the tangential derivatives of u and of $\bar{u}_D$ to be equal on $\tilde{\Gamma}$.

Let us also write in the classical form the variational or weak formulation using the bilinear form $a^h(\cdot,\cdot)$ and the linear form $l^h(\cdot)$.

Find $u^h \in V^h(\tilde{\Omega})$ such that $\forall w^h \in V^h(\tilde{\Omega})$:

$$a_u^h(u^h, w^h) = l_u^h(w^h) \tag{28}$$

Where the linear and bilinear forms are defined in the following:

$$\begin{aligned} a_u^h(u^h, w^h) =& (\nabla w^h, \nabla u^h)_{\tilde{\Omega}} - \langle w^h + \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}}_{\tilde{\Gamma}_D} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\Gamma} + \langle \alpha / h^{\perp} (w^h + \nabla w^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} + \langle \beta h^{\perp} w^h_{,\bar{\tau}_i}, u^h_{,} \rangle_{\tilde{\Gamma}_D} \\ l_u^h(w^h) =& (w^h, f)_{\tilde{\Omega}} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, \bar{u}_D \rangle_{\tilde{\Gamma}_D} + \langle \alpha / h^{\perp} (w^h + \nabla w^h \cdot \boldsymbol{d}), \bar{u}_D \rangle_{\tilde{\Gamma}_D} + \langle \beta h^{\perp} w^h_{,,}, \bar{u}_{D,\bar{\tau}_i} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{29}$$

One may notice that the bilinear form $a_u^h(u^h, w^h)$ is not symmetric. For this reason, it has the subscript u; in fact, a single term is responsible for the asymmetry, that is:

$$\langle \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\Gamma_D} \tag{30}$$

A symmetric weak formulation is generally preferred in the development of a finite element method because it leads to several important benefits, including improved numerical stability, accuracy, and efficiency. For this reason, in the next paragraph, we will introduce an additional consistent term that allows us to have a symmetric bilinear form.

3.3 Nitsche symmetric variational form for the SB method

Let's try to obtain a symmetric Nitsche formulation. To do so, we need to modify the term of Equation 30 and make it symmetric. If one considers the closest-point projection as the map $\boldsymbol{M}$, then we have seen before that the distance vector $\boldsymbol{d}_{\boldsymbol{M}} = \boldsymbol{d}$ can be decomposed as $\boldsymbol{d} = ||\boldsymbol{d}||\boldsymbol{n}$ (see Figure 6). Let us now consider the following decomposition:

$$\nabla u^h \cdot \tilde{\boldsymbol{n}} = ((\nabla u^h \cdot \boldsymbol{n})\boldsymbol{n} + (\nabla u^h \cdot \boldsymbol{\tau}_i)\boldsymbol{\tau}_i) \cdot \tilde{\boldsymbol{n}} \tag{31}$$

Then let's substitute the term $\nabla u^h \cdot \boldsymbol{n}$ by the following:

$$\nabla u^h \cdot \boldsymbol{n} = \nabla u^h \cdot \frac{\boldsymbol{d}}{||\boldsymbol{d}||} \tag{32}$$

Moreover, let's introduce a first-order approximation on $\tilde{\Gamma}_D$.

$$\nabla u^h\big|_{\tilde{\Gamma}_D} \cdot \boldsymbol{\tau}_i \approx \nabla \bar{u}_D \cdot \boldsymbol{\tau}_i \tag{33}$$

Then the asymmetric term of Equation 30 becomes:

$$\langle \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\Gamma_D} \approx \langle \nabla w^h \cdot \boldsymbol{d}, \frac{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})}{\|\boldsymbol{d}\|} \nabla u^h \cdot \boldsymbol{d} + (\nabla \bar{u}_D \cdot \boldsymbol{\tau}_i)\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}} \rangle_{\Gamma_D} \tag{34}$$

It may be split into two terms. The first depends on the unknown u^h and is symmetric, and the second term depends only on the boundary terms and will be moved to the left-hand side into the linear form l_s^h.

Therefore, thanks to the approximation in Equation 33, we can now write a symmetric Nitsche variational formulation that is still second-order accurate because the approximation we have done is first-order on the boundary $\tilde{\Gamma}_D$.

Let us also write the variational or weak formulation in the classical form using the bilinear form $a^h(\cdot,\cdot)$ and the linear form $l^h(\cdot)$.

Find $u^h \in V^h(\tilde{\Omega})$ such that $\forall w^h \in V^h(\tilde{\Omega})$:

$$a_s^h(u^h, w^h) = l_s^h(w^h) \tag{35}$$

Where $a_s^h(\cdot,\cdot)$ and $l_s^h(\cdot,\cdot)$ are the bilinear and linear form of the symmetric variation formulation and are defined as:

$$\begin{aligned} a_s^h(u^h, w^h) =& (\nabla w^h, \nabla u^h)_{\tilde{\Omega}} - \langle w^h + \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \nabla w^h \cdot \boldsymbol{d}, (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})/\|\boldsymbol{d}\| \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} + \langle \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ l_s^h(w^h) =& (w^h, f)_{\tilde{\Omega}} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, \bar{u}_D \rangle_{\tilde{\Gamma}_D} - \langle \nabla w^h \cdot \boldsymbol{d}, (\nabla \bar{u}_D \cdot \boldsymbol{\tau}_i)\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), \bar{u}_D \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{36}$$

One may note that we have removed the additional term involving the tangential derivative of u^h and $\bar{u}_D$ that we have introduced in the unsymmetric variational form 28 to simplify the proof of well-posedness. In the symmetric case is not needed and therefore has been omitted.

Once again, one can check that the terms introduced and the SB approximation of the Dirichlet boundary conditions keep the formulation second-order accurate by replacing the exact solution u, in fact:

$$\begin{aligned} -(w^h, \Delta u + f)_{\tilde{\Omega}} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}} - \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D \rangle_{\tilde{\Gamma}_D} \\ - \langle \underbrace{\nabla w^h \cdot \boldsymbol{d}}_{\|\boldsymbol{d}\| \nabla w^h \cdot \boldsymbol{n}}, \nabla u \cdot \tilde{\boldsymbol{n}} - \underbrace{\frac{\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}}{\|\boldsymbol{d}\|} \nabla u \cdot \boldsymbol{d}}_{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \nabla u \cdot \boldsymbol{n}} - (\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}}) \nabla \bar{u}_D \cdot \boldsymbol{\tau}_i \rangle_{\tilde{\Gamma}_D} = 0 \end{aligned} \tag{37}$$

Where we can see the weakly enforcement of the differential equation in $\tilde{\Omega}$ (first term), and we have used the fact that $\boldsymbol{d} = ||\boldsymbol{d}||\boldsymbol{n}$.

$$\begin{aligned} -(w^h, \Delta u + f)_{\tilde{\Omega}} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}} - \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), \overbrace{u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D}^{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ \underbrace{- \langle \underbrace{\|\boldsymbol{d}\|}_{O(h)} \nabla w^h \cdot \boldsymbol{n}, \underbrace{\nabla u \cdot \tilde{\boldsymbol{n}} - (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \nabla u \cdot \boldsymbol{n} - (\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}}) \nabla \bar{u}_D \cdot \boldsymbol{\tau}_i}_{O(h)} \rangle_{\tilde{\Gamma}_D}}_{O(h^2)} = 0 \end{aligned} \tag{38}$$

In the final equation, both the error due to the shifted Dirichlet boundary conditions on the shifted boundary $\tilde{\Gamma}_D$ and the error introduced to make symmetric the bilinear form $a^h(\cdot,\cdot)$ are depicted. Since both errors are second-order, they do not affect the convergence velocity of the original FEM scheme.

3.4 Stability of the symmetric formulation

At this point, it is necessary to examine the stability of the Nitsche variational formulation described in section 3.3. A problem is well-posed, according to Hadamard [35], if it fulfils the

requirements of existence, uniqueness, and stability. These conditions are necessary to guarantee that the problem is well-defined and has a unique, stable solution. A problem is said to be well-posed if it is mathematically well-defined, has a unique solution that is stable and constantly dependent on the input data, and has a unique solution.
Therefore it is sufficient to show that the symmetric form satisfies the hypothesis of Lax-Milgram's theorem [35, 39]. In fact, it provides a sufficient condition for the existence and uniqueness of a solution to a linear boundary value problem. One of the main hypotheses of Lax-Milgram's theorem is that the bilinear form is coercive and bounded.

3.4.1 Coercivity of $a_s^h(\cdot,\cdot)$

Coercivity of the bilinear form $a_s^h(\cdot,\cdot)$ requires that there exists a real number $C_{\text{SB}} > 0$ such that:

$$a_s^h\left(u^h, u^h\right) \geq C_{\text{SB}} \left|\left|\left|u^h\right|\right|\right|_{\text{SB}}^2 \tag{39}$$

Where, the $|||\cdot|||$ is a norm of the underlying Hilbert space V^h that we are considering after having imposed the shifted boundary approximation. We will see soon which is a suitable definition for this norm.

From the definition of the symmetric bilinear form given in Equation 36, one can write the following equality:

$$\begin{aligned} a_s^h(u^h,u^h) =& (\nabla u^h, \nabla u^h)_{\tilde{\Omega}} - \langle u^h + \nabla u^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} - \langle \nabla u^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d}\rangle_{\tilde{\Gamma}_D} \\ &+ \langle \nabla u^h \cdot \boldsymbol{d}, (\boldsymbol{n}\cdot\tilde{\boldsymbol{n}})/\|\boldsymbol{d}\| \nabla u^h \cdot \boldsymbol{d}\rangle_{\tilde{\Gamma}_D} + \langle \alpha/h^{\perp}(u^h + \nabla u^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d}\rangle_{\tilde{\Gamma}_D} \\ =& \|\nabla u^h\|_{0,\tilde{\Omega}}^2 - 2\langle u^h + \nabla u^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} + \alpha \|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0,\tilde{\Gamma}_D}^2 \\ &+ \|\sqrt{(\boldsymbol{n}\cdot\tilde{\boldsymbol{n}})/\|\boldsymbol{d}\|}\nabla u^h \cdot \boldsymbol{d}\|_{0,\tilde{\Gamma}_D}^2 \end{aligned} \tag{40}$$

Note that we are applying condition 12 when we take the square root in the last term.
We introduce the ϵ-inequality that comes from the combination of Young's inequality and Cauchy-Schwartz inequality: for any $\epsilon \in \mathbb{R}^+$:

$$\langle v, w\rangle \leq \|v\|\|w\| \leq \frac{1}{2}\left(\epsilon\|v\|^2 + \frac{1}{\epsilon}\|w\|^2\right) \tag{41}$$

Thanks to the ϵ-inequality we can bound the second term in Equation 40 with the following:

$$2\langle u^h + \nabla u^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} \leq \frac{1}{\epsilon}\|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0,\tilde{\Gamma}_D}^2 + \epsilon\|\sqrt{h^{\perp}}\nabla u^h \cdot \tilde{\boldsymbol{n}}\|_{0,\tilde{\Gamma}_D}^2 \tag{42}$$

Where $\epsilon \in \mathbb{R}^+$ is an arbitrary number. Now we need to state some inequalities that have been proven in [4].

For any element $\Omega_e \subset \tilde{\Omega}$ with an edge $\gamma_e \subset \tilde{\Gamma}$ and any unit vector $\boldsymbol{v}$ such that $\boldsymbol{v}\cdot\tilde{\boldsymbol{n}} > 0$:

$$\begin{aligned} \left\|\sqrt{h^{\perp}}w\right\|_{0,\gamma_e}^2 &\leq C_I\left(\|w\|_{0,\Omega_e}^2 + h^2|w|_{1,\Omega_e}^2\right), \quad \forall w \in H^1(\Omega_e), \\ \left\|\sqrt{h^{\perp}}\nabla w \cdot \boldsymbol{v}\right\|_{0,\tilde{\Gamma}}^2 &\leq C_I\left(|w|_{1,\Omega_e}^2 + h^2|w|_{2,\Omega_e}^2\right), \quad \forall w \in H^2(\Omega_e), \end{aligned} \tag{43}$$

Where $|w|_{1,\Omega_e}^2$ and $|w|_{2,\Omega_e}^2$ are the H^1 - and H^2-seminorms, respectively. Summing over all the elements with at least one of their edges on the boundary $\tilde{\Gamma}$, we obtain:

$$\begin{aligned} \left\|\sqrt{h^{\perp}}w\right\|_{0,\tilde{\Gamma}}^2 &\leq C_I\left(\|w\|_{0,T(\tilde{\Omega})}^2 + h^2|w|_{1,T(\tilde{\Omega})}^2\right), \quad \forall w \in H^1(T(\tilde{\Omega})), \\ \left\|\sqrt{h^{\perp}}\nabla w \cdot \boldsymbol{v}\right\|_{0,\tilde{\Gamma}}^2 &\leq C_I\left(|w|_{1,T(\tilde{\Omega})}^2 + h^2|w|_{2,T(\tilde{\Omega})}^2\right), \quad \forall w \in H^2(T(\tilde{\Omega})). \end{aligned} \tag{44}$$

For a function, $w^h \in V^h(\tilde{\Omega})$, that is a piecewise linear and globally continuous function over the mesh $T(\tilde{\Omega})$, inequality (26b) reduces to:

$$\left\| \sqrt{h^{\perp}} \nabla w \cdot \boldsymbol{v} \right\|_{0,\tilde{\Gamma}}^2 \leq C_I \|\nabla w\|_{0,\tilde{\Omega}}^2. \tag{45}$$

Using disequality 45 considering $\boldsymbol{v} = \tilde{\boldsymbol{n}}$ on the second term of Equation 42 we have that:

$$\epsilon \|\sqrt{h^{\perp}} \nabla u^h \cdot \tilde{\boldsymbol{n}}\|_{0,\tilde{\Gamma}_D}^2 \leq \epsilon C_I \|\nabla u^h\|_{0,\tilde{\Omega}}^2 \tag{46}$$

Rearranging Equation 40 with the latter results and collecting some terms, we get the following:

$$a_s^h(u^h, u^h) \geq (1 - \epsilon C_I)\|\nabla u^h\|_{0,\tilde{\Omega}}^2 + (\alpha - \frac{1}{\epsilon})\|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0,\tilde{\Gamma}_D}^2 + \|\sqrt{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})/\|\boldsymbol{d}\|}\nabla u^h \cdot \boldsymbol{d}\|_{0,\tilde{\Gamma}_D}^2 \tag{47}$$

The last term can be written more suitably so that we can define an appropriate norm of $V^h(\tilde{\Omega})$:

$$\sqrt{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})/\|\boldsymbol{d}\|}\nabla u^h \cdot \boldsymbol{d} = \sqrt{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})\|\boldsymbol{d}\|}\nabla u^h \cdot \boldsymbol{n} \tag{48}$$

Then the following choice for the norm of $V^h(\tilde{\Omega})$ is possible:

$$\left|\left|\left| u^h \right|\right|\right|_{\mathrm{SB}}^2 = \left\| \nabla u^h \right\|_{0,\tilde{\Omega}}^2 + \left\| \sqrt{1/h^{\perp}} \left(u^h + \nabla u^h \cdot \boldsymbol{d} \right) \right\|_{0,\tilde{\Gamma}_D}^2 + \left\| \sqrt{(\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})\|\boldsymbol{d}\|} \nabla u^h \cdot \boldsymbol{n} \right\|_{0,\tilde{\Gamma}_D}^2 \tag{49}$$

Therefore, we can conclude that the coercivity property is achieved when the following inequalities hold:

$$\begin{aligned} 1 - \epsilon C_I &> 0 \\ \alpha - \frac{1}{\epsilon} &> 0 \end{aligned} \tag{50}$$

And thus the positive constant C_{SB} can be defined as the minimum between $1 - \epsilon C_I$ and $\alpha - 1/\epsilon$:

$$C_{\mathrm{SB}} = \min\left(1 - \epsilon C_I, \alpha - \frac{1}{\epsilon}\right) \tag{51}$$

Since all the constants were arbitrary, we conclude that is always possible to find a positive constant C_{SB} such that:

$$a_s^h\left(u^h, u^h\right) \geq C_{\mathrm{SB}} \left|\left|\left| u^h \right|\right|\right|_{\mathrm{SB}}^2$$

That proves the coercivity of the symmetric bilinear form $a_s^h(\cdot, \cdot)$.

3.4.2 Continuity of $a_s^h(\cdot, \cdot)$

The continuity of the bilinear form is a fundamental property in the context of the Lax-Milgram theorem and is necessary to ensure that the weak formulation of the problem is well-posed and that the solution is unique.
A bilinear form $a_s^h(\cdot, \cdot)$ is said to be bounded or to satisfy the continuity condition if there exists a constant $C_b > 0$ such that for all $u^h, w^h \in V^h$, where V^h is a Hilbert space, the following inequality holds:

$$\left| a_s^h(u^h, w^h) \right| \leq C_b \left\| u^h \right\|_{V^h} \left\| w^h \right\|_{V^h} \tag{52}$$

Where $\|\cdot\|_{V^h}$ is a norm on the space V^h. This condition ensures that the bilinear form is well-defined and bounded on the space V^h.

With the bilinear form defined by the Equation 36, we may write:

$$\begin{aligned} a_s^h(u^h, w^h) =&(\nabla w^h, \nabla u^h)_{\tilde{\Omega}} - \langle w^h + \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \nabla w^h \cdot \boldsymbol{d}, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} + \langle \nabla w^h \cdot \boldsymbol{d}, (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})/\|\boldsymbol{d}\| \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{53}$$

Let us now recall Cauchy-Schwarz inequality:

$$(\nabla w^h, \nabla u^h)_{V^h} \leq \left\| \nabla w^h \right\|_{V^h} \left\| \nabla u^h \right\|_{V^h} \tag{54}$$

That holds for any real inner product space, such as $\langle \cdot, \cdot \rangle_{\tilde{\Gamma}_D}$ and its norm on $\|\cdot\|_{0;\tilde{\Gamma}_D}$.
Applying Cauchy-Schwarz inequality on each inner product of Equation 53 and inserting an artificial factor $h^{\perp}$ and $1/h^{\perp}$ which is used later, we get the following:

$$\begin{aligned} a_s^h(u^h, w^h) \leq&\|\nabla w^h\|_{0;\tilde{\Omega}} \|\nabla u^h\|_{0;\tilde{\Omega}} + \|\sqrt{1/h^{\perp}}(w^h + \nabla w^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \|\sqrt{h^{\perp}} \nabla u^h \cdot \tilde{\boldsymbol{n}}\|_{0;\tilde{\Gamma}_D} \\ &+ \|\sqrt{h^{\perp}} \nabla w^h \cdot \tilde{\boldsymbol{n}}\|_{0;\tilde{\Gamma}_D} \|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \\ &+ \|\sqrt{1/h^{\perp}} \nabla w^h \cdot \boldsymbol{d}\|_{0;\tilde{\Gamma}_D} \|\sqrt{h^{\perp}} \nabla u^h \cdot \tilde{\boldsymbol{n}}\|_{0;\tilde{\Gamma}_D} \\ &+ \|\sqrt{1/h^{\perp}} \nabla w^h \cdot \boldsymbol{d}\|_{0;\tilde{\Gamma}_D} \|\sqrt{1/h^{\perp}} \nabla u^h \cdot \boldsymbol{d}\|_{0;\tilde{\Gamma}_D} h^{\perp} \boldsymbol{n} \cdot \tilde{\boldsymbol{n}}/\|\boldsymbol{d}\| \\ &+ \alpha \|\sqrt{1/h^{\perp}}(w^h + \nabla w^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \end{aligned} \tag{55}$$

Note that the fourth and the fifth term, due to the construction of the SB method, can be bound as follows:

$$\begin{aligned} \left\| \sqrt{1/h^{\perp}} \nabla w^h \cdot \boldsymbol{d} \|_{0;\tilde{\Gamma}_D} \right\| &= \left\| \sqrt{1/h^{\perp}} \nabla w^h \cdot \boldsymbol{n} \|\boldsymbol{d}\| \right\|_{0;\tilde{\Gamma}_D} \\ &\leq \left\| \sqrt{1/h^{\perp}} \nabla w^h \cdot \boldsymbol{n}\ h^{\perp} \right\|_{0;\tilde{\Gamma}_D} \\ &\leq \left\| \sqrt{h^{\perp}} \nabla w^h \cdot \boldsymbol{n} \right\|_{0;\tilde{\Gamma}_D} \end{aligned} \tag{56}$$

At this point consider the lemma in Equation 45 choosing $\boldsymbol{v}$ equal to $\tilde{\boldsymbol{n}}$ or equal to $\boldsymbol{n}$, then the following inequalities hold true:

$$\begin{aligned} \|\sqrt{h^{\perp}} \nabla u^h \cdot \tilde{\boldsymbol{n}}\|_{0;\tilde{\Gamma}_D} &\leq C_I \|\nabla u^h\|_{0;\tilde{\Omega}} \\ \|\sqrt{h^{\perp}} \nabla w^h \cdot \tilde{\boldsymbol{n}}\|_{0;\tilde{\Gamma}_D} &\leq C_I \|\nabla w^h\|_{0;\tilde{\Omega}} \\ \|\sqrt{h^{\perp}} \nabla w^h \cdot \boldsymbol{n}\|_{0;\tilde{\Gamma}_D} &\leq C_I \|\nabla w^h\|_{0;\tilde{\Omega}} \end{aligned} \tag{57}$$

Introducing everything in Equation 55:

$$\begin{aligned} a_s^h(u^h, w^h) \leq&(1 + C_I^2) \|\nabla w^h\|_{0;\tilde{\Omega}} \|\nabla u^h\|_{0;\tilde{\Omega}} + C_I \|\sqrt{1/h^{\perp}}(w^h + \nabla w^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \|\nabla u^h\|_{0;\tilde{\Omega}} \\ &+ C_I \|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \|\nabla w^h\|_{0;\tilde{\Omega}} + C_I^2 \nabla w^h\|_{0;\tilde{\Omega}} \|\nabla u^h\|_{0;\tilde{\Omega}}\ (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})(h^{\perp})^2 \\ &+ \alpha \|\sqrt{1/h^{\perp}}(w^h + \nabla w^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \|\sqrt{1/h^{\perp}}(u^h + \nabla u^h \cdot \boldsymbol{d})\|_{0;\tilde{\Gamma}_D} \end{aligned} \tag{58}$$

As one may remember, to get a symmetric bilinear form, we had to add an additive artificial term from which the fourth term comes. It depends quadratically on the characteristic mesh size $h^{\perp}$ and, therefore, can be considered negligible.
Equation 58 suggests to define a new norm for the $V^h(\tilde{\Omega})$:

$$\left|\left|\left|u^h\right|\right|\right|_{\mathrm{SB}} = \left\|\nabla u^h\right\|_{0,\tilde{\Omega}} + \left\|\sqrt{1/h^{\perp}}\left(u^h + \nabla u^h \cdot \boldsymbol{d}\right)\right\|_{0,\tilde{\Gamma}_D} \tag{59}$$

Thus we can choose the continuity constant C_b as the maximum between $1+C_I^2$ and α and prove that the bilinear form $a_s^h(\cdot,\cdot)$ satisfies the continuity condition, i.e.:

$$a_s^h(u^h, w^h) \leq C_b \left|\left|\left|u^h\right|\right|\right|_{\mathrm{SB}} \left|\left|\left|w^h\right|\right|\right|_{\mathrm{SB}} \tag{60}$$

3.4.3 Well-posedness

The bilinear symmetric form $a_s^h(u^h, w^h)$ has been shown in the previous two sections to have important properties. Firstly, it has been demonstrated to be coercive, meaning that it fulfils the requirement that it grows as the magnitude of the inputs increases. It ensures that the solution to the problem will not degenerate into an unphysical state. Secondly, it has been shown that the bilinear form is continuous, meaning that its outputs will change smoothly as the inputs change. When these properties are combined with the weak formulation in equation defined 35 and equations 36, it turns out that the formulation satisfies the hypothesis of Lax-Milgram's theorem, which states that the problem is well-posed. It implies that the problem has a unique solution and that can be obtained efficiently and accurately using numerical methods.

3.4.4 Consistency

Consider the estimate of the consistency error. As is common knowledge, it is an indispensable instrument for ensuring the precision and efficacy of numerical approaches. It is a measurement of the difference between the exact solution, and the numerical solution to the problem. The SB method is asymptotically consistent, i.e. the error committed in the weak formulation is proportional to some power of the characteristic size of the mesh h. We introduce the error between the exact solution u of the differential problem and the approximate solution u^h:

$$e = u - u^h \tag{61}$$

Then, by the considerations given in Chapter 3.3, we should obtain a second-order convergence when considering linear basis functions.

$$a_s^h(u - u^h, w^h) = O(h^2) \tag{62}$$

Instead of considering the SB symmetric weak formulation of Equation 35, we take the weak formulation of the exact solution u, but on the approximate domain $\tilde{\Omega}$:

Find $u \in H^1(\tilde{\Omega})$ such that $\forall w \in H^1_{0;D}(\tilde{\Omega})$:

$$(\nabla u, \nabla w)_{\tilde{\Omega}} = (f, w)_{\tilde{\Omega}} - \langle w, \nabla u \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \tag{63}$$

It is significant to notice that the boundary term arises from the fact that the border of $\tilde{\Omega}$ is the shifted one $\tilde{\Gamma}$. The bilinear and linear form of this discrete formulation will be marked by a tilde and an apex h since they are computed on $\tilde{\Omega}$:

$$\begin{aligned} \tilde{a}^h(u, w^h) &= (\nabla w^h, \nabla u)_{\tilde{\Omega}} - \langle w^h, \nabla u \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \\ \tilde{l}^h(w^h) &= (w^h, f)_{\tilde{\Omega}} \end{aligned} \tag{64}$$

Therefore, we can state the following estimate of the error that we commit when applying the exact solution u to the symmetric weak formulation:

$$\begin{aligned} a_s^h(u,w^h) - l_s^h(w^h) =& \overbrace{\tilde{a}^h(u,w^h) - \tilde{l}^h(w^h)}^{=0} - \langle \nabla w^h \cdot \tilde{\boldsymbol{n}}, \overbrace{u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D}^{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \alpha/h^{\perp}(w^h + \nabla w^h \cdot \boldsymbol{d}), \underbrace{u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D}_{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ &- \langle \sqrt{\|\boldsymbol{d}\|} \nabla w^h \cdot \boldsymbol{n}, \underbrace{\sqrt{\|\boldsymbol{d}\|}(\nabla u \cdot \tilde{\boldsymbol{n}} - (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \nabla u \cdot \boldsymbol{n} - (\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}}) \nabla \bar{u}_D \cdot \boldsymbol{\tau}_i)}_{O(h^{3/2})} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{65}$$

Where we have used the considerations done in the Taylor expansion when we have approximated the Dirichlet boundary conditions imposing them on the shifted boundary. Thus, we obtain an estimate of the consistent error in our symmetric weak formulation:

$$a_s^h(u,w^h) - l_s^h(w^h) \leq \left|\left|\left| w^h \right|\right|\right|_{\mathrm{SB}} O\left(h^{3/2}\right) \tag{66}$$

3.4.5 Convergence of the error

So far, we have proven that the SB method is not consistent, but it is only asymptotically consistent. Therefore it becomes consistent as the characteristic size of the mesh goes to zero ($h \to 0$). For the symmetric form of the SB method, we can use a trick to obtain a convergence rate that allows us to estimate the accuracy of the SB algorithm.

Consider the following differential equation:

$$\begin{aligned} -\Delta\psi &= u - u^h \qquad \text{on } \tilde{\Omega} \\ \psi &= 0 \qquad\qquad\quad \text{on } \tilde{\Gamma} = \partial\tilde{\Omega} \end{aligned} \tag{67}$$

This differential problem is a trick known as the Nitsche-Aubin trick [5, 35], and allows us to obtain an estimate of the L^2 norm of the error $e = u - u^h$.
To establish L^2-error estimates, we must utilize elliptic regularity estimates for ψ. Thus, we assume that both Ω and $\tilde{\Omega}$ are convex, and that $u \in H^2(\Omega)$ and as a result, $u \in H^2(\tilde{\Omega})$. Since Ω is convex, it is possible to create a set of discrete mesh refinements such that $\tilde{\Omega}$ remains convex for each mesh size h. Under these conditions and elliptic regularity [5, 8] we have that for a positive constant $c > 0$:

$$|\psi|_{2,\tilde{\Omega}} \leq c \left| u - u^h \right|_{0,\tilde{\Omega}} \tag{68}$$

Additionally, ψ satisfies the weak version which is the Nitsche formulation for the SBM. $\forall w \in V(\tilde{\Omega})$:

$$0 = (\nabla w, \nabla\psi)_{\tilde{\Omega}} - \langle w, \nabla\psi \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}} - \langle \nabla w \cdot \tilde{\boldsymbol{n}}, \psi - 0 \rangle_{\tilde{\Gamma}} + \langle \alpha/h^{\perp}(w + \nabla w \cdot \boldsymbol{d}), \psi - 0 \rangle_{\tilde{\Gamma}} - (w, u - u^h)_{\tilde{\Omega}} \tag{69}$$

Where the 0's come from the fact that the Dirichlet boundary conditions of the differential problem are now homogeneous. Know consider $a_s^h(\psi, w)$ from Equation 36:

$$\begin{aligned} a_s^h(\psi,w) =& (\nabla w, \nabla\psi)_{\tilde{\Omega}} - \langle w + \nabla w \cdot \boldsymbol{d}, \nabla\psi \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} - \langle \nabla w \cdot \tilde{\boldsymbol{n}}, \psi + \nabla\psi \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \\ &+ \langle \nabla w \cdot \boldsymbol{d}, (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}})/\|\boldsymbol{d}\| \nabla\psi \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} + \langle \alpha/h^{\perp}(w + \nabla w \cdot \boldsymbol{d}), \psi + \nabla\psi \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{70}$$

Therefore, Equation 69 can be rearrange as follows:

$$\begin{aligned} 0 =& a_s^h(\psi, w) + \langle \nabla w \cdot \boldsymbol{n}, \|\boldsymbol{d}\| \nabla\psi \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} + \langle \nabla w \cdot \tilde{\boldsymbol{n}}, \|\boldsymbol{d}\| \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} \\ & - \langle \nabla w \cdot \boldsymbol{n}, (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \|\boldsymbol{d}\| \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} - \langle \alpha / h^{\perp} (w + \nabla w \cdot \boldsymbol{d}), \nabla\psi \cdot \boldsymbol{d} \rangle_{\tilde{\Gamma}_D} - (w, u - u^h)_{\tilde{\Omega}} \end{aligned} \tag{71}$$

Now the trick continues by taking as w the error $u - u^h \in V(\tilde{\Omega})$. Doing it, the last term of the previous equation becomes our L^2-error estimate and can be taken to the left-hand side:

$$\begin{aligned} \|u - u^h\|_{0;\tilde{\Omega}}^2 = & a_s^h(\psi, u - u^h) + \langle \nabla(u - u^h) \cdot \boldsymbol{n}, \|\boldsymbol{d}\| \nabla\psi \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \\ & + \langle \nabla(u - u^h) \cdot \tilde{\boldsymbol{n}} - (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \nabla(u - u^h) \cdot \boldsymbol{n}, \|\boldsymbol{d}\| \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} \\ & - \langle \alpha / h^{\perp} (u - u^h + \nabla(u - u^h) \cdot \boldsymbol{d}), \|\boldsymbol{d}\| \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{72}$$

Then, to bound one by one the terms on the right-hand side, we can rewrite them more conveniently:

$$\begin{aligned} \|u - u^h\|_{0;\tilde{\Omega}}^2 = & a_s^h(\psi, u - u^h) + \langle \sqrt{\|\boldsymbol{d}\|} \nabla(u - u^h) \cdot \boldsymbol{n}, \sqrt{\|\boldsymbol{d}\|} \nabla\psi \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \\ & + \langle \sqrt{\|\boldsymbol{d}\|} (\boldsymbol{\tau} \cdot \tilde{\boldsymbol{n}}) \nabla(u - u^h) \cdot \boldsymbol{\tau}, \sqrt{\|\boldsymbol{d}\|} \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} \\ & - \langle \alpha \sqrt{1/h^{\perp}} (u - u^h + \nabla(u - u^h) \cdot \boldsymbol{d}), \|\boldsymbol{d}\| \sqrt{1/h^{\perp}} \nabla\psi \cdot \boldsymbol{n} \rangle_{\tilde{\Gamma}_D} \end{aligned} \tag{73}$$

Consider now the Cauchy-Schwarz inequality and the trace-type inequality that can bound the gradient of ψ by the seminorm of ψ:

$$\|\nabla\psi \cdot \boldsymbol{v}\|_{0;\tilde{\Gamma}_D} \leq C_{\tilde{\Gamma}} |\psi|_{2;\tilde{\Omega}} \tag{74}$$

Where we can choose as $\boldsymbol{v}$ the unit normals $\boldsymbol{n}$ and $\tilde{\boldsymbol{n}}$. The L^2-norm squared of the error can be bound by:

$$\|u - u^h\|_{0;\tilde{\Omega}}^2 \leq a_s^h(\psi, u - u^h) + C_{\tilde{\Gamma}} |||u - u^h||_{V^h(\tilde{\Omega})} h^{1/2} |\psi|_{2;\tilde{\Omega}} \tag{75}$$

Furthermore, consider the nodal interpolate ψ_I of ψ, that are such that $\psi_I = 0$ on the boundary of $\tilde{\Omega} = \tilde{\Gamma}_D$:

$$\psi = \psi - \psi_I + \psi_I \tag{76}$$

We apply this trick to bound $a_s^h(u - u^h, \psi_I)$. Consider the following discrete trace inequalities:

$$\|\nabla\psi_I \cdot \boldsymbol{v}\|_{0;\tilde{\Gamma}_D} \leq \|\nabla(\psi_I - \psi) \cdot \boldsymbol{v}\|_{0;\tilde{\Gamma}_D} + \|\nabla\psi \cdot \boldsymbol{v}\|_{0;\tilde{\Gamma}_D} \leq (C_{\text{int}}\, h + C_{\Gamma}) |\psi|_{2;\tilde{\Omega}} \tag{77}$$

Then, we rewrite inequality 75 and apply the trick of Equation 76:

$$\|u - u^h\|_{0;\tilde{\Omega}}^2 \leq a_s^h(\psi - \psi_I, u - u^h) + a_s^h(\psi_I, u - u^h) + C_{\tilde{\Gamma}} |||u - u^h||_{V^h(\tilde{\Omega})} h^{1/2} |\psi|_{2;\tilde{\Omega}} \tag{78}$$

The first term, by the already proven continuity of the bilinear form $a_s^h(\cdot, \cdot)$ and by elliptic regularity, can be bound as follows:

$$a_s^h(\psi - \psi_I, u - u^h) \leq C_b |||u - u^h|||_{\text{SB}} |||\psi - \psi_I|||_{\text{SB}} \leq C_b C_{int} \|u - u^h\|_{V^h(\tilde{\Omega})} h |\psi|_{2;\tilde{\Omega}}. \tag{79}$$

Instead, the second term, the one that depends only on ψ_I is bound by the following:

$$\begin{aligned} a_s^h(\psi_I, u - u^h) =& a_s^h(u - u^h, \psi_I) = a_s^h(u, \psi_I) - a_s^h(u^h, \psi_I) = a_s^h(u, \psi_I) - l_s^h(\psi_I) \\ =& \langle \alpha \sqrt{1/h^{\perp}} \nabla\psi_I \cdot \boldsymbol{d}, \sqrt{1/h^{\perp}} \underbrace{(u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D)}_{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ & - \langle \sqrt{1/h} \nabla\psi_I \cdot \tilde{\boldsymbol{n}}, \sqrt{h} \underbrace{(u + \nabla u \cdot \boldsymbol{d} - \bar{u}_D)}_{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ & - \langle \sqrt{h} \nabla\psi_I \cdot \boldsymbol{n}, \sqrt{1/h} \underbrace{\|\boldsymbol{d}\| (\nabla u \cdot \tilde{\boldsymbol{n}} - (\boldsymbol{n} \cdot \tilde{\boldsymbol{n}}) \nabla u \cdot \boldsymbol{n} - (\boldsymbol{\tau}_i \cdot \tilde{\boldsymbol{n}}) \nabla \bar{u}_D \cdot \boldsymbol{\tau}_i)}_{O(h^2)} \rangle_{\tilde{\Gamma}_D} \\ \leq & C'_{int} |\psi|_{2;\tilde{\Omega}} O(h^{3/2}) \end{aligned} \tag{80}$$

Where we have used the definition of the symmetric bilinear form $a_s^h(\cdot,\cdot)$ and the consistency error estimates of Equation 66.

Finally we are ready to put all together continuing Equation 78 and inserting the last two bounds (Equation 79 and Equation 80):

$$\begin{aligned}\|u-u^h\|_{0;\tilde{\Omega}}^2 &\leq (C_b C_{int} h|||u-u^h|||_{\mathrm{SB}} + C_{\tilde{\Gamma}} h^{1/2}|||u-u^h|||_{\mathrm{SB}} + C'_{int} O(h^{3/2}))|\psi|_{2;\tilde{\Omega}} \\ &\leq C_{\mathrm{NA}} h^{3/2}\|u-u^h\|_{0;\tilde{\Omega}}\end{aligned} \tag{81}$$

Where simplifying a factor $\|u-u^h\|_{0;\tilde{\Omega}}$ on both sides, leads to an accuracy estimate of the L^2 norm of the error $u-u^h$ of the SB method with Dirichlet boundary condition.

$$\left\|u-u^h\right\|_{0;\tilde{\Omega}} \leq C_{\mathrm{NA}} h^{3/2} \tag{82}$$

It is relevant to underline that this estimate is probably not optimal. In fact, during the numerical experiments performed, we always obtained second-order convergence in the L^2-norm of the error. It suggests that there exists another way to show that actually the consistency error estimates are proportional to h^2.

4 Convection-Diffusion problem

4.1 Nitsche variational form for the convection-diffusion problem

Let us now consider a new problem that adds a complex term to the Poisson problem, which is the convection term. This step is fundamental if the final aim is to simulate CFD problems, i.e. if the equations we would like to represent using the shifted boundary method are the Navier-Stokes equations. These kinds of equations have already been considered, by Main and Scovazzi in [27], using the SB method. In this context, the convection-diffusion problem is the more complex manner in which we have developed and tested the SB method.

The following system of equations represents the strong form of the convection-diffusion problem that is going to be evaluated:

Find $u \in C^2(\Omega)$ such that:

$$\begin{aligned} &\nabla \cdot (\boldsymbol{a}u - \kappa \nabla u) = f && \text{on } \Omega \\ &u = u_D && \text{on } \Gamma_D \\ &-(\boldsymbol{a}u - \kappa \nabla u) \cdot \boldsymbol{n} = t_N && \text{on } \Gamma_N^- \\ &\kappa \nabla u \cdot \boldsymbol{n} = t_N && \text{on } \Gamma_N^+ \end{aligned} \tag{83}$$

Where, as we have considered in the Poisson problem, Ω is the computational domain, $\Gamma = \partial\Omega$ is the boundary of Ω, and in this case, is composed of different types of boundaries. There is a portion Γ_D where are required Dirichlet boundary conditions and a portion Γ_N where Neumann boundary conditions are requested. Moreover, we suppose the vector field $\boldsymbol{a}$ as solenoidal, that is:

$$\nabla \cdot \boldsymbol{a} = 0 \tag{84}$$

Thanks to this hypothesis, the differential equation in the strong form 83 can be rewritten as:

$$\boldsymbol{a} \cdot \nabla u - \kappa \Delta u = f \qquad \text{on } \Omega \tag{85}$$

The first term is the convection contribution that distinguishes this problem concerning the Poisson one seen in Chapter 3. In the following, it is useful to define the so-called inflow and outflow part of the Neumann boundary condition.

$$\begin{aligned} \Gamma_N^- &= \{\boldsymbol{x} \in \Gamma_N \mid \boldsymbol{a} \cdot \boldsymbol{n} < 0\} \\ \Gamma_N^+ &= \{\boldsymbol{x} \in \Gamma_N \mid \boldsymbol{a} \cdot \boldsymbol{n} \geq 0\} \end{aligned} \tag{86}$$

Similarly, we can define the portion of Γ where Dirichlet boundary conditions are imposed, and the flow is going in or out with respect to the external unit normal $\boldsymbol{n}$ to the boundary.

$$\begin{aligned} \Gamma_D^- &= \{\boldsymbol{x} \in \Gamma_D \mid \boldsymbol{a} \cdot \boldsymbol{n} < 0\} \\ \Gamma_D^+ &= \{\boldsymbol{x} \in \Gamma_D \mid \boldsymbol{a} \cdot \boldsymbol{n} \geq 0\} \end{aligned} \tag{87}$$

Where naturally, the union of the Dirichlet boundary conditions is called Γ_D, and the union of the Neumann is Γ_N.

Analogously to what we did with the Poisson problem in Chapter 3, we consider the Nitsche approach to write the weak formulation of the problem. Consider as previously the discrete subspace defined in Ω and that is a contained in $H^1(\Omega)$:

$$V^h(\Omega) = \left\{ w^h : w^h \in C^0(\Omega) \cap P^1\left(\Omega_e\right) \right\} \tag{88}$$

Therefore the functions in $V^h(\Omega)$ are continuous piecewise polynomials. In order to manage the additional convection term, is required a stabilization additional term. Following what has been done in [27], we can write the Nitsche weak formulation of the convection-diffusion problem 83:

Find $u^h \in V^h(\Omega)$ such that, $\forall w^h \in V^h(\Omega)$:

$$\begin{aligned}
&- (\nabla w^h, \boldsymbol{a}u^h)_\Omega + \langle w^h, u^h \boldsymbol{a}\cdot\boldsymbol{n}\rangle_{\Gamma^+} + \langle w^h, u_D \boldsymbol{a}\cdot\boldsymbol{n}\rangle_{\Gamma_D^-} + (\nabla w^h, \kappa\nabla u^h)_\Omega - \langle w^h, \kappa\nabla u^h\cdot\boldsymbol{n}\rangle_{\Gamma_D} \\
&- \langle \kappa\nabla w^h\cdot\boldsymbol{n}, u^h - u_D\rangle_{\Gamma_D} + \langle \alpha\kappa/h^\perp w^h, u^h - u_D\rangle_{\Gamma_D} \\
&+ (\boldsymbol{a}\cdot\nabla w^h, \tau(\boldsymbol{a}\cdot\nabla u^h - \kappa\Delta u^h - f))_{\Omega'} = (w^h, f)_\Omega + \langle w^h, t_N\rangle_{\Gamma_N}
\end{aligned} \tag{89}$$

Where α is a penalty parameter that was also present in the Nitsche formulation of the Poisson problem, $h^\perp$ is the characteristic element length and τ is another parameter that controls the amount of stabilization. The last term containing τ is the stabilization term and is needed to stabilize the convection term in cases the convection phenomena are much stronger than diffusion. The stabilization term is a scalar product and it is added just on the union of interior elements:

$$\Omega' = \bigcup_e \Omega_e^\circ \tag{90}$$

By the theory of Nitsche's formulation, we can use as τ a quantity that depends on the amount of convection and diffusion. In [22] is defined as follows:

$$\tau = \left(\boldsymbol{a}\cdot\boldsymbol{G}\boldsymbol{a} + 9(2\kappa)^2\frac{\boldsymbol{G}:\boldsymbol{G}}{3}\right)^{-1/2} \tag{91}$$

Where $\boldsymbol{G}$ is a tensor that depends on the geometry of the finite element mesh. Once again, we can consider the Euler-Lagrange equations of the Nitsche weak formulation. Substituting u^h with the exact solution u, which is supposed to be sufficiently regular (at least in $H^2(\Omega)$), we get the following:

$$\begin{aligned}
&(w^h + \tau\boldsymbol{a}\cdot\nabla w^h, \boldsymbol{a}\cdot\nabla u - \kappa\Delta u - f)_{\Omega'} - \langle \kappa\nabla w^h\cdot\boldsymbol{n} - \alpha\kappa/h^\perp w^h, u - u_D\rangle_{\Gamma_D} \\
&- \langle w^h\boldsymbol{a}\cdot\boldsymbol{n}, u - u_D\rangle_{\Gamma_D^-} - \langle w^h, (\boldsymbol{a}u - \kappa\nabla u)\cdot\boldsymbol{n} + t_N\rangle_{\Gamma_N^-} \\
&+ \langle w^h, \kappa\nabla u - t_N\rangle_{\Gamma_N^+} = 0, \quad \forall w^h \in V^h(\Omega)
\end{aligned} \tag{92}$$

One can note that the first term enforces weakly the differential equation of the strong formulation on the interior union of elements of Ω, and the remaining terms enforce the boundary conditions both on the Dirichlet and on the Neumann boundaries.

4.2 Nitsche variational form for the SB method

Moving towards the shifted boundary method, consider this time that just a portion of Γ is embedded, in particular, we will consider for simplicity the following conditions:

1. the portion of Γ with Dirichlet boundary conditions and with an inflow condition ($\boldsymbol{a}\cdot\boldsymbol{n} < 0$) are treated with a body-fitted approach.

2. the portion of Γ where Neumann boundary conditions are requested, are treated with a body-fitted approach.

Note that these hypotheses are made only to simplify the proof of the stability of the SB method in the convection-diffusion problem. In particular, later in the tests, we will treat Dirichlet boundary conditions with an inflow condition exactly in the same way as outflow Dirichlet boundary conditions without any loss of precision or instabilities. Neumann boundary conditions instead will not be tested. But, it has been shown in [26] that from the numerical point of view also Neumann boundary conditions can be treated as embedded with an SB approach.
Consider then an embedded boundary such that the previous assumptions are satisfied, then following Chapter 3.2 we can introduce the surrogate boundary $\tilde{\Gamma}$ and the surrogate domain $\tilde{\Omega}$. Then, we use again the discrete space of piecewise linear continuous functions defined on the surrogate domain $\tilde{\Omega}$:

$$V^h(\tilde{\Omega}) = \left\{ w^h : w^h \in C^0(\tilde{\Omega}) \cap P^1\left(\Omega_e\right) \right\} \tag{93}$$

Moreover, according to the hypotheses on the embedded boundary, the only embedded boundaries will be where Dirichlet outflow boundary conditions are required. Therefore, for the rest of the boundary conditions, we are going to use the following nomenclature $\tilde{\Gamma}_D^- = \Gamma_D^-$ and $\tilde{\Gamma}_N = \Gamma_N$.

Introducing the surrogate boundary $\tilde{\Gamma}$, the surrogate domain $\tilde{\Omega}$ and its approximation defined in Chapter 2.2, we can state the Nitsche weak formulation for the shifted boundary method.

Find $u^h \in V^h(\tilde{\Omega})$ such that, $\forall w^h \in V^h(\tilde{\Omega})$:

$$\begin{aligned}
&- (\nabla w^h, \boldsymbol{a} u^h)_{\tilde{\Omega}} + \langle w^h, u^h \boldsymbol{a} \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}^+} + \langle w^h, u_D \boldsymbol{a} \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D^-} + (\nabla w^h, \kappa \nabla u^h)_{\tilde{\Omega}} - \langle w^h, \kappa \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \\
&- \langle \kappa \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h + \nabla u^h \cdot \boldsymbol{d} - u_D \rangle_{\tilde{\Gamma}_D^+} - \langle \kappa \nabla w^h \cdot \tilde{\boldsymbol{n}}, u^h - u_D \rangle_{\tilde{\Gamma}_D^-} + \langle \alpha \kappa / h^{\perp} w^h, u^h - u_D \rangle_{\tilde{\Gamma}_D^-} \\
&+ \langle \alpha \kappa / h^{\perp} (w^h + \nabla w^h \cdot \boldsymbol{d}), u^h + \nabla u^h \cdot \boldsymbol{d} - u_D \rangle_{\tilde{\Gamma}_D^+} + \langle \beta \kappa h^{\perp} w^h_{,\bar{\tau}_i}, u^h_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i} \rangle_{\tilde{\Gamma}_D^+} \\
&+ (\boldsymbol{a} \cdot \nabla w^h, \tau(\boldsymbol{a} \cdot \nabla u^h - \kappa \Delta u^h - f))_{\tilde{\Omega}'} = (w^h, f)_{\Omega} + \langle w^h, t_N \rangle_{\tilde{\Gamma}_N}
\end{aligned} \tag{94}$$

As we can see, only the term related with $\tilde{\Gamma}_D^+$ has the second-order approximation for the imposition of the SB Dirichlet boundary conditions. We recall the approximation of Dirichlet boundary conditions on $\tilde{\Gamma}$:

$$u^h + \nabla u^h \cdot \boldsymbol{d} = \bar{u}_D$$

One may also notice that is also present the additional term that enforces the equality between the tangential derivatives of the shifted boundary, i.e. on $\tilde{\Gamma}_D^+$. It was also present in the unsymmetric weak formulation of the Poisson problem. We have shown that the introduction of the additional term does not affect the convergence rate of the formulation. Actually, we have seen that this term is not consistent, but is asymptotically consistent, and it introduces a second-order consistency error, the same rate as the consistency error that shifted boundary conditions introduce.
Differently from the Poisson weak formulation of Equation 25, here is also present the stabilization term that is controlled by τ. The stabilization term is applied to the union of interior elements; the union has intuitively been defined as follows:

$$\tilde{\Omega}' = \bigcup_e \tilde{\Omega}_e^{\circ} \tag{95}$$

Apply again integration by parts backwards. What we get is the associated Euler-Lagrange equation:

$$\begin{aligned}0 = \; & (w^h + \boldsymbol{a}\cdot\nabla w^h, \tau(\boldsymbol{a}\cdot\nabla u^h - \kappa\Delta u^h - f))_{\tilde{\Omega}'} - \langle \boldsymbol{a}\cdot\tilde{\boldsymbol{n}} w^h, u^h - u_D\rangle_{\tilde{\Gamma}_D^-} \\ & - \langle w^h, (\boldsymbol{a}u^h - \kappa\nabla u^h)\cdot\tilde{\boldsymbol{n}} + t_N\rangle_{\tilde{\Gamma}_N^-} + \langle w^h, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}} - t_N\rangle_{\tilde{\Gamma}_N^+} \\ & - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}} - \alpha\kappa/h^\perp w^h, u^h - u_D\rangle_{\tilde{\Gamma}_D^-} \\ & - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}} - \alpha\kappa/h^\perp(w^h + \nabla w^h\cdot\boldsymbol{d}), u^h + \nabla u^h\cdot\boldsymbol{d} - u_D\rangle_{\tilde{\Gamma}_D^+} \\ & + \langle \beta\kappa h^\perp w^h_{,\bar{\tau}_i}, u^h_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i}\rangle_{\tilde{\Gamma}_D^+}\end{aligned} \tag{96}$$

The associated Euler-Lagrange equation allows us to see what is weakly enforced in the Nitsche weak formulation. The differential equation is enforced on $\tilde{\Omega}'$ and conforming boundary conditions on $\tilde{\Gamma}_D^-$, $\tilde{\Gamma}_N^-$ and $\tilde{\Gamma}_N^+$; the imposition of the shifted Dirichlet boundary conditions are enforced on the surrogate boundary $\tilde{\Gamma}_D^+$ and the equality of the tangential derivatives of the solution u^h and of $\bar{u}_D$ on $\tilde{\Gamma}_D^+$.

In the following, we report the Nitsche weak formulation using the bilinear form $a^h(\cdot,\cdot)$ and linear form $l^h(\cdot)$.

Find $u^h \in V^h(\tilde{\Omega})$ such that $\forall w^h \in V^h(\tilde{\Omega})$:

$$a^h(u^h, w^h) = l^h(w^h) \tag{97}$$

Where the bilinear and linear forms are defined in the following:

$$\begin{aligned}a^h(u^h, w^h) = & -(\nabla w^h, \boldsymbol{a}u^h)_{\tilde{\Omega}} + (\nabla w^h, \kappa\nabla u^h)_{\tilde{\Omega}} + (\tau\boldsymbol{a}\cdot\nabla w^h, \boldsymbol{a}\cdot\nabla u^h - \kappa\Delta u^h)_{\tilde{\Omega}'} \\ & + \langle w^h, u^h\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}^+} - \langle w^h, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} \\ & - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}}, u^h + \nabla u^h\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+} - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}}, u^h\rangle_{\tilde{\Gamma}_D^-} \\ & + \langle \alpha\kappa/h^\perp w^h, u^h\rangle_{\tilde{\Gamma}_D^-} + \langle \alpha\kappa/h^\perp(w^h + \nabla w^h\cdot\boldsymbol{d}), u^h + \nabla u^h\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+} \\ & + \langle \beta\kappa h^\perp w^h_{,\bar{\tau}'}, u^h_{,\bar{\tau}_i}\rangle_{\tilde{\Gamma}_D^+} \\ l^h(w^h) = & (w^h, f)_{\tilde{\Omega}} + (\tau\boldsymbol{a}\cdot\nabla w^h, f)_{\tilde{\Omega}'} \\ & + \langle w^h, t_N\rangle_{\tilde{\Gamma}_N} - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}}, u_D\rangle_{\tilde{\Gamma}_D} - \langle w^h, u_D\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^-} \\ & + \langle \alpha\kappa/h^\perp w^h, u_D\rangle_{\tilde{\Gamma}_D^-} + \langle \alpha\kappa/h^\perp(w^h + \nabla w^h\cdot\boldsymbol{d}), u_D\rangle_{\tilde{\Gamma}_D^+} + \langle \beta\kappa h^\perp w^h_{,\bar{\tau}_i}, \bar{u}_{D,\bar{\tau}_i}\rangle_{\tilde{\Gamma}_D^+}\end{aligned} \tag{98}$$

4.3 Conservation properties

Before discussing the stability and well-posedness of the weak formulation 94, let's report the observation done in [27]. One may ask if the Nitsche weak formulation we have proposed has the conservation mass properties, which are crucial in a finite element method that deals with convection. Let us adopt as a test function $w^h = 1$. This implies that $\nabla w^h = 0$, Equation 94 becomes:

$$\begin{aligned}& \langle 1, u_D\boldsymbol{a}\cdot\tilde{\boldsymbol{n}} - \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}} + \alpha\kappa/h^\perp(u^h - u_D)\rangle_{\tilde{\Gamma}_D^-} \\ & + \langle 1, u^h\boldsymbol{a}\cdot\tilde{\boldsymbol{n}} - \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}} + \alpha\kappa/h^\perp(u^h + \nabla u^h\cdot\boldsymbol{d} - u_D)\rangle_{\tilde{\Gamma}_D^+} - \langle 1, t_N\rangle_{\tilde{\Gamma}_N^-} \\ & + \langle 1, u^h\boldsymbol{a}\cdot\tilde{\boldsymbol{n}} - t_N\rangle_{\tilde{\Gamma}_N^+} = (1, f)_{\tilde{\Omega}}\end{aligned} \tag{99}$$

We get the statement of conservation of the numerical fluxes of the Dirichlet and Neumann boundaries. The first term is the flux through the inflow conforming Dirichlet boundaries (convection and diffusion flux), and the second term is the flux associated with the outflow embedded Dirichlet boundaries. It is similar to the first term, but in the penalty term, instead of u^h, we have the first order Taylor expansion $u^h + \nabla u^h \cdot \boldsymbol{d}$, due to the shift of the true boundary. Then, the third and fourth terms are the numerical fluxes associated with the Neumann boundary conditions, inflow and outflow, respectively.
Actually, the only difference from the standard Nitsche's conservation statement is the Taylor expansion for imposing the Dirichlet surrogate boundary condition. In fact, if we apply only body-fitted boundary conditions, i.e. only conforming grids, we end up with the following conservation law:

$$\begin{aligned}&\langle 1, u_D \boldsymbol{a}\cdot\boldsymbol{n} - \kappa\nabla u^h\cdot\boldsymbol{n} + \alpha\kappa/h^{\perp}(u^h - u_D)\rangle_{\Gamma_D^-} + \langle 1, u^h\boldsymbol{a}\cdot\boldsymbol{n} - \kappa\nabla u^h\cdot\boldsymbol{n} + \alpha\kappa/h^{\perp}(u^h-u_D)\rangle_{\Gamma_D^+}\\ &- \langle 1, t_N\rangle_{\Gamma_N^-} + \langle 1, u^h\boldsymbol{a}\cdot\boldsymbol{n} - t_N\rangle_{\Gamma_N^+} = (1,f)_{\Omega}\end{aligned} \tag{100}$$

Introducing a triangulation is possible to extend the conservation properties also locally. Thus the conservation of numerical flux holds both globally and locally between elements.

4.4 Stability of the formulation

In the following, we will discuss some crucial passages on the stability of the convection-diffusion weak formulation for the shifted boundary method. In particular, we will prove the coercivity of the bilinear form and an estimate of the asymptotic consistency and of the error convergence will be obtained.

4.4.1 Coercivity of $a^h(\cdot,\cdot)$

Coercivity of the bilinear form $a^h(\cdot,\cdot)$ requires that there exists a real number $C_{\mathrm{SB}} > 0$ such that:

$$a^h\left(u^h, u^h\right) \geq C_{\mathrm{SB}} \left|\left|\left| u^h \right|\right|\right|_{\mathrm{SB}}^2 \tag{101}$$

Where the $|||\cdot|||$ is a suitable norm of the Hilbert space V^h. Given the definition of the bilinear form of Equation 98, we obtain the following:

$$\begin{aligned}a^h(u^h,u^h) = &-(\nabla u^h, \boldsymbol{a}u^h)_{\tilde{\Omega}} + (\nabla u^h, \kappa\nabla u^h)_{\tilde{\Omega}} + (\boldsymbol{a}\cdot\nabla u^h, \tau(\boldsymbol{a}\cdot\nabla u^h - \kappa\Delta u^h))_{\tilde{\Omega}'}\\ &+ \langle u^h, u^h\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}^+} - \langle u^h, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} - \langle \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}, u^h + \nabla u^h\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+} - \langle \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}, u^h\rangle_{\tilde{\Gamma}_D^-}\\ &+ \langle \alpha\kappa/h^{\perp}u^h, u^h\rangle_{\tilde{\Gamma}_D^-} + \langle \alpha\kappa/h^{\perp}(u^h + \nabla u^h\cdot\boldsymbol{d}), u^h + \nabla u^h\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+} + \langle \beta\kappa h^{\perp}u^h_{,\tilde{\tau}_i}, u^h_{,\tilde{\tau}_i}\rangle^+_{\Gamma_D^+}\end{aligned} \tag{102}$$

Recall the following property of the divergence:

$$\nabla\cdot(u^h\boldsymbol{a}u^h) = \nabla u^h\cdot\boldsymbol{a}u^h + u^h\nabla\cdot(\boldsymbol{a}u^h) \tag{103}$$

Then, we can expand the divergence on the right-hand side and remember the hypothesis that $\nabla\cdot\boldsymbol{a} = 0$:

$$\begin{aligned}\nabla\cdot(u^h\boldsymbol{a}u^h) =&\nabla u^h\cdot\boldsymbol{a}u^h + u^h\nabla\cdot(\boldsymbol{a}u^h) = \nabla u^h\cdot\boldsymbol{a}u^h + u^h(u^h\nabla\cdot\boldsymbol{a} + \boldsymbol{a}\cdot\nabla u^h)\\ =&\nabla u^h\cdot\boldsymbol{a}u^h + u^h\boldsymbol{a}\cdot\nabla u^h = 2\nabla u^h\cdot\boldsymbol{a}u^h\end{aligned} \tag{104}$$

We now apply the latter equality on the first term of Equation 102 together with the divergence theorem:

$$(\nabla u^h, \boldsymbol{a}u^h)_{\tilde{\Omega}} = \frac{1}{2}\langle u^h, u^h \boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}} \tag{105}$$

Then, summing up the terms, we get:

$$\begin{aligned} a^h(u^h,u^h) = {} & 1/2\langle u^h, u^h|\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|\rangle_{\tilde{\Gamma}} + \kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} - (\boldsymbol{a}\cdot\nabla u^h, \tau\kappa\Delta u^h)_{\tilde{\Omega}'} \\ & - 2\langle u^h, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^-} - 2\langle u^h + \nabla u^h\cdot\boldsymbol{d}, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+} + \langle\nabla u^h\cdot\boldsymbol{d}, \kappa\nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+} \\ & + \alpha\kappa\|\sqrt{1/h^\perp}u^h\|^2_{0,\tilde{\Gamma}_D^-} + \alpha\kappa\|\sqrt{1/h^\perp}(u^h + \nabla u^h\cdot\boldsymbol{d})\|^2_{0,\tilde{\Gamma}_D^+} + \beta\kappa\|\sqrt{h^\perp}u^h_{,\bar{\tau}_i}\|^2_{0,\tilde{\Gamma}_D} \end{aligned} \tag{106}$$

Moreover, it is possible to prove the following bound (see [22]):

$$\kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} - (\boldsymbol{a}\cdot\nabla u^h, \tau\kappa\Delta u^h)_{\tilde{\Omega}'} \geq \frac{1}{2}\kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \frac{1}{2}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} \tag{107}$$

We now try to find a bound for the term $-2\kappa\langle u^h + \nabla u^h\cdot\boldsymbol{d}, \nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+}$. Recall the ϵ-inequality that comes from the combination of Young's and Cauchy-Schwartz's inequality. For any $\epsilon\in\mathbb{R}^+$:

$$\langle v, w\rangle \leq \|v\|\|w\| \leq \frac{1}{2}\left(\epsilon\|v\|^2 + \frac{1}{\epsilon}\|w\|^2\right) \tag{108}$$

Where v and w belongs in an Hilbert space with norm $\|v\| = \sqrt{\langle v, v\rangle}$, where $\langle\cdot,\cdot\rangle$ is its inner product. Then our term can be bound by the following:

$$-2\kappa\langle u^h + \nabla u^h\cdot\boldsymbol{d}, \nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+} \geq -\kappa\epsilon_1\|\sqrt{h^\perp}\nabla u^h\cdot\tilde{\boldsymbol{n}}\|^2_{0,\tilde{\Gamma}_D^-} - \frac{\kappa}{\epsilon_1}\|\sqrt{1/h^\perp}u^h\|^2_{0,\tilde{\Gamma}_D^-} \tag{109}$$

Then, recall the disequality already mentioned in Equation 45:

$$\|\sqrt{h^\perp}\nabla w\cdot\boldsymbol{v}\|^2_{0,\tilde{\Gamma}} \leq C_I\|\nabla w\|^2_{0,\tilde{\Omega}}. \tag{110}$$

Where we can choose $\boldsymbol{v} = \tilde{\boldsymbol{n}}$ due to the hypothesis that $\boldsymbol{n}\cdot\tilde{\boldsymbol{n}} \geq 0$. The term of Equation 109 becomes:

$$-2\kappa\langle u^h + \nabla u^h\cdot\boldsymbol{d}, \nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+} \geq -\epsilon_1 C_I\kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} - \frac{\kappa}{\epsilon_1}\|\sqrt{1/h^\perp}u^h\|^2_{0,\tilde{\Gamma}_D^-} \tag{111}$$

Furthermore, we can identically bound another term as we did in Chapter 3 when we discussed the coercivity of the bilinear form. Using the fact that $\boldsymbol{d} = \|\boldsymbol{d}\|\boldsymbol{n}$ and $\|\boldsymbol{d}\| \leq h^\perp$ we have proved that:

$$\langle\nabla u^h\cdot\boldsymbol{d}, \nabla u^h\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^+} \geq -\frac{\epsilon_2}{2}C_I\|\nabla u^h\|^2_{0,\tilde{\Omega}} - \frac{1}{2\epsilon_2}\|\sqrt{h^\perp}u^h_{,\bar{\tau}}\|^2_{0,\tilde{\Gamma}_D^+} \tag{112}$$

Where ϵ_2 comes from the ϵ-inequality, thus, it is an arbitrary positive real number. Note also the use of the shorter notation $u^h_{,\bar{\tau}} = u^h_{,\bar{\tau}_i}\bar{\tau}_i$.
We now insert the previous bounds (equations 111 and 112) in the bound of the bilinear form of Equation 106:

$$\begin{aligned} a^h(u^h,u^h) \geq {} & \frac{1}{2}\||\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|^{1/2}u^h\|_{0,\tilde{\Gamma}} + (\frac{1}{2} - C_I(2\epsilon_1 + \frac{1}{2}\epsilon_2))\kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \frac{1}{2}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} \\ & + (\alpha - \frac{1}{\epsilon_1})\kappa\|\sqrt{1/h^\perp}u^h\|^2_{0,\tilde{\Gamma}_D^-} + (\alpha - \frac{1}{\epsilon_1})\kappa\|\sqrt{1/h^\perp}(u^h + \nabla u^h\cdot\boldsymbol{d})\|^2_{0,\tilde{\Gamma}_D^+} \\ & + (\beta - \frac{1}{2\epsilon_2})\kappa\|\sqrt{h^\perp}u^h_{,\bar{\tau}}\|^2_{0,\tilde{\Gamma}_D} \end{aligned} \tag{113}$$

Consider now the definition of a convenient norm for the Hilbert space V^h:

$$\begin{aligned}|||u^h \mid |||^2_{\mathrm{SB}} =& |||\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|^{1/2}u^h\|_{0,\tilde{\Gamma}} + \kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} \\ &+ \kappa\|\sqrt{1/h^{\perp}}u^h\|^2_{0,\tilde{\Gamma}_D^-} + \kappa\|\sqrt{1/h^{\perp}}(u^h + \nabla u^h\cdot\boldsymbol{d})\|^2_{0,\tilde{\Gamma}_D^+} + \kappa\|\sqrt{h^{\perp}}u^h_{,\bar{\tau}}\|^2_{0,\tilde{\Gamma}_D}\end{aligned} \tag{114}$$

Then, taking ϵ_1 and ϵ_2 in such a way the coefficient in front of the term $\kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}}$ is at least less than $1/2$, i.e.:

$$C_I(2\epsilon_1 + \frac{1}{2}\epsilon_2) < \frac{1}{2} \tag{115}$$

Then we can always find a real positive number C_{SB} such that the bilinear is coercive, that is:

$$a^h\left(u^h, u^h\right) \geq C_{\mathrm{SB}}\left|\left|\left|u^h\right|\right|\right|^2_{\mathrm{SB}}$$

And it is possible if taking C_{SB} as:

$$C_{\mathrm{SB}} = \min\left(\frac{1}{2} - C_I\left(2\epsilon_1 + \frac{\epsilon_2}{2}\right), \alpha - \frac{1}{\epsilon_1}, \beta - \frac{1}{2\epsilon_2}\right) \tag{116}$$

One may notice that if we consider just conforming boundaries, or in other words, if the SB method is not applied on any portion of Γ, then everything degenerates to a standard study of the Nitsche formulation in a convection-diffusion problem. In fact, if shifted boundaries are not present is equivalent to set $\boldsymbol{d} = \boldsymbol{0}$; also remember the additional term we have added to match the tangential derivative that was weighted by the parameter β. Thus by setting $\beta = 0$ and $\boldsymbol{d} = \boldsymbol{0}$ we get the proof of the coercivity of the bilinear form $a^h_{\mathrm{NI}}(\cdot,\cdot)$ of the variational form of Equation 89:

$$\begin{aligned}&a^h_{\mathrm{NI}}(u^h, u^h) \geq C_{\mathrm{NI}}|||u^h|||^2_{\mathrm{NI}} \\ &|||u^h|||^2_{\mathrm{NI}} = |||\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|^{1/2}u^h\|_{0,\tilde{\Gamma}} + \kappa\|\nabla u^h\|^2_{0,\tilde{\Omega}} + \|\tau^{1/2}\boldsymbol{a}\cdot\nabla u^h\|^2_{0,\tilde{\Omega}} + \kappa\|\sqrt{1/h^{\perp}}u^h\|^2_{0,\tilde{\Gamma}_D} \\ &C_{\mathrm{NI}} = \min\left(\frac{1}{2} - C_I\epsilon_1, \alpha - \frac{1}{\epsilon_1}\right)\end{aligned}$$

4.4.2 Consistency

Following what has been discussed in Chapter 3.4.4, we propose now the asymptotic consistency estimate of the SB method on $\tilde{\Omega}$ for the convection-diffusion Nitsche weak formulation. Let's use the same notation for the error $e = u - u^h$, and what we would like to estimate is the exponent k of the following equation:

$$a^h_s(u - u^h, w^h) = |||w^h|||_{\mathrm{SB}}O(h^k) \tag{117}$$

for any test function w^h that belongs to the test space $V^h(\tilde{\Omega})$. In the case of the Poisson equation using the SB method for the imposition of Dirichlet boundary conditions, we proved that at least we have $k = 1.5$. To give a similar estimate for k in the convection-diffusion case, consider the exact solution u on the surrogate domain $\tilde{\Omega}$. We have a standard Nitsche formulation without the SB term for the Dirichlet boundary condition and without the matching condition for the tangential derivatives on the surrogate boundary $\tilde{\Gamma}$.
The weak formulation can be expressed using the bilinear and linear form and, as done in Chapter 3.4.4, we put a tilde on top of $a^h(\cdot,\cdot)$ and $l^h(\cdot)$ to distinguish from the SB bilinear and linear form.

Find $u \in H^1(\tilde{\Omega})$ such that $\forall w^h \in V^h(\tilde{\Omega})$:

$$\tilde{a}^h(u, w^h) = \tilde{l}^h(w^h) \tag{118}$$

Where the $\tilde{a}^h(\cdot,\cdot)$ and $\tilde{l}^h(\cdot)$ are defined as follows:

$$\begin{aligned}
\tilde{a}^h(u, w^h) &= -(\nabla w^h, \boldsymbol{a}u)_{\tilde{\Omega}} + (\nabla w^h, \kappa \nabla u)_{\tilde{\Omega}} + (\boldsymbol{a}\cdot\nabla w^h, \tau(\boldsymbol{a}\cdot\nabla u - \kappa\Delta u))_{\tilde{\Omega}'} \\
&\quad + \langle w^h, u\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}^+} - \langle w^h, \kappa\nabla u\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D} - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}}, u\rangle_{\tilde{\Gamma}_D^-} + \langle \alpha\kappa/h^{\perp} w^h, u\rangle_{\tilde{\Gamma}_D^-} \\
\tilde{l}^h(w^h) &= (w^h + \tau\boldsymbol{a}\cdot\nabla w^h, f)_{\tilde{\Omega}} + \langle w^h, t_N\rangle_{\tilde{\Gamma}_N} - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}}, u_D\rangle_{\tilde{\Gamma}_D^-} - \langle w^h, u_D\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D^-} \\
&\quad + \langle \alpha\kappa/h^{\perp} w^h, u_D\rangle_{\tilde{\Gamma}_D^-}
\end{aligned} \tag{119}$$

As one may notice, the Dirichlet boundary conditions are imposed also on $\tilde{\Gamma}$ because this boundary was conforming. We now write the SB bilinear form of Equation 98 in terms of the bilinear form of the exact solution of Equation 119:

$$\begin{aligned}
a^h(e,w^h) &= a^h(u-u^h, w^h) = a^h(u, w^h) - a^h(u^h, w^h) = a^h(u, w^h) - l^h(w^h) \\
&= \tilde{a}^h(u, w^h) - \tilde{l}^h(w^h) - \langle \kappa\nabla w^h\cdot\tilde{\boldsymbol{n}} - \alpha\kappa/h^{\perp}(w^h + \nabla w^h\cdot\boldsymbol{d}), u + \nabla u\cdot\boldsymbol{d} - \bar{u}_D\rangle_{\tilde{\Gamma}_D^+} \\
&\quad + \langle \beta\kappa h^{\perp} w_{,\bar{\tau}_i}, u_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i}\rangle_{\tilde{\Gamma}_D^+}
\end{aligned} \tag{120}$$

Where notice that the terms that come from comparing the two linear and bilinear forms are the last two terms. They involve the Dirichlet SB conditions $u + \nabla u\cdot\boldsymbol{d} - \bar{u}_D$ and the tangential derivatives on $\tilde{\Gamma}$, i.e. $u_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i}$. Then, we introduce the estimates on the exact solution u we have discussed in Chapter 3.4.4:

$$\begin{aligned}
a^h(e,w^h) &= \overbrace{\tilde{a}^h(u, w^h) - \tilde{l}^h(w^h)}^{=0} - \langle \sqrt{\kappa h^{\perp}}\nabla w^h\cdot\tilde{\boldsymbol{n}}, \sqrt{\kappa/h^{\perp}}\overbrace{(u + \nabla u\cdot\boldsymbol{d} - \bar{u}_D)}^{O(h^2)}\rangle_{\tilde{\Gamma}_D^+} \\
&\quad + \langle \beta\sqrt{\kappa h^{\perp}} w^h_{,\bar{\tau}_i}, \underbrace{\sqrt{\kappa h^{\perp}}\overbrace{(u_{,\bar{\tau}_i} - \bar{u}_{D,\bar{\tau}_i})}^{O(h)}}_{O(h^{3/2})}\rangle_{\tilde{\Gamma}_D^+} \\
&\quad + \langle \alpha\sqrt{\kappa/h^{\perp}}(w^h + \nabla w^h\cdot\boldsymbol{d}), \sqrt{\kappa/h^{\perp}}\underbrace{(u + \nabla u\cdot\boldsymbol{d} - \bar{u}_D)}_{O(h^2)}\rangle_{\tilde{\Gamma}_D^+}
\end{aligned} \tag{121}$$

Then, applying similar passages described in Chapter 4.4.1 and introducing a suitable norm, we obtain a bound for the consistency of the SB Nitsche formulation:

$$a^h(e, w^h) \leq \left|\left|\left| w^h \right|\right|\right|_{\mathrm{SB}} O\left(h^{3/2}\right) \tag{122}$$

Obtaining an estimate equal to 1.5, the same value that we got in the case of the Poisson equation.

4.4.3 Convergence of the error

One may use a smart trick, already used in [22], to prove convergence. First of all, we split the total error into two components, the numerical error e^h and the interpolation error η:

$$\begin{aligned}
e &= e^h + \eta \\
e^h &= u^h - \tilde{u}^h \\
\eta &= \tilde{u}^h - u
\end{aligned} \tag{123}$$

Where with $\tilde{u}^h$, it is meant the polynomial interpolant of the exact solution u. Let's use equation 122 that holds true $\forall w^h \in V^h(\tilde{\Omega})$, and take $w^h = e^h$, i.e. the numerical error $e^h \in V^h(\tilde{\Omega})$.

$$a^h(e, e^h) \leq \left|\left|\left|e^h\right|\right|\right|_{\text{SB}} O\left(h^{3/2}\right) \tag{124}$$

Then, using the definitions in equation 123 we can write:

$$\begin{aligned} a^h(e^h, e^h) = a^h(e-\eta, e^h) &= |a^h(e, e^h)| + |a^h(\eta, e^h)| \\ &\leq \left|\left|\left|e^h\right|\right|\right|_{\text{SB}} O\left(h^{3/2}\right) + |a^h(\eta, e^h)| \end{aligned} \tag{125}$$

Recall then that the bilinear form $a^h(\cdot,\cdot)$ satisfies the coercivity property, Equation 101, and therefore we obtain the following disequality:

$$C_{\text{SB}} \left|\left|\left|e^h\right|\right|\right|_{\text{SB}}^2 \leq \left|\left|\left|e^h\right|\right|\right|_{\text{SB}} O\left(h^{3/2}\right) + |a^h(\eta, e^h)| \tag{126}$$

Focusing on the term $|a^h(\eta, e^h)|$ that we would like to bound we can write the extended form of the bilinear form $a^h(\cdot,\cdot)$:

$$\begin{aligned} |a(\eta, e^h)| \leq\ & |(\nabla e^h, \boldsymbol{a}\eta)_{\tilde{\Omega}}| + |(\nabla e^h, \kappa\nabla\eta)_{\tilde{\Omega}}| + |(\boldsymbol{a}\cdot\nabla e^h, \tau(\boldsymbol{a}\cdot\nabla\eta - \kappa\Delta\eta))_{\tilde{\Omega}'}| \\ &+ |\langle e^h, \eta\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}^+}| + |\langle e^h, \kappa\nabla\eta\cdot\tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_D}| + |\langle\kappa\nabla e^h\cdot\tilde{\boldsymbol{n}}, \eta + \nabla\eta\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+}| + |\langle\kappa\nabla e^h\cdot\tilde{\boldsymbol{n}}, \eta\rangle_{\tilde{\Gamma}_D^-}| \\ &+ |\langle\alpha\kappa/h^\perp e^h, \eta\rangle_{\tilde{\Gamma}_D^-}| + |\langle\alpha\kappa/h^\perp(e^h + \nabla e^h\cdot\boldsymbol{d}), \eta + \nabla\eta\cdot\boldsymbol{d}\rangle_{\tilde{\Gamma}_D^+}| + |\langle\beta\kappa h^\perp e^h_{,\bar{\tau}_i}, \eta, \bar{\tau}_i\rangle_{\tilde{\Gamma}_D^+}| \end{aligned} \tag{127}$$

Apply now the ϵ-inequality, already used and stated in equation 108, multiple times:

$$\begin{aligned} |a(\eta, e^h)| & \frac{\epsilon_1}{2}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla e^h\|_{0,\tilde{\Omega}}^2 + \frac{1}{2\epsilon_1}\|\tau^{-1/2}\eta\|_{0,\tilde{\Omega}^2}^2 + \frac{\epsilon_2}{2}\|\kappa^{1/2}\nabla e^h\|_{0,\tilde{\Omega}}^2 + \frac{1}{2\epsilon_2}\|\kappa^{1/2}\nabla\eta\|_{0,\tilde{\Omega}}^2 \\ &+ \frac{\epsilon_3}{2}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla e^h\|_{0,\tilde{\Omega}^2}^2 + \frac{1}{2\epsilon_3}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla\eta\|_{0,\tilde{\Omega}}^2 + \frac{\epsilon_4}{2}\|\tau^{1/2}\boldsymbol{a}\cdot\nabla e^h\|_{0,\tilde{\Omega}}^2 + \frac{1}{2\epsilon_4}\|\tau^{1/2}\kappa\Delta\eta\|_{0,\tilde{\Omega}'}^2 \\ &+ \frac{\epsilon_5}{2}\||\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|e^h\|_{0,\tilde{\Gamma}^+}^2 + \frac{1}{2\epsilon_5}\||\boldsymbol{a}\cdot\tilde{\boldsymbol{n}}|\eta\|_{0,\tilde{\Gamma}^+}^2 + \frac{\epsilon_6}{2}\|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D}^2 + \frac{1}{2\epsilon_6}\|\sqrt{\kappa h^\perp}\nabla\eta\cdot\tilde{\boldsymbol{n}}_{0,\tilde{\Gamma}_D}^2 \\ &+ \frac{\epsilon_7}{2}\|\sqrt{\kappa h^\perp}\nabla e^h\cdot\tilde{\boldsymbol{n}}\|_{0,\tilde{\Gamma}_D^+}^2 + \frac{1}{2\epsilon_7}\|\sqrt{\kappa/h^\perp}(\eta + \nabla\eta\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 + \frac{\epsilon_8}{2}\|\sqrt{\kappa h^\perp}\nabla e^h\cdot\tilde{\boldsymbol{n}}\|_{0,\tilde{\Gamma}_D^-}^2 \\ &+ \frac{1}{2\epsilon_8}\|\sqrt{\kappa/h^\perp}\eta\|_{0,\tilde{\Gamma}_D^-}^2 + \frac{\alpha\epsilon_9}{2}\|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^-}^2 + \frac{\alpha}{2\epsilon_9}\|\sqrt{\kappa/h^\perp}\eta\|_{0,\tilde{\Gamma}_D^-}^2 \\ &+ \frac{\alpha\epsilon_{10}}{2}\|\sqrt{\kappa/h^\perp}(e^h + \nabla e^h\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 + \frac{\alpha}{2\epsilon_{10}}\|\sqrt{\kappa/h^\perp}(\eta + \nabla\eta\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 \\ &+ \frac{\beta\epsilon_{11}}{2}\|\sqrt{\kappa h^\perp}e^h_{,\bar{\tau}_i}\|_{0,\tilde{\Gamma}_D^+}^2 + \frac{\beta}{2\epsilon_{11}}\|\sqrt{\kappa h^\perp}\eta, \bar{\tau}_i\|_{0,\tilde{\Gamma}_D^+}^2 \end{aligned} \tag{128}$$

One of the problematic terms to bound that arises is the term $\|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D}^2$, but splitting it and applying first the triangle inequality and then the discrete trace inequality, we get:

$$\begin{aligned} \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D}^2 &= \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^-}^2 + \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^+}^2 \\ &\leq \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^-}^2 + \|\sqrt{\kappa/h^\perp}(e^h + \nabla e^h\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 + \|\sqrt{\kappa/h^\perp}\nabla e^h\cdot\boldsymbol{d}\|_{0,\tilde{\Gamma}_D^+}^2 \\ &\leq \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^-}^2 + \|\sqrt{\kappa/h^\perp}(e^h + \nabla e^h\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 + \|\sqrt{\kappa h^\perp}\nabla e^h\cdot\boldsymbol{n}\|_{0,\tilde{\Gamma}_D^+}^2 \\ &\leq \|\sqrt{\kappa/h^\perp}e^h\|_{0,\tilde{\Gamma}_D^-}^2 + \|\sqrt{\kappa/h^\perp}(e^h + \nabla e^h\cdot\boldsymbol{d})\|_{0,\tilde{\Gamma}_D^+}^2 + C_I\|\kappa^{1/2}\nabla e^h\|_{0,\tilde{\Omega}}^2 \end{aligned} \tag{129}$$

Considering again equation 128, all the ϵ_i are positive and arbitrary, then can be chosen in such a way the following system is satisfied:

$$\begin{cases} \epsilon_1, \epsilon_3, \epsilon_4 \leq C_{\text{SB}}/3 \\ \epsilon_2 + C_I \epsilon_6 \leq C_{\text{SB}} \\ \alpha\epsilon_{10} + \epsilon_6 \leq C_{\text{SB}} \\ \beta\epsilon_{11} \leq C_{\text{SB}} \\ \alpha\epsilon_9 + \epsilon_6 \leq C_{\text{SB}} \\ \alpha\epsilon_5 \leq C_{\text{SB}} \end{cases} \tag{130}$$

In this way all the term involving e^h of equation 128 are less or equal of $1/2\, C_{\text{SB}} |||e^h|||^2_{\text{SB}}$, thus equation 126 can be written as:

$$C_{\text{SB}} \left|\left|\left| e^h \right|\right|\right|^2_{\text{SB}} \leq \left|\left|\left| e^h \right|\right|\right|_{\text{SB}} O\left(h^{3/2}\right) + 1/2 C_{\text{SB}} \left|\left|\left| e^h \right|\right|\right|^2_{\text{SB}} + \Phi(\eta) \tag{131}$$

Where $\Phi(\eta)$ contains all the terms depending on the interpolation error η of equation 128. For linear interpolation functions, one can prove that the terms depending only on the interpolation error can be bounded by $C_{int} h^2$. The previous equation then becomes:

$$1/2\, C_{\text{SB}} \left|\left|\left| e^h \right|\right|\right|^2_{\text{SB}} \leq \left|\left|\left| e^h \right|\right|\right|_{\text{SB}} O\left(h^{3/2}\right) + C_{int} h^2 \tag{132}$$

That can be analyzed deeply by studying when the parabola depending on $|||e^h|||_{\text{SB}}$ takes values less or equal to zero:

$$\frac{1}{2} y^2 - c_1 h^{3/2} - c_2 h^2 \leq 0 \tag{133}$$

where $y = |||e^h|||_{\text{SB}} \geq 0$ and $c_1, c_2 > 0$ and they are independent on h. The parabola has a positive concavity and its zeros are the following:

$$y_{\pm} = c_1 h^{3/2} \pm \sqrt{c_1^2 h^3 + 2 c_2 h^2} \tag{134}$$

Therefore the disequality is satisfied when $y \in [y_-, y_+]$, but $y \geq 0$, therefore the interval is the following:

$$0 \leq y = |||e^h|||_{\text{SB}} \leq c_1 h^{3/2} + \sqrt{c_1^2 h^3 + 2 c_2 h^2} \tag{135}$$

This implies that the H^1-norm of e^h is only first-order accurate, since considering h small enough we get that:

$$\left|\left|\left| e^h \right|\right|\right|^2_{\text{SB}} \leq C' h \tag{136}$$

That, using the triangle inequality and the second-order bound for the term depending on η is equivalent to:

$$|||e|||^2_{\text{SB}} \leq Ch \tag{137}$$

We have finally found an estimate for the H^1-norm of the error for the convection-diffusion problem. An estimate for the L^2-norm is not yet available. The strategy, i.e. the Nitsche-Aubin trick, which has been used in Chapter 3.4.5 in the convergence estimate of the error for the Poisson problem couldn't be applied in this case.
Regardless of this estimate, the results chapter (Chapter 6) will show that in all convection-diffusion problems, in both stationary and transient cases, the second-order convergence of the L^2-norm of the error has been achieved.

5 Implementation details

In this chapter, we describe the specifics of the shifted boundary implementation technique that, so far, has been explained only theoretically in [6, 7, 26, 27, 32]. In fact, in the papers we have mentioned, implementation details were not present. So, what we are going to explain in this context has been created from scratch and personally adapted to the open-source software Kratos Multiphysics.

5.1 Kratos Multi-physics

Kratos is a free and open-source framework developed and maintained mainly by International Center for Numerical Methods in Engineering (CIMNE), a research organization based in Spain. Since 2005, Kratos has been actively developed and used in various fields such as aerospace, civil engineering, and biomechanics. Its strength lies in its flexibility and modularity, which allows for easy customization to meet specific application and discipline needs. It supports a wide range of numerical methods, solvers, time integration algorithms, and optimization tools.

Figure 7: CIMNE logo

Kratos is also highly customizable in terms of the programming language used to interface with it. While the core framework is written in C++, users can also interact with Kratos using Python. It is well-suited for large-scale simulations and high-performance computing environments, having been extensively tested and benchmarked against industry-standard software packages.

Figure 8: Kratos Multiphysics logo

One of Kratos' key features is its ability to handle complex geometries and topologies, supporting a wide range of meshing techniques, including structured, unstructured, and adaptive meshes, as well as mesh-free methods.

Kratos has a large and active community of developers and users who contribute to its development and maintenance. It is intended for use by finite element developers, application developers, and package users in engineering and design. And its multi-disciplinary nature supports the involvement of people from various fields. Kratos provides a flexible external interface that enables users to use different features of Kratos without changing its implementation.
More information about the Kratos Multiphysics' structure and its potentialities are present in [11, 12, 19], whereas the repository is in GitHub: `https://github.com/KratosMultiphysics`.

5.2 General strategy

In the following sections, all the Kratos steps that are necessary to solve a problem using the shifted boundary method will be explained in detail.
Specifically, thanks to Kratos's structure, we will not need to create a finite element code from scratch. Rather, the additions will be done to a type of class called solver. Kratos contains a variety of applications, but our efforts were focused on the so-called convection-diffusion application. This application, in order to run properly, needs some input parameters that contain all the geometric details, such as coordinates of the nodes, the triangulation, eventual conforming boundary conditions, material properties, etc.
In addition to this information, Kratos requires the solver settings, the implementation of the element types, and the data and methods necessary to solve the final FEM linear system. Thanks to the structure of Kratos, the majority of these implementation steps are completely suitable for an SB simulation. An SB implementation primarily requires the modification of the imposition of boundary conditions and a few other things. Thanks to the modularity and extensibility of Kratos, we can basically intervene where it is needed that, as already mentioned, in the Kratos' framework is called the solver.

There are many existing solvers in Kratos. In the Poisson problem, the solver more similar to our purpose is the so-called `ConvectionDiffusionTransientSolver`. This solver class is derived from its parent class `ConvectionDiffusionSolver`, that is the base class for the convection-diffusion solvers. This class provides functions for importing and exporting models, adding nodal variables and degrees of freedom, and solving each solution step. Thus the best way to proceed is to create our own derived class of the so-called `ConvectionDiffusionTransientSolver` to inherit all the methods defined in the parents. Moreover, at the same time, we can add our methods to implement the imposition of shifted boundary conditions on the embedded boundaries.
We then have to create our personal SB solvers that can be used for solving, for instance, stationary convection-diffusion problems (`SBMConvectionDiffusionStationarySolver`) or transient ones (`SBMConvectionDiffusionTransientSolver`).

Therefore, all the descriptions that will be described in the following chapters have to be considered as implemented in these derived classes just mentioned. Only one additional consideration regarding the flux through the shifted boundary needs to be taken into account. This detail will lead to a creation of a derived class of the element type we are using in the simulation. We will discuss it in Chapter 5.6, but, for the moment, let's only mention the elements that are going to be used: the `LaplacianElement` is used in pure diffusion problems, and the `EulerianConvection-DiffusionElement` that instead can also manage the presence of convection. Both of them have been implemented in Kratos by Professors Riccardo Rossi and Rubén Zorrilla, and as we are going to see, we will need to derive suitable classes to add the flux term through the surrogate boundary.

5.3 Shifted boundary

The first implementation step of the SB method in our solver class, derived from a solver class that uses a standard body-fitted approach, is the importation of the true boundary.
In fact, one must understand that previously the background mesh of the computational domain Ω has already been imported. It represents the computational mesh we may use for many cases with different embedded boundaries. Consider a general rectangular domain. Then, imagine the common problem of representing an immersed object inside the rectangular computational domain. Unfitted methods, such as the SB method, can use the same background mesh for different immersed shapes. They import the so-called true boundary and then put it on top of the back-

ground mesh. The true boundary often will be called the "skin boundary" because it represents the skin between the immersed object and the remaining computational domain. In the context of the finite element method, it is a closed line or surface (in 2D or 3D respectively) made by element edges; their dimension is one less than the dimension of the problem, i.e. for a 2D problem, the skin elements are one-dimensional segments and in a 3D problem elements can be triangular or quadrilateral faces. From now we consider for simplicity a 2D problem, but all the analysis we perform can be generalized for the 3D case.

Once the skin boundary is imported in Kratos, the object's shape will naturally intersect the background mesh cutting its elements. It happens in any unfitted method. Each method differs based on how they impose the interface conditions between the object and the rest of the domain. As said, the SB method considers an approximation of the skin boundary using a surrogate one. In particular, it uses an approximation of the geometry and imposes some modified boundary conditions on this approximated boundary.

5.3.1 Import the skin boundary

Thus, the first step in our SB solver class is importing the skin boundary. The skin or true boundary in real applications is part of the geometry and it is provided as a CAD entity. For this reason, in Kratos, the geometry entities, such as the skin boundary, can be imported through an MDPA file. MDPA stands for Mesh Data Parallel Archive, and are utilized for storing information about the mesh, such as the coordinates of the mesh nodes and the connectivity of the mesh elements. Even the background mesh usually is imported into Kratos through an MDPA file provided together with the project parameters of the simulation.
The MDPA for the skin boundary needs to contain the coordinates of the nodes, which form the true boundary, and their connectivity. The connectivity is necessary for the software in order to reconstruct the closed curved that defines the skin boundary. A straightforward example of an MDPA file that describes a closed triangular shape is the following:

```
Begin Nodes
    1   0.0   0.0   0.0
    2   1.0   0.0   0.0
    3   0.0   1.0   0.0
End Nodes

Begin Conditions LineCondition2D2N
    1 0 1 2
    2 0 2 3
    3 0 3 1
End Conditions
```

Note that what we've referred to as skin elements in the MDPA file are represented by `LineConditions2D2N`.

Then, in the solver, we can import this MDPA file, which needs to be present in the same folder of the current simulation, by the `CreateModelPart` and `ReadModelPart` methods of Kratos:

```
current_model = KratosMultiphysics.Model()
skin_model_part = current_model.CreateModelPart("skin_model_part")
KratosMultiphysics.ModelPartIO("name_structural_mdpa").ReadModelPart(
    skin_model_part)
```

Where `"skin_model_part"` will be the name of the model part containing nodes and conditions defined in the MDPA file, and `"name_structural_mdpa"` is the name of the MDPA file we are importing.

5.3.2 Level set function

In all unfitted techniques, the immersed object skin mesh is added to the model after the background mesh and put on top of it. Therefore, in general, there will be cut elements. We have already said that the peculiarity of the SB method is that the cut elements are removed from the simulation. Later, this fact will be explained in detail, and how this is possible in Kratos will be discussed. The problem at this point is how to detect these cut elements and how it is possible to find the shifted boundary.

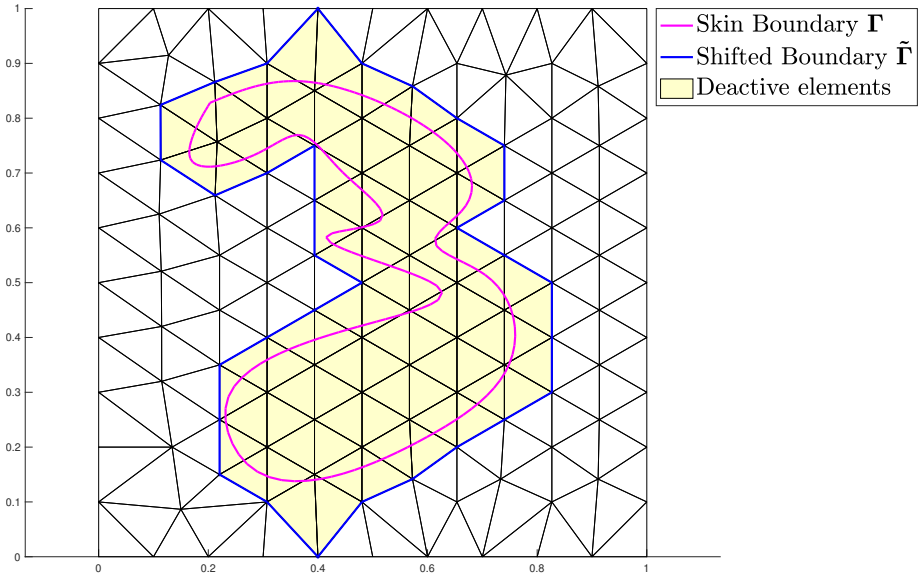


Figure 9: Example of skin boundary Γ and the corresponding surrogate boundary $\tilde{\Gamma}$.

In the SB method, the shifted boundary is defined as the most similar polyline connecting nodes belonging to the background mesh. Thus, it can be interpreted as the closest external polyline that includes the cut elements (see Figure 9). So, once we have the cut elements will be possible to highlight the surrogate nodes that form the shifted or surrogate boundary. At this point, the problem is how to find the cut elements. It is not yet clear which is the best way to tell the software that on top of the background mesh there is a skin boundary. First, we would like to know which background elements are cut by it.

The solution to this problem was already implemented in Kratos and the idea is to use a level set function [34] that is defined as the signed distance to the object skin. Due to its definition, the level set function allows the reconstruction of the true boundary as the zero iso-surface of the level set function. In Kratos, the algorithm that takes as inputs the background mesh and a skin boundary and returns the signed distance of each node of the background mesh is called `CalculateDistanceToSkinProcess2D`. The C++ function can be used in Kratos as follows:

```
1 main_model_part.AddNodalSolutionStepVariable(KratosMultiphysics.DISTANCE
    )
```

```
KratosMultiphysics.CalculateDistanceToSkinProcess2D(main_model_part,
    skin_model_part).Execute()
```

First, note that we have added a nodal variable called `DISTANCE`, it is the variable where the signed distance function is stored. When the function `CalculateDistanceToSkinProcess2D` is executed, each node of the background mesh has a new attribute variable called `KratosMultiphysics.DISTANCE` that returns the signed distance from the skin boundary.

The `CalculateDistanceToSkinProcess2D` is an advanced tool available in Kratos and the structure of its algorithm can be divided into the following steps:

Algorithm 1 Idea of `CalculateDistanceToSkinProcess2D`

```
Obtain an octree with candidate background elements for each of the skin elements.
for each of the skin elements do
    Get the candidate background elements from the octree.
    for each of the candidate background elements do
        Check if the candidate background element is cut by the skin element.
        if is cut then
            Compute the discontinuous distance.
Make the distance continuous (no signed).
Inside/Outside with ray-casting.
```

In the first step, a search algorithm is implemented, where for each of the skin elements a set of candidate background elements are provided. These candidate elements may be cut by the skin boundary.
Then, each of these candidates is taken singularly, and the algorithm checks if they are actually cut or not. In the case they are cut by the skin boundary, a discontinuous distance is computed without taking into account the sign, i.e. without knowing if they are inside or outside the immersed object. In the fourth step, this discontinuous distance is made continuous by an averaging procedure between all the background mesh elements.
The last step is essential for understanding if a node is inside or outside the immersed object. Because, so far, the algorithm did not care about positive (outside) or negative (inside) distances. It is done by a technique called ray-casting [34].

In Figure 10 are reported as an example the results we got applying the `CalculateDistanceToSkinProcess2D` to a very simple mesh. As described, the results consist of a new attribute variable called distance that each node of the background mesh has. This value is the minimum signed distance from the true boundary Γ marked in dark green in the figure.

One may notice in Figure 10 that there are many signed distances equal to 1.41. It is due to the first step of the algorithm of the process `CalculateDistanceToSkinProcess2D`. Thanks to the searching process of the candidates, the algorithm skips the calculations on the elements that are "far" from the skin boundary. In those elements, the value of the variable distance is set to a fixed and large one. It is a smart way to avoid heavy calculations that are not needed; in the next chapter is discussed how one can use the signed distance results for computing the surrogate boundary $\tilde{\Gamma}$ and the surrogate boundary nodes.

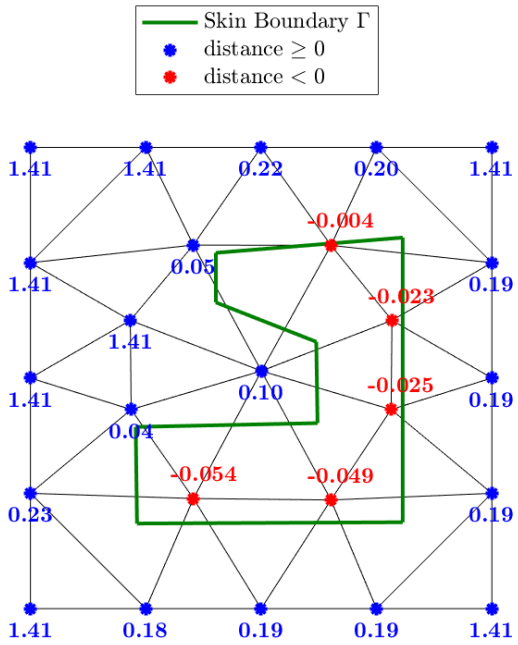


Figure 10: Example of the signed distance results of the Kratos' process `CalculateDistanceToSkinProcess2D`.

5.3.3 Find the surrogate boundary

The next step uses the results of the level set function, which provides the signed distance between each node and the true boundary, for finding the cut elements and the surrogate boundary. The idea is simple: an element is cut by the skin boundary if at least one of its three nodes has `DISTANCE` > 0 and at least one node with `DISTANCE` < 0. This definition allows us to rapidly identify which are the cut elements and mark them using a flag. Flags are a tool implemented in Kratos that nodes or elements can have. For instance, we will mark the surrogate nodes, i.e. the nodes that form the surrogate boundary, as `BOUNDARY`. Flags allow us to rapidly identify a set of nodes or elements, for example, the set of cut elements or the set of surrogate nodes.

The process we have developed for computing the cut elements is called `FindSurrogateNodesProcess2D` and has the following structure:

```
for (auto &i_elem : rVolumePart.Elements()) {
    int count_pos = 0 ;
    int count_neg = 0 ;
    for (auto &i_node : i_elem.GetGeometry()) {
        auto phi = i_node.GetSolutionStepValue(DISTANCE) ;
        if (phi > 0) {
            count_pos = count_pos + 1 ;
        }else{
            count_neg = count_neg + 1 ;
        }
    }
    // When count_neg * count_pos != 0 --> the element is cut
    if (count_neg * count_pos != 0) {
        // The element is cut
        i_elem.Set(VISITED, true) ;
```

```
16          for (auto &i_node : i_elem.GetGeometry()) {
17              if (i_node.GetSolutionStepValue(DISTANCE) > 0) {
18                  i_node.Set(BOUNDARY, true);
19              }
20          }
21      }
22  }
```

As one can notice, the flags and the variables in Kratos are called in upper cases. The cut elements are highlighted using a flag called `VISITED` and the surrogate nodes, which will be crucial later on, are flagged as `BOUNDARY`.

Actually, the process `FindSurrogateNodesProcess2D` does more things. First, it deactivates the cut elements and the ones completely inside the skin boundary. It means that all elements that have at least one node with a negative distance are set to be non-active. In Kratos, one element is non-active to avoid assembling it when building the stiffness matrix and the residual vector of the problem. Moreover, it flagged as `BOUNDARY` all the elements that have at least one node that is a surrogate. This fact will be fundamental in Chapter 5.5 when we will introduce the Master-Point constraints (MPCs).

The shifted boundary is, at this point, completely identified in the context of the SB method. We now know the surrogate nodes since they are flagged. By connecting them in order, we can draw out the shifted boundary. But in the SB context, this is not necessary. In fact, the modified Dirichlet boundary conditions, that is some way we want to impose along the shifted boundary, will be imposed on the surrogate nodes and not on the surrogate boundary elements. Therefore, the process of identification of the surrogate boundary can stop here since what is needed further are only the surrogate nodes.
In Figure 11, an example is reported in which the algorithm is applied to the level set function results shown previously in Figure 10.

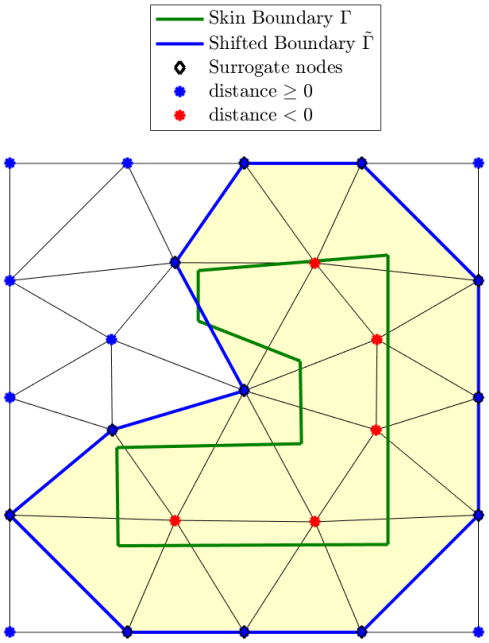


Figure 11: Example of the skin boundary (green) and the corresponding surrogate boundary (blue) in a very coarse mesh.

Note that in the algorithm just described, it is possible, for complicated geometry and very coarse mesh, that the process `FindSurrogateNodesProcess2D` may fail, in the sense of Figure 11. Indeed, one element is clearly cut by Γ, but since all its three nodes have a positive distance from the skin, the algorithm does not identify it as a cut element.

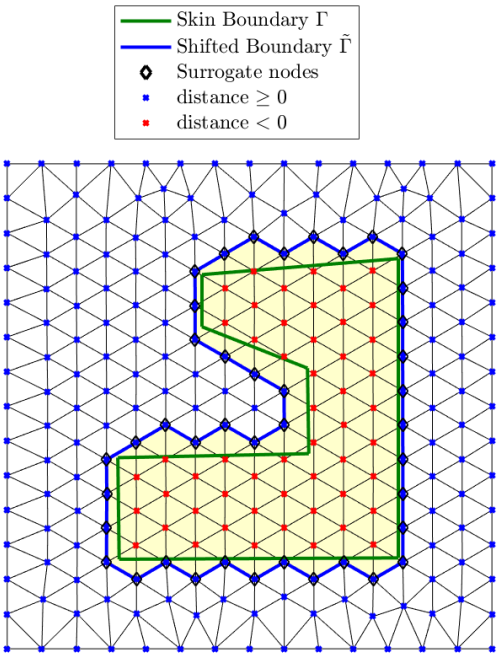


Figure 12: Example of a mesh refinement with respect to Figure 11.

Naturally, the mesh in figure 11 is exaggeratedly coarse, but by refining it, we expect that the algorithm gets a surrogate boundary that tends to the skin boundary. In other words, as $h \to 0$, we want that the surrogate boundary $\tilde{\Gamma}$ tends to coincide with the true boundary Γ. In Figure 12, we have an example of a refinement of the background mesh with respect to the one used in Figure 11, while it is used the same skin boundary. The new surrogate boundary has already considerably reduced the area between the skin boundary and the surrogate boundary. In this area, the SB method cannot capture the behaviour of the solution since the elements are deactivated. To carefully represent the phenomenon in the neighbourhood of the skin boundary are requested very refined meshes such that the cut elements are smaller. This fact is a common and well-known problem of all unfitted methods. It is usually solved by using local mesh refinements in the location of the embedded boundary.

5.4 Imposition of the Dirichlet BC

In this section, we discuss the core of the SB method, that is the imposition of the modified boundary conditions at the location of the surrogate boundary $\tilde{\Gamma}$.

A crucial point for keeping the second-order convergence estimate in the L^2-norm when using linear basis functions is the imposition of modified Dirichlet boundary conditions at the surrogate or shifted boundary. Naturally, we cannot impose the same Dirichlet value that we know at the true boundary. Otherwise, the expected convergence rate would be first-order since, in this case, we would just be shifting the boundary of at most h, keeping the same imposed

value. It just leads to a modification of the object's shape, which tends to be the exact shape as we refine the background mesh.

5.4.1 Taylor expansion

As said, the aim is to keep a second-order convergence. So, we have to introduce a modified value that will be imposed at the surrogate boundary nodes. Let us write the Taylor expansion of the generic scalar unknown function $u(\boldsymbol{x})$ around a generic point $\tilde{\boldsymbol{x}}$ and stop the expansion at the second-order term:

$$u(\boldsymbol{x}) = u(\tilde{\boldsymbol{x}}) + \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x} - \tilde{\boldsymbol{x}}) + o(|(\boldsymbol{x} - \tilde{\boldsymbol{x}})^2|) \tag{138}$$

Suppose now that $\tilde{\boldsymbol{x}}$ is one of the nodes of the surrogate boundary, so it is one of the nodes where we would like to impose the modified boundary conditions such that we commit an error proportional to h^2.
Consider now the concept of the closest point on the true boundary introduced in Chapter 2.2. We can state that for every surrogate node $\tilde{\boldsymbol{x}} \in \tilde{\Gamma}$ we can find at least one point $\boldsymbol{x_t} \in \Gamma$ that belongs to the true boundary such that:

$$|\boldsymbol{x_t} - \tilde{\boldsymbol{x}}| \leq h \tag{139}$$

As for the concept of the closest point, we may take the definition of the closest-point projection. Therefore, we do an orthogonal projection of the surrogate node $\tilde{\boldsymbol{x}}$ onto the true boundary Γ. Actually, in the finite elements world, the true boundary is a polyline; therefore, it will be even easier since we will only need to find the closest skin element and then find the orthogonal projection of $\tilde{\boldsymbol{x}}$ onto it.
Since the true boundary is a polyline, i.e. it is made by segments, there are cases where an orthogonal projection does not exist. In terms of implementation and application, this is not a problem; in fact, we only need a point belonging to the true boundary that satisfies Equation 139. Later the solution for this kind of issue will be treated.

So, once we have the closest-point to $\tilde{\boldsymbol{x}}$ that belongs to the true boundary Γ, we can rewrite the Taylor expansion in terms of $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x_t}$:

$$\bar{u}(\boldsymbol{x_t}) = u(\tilde{\boldsymbol{x}}) + \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) + o(|(\boldsymbol{x_t} - \tilde{\boldsymbol{x}})^2|) \tag{140}$$

Note that the term $\boldsymbol{x_t} - \tilde{\boldsymbol{x}}$ is exactly the definition of $\boldsymbol{d_M}$ given in Chapter 2.2.

One should keep in mind the initial objective of the Taylor expansion: impose a modified Dirichlet boundary condition at the surrogate node location such that the error related to this approximation is still second-order.
Suppose we have the location of the projection $\boldsymbol{x_t}$ and its Dirichlet exact value $\bar{u}(\boldsymbol{x_t})$ provided by the problem data. Then, we can express the approximated Dirichlet value at the surrogate node location $\tilde{\boldsymbol{x}}$ by rewriting Equation 140:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) + o(|(\boldsymbol{x_t} - \tilde{\boldsymbol{x}})^2|) \tag{141}$$

Therefore once we understand how to compute the gradient term and after neglecting the $o(|\boldsymbol{d}|^2)$, we can impose an approximated Dirichlet value at the surrogate node location. In theory, the error at the surrogate node that this approximation brings is proportional at least to h^2, with the same convergence velocity of the origin fem scheme.
Thus, the number of SB Dirichlet boundary conditions we will impose along the embedded

boundary is equal to the number of surrogate nodes computed with the algorithm in Chapter 5.3.3. On each of the surrogate boundary nodes, the following condition should be imposed:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot \boldsymbol{d} \tag{142}$$

In the next chapter, the description of how to compute the projection $\boldsymbol{x_t}$ for each surrogate node $\tilde{\boldsymbol{x}}$ will be discussed. After that will also explain how we can compute the gradient term evaluated at the surrogate node locations: $\nabla u|_{\tilde{\boldsymbol{x}}}$.

5.4.2 Computation of the closest-point

The computation of the closest-point belonging to the true boundary of a generic surrogate boundary node can seem an easy task, but there may be some cases that complicate the situation.
As said, a possible idea that also Professor Scovazzi used in his papers [26, 27], is to intend the concept of "closest-point" with the closest-point projection. That is, finding the orthogonal projection onto the closest true boundary element. The idea of the algorithm can be summarized as follows:

Algorithm 2 Find closest-point

```
for each surrogate node do
    Find the closest skin element that minimizes the distance between the surrogate
    node and the element.
    Compute the orthogonal projection of the surrogate node onto the skin element.
    Check that the projection lies inside the skin element.
    if the projection lies on the skin element then
        return projection
    else
        Take the second closest skin element that is close to the previous candidate
        element and repeat the procedure.
        Check that the projection lies inside this second skin candidate element.
        if the projection lies on the skin element then
            return projection
        else
            Use as closest-point projection the true boundary node between the two
            previous skin candidate elements.
            return node between the two candidates' element
```

The first step that individuates the closest skin element to the surrogate node we are considering is the more expensive in terms of computational cost. In fact, the idea that may be used is to compare the distance between the surrogate nodes and each of the skin elements. But this is very heavy, in particular for very complex immersed object shapes, since the number of skin elements needed for defining them properly may be huge. Especially, thinking about the application of this SB code for moving objects where the procedure of searching the surrogate boundary needs to be done at each iteration, it is necessary to find a different approach.
In Kratos' framework is present an optimized C++ function, called `BinBasedFastPointLocator`, which can work both in two and three dimensions. Given a point position, it returns the elements in which the point lies. It may be useful in our case to reduce the set of skin elements we want to compare in order to find the closest one.

Once the candidate skin closest element is selected, the computation of the projection is made

analytically with a small computational cost. First, is computed the slope m of the line where the skin element lies. Then the projection must be perpendicular to it. Thus, we can compute the line perpendicular to the skin element, i.e. with slope $-1/m$, that passes through the surrogate node. Finally, the algorithm computes the intersection between the two lines: this point is the candidate's orthogonal projection onto the skin element candidate.

```
for node in model_part.Nodes :
    if node.Is(BOUNDARY) :
        # 1.1 take its closest skin element --> closest_element[i]
        # 1.2 Get the two nodes
        node1 = skin_model_part.Conditions[closest_element[i]].GetNodes
            ()[0]
        node2 = skin_model_part.Conditions[closest_element[i]].GetNodes
            ()[1]
        # 1.3 Compute m and q
        m = (node1.Y - node2.Y) / (node1.X - node2.X)
        q = node1.Y - m * node1.X
        # 1.4 Compute Q
        Q = node.Y + 1/m * node.X
        projection_surr_nodes[i][0] = (Q-q) / (m+1/m)
        projection_surr_nodes[i][1] = m * projection_surr_nodes[i][0]+q
```

Where we have indicated with m and q the slope and the y-intercept of the line lying on the skin candidate element. Instead, Q has been used for indicating the y-intercept of the projection line that connects the surrogate node and the projection. Note that it is possible that $m = 0$ or $m = \infty$. It may happen when the skin element is parallel to the x-axis or y-axis, respectively. In this case, an if-loop has been added, and the projection can be computed straightforwardly.

So far, the procedure may seem direct, but there are some particular cases where two details may lead to wrong projections:

1. the true boundary elements can have different lengths;
2. the surrogate nodes do not have an orthogonal projection onto one of the skin boundary elements.

To overcome these issues we added two additional checks as one can see from the pseudo-code above. The first one is to check that actually, the candidate projection that we have found belongs to the skin element candidate. Checking it is possible by comparing the distances between the projection and the two skin element nodes; if both these distances are less than the length of the skin element, then the candidate projection is the correct one.

If the candidate projection does not lie on the candidate element, there could be two possibilities: either the projection is on the second closest element (due to different lengths of the skin elements), or a projection does not exist. Thus we simply apply the same procedure to the second closest element and if the projection is not suitable, it implies the projection does not exist. In this scenario, we are in a situation similar to Figure 14. In this last case, there are no problems since in the SB method what matters is that the vector connecting the projection $\boldsymbol{x_s}$ and the surrogate node $\tilde{\boldsymbol{x}}$ is shorter than h. Such that the Taylor expansion of Equation 142 still introduces at most a second-order error in h.

In Figure 13, we can see an example of an embedded object which has a circular boundary Γ. In black, we can see the embedded true boundary Γ, and in blue dots the surrogate boundary nodes. As said, the surrogate boundary nodes are the closest external point. Moreover, the

projection operator (in red) finds the projections underlined in purple in the figure.

It may happen that close to the corners, there are surrogate boundary nodes that do not have any orthogonal projections onto the true boundary. Thus, in these cases, it is set in the algorithm to take as the projection the corner which actually is the closest point.

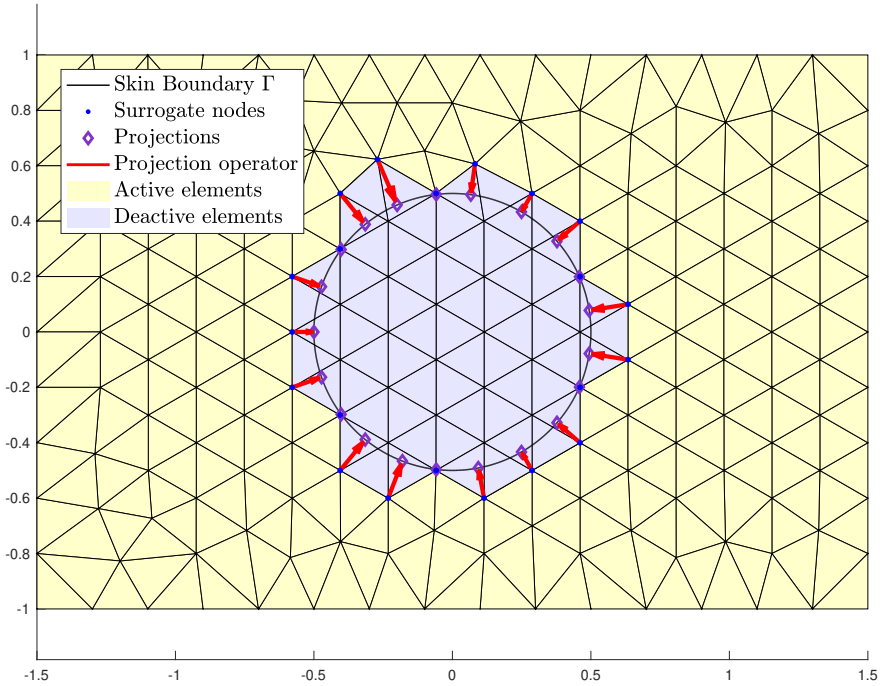


Figure 13: Example of a circular embedded boundary. The results of the projections of the surrogate boundary nodes onto the true boundary Γ are shown.

For instance, Figure 14 reports a case where the true boundary Γ has corners and in correspondence with some of them, an orthogonal projection does not exist. The projection operator points directly to the corner, which is also the closest point on Γ.

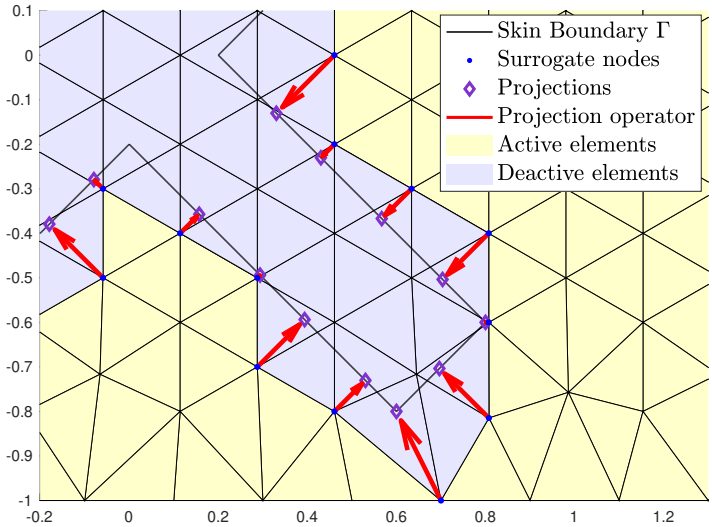


Figure 14: Example of an embedded boundary with corners. The lower surrogate node does not have any orthogonal projection onto the true boundary Γ.

5.4.3 Computation of the gradient term

At this point of the implementation, we have for each of the surrogate boundary nodes its "closest point" belonging to the true boundary. We remind that the initial objective was to compute the Dirichlet value to be imposed at the surrogate boundary node locations, which we want to write with Equation 141. That, neglecting the second-order error, reads as follows:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) \tag{143}$$

In the previous subsection, we have computed the coordinates of $\boldsymbol{x_t}$, and by the data of the problem $\bar{u}(\boldsymbol{x_t})$ is known. In the example of a CFD problem the scalar function u can represent the x-component of the velocity field; in the hypothesis of no-slip interface condition, the value of $\bar{u}(\boldsymbol{x_t})$ will constantly be equal to zero if the immersed object is motionless.
Therefore, what is missing now is the computation on the gradient term: $\nabla u|_{\tilde{\boldsymbol{x}}}$, that is the gradient of the unknown function $u(\boldsymbol{x})$ evaluated at the surrogate boundary node location. Its computation is not a straightforward operation because u is actually our unknown function of the problem.

The most intuitive option is to add a non-linear loop for each time-step iteration. In fact, for computing the gradient, one may use the value of the scalar field u at the previous iteration; this is a well-known procedure in finite elements taking the derivative of the basis functions on each element.
It means that each time iteration will require some sub-iterations to satisfy the tolerance that has been set. This idea has been implemented and, in the cases tested, the method works nicely, but, in general, it requires additional iterations.

The risk of this formulation is that when strong gradients are present, we may lead to a wrong value of ∇u. Moreover, we have tested this method for solving stationary diffusion problems (Poisson problem of Chapter 3) and it gives correct results but just after has been done many iterations so that the gradient term ∇u at the surrogate node locations has converged to the exact value. Thus, already in the Poisson stationary problem, where using a body-fitted approach one iteration is needed, the SB solver requires 15/20 iterations in order to have a global error of the same order of magnitude as the body-fitted approach.

In conclusion, one can argue that the SB method using the gradient term at the previous iteration is not robust a priori and needs more iterations to converge. We must find an alternative way that ensures robustness and that does not require additional iterations. In the next chapter will be discussed a novel method for reconstructing the gradient term and using it in the imposition of the Dirichlet shifted boundary conditions.

5.5 Multi-point constraints in SB method

The alternative that represents the novelty introduced in this thesis is the use of the so-called Multi-Point Constraints (MPCs) to compute the gradient term $\nabla u|_{\tilde{\boldsymbol{x}}}$ of Equation 143.
Also known as master-slave constraints, the MPCs will be used to enforce the Dirichlet boundary conditions on each surrogate node. In synthesis MPCs allows us to introduce extra kinematic constraints to the problem. These kinematic constraints are obtained from the modified Dirichlet boundary conditions of the SB method. Moreover, once the MPCs are set we don't need to do any non-linear loop for computing the gradient term because its contribution is already in the MPCs. Therefore, for a stationary problem, we will expect to find the FEM solution in one

iteration.
This novel method will solve the two problems discussed in the previous chapter; in fact, it ensures robustness and maintains a low computational cost computing the gradient $\nabla u|_{\tilde{\boldsymbol{x}}}$ and the solution u at the same time.

Master-point constraints are usually used in structural simulations to control the position of one object in relation to another. The object controlled is called the "slave" object, and the object that it is controlled by is called the "master" object.
An easy example of a master-slave constraint in 2D might be a simple drawing program where the user can draw a line between two points. The position of the second point (the slave object) is controlled by the position of the first point (the master object). As the user moves the first point, the second point follows along, keeping the line between the two points straight. This is an example of a "point-to-point" master-slave constraint.
Another application is when simulating mechanical assemblies, where one component (the master) is responsible for driving the motion of another (the slave). For example, a piston in an engine might be modelled as the master, with the connecting rod as the slave. Then, the motion of the connecting rod would be constrained to follow the motion of the piston. In the next sections, we will review the theory of MPCs in the finite element method. Moreover, it will be shown how this powerful tool can help the SB method in the calculation of the gradient term.

5.5.1 MPCs

To simply explain what MPCs are and how they work, consider an easy example taken by the book of Professor Felippa [18].

Consider the structural problem where one wants to add a constraint involving the displacement of two nodes. Suppose we have a bar in one dimension discretized in seven nodes, and the unknown variable is only the x-displacement. In Figure 15, we report a draw for a better understanding.

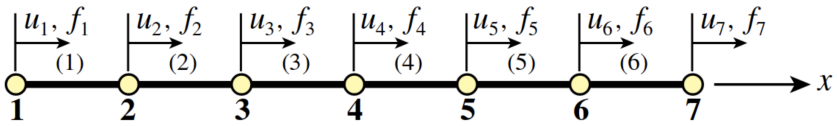


Figure 15: Bar in one dimension discretized in seven nodes, image taken from [18].

This structure consists of six bar elements and seven nodes that can only displace in the horizontal direction. Before imposing any MPCs, as it is well known, the stiffness matrix and the FEM linear system for this problem are the following:

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 & 0 \\ K_{21} & K_{22} & K_{23} & 0 & 0 & 0 & 0 \\ 0 & K_{32} & K_{33} & K_{34} & 0 & 0 & 0 \\ 0 & 0 & K_{43} & K_{44} & K_{45} & 0 & 0 \\ 0 & 0 & 0 & K_{54} & K_{55} & K_{56} & 0 \\ 0 & 0 & 0 & 0 & K_{65} & K_{66} & K_{67} \\ 0 & 0 & 0 & 0 & 0 & K_{76} & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} \tag{144}$$

That, in a shorter notation, we can write as:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \tag{145}$$

Now we define a multi-point constraint that states that nodes 2 and 6 must move by the same amount. One might express this relation in terms of the following equation:

$$u_2 = u_6 \tag{146}$$

This condition can be read as an MPC. We can consider one of the two degrees of freedom involved as the slave and the other as the master. In synthesis, the MPC allows us to add Equation 146 to the linear system 144. Moreover, it also reduces the vector of the unknowns by removing the slave degree of freedom. In fact, the slave degree of freedom u_6 is completely defined by Equation 146 that says that $u_2 = u_6$.

We now go further in the process of actually imposing a set of master-point constraints. First, the MPCs are taken one at a time; for each constraint, a slave degree of freedom is chosen. The freedoms remaining in that constraint are labelled master indeed we may have more than one master degree of freedom.
Then, a new set of degrees of freedom $\hat{\boldsymbol{u}}$ is established by removing the slave degree of freedom from $\boldsymbol{u}$. This new vector of unknowns $\hat{\boldsymbol{u}}$ contains the master freedoms as well as those that do not appear in the MPC. Moreover, a transformation matrix $\boldsymbol{T}$ that relates $\boldsymbol{u}$ to $\hat{\boldsymbol{u}}$ is generated; this transformation is also used to apply a congruential transformation to the stiffness matrix. This procedure yields a set of modified stiffness equations which are expressed in terms of the new vector of degrees of freedom $\hat{\boldsymbol{u}}$. Thus, in the end, we should obtain a modified system that does not contain the slave degree of freedom, since it has been effectively eliminated, and the new linear system should take into account the original MPC equation.

The mechanics of the process is best seen by going through the previous example where the MPC we want to apply is the one in Equation 146. Consider the degree of freedom 6 as the slave one (the one that will be removed) and the degree of freedom 2 as the master one. Therefore at the end of the procedure, we want to have a relation between $\boldsymbol{u}$ and $\hat{\boldsymbol{u}}$, where the new vector of unknowns $\hat{\boldsymbol{u}}$ is defined as:

$$\hat{\boldsymbol{u}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} \tag{147}$$

Note that u_6 is not present. Thus, we can find the linear transformation $\boldsymbol{T}$ such that:

$$\boldsymbol{u} = \boldsymbol{T}\hat{\boldsymbol{u}} \tag{148}$$

For the multi-point constraint of the example we are treating, that is Equation 146, the transformation $\boldsymbol{T}$ is exactly the following:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} \tag{149}$$

Therefore now the unknown new vector $\hat{\boldsymbol{u}}$ has 6 degrees of freedom, but the sixth equation in the linear system of Equation 149 reads as our MPC:

$$u_6 = u_2$$

At this point, we can come back to our FEM linear system $\boldsymbol{Ku} = \boldsymbol{f}$ and replacing $\boldsymbol{u}$ using Equation 148, we get the following:

$$\boldsymbol{Ku} = \boldsymbol{KT\hat{u}} = \boldsymbol{f} \tag{150}$$

And to obtain a square linear system, we can left-multiply both sides by $\boldsymbol{T^T}$:

$$\boldsymbol{T^T KT\hat{u}} = \boldsymbol{T^T f} \tag{151}$$

We can rewrite it in a more compact notation suitable for defining the matrix $\hat{\boldsymbol{K}}$ and the forcing $\hat{\boldsymbol{f}}$:

$$\hat{\boldsymbol{K}}\hat{\boldsymbol{u}} = \hat{\boldsymbol{f}} \tag{152}$$

Now the final linear system is 6×6 instead of 7×7 and it ensures that the MPC holds. We may report the explicit modified linear system of equation $\hat{\boldsymbol{K}}\hat{\boldsymbol{u}} = \hat{\boldsymbol{f}}$ containing the MPC requested:

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 \\ K_{21} & K_{22}+K_{66} & K_{23} & 0 & 0 & 0 \\ 0 & K_{32} & K_{33} & K_{34} & 0 & 0 \\ 0 & 0 & K_{43} & K_{44} & K_{45} & 0 \\ 0 & K_{56} & 0 & K_{54} & K_{55} & 0 \\ 0 & K_{76} & 0 & 0 & 0 & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2+f_6 \\ f_3 \\ f_4 \\ f_5 \\ f_7 \end{bmatrix} \tag{153}$$

Probably it is still unclear how the concept of MPCs can help in the SB method but is essential to say that we can generalize the example we have just reported for a general equation that relates different degrees of freedom. For instance, consider again the problem in Figure 15 and an equation that involves more than two degrees of freedom, for example, the following:

$$Au_1 + Bu_2 + Cu_3 + Du_4 = 0 \tag{154}$$

In this case, an MPC condition can still be imposed on the linear system 144. In fact, we have to choose one degree of freedom to be the slave, i.e. the one that will be removed, and the remaining ones will be considered masters. Taking u_1 as slave, Equation 154 can be written as follows:

$$u_1 = -\frac{B}{A}u_2 - \frac{C}{A}u_3 - \frac{D}{A}u_4 = bu_2 + cu_3 + du_4 \tag{155}$$

Then, the transformation $\boldsymbol{T}$ associated to this MPC is the following:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} b & c & d & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \boldsymbol{T\hat{u}} \tag{156}$$

Where we have generalized the method in such a way the row associated with the slave degree of freedom turns out to be the MPC equation. The modified and reduced linear system is represented by Equation 151 that is:

$$\boldsymbol{T^T KT\hat{u}} = \boldsymbol{T^T f}$$

5.5.2 MPCs in Kratos

MPCs in Kratos have already been efficiently implemented mostly by Professor Riccardo Rossi and the method for using them is called `CreateNewMasterSlaveConstraint`. There can exist non-linear master point constraints where the MPC equation is non-linear, but they are not necessary for this context. We report the piece of code that creates new MPCs in Kratos:

```
main_model_part.CreateNewMasterSlaveConstraint("
    LinearMasterSlaveConstraint", j, DofMasterVector, DofSlaveVector,
    CoeffMatrix, ConstantVector)
```

where j is the id of the MPC we are creating, `DofSlaveVector` is the slave degree of freedom, `DofMasterVector` is the set of the master degrees of freedom (one or more), `CoeffMatrix` is a matrix containing the coefficients that describe the MPC equation.
As we will see later on, the `ConstantVector` is fundamental in the applications of MPCs to the SB method; it allows the addition of a constant term to the MPC equation. For instance, instead of the MPC Equation 154, we can impose the following MPC equation:

$$Au_1 + Bu_2 + Cu_3 + Du_4 = E \tag{157}$$

where E is simply a scalar.

The last Kratos tool regarding the MPCs, which will be used during the next chapters, is the possibility of erasing one or more MPCs.

```
## Remove MPCs
for constraint in main_model_part.MasterSlaveConstraints :
    constraint.Set(KratosMultiphysics.TO_ERASE)
main_model_part.RemoveMasterSlaveConstraintsFromAllLevels(
    KratosMultiphysics.TO_ERASE)
```

At this point, we have reviewed the MPC theory and the Kratos tools we need for applying MPCs to the computation of the gradient terms in the SB method.

5.5.3 MPCs for SB method

At this point, one may wonder why MPCs can be useful for imposing modified Dirichlet boundary conditions at the surrogate boundary nodes in the SB method.
Recall the modified Dirichlet conditions one would like to impose at the surrogate boundary nodes. Let us report Equation 143:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}})$$

Where what was causing the problems discussed in Chapter 5.4.3 is the computation of the gradient term at the surrogate boundary node location: $\nabla u|_{\tilde{\boldsymbol{x}}}$. In the finite element method, the gradient value of the unknown scalar variable u can be expressed as a linear combination of the u values of the neighbouring nodes. Note that a second-order error estimate is still guaranteed if we use the nodes that share at least one element with the node we are considering. Indeed we are using a first-order approximation for writing a second-order term of the Taylor expansion, that is, the gradient term. These neighbouring degrees of freedom are the so-called first-level neighbours.

In other words, we can write the following equality:

$$\nabla u|_{\tilde{\boldsymbol{x}}} = \boldsymbol{G}\boldsymbol{u_n} \tag{158}$$

Where $\boldsymbol{G}$ is a linear transformation containing the contribution coefficients of the different neighbours. The size of $\boldsymbol{G}$ depends on the number of neighbouring nodes of the surrogate node $\tilde{\boldsymbol{x}}$. Since in a two-dimensional problem, the gradient is a vector with two components, the number of rows of $\boldsymbol{G}$ will always be two, and the number of columns will depend on how many neighbours we consider.
Consider Figure 16, which represents a small portion of the mesh surrounding node with id=2. Suppose that node with id=2 is a surrogate boundary node, and therefore we have to compute

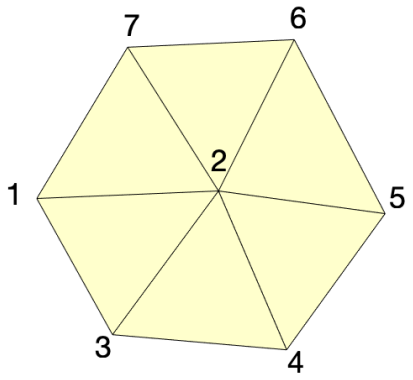


Figure 16: Example of surrounding elements of an internal node.

the gradient of the unknown function u at the $\boldsymbol{x_2}$ location: $\nabla u|_{\boldsymbol{x_2}}$. Then, using the basis functions defined on the neighbouring elements, we can obtain the following:

$$\nabla u|_{\boldsymbol{x_2}} = \begin{pmatrix} \frac{\partial u}{\partial x}\big|_{\boldsymbol{x_2}} \\ \\ \frac{\partial u}{\partial y}\big|_{\boldsymbol{x_2}} \end{pmatrix} = \begin{pmatrix} G_{1,x} & G_{2,x} & G_{3,x} & G_{4,x} & G_{5,x} & G_{6,x} & G_{7,x} \\ \\ G_{1,y} & G_{2,y} & G_{3,y} & G_{4,y} & G_{5,y} & G_{6,y} & G_{7,y} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \tag{159}$$

In this case, note that $\nabla u|_{\boldsymbol{x_2}}$ also depends on u_2 itself; this fact will imply an additional implementation later. First of all, let us understand how we can compute the coefficients $G_{i,x}$ and $G_{i,y}$ that compose the matrix that has been called $\boldsymbol{G}$ in Equation 158.
Basically, to compute the $\nabla u|_{\boldsymbol{x_2}}$ of Figure 16, we first find the neighbouring elements. Then, for each of these elements, we compute the gradient of the three linear basis functions. Moreover, we integrate over the element these gradients using the Gauss points. At this point, we have the gradient contribution in $\boldsymbol{x_2}$ due to the first element as a function of its three nodes.
Then we do the same for each neighbouring element and weigh the node contribution based on how many elements it is present. For instance, all the nodes, except for node with id=2, are present in two neighbouring elements. Node with id=2 instead belongs to all the six neighbouring elements. In Appendix 8.1, we have reported the corresponding code that can be used in Kratos to obtain the coefficients in matrix $\boldsymbol{G}$ in terms of an unordered map. The C++ function is called `ShiftedBoundaryMeshlessInterfaceUtility`.

Therefore, we have available the matrix $\boldsymbol{G}$ for each of the surrogate boundary nodes. It might be clear how to use these matrices $\boldsymbol{G}$'s to impose the Dirichlet shifted boundary conditions. Let us write Equation 143 and substitute Equation 158:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot \boldsymbol{d} = \bar{u}(\boldsymbol{x_t}) - \boldsymbol{G u_n} \cdot \boldsymbol{d} \tag{160}$$

Where $\boldsymbol{u_n}$ is the set of neighbours degrees of freedom of $\tilde{\boldsymbol{x}}$. $\boldsymbol{x_t}$ is the projection of $\tilde{\boldsymbol{x}}$ onto the true boundary and $\bar{u}(\boldsymbol{x_t})$ was supposed to be known. In the example of Figure 16, the node with id=2 is considered as a surrogate boundary node ($\tilde{\boldsymbol{x}} = \boldsymbol{x_2}$). Then, the previous equation can be rewritten as follows:

$$u_2 = \bar{u}(\boldsymbol{x_t}) - \boldsymbol{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \cdot \boldsymbol{d} = \bar{u}(\boldsymbol{x_t}) - \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \cdot (\boldsymbol{G^T d}) \tag{161}$$

Where $\boldsymbol{G^T d}$ is a vector 7×1 and is completely known since it depends only on geometric mesh quantities. In fact, $\boldsymbol{G}$ is composed of coefficients that depend on the neighbouring degrees of freedom, and $\boldsymbol{d}$ is the distance vector between $\boldsymbol{x_t}$ and $\boldsymbol{x_2}$.
We indicate $\boldsymbol{G^T d} = \tilde{\boldsymbol{g}} = (\tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4, \tilde{g}_5, \tilde{g}_6, \tilde{g}_7)^T$ to simplify the notation; note that the Equation 161 it is not ready to be used for imposing a linear MPC, in fact in the right-hand side we have the dependency on u_2 itself. Rearranging, we get:

$$u_2 + u_2\tilde{g}_2 = \bar{u}(\boldsymbol{x_t}) - \begin{pmatrix} u_1 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \cdot \begin{pmatrix} \tilde{g}_1 \\ \tilde{g}_3 \\ \tilde{g}_4 \\ \tilde{g}_5 \\ \tilde{g}_6 \\ \tilde{g}_7 \end{pmatrix} \tag{162}$$

That implies that the modified Dirichlet value u_2 can be imposed as an MPC, and the corresponding MPC equation reads as:

$$u_2 = \frac{1}{1+\tilde{g}_2} \left[\bar{u}(\boldsymbol{x_t}) - \begin{pmatrix} u_1 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \cdot \begin{pmatrix} \tilde{g}_1 \\ \tilde{g}_3 \\ \tilde{g}_4 \\ \tilde{g}_5 \\ \tilde{g}_6 \\ \tilde{g}_7 \end{pmatrix} \right] \tag{163}$$

Where the $\tilde{g}_i$'s are all known coefficients, $\bar{u}(\boldsymbol{x_t})$ is also known and is called kinematic term or constant term, u_2 is the slave degree of freedom and $(u_1, u_3, u_4, u_5, u_6, u_7)$ are the masters.

5.5.4 Additional problems

So far, it seems that everything is ready for implementation in the Kratos Multiphysics framework, but two problems need to be solved. The first problem is in the computation of the transformation $\boldsymbol{G}$, which allows us to compute the gradient. Before we considered that all the

neighbouring elements were available. But since u_2, in Figure 16, is supposed to be a surrogate node, it implies that some of its neighbouring elements will naturally be cut. It means that these cut elements are partially inside the immersed object and they are completely out of the simulations. Thus they can not be considered as contributions for the computation of $\nabla u|_{\tilde{\boldsymbol{x}}}$.

The second problem is that a slave degree of freedom (such as u_2 in Figure 16) cannot be a master degree of freedom for another MPC, indeed when the MPC for u_2 is set, the degree of freedom u_2 is removed from the vector of unknowns. It is a critical point for our novel SB method; in fact, u_2 will have for sure as neighbours at least two surrogate nodes, but their elemental contribution cannot be taken into account in the MPC since they will be slave degrees of freedom for their MPCs, since they also are surrogate node and they need an MPC for imposing their modified Dirichlet boundary condition.

In order to solve the two problems just mentioned, we introduce some modifications related to the MPCs described in the previous section. Consider Figure 17 in which some details have been added with respect to Figure 16. As one can see in green is highlighted the skin boundary that cuts the three elements, which are coloured light yellow.

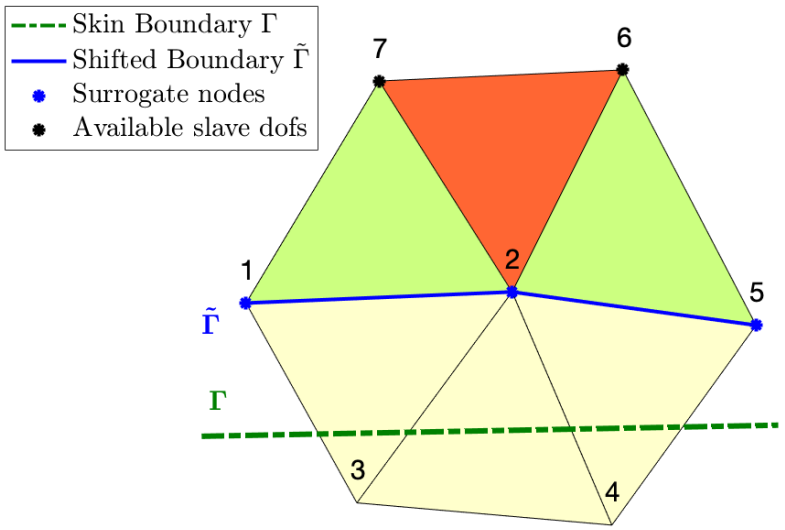


Figure 17: First-level neighbouring degrees of freedom of the surrogate node with id=2.

Moreover, in blue are indicated the surrogate boundary nodes we have detected using the algorithm previously described.
Suppose again we want to find the MPC equation representing the imposition of the modified Dirichlet SB boundary condition in the node having id=2. For the problems previously mentioned, we cannot use the yellow elements in Figure 17 since they have been set as non-active. So, u_3 and u_4 cannot be used. Moreover, degrees of freedom u_1 and u_5 neither can be used because they are surrogate boundary nodes. Therefore u_1 and u_5 are slave degrees of freedom for other MPCs. Thus, the green neighbouring elements cannot be used for computing the gradient coefficients. Only the orange one is available and, therefore, the two available master degrees of freedom for this MPC are u_6 and u_7.
Equation 163, using only the available degrees of freedom, becomes:

$$u_2 = \frac{1}{1+\tilde{g}_2}\left[\bar{u}(\boldsymbol{x_t}) - \begin{pmatrix} u_6 \\ u_7 \end{pmatrix} \cdot \begin{pmatrix} \tilde{g}_6 \\ \tilde{g}_7 \end{pmatrix}\right] \tag{164}$$

The fact that we are considering just one out of six elements for computing the gradient in $\boldsymbol{x_2}$ does not cause any problem, as it will be shown in Chapter 6 where the validation of this method is discussed.

The algorithm that computes the gradient explained in Chapter 5.5.3 can be slightly modified by adding additional checks. The checks are made on the set of neighbouring elements to be used for computing the gradient; a first check is a control that the element is active, such that we are sure that the element is not cut (like the yellow elements in Figure 17); a second check on the elements is done by controlling that they do not have as node any other surrogate boundary nodes (green elements in Figure 17) otherwise the element is useless.

Unfortunately, there exist special cases where there are no available neighbouring elements for computing the gradient term. Consider for instance Figure 18. In this case, there are four out of six elements that are cut by the skin boundary (elements marked in yellow). Moreover, the remaining two contain respectively the degree of freedom of the node with id=1 (u_1) and of the node with id=6 (u_6), which are surrogate nodes.

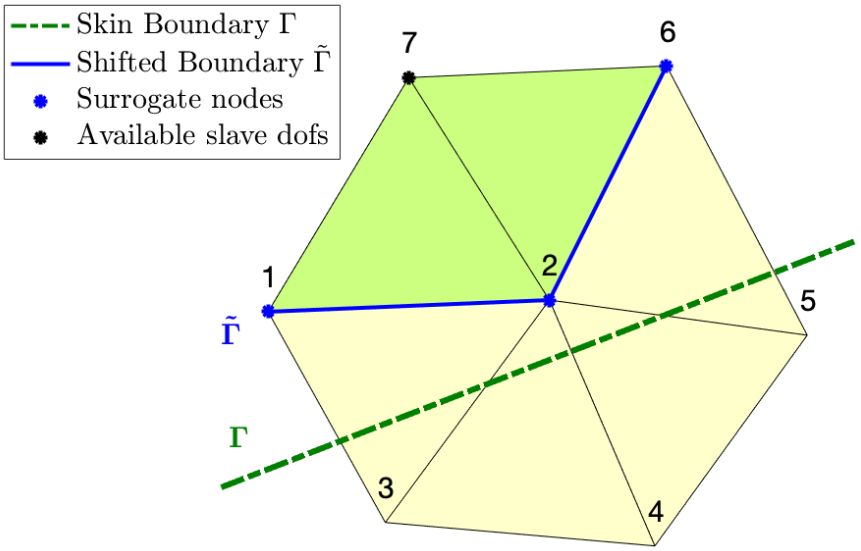


Figure 18: First-level neighbouring degrees of freedom of the surrogate node with id=2. In this case, there are no available first-level neighbouring elements for computing $\boldsymbol{G}$.

The phenomenon in Figure 18 may happen in practical applications where no elements are available. What has been implemented to solve this problem is something that may reduce the rate of error convergence. Since no elements are available for computing the gradient evaluated in $\tilde{\boldsymbol{x}}_{\boldsymbol{2}}$, what we can do is use what we call the second-level neighbouring elements. It means that we consider the elements that do not have the node with id=2 as one of the nodes, but the elements that have at least one of the neighbouring nodes of the node having id=2 as a node. From Figure 18, imagine taking a second layer of elements around the node with id=2. Then, we expect that some of these elements are available. Available in the sense that they are neither cut nor inside the immersed object, and they don't own any other surrogate nodes.

Figure 19 shows yellow elements that are either the cut or elements completely inside the immersed object, the green ones contain at least one surrogate node different from the node with id=2, and the orange elements are the available ones.

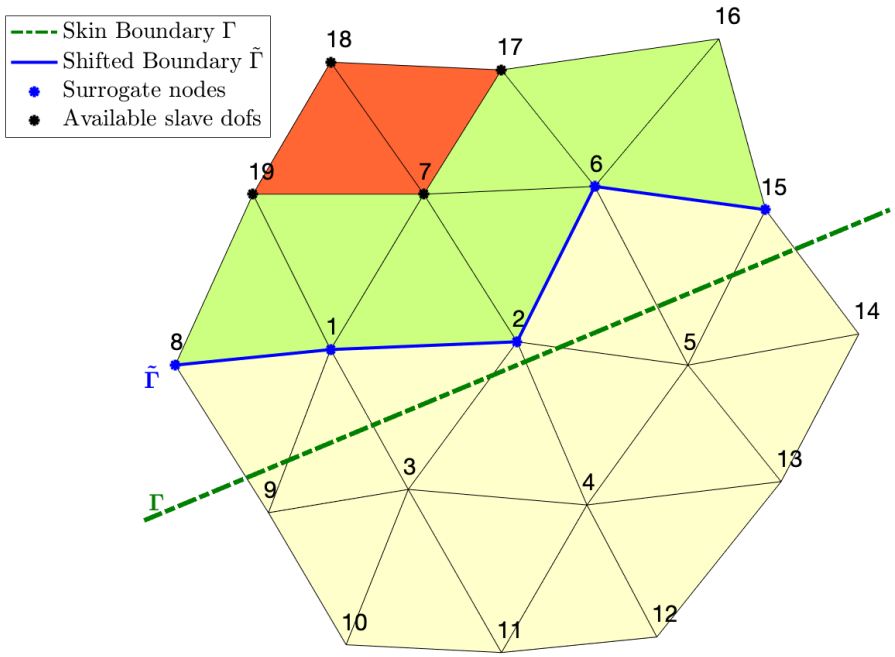


Figure 19: First-level and second-level neighbouring elements of the surrogate node with id=2. In this case, there are no available first-level neighbouring elements for computing $\boldsymbol{G}$, but two second-level neighbouring elements (orange) are available.

As we can see, we may use four degrees of freedom (u_7, u_{17}, u_{18} and u_{19}) for computing the gradient at $\tilde{\boldsymbol{x}}_{\boldsymbol{2}}$, the best MPC equation that we can provide for the modified Dirichlet boundary condition for u_2 is the following:

$$u_2 = \bar{u}(\boldsymbol{x_t}) - \begin{pmatrix} u_7 \\ u_{17} \\ u_{18} \\ u_{19} \end{pmatrix} \cdot \begin{pmatrix} \tilde{g}_7 \\ \tilde{g}_{17} \\ \tilde{g}_{18} \\ \tilde{g}_{19} \end{pmatrix} \tag{165}$$

Differently from Figure 17, we are not only using the degrees of freedom in the first-level of neighbouring elements, but instead, we are using also u_{17}, u_{18} and u_{19} that are distant $2h$ from the node with id=2. It will lead to a slight lack of precision in the imposition of the modified Dirichlet boundary condition. The quantification of this additional error may lead to a non-optimal convergence error estimate, but it is discussed in Chapter 6.

In Chapter 6.4, where complicated skin boundaries are considered, the SB algorithm may need additional element layers because there can be cases where even in the second layer of elements there are no available ones. It happens only when non-convex object shapes are taken into account as skin boundaries. In our algorithm the already mentioned C++ function `ShiftedBoundaryMeshlessInterfaceUtility` has been modified. If no first-level neighbouring elements are available, then we check if elements in the second layer are available. If neither is the second layer we go to the third, and so on.

In the following, we report the steps that explain how it is possible in Kratos to obtain the coefficients $\tilde{g}_i$ for each of the MPC equations that are needed for imposing the shifted Dirichlet boundary conditions.

1. Apply the Kratos process for finding the map of the neighbouring nodes of each node.
2. Apply the Kratos process for finding the map of the neighbouring element of each node.
3. Apply the `ShiftedBoundaryMeshlessInterfaceUtility`.
4. Obtain the matrices $\boldsymbol{G}$ and the vector of available neighbouring degrees of freedom $\boldsymbol{u_n}$ for each of the surrogate node.
5. Compute the distance vector $\boldsymbol{d} = \boldsymbol{x_t} - \tilde{\boldsymbol{x}}$ between the surrogate node and the projection.
6. Compute the vectors $\tilde{\boldsymbol{g}}$'s by the following matrix multiplication: $\tilde{\boldsymbol{g}} = \boldsymbol{G^T d}$

The first three steps can be achieved by the following piece of code present in the solver:

```
# Find the neighbour nodes map
KratosMultiphysics.FindGlobalNodalNeighboursProcess(main_model_part).
    Execute()
# Find the neighbour elements map
KratosMultiphysics.GenericFindElementalNeighboursProcess(main_model_part
    ).Execute()
# Computations of coefficients and dofs for computations of the gradient
a = KratosMultiphysics.ShiftedBoundaryMeshlessInterfaceUtility(model,
    Parameters)
result = a.SetSurrogateBoundaryNodalGradientWeights()
```

The `SetSurrogateBoundaryNodalGradientWeights` function outputs an unordered map that, for each surrogate node, it returns the degrees of freedom $\boldsymbol{u_n}$ and the coefficients of the matrix $\boldsymbol{G}$ such that:

$$\nabla u|_{\tilde{\boldsymbol{x}}} = \boldsymbol{G u_n} \tag{166}$$

The function `SetSurrogateBoundaryNodalGradientWeights` also contains the search in an additional layer of neighbouring elements if no elements are available. That is what happened in Figure 19.

Then, in order to have the vector $\tilde{\boldsymbol{g}} = \boldsymbol{G^T d}$ of Equation 161, we have to get the distance vector $\boldsymbol{d}$ that connects the surrogate node to its projection computed in Chapter 5.4.2.

```
i = 0
for node in surrogate_sub_model_part.Nodes :
    # Create the G matrices: 0 , 1 , ... , len(boundary_sub_model_part.
        Nodes)-1
    name_g_tilde = "g_tilde_" + str(i)
    # Get the dofs and the coefficients of G
    result_surr_node = result[node.Id]
    G_matrix = [[0 for _ in range(len(my_result))] for _ in range(2)]
    j = 0
    for key, value in result_surr_node.items():
        G_matrix[0][j] = value[0]
        G_matrix[1][j] = value[1]
        j = j + 1
    # Get the distance vector using the projections
    d_vector = [[0 for _ in range(1)] for _ in range(2)]
    d_vector[0] = projection_surr_nodes[i][0] - node.X
    d_vector[1] = projection_surr_nodes[i][1] - node.Y
```

```
17      # Matric multiplication G^T d
18      g_tilde = np.matmul( np.transpose(G_matrix) , d_vector )
19      globals()[name_g_tilde] = g_tilde
20      i = i + 1
```

In this way, we create the vectors $\tilde{\boldsymbol{g}}$'s that in number are equal to the number of surrogate nodes and can be recalled by the notation `g_tilde_0`, `g_tilde_1`, etc.

Finally, the MPCs can be imposed based on the coefficients $\tilde{g}_i$'s and the vector of available degrees of freedom $\boldsymbol{u_n}$. In Appendix 8.2, we report the algorithm we have used in the SB solver that allows imposing all the MPCs.

In conclusion, the imposition of the Dirichlet boundary conditions at the surrogate boundary through the use of MPCs solves the problem of computing the gradient term. In fact, using the MPCs, we don't explicitly compute the gradient of u at the surrogate boundary nodes, but we simply add to the FEM linear system as many equations as the number of surrogate nodes. These equations are the MPC equations that allow us to ensure the Dirichlet conditions of Equation 143 for each of the surrogate boundary nodes.

5.6 Flux through the surrogate boundary

What has been discussed so far has been basically introduced in the SB solver of Kratos, but still, one step is missing for having a working SB method implemented in Kratos with the correct convergence.
We have said that the surrogate boundary $\tilde{\Gamma}$ has become the new boundary of our computational mesh. In fact, the cut elements are completely taken out from the simulation being set as non-active. If we impose the modified Dirichlet boundary conditions in a standard way, i.e. without using the MPCs, nothing additional is required. In this case, we would be classically fixing the unknown variable along the surrogate boundary nodes, and the flux of the unknown scalar quantity through this boundary is automatically correct. In fact, the flux through Dirichlet boundaries is such that the Dirichlet conditions are satisfied.
Instead, when the MPCs are used for imposing the shifted boundary conditions, we are not actually imposing any Dirichlet boundary conditions. As we saw in the previous chapters, we are only adding equations to the linear system. Thus, is the flux through the surrogate boundary imposed in such a way the modified Dirichlet boundary conditions are satisfied? At this point of the implementation, the answer is no.

When the domain is cut, either if MPCs are used instead of classical Dirichlet boundary conditions or not, the flux through the surrogate boundary is not related to the values we want to impose at the surrogate boundary nodes. It is also clear from the weak formulations we have treated in chapters 3 and 4. Consider, for instance, the symmetric bilinear form of the Poisson problem when we have cut the computational domain Ω with the surrogate boundary $\tilde{\Gamma}$; in Equation 29 the following boundary term arose:

$$- \langle w^h, \nabla u^h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_D} \tag{167}$$

This term is the classical boundary term in finite elements, that is relevant only on the boundary of the domain. In fact, between internal element edges, this term exists but is perfectly balanced. Therefore, it is usually simplified. The external flux of the general scalar unknown function u

from an internal element edge is equal to the external flux through the same edge of the neighbour element, see Figure 20.

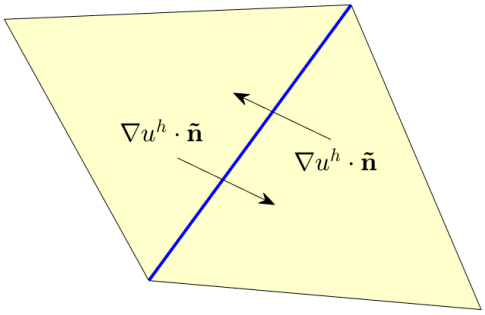


Figure 20: Flux of u^h between an internal element edge.

In the SB method, we are actually cutting the domain with the surrogate boundary. It implies that the internal flux term, which is usually balanced between internal element edges, is not balanced along the surrogate boundary because outside this boundary everything is turned off. If we don't add the term of Equation 167 along the surrogate boundary, it means we are imposing a zero flux of u at $\tilde{\Gamma}$.

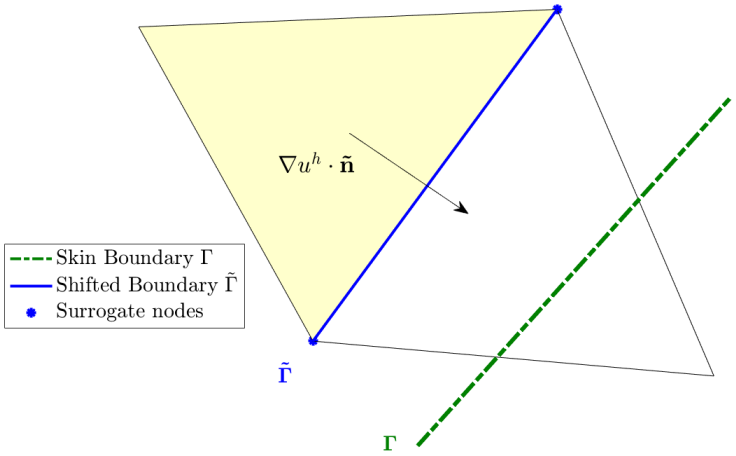


Figure 21: Flux of u^h between an edge belonging to the surrogate boundary. The white element is deactivated since is cut by the skin boundary Γ.

Thus, an additional step is required. All the element classes in Kratos are derived from the Element.h class, for instance, in the convection-diffusion application, one may use `LaplacianElement` or `EulerianConvDiff`. Both these two class elements are derived by the `Element` class. They both compute the local stiffness matrix and the local right-hand side of the Poisson problem and of the convection-diffusion problem, respectively.

What we must add to the classical implementations of the `LaplacianElement` or `EulerianConvDiff` is the flux through the surrogate boundary. What we are going to do is to create a derived class for each of the two elements class mentioned. We have called them `LaplacianShiftedBoundaryElement` and `ConvDiffShiftedBoundary`. Both these elements are derived from their parent classes, thus they do exactly what their parents do, plus they add a flux contribution through the surrogate boundary that now will be explained in detail.

We can add this flux contribution through the surrogate boundary by adding it to the computation of the local right-hand side and left-hand side of each element. The easier way is to use a flag called `BOUNDARY` that marks the elements with at least one surrogate node. Then, in the implementation of our element class, we insert an if statement that checks if the element is flagged. If the statement holds true, the flux contribution is added to the left and right-hand sides of the elemental stiffness matrix.

The code is a bit articulate because there may exist elements having more than one edge belonging to the surrogate boundary $\tilde{\Gamma}$. It can happen when the skin boundary has sharp corners. Consider for example Figure 22.

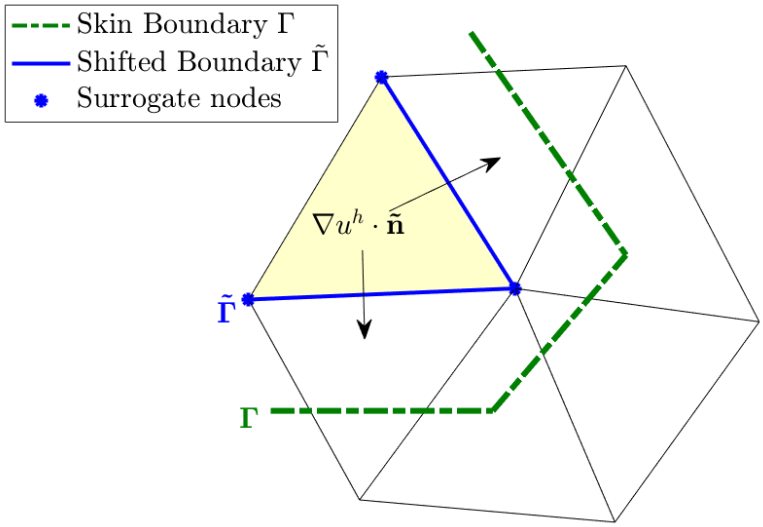


Figure 22: Flux of u^h between two edges belonging to the surrogate boundary (the white elements are deactivated).

The code for taking into account the flux through the surrogate boundary follows these steps:

Algorithm 3 Flux contribution

```
for each element do
    if element is VISITED then
        Get the parent geometry data.
        Find the surrogate face local ids.
        Get the unknowns vector for the three nodes.
        for each surrogate edge do
            Get the normal ñ.
            Calculate the gradient projection.
            Integrate along the edge.
            Add the surrogate boundary flux contribution.
```

In particular, the "Find the surrogate face local ids" step, manages the problem of Figure 22 computing the local ids of the faces of the boundary elements. A boundary element may have zero surrogate faces and therefore no flux is added. Or it may have all three faces that belong to the surrogate boundary. The following function can return the local ids of the current element

that belong to the surrogate boundary:

```
template<std::size_t TDim>
std::vector<std::size_t> LaplacianShiftedBoundaryElement<TDim>::
    GetSurrogateFacesIds()
{
    const std::size_t n_faces = TDim + 1;
    auto& r_neigh_elems = GetValue(NEIGHBOUR_ELEMENTS);
    // Check the current element faces
    std::vector<std::size_t> surrogate_faces_ids;
    for (std::size_t i_face = 0; i_face < n_faces; ++i_face) {
        // Get the neighbour element that shares the edge
        auto p_neigh_elem = r_neigh_elems(i_face).get();
        if (p_neigh_elem != nullptr && p_neigh_elem->Is(VISITED)) {
            // VISITED = the element is cut!
            surrogate_faces_ids.push_back(i_face);
        }
    }
    return surrogate_faces_ids;
}
```

Note that the algorithm is based on the flag `VISITED` which was already mentioned in Chapter 5.3.3; this flag marks if an element is cut by the skin boundary. The algorithm takes each of the faces and finds the element that shares the same face and checks if this element is cut or not using the flag `VISITED`. If it is cut it means that the face is part of the surrogate boundary and the flux term needs to be added.

One may notice that in the second step "get the unknowns vector for the three nodes", we are using the unknown function u^h. The piece of code is the following:

```
// Get the unknown vector
BoundedVector<double, NumNodes> nodal_unknown;
for (std::size_t i_node = 0; i_node < NumNodes; ++i_node) {
    nodal_unknown[i_node] = r_geom[i_node].FastGetSolutionStepValue(
        r_unknown_var);
}
```

Therefore, it has been used the Kratos method `FastGetSolutionStepValue` which takes the solution u^h at the current iteration.

5.7 SB Solvers

We now recap all the work done in the derived Kratos' SBM solver class for the stationary Poisson problem. Then, we explain the structure of the transient solver and the moving objects solver. All three Kratos' solver classes have been developed from scratch and are located in the convection-diffusion application.

5.7.1 Stationary SBM solver

The stationary SB solver is called `SBMConvectionDiffusionStationarySolver` and its structure can be outlined as follows:

Algorithm 4 Stationary SBM solver

1: `Initialization`
2: Import the skin boundary.
3: Obtain the level set function.
4: Find the surrogate boundary nodes.
5: Compute the projections.
6: Compute the MPCs equations ($\boldsymbol{G}$ and $\boldsymbol{u_n}$).
7: Create and impose the MPCs.
8: `InitializeSolutionStep`
9: Assembly the linear system + add the flux contribution.
10: Solve the linear system.

All the steps of the shifted boundary method, i.e. from line 2 to line 7 of the Algorithm 4, can be applied once during the simulation. The imposition of the boundary conditions throughout the MPCs allows us to overcome the problem of computing the gradient term. For this reason, we write the $\nabla u|_{\tilde{\boldsymbol{x}}}$ term, using the solution u^h of the problem so that no fictitious iteration loops are requested.

This solver is usable also in the case of stationary convection-diffusion problems that will be treated in Chapter 6. In fact, once the known velocity vector field is set in the project parameters or directly in the solver, then we can use the same solver (`SBMConvectionDiffusionStationarySolver`), changing the element type. For the Poisson equation, an element type derived from the `LaplacianElement` is needed: `LaplacianShiftedBoundaryElement`. Instead, for a convection-diffusion problem, an element type is derived from the `EulerianConvDiff` element, the so-called `ConvDiffShiftedBoundary`. Recall that both these derived elements add the feature of adding the flux term through the surrogate boundary for the reasons discussed in Chapter 5.6.

5.7.2 Transient SBM solver

A different solver is required in the case of transient problems. Generally in transient problems, the Dirichlet boundary conditions at the skin boundary may change over time. If this is the case, then the Dirichlet value at the projection changes and thus, the MPCs change at each time step. Recall in fact Equation 160 and introduce the dependency on time t:

$$\begin{aligned} u(\tilde{\boldsymbol{x}}, t^{n+1}) &= \bar{u}(\boldsymbol{x_t}, t^{n+1}) - \nabla u|_{\tilde{\boldsymbol{x}}}(t^{n+1}) \cdot \boldsymbol{d} \\ &= \bar{u}(\boldsymbol{x_t}, t^{n+1}) - \boldsymbol{G u_n} \cdot \boldsymbol{d} \end{aligned} \tag{168}$$

Where, in this case, we are using an implicit Euler method for the integration in time. Note that the known Dirichlet value at the skin boundary depends on time, then also the neighbouring

degrees of freedom $\boldsymbol{u_n}$ will be evaluated at t^{n+1}. Due to the nature of MPCs, it is automatically done. Then, if the geometry does not change, the matrix $\boldsymbol{G}$, the vector of degrees of freedom $\boldsymbol{u_n}$ and the distance vector $\boldsymbol{d}$ are always the same independently on time.

Equation 168 implies that the MPCs change at each time step. If the immersed object is fixed, what changes are only the constant $\bar{u}(\boldsymbol{x_t}, t^{n+1})$, that is the exact Dirichlet value evaluated at the projection onto the true boundary. Unfortunately, it is not implemented yet in Kratos a method for modifying only what we have called the `ConstantVector` in an MPC. What is possible is to erase the MPCs at the beginning of each iteration step and impose the new one changing only the `ConstantVector`, i.e. $\bar{u}(\boldsymbol{x_t}, t^{n+1})$ in Equation 168. The sketch of the algorithm of the transient solver, which has been called in Kratos as `SBMConvectionDiffusionTransientSolver`, looks like the following:

Algorithm 5 Transient SBM solver

`Initialization`
    Import the skin boundary.
    Obtain the level set function.
    Find the surrogate boundary nodes.
    Compute the projections.
    Compute the MPCs equations ($\boldsymbol{G}$ and $\boldsymbol{u_n}$).
`InitializeSolutionStep`
    Erase the old MPCs.
    Create and impose the new MPCs using $\bar{u}(\boldsymbol{x_t}, t^{n+1})$.
    Assembly the linear system + add the flux contribution.
    Solve the linear system.

Note that also a θ-scheme can be applied for the integration in time of the diffusion or convection-diffusion equation. The MPCs, in this case, are applied in exactly the same manner since, in the end, MPCs are simply relations between the degrees of freedom. Therefore they are equations added to the linear system. The time integration scheme will automatically evaluate them in order to be consistent with the rest of the linear system.
There is one aspect that needs to be taken into account when for instance Crank-Nicolson time integration scheme is used, that is regarding the flux contribution through the surrogate boundary. In that case, we have to add explicitly the dependence on theta to obtain the expected optimal convergence rate for the time integration (2nd order for Crank-Nicolson algorithm):

```
double theta = Variables.theta ;
// Get the unknown vector
BoundedVector<double, NumNodes> nodal_unknown;
for (std::size_t i_node = 0; i_node < NumNodes; ++i_node) {
    nodal_unknown[i_node] = (1-theta) * r_geom[i_node].
        FastGetSolutionStepValue(r_unknown_var,1) + theta * r_geom[i_node
        ].FastGetSolutionStepValue(r_unknown_var);
}
```

Where the usual θ-scheme has been applied for obtaining the unknown variable $u^h_{n+\theta}$:

$$u^h_{n+\theta} = (1-\theta)u^h_n + \theta u^h_{n+1} \tag{169}$$

Regarding the application of this solver to the Poisson transient or the transient convection-diffusion problems, it works as the stationary solver. In other words, in the case of a pure

diffusion problem, the `LaplacianShiftedBoundaryElement` element-type is used. In the case of convection-diffusion, the `ConvDiffShiftedBoundary` element-type is chosen.

5.7.3 Moving objects solver

The last part of Chapter 6, which is dedicated to the results, will show the potentialities and the ideas of the future development of the SB method. One of the main advantages of unfitted methods is, in fact, to simulate large boundary movements or large deformations of the embedded boundaries without the need for re-meshing techniques. The feature of simulating moving embedded objects can be introduced quite easily in our SBM algorithm developed in Kratos.

The effect that the immersed object is moving reflects on the skin boundary and, thus, on the surrogate boundary. In general, we may have that the surrogate boundary changes completely in one time-step or that just a part of it shifts on new elements' edges.
The movements of the skin boundary in time lead to the calculation of the level set function at each time-step. In the case some nodes of the background mesh change from being inside to outside or the opposite, then a new surrogate boundary is present, and all the rest SB algorithm needs to be applied again.
Therefore a sketch of the solver algorithm that can simulate moving objects is the following:

Algorithm 6 Moving object SBM solver

1: `Initialization`
2: Import the skin boundary.
3: `InitializeSolutionStep`
4: Motion of the skin boundary.
5: Obtain the current level set function.
6: Find the current surrogate boundary nodes.
7: Compute the current projections.
8: Compute the current MPCs equations ($\boldsymbol{G}$ and $\boldsymbol{u_n}$).
9: Erase the old MPCs.
10: Create and impose the new MPCs using $\bar{u}(\boldsymbol{x_t}, t^{n+1})$.
11: Assembly the linear system + add the flux contribution.
12: Solve the linear system.

Note that now almost all the passages regarding the SB method are inside the solution step, i.e. the part of the algorithm that is repeated at the beginning of each time step.

6 Results

This last chapter is dedicated to the validations and results of the SB method that has been implemented in Kratos. In Kratos, it is possible to easily create pre-processing files (MDPA files) for very simple geometries, but in general, it is easier to use a different software both for the pre- and post-process. For everything concerning the pre-process, it will be used the software GiD [10], `https://www.gidsimulation.com/`. GiD is a user-friendly pre and post-processor for numerical simulation of engineering and scientific phenomena. GiD offers a wide variety of possibilities within its framework:

- Geometrical modelling (CAD)
- Mesh generation
- Definition of analysis data
- Data transfer to analysis software
- Post-processing operations
- Visualization of results

Figure 23: GiD logo [29]

In this thesis, Gid has been used especially for all the steps of the pre-process, i.e. modelling the geometry, mesh generation and creation of the groups. All these steps return the MDPA files that Kratos and the SB method require, that is, an MDPA of the background mesh (with eventual groups related to standard Dirichlet or Neumann boundary conditions) and an MDPA of the skin boundary. Regarding the post-process and the analysis of the results both GiD and Matlab [28] are used.
Similar to how the SB method was developed during the internship with the Kratos Multi-physics group, the results will be presented in a progressive order of increasing difficulty. We have started with the stationary Poisson problem with a linear exact solution and embedded objects with smooth shapes. Progressively we tested more complex object shapes, considering also the convection term in both stationary and transient regimes.

6.1 Poisson problem with linear exact solution

As a basic check, we have applied our SBM solver in a linear case of stationary Poisson problem to see if the exact solution is obtainable. Consider the linear function in two dimensions:

$$u(x, y) = x + y \tag{170}$$

It clearly satisfies the following differential equation

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= 0 && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= x + y && \boldsymbol{x} \in \Gamma_1 \\ u(\boldsymbol{x}) &= x + y && \boldsymbol{x} \in \Gamma_2 \end{aligned} \tag{171}$$

Where the boundary of Ω, i.e. $\partial\Omega$, has been split in two parts such that $\Gamma_1 \cup \Gamma_2 = \partial\Omega$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$. Actually, in the next results, we will often deal with problems with an immersed object, and its boundary will be named Γ_2. The interface of the immersed object is considered embedded. Whereas the external boundary of the computational domain Γ_1 will be treated with a standard conforming body-fitted approach.

The SB solver implemented in Kratos will read the skin boundary Γ_2, where the exact solution is known, and following the procedure we have explained in detail in the implementation chapter, will find the surrogate boundary $\tilde{\Gamma}_2$ and the MPCs will be imposed for each of the surrogate nodes.

In the case of a linear solution, we expect that the linear basis functions of the finite element space are able to get the exact solution u. Moreover, the modified Dirichlet SB conditions are precise up to the second order because we neglected the second-order term in the Taylor expansion. In fact, the gradient is constant in all the domains and, therefore, the MPCs equations should impose the exact solution at the surrogate nodes.

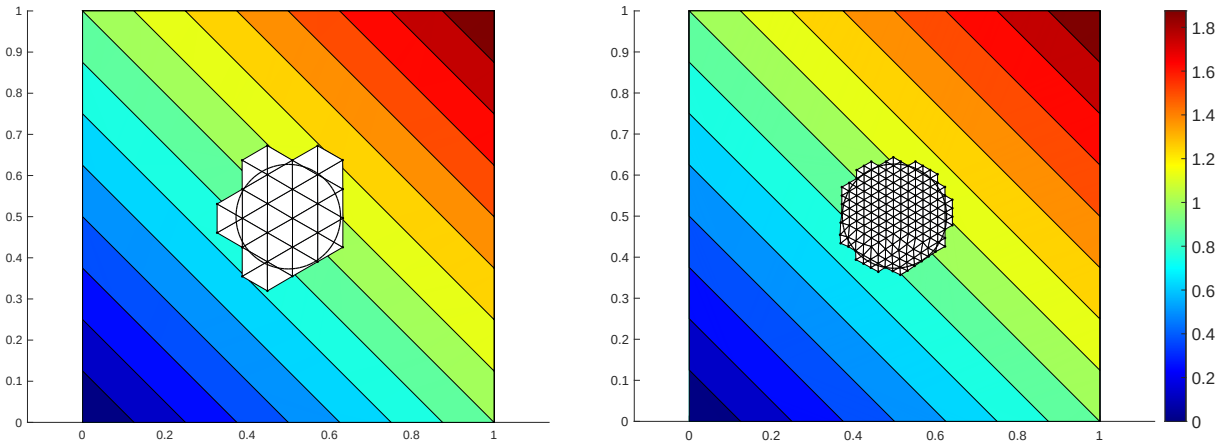


Figure 24: SB solution of Equation 171 using two different mesh sizes $h = 0.065$ and $h = 0.025$.

We expect, independently from the mesh size, that the SB method can obtain the exact solution up to the machine's precision. The SB solution is reported in Figure 24 for two different mesh sizes. The absolute pointwise error with respect to the exact solution $u = x + y$ is reported for the two meshes in Figure 25.

The results are the ones expected, in fact, the embedded boundary Γ_2 and the MPCs on the surrogate boundary $\tilde{\Gamma}_2$ do not affect the FEM algorithm keeping it able to perfectly capture the behaviour of linear functions. The pointwise error is of the order of the machine precision adopted (double precision). The mesh sizes and global error estimates for the two cases are presented in Table 1; the L^∞- and L^2- norm of the error are defined in the next chapter.

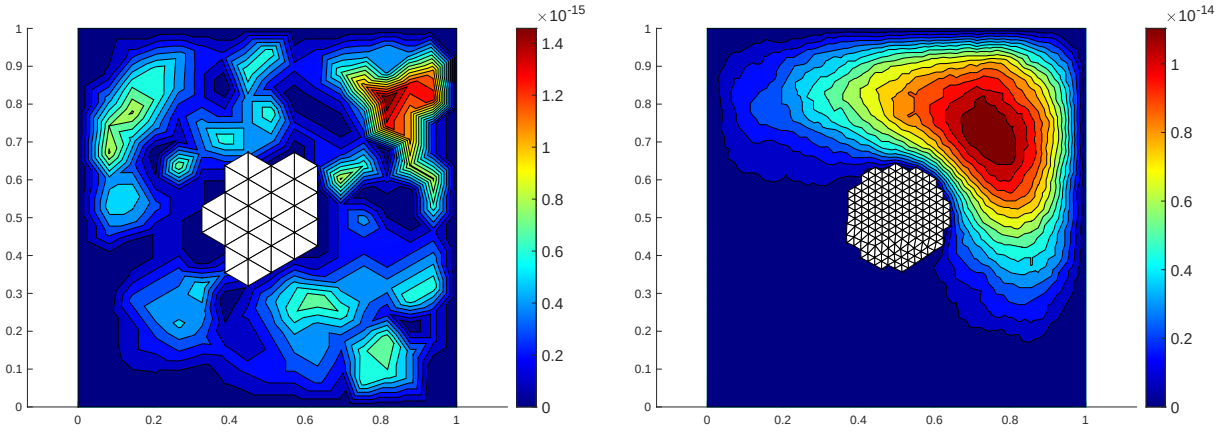


Figure 25: Module of the error of the SB solution with respect to the exact solution $u = x + y$ using two different mesh sizes.

	h	L^∞	L^2
mesh1	0.065	$1.5543e-15$	$4.3441e-16$
mesh2	0.025	$1.1546e-14$	$4.0928e-15$

Table 1

6.2 Poisson equation with uniform boundary conditions

Let us increase the difficulty of the problem step by step considering a Poisson equation with Dirichlet uniform boundary conditions:

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= 0.0 && \boldsymbol{x} \in \Gamma_1 \\ u(\boldsymbol{x}) &= 2.0 && \boldsymbol{x} \in \Gamma_2 \end{aligned} \tag{172}$$

Where Ω is the computational domain of interest and $\Gamma_1 \cup \Gamma_2 = \Gamma = \partial\Omega$ as in the previous case. Consider as Ω a two-dimensional disk of radius $R = 3.0$ centred in the origin. Consider also a unit source term $f = 1$. In practice, the problem is simply a diffusion equation with a positive source term that tends to increase u. The analytical solution to the differential problem is the following second-order polynomial function:

$$u(x, y) = \frac{1}{4}\left(R^2 - x^2 - y^2\right) \tag{173}$$

That is a paraboloid that takes its maximum value equal to $R^2/4$ at the origin. To represent an immersed object, we draw an internal boundary with a circular shape representing a smooth immersed object shape. In the external boundary Γ_1, i.e. when $x^2 + y^2 = 3$, we impose classical conforming Dirichlet boundary condition, and in the internal boundary Γ_2 ($x^2 + y^2 = 1$) we apply an unfitted imposition of the Dirichlet boundary condition using the SB method.

The following validation examples use a set of circular meshes with different characteristic lengths h's. In Table 2, the meshes' characteristic lengths h's and names are reported.

In Figure 26, we report the FEM solution in a contour plot using the SB method on mesh1 and mesh2. As one can notice, especially in the case of mesh1, we can really distinguish the white

	mesh1	mesh2	mesh3	mesh4	mesh5	mesh6	mesh7
h	0.2	0.1	0.05	0.025	0.0125	0.00625	0.003125

Table 2

elements taken out of the simulation. Those elements, according to the theory, are either the cut elements or the internal ones.

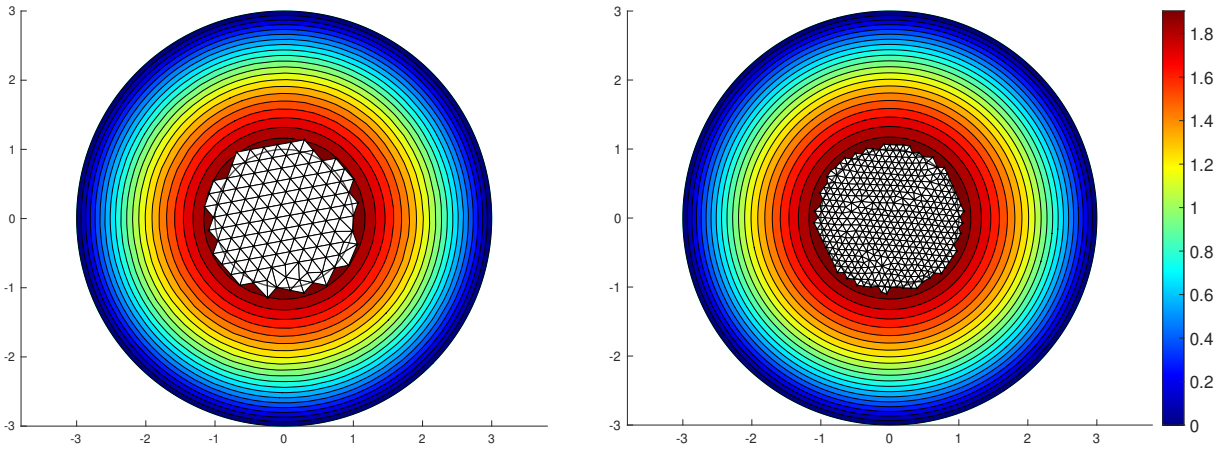


Figure 26: SB solution $u^h(x, y)$ of the differential problem 172 using mesh1 and mesh2.

Naturally, as one can expect when we use more refine meshes, the shape of the surrogate boundary tends to be closer and closer to the true one. But the most important thing is that the surrogate boundary nodes are closer to their projections and, therefore, the error related to the use of SBM scales correctly, i.e is proportional to h^2.

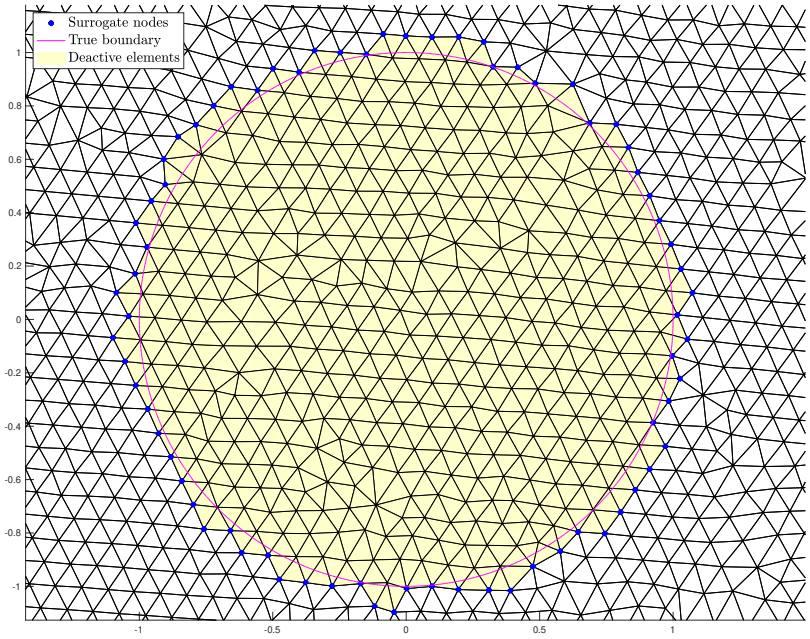


Figure 27: Zoom of the true boundary and of the surrogate boundary nodes in the case of mesh2.

We now quantify globally the error. Given the analytical solution of Equation 173, one can consider a global error on the domain on interest $\tilde{\Omega}$. In particular, we will consider the L^2-norm of the error that is defined as follows:

$$||\text{err}||^2_{L^2(\tilde{\Omega})} = ||u - u^h||^2_{L^2(\tilde{\Omega})} = \int_{\tilde{\Omega}} (u - u^h)^2 d\tilde{\Omega} \tag{174}$$

And in the finite element space, it can be approximated as the following:

$$||\text{err}||^2_{L^2(\tilde{\Omega})} = \sum_{A_n \in \mathcal{A}} \int_{A_n} (u - u^h)^2 dA_n = \sum_{A_n \in \mathcal{A}} \left[A_n \left[u(\boldsymbol{x_n}) - u^h(\boldsymbol{x_n}) \right]^2 \right] \tag{175}$$

Where A_n is the nodal area of node $\boldsymbol{x_n}$ and A is the set containing all the nodal areas of all the nodes belonging to $\tilde{\Omega}$.

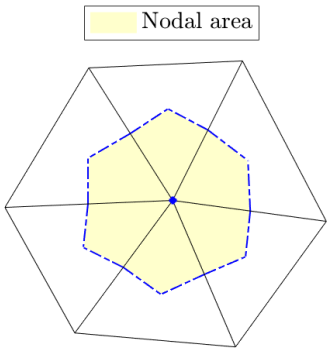


Figure 28: Nodal area of a general point that is not a surrogate boundary node.

In the case of a surrogate boundary node, the algorithm considers a reduced nodal area. In fact, the SB method completely eliminates the cut elements and, therefore, does not have any sense to include their contribution to the nodal area. Figure 28 and Figure 29 show two general cases. In the first case, the node is not surrogate, thus the nodal area is the standard one. In the second figure, the node is a surrogate node and therefore the nodal area is reduced depending on how many cut elements are present.

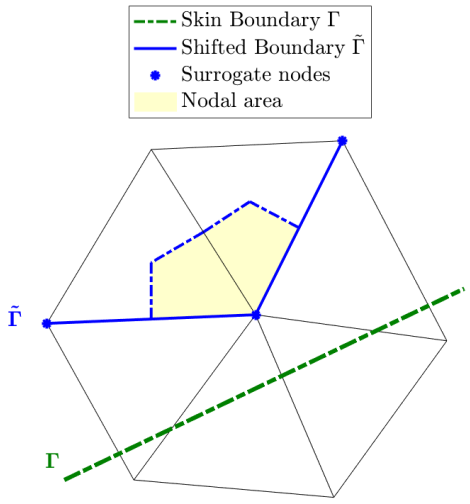


Figure 29: Nodal area of a surrogate boundary node.

It is fundamental to underline that when the shifted boundary method is applied, the elements cut by the true boundary are eliminated. Therefore they will not be taken into account also during the calculation of the error. But, as we have seen in Figure 27, refining the grid means reducing the area between the surrogate and the true boundary, and thus $\Gamma \to \tilde{\Gamma}$ as $h \to 0$.

In Figure 30, we report the nodal error, i.e. the module of the difference between the exact solution and the one obtained using the SB method, using mesh1 and 2 of Table 2. As we can see

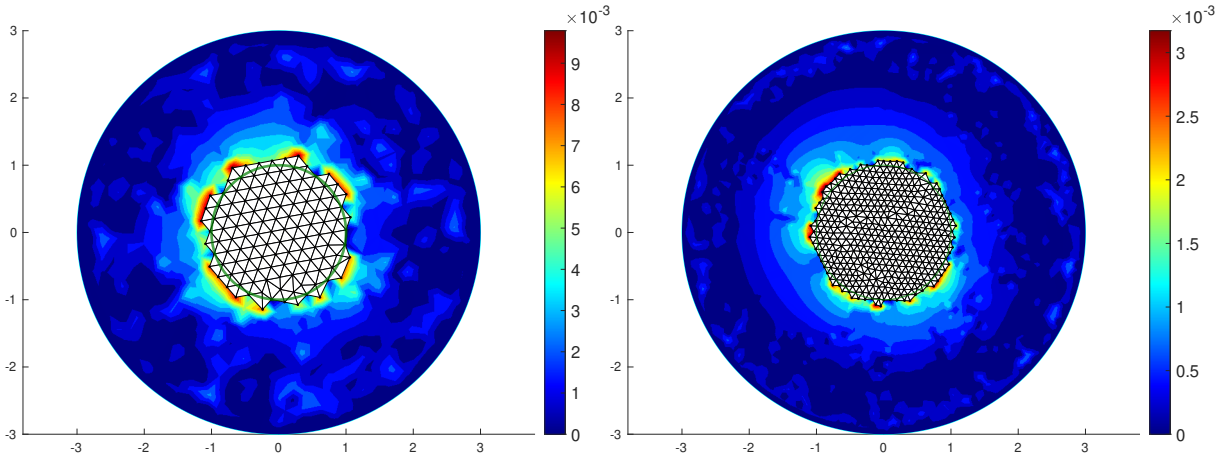


Figure 30: Nodal error in Ω using mesh1 in the left panel and mesh2 in the right panel.

from the colour bar, the pointwise error is of order 10^{-3}, and the larger errors are concentrated in the region close to the surrogate boundary.
In Figure 31, a zoom of the left panel of Figure 30 is reported, where we can clearly see different regions around the surrogate boundary. First, the true boundary Γ_2 is highlighted in dark green. Moreover, recall that the cut elements are white because they have been set as non-active. In fact, for this reason, the contribution of the global error coming from the region inside the surrogate boundary, cannot be taken into account.

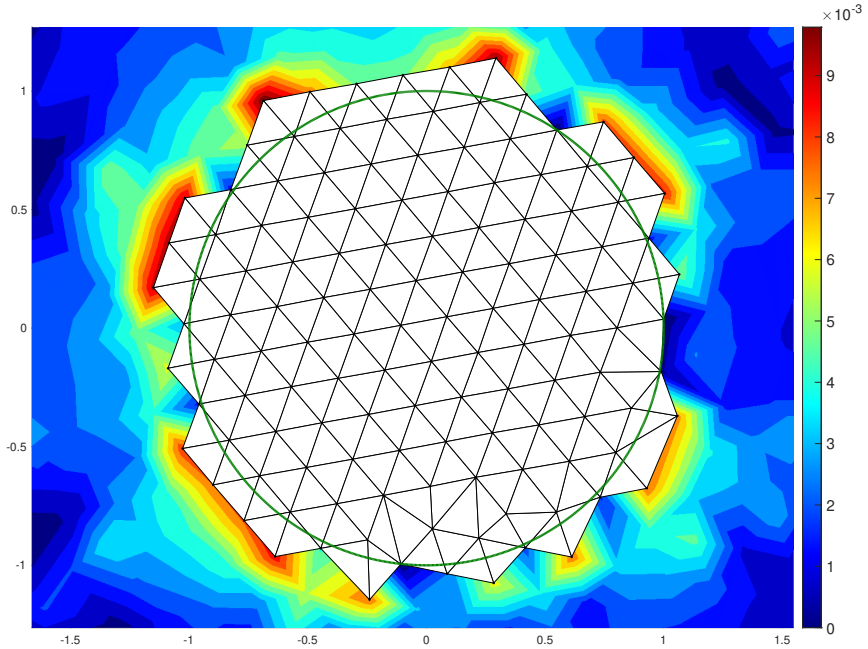


Figure 31: Zoom of the nodal error of left panel of Figure 30 using mesh1.

Then, note also that around the surrogate boundary, there are regions where the error is bigger and others where the error is very small. The peaks of the nodal error are in the regions corresponding to the surrogate boundary nodes that are more distant from the true boundary, i.e. when the term $(\boldsymbol{x_t} - \tilde{\boldsymbol{x}})$ in Equation 141 is larger. Let us report Equation 141:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) + o(|\boldsymbol{x_t} - \tilde{\boldsymbol{x}}|^2)$$

This equation represents the approximation of imposing the modified Dirichlet boundary conditions at the surrogate boundary. When the projection $\boldsymbol{x_t}$ is far from its surrogate node $\tilde{\boldsymbol{x}}$, i.e. when the module of the distance vector $\boldsymbol{d}$ is large, then the neglected term $o(|\boldsymbol{x_t} - \tilde{\boldsymbol{x}}|^2)$ is larger. It leads to relevant approximations when the MPCs are imposed.
From the other side, we have regions around other surrogate boundary nodes where the error is much smaller. These surrogate nodes are the ones closer to the true boundary, that is when $(\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) << h$. Here the MPCs are almost imposing the exact solution as in the body-fitted approach.

In conclusion, we may say that SBM clearly commits an additional error with respect to the standard conforming FEM. In fact, the error peaks in Figure 30 are close to the embedded object, but, as we will see in the next figures, the global error, i.e. the L^2-norm of the error, described in Equation 175 still goes to zero quadratically, that is the most relevant consideration.

In the following tables, we report how the global error decreases as we adopt finer grids. Moreover, as a comparison, we report also the error in the body-fitted (B-F) case. Naturally, the mesh cannot be the same for obvious reasons, but we have kept the same characteristic mesh sizes of Table 2. In Figure 32, the convergence study in the L^2-norm is shown, and the corresponding data is reported in Table 3.

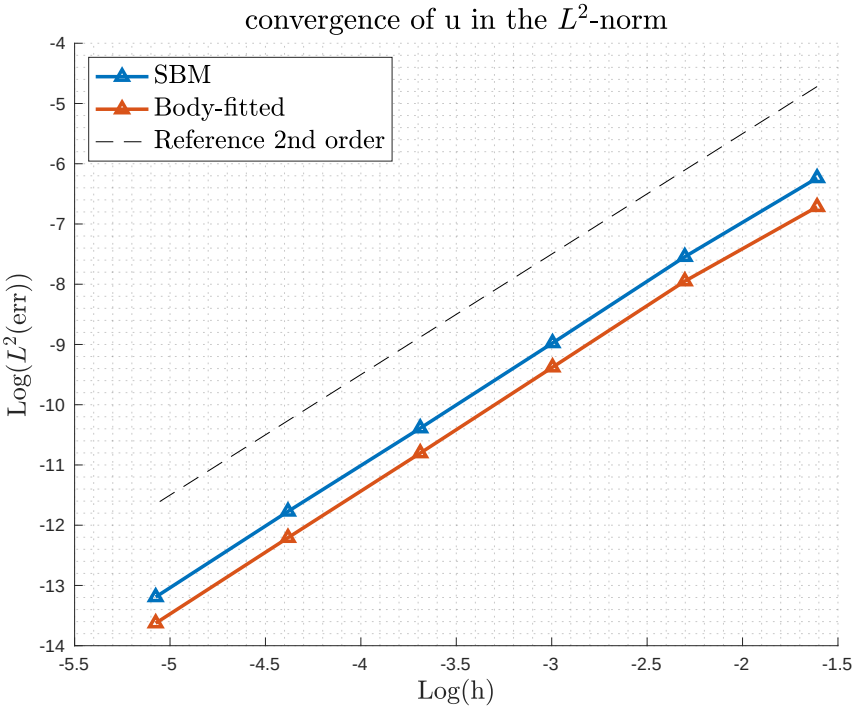


Figure 32: Convergence study for SBM method in the case of Poisson equation with uniform boundary conditions.

	mesh1	mesh2	mesh3	mesh4	mesh5	mesh6
$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$ SBM	0.001956	0.0005290	0.0001263	$3.075e-05$	$7.720e-06$	$1.865e-06$
$\|\|\text{err}\|\|_{L^2(\Omega)}$ B-F	0.001211	0.0003528	$8.454e-05$	$2.033e-05$	$4.990e-06$	$1.208e-06$
$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$ SBM	0.01044	0.003382	0.001031	0.0002311	$5.396e-05$	$1.418e-05$
$\|\|\text{err}\|\|_{L^\infty(\Omega)}$ B-F	0.003663	0.001118	0.0002974	$7.477e-05$	$1.953e-05$	$5.074e-06$

Table 3

The slope results of the log-log plot of the L^2-norm of the error of Figure 32 are reported in the following table for both the SB and the body-fitted case:

	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	**average**
slope_{L^2} SBM	1.8865	2.0656	2.03909	1.9939	2.0487	2.0068
slope_{L^2} B-F	1.7791	2.0613	2.0555	2.0270	2.0462	1.9938
slope_{L^∞} SBM	1.6260	1.7139	2.157	2.0986	1.9280	1.9047
slope_{L^∞} B-F	1.711	1.9113	1.9920	1.9360	1.9449	1.8990

Table 4

In Table 3 and Table 4 we also reported the L^∞-norm of the error, which is nothing but the maximum nodal error between the exact solution and the SBM one:

$$||\text{err}||_{L^\infty(\tilde{\Omega})} = \max_{\boldsymbol{x}\in\tilde{\Omega}}(|u(\boldsymbol{x}) - u^h(\boldsymbol{x})|) \tag{176}$$

Once again, we underline that the error in the SB method is expected to be higher than the one that occurred in the standard conforming FEM. In fact, the error in the case of SBM has two different natures, the one shared with the body-fitted one, deriving from the use of linear basis functions, and the second deriving from the imposition of modified Dirichlet boundary conditions.

About the L^∞-norm or the error, i.e. the maximum pointwise error, we may say that is always higher when the SB method is used; moreover, that the location of the peaks is on the surrogate boundary. But, as one may notice from Table 4, the convergence rate is practically the same. It is guaranteeing that as $h \to 0$, the SBM method behaves with the same convergence rate as the body-fitted counterpart.

In Figure 33 and Figure 34, we can see the error and the gradient error comparisons between the two formulations (B-F and SBM) utilizing mesh1. As one can see in the body-fitted case, the pointwise error and the pointwise norm of the error gradient around the internal boundary Γ_2 is smaller than the error in the region between the two boundaries Γ_1 and Γ_2. In fact, the body-fitted approach applies conforming boundary conditions on all Γ, and therefore the exact solution is imposed on Γ_2.
Instead, in the SBM case, the relevant error comes from the imposition of the boundary conditions on the internal embedded boundary Γ_2, as we already discussed previously.

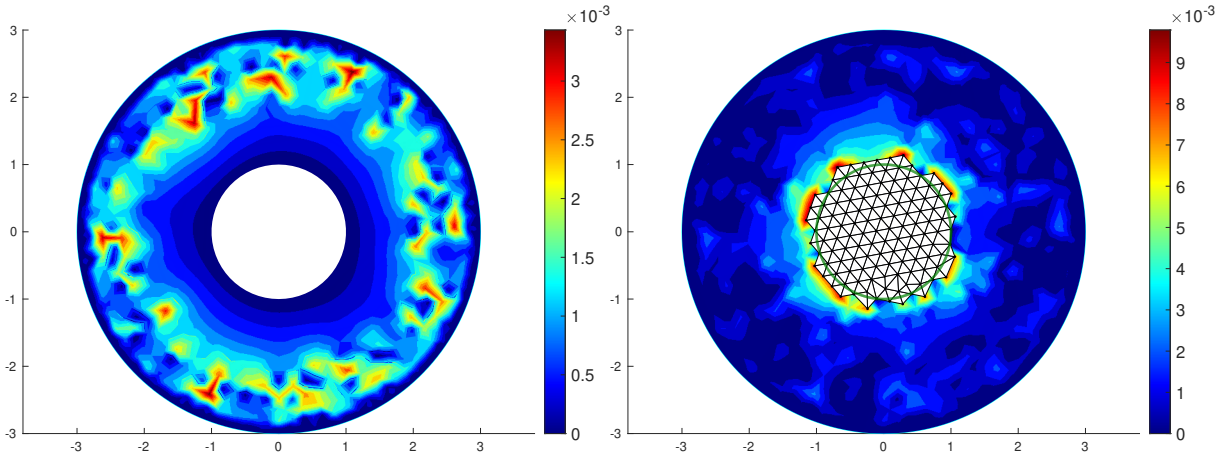


Figure 33: Pointwise error comparison between body-fitted (left) and SB method (right) using mesh1

The gradient error comparison shows again that the body-fitted case has a smaller error also in terms of the norm of the gradient. Moreover, the pointwise error of the gradient is larger close to the surrogate boundary, and it is a consequence of the larger error close to $\tilde{\Gamma}_2$.

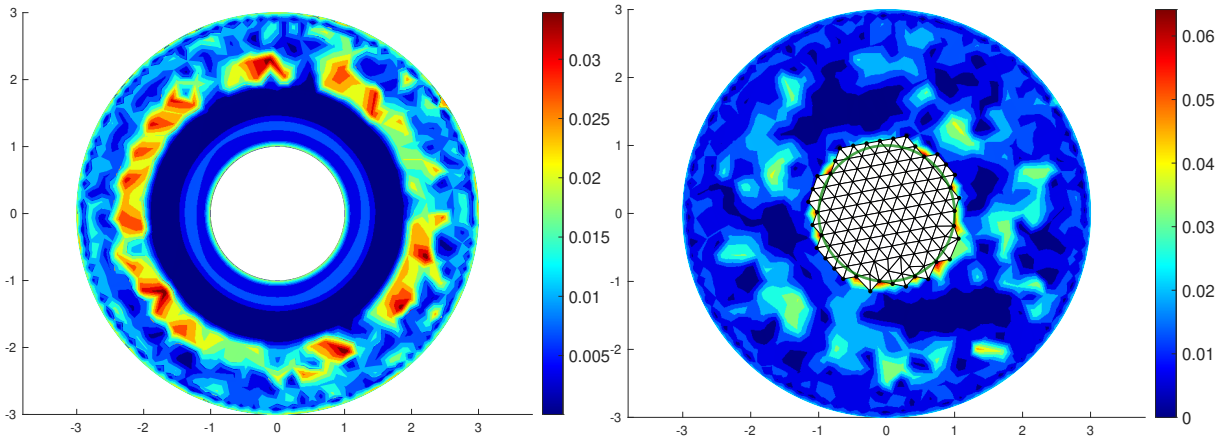


Figure 34: Pointwise norm gradient of the error comparing the body-fitted (left) and SBM formulation (right) using mesh1.

6.3 Poisson equation with non-uniform boundary conditions

Let us now consider a more complicated case in order to test the SBM algorithm developed in Kratos Multiphysics. Consider the following Poisson problem with non-uniform boundary conditions. An identical example was proposed in [26].

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= 1 && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= \overline{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma_1 \\ u(\boldsymbol{x}) &= \overline{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma_2 \end{aligned} \tag{177}$$

where:

$$\overline{u}(x,y) = \frac{1}{4}\left(9 - x^2 - y^2 - 2\ln 3 + \ln(x^2+y^2)\right) + \frac{1}{4}\sin x \sinh y \tag{178}$$

That, in this example, is also the analytical solution of the problem. One may check that $\Delta\overline{u}(\boldsymbol{x}) = -1 \;\; \forall \boldsymbol{x} \in \mathbb{R}^2 \setminus (0,0)$.

Note that, as previously done, the boundary of Ω, $\Gamma = \partial\Omega$, has been split into two parts. Ω, in particular, is again the two-dimensional disk of radius R = 3.0 centred in the origin, Γ_1 is the external boundary of the disc and Γ_2 it is considered embedded, and it's a circular skin boundary of radius $r = 1.0$ centred in the origin.
Let us perform the same analysis as the previous chapter and, for simplicity, we are going to use also the same meshes given in Table 2. The geometry of the problem is exactly the same. Note that the origin, where the differential equation is not verified, is fully contained in the immersed object and, therefore, it will not be considered in the computational domain.

The analytical solution in Equation 178 is known, so we can compute the pointwise error, the L^2-norm and the L^∞-norm of the error. The next table shows the comparison in terms of global errors measured between the SB method and the standard body-fitted approach. Note that the meshes for the two cases are not the same, but the characteristic lengths have been chosen to be equal.
Table 5 reports the L^2- and L^∞-norm of the error using the first six meshes of Table 2.

	mesh1	mesh2	mesh3	mesh4	mesh5	mesh6
$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$ SBM	0.003493	0.001021	0.0002506	$5.976e-05$	$1.568e-05$	$3.789e-06$
$\|\|\text{err}\|\|_{L^2(\Omega)}$ B-F	0.001420	0.0003951	0.0001015	$2.430e-05$	$5.856e-06$	$1.483e-06$
$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$ SBM	0.02485	0.007753	0.002069	0.0004796	0.0001322	$3.652e-05$
$\|\|\text{err}\|\|_{L^\infty(\Omega)}$ B-F	0.007751	0.001886	0.0007307	0.0002625	$6.068e-05$	$1.951e-05$

Table 5

The convergence study in terms of the L^2-norm of the error is shown in Figure 35, where also the body-fitted data has been drawn in order to compare the two. The corresponding table of the slopes for both the L^2- and L^∞-norm is reported in Table 6.

The results show that the presence of non-uniform embedded boundary conditions does not affect the goodness of the results. In particular, we can see that the second-order convergence of the error in the L^2-norm is respected and that we have an additional error due to the imposition on the modified Dirichlet boundary conditions; in fact, the SB line in Figure 35 has a constant shift meaning that an additional error is present, but it does not affect the order of convergence.

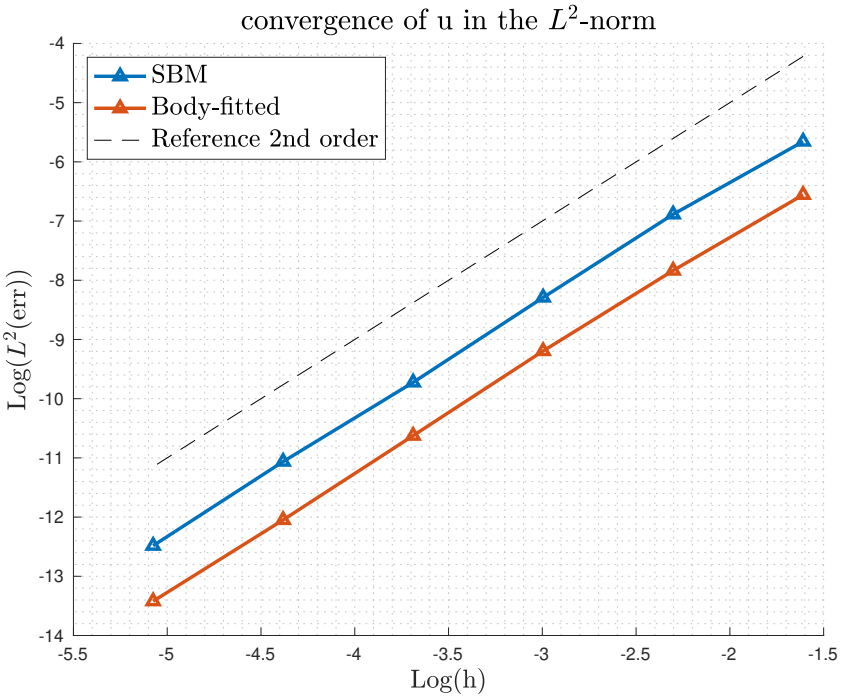


Figure 35: Convergence study for SBM method in the case of Poisson equation with non uniform boundary conditions.

	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	**average**
slope$_{L^2}$ SBM	1.7744	2.0262	2.0685	1.9301	2.0491	1.9697
slope$_{L^2}$ B-F	1.8462	1.9598	2.0631	2.0533	1.9807	1.9806
slope$_{L^\infty}$ SBM	1.6806	1.9056	2.1092	1.8586	1.8561	1.8820
slope$_{L^\infty}$ B-F	2.0385	1.3684	1.4768	2.1131	1.6370	1.7268

Table 6

In the next figures, Figure 36 and 37, we can see the SBM solution using mesh2 and the corresponding pointwise module of the error in $\tilde{\Omega}$.

In particular in Figure 37, we can see the zoom in two regions of the domain close to the skin boundary: the upper left and the upper right parts. The left part presents a region where the error seems to be higher, while in the right one, the error is smaller. The reason for this is inside the nature of the function we are approximating: the analytical solution of Equation 178. As one can note from Figure 36, the function u^h in the region of high error has the gradient vector perpendicular to the surrogate boundary. Whereas, in the low error region, the gradient is almost parallel and close to zero because the function is almost constant.
This phenomenon is in accordance with the equations we have written before. When we used the Taylor expansion for imposing the modified Dirichlet boundary conditions, we neglected the second-order term in Equation 143. Then, we have basically imposed the following:

$$u(\tilde{\boldsymbol{x}}) = \bar{u}(\boldsymbol{x_t}) - \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}}) \tag{179}$$

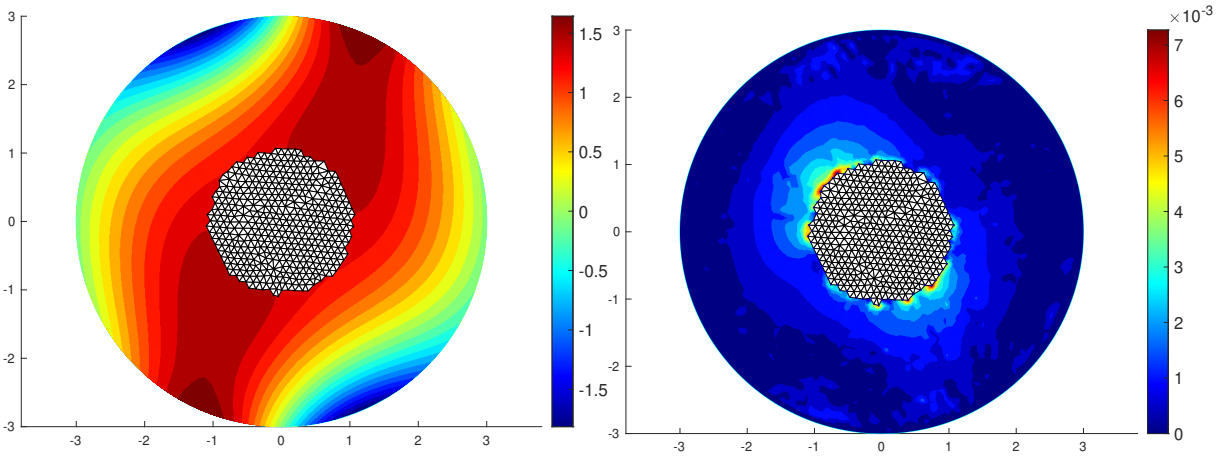


Figure 36: Fem solution u^h using the SB method in the left panel and the pointwise module of the error in the right one using mesh2.

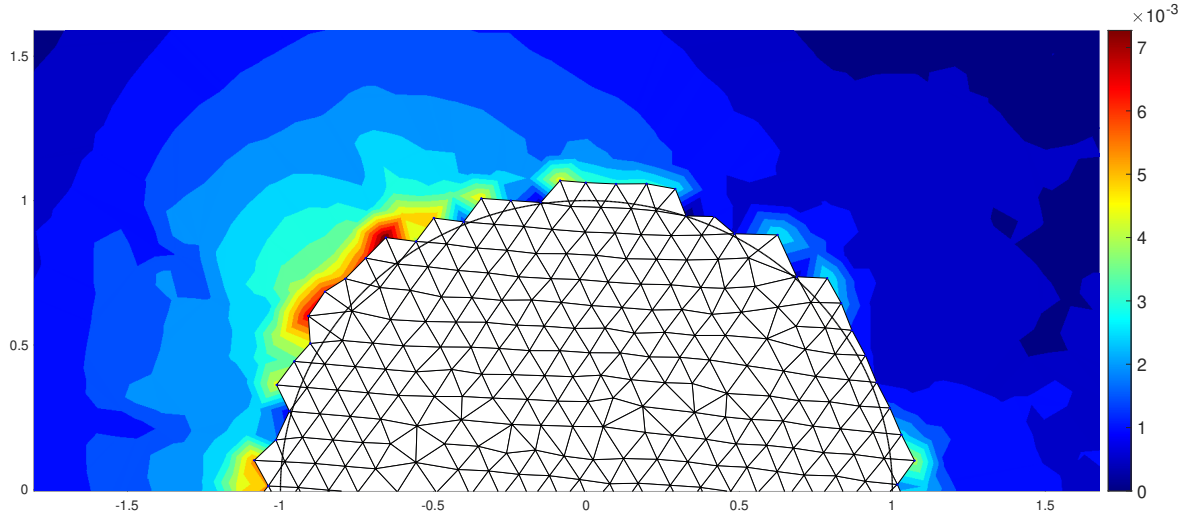


Figure 37: Zoom of the upper part of the pointwise module error of right panel of Figure 36.

Neglecting the second-order term, we have implicitly assumed that the $\nabla u|_{\tilde{\boldsymbol{x}}}$ was constant between the surrogate boundary node $\tilde{\boldsymbol{x}}$ and the true boundary node projection $\boldsymbol{x_t}$. For this reason, in the cases where the gradient vector is almost constant, or in the cases where the scalar product $\nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\boldsymbol{x_t} - \tilde{\boldsymbol{x}})$ is almost zero, the error related to the imposition of the modified Dirichlet boundary conditions on the surrogate boundary tends to be small.
Expanding to the third order term, one obtains the quadratic term that contains the Hessian matrix $\boldsymbol{\nabla}^2 u$ evaluated at the surrogate boundary node.

$$u(\tilde{\boldsymbol{x}}) = u(\boldsymbol{x_t}) + \nabla u|_{\tilde{\boldsymbol{x}}} \cdot (\tilde{\boldsymbol{x}} - \boldsymbol{x}) + \frac{1}{2}(\tilde{\boldsymbol{x}} - \boldsymbol{x})^T \boldsymbol{\nabla}^2 u|_{\tilde{\boldsymbol{x}}} (\tilde{\boldsymbol{x}} - \boldsymbol{x}) + O(||\tilde{\boldsymbol{x}} - \boldsymbol{x}||^3) \tag{180}$$

The neglecting term, depending on the Hessian matrix, explains why exist different error regions and where one may expect larger errors.

6.4 Complicated domains with corners

In this chapter, we will test the SB method with more complicated object shapes similar to what has been done in [7, 32]. In particular, it will be considered a standard Poisson problem (with zero forcing) with homogeneous boundary conditions both at the border of the fluid domain and at the object interface, but as before, at the interface, we will consider, in the case of SBM, surrogate Dirichlet boundary conditions.

Consider a stationary Poisson problem similar to the ones seen before:

$$\begin{aligned} \Delta u(\boldsymbol{x}) &= 0 && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= 1 && \boldsymbol{x} \in \Gamma_1 \\ u(\boldsymbol{x}) &= 0 && \boldsymbol{x} \in \Gamma_2 \end{aligned} \tag{181}$$

Where Γ_1 is the external boundary of the background domain and Γ_2 is the interface between the domain and the immersed object. Therefore, what we expect is a pure diffusion phenomenon that takes value $u = 1$ at the external boundary and goes to $u = 0$ at the object's skin.
As objects, we are going to consider three different shapes that are increasingly more complex. The first is a simple circle we have already treated, the second is a cross, and the third is a more complicated shape that we will call "complex".

As background domain Ω, we will use a rectangular domain, [-1.5, 1.5] $\times$ [-1,1]. To test the algorithm, we will perform a convergence study and, to do so, we have created different meshes with different characteristic lengths h's. In the following table the characteristic lengths that will be used for the convergence estimate are reported:

	mesh1	mesh2	mesh3	mesh4	mesh5	mesh6	mesh7	mesh8
h	0.2	0.1	0.05	0.025	0.0125	0.00625	0.003125	0.0015625

Table 7

These meshes are simple rectangular meshes with unstructured grids and will be used for the SBM simulations with the different immersed object shapes. For validating the algorithm, one would like to perform a comparison. Unfortunately, an analytical solution is not available in this case and, therefore, we have to use the body-fitted solution as a reference. Three very refined body-fitted meshes have been created in order to consider them as the reference exact solutions for the three cases.
One of the disadvantages of the body-fitted or conforming formulation is that the mesh perfectly matches the object's shape. Thus we create a new mesh that matches the immersed object shape.

As said, for each of the three examples, we will perform a simulation on a very refine mesh applying the body-fitted formulation of Kratos and use the results as a reference for the convergence study of the SB method. The mesh size chosen for the body-fitted case is $h = 0.001$.

In Figure 38 and 39, we report the body-fitted FEM solutions u^h's in the three cases mentioned before. We recall that in body-fitted cases we have imposed standard conforming Dirichlet boundary conditions on both at the external interface ($u = 1$) and at the immersed object interface ($u = 0$).

The main difficulties that the SBM formulation might encounter when dealing with complex skin boundary shapes are:

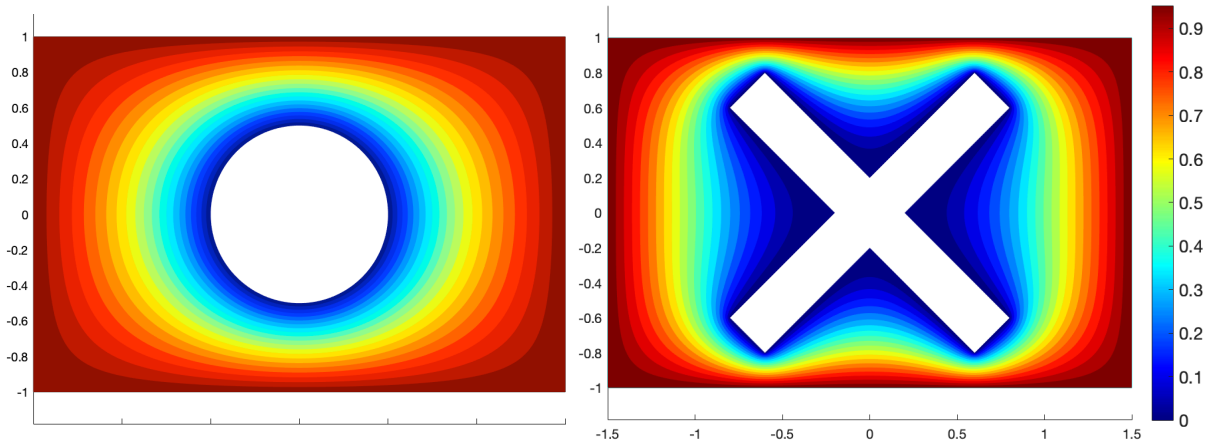


Figure 38: Body-fitted solutions of the case "circle" and "cross".

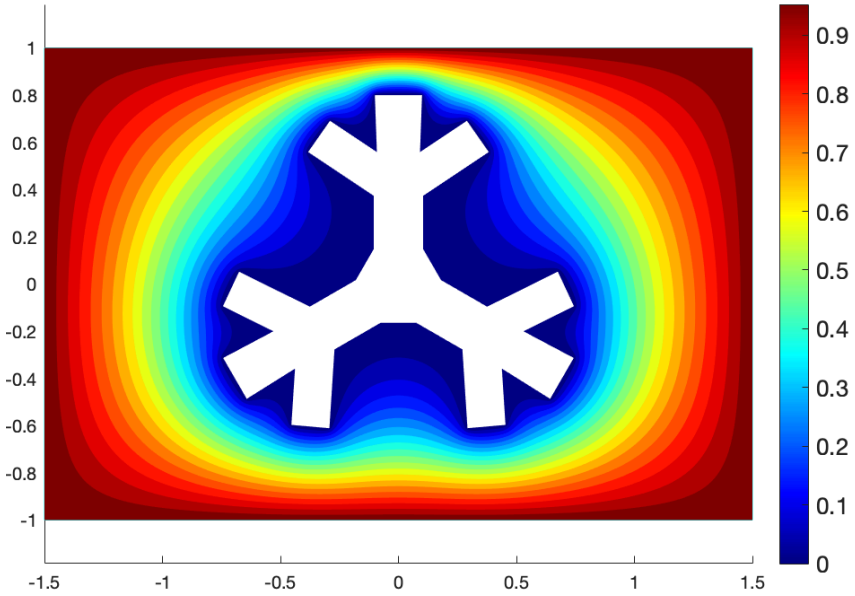


Figure 39: Body-fitted solution of the case called "complex".

1. for each surrogate boundary node to find the best closest point to the true boundary because complex shapes may force the algorithm to fail.;

2. finds the available neighbouring degrees of freedom of each surrogate boundary node to impose correctly the MPCs. In fact, sharp corners may lead to non-available nodes also in the second-level neighbouring nodes (recall the discussion in Chapter 5.5.4).

We first look at the results and then comment on how the SBM algorithm performs. In Figure 40 and 41, we have reported the results of the algorithm that finds the surrogate boundary nodes and their projections onto the true boundary (in green in the figures).

As it is intuitive, the presence of sharp corners in the true boundary shape may lead to a worse approximation of the immersed object shape. The algorithm's robustness permits results even with very coarse meshes, and grid refinement still reduces the area between the surrogate and the true boundary, i.e. as $h \to 0$, $\tilde{\Gamma} \to \Gamma$. Moreover, a clever idea for future applications would be to use a localize refinement around the interface fluid object, so that we can impose better

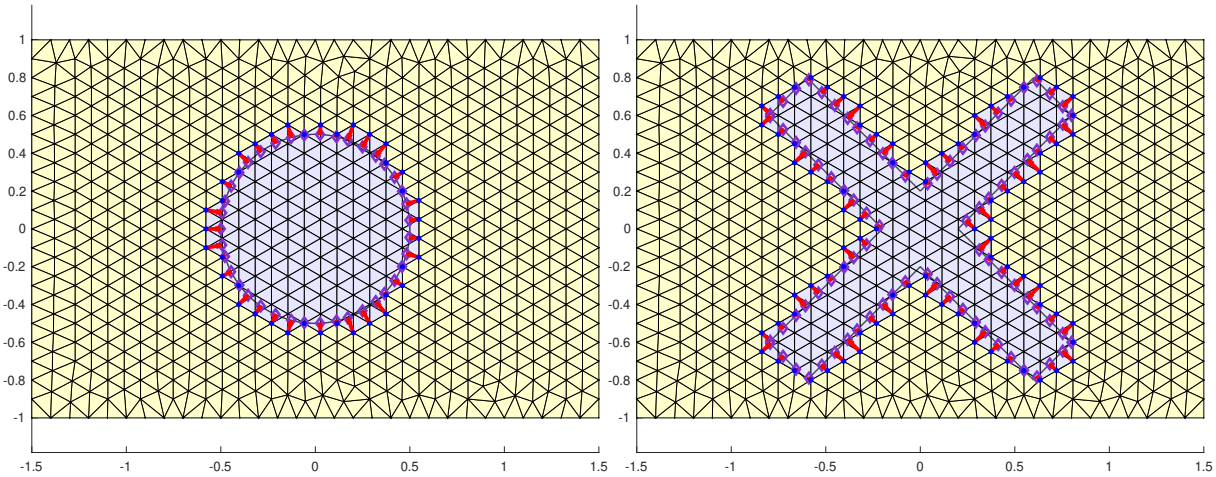


Figure 40: Mesh, surrogate boundary nodes and projections onto the true boundary for the case of the "circle" and "cross" using mesh2 ($h = 0.1$).

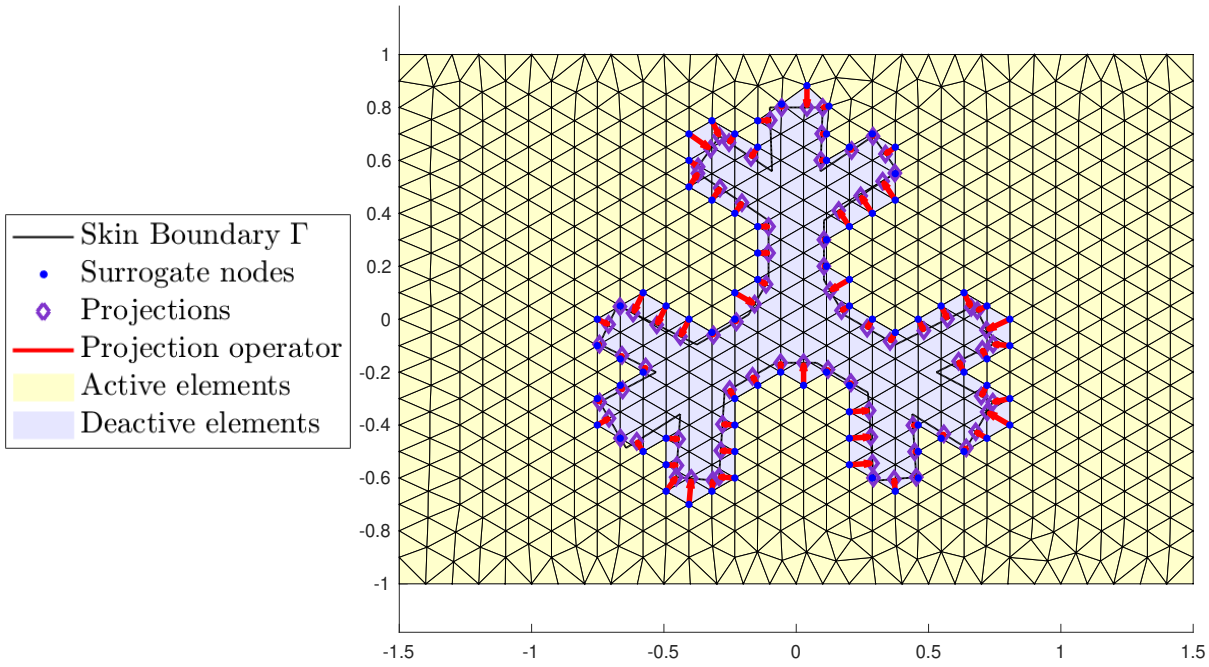


Figure 41: Mesh, surrogate boundary nodes and projections onto the true boundary for the case called "complex" using mesh2 ($h = 0.1$).

the surrogate boundary conditions by finding closer projections.
In Figure 42, for instance, we have used mesh4, but one can see that when sharp angles are present, usually there are no projections for the surrogate boundary nodes. Anyway, we do not expect a lack of precision in this kind of scenario since it has been shown in Chapter 5.4.2 that what keeps the expected convergence is the fact that $|\tilde{\boldsymbol{x}} - \boldsymbol{x_t}| \leq h$, and this is always verified.

We now try to perform a velocity convergence estimate of the SBM solution of the three cases. As said before, since it does not exist an exact solution for these three cases, it is requested to use the body-fitted solution as a reference. In particular, the body-fitted simulation used as the "exact" one is performed with a mesh characteristic length h equal to 0.001. It allows us to consider its error almost negligible even when we compare the solution with mesh8 of Table 7.

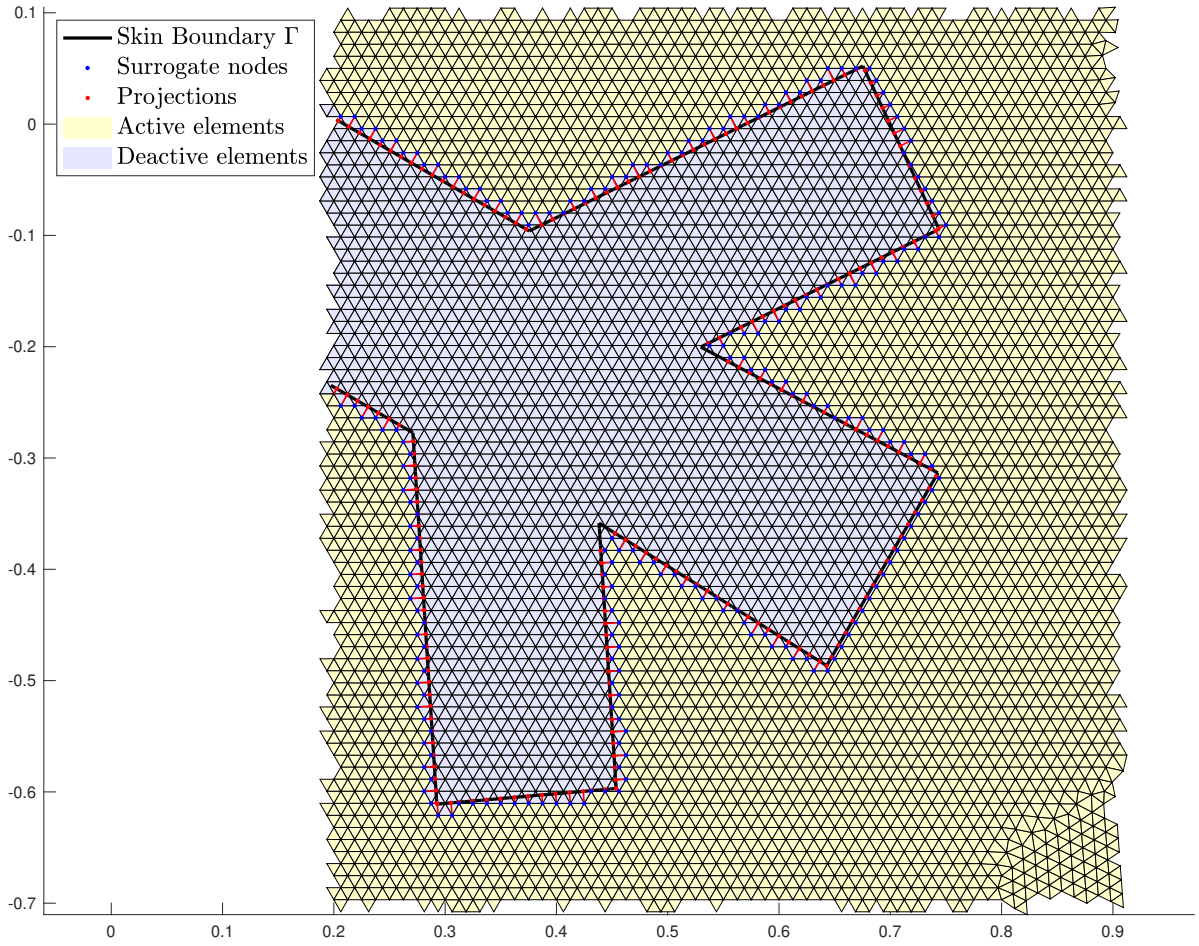


Figure 42: Zoom of the lower right part of the mesh in the case complex using mesh4 ($h = 0.025$).

Furthermore, as done before, we will consider two global convergence indexes, the L^2-norm of the error and the L^∞-norm. Once again, we underline that the error is derived using Equation 175 and Equation 176, therefore, the contribution of the cut elements is not taken into account since they are non-active.

An evolution of the phenomena in which $\tilde{\Gamma} \to \Gamma$, as the characteristic length h is refined, is shown in Figure 43. The results for the L^2-norm convergence error are shown in Figure 44 and in Figure 45 for the L^∞-norm.

From the convergence results in Figure 44 and 45 may observe two main things:

1. the "circle" case is working perfectly, having a perfect second-order convergence;
2. the "cross" and the "complex" cases are very similar in terms of L^2-norm and L^∞-norm of the error. In particular, the obtained slope of the L^2-norm of the error is not second-order, but it is almost equal between the cases "cross" and "complex". Therefore one may conclude that the error related to complicate object shapes does not increase if sharper angles are present.

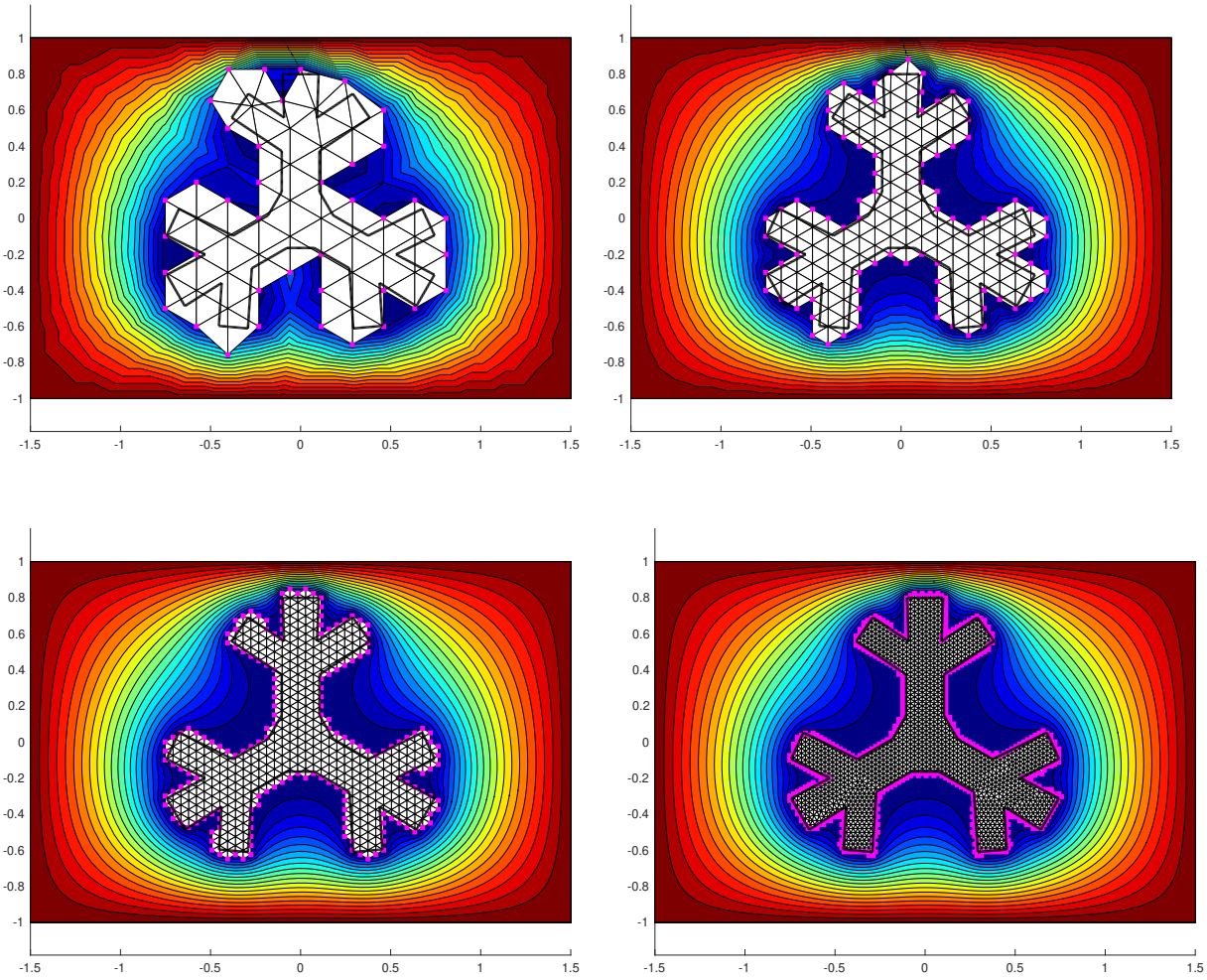


Figure 43: FEM SB solution u^h for the case "complex" using mesh0, mesh1, mesh2 and mesh3 of Table 7. The white elements are the ones that have been deactivated and the nodes marker in pink are the surrogate nodes.

Moreover, we can say that the problem of not having second-order convergence but an approximate value of 3/2 (see Figure 44) is not completely due to the presence of the corners. Applying the same object shape to the example of Chapter 6.2 or Chapter 6.3 (Poisson problem with homogeneous or non-homogeneous boundary condition) where we have the analytical solution, we get perfect second-order convergence.
It means that the fact that we get $\approx h^{3/2}$ convergence is due to the discontinuity in terms of smoothness solution u present in the case "cross" and "complex". The exact solution at the corner locations has a kind of discontinuity. The body-fitted formulation is not affected by this because in each corner location, there is a boundary node and, thus, it can afford a discontinuity of this type. The SBM does not have a boundary node at the exact corner location, so the presence of discontinuities decreases the convergence order. To enforce this theory, we may refer to the pointwise error with respect to the body-fitted reference solution. As will be shown, the peaks of the error are at the external corners where this smoothness discontinuity is stronger.

We report in Table 8 the data of the previous graph of $||\text{err}||_{L^2(\Omega)}$ for the three cases. Also, the slope of each segment and the average corresponding to the L^2-error convergence of Figure 44 is reported in Table 9.

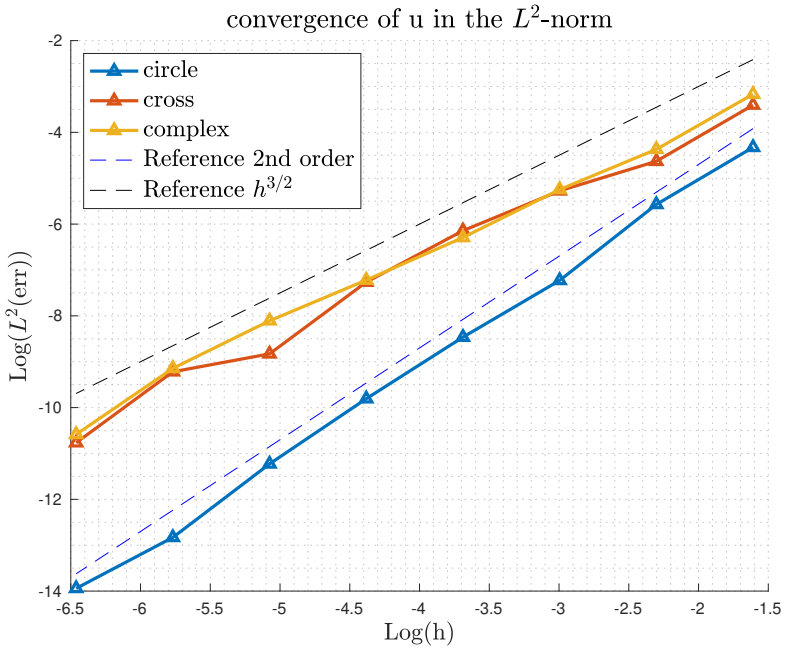


Figure 44: Convergence study of the error in the L^2-norm for the SBM method, using as a reference the body-fitted formulation, in the three cases of Figures 38 and 39.

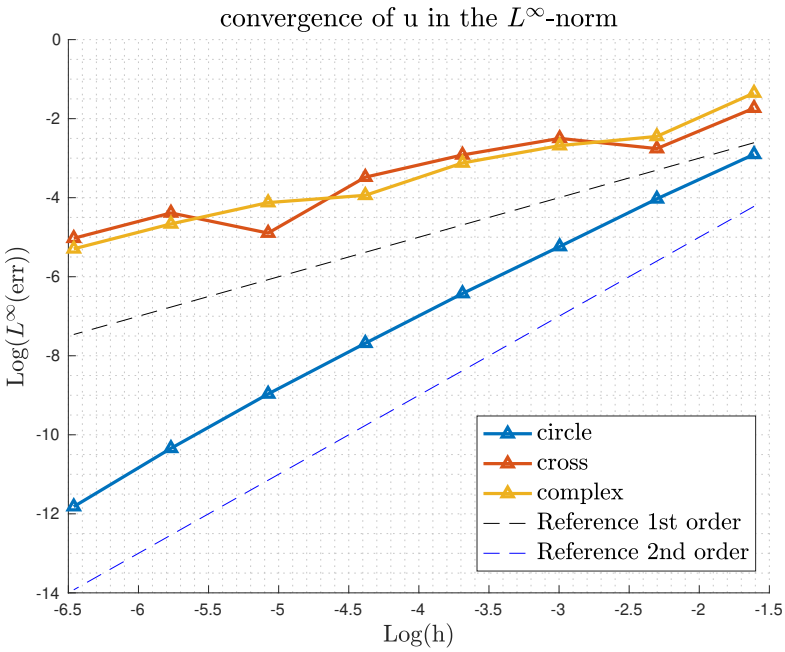


Figure 45: Convergence study of the error in the L^∞-norm for the SBM method, using as reference the body-fitted formulation, in the three cases of Figures 38 and 39.

In the images in Figure 46 and 47, we report the module of the error nodes by nodes between the SBM using mesh3 and the reference body-fitted solution using a very refined grid ($h = 0.001$). As we can see, the module of the error has some peaks located at the corners where the solution has a sort of discontinuity due to the sharp geometry. The different order of magnitude of the error between smooth geometry (circle) and sharp skin boundary (cross and complex) is evident in Figure 46, Figure 47 and in the error Table 8. This difference is due to the contributions of the error at the external sharp corners.

	$\|\|\text{err}\|\|^{\text{circle}}_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|^{\text{cross}}_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|^{\text{complex}}_{L^2(\tilde{\Omega})}$
mesh1	0.01320	0.03319	0.04205
mesh2	0.003804	0.009749	0.01265
mesh3	0.0007260	0.005115	0.005262
mesh4	0.0002107	0.002145	0.001849
mesh5	$5.507e-05$	0.0006992	0.0007310
mesh6	$1.329e-05$	0.0001464	0.0003010
mesh7	$2.677e-06$	$9.907e-05$	0.0001067
mesh8	$8.813e-07$	$2.110e-05$	$2.528e-05$

Table 8

	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	$6 \to 7$	$7 \to 8$	**average**
slope_{L^2} circle	1.7947	2.3896	1.7848	1.9359	2.0501	2.3120	1.6034	1.9815
slope_{L^2} cross	1.7676	0.9304	1.2538	1.6171	2.2556	0.5637	2.2311	1.5170
slope_{L^2} complex	1.7329	1.2655	1.5084	1.3393	1.2799	1.4959	2.0778	1.5285

Table 9

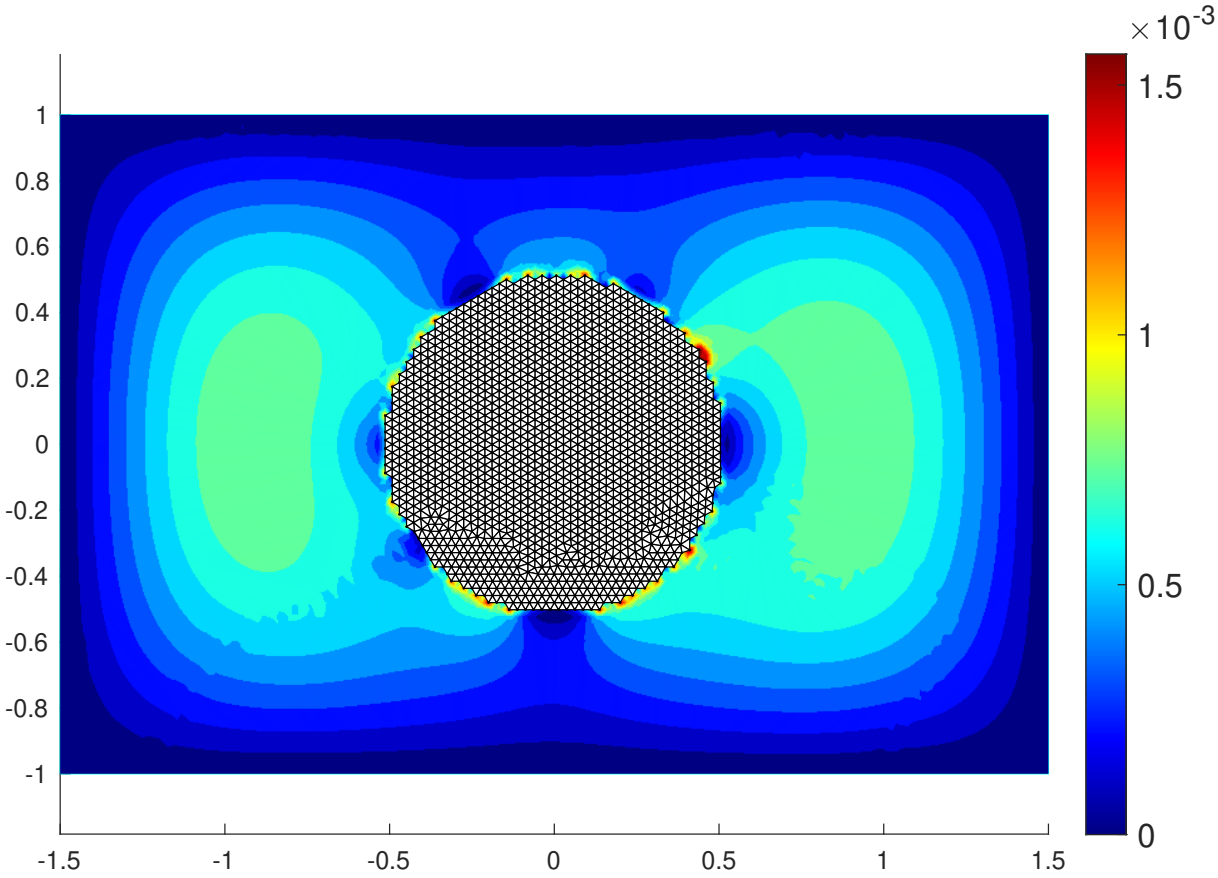


Figure 46: Comparison of the module of the error between the SBM simulations using mesh3 and body-fitted solution as reference, in the cases "cross" and "complex".

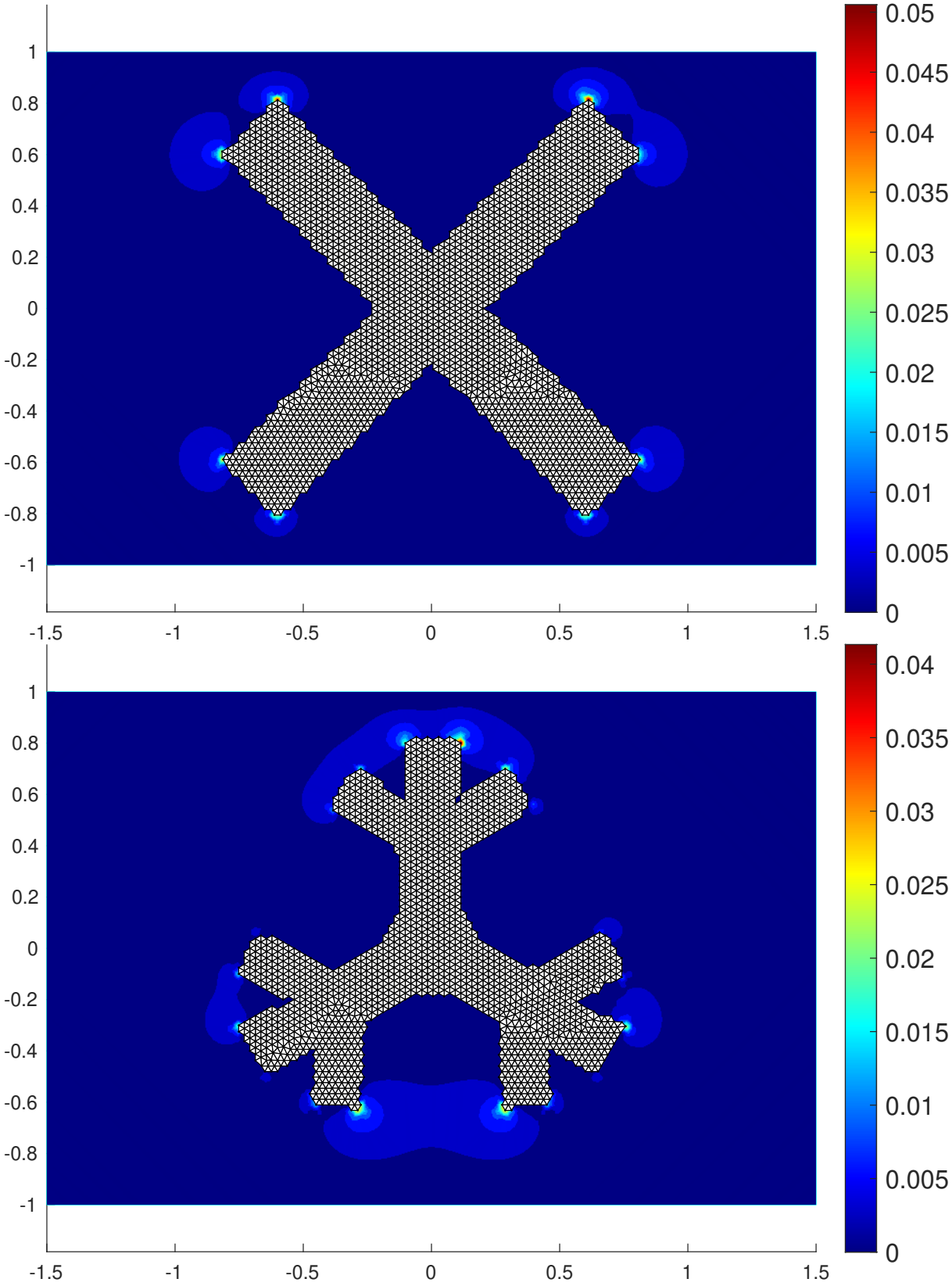


Figure 47: Comparison of the module of the error between the SBM simulations using mesh3 and body-fitted solution as reference in the cases "circle".

To prove that the slope of the convergence estimate is equal to 3/2 in the cases "cross" and "complex" is caused by the exact solution u and its smoothness discontinuity, we now take the following example. Consider the problem case of Chapter 6.3 where a two-dimensional disk has been taken as background mesh Ω; moreover, we use the same differential equation, i.e.:

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= 1 && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= \overline{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma_1 \\ u(\boldsymbol{x}) &= \overline{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma_2 \end{aligned} \tag{182}$$

Where $\overline{u}$ is also the exact solution of the problem and it is the following:

$$\overline{u}(x,y) = \frac{1}{4}\left(9 - x^2 - y^2 - 2\ln 3 + \ln(x^2+y^2)\right) + \frac{1}{4}\sin x \sinh y \tag{183}$$

Then, instead of the embedded boundary Γ_2 to be a circle, as in Chapter 6.3, we use the three shapes: circle, cross and complex. The Γ_2 is embedded, and a surrogate boundary $\tilde{\Gamma}_2$ is considered for the imposition of the MPCs. Γ_1 is the external boundary of Ω and conforming Dirichlet boundary conditions are used.
If sharp corners are a problem for the SB algorithm, then we expect a convergence estimate in the L^2-norm of the error less than second-order.

The convergence studies of the three shapes have been performed using the meshes in Table 2 and computing the error using the analytical solution that is now available.
In the two panels of Figure 48, we report the convergence studies in the L^2- and L^∞-norm. Comparing them with the one of Figure 35 in Chapter 3, one might note that the presence of a complicated shape does not affect the L^2-norm of the error that keeps the second-order convergence. It is true that as the geometry becomes more complex, the global error is, in general, slightly larger and more irregular. Anyway, convergence is still the one expected, and the three lines almost overlapped in the left image of Figure 48.

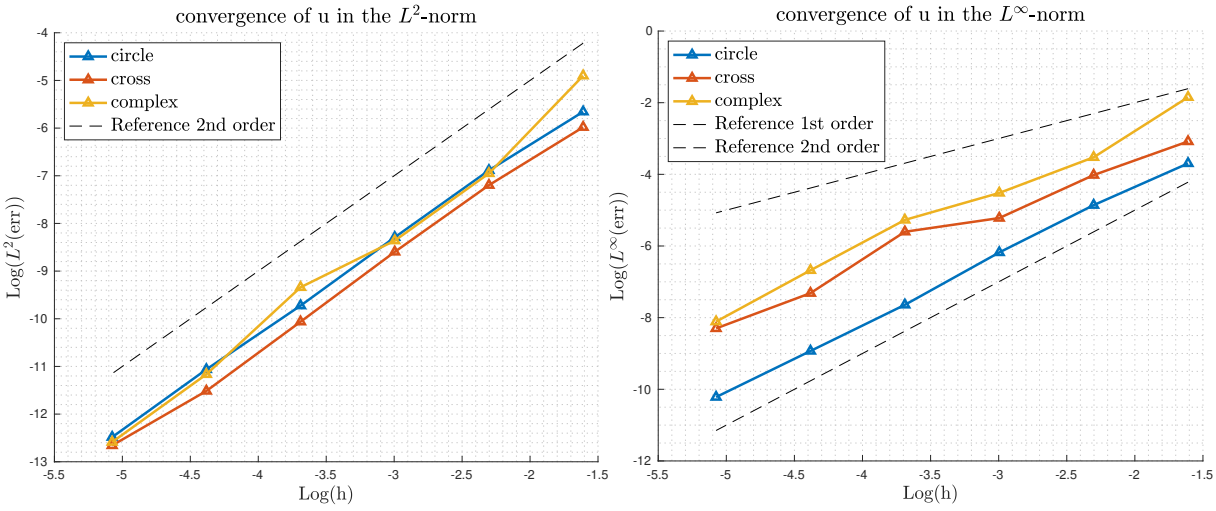


Figure 48: On the left is shown the convergence study in the L^2-norm of the error of the SBM in the case of the Poisson problem with the non-homogeneous boundary condition of Equation 177, with the three embedded shape cases. On the right, the same comparison is done in the L^∞-norm.

Instead, the L^∞-norm is more affected by the presence of complex geometries. The global L^∞ error estimates in the case cross and complex are more irregular, and their slopes may vary

considerably. In general, we expect to see a few critical points where, due to the problems of searching available points for the MPCs, the approximation of the modified Dirichlet boundary conditions is proportional to h instead of h^2. These critical points may be located where the geometry has sharp corners where all the neighbours of a surrogate node are either cut or other surrogate nodes.
Table 10 and Table 11 refer to the convergence studies of Figure 48.

	$\|\|\text{err}\|\|^{\text{circle}}_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|^{\text{cross}}_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|^{\text{complex}}_{L^2(\tilde{\Omega})}$
mesh1	0.003493	0.002526	0.007431
mesh2	0.001021	0.0007486	0.0009583
mesh3	0.0002506	0.0001846	0.0002338
mesh4	$5.976e-05$	$4.264e-05$	$8.782e-05$
mesh5	$1.568e-05$	$9.976e-06$	$1.415e-05$
mesh6	$3.789e-06$	$3.183e-06$	$3.415e-06$

Table 10

SLOPE	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	**average**
circle(L^2-norm)	1.7743	2.0262	2.0683	1.9287	2.0425	1.9680
cross(L^2-norm)	1.7548	2.0192	2.114	2.0960	1.9910	1.9244
complex(L^2-norm)	2.9547	2.0330	1.4105	2.5919	2.0778	2.1962
circle(L^∞-norm)	1.6805	1.9054	2.1073	1.8557	1.8373	1.8772
cross(L^∞-norm)	1.3535	1.7306	0.5527	2.4711	1.4208	1.5057
complex(L^∞-norm)	2.4175	1.4314	1.0865	1.9862	1.9060	1.7655

Table 11

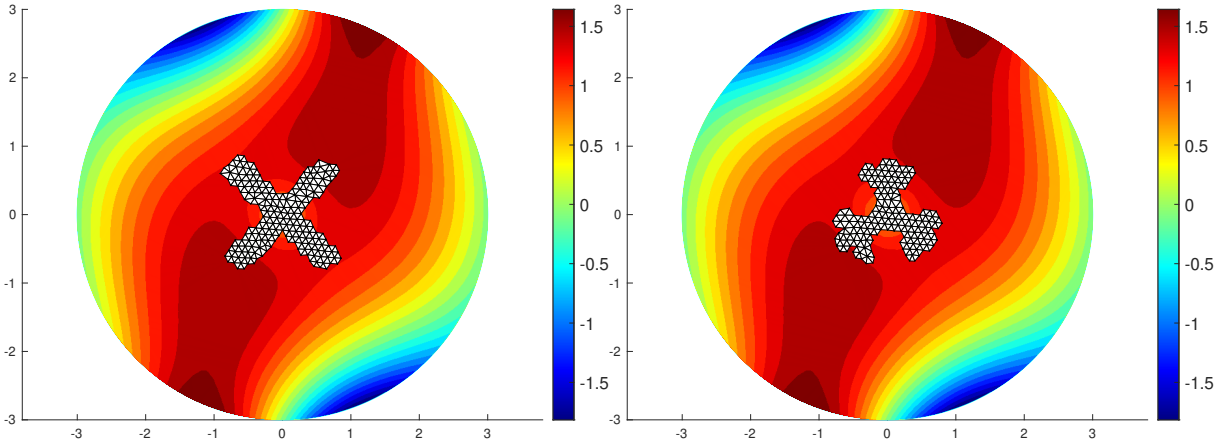


Figure 49: The SB solution u^h of the cases "cross" and "complex" using mesh2 ($h = 0.1$).

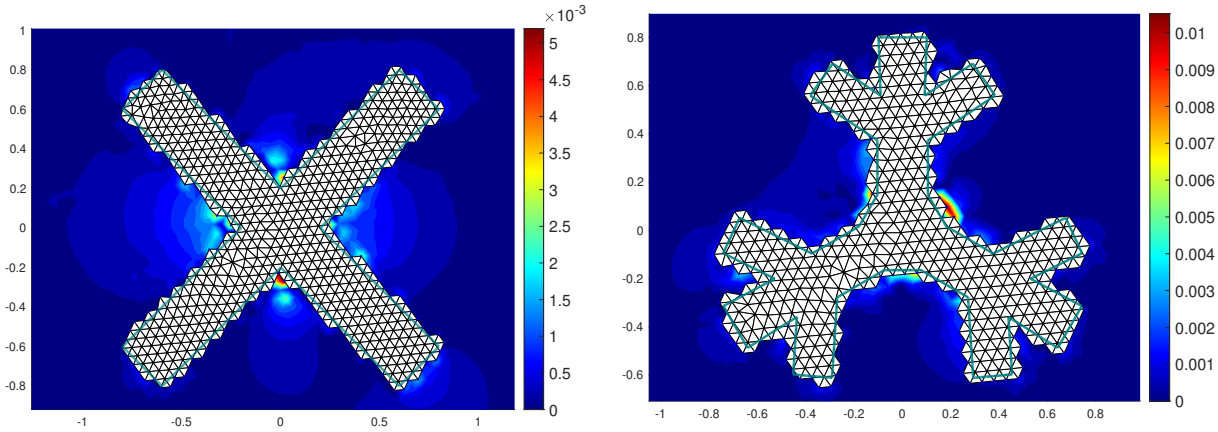


Figure 50: Zoom of the contour of the pointwise module of the error in the cases "cross" and "complex" using mesh3.

Figure 50 also shows the pointwise module of the error in cases with sharp corners. Because the exact solution has all the possible smoothness quality in the computational domain $\tilde{\Omega}$, it is interesting to note that the peaks of the error module are no longer in the external corners as they were in Figure 47. Now the peaks are located in the internal corners. The fact that there are error peaks that are responsible for the larger L^∞-norm of the error (see left panel Figure 48) in the internal corners is due to the applications of MPCs for imposing the Dirichlet surrogate boundary conditions. In fact, in the internal corners, few neighbours nodes are available for computing the gradient needed for the imposition of MPCs.

This phenomenon is the main lack that the method has when dealing with complicated object shapes. But, as we can see from the convergence study in the L^2- and L^∞-norm, the algorithm is performing optimally even if there may be some larger error peaks. Anyway, the method seems to converge at least as $h^{3/2}$ when the kind of discontinuities are present in the solution and as h^2 when the solution is smooth.

In conclusion, we may say that the SB method developed in Kratos is robust in the case of a stationary Poisson problem, even with complex geometries. As with all the unfitted methods, it requires finer meshes if one wants to achieve high precision around the embedded boundaries. Moreover, the introduced MPCs approach is also robust, but it may lead to a lack of precision in internal corners, which is where the problem of available nodes for the computation of the gradient arises.

6.5 Transient Poisson problem

Consider now the case of an evolution of a diffusion problem. We can model the phenomenon with a transient Poisson problem. The differential problem has the additional time-dependent term $\partial u/\partial t$, and the initial conditions $u(\boldsymbol{x}, t = 0)$ must be provided.

$$\begin{aligned} &\frac{\partial u}{\partial t} - \Delta u = f(\boldsymbol{x}, t) && \boldsymbol{x} \in \Omega,\ t > 0 \\ &u(\boldsymbol{x}, t) = \overline{u}(\boldsymbol{x}, t) && \boldsymbol{x} \in \Gamma,\ t > 0 \\ &u(\boldsymbol{x}, 0) = \tilde{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \end{aligned} \tag{184}$$

Where the boundary Γ is considered as split into two parts: Γ_1 where conforming boundary conditions are required and Γ_2 that is embedded and the SB method will be used. Thus the algorithm will find the surrogate boundary $\tilde{\Gamma}_2$, the projections and will impose the MPCs. Recall that now iterations in time are needed. Therefore, as mentioned in Chapter 5.7.2, a transient solver is necessary. The transient solver from which we have derived the `SBMConvectionDiffusionTransientSolver` is called `ConvectionDiffusionTransientSolver`. The main difference from the SB stationary solver is that it has the imposition of the MPCs at each time-step. The boundary conditions $\overline{u}(\boldsymbol{x}, t)$ at Γ_2, in general, may depend on time, and therefore the Dirichlet values at the projections may vary at each time-step.

It would be likely to compare the SB solution with the analytical one at a certain time $t = t^*$. To do so, we can use a common trick and first choose the analytical solution $u(\boldsymbol{x}, t)$ and, then, derive the forcing term $f(\boldsymbol{x}, t)$ such that the differential equation of the Poisson transient problem is satisfied. Consider as the analytical solution the following:

$$u(x, y, t) = t^2 \sin(2x) \cos(2y) \tag{185}$$

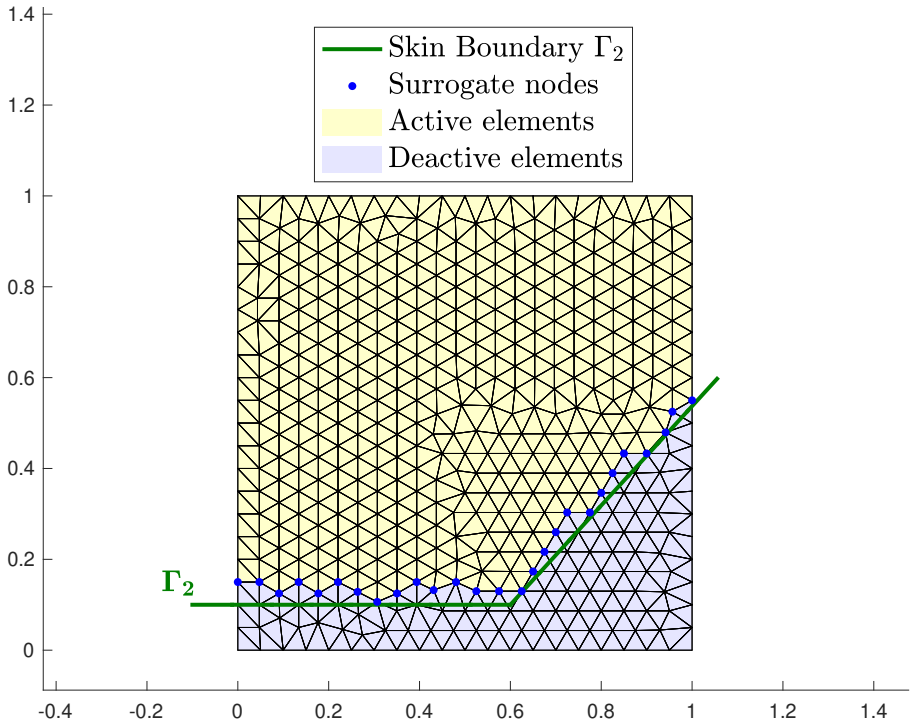


Figure 51: Geometry of the transient Poisson problem and mesh2 ($h = 0.05$).

In particular, in the figure, the algorithm has already found the surrogate boundary nodes (blue dots) and the active and deactivated elements. The boundary conditions on Γ_1 and Γ_2 are taken from the exact solution $u(x, y, t)$ in Equation 185.
This exact solution is not in our finite element space of piece-wise linear functions and can satisfy the system 184 if we consider the following forcing function:

$$f(x, y, t) = 2t(1 + 4t)\sin(2x)\cos(2y) \tag{186}$$

In fact, u satisfies $\partial u/\partial t - \Delta u = f(\boldsymbol{x}, t)$ for every choice of Ω. Thus, one can choose any regular domain Ω and consider as Γ_2 an immersed object or an embedded external boundary. To show the robustness of the SB solver, in the case of an external embedded boundary, a portion of the external boundary of Ω will be treated with the SB method.

Consider Figure 51 that represents the geometry of the problem. Ω is a square computational domain [0,0] $\times$ [1,1], and it is cut by a skin boundary, in green in the figure, that represents the true boundary Γ_2. The rest of the boundary of the yellow domain (active elements) is Γ_1, and is where standard body-fitted boundary conditions are considered.

Moreover, the following set of square computational mesh domains is considered for the convergence study:

	mesh1	mesh2	mesh3	mesh4	mesh5	mesh6	mesh7
h	0.1	0.05	0.025	0.0125	0.00625	0.003125	0.0015625

Table 12

Furthermore, a time discretization and a time integration scheme are needed since the problem is now time-dependent. The θ-scheme, already present in Kratos, has been extended to the SB algorithm; in particular, it is crucial to apply it in the flux through the surrogate boundary described in Chapter 5.6.
Therefore, consider a Crank-Nicolson time integration scheme setting $\theta = 0.5$. This context is focused on the SB method and its spatial error and not the one coming from the integration in time. Thus, we have to set a time-step such that the error related to the time integration is negligible with respect to the spatial one.

We consider a time $t^* = 2.0$ where we compute the error estimations comparing the exact and SB solutions. The time step used is $\Delta t = 0.1$, where it has been checked that the plateau of the time integration scheme has already been reached.

In Figure 52, the solution and the pointwise error are reported by applying the SB algorithm using mesh2 and mesh3. It may be interesting to note that the main spatial error derives from the imposition of the modified Dirichlet boundary conditions. In particular, looking carefully, one may notice that the error peaks are located where the distance vector $\boldsymbol{d}$ is larger. That is when the surrogate nodes are far from the true boundary.

The introduction of modified Dirichlet boundary conditions really affects the error, but what matters is that this error is consistent with the rest of the formulation. If the second-order convergence of the error in the L^2-norm is achieved, then the SB method is consistent with the linear finite element method we are using.

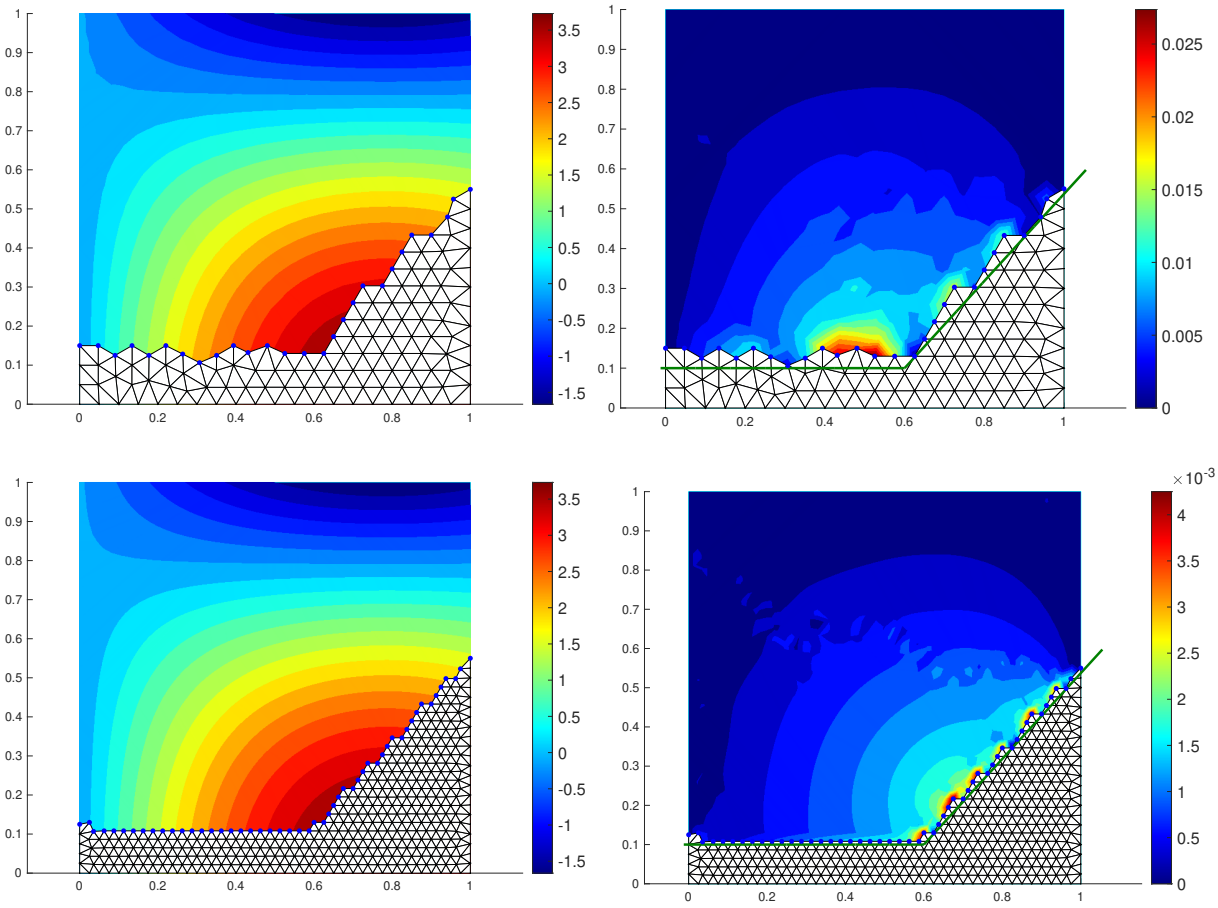


Figure 52: In the images on the left, the SB solutions of the Poisson stationary problem using mesh2 and mesh3 are represented. On the right are reported the corresponding pointwise errors with respect to the exact solution $u(x, y, t)$. The comparisons are performed at $t^* = 2.0$, using a Crank-Nicolson time integration scheme with $\Delta t = 0.1$.

Figure 53 represents the convergence study of the example we are considering. Moreover, the data and the analysis of the convergence are reported in Table 13 and Table 14.

Concluding this chapter, we may state that transient Poisson problems are perfectly compatible with the Kratos SB algorithm with MPCs. Moreover, the SB method works as expected in the case of external boundaries and not only with immersed embedded objects. One more observation is necessary, in fact looking at Figure 53 and Table 14, it is clear that the convergence study is not homogeneous. There are parts where the slope is 1.4 and others where it is equal to 2.6. This variance is due to the nature of the SB method; it may happen that in one mesh configuration, there are many surrogate nodes that are far from the skin boundary. It is also true that what is important is the global scale, where the SBM in the L^2-norm on average has a slope of ≈ 2.

	$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$
mesh1	0.02199	0.08109
mesh2	0.005056	0.02919
mesh3	0.0007973	0.004534
mesh4	0.0003517	0.001710
mesh5	$5.578e-05$	0.0003400
mesh6	$1.090e-05$	$8.417e-05$
mesh7	$4.135e-06$	$2.963e-05$

Table 13

SLOPE	L^2-norm	L^∞-norm
$1 \rightarrow 2$	2.1207	1.4738
$2 \rightarrow 3$	2.6648	2.6868
$3 \rightarrow 4$	1.1806	1.4064
$4 \rightarrow 5$	2.6564	2.3307
$5 \rightarrow 6$	2.3551	2.0141
$6 \rightarrow 7$	1.3989	1.5061
average	2.0627	1.9030

Table 14

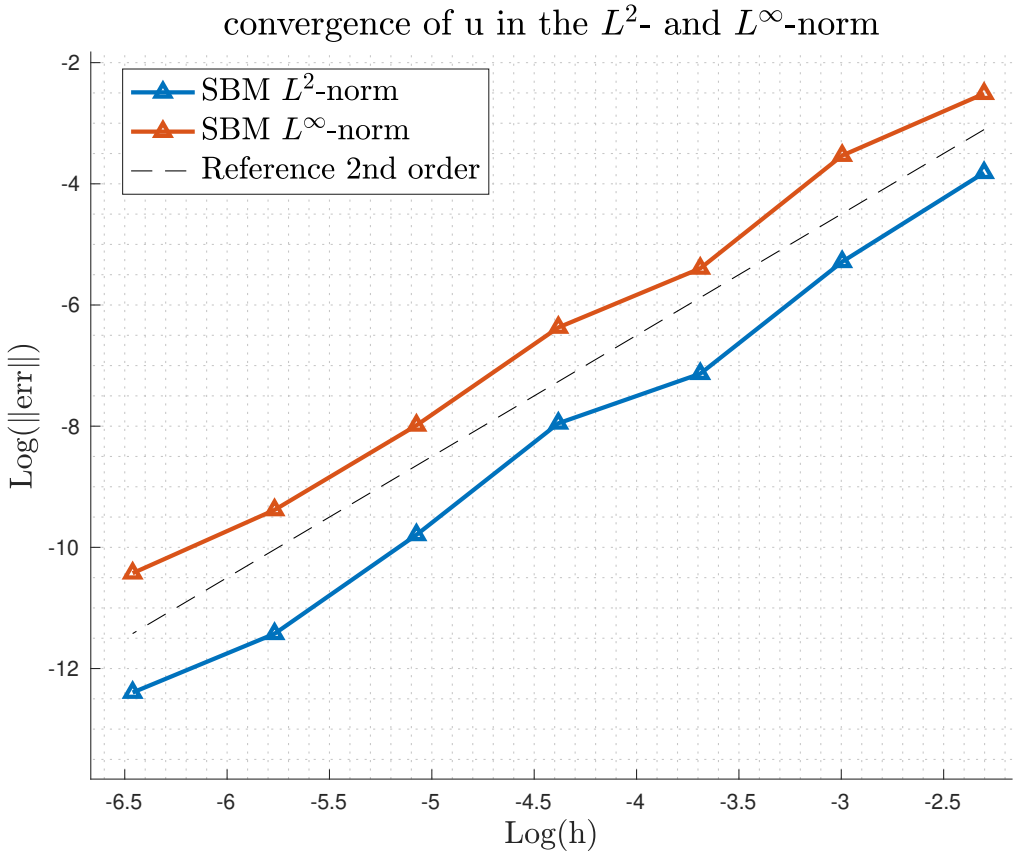


Figure 53: Convergence study of the error in the L^2- and L^∞-norm of the SB solution using the set of meshes in Table 12, at $t^* = 2.0$, using a Crank-Nicolson time integration scheme with $\Delta t = 0.1$.

6.6 Stationary convection-diffusion problem

Now move one step further considering the convection-diffusion case in both the stationary and transient regimes. The general case, considering that only Dirichlet boundary conditions are included, reads as follows:

$$\begin{aligned} &\frac{\partial u}{\partial t} + \boldsymbol{v} \cdot \nabla u - \Delta u = f(\boldsymbol{x}, t) && \boldsymbol{x} \in \Omega,\ t > 0 \\ &u(\boldsymbol{x}, t) = \overline{u}(\boldsymbol{x}, t) && \boldsymbol{x} \in \Gamma,\ t > 0 \\ &u(\boldsymbol{x}, 0) = \tilde{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \end{aligned} \tag{187}$$

Where $\tilde{u}(\boldsymbol{x})$ are the initial conditions of the problem that are essential in the transient case. The unknown function $u(\boldsymbol{x}, t)$ might be only a function of $\boldsymbol{x} \in \Omega$. In this case, we call the problem stationary, and the system of equations 187 can be rewritten as the following:

$$\begin{aligned} &\boldsymbol{v} \cdot \nabla u - \Delta u = f(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \\ &u(\boldsymbol{x}) = \overline{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma \end{aligned} \tag{188}$$

To check our SBM method in the case of a convection-diffusion problem, we will simulate cases where an immersed boundary is present in the computational domain. In particular, we will first start with a stationary problem.

$$u(x, y) = \sin(x)\cos(y) + \log(1 + x^2 + y^2) \tag{189}$$

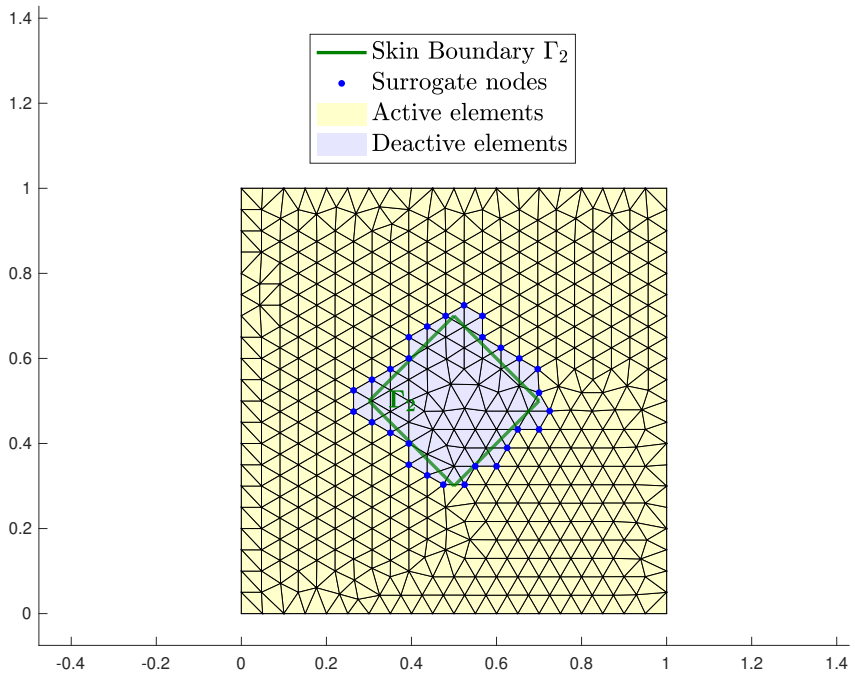


Figure 54: Geometry of the stationary convection-diffusion problem and mesh2 ($h = 0.05$).

The exact solution is not in our finite element space of piece-wise linear functions and can satisfy the system 188 if, for example, we consider a uniform velocity vector field $\boldsymbol{v} = [1, 1]$ and the following forcing function:

$$f(x, y) = \frac{2\left(-2 + (x + y)\left(1 + x^2 + y^2\right)\right)}{\left(1 + x^2 + y^2\right)^2} + \cos(x + y) + 2\cos(y)\sin(x) \tag{190}$$

It can be checked that u satisfies $\boldsymbol{v}\cdot\nabla u - \Delta u = f(\boldsymbol{x})$ for every choice of Ω. Thus, for instance, we can choose any regular domain Ω and consider an immersed object. To check the FEM solution using the SB method, we consider at the external boundary of Ω standard Dirichlet boundary conditions and modified SB Dirichlet boundary conditions at the embedded object interface. The immersed object is taken as a square centred in the square domain used in the previous chapter. Thanks to the nature of the SB method the meshes, needed for the convergence study, can be taken exactly the same as in the previous chapter, where the characteristic lengths h's were reported in Table 12.

In Figure 54, the geometry of the stationary convection-diffusion problem is reported applying the algorithm to mesh2. Γ_2 indicates the square true boundary considered as embedded. As usual, the surrogate nodes have been computed thanks to the level set function, and the cut and internal elements have been set to be de-active.

In the images of Figure 55 the results for the mesh2 and mesh4 and their corresponding pointwise errors are reported. The convergence study is performed using the seven meshes of Table 12 and the results in terms of L^2- and L^∞-norm are reported in Figure 56, Table 15 and Table 16.

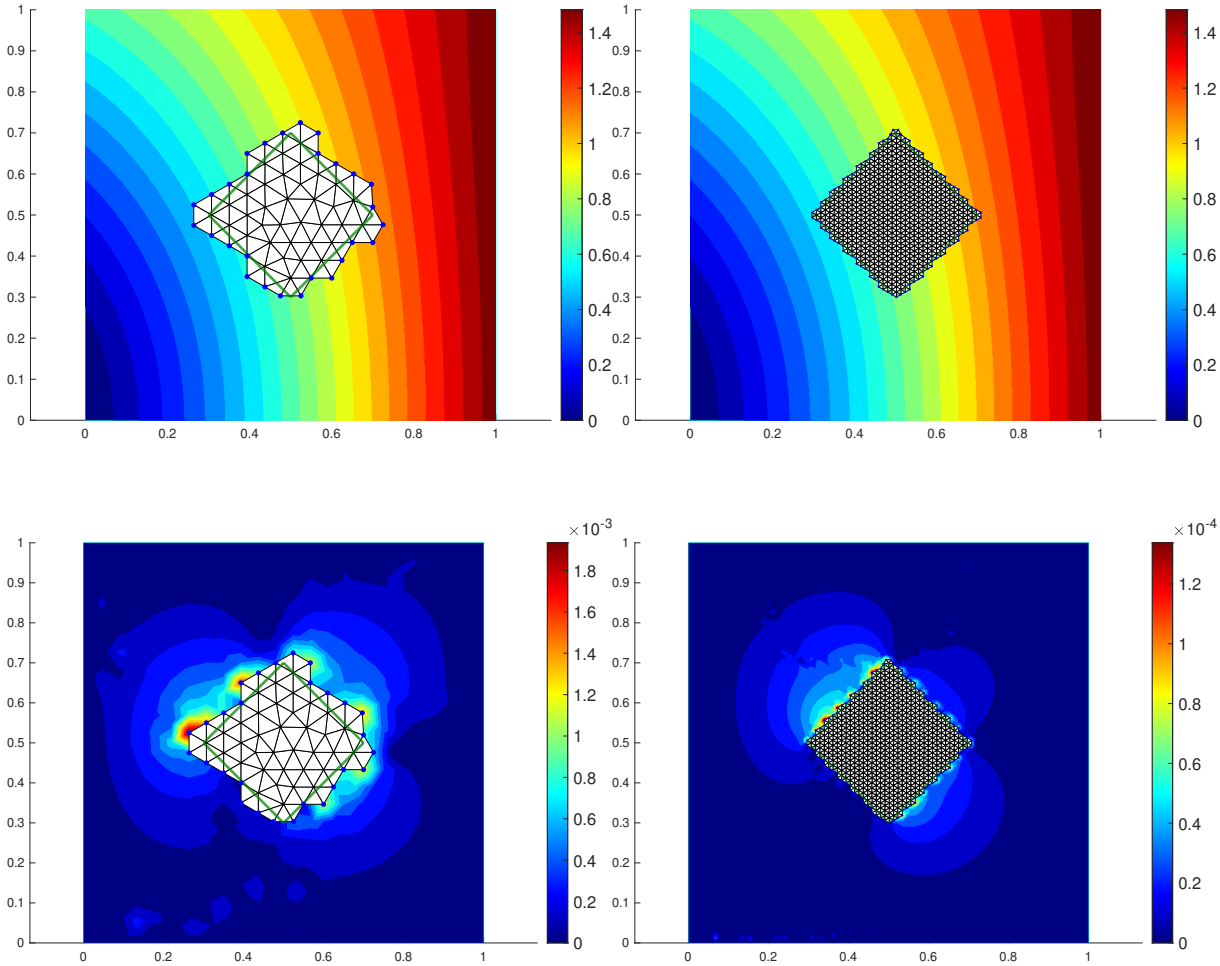


Figure 55: In the two images on the upper part, the SB solutions of the Poisson stationary problem using mesh2 and mesh4 are represented; below are the corresponding pointwise errors with respect to the exact solution $u(x, y)$.

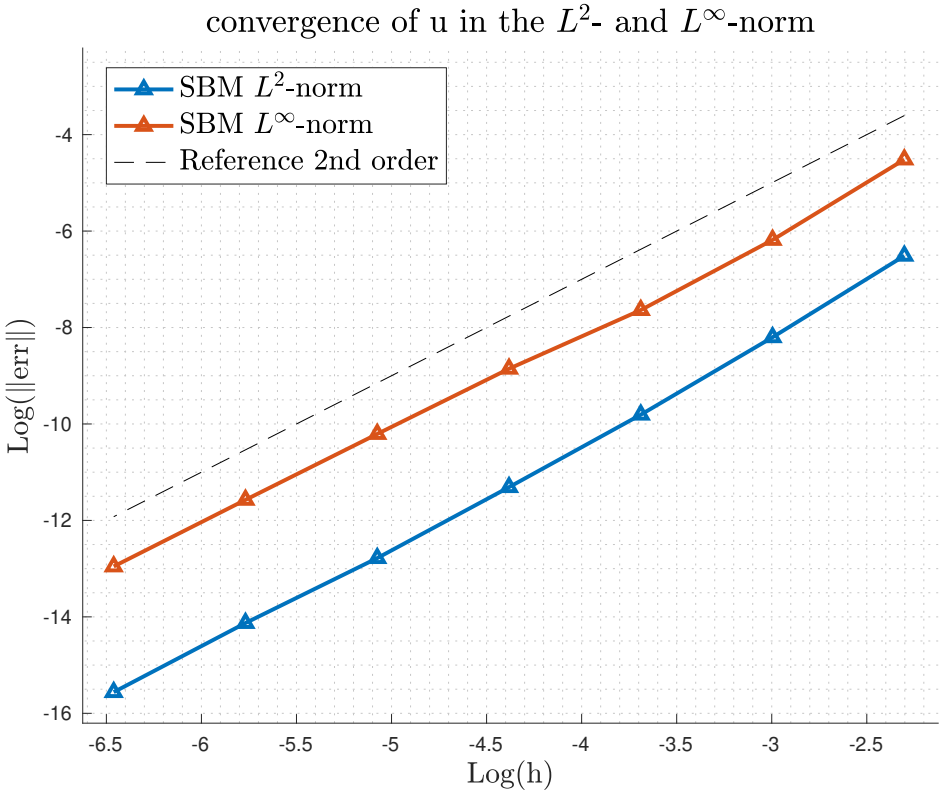


Figure 56: Convergence study of the error in the L^2- and L^∞-norm of the SB solution of the stationary convection-diffusion problem using the set of meshes in Table 12.

	$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$
mesh1	0.001484	0.01092
mesh2	0.0002740	0.002062
mesh3	$5.502e-05$	0.0004815
mesh4	$1.224e-05$	0.0001426
mesh5	$2.811e-06$	$3.688e-05$
mesh6	$7.316e-07$	$9.409e-06$
mesh7	$1.745e-07$	$2.365e-06$

Table 15

SLOPE	L^2-norm	L^∞-norm
$1 \to 2$	2.4376	2.4054
$2 \to 3$	2.3165	2.0985
$3 \to 4$	2.1680	1.7552
$4 \to 5$	2.1224	1.9513
$5 \to 6$	1.9422	1.9709
$6 \to 7$	2.0673	1.9921
average	2.1757	2.0289

Table 16

Consider now another example of a stationary convection-diffusion problem, but this time the embedded boundary is external, and it is a smooth curve. Moreover, consider a physical differential problem and not a manufactured solution; the conditions are the following:

$$\begin{aligned} f(x,y) &= 10.0 \\ \overline{u}(x,y) &= 1 - x \\ \boldsymbol{v} &= [5, 10] \end{aligned} \tag{191}$$

An exact solution for the differential problem 188 with conditions in Equation 191 is not available in particular because the external boundary of the domain is complex. Consider Figure 57, where the geometry of the problem is described using a body-fitted approach.

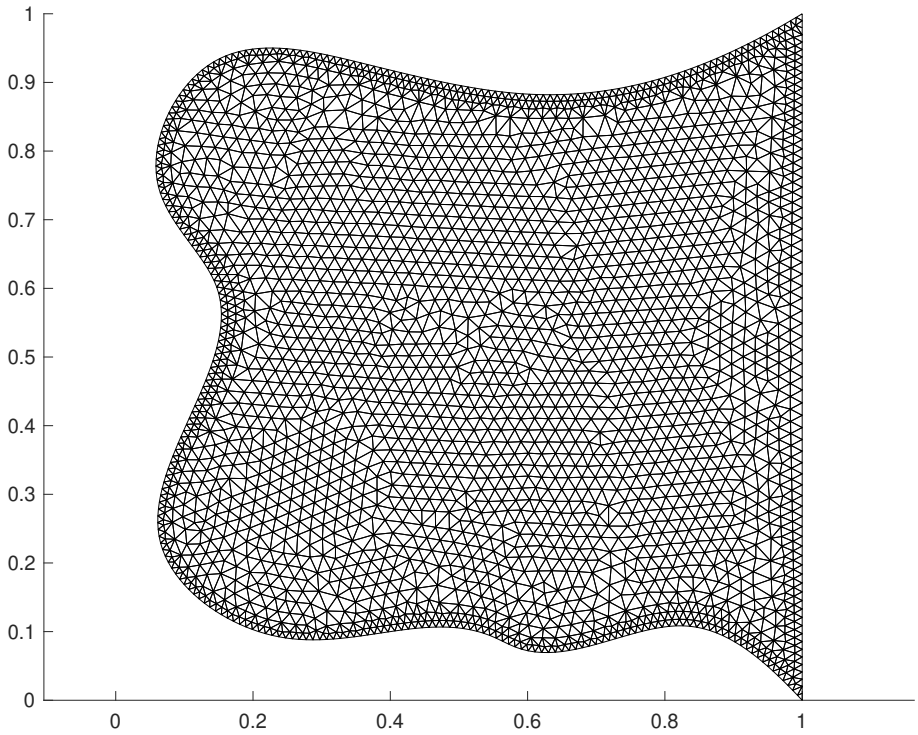


Figure 57: Body-fitted mesh of the domain Ω, $h = 0.02$.

The figure shows a body-fitted mesh with characteristic length $h = 0.02$. The body-fitted simulation will be used as a reference for the convergence study. For this reason, since we want to test the SB method with very refined meshes, we must perform the body-fitted reference simulation using a finer grid. We consider a mesh size of $h = 0.001$.

In a real application, the meshing process of a curve border may bring problems or lead to a bad mesh quality. One of the advantages of the SB method is that we can reuse the simple square meshes used in the previous examples, reported in Table 12, and import as skin boundary the curve line.
In Figure 58, the mesh2 has been used and, applying the SB algorithm, we obtained the active/deactive elements and the surrogate boundary nodes.

Actually, what we have done is consider the boundary of the problem $\Gamma = \partial\Omega$ as made by two complementary parts Γ_1 and Γ_2. The right straight side of the domain Ω in Figure 57 has been called Γ_1. This side of the square is easy to mesh and, therefore, we can treat it with a body-fitted approach. Whereas the rest of the boundary, that is curvilinear, $\Gamma_2 = \Gamma \setminus \Gamma_1$ is considered

embedded. The situation is described in Figure 58, where a square domain $[0,0] \times [1,1]$, that contains the body-fitted domain, is taken. Then, the embedded curve boundary Γ_2 is imported and used in the SB algorithm.

The yellow elements are the active ones, and in the theory of the SB method they form the surrogate domain $\tilde{\Omega}$. Instead, the square composed of active and de-active elements has been called the computational domain Ω.

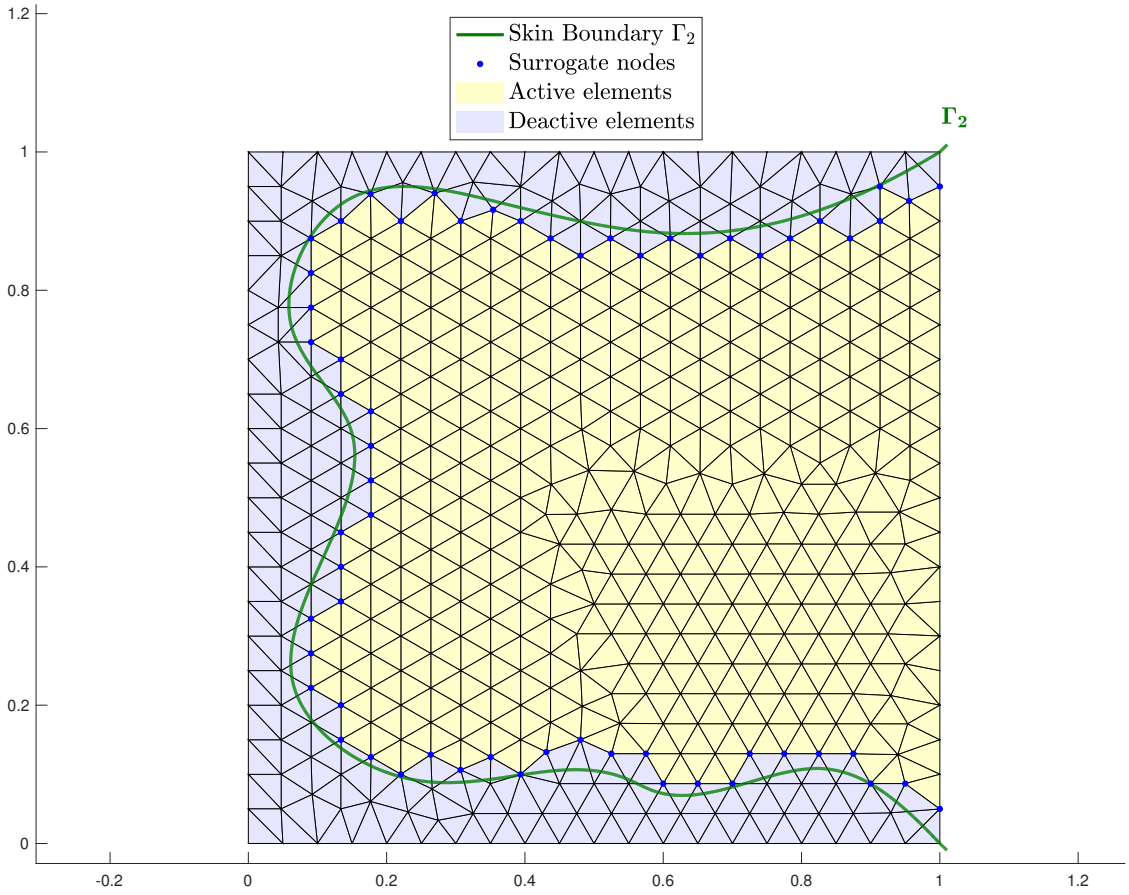


Figure 58: Geometry of the SB case using mesh2: $h = 0.05$.

Figure 59 compares the SB results with the body-fitted solution. In particular, one can see how refining the grid leads to the process where $\Gamma \to \tilde{\Gamma}$ are $h \to 0$. Already using mesh4, the surrogate boundary $\tilde{\Gamma}_2$ approximates with good precision the true embedded boundary Γ_2. In the convergence study, we apply the refinement process up to mesh7, which has a characteristic length h equal to 0.0015625.

The converge study is performed using instead of the analytical solution, which is not available, the body-fitted one with $h = 0.001$. We suppose that the error of the reference body-fitted solution is negligible compared to the one committed by the SB solution, which uses coarser grids.

It may be interesting to look at the pointwise error between the SB solution and the reference body-fitted solution. Note that, to compare the solutions an interpolation process has been performed between the nodes of the SBM mesh and the body-fitted mesh since the meshes are completely different. Figure 60 shows the pointwise module error for mesh4 and mesh5.

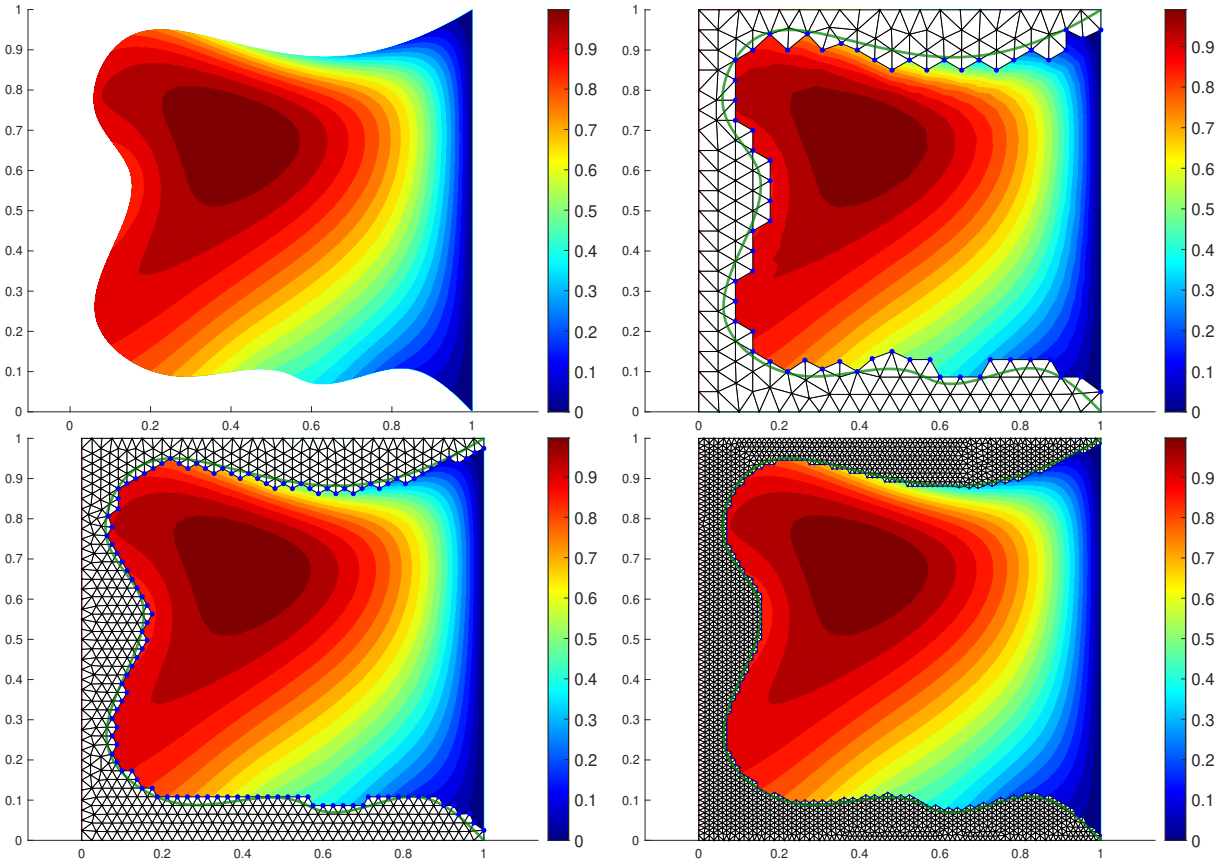


Figure 59: FEM solutions of the convection-diffusion stationary problem with $\boldsymbol{v} = [5, 10]$. In the upper left panel is reported the body-fitted solution with $h = 0.02$; in the upper right, the SB solution with mesh2 ($h = 0.05$); in the lower panels, we have SB solutions with mesh3 ($h = 0.025$) and mesh4 ($h = 0.0125$).

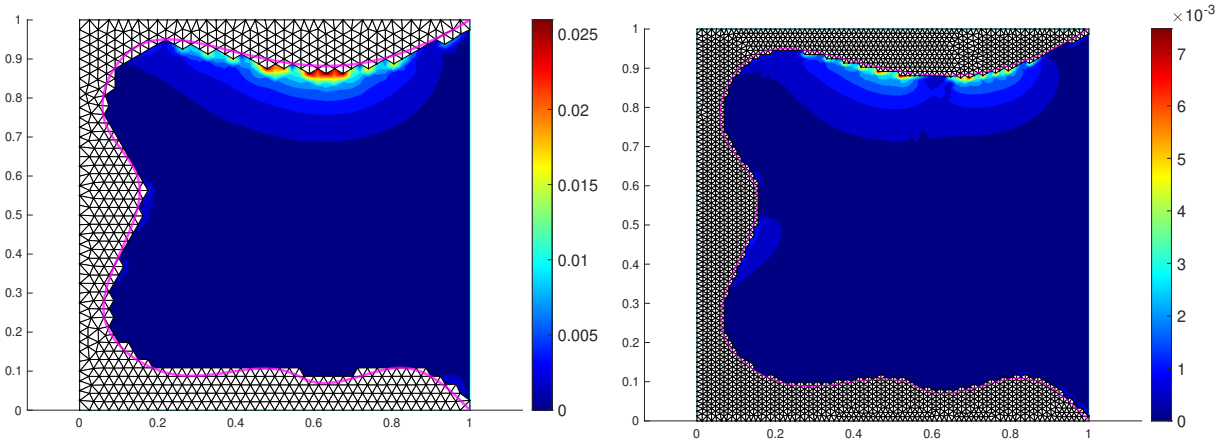


Figure 60: The pointwise module of the error of the SB solution with respect to the reference body-fitted solution is reported considering mesh3 ($h = 0.025$) and mesh4 ($h = 0.0125$).

The convergence study in Figure 61 shows, once again, a perfect second-order convergence in the L^2-norm. The presence of the convective term and the external curve embedded boundary is perfectly manageable by the SB Kratos algorithm.

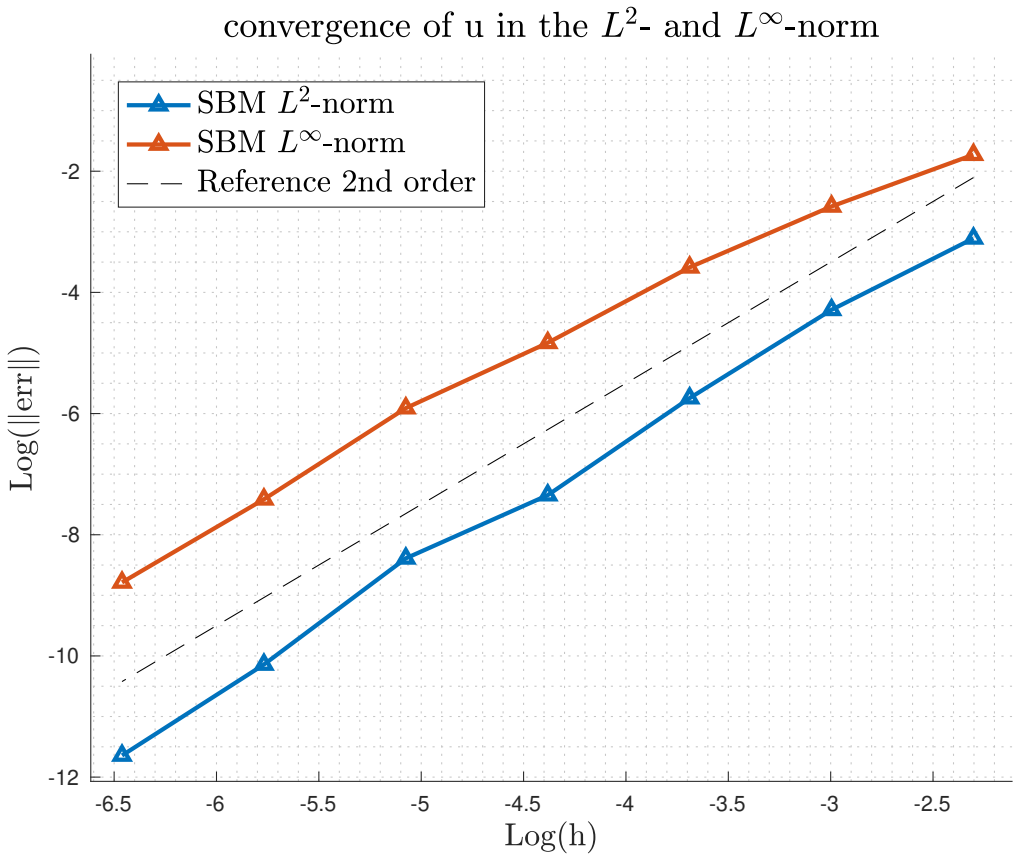


Figure 61: Convergence study of the error in the L^2- and L^∞-norm of the SB solution of the stationary convection-diffusion problem with conditions 191 and using the set of meshes in Table 12.

	$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$
mesh1	0.04468	0.1777
mesh2	0.01373	0.07569
mesh3	0.003197	0.02769
mesh4	0.0006449	0.007967
mesh5	0.0002272	0.002711
mesh6	$3.940e-05$	0.0006022
mesh7	$8.779e-06$	0.0001527

Table 17

SLOPE	L^2-norm	L^∞-norm
$1 \to 2$	1.7018	1.2314
$2 \to 3$	2.1027	1.4504
$3 \to 4$	2.3100	1.7977
$4 \to 5$	1.5049	1.5549
$5 \to 6$	2.5278	2.1707
$6 \to 7$	2.1661	1.9792
average	2.0522	1.6974

Table 18

6.7 Transient convection-diffusion problem

The chapter's objective is to check the convergence of a transient convection-diffusion problem with an embedded boundary, i.e. we can consider an immersed object as done so far or a portion of the external boundary as embedded.

Let us rewrite the general convection-diffusion system of equations:

$$\begin{aligned} &\frac{\partial u}{\partial t} + \boldsymbol{v} \cdot \nabla u - \Delta u = f(\boldsymbol{x}, t) && \boldsymbol{x} \in \Omega,\ t > 0 \\ &u(\boldsymbol{x}, t) = \overline{u}(\boldsymbol{x}, t) && \boldsymbol{x} \in \Gamma,\ t > 0 \\ &u(\boldsymbol{x}, 0) = \tilde{u}(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \end{aligned} \tag{192}$$

Where to check the results, we choose to use a manufactured solution $u(\boldsymbol{x}, t)$. We take a vector field $\boldsymbol{v}$, and we compute the fictitious forcing function $f(\boldsymbol{x}, t)$ such that the partial differential Equation 192 is satisfied.

For instance, let us consider the following function that will be the analytical solution to the problem:

$$u(x, y, t) = t^{\beta} \sin(\alpha x) \cos(\alpha y) \tag{193}$$

Where α and β are scalar parameters we can fix. Therefore, one can compute the following terms:

$$\begin{aligned} &\frac{\partial u}{\partial t} = \beta t^{\beta - 1} \sin(\alpha x) \cos(\alpha y) \\ &\nabla u = \begin{bmatrix} \alpha t^{\beta} \cos(\alpha x) \cos(\alpha y) \\ -\alpha t^{\beta} \sin(\alpha x) \sin(\alpha y) \end{bmatrix} \\ &\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\alpha^2 t^{\beta} \sin(\alpha x) \cos(\alpha y) \end{aligned} \tag{194}$$

Thus, taking an arbitrary uniform convective term $\boldsymbol{v} = [a, b]^{\mathrm{T}}$ we can write the fictitious forcing function of Equation 192 as function of the parameters α, β, a and b:

$$\begin{aligned} f(x, y, t) =& \frac{\partial u}{\partial t} + \boldsymbol{v} \cdot \nabla u - \Delta u \\ =& \beta t^{\beta - 1} \sin(\alpha x) \cos(\alpha y) + a\alpha t^{\beta} \cos(\alpha x) \cos(\alpha y) - b\alpha t^{\beta} \sin(\alpha x) \sin(\alpha y) \\ &+ 2\alpha^2 t^{\beta} \sin(\alpha x) \cos(\alpha y) \end{aligned} \tag{195}$$

As the computational domain Ω, we will consider a square domain defined by the points $[0, 0] \times [1, 1]$. As before, the external boundary of Ω will be treated using conforming Dirichlet boundary conditions. An immersed shape is considered in the centre of Ω. In particular, we will consider a circle since we have already proven that sharp corners don't affect global error convergence.
Naturally, we have to provide adequate boundary conditions. To have $u(x, y, t)$ of Equation 193 as the analytical solution, on both the standard and SB boundaries, we will consider the exact solution as the Dirichlet values to be imposed. Instead, the initial conditions are simply the following:

$$u(\boldsymbol{x}, 0) = u(\boldsymbol{x}, t = 0) = 0 \tag{196}$$

Since we will always consider $\beta \geq 1$.

An essential fact that needs to be taken into account is that now the problem is time-dependent, and in particular, the boundary conditions are time-dependent. Therefore the MPCs related to the SBM Dirichlet boundary conditions change the so-called "constant vector" at each iteration.

Recall Equation 160 that represents the SBM Dirichlet boundary condition of the surrogate boundary node $\boldsymbol{x_s}$. The transient case can be generalized in the following:

$$u(\tilde{\boldsymbol{x}}, t) = \bar{u}(\boldsymbol{x_t}, t) - \boldsymbol{G u_n} \cdot \boldsymbol{d} \tag{197}$$

Where we can notice that:

1. since the embedded Dirichlet boundary conditions depend on time, naturally, the value at the projection $\bar{u}(\boldsymbol{x_t}, t)$ depends on time;
2. if the immersed object does not move in time, the surrogate boundary always remains the same during the simulation;
3. if the surrogate boundary and the geometry are the same at each iteration, the available neighbour degrees of freedom vector $\boldsymbol{u_n}$ of node $\tilde{\boldsymbol{x}}$ and the coefficients in matrix $\boldsymbol{G}$ remain the same for each iteration.

Therefore, what we must do is only modify the constant term $u(\boldsymbol{x_t}, t)$ of the MPCs at each solution step. Notice that in the case of moving immersed object, we would have to re-compute the surrogate boundary nodes $\tilde{\boldsymbol{x}}$'s, the projections $\boldsymbol{x_t}$'s and the available neighbour degrees of freedom at each time-step.

Let us use the following parameters for fixing the analytical solution and the corresponding forcing term: $\alpha = 8$, $\beta = 2$, and $\boldsymbol{v} = [1, 1]$. Then, the analytical solution becomes:

$$u(x, y, t) = t^2 \sin(8x) \cos(8y) \tag{198}$$

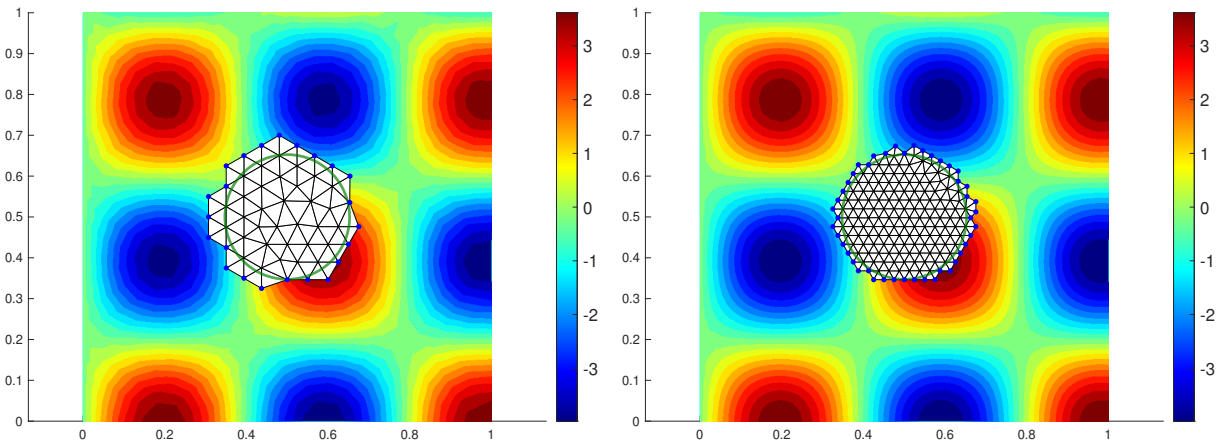


Figure 62: In the images, the SB solution of the Poisson transient problem using mesh2 and mesh3 is represented. The solutions are taken at $t^* = 2.0$, using a Crank-Nicolson time integration scheme with $\Delta t = 0.1$.

As in Chapter 6.5, where we have dealt with a transient Poisson problem, we first have to choose a time integration scheme, then fix a time t^* where we want to compare the error between SB and analytical solution. But before it, we have to perform an error analysis related to the time integration. We want to use a time-step that allows us to neglect the error due to the time integration with respect to the error related to the spatial integration.
Consider the solution at time $t^* = 2.0$ and use a Crank-Nicolson time integration scheme. We checked that the time-step $\Delta t = 0.1$ is sufficient for considering the error related to time integration as negligible.

In Figure 62, the SB solution is reported. The L^2- and L^∞-norm can be used as a measure for the convergence study of the transient convection-diffusion problem we are considering. Figure 63 shows the results, and the corresponding data has been catalogued in Table 63 and Table 19.

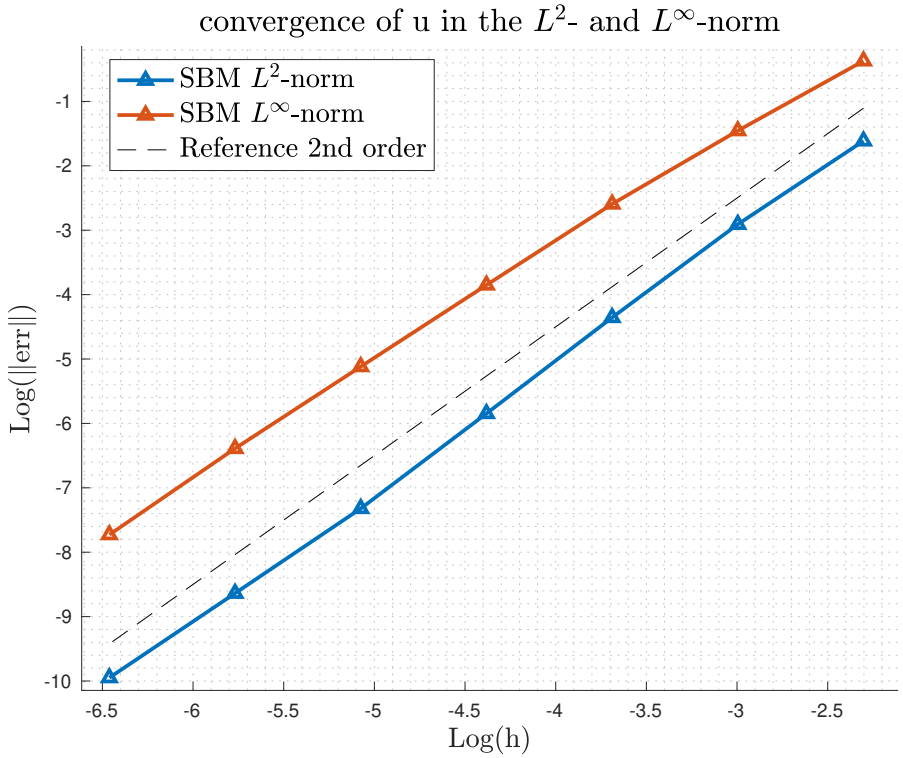


Figure 63: Convergence study of the error in the L^2- and L^∞-norm of the SB solution using the set of meshes in Table 12, at $t^* = 2.0$, using a Crank-Nicolson time integration scheme with $\Delta t = 0.1$.

	$\|\|\text{err}\|\|_{L^2(\tilde{\Omega})}$	$\|\|\text{err}\|\|_{L^\infty(\tilde{\Omega})}$
mesh1	0.1992	0.6900
mesh2	0.05455	0.2334
mesh3	0.01283	0.07473
mesh4	0.002889	0.02125
mesh5	0.0006613	0.005987
mesh6	0.0001772	0.001679
mesh7	$4.779e-05$	0.0004404

Table 19

SLOPE	L^2-norm	L^∞-norm
$1 \to 2$	1.8686	1.5636
$2 \to 3$	2.087	1.6433
$3 \to 4$	2.1509	1.8137
$4 \to 5$	2.1274	1.8280
$5 \to 6$	1.8992	1.8337
$6 \to 7$	1.8911	1.9311
average	2.0042	1.7689

Table 20

6.8 Moving objects

In this last part of the results and validations section, we would like to show the potentialities of the SB method in some particular applications. As said in the introduction, Chapter 1, the SB method, and in general unfitted methods, have worse accuracy and a more complicated implementation in the imposition of the boundary conditions at the embedded boundaries. It is also true that from the theory chapters (Chapter 3 and Chapter 4) and the results shown so far, the properties related to the convergence in the L^2- and L^∞-norm are optimal, in the sense that the SB method does not affect the convergence estimate of the FEM solution.
But in none of the previous examples we faced problems where unfitted methods were necessary. All the convection-diffusion cases treated didn't have problems such as large boundary movements or large deformations. We have only seen fixed immersed bodies in Chapter 6.1, 6.2, 6.3, 6.4 and 6.7. Moreover, in Chapter 6.5 and Chapter 6.6, we have also proven that the SB method works perfectly in the case of embedded external boundaries. But in all these cases a body-fitted approach was, with no doubt, the best and easiest strategy we could use; actually, we would have obtained smaller errors, with the same convergence properties, with higher accuracy around the embedded interface and without any complex algorithm.

For this reason, it is necessary to show some applications where an SB method really simplifies the problem and where the body-fitted approach is difficultly applicable. As now should be clear, these cases are when large displacements or boundary movements need to be described. So far, what we have called the skin boundary was fixed. What happens to the SB algorithm if the true boundary moves in time? Naturally, some changes are fundamental because, in all the previous cases, the shifted boundary and the shifted boundary nodes were computed in the `Initialization` class of the SB solver.
In Chapter 5.7.3, where the moving objects solver has been described, the algorithm needed has been explained. This solver basically contains all the steps of the SB method in the `Initialize SolutionStep` class of the solver. In fact, at each time step, the true boundary Γ can move and deform, and the surrogate boundary $\tilde{\Gamma}$ may change. If the skin boundary changes, everything might change, starting from the level set function up to the MPCs. The projections of the surrogate nodes need to be computed again, such as the coefficients and the neighbour degrees of freedom for computing the gradient term. Finally, the old MPCs must be erased, while the new ones have to be set. For this reason, all the previous passages need to be optimized as possible since they are repeated as many times as the number of time-step.

The body-fitted approach in the case of large boundary movements is highly not recommended due to its conforming nature. The boundary nodes are stuck to follow the boundary movements. Therefore, the elements surrounding the boundary can easily be stretched or even become inverted. In this case, a re-meshing algorithm is required if the body-fitted approach is the only alternative. Unfortunately, it is computationally very expensive and requires particular attention in the case of complex geometries where bad mesh quality is a risk.

Therefore consider a quasi-static evolution of a moving immersed object in a convection-diffusion problem. The immersed object will be treated as an embedded boundary, whereas the external boundary conditions can be applied using a standard conforming method. Since a fluid-structure interaction problem is not yet implemented, an external motion is imposed on the embedded shape. Imposing conforming Dirichlet boundary conditions at the external boundaries and modified SB Dirichlet boundary conditions at the surrogate boundary, we can compute the distribution of the unknown scalar variable u in the surrogate domain $\tilde{\Omega}$.
The objective of this subsection is only to show the capability and future possible development of

the algorithm. For that reason, no complicated processes have been designed to manage the time evolution of u when an element changes from being active to de-active. In fact, when the true boundary moves in time, some elements that at the previous iteration were active may now be cut or fully inside the immersed moving object. In this case, the element can be made de-active, and no additional problems arise; but imagine the opposite case, which happens when an immersed element turns out to be non-immersed in the current iteration. In this case, there would be no available values of u at the previous time step. An estimate based on the neighbours can be implemented, but it is out of the scope of this thesis. Therefore, a quasi-static evolution of a moving object is considered. Such that we can use the stationary solver that does not request any value of u at the previous time-step since the solution is supposed to be stationary. Physically, we can imagine the object moving infinitely slowly, such that the system is always passing through equilibrium states.
Therefore, the results we are going to propose soon are analogous to the ones obtained in Chapter 6.6 that we got solving the stationary convection-diffusion problem. In fact, we are practically just solving the stationary solution of many configurations where at each time step the geometry might change. The power of the SB method is that we can get easily and quickly the stationary solutions of many configurations always using the same background mesh. The same results applying a body-fitted approach would require a re-meshing technique and much more computational time and effort.

The first example is represented by an immersed object rotating around a fixed point. As said before, it makes sense to treat the boundary of the immersed object as embedded with the SB method, and the external boundary using a body-fitted approach. Consider a rectangle rotating around a fixed point in a square domain $[0, 0] \times [1, 1]$. Suppose we know that the unknown scalar function, for instance a temperature, is always zero at the true interface between the object and the domain.

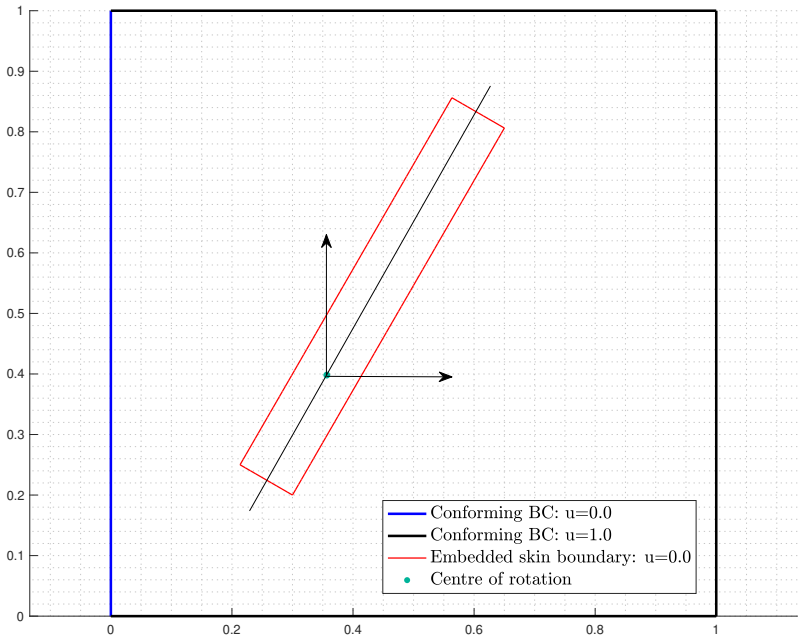


Figure 64: Geometry of the initial condition of the problem and the boundary conditions.

Moreover, suppose that from the boundary on the right, a convection heat flux with a solenoidal vector field $\boldsymbol{v} = [10, 0]$ is coming, where the temperature at $x = 0.0$ is fixed with Dirichlet conforming boundary conditions to be $u = 1.0$. Let's complete the remaining external boundary

conditions fixing $u = 0.0$. The geometry of the initial configuration of the problem is summarized in Figure 64.

Consider that the rectangular embedded shape rotates clockwise from the configuration of Figure 64 up to be horizontal. One may be interested to study the distribution of the temperature scalar field during the equilibrium states between these two configurations. In the SB algorithm implemented in Kratos, one may import the geometry of the initial skin boundary and then, at each time step, rotate each point of a certain angle with respect to the centre of rotation. The value of the time step, in this case, is useless since each solution step is the stationary one. In fact, the stationary solver is used.

The evolution of the rotation of the immersed valve can be observed in Figure 65, where only the algorithm related to the research of the surrogate boundary and active/deactivate elements have been applied.

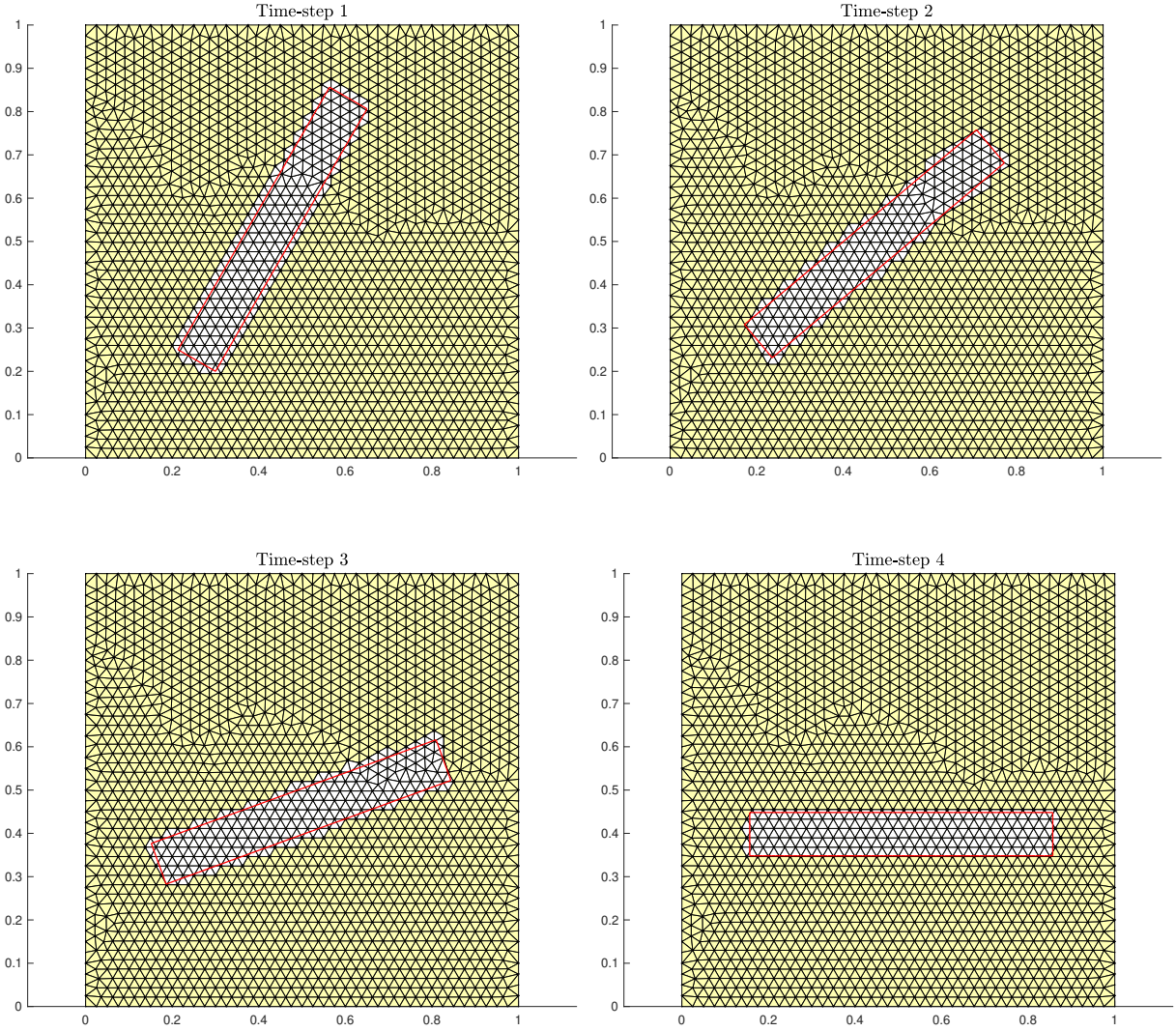


Figure 65: Rotation of the valve from the initial condition (60°) up to the horizontal position (0°) using four steps: 60°, 40°, 20° and 0°. The surrogate nodes are marked with black dots, and the white elements are deactivated.

In Figure 66 the solutions of the six time-steps have been reported. The rotation of 60° has

been subdivided into six uniform steps and, therefore, six equilibrium states have been obtained. The mesh that has been used is mesh3 in Table 12, and the characteristic length of the mesh is $h = 0.025$. Naturally, more refine grids can be used to reduce the areas of the portions of cut elements, which are not considered in the simulation.

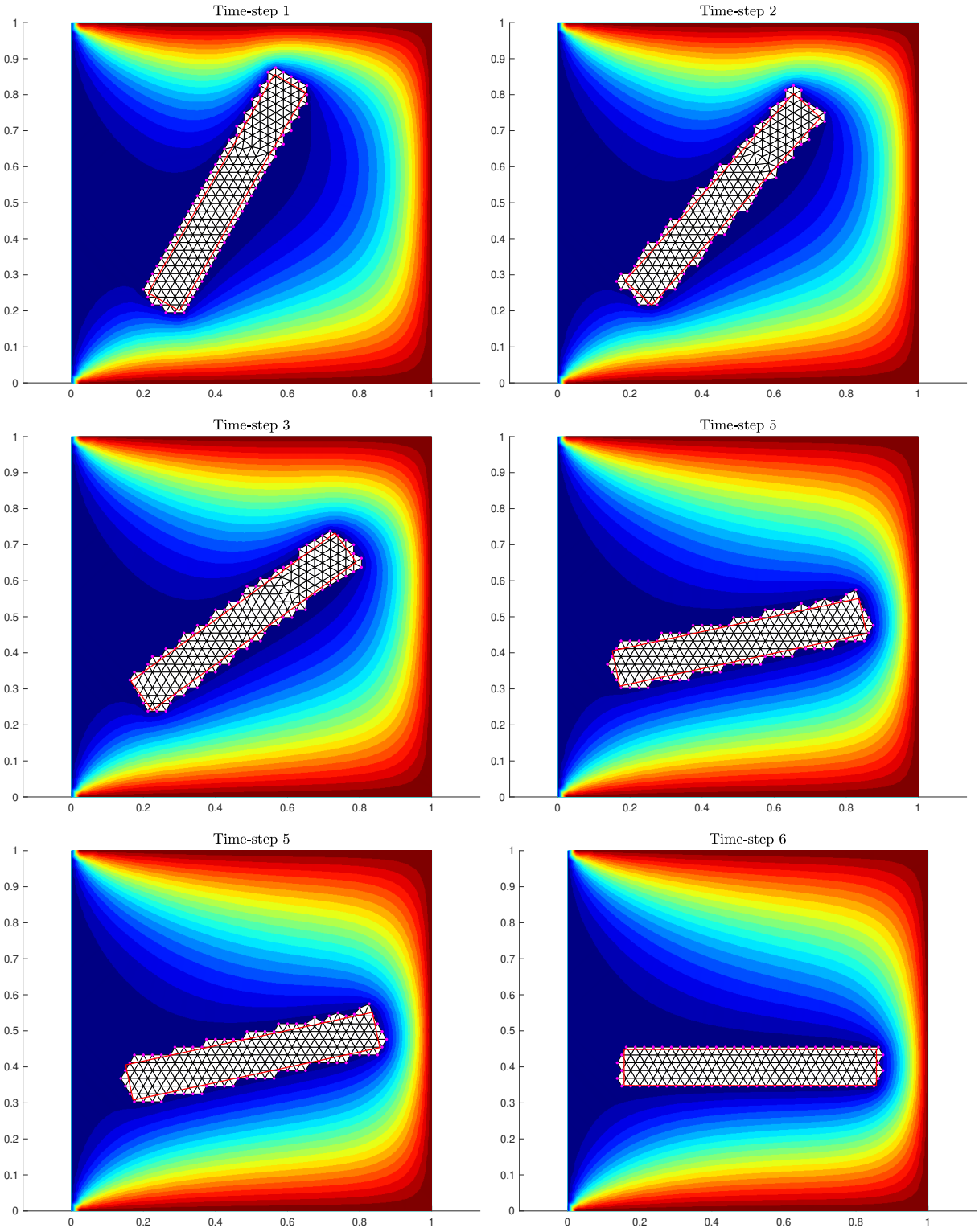


Figure 66: Contours of the scalar unknown variable u while the valve is rotating around the centre of rotation in six steps.

To show that not only large movements or rotations can be handled, without any computationally expensive techniques, by the SB method, we also report an example where a large deformation is present. As in the previous example, the boundary movements are imposed a priori since the fluid-structure interaction application with the SB method is not yet ready.
Consider a circle stretched and compressed such that it deforms in ellipses with the same area. The initial and final configurations of the object are represented in Figure 67. The problem is still a convection-diffusion stationary problem. Stationary because the solver used catches the stationary solution depending on the boundary conditions, which change as the immersed object deforms. The convection term is set to be $\boldsymbol{v} = [-20, -20]$, and the boundary conditions are similar to the ones used in the first example. Standard conforming boundary conditions are applied at the external boundaries of the usual square $[0, 0] \times [1, 1]$. In particular, $u = 0.0$ at the lower and left sides and $u = 1.0$ on the remaining sides, exactly as is shown in Figure 67.

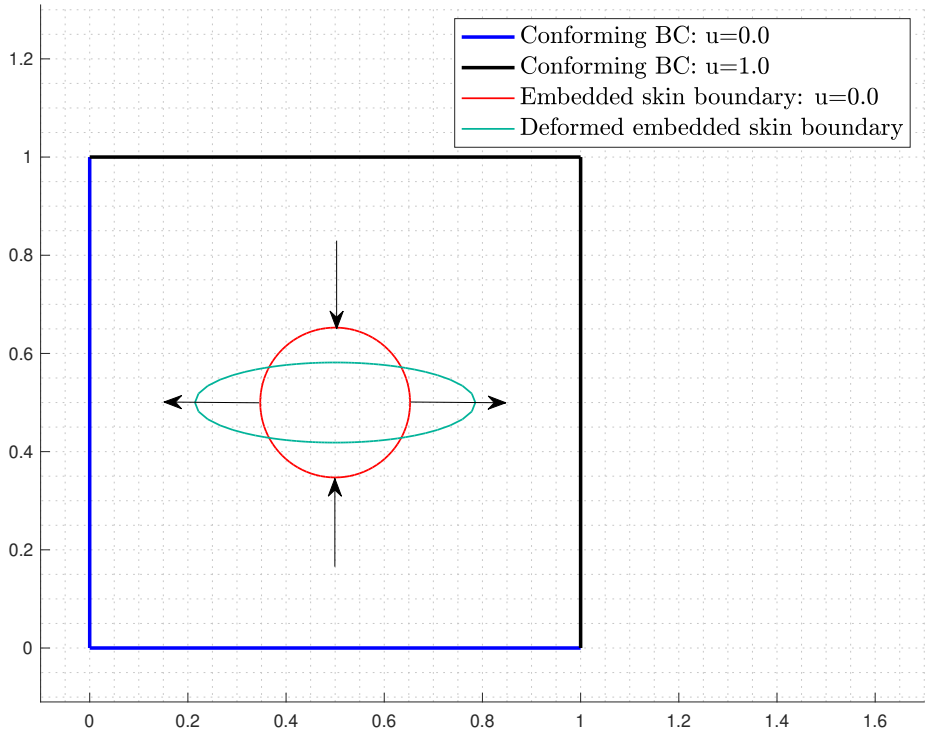


Figure 67: The geometry and boundary conditions of the second example in the moving object chapter are summarised. The circle in red (initial configuration) is deforming into ellipses of the same area.

Also in this example is important to specify that the deformations of the circle are imposed and have been computed analytically such that the object conserves its area. In particular, given the radius of the circle r, we can vary the constant K obtaining infinite ellipses with the same area, where the semi-major and semi-minor axes are defined by the following:

$$\begin{aligned} a &= \sqrt{K} r \\ b &= \frac{r}{\sqrt{K}} \end{aligned} \tag{199}$$

In our case, the value of K has been varied between 1 and 4 using 12-time steps. The results of the SB algorithm in terms of true boundary, shifted boundary and active/deactivate elements of some steps are reported in Figure 68. The figure also shows a comparison of the same results using mesh2. Instead, in Figure 69, the contours of the scalar solutions u are shown using mesh3 of Table 12.

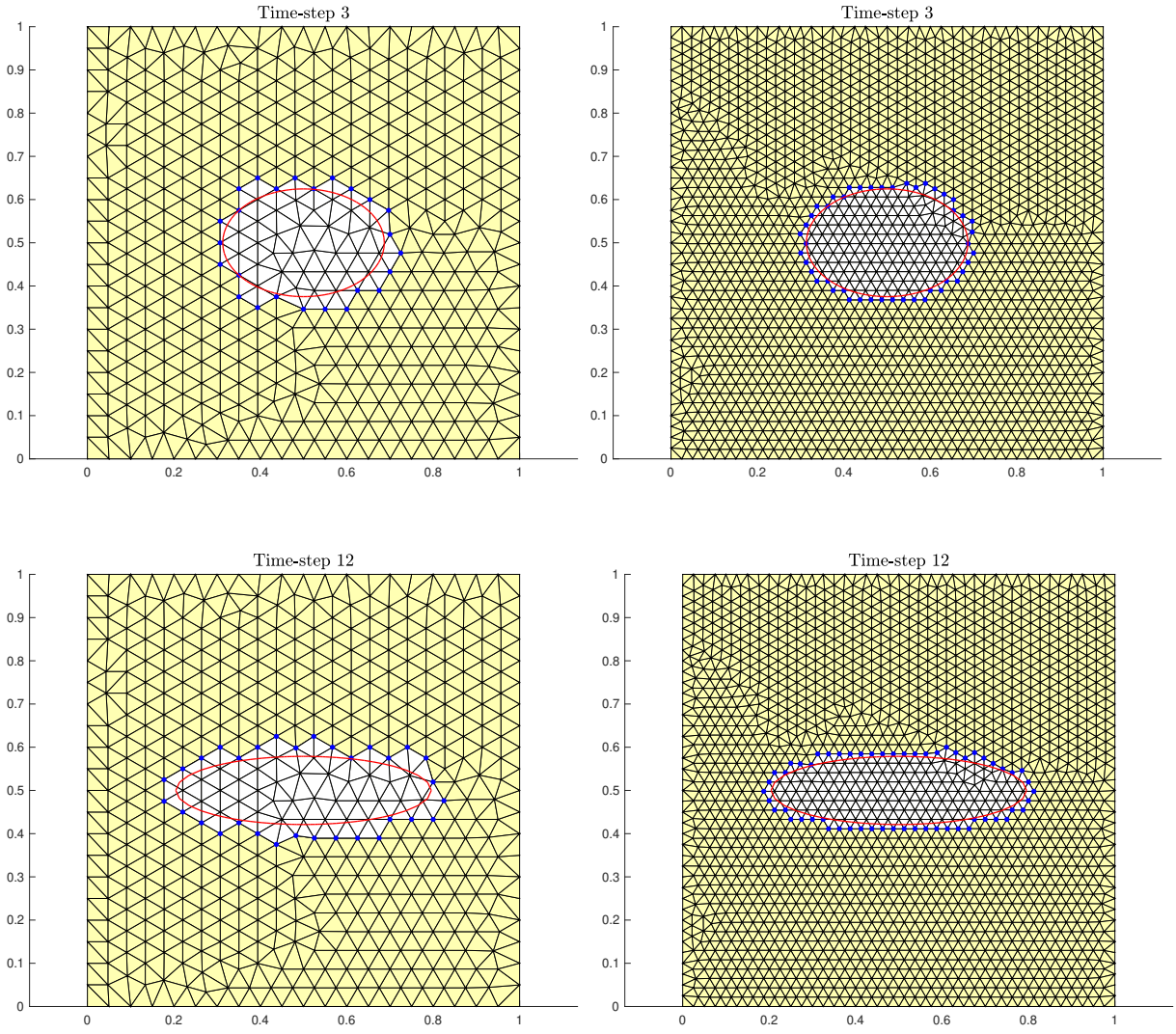


Figure 68: The figure shows the comparison between mesh2 and mesh3 of two configurations of the second example of the moving objects chapter. The blue dots represent the surrogate nodes and the white elements are de-activate, whereas the yellow ones are active.

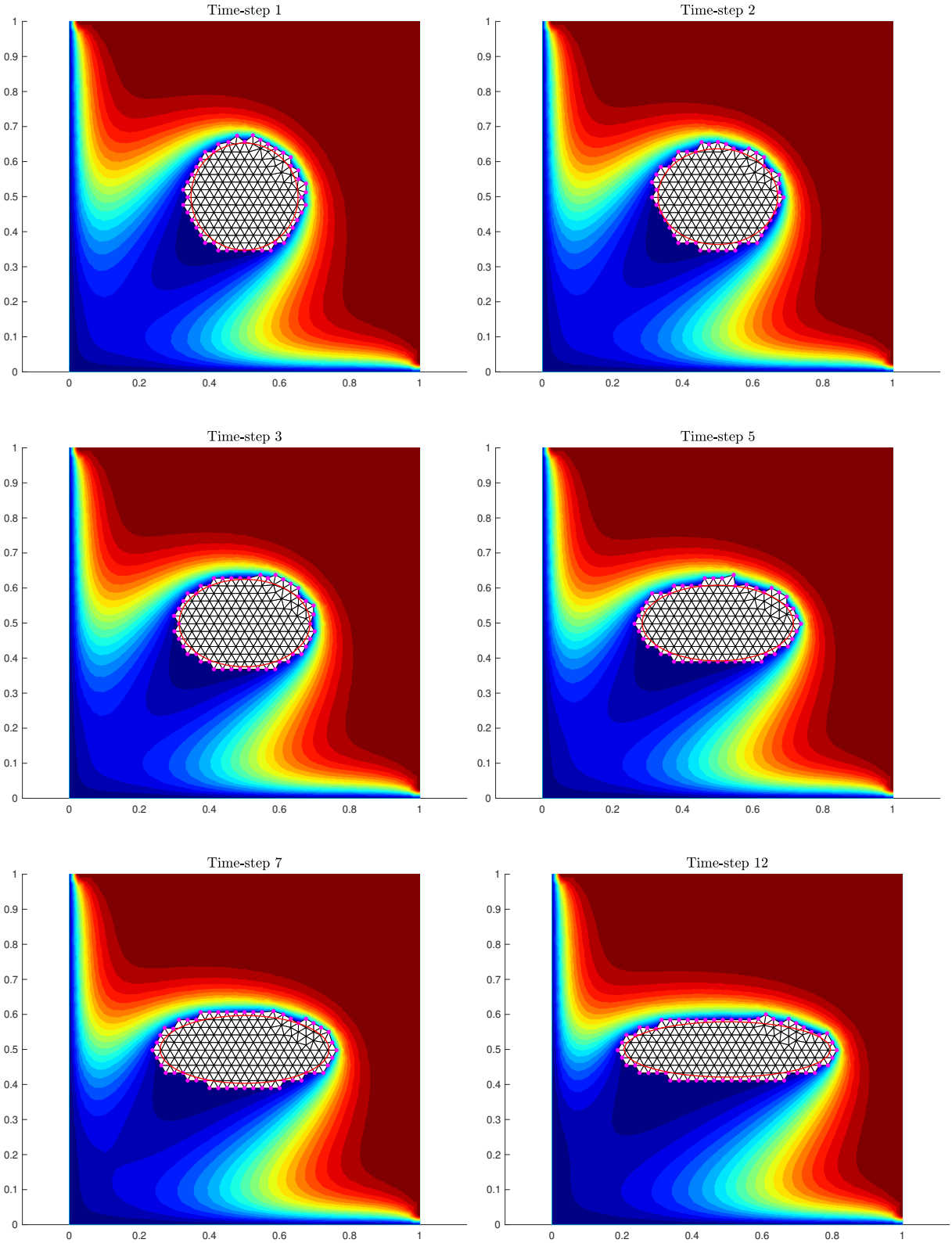


Figure 69: Contours of the scalar unknown variable u while the circle is deforming in twelve steps between the initial and final configurations in Figure 67.

In the previous two examples, it has been shown that immersed objects can deform and moves while the SB solver can capture its boundary and correctly impose the boundary conditions at the locations of the surrogate nodes using the MPCs. Another potentiality of the SB method, and in general of the unfitted methods, is to simulate easily complex external boundaries imposing the Dirichlet boundary conditions. It is the reason why we report one last example of moving boundaries where an external boundary is moving and deforming at the same time.

The boundary conditions are represented in Figure 70. In particular, on the left side is imposed $u = 1.0$, and on the rest of the background external boundaries are set $u = 0.0$. In this case, no immersed objects are immersed in the domain, but the right side is a moving and deforming external boundary. In Figure 70, the initial geometry is highlighted in red, and the evolution of the boundary itself is represented in blue. In particular, we set in total six iterations where the nodes of the skin boundary follow a wave equation.

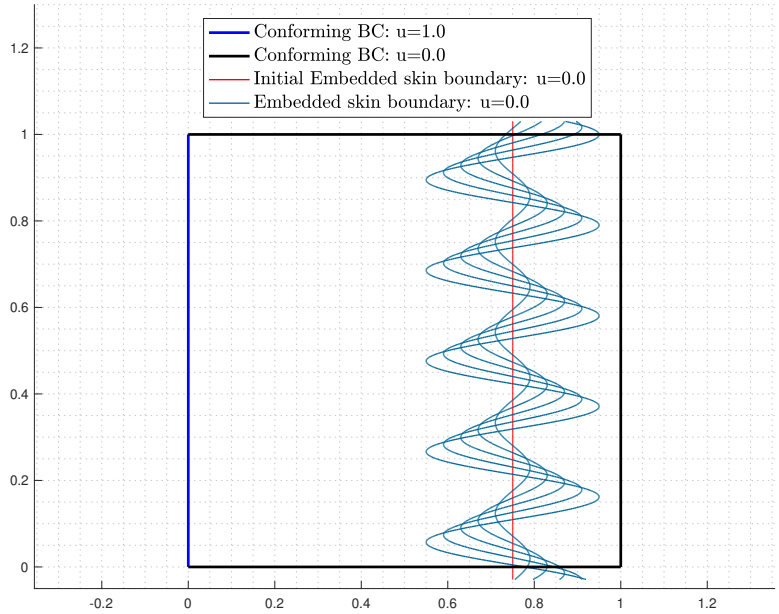


Figure 70: Geometry and boundary conditions of the third example in the moving object chapter. The right external boundary in red (initial configuration) moves and deforms following the shape of a sinusoidal wave.

The wave equation has an amplitude A, which depends on the time-step. Moreover, also the argument of the sinus is dependent on both the ordered and the time-step. The motion of the external embedded boundary is defined by the following equations:

$$\begin{aligned} x &= x_0 + A\sin(\omega y + \text{iter}/2) \\ A &= 0.2 * (\text{iter} - 1)/5 \\ \omega &= 30 \end{aligned} \tag{200}$$

Where $x_0 = 0.75$ is the abscissa of the initial position of the embedded external boundary. The value of iter represents the current time-step. Six iterations will be performed in total. The results of the deformation and translation of the embedded boundary during the six time steps are represented in light blue in Figure 70. A convection term is present and is equal to $\boldsymbol{v} = [40, 5]$; the forcing term of the convection-diffusion problem instead is equal to -1.0.
The results of the convection-diffusion simulation are shown in Figure 71. As in the previous two examples, the stationary solver has been used for these results, therefore we may think of an infinitely slow deformation of the embedded boundary.

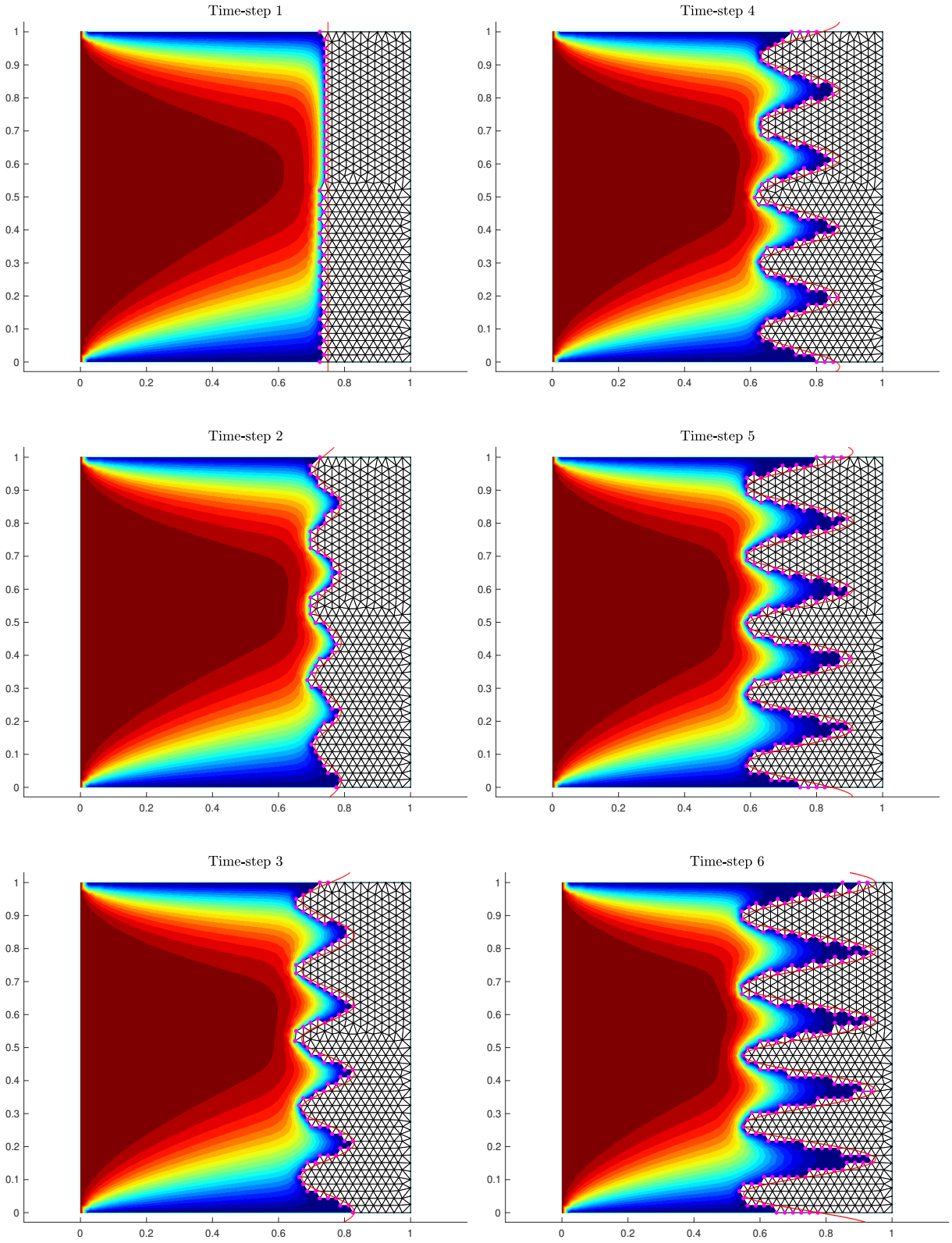


Figure 71: Contours of the scalar unknown variable u while the right external boundary is deforming and moving in six steps between the initial and intermediate configurations of Figure 70.

7 Conclusions

The main objectives of this project were accomplished. First, we analysed in depth the theory behind the SB method for the Poisson equation and for the convection-diffusion problem in Chapter 3 and Chapter 4, respectively. They have shown in both cases that the SB method leads to a well-posed problem. Moreover, the convergence estimate in the L^2-norm is at least proportional to $h^{3/2}$ for the Poisson problem and the H^1-norm of the error converges at least first-order in the convection-diffusion case.

In Chapter 5, we discussed all the implementation details regarding the SB method in the open-source software Kratos. In particular, after describing Kratos Multiphysics' structure of the convection-diffusion application, were developed from scratch three derived solver classes: the stationary, the transient, and the moving objects solver.
Between the crucial steps, we may mention the use of a level set function (Chapter 5.3.2) that allows us to determine if a node is inside the surrogate domain $\tilde{\Omega}$ or not. Thanks to it, in Chapter 5.3.3, it has been explained how to individuate the surrogate boundary nodes that compose the surrogate boundary $\tilde{\Gamma}$. Once the surrogate boundary nodes are identified, the crucial point of the SB method comes with the imposition of modified Dirichlet boundary conditions at the surrogate node locations. It allows the method to preserve the second-order convergence in the L^2-norm. In Chapter 5.4.1, we motivated the Taylor expansion between the surrogate node and its closest point belongings to the true boundary. In Chapter 5.4.2, we explained the algorithm that allows us to find the closest-point projection of each surrogate node onto the skin boundary.

Furthermore, in the implementation section, the problems and the limitations on how to compute the gradient term of the first-order Taylor expansion have been examined. In Chapter 5.5, we introduced the novelty of the SB method that really makes it different from [26] and [27]: the use of multi-point constraints (MPCs) for the imposition of the modified Dirichlet boundary conditions. We reported in Chapter 5.5.1 and 5.5.2 an introduction to MPCs and how they are implemented and can be used in Kratos.
We studied in Chapter 5.5.3 the adaptation of the MPCs to the SB method, together with the computation of the coefficients of the MPC equations. The introduction of the MPCs for imposing the modified Dirichlet boundary conditions on the shifted boundary represents a novel and alternative way of applying the SB method. What makes interesting this new approach is that allows for avoiding an additional non-linear loop for computing the gradient term. On the other hand, the use of MPCs adds additional complexity especially due to additional problems of finding available neighbour elements for computing the gradient coefficients (Chapter 5.5.4).
The additional flux term through the surrogate boundary has been managed in Chapter 5.6. It allows us to use appropriately the MPCs keeping the second-order convergence in the L^2-norm. At this point, the implementation part of the SB method in the Kratos Multiphysics framework ended, and the three SB solvers that use the MPCs for the imposition of the modified Dirichlet boundary conditions were available for convection-diffusion problems.

Next, the result section (Chapter 6), where the aim was to validate the novel SB method algorithm developed in Kratos. Consequently, the easiest case has been taken, i.e a stationary Poisson problem with a linear exact solution, and embedding an immersed object, it was possible to get the exact solution up to the machine's precision. That was what one expects from a second-order accurate FEM method.
Then, in Chapter 6.2 and 6.3, a paraboloid and a complex function have been used as exact solutions where, once again, a circular object was considered embedded inside the computational domain. In both cases, a convergence study was performed, and a perfect second-order conver-

gence in the L^2-norm has been achieved. As expected, the errors were slightly larger with respect to the body-fitted approach of the same problem, but this is a price that can be paid for the flexibility and robustness of the SB method.
In Chapter 6.4, for the first time, were considered complicated embedded shapes. It has been shown that, if the exact solution has a sort of discontinuity in the location of the corners, the SB method behaves worse than the body-fitted counterpart. In fact, the body-fitted approach, in general, ensures that one mesh node is located at the discontinuity. Instead, in the SB method, it is not guaranteed. This fact leads to a convergence error equal to $h^{3/2}$ in the L^2-norm. The phenomena do not appear in regular cases, therefore, we concluded that it is an intrinsic problem of some particular applications, and it is not related to complex embedded shapes. We have proved it at the end of Chapter 6.4.
In Chapter 6.5, the Poisson problem becomes time-dependent. We performed a spatial convergence study at a fixed time. We used a time-step such that the error related to the time integration could be considered completely negligible with respect to the spatial one. Moreover, instead of an immersed object, a portion of the external boundary has been considered embedded. The results confirmed that the algorithm works perfectly also in the transient Poisson problem obtaining a second-order convergence.
In Chapter 6.6 and 6.7, the convection term has been introduced in the differential equation in the stationary and transient case, respectively. In the stationary convection-diffusion problem, we checked the convergence estimate with an immersed embedded object and an embedded external boundary, using both an existing exact solution and the body-fitted solution as a reference. Instead, in the transient case, an highly non-linear exact solution has been taken into account, comparing the exact solution and the SB solution at a fixed time considering a time-step such that the time integration error didn't affect the spatial convergence study. In all these cases, a perfect second-order convergence has been proved, leading to the result that the SB algorithm implemented in Kratos, and using the introduced novel, i.e. the MPCs for the imposition of the modified Dirichlet boundary conditions, is working and robust in convection-diffusion applications.
Finally, in Chapter 6.8, the real potentialities of the method have been shown. The real advantages of unfitted methods, with respect to the classical body-fitted approach, are evident when one considers large moving boundaries. Three examples have been reported for understanding how is easy to deal with moving objects or moving external boundaries in an SB manner. In these examples, we imposed large movements to embedded boundaries. Using the same background mesh the algorithm was able to apply all the steps of the SB method at each iteration capturing the position and the effect of the immersed boundaries.

Therefore we successfully made a first step toward the application of the SB method in the Kratos Multiphysics framework for computational fluid dynamics problems. The simplest version of the Navier-Stokes, i.e. the convection-diffusion problem, can afford different types of embedded boundaries, such as immersed objects, also with complex shapes or external straight or curvilinear boundaries without any loss in terms of convergence estimate.
It is necessary to underline that the SB method has intrinsically a loss of precision for two main reasons. The first is that the shifting of the boundary leads to cutting elements that are completely taken out of the simulation de-activating them. It leads to an empty zone where we obtain no results. Even in the condition $h \to 0$, i.e. taking a more refined mesh, it is true that the shifted boundary tends to overlap the true one, but always lack of precision needs to be expected when we compare the method with a simulation that uses a body-fitted approach.
Secondly, when the SB method applies the modified Dirichlet boundary conditions, it is neglecting some parts of the Taylor expansion. In fact, we have shown the truncation of the Taylor expansion in Chapter 5.4.1. It implies an additional second-order error that is not introduced in

a standard body-fitted approach. It is the reason why in the comparisons between the SB solution and body-fitted one, the latter always had a lower global error both in the L^2- and L^∞-norm.

The next step for having an SB algorithm for CFD applications is firstly moving to Stokes fluid, where the main work that has to be done is related to having three scalar unknowns in the problem (x-velocity, y-velocity, and pressure) and not only one. It does not change the implementation of the SB algorithm seen in this project, since the scalar unknowns can be treated one by one. We can apply, on each surrogate node, one MPC for each scalar degree of freedom that has a Dirichlet condition at the embedded boundary. For instance, the most common application is an object immersed in a fluid flow; the Dirichlet boundary conditions are usually a no-slip condition between the immersed object and the fluid. Thus, in this case, two MPCs should be imposed on each surrogate node. One for the x-velocity and the second regarding the y-velocity degree of freedom. Finally, one may consider the non-linear convective term leading to the incompressible Navier-Stokes equation. In the Kratos Multiphysics framework, all the CFD strategy implementations have already been validated and, thanks to its flexibility, reusability, and object-oriented structure, the modifications for having an SB solver for CFD are similar to the ones done in this context in the implementation chapter. Therefore, future developments are ready to be done in Kratos for bringing an innovative, robust, and flexible method for the imposition of Dirichlet embedded conditions in CFD problems.

8 Appendix

8.1 Appendix 1

```
    auto& r_elem_neigh_vect = r_bd_node.GetValue(NEIGHBOUR_ELEMENTS);
    // Set an auxilary map to calculate the current node nodal gradient
       contributions
    std::map<std::size_t, Vector> neigh_dn_dx_map;
    // Calculate the nodal gradient coefficients
    double w_total = 0.0;
    for (std::size_t i_neigh = 0; i_neigh < r_elem_neigh_vect.size(); ++
       i_neigh) {
        auto p_elem_neigh = r_elem_neigh_vect(i_neigh).get();
        // Calculate the current element weight
        const auto& r_geom = p_elem_neigh->GetGeometry();
        const std::size_t n_nodes = r_geom.PointsNumber();
        const double w = r_geom.DomainSize() / n_nodes;
        w_total += w;
        // Calculate the current element nodal gradient
        ShapeFunctionsGradientsType grad_N;
        r_geom.ShapeFunctionsIntegrationPointsGradients(grad_N,
           GeometryData::IntegrationMethod::GI_GAUSS_1);
        // Save the current element nodal weights
        for (std::size_t i_node = 0; i_node < n_nodes; ++i_node) {
            std::size_t aux_id = r_geom[i_node].Id();
            const Vector r_node_dn_dx = row(grad_N[0], i_node);
            auto it_found = neigh_dn_dx_map.find(aux_id);
            if (it_found != neigh_dn_dx_map.end()) {
                auto& r_val = it_found->second;
                r_val += w*r_node_dn_dx;
            } else {
                neigh_dn_dx_map.insert(std::make_pair(aux_id, w*
                   r_node_dn_dx));
            }
        }
    }
    // Divide current coefficients by the total weight
    for (auto& neigh_data : neigh_dn_dx_map) {
        auto& r_val = neigh_data.second;
        r_val /= w_total;
    }
    // Update the surrogate nodes data
    sur_bd_nodes_map.insert(std::make_pair(r_bd_node.Id(),
       neigh_dn_dx_map));
```

8.2 Appendix 2

```
def Impose_MPC_Globally (main_model_part, result, skin_model_part,
    closest_element, projection_surr_nodes, T, node, j) :
# Interpolate the value at the projection
dirichlet_projection = Interpolation(skin_model_part, closest_element,
    projection_surr_nodes, j-1, node)
my_result = result[node.Id]
DofMasterVector = []
if node.IsNot(SLAVE) :
```

```
    CoeffVector = KratosMultiphysics.Vector(len(my_result)-1)
    ConstantVector = 0.0*KratosMultiphysics.Vector(len(my_result)-1)
else :
    CoeffVector = KratosMultiphysics.Vector(len(my_result))
    ConstantVector = 0.0*KratosMultiphysics.Vector(len(my_result))
i = 0
k = 0
Coeff_Slave = 0
for key, value in my_result.items() :
    node_master = main_model_part.GetNode(key)
    if node.IsNot(SLAVE) :
        # Need to find the "node" term and bring it to the left-hand-
            side
        if node.Id != key :
            DofMasterVector.append(node_master.GetDof(KratosMultiphysics
                .TEMPERATURE))
            ConstantVector[i] = dirichlet_projection
            CoeffVector[i] = - T[k]
            i = i + 1
            k = k + 1
        else :
            Coeff_Slave = - T[k]
            k = k + 1
    else :
        DofMasterVector.append(node_master.GetDof(KratosMultiphysics.
            TEMPERATURE))
        ConstantVector[i] = dirichlet_projection
        CoeffVector[i] = - T[i]
        i = i + 1
CoeffVector = CoeffVector / (1-Coeff_Slave)
ConstantVector = ConstantVector / (1-Coeff_Slave)
CoeffMatrix = KratosMultiphysics.Matrix(np.array(CoeffVector).reshape
    (1,-1))
DofSlaveVector = [node.GetDof(KratosMultiphysics.TEMPERATURE)]
# Create the constraint
main_model_part.CreateNewMasterSlaveConstraint("
    LinearMasterSlaveConstraint", j, DofMasterVector, DofSlaveVector,
    CoeffMatrix, ConstantVector)
return 0
```

References

[1] Rémi Abgrall and Smadar Karni. Computations of compressible multifluids. *J. Comput. Phys.*, 169(2):594–623, 2001.

[2] Chandrasekhar Annavarapu. *An Efficient Finite Element Method for Interface Problems*. PhD thesis, Duke University, 2013.

[3] Chandrasekhar Annavarapu, Martin Hautefeuille, and John E. Dolbow. A robust nitsche's formulation for interface problems. *Comput. Methods Appl. Mech. Eng.*, 225:44–54, 2012.

[4] Douglas N Arnold. An interior penalty finite element method with discontinuous elements. *SIAM Journal on Numerical Analysis*, 19(4):742–760, 1982.

[5] Douglas N Arnold, Franco Brezzi, Bernardo Cockburn, and L Donatella Marini. Unified analysis of discontinuous galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002.

[6] Nabil M Atallah, Claudio Canuto, and Guglielmo Scovazzi. The second-generation shifted boundary method and its numerical analysis. *Mathematical Models and Methods in Applied Sciences*, 30(14):2633–2673, 2020.

[7] Nabil M. Atallah, Claudio Canuto, and Guglielmo Scovazzi. Analysis of the shifted boundary method for the poisson problem in domains with corners. *Mathematics of Computation*, 90(331):2041–2069, 2021.

[8] Constantin Bacuta, James Bramble, and Jinchao Xu. Regularity estimates for elliptic boundary value problems in besov spaces. *Mathematics of Computation*, 72(244):1577–1595, 2003.

[9] Jingyi Chung, Steve Basting, and Sophia Lefantzi. Adaptive mesh refinement for multiscale and multiphysics problems. *SIAM Journal on Scientific Computing*, 35(2):B269–B292, 2013.

[10] A. Coll, R. Ribó, M. Pasenau, E. Escolano, J.Suit. Perez, A. Melendo, A. Monros, and J. Gárate. *GiD v.14 User Manual*, 2018.

[11] Pooyan Dadvand, Riccardo Rossi, Mariano Gil, Xavier Martorell, Juan Cotela, Estanislao Juanpere, Sergio Idelsohn, and Eugenio Onate. Migration of a generic multi-physics framework to hpc environments. *Computers and Fluids*, 80:301–309, 2013.

[12] Pooyan Dadvand, Riccardo Rossi, and Eugenio Onate. An object-oriented environment for developing finite element codes for multi-disciplinary applications. *Archives of Computational Methods in Engineering*, 17(3):253–297, 2010.

[13] M. De Corato and J.J.M. Slot. Finite element formulation of fluctuating hydrodynamics for fluids filled with rigid particles using boundary fitted meshes. 2018.

[14] Leszek Demkowicz, Jennifer Kurtz, and Walter Rachowicz. Anisotropic mesh adaptation for conforming finite element methods. *Computer methods in applied mechanics and engineering*, 167(1-2):191–220, 1998.

[15] Charbel Farhat, Arthur Rallu, and Sriram Shankaran. A higher-order generalized ghost fluid method for the poor for the three-dimensional two-phase flow computation of underwater implosions. *J. Comput. Phys.*, 227(16):7674–7700, 2008.

[16] Ronald Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152(2):457–492, 1999.

[17] Carlos Felippa. Nitsche's method for enforcing essential boundary conditions. *Computers & Structures*, 33(4-5):1085–1092, 1989.

[18] Carlos Felippa. *Flexible Multibody Dynamics: A Finite Element Approach*. Springer Science & Business Media, 2001.

[19] Vicente Mataix Ferrándiz, Philipp Bucher, Rubén Zorrilla, Riccardo Rossi, Jordi Cotela, Alejandro Cornejo Velázquez, Miguel Angel Celigueta, Josep Maria, Tobias Teschemacher, Carlos Roig, Miguel Maso, Guillermo Casas, Suneth Warnakulasuriya, Marc Núnez, Pooyan Dadvand, Salva Latorre, Ignasi de Pouplana, Joaquín Irazábal González, Ferran Arrufat, and Javi Gárate. Kratosmultiphysics/kratos: Release 9.2 (v9.2), May 2022.

[20] Anita Hansbo and Peter Hansbo. An unfitted finite element method, based on nitsche's method, for elliptic interface problems. *Comput. Methods Appl. Mech. Eng.*, 191(47):5537–5552, 2002.

[21] Jian Huang and Chi-Wang Shu. A moving mesh finite element method for the solution of the convection-diffusion-reaction equation. *Journal of Computational Mathematics*, 28(2-3):207–226, 2010.

[22] T. J. R. Hughes, G. Scovazzi, and L. P. Franca. Multiscale and stabilized methods. In *Encyclopedia of Computational Mechanics*. John Wiley, 2004.

[23] Rohit Jain, Sandeep Jain, and Bajpai Lokesh. Investigation on 3-d wing of commercial aeroplane with aerofoil naca 2415 using cfd fluent. *International Research Journal of Engineering and Technology (IRJET)*, 3, 2016.

[24] Young Joon Choi, Martien Martien, and Han Meijer. Simulation of the flow of a viscoelastic fluid around a stationary cylinder using an extended finite element method. *Computers & Fluids*, 57, 2012.

[25] Guoping Liu and Mengfei Wang. Advancing front method for 2d and 3d mesh generation: A survey. *Archives of Computational Methods in Engineering*, 24(2):253–294, 2017.

[26] Alex Main and Guglielmo Scovazzi. The shifted boundary method for embedded domain computations. part i: Poisson and stokes problems. *Journal of Computational Physics*, 372:972–995, 2018.

[27] Alex Main and Guglielmo Scovazzi. The shifted boundary method for embedded domain computations. part ii: Linear advection-diffusion and incompressible navier-stokes equations. *Journal of Computational Physics*, 372:996–1026, 2018.

[28] MathWorks. Matlab, 2020.

[29] A. Melendo, A. Coll, M. Pasenau, E. Escolano, and A. Monros. www.gidhome.com, 2018.

[30] Chandra Mohan and Prabal Talukdar. Three dimensional numerical modeling of simultaneous heat and moisture transfer in a moist object subjected to convective drying. *International Journal of Heat and Mass Transfer*, 53, 2010.

[31] Hashem M. Mourad, John Dolbow, and Isaac Harari. A bubble-stabilized finite element method for dirichlet constraints on embedded interfaces. *Int. J. Numer. Methods Eng.*, 69(4):772–793, 2007.

[32] Guglielmo Scovazzi Nabil M. Atallah, Claudio Canuto. Analysis of the shifted boundary method for the poisson problem in general domains. *Math. Comp.*, 2020.

[33] J. A. Nitsche. A variation method for the solution of dirichlet problems using subspaces which do not satisfy the boundary conditions. *Mathematical Methods in the Applied Sciences*, 23:261–265, 2000.

[34] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag New York, 2003.

[35] Alfio Quarteroni. *Numerical Mathematics*. Springer, 2014.

[36] Jim Ruppert. Delaunay triangulation and meshing: Application to finite elements. *Computational Geometry*, 3(1):61–86, 1995.

[37] Kailiang Wang. *A Computational Framework Based on an Embedded Boundary Method for Nonlinear Multi-Phase Fluid-Structure Interactions*. PhD thesis, Stanford University, 2011.

[38] Kevin G. Wang, Patrick Lea, Alex Main, Owen McGarity, and Charbel Farhat. Predictive simulation of underwater implosion: coupling multi-material compressible fluids with cracking structures. American Society of Mechanical Engineers, 2014.

[39] Eberhard Zeidler. *Applied Functional Analysis: Main Principles and Their Applications*. Springer Science, 2013.

[40] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 6th edition, 2005.

[41] Rubén Zorrilla, Antonia Larese, and Riccardo Rossi. A modified finite element formulation for the imposition of the slip boundary condition over embedded volumeless geometries. *Comput. Methods Appl. Mech. Engrg.*, 353:123–157, 2019.

[42] Rubén Zorrilla, Antonia Larese, and Riccardo Rossi. A discontinuous nitsche-based finite element formulation for the imposition of the general navier-slip condition over embedded volumeless geometries. *Comput. Methods Appl. Mech. Engrg.*, 2020.

[43] Rubén Zorrilla, Riccardo Rossi, Ronny Wuchner, and Eugenio Onate. An embedded finite element framework for the resolution of strongly coupled fluid-structure interaction problems. application to volumetric and membrane-like structures. *Comput. Methods Appl. Mech. Engrg.*, 368:113179, 2020.