Université catholique de Louvain
École Polytechnique de Louvain
Institute of Mechanics, Materials and Civil Engineering

Engineering analysis with non conforming meshes and levelsets

Gaëtan Bricteux

Thesis submitted in fulfillment of the requirements for the Ph. D. degree in Engineering Sciences

Jury members:

Prof. **J.F. Remacle**, Supervisor (Université catholique de Louvain)
Dr. **J. Lambrechts**, Supervisory panel (Université catholique de Louvain)
Prof. **V. Legat**, Supervisory panel (Université catholique de Louvain)
Prof. **É. Béchet** (Université de Liège)
Dr. **N. Chevaugeon** (Centrale Nantes)
Dr. **É. Wyart** (Cenaero)
Prof. **A. Holeyman**, President (Université catholique de Louvain)

September 30, 2016

Acknowledgement

This thesis is the result of several years of research at the Université catholique de Louvain. I would like to take the opportunity here to gratefully acknowledge all the people who supported me during these years.

First of all, I would like to express my sincere gratitude to my advisor Professor Jean-François Remacle for his continuous support, motivation and for his immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I wish to express my warm and sincere thanks to the members of my doctoral committee : Doctor Marc Duflot and Professor Vincent Legat for their pertinent advices and enriching discussions. I am thankful to the other members of my jury : Doctor Jonathan Lambrechts, Professor Éric Béchet, Doctor Nicolas Chevaugeon, Doctor Éric Wyart and Professor Alain Holeyman for their conscientious reading and their helpful and constructive comments.

I wish to thank my collaborators at MEMA : Amine Ouaar, Thomas Toulorge, Dieu-Linh Quan, François Henrotte and Émilie Marchandise for their stimulating and fruitful discussions.

Finally, I would like to express my deepest gratitude to my family and friends for their encouragements and their support through hard times.

Last but not the least, a special thanks to my beloved Donatienne for her love, her patience, her encouragements, constant supporting me and for being beside me always throughout my life.

Contents

Introduction

To design new mechanical parts or new structures, it has become the norm for engineers to use engineering analyses to predict their behaviour in real loading conditions. This process is very common in engineering industries such as ship-building, aerospace and automotive. More than a million finite element analyses are performed every day in engineering design offices throughout the world [32]. Numerical simulations enable to obtain an approximation of the displacements and the stresses that a part undergoes and reduces the costs of development by avoiding real size experimentations. The Finite Element Method (FEM) is predominant among numerical simulations in industry. It is based on the subdivision of the computational domain in smaller parts called elements where polynomial functions are defined to form a discrete piecewise polynomial subspace of the solution space. A variational formulation of the problem can be discretized. The discretized equations are written in each element and assembled to form a global linear system to solve. The advantages of the method are its ability to solve non-linear problems, its convergence that has been mathematically demonstrated and the fact that complex boundaries can be easily represented through unstructured meshes. Yet it comes with the price of mesh generation.

The engineering analysis process can be separated in several steps illustrated on figure 1. First, the geometry of the part is drawn in a Computer Aided Design (CAD) system. A CAD representation consists in a topological datastructure that represents topological adjacencies in the model and geometrical informations. In 3 dimensions, we have four kinds of model entities, one for each dimension : vertices (0D), edges (1D), faces (2D) and volumes (3D). Each model entity is associated to a geometrical representation, as NURBS, splines, surfaces of revolution and many others. The CAD model can be considered as an exact representation of the geometry, CAD tolerances being way beyond manufacturing tolerances.

The second step is the mesh generation. Most of the numerical computations use unstructured meshes composed of triangular elements in 2D or tetrahedral elements in 3D. Quadrangular or hexahedral elements present interesting properties as the decrease of the number of mesh elements yet the mesh generation

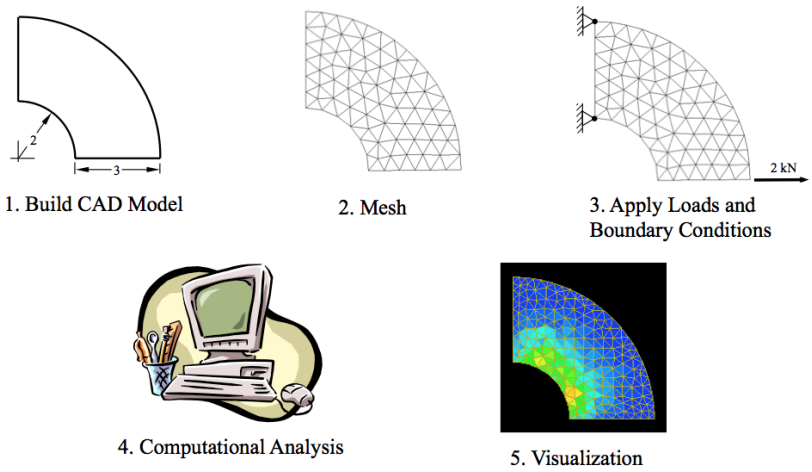


Figure 1 – Different steps of an engineering analysis.

is a much harder problem [54]. This step can be complex as the CAD representation is not suited for the finite element method. Problems can occur when an edge of the CAD model is much smaller than the mesh size. The CAD model then has to be simplified by removing small details. Problems also occur when the geometry is invalid with common surfaces of two adjacent solids not being coincident. A large amount of time cost is to be devoted making the geometry valid and generating the mesh. It can account for more than 80% of the overall analysis time [32]. These modifications of the geometry create approximations and are only guided by concerns linked to the numerical method. Moreover, these geometry modifications can create significant error for problems that are strongly sensitive to geometric imperfections as for example the buckling of thin shells where small geometrical imperfections can lead to a considerable reduction in buckling load [31].

The third step of the design process is the application of the loads and the boundary conditions. Again, this step creates approximations as the real boundary conditions are simplified. In standard FEM, the loadings and boundary conditions are applied on the nodes of the mesh which is an inaccurate reproduction of the geometry.

Then the boundary value problem is posed and can be solved. To solve a problem with the FEM, the partial differential equations are first written in a variational form. The equations are then discretized, expressed inside the finite elements, and recombined into a global system. This linear system can be

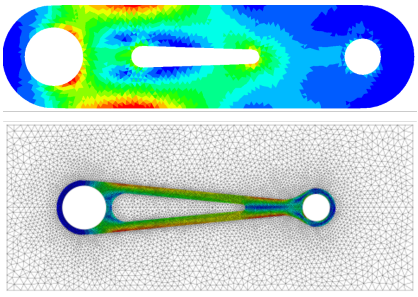
solved thanks to linear solvers. As the number of elements can be important, attention has been brought to the optimization of linear systems. Direct solvers are preferred for their robustness but can be more expensive in terms of run time and memory usage [36]. Iterative methods, enabling to find the solution by improving step by step a first approximation of the solution, have to be used for large problems.

Finally, once the problem is solved, the solution and data of interest can be visualized in a post-processing software. A posteriori error estimations can be computed to validate the obtained result [1]. If the error is larger than what is acceptable, the discretization can be modified to obtain a better solution.

In this work, we would like to find a method that avoids the geometry modification during the meshing step, allowing to reduce the time devoted to this task. Simulations with evolving interfaces are particularly concerned as the geometry has to be remeshed for each step. Examples are shape optimization where medium is added or removed to a part to improve its resistance and reduce its weight (see figure 2(a)), moving interfaces between several bodies like in fluid-structure problems (see figure 2(b)), multiphase flows (see figure 2(c)) or phase transfer and fracture mechanics where cracks create discontinuities by propagating inside a body (see figure 2(d)). An important application is also the addition or removing of geometrical features inside an existing mesh. The method should avoid restarting the process from scratch.

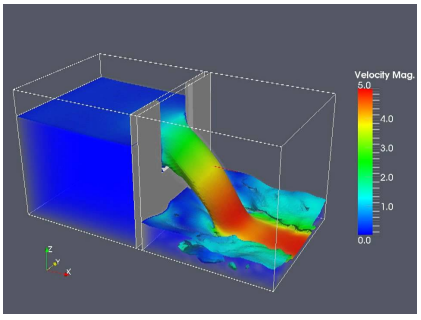
We propose to use levelsets to represent the geometry. Levelsets describe the boundary of the computational domain and the interfaces with an implicit representation. It is elaborated as a distance function that gives the distance from a point in space to the closest point of the boundary. The iso-zero of the distance function gives the boundary representation of the geometry. An advantage of this method is that it enables to combine several geometry representations by performing boolean operations on the distance functions. Several geometries can be easily intersected, united or cut one from each other thanks to boolean operators.

To obtain a mesh of the geometry represented with the levelset technique, the representation is applied on a mesh surrounding the geometry. But this creates embedded boundaries not conforming to the mesh. The eXtended Finite Element Method (X-FEM), first presented by Nicolas Moës et al. [42], enables to solve problems with interfaces passing through the elements by extending the space of shape functions inside the elements crossed by the interfaces. Neumann boundary conditions are easily applied on the embedded boundaries as they are applied weakly in the variational formulation. Yet Dirichlet boundary conditions are more complex to impose on these embedded interfaces. Lagrange multipliers can be used but this leads to instabilities in the solution [34]. Several methods have been proposed to tackle this problem, by building a stable Lagrange mul-

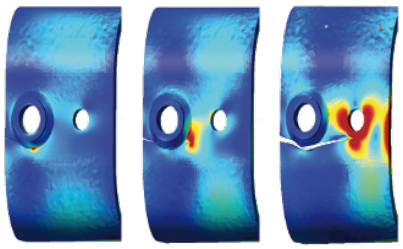


(a) Shape optimization of a connecting rod (L. Van Miegroet).

(b) Fluid-structure interaction around a wind turbine (Siemens PLM).



(c) Multiphase flow (OpenFOAM).



(d) Crack propagation (Cenaero).

Figure 2 – Examples of evolving interfaces.

tipliers space [41] or by adding stabilizing terms to the formulation [8, 39]. Yet these solutions are not optimal. Algorithms to build stable Lagrange multipliers spaces are not straightforward to extend to complex problems. And stabilizing terms require the computation of stabilizing parameters.

In this thesis, we present two new methods to solve the instability problem. These methods are expected to be easily generalized while not being time consuming and showing optimal rates of convergence. The first method is based on the addition of a Laplace stabilization term to the formulation. It is compared to other stabilization methods to enlighten its advantages and calibrate its parameter. The second method is based on local mesh adaptation around the interface to avoid non conforming formulation. It uses the classical finite element formulation.

The thesis is organized as follows :

Chapter 1 presents the levelset method used to describe the geometry. Several methods to compute a distance function and their implementation are explained and different combinations of levelsets are described. The cut of the finite elements crossed by the iso-zero of the levelset is explained and examples are shown.

Chapter 2 describes the instability issue caused by the application of Dirichlet boundary conditions on non conforming boundaries for a Laplace problem. Several existing solutions are presented and the new stabilized formulation is detailed. The stabilizing parameter is then calibrated to an adimensional value in order to be valid for any problem.

Chapter 3 presents the method based on local mesh adaptation. The need to anisotropic mesh refinement is explained and the implementation of the method using mesh metrics is described. A numerical example is analysed to validate the method and enlighten its advantages.

Chapter 4 presents test cases on linear elasticity problems using both presented methods.The methods are compared in terms of computational costs and discretization error to draw a rule of good practice to solve problems with Dirichlet boundary conditions on non conforming interfaces.

A final chapter summarizes the conclusions of the whole work and presents perspective for future work that can be done to improve the methods and broaden their domain of application.

Several papers have been published and presented in international conferences during this thesis to report the obtained results :

Papers :

- Emilie Marchandise, Gaëtan Compère, Marie Willemet, Gaëtan Bricteux, Christophe Geuzaine and Jean-François Remacle. **Quality meshing based on STL triangulations for biomedical simulations.** *International Journal for Numerical Methods in Biomedical Engineering*, 26(7):876-889, 2010.

- Dieu-Linh Quan, Thomas Toulorge, Gaëtan Bricteux, Jean-François Remacle, and Emilie Marchandise. **Anisotropic adaptive nearly body-fitted meshes for cfd.** *Engineering with Computers*, 30(4):517-533, 2014.

- Dieu-Linh Quan, Thomas Toulorge, Emilie Marchandise, Jean-François Remacle, and Gaëtan Bricteux. **Anisotropic mesh adaptation with optimal convergence for finite elements using embedded geometries.** *Computer Methods in Applied Mechanics and Engineering*, 268:65-81, 2014.

- Gaëtan Bricteux and Jean-François Remacle. **A new stabilized method for imposing Dirichlet boundary conditions on embedded interfaces.** *In preparation*, 2016.

Conference presentations :

- G. Bricteux, J-F. Remacle, A. Ouaar and M. Duflot, **Discontinuous fields integration in a structured mesh using the level set method.** *ACOMEN 2008, Liège.*

- G. Bricteux, J-F. Remacle, A. Ouaar and M. Duflot, **Discontinuous fields integration in a structured mesh using the level set method.** *WCCM-ECCOMAS 2008, Venice.*

- G. Bricteux, M. Duflot and J-F. Remacle, **Two approaches for imposing Dirichlet boundary conditions on embedded interfaces.** *ECCM 2010, Paris.*

- G. Bricteux, M. Duflot and J-F. Remacle, **Imposing Dirichlet boundary conditions in the eXtended Finite Element Method.** *ACOMEN 2011, Liège.*

- G. Bricteux, E. Marchandise and J-F. Remacle, **Alternative methods to represent embedded interfaces in a mesh.** *PUM Workshop 2012, Berlin.*

Chapter 1

Geometrical model description using levelsets

The first step to carry out a numerical computation is to define the geometry. The classical way in most computational software is to draw the geometry with a CAD model. It enables to design curves in 2D and surfaces and solids in 3D. It also enables to store informations such as materials, dimensions and tolerances.

To perform a mechanical analysis of the behaviour of the geometry, the use of the finite element method is the most spread. But it comes with the price of mesh generation that is a difficult problem. Mesh generation accounts for more than 80% of the overall analysis time. This step can be difficult to execute if the CAD model has some defects. For example, common surfaces of two adjacent solids may not be coincident. This can cause the geometry to be invalid with non closed volumes or overlapping volumes. Moreover, when the geometry changes, the geometry has to be meshed again, what causes a major lack of time.

Our goal is to develop a technique that will not be affected by the defects of the geometry and that will allow to remesh easily the geometry when changes are performed. The levelset method enables to reach those goals. This implicit method enables to determine whether a point in space is inside or outside the geometry.

First, the levelset method will be presented. Several types of levelsets will be developed as analytical levelsets and the distance to a triangulation. Then, operators acting on levelsets will be presented. These enable to use the constructive solid geometry to design complex domain shapes. Finally, the cut of the elements in a mesh surrounding the domain will be discussed to be able to integrate a field on the geometry. Some examples will end the chapter to show

the possibilities the method can handle.

1.1 Definitions

A *levelset* $L_c(\phi)$ is defined in mathematics as the set of points (x, y, z) whose location satisfies :

$$L_c(\phi) = \{(x, y, z) | \phi(x, y, z) = c\} \tag{1.1}$$

for a defined scalar levelset function $\phi(x, y, z)$. In 2D, this set of points defines an iso-contour. In 3D, it defines an iso-surface.

The *levelset method* is a numerical technique used to define implicitly a geometry and to compute interface motion. It was first presented by Sethian and Osher [45, 53]. Levelsets are used in many different domains : fluid mechanics to model wave motion [44], fracture mechanics with crack propagation [57], shape optimization [3] or image segmentation [15].

The boundary of the geometry is represented by the iso-zero $L_0(\phi)$ of the levelset function inside a computational domain. If the levelset is signed, the sign of the levelset function can be used to determine the interior and the exterior parts of the geometry. The function $\phi(x, y, z, t)$ may depend on time if the boundary moves. If the speed of the boundary in the normal direction is $\mathbf{v}$ then the function ϕ satisfies the Hamilton-Jacobi equation of the levelset :

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} = \mathbf{v}.|\nabla \phi| \tag{1.2}$$

Another implicit method to describe the evolution of a boundary is the Fast Marching method [53]. It consists in solving the boundary value problem :

$$\begin{aligned} |\nabla t(\mathbf{x})| \cdot \mathbf{v}(\mathbf{x}) &= 1 \quad \text{for} \quad \mathbf{x} \in \Omega \\ t(\mathbf{x}) &= 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega \end{aligned}$$

with $\mathbf{v}(\mathbf{x})$ the speed of the boundary in the normal direction and $t(\mathbf{x})$ the time it would take to reach $\partial\Omega$ starting from the point $\mathbf{x}$. The boundary at time t_0 is described by $\{(\mathbf{x}) | t(\mathbf{x}) = t_0\}$. As we will deal with static problems in the following analyses, the levelset method is more suited.

A simple levelset function can be the signed distance function to the boundary of the geometry. For example, figure 1.1 shows the distance function of the cross-section of a wing profile. The equation of the NACA family of airfoils is given analytically :

$$\phi(x,y) = y \pm 5ct \left[0.297\sqrt{\frac{x}{c}} - 0.126\left(\frac{x}{c}\right) - 0.352\left(\frac{x}{c}\right)^2 + 0.284\left(\frac{x}{c}\right)^3 - 0.102\left(\frac{x}{c}\right)^4 \right]$$

where c is the chord length, t is the maximum thickness as a fraction of the chord and x is the position along the chord varying from 0 to c [47]. The represented airfoil is a NACA0012, meaning it has no camber (it is symmetric) and $t = 0.12$.

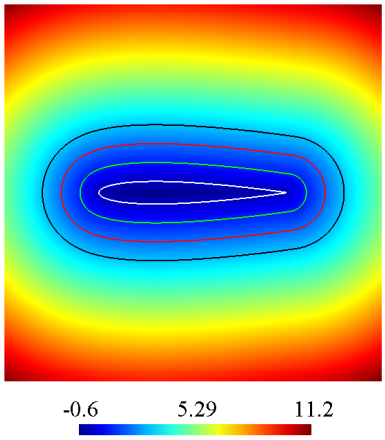


Figure 1.1 – Analytical levelset function of a NACA0012 wing profile in 2D with iso-contours (white : iso-0, green : iso-1, red : iso-2 and black : iso-3).

The boundary of the wing profile is described by the iso-zero of the levelset function. If the computation attempts to solve the fluid dynamics around the wing, the computational domain will be described by the part of the square domain with positive values. While an analysis of the mechanical behaviour of the solid wing will consider the part of the domain with negative values.

Levelsets can be obtained from different sources. The simplest one is to define an analytical function that gives a signed value at any point (x, y, z). Simple geometries can be defined by analytical functions as planes, cylinders, spheres, quadrics, and more complicated ones as the NACA airfoil.

Analytical functions of the geometry are often not available for complex geometries. Other sources of information can produce a levelset function. For example, a levelset function can be computed with a colour code on an image where the value is given by the colour at a certain point relating to a predefined scale [53].

Another example of levelset function can be computed as the distance to a set of triangles covering the geometry. The value is defined as the distance to the closest triangle and the sign can be computed with the help of the outward normal of the triangles. This is convenient for geometries obtained from scans where a triangulation is obtained. This triangulation is often not adapted for a finite element computation as the triangles have a poor quality but can be used to compute a levelset function. Figure 1.2 shows the levelset value in a plane around the triangulation of a pelvis bone obtained from a scan.

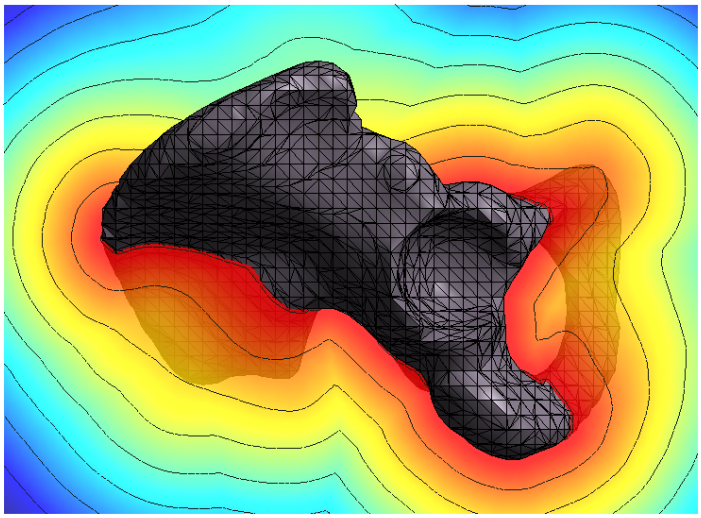


Figure 1.2 – Levelset function of a pelvis bone obtained from a scan triangulation with iso-contours.

A last example of levelset function can be computed as the distance to a set of points or lines with an offset. The distance to a set of points is used to reconstruct objects with radial basis functions [14]. The distance to lines can be used to model yarns following a centerline in woven composites for example [55].

1.2 Distance functions

A levelset function can be the distance to the boundary of the geometry. Here are presented the most used distance functions : analytical distance functions and the distance to a triangle mesh.

1.2.1 Analytical distance functions

Analytical levelset functions are simple. The coordinates of the point in space are set in an analytical scalar function that returns the distance to the boundary of the geometry. Here are some simple examples :

- $L_0(\phi) = ax+by+cz-d$ gives the distance to a plane with outward normal (a, b, c) and passing through the point $(0, 0, \frac{d}{c})$ (if not parallel to the z axis);
- $L_0(\phi) = \sqrt{(x_c - x)^2 + (y_c - y)^2 + (z_c - z)^2} - r$ gives the distance to a sphere centered on (x_c, y_c, z_c) and with a radius equal to r.

1.2.2 Distance to a triangle mesh

The distance to a triangle mesh is defined as the distance to the closest triangle of the mesh. The distance to a triangle has to be defined first. Then the sign of the distance is needed to determine the interior of the domain.

Distance to a triangle

The distance of a vertex $\mathbf{p}$ to a triangle is the distance to the closest point in the triangle $\mathbf{p}_t$ [6]. But this closest point is not that straightforward to determine. Three cases can occur, depending on which region lies the normal projection of the vertex on the triangle plane ($\mathbf{p}_p$) (see figure 1.3).

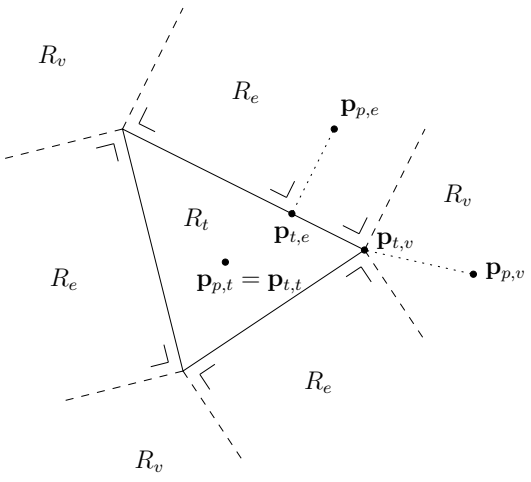


Figure 1.3 – Computing the distance to a triangle : regions of the normal projection of a vertex on a triangle plane and closest points.

- The normal projection lies into the triangle or on an edge or a vertex of the triangle (region R_t). The closest point $\mathbf{p}_{t,t}$ is equal to the projection $\mathbf{p}_{p,t}$ and the distance to the triangle is the distance between the vertex $\mathbf{p}$ and $\mathbf{p}_{t,t}$: $\|\mathbf{p} - \mathbf{p}_{t,t}\|$.

- The normal projection lies in regions R_e defined by the outward normal projection of the edges. The closest point $\mathbf{p}_{t,e}$ is the normal projection on the closest edge $\mathbf{p}_{p,e}$ and the distance to the triangle is the distance between the vertex $\mathbf{p}$ and the closest point : $\|\mathbf{p} - \mathbf{p}_{t,e}\|$.

- The normal projection lies in regions R_v defined as the resulting area in the triangle plane. The closest point $\mathbf{p}_{t,v}$ is the closest vertex of the triangle and the distance to the triangle is the distance of the vertex $\mathbf{p}$ to the closest point : $\|\mathbf{p} - \mathbf{p}_{t,v}\|$.

Signed distance

Once the distance is computed, it needs to be signed to define the interior of the domain. The sign is usually defined as negative if the vertex is inside the domain and positive otherwise. Several methods exist to determine the sign at one point.

Some methods to determine the sign of the levelset have been proposed based on scan conversion. The sign is computed in a separate pass after the computation of the distance. The intersection of the mesh with a plane perpendicular to the z-axis produces a 2D contour that can be scan converted (see figure 1.4). Knowing a point outside the domain $\mathbf{p}_{ext}$ enables to draw a line segment joining a vertex and this point. Then counting the number of intersections between this segment and the contour tells if the vertex is inside or outside the domain. An uneven number of intersections means the vertex is inside the mesh ($\mathbf{p}_1$) and it is given a negative sign. An even number of intersections means the vertex is outside the mesh ($\mathbf{p}_2$) and it is given a positive sign [46]. This method always gives the sign of the levelset at one vertex but it involves a scan conversion step and the computation of the intersections is complex and has to be executed for each vertex.

Another approach, the *Generalized Winding Number* method is a robust method that uses the solid angles [33]. A solid angle of an object at a point $\mathbf{p}$ is the relation between the area of the projection of the object on a sphere centered at $\mathbf{p}$ and the square of the radius of the sphere (see figure 1.5). The generalized winding number of a point $\mathbf{p}$ is defined as the sum of the weighted solid angles :

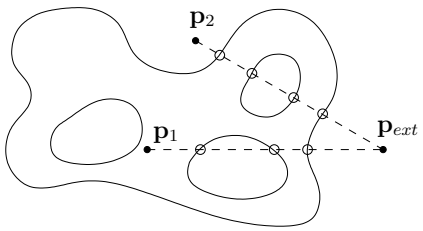


Figure 1.4 – Scan of the intersection of a 3D geometry with a plane. Intersections of segments linking query points to an exterior point with the contour to determine the sign of the levelset.

$$w(\mathbf{p}) = \sum_{f_i F} \frac{1}{4\pi} \Omega_{f_i}(\mathbf{p}) \tag{1.3}$$

where f_i is the ith triangle of the surface of the domain F and Ω_{f_i} is the solid angle at point $\mathbf{p}$ subtended by f_i. If $w(\mathbf{p}) = 1$, then $\mathbf{p}$ lies inside the domain and if $w(\mathbf{p}) = 0$, then $\mathbf{p}$ lies outside the domain. If the surface of the domain is self-intersecting or not closed (but correctly oriented), a generalized winding number can be computed and tends toward 1 as $\mathbf{p}$ is *more* inside the domain and toward 0 as $\mathbf{p}$ is *more* outside.

This method is robust as it can compute the sign for complex geometries with non closed boundary but it is not fast as all the solid angles have to be computed to find the sign at one point. The method is $\mathcal{O}(\sqrt{n})$ for a point, with n the number of triangles in the triangulation.

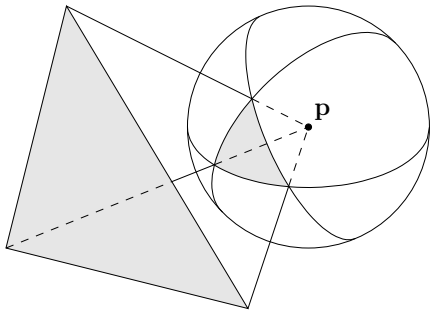


Figure 1.5 – Solid angle subtended by a triangle at point $\mathbf{p}$.

Finally, if the triangulation defines a closed non-self-intersecting manifold, a local method based on the outward normal of the triangles can be used. The

sign of the levelset is computed as the sign of the scalar product between the outward normal of the closest triangle and the vector pointing from the closest point $\mathbf{p}_p$ to the vertex $\mathbf{p}$.

This involves that all the triangular elements should have an outward normal pointing in the same direction (inward or outward the domain). If this is not the case, then one point outside the domain has to be known. One triangle can be oriented in the correct direction with respect to this point. The triangulation can be oriented by propagating the orientation of this triangle step by step to the adjacent triangles.

When the normal projection $\mathbf{p}_p$ does not lie in the triangle, the determination of the sign becomes more complicated as the closest point belongs to several triangles (2 triangles when the closest point is on an edge and at least 3 triangles when the closest point is on a vertex).

A *pseudo-normal* on the edges and vertices of the triangle mesh has to be determined. A naive definition of this pseudo-normal would be to compute the mean normal between the normals of the adjacent triangles. But a simple counter example can be found for the case where the closest point is a vertex. If one adjacent triangle is split into several triangles, all adjacent to the vertex, then the mean normal will be modified while the geometry remains the same.

An *angle weighted normal* has been proposed by Sequin [52] and Bærentzen [6] has proven that this choice of pseudo-normal leads to a correct technique for computing the sign. The idea is to weigh the normals of the adjacent triangles by the incident angle :

$$\mathbf{n}_\alpha = \sum_i \alpha_i \mathbf{n}_i \tag{1.4}$$

where α_i is the incident angle of triangle i and $\mathbf{n}_i$ its normal. For an edge, the 2 incident angles are equal to π. For a vertex, the incident angles have to be computed.

Several advices can be formulated to optimize the computation of the distance function of a large set of nodes, as the vertices of a mesh surrounding the triangulation. First, the angle weighted normals on each vertex and each edge of the triangulation have to be computed in a preliminary step. Then it is faster to compute the distance from one triangle at a time to all the points as the transformation matrix used to project points into the plane of the triangle can be reused. To compare the distances, the square distances can be used and the square root will only be computed for the closest triangle. Finally, the points can be stored in a tree of bounding boxes to eliminate remote sets of points.

In our implementation, the method has been designed to compute the distance of a single point to a triangulation. The initialization of the levelset function is executed in two steps. First the relation between the vertices and the triangles is stored. And secondly the vertices are stored in a kd-tree [10]. A kd-tree is a binary data structure that subdivides the space into rectangular or parallelepipedic cells. Each node is associated with a set of points that lie in a cell. To construct this binary tree, a bounding box containing all the data points is recursively split in two parts with an axis-orthogonal plane following predefined rules until a node contains less than a maximum number of points (see figure 1.6). The ANN library (Approximate Nearest Neighbor) uses this data structure to find efficiently the closest points of a query point in a point set [38]. The number of closest returned points can be defined by the user. The algorithm operates recursively on the nodes of the tree, following the closest nodes of the query point.

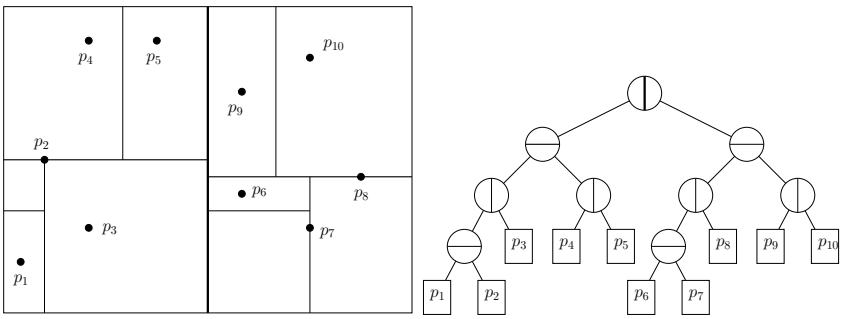


Figure 1.6 – Subdivision of space and corresponding kd-tree.

But to compute the levelset, the closest triangle is needed and not the closest point. The triangles connected to the closest nodes returned by the ANN algorithm are selected. 5 closest nodes are chosen to be returned by default. These 5 nodes are connected to about 20 triangles for a smooth mesh. The distance to these triangles is computed. If the closest point lies on an edge or a vertex of the triangulation, several triangles are the closest. The angle weighted normal is then computed. The sign of the levelset is determined with the normal of the triangle if the closest point lies inside the triangle or using the angle weighted normal otherwise.

This method is the simplest to compute the distance of the query point to the triangulation but could not be exact. If triangles have low aspect ratio and the geometry is thin, the closest point in the triangulation can be outside the triangles connected to the closest vertices. In this case, the number of closest points returned by the ANN algorithm has to be increased. Considering the worst case, if this number is increased to the number of vertices, the distance

will be exact but the distance to every triangle will have to be computed.

To avoid this problem, the triangles could be directly stored in a tree of bounding boxes. The smallest and greatest distance to anything inside the box can be easily computed. Axis aligned bounding boxes (AABB) are easy to construct. Oriented bounding boxes (OBB) need to compute the orientation of the distribution of points but accelerates the queries in the tree structure[7]. Another approach similar to an octree based method is the Meshsweeper algorithm proposed by Guezlec[26]. It is based on a hierarchical representation of the mesh. At each level, a bounding volume is associated with every triangle and a distance interval is computed for each bounding volume.

To analyse the performance of the method, the distance function to a geometry is computed on the vertices of a box surrounding the geometry. The geometry of a squirrel composed of 20 000 triangles is chosen. The levelset function is represented on figure 1.7. The mesh is refined four times to increase the number of triangles. Each refinement cuts the triangles into 4 subtriangles so it multiplies the number of elements by 4.

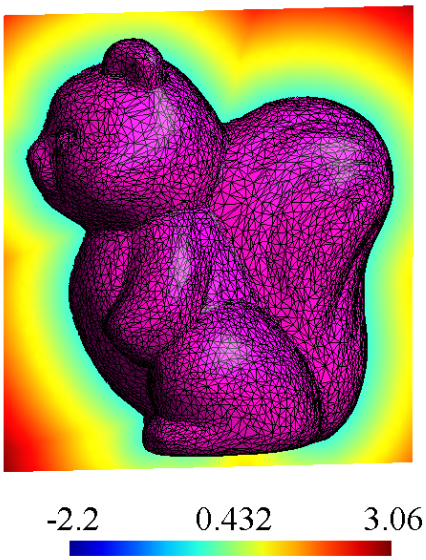


Figure 1.7 – Distance function to a geometry to analyse the time of computation.

Table 1.1 shows the computation time of the distance function to the triangulation of the 4 meshes of the geometry. The number of vertices of the cube surrounding the domain where the function is evaluated varies also from 3366 to 586 760. We can see that the time of computation varies almost linearly with

the number of points where the distance function is evaluated. The number of triangles in the triangulation is not directly linked to the time of computation as the ANN algorithm finds efficiently the closest points of the triangulation and the distance to the triangles is only computed for a constant number of closest triangles. As the number of points in a smooth triangulation is about the half of the number of triangles, the time of computation should vary as the order of the ANN algorithm, $\log_2(\frac{n}{2})$, with n the number of triangles, as the space around the triangulation is recursively split into 2 parts to find the closest points.

		Number of points			
		3 366	12 154	87 771	586 760
Number of triangles	20 000	0.071	0.25	1.77	11.8
	80 000	0.095	0.31	2.09	14.1
	320 000	0.15	0.44	3.28	17.8
	1 280 000	0.29	0.70	4.35	26.2

Table 1.1 – Time of computation (in seconds) of the distance function to a triangulation for different numbers of triangles in the triangulation and different numbers of query points in the mesh surrounding the geometry.

1.3 Implementation

Some levelset functions have been presented. To design complex geometries, the combination of several levelset functions can be useful. Therefore, two types of objects are needed, primitive levelsets and operators acting on other levelsets. Several combinations can be performed as in the constructive solid geometry. The different primitive levelsets and operators will be piled in a stack to obtain a Reverse Polish Notation of the constructive geometry. This history based procedure enables to access the levelsets that have been used to create the final geometry.

The levelset functions have been implemented in the source code of the `Gmsh` software in C++ code. The different levelset functions are implemented as deriving from an abstract class `gLevelset`. We want to have a general class that defines a levelset that returns a distance function independently of its implementation. Two types can derive from the abstract class : primitive levelsets and operators acting on other levelsets. The `gLevelset` class defines several functions. The main function is `operator()` that returns a scalar with the coordinates of a point in space as parameters. It also can determine if the levelset is a primitive, and otherwise give the children it acts on. Finally, it has a tag that enables to distinguish every levelset.

Listing 1.1 shows the data structure of our code in C++ [58]. We can see the abstract class `gLevelset` (line 1) with its pure virtual functions. No object can be instantiated directly from that class. Then the `gLevelsetPrimitive` class (line 16) derives from `gLevelset`. It still has one pure virtual function, the operator that returns a scalar with coordinates as parameters. That method is required to be overridden by the inheriting classes. Finally, the class of a primitive levelset is shown, `gLevelsetSphere` (line 24). It defines the analytical distance function to a sphere, given the center and the radius of the sphere.

```
1  class gLevelset
2  {
3   protected:
4    int tag_;
5   public:
6    gLevelset() {}
7    virtual double operator() (double x, double y, double z) const = 0;
8    virtual std::vector<gLevelset *> getChildren() const = 0;
9    virtual double operate (double d1, double d2) const = 0;
10   virtual bool isPrimitive() const = 0;
11   void getPrimitives(std::vector<gLevelset *> &primitives);
12   void getPrimitivesPostOrder(std::vector<gLevelset *> &primitives);
13   void getReversePolishNotation(std::vector<gLevelset *> &gLsRPN);
14 };
15
16 class gLevelsetPrimitive : public gLevelset
17 {
18  public:
19   gLevelsetPrimitive(int tag) : tag_(tag) {}
20   virtual double operator()(double x, double y, double z) const = 0;
21   bool isPrimitive() const { return true; }
22 };
23
24 class gLevelsetSphere : public gLevelsetPrimitive
25 {
26  protected:
27   double xc, yc, zc, r;
28  public:
29   gLevelsetSphere(const double &x, const double &y, const double &z,
30                   const double &R, int tag = 1);
31   virtual double operator()(double x, double y, double z) const
32   {
33     return sqrt((xc - x) * (xc - x) + (yc - y) * (yc - y) +
34                 (zc - z) * (zc - z)) - r;
35   }
36 };
```

Listing 1.1 – C++ code of the levelset data structure.

To facilitate the run of test cases, a `Python` wrapper exist that translates the C++ functions in `Python`. This enables to run simulations without need to compile the code. The only difficulty is that `Python` does not support the pointers. Methods that use tables in pointers format as parameters have to be rewritten with vectors instead.

1.4 Operations on levelsets

To model complex geometries, several primitive levelsets can be combined to obtain a resulting distance function. Boolean operations can be performed on levelsets. Taking the minimum value of several levelset functions will result in the union of the domains with negative values (see figure 1.8).

$$\bigcup(\phi_1, \phi_2) = \min(\phi_1, \phi_2)$$

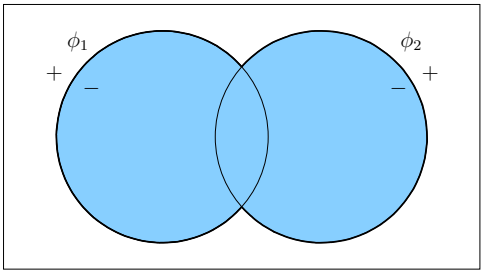


Figure 1.8 – Union of two levelsets ϕ_1 and ϕ_2.

While taking the maximum value will result in their intersection (see figure 1.9).

$$\bigcap(\phi_1, \phi_2) = \max(\phi_1, \phi_2)$$

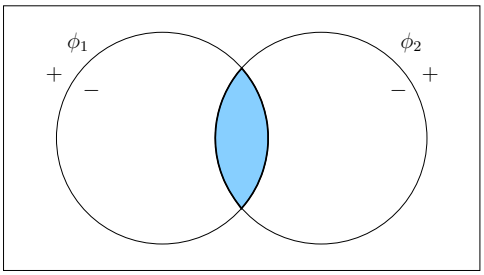


Figure 1.9 – Intersection of two levelsets ϕ_1 and ϕ_2.

A third operation can define the cut of a domain by another by taking the maximum of the first value and the opposite of the second one (see figure 1.10). This operator is not commutative. The order of the levelsets is important as the first levelset is cut by the second.

$\backslash(\phi_1, \phi_2) = max(\phi_1, -\phi_2)$

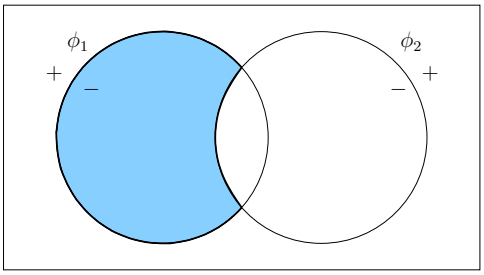


Figure 1.10 – Cut of levelset ϕ_1 by levelset ϕ_2.

Reversing the sign of a levelset will consider the opposite of the domain. These operations can also be executed one onto another.

Listing 1.2 shows the data structure of the operator levelsets. An abstract class deriving from the `gLevelset` class is defined for the operators (line 1). It stores the children in a vector. Operators can act on one or more than two levelsets. The `operator()` function compares the levelset value of its children at the required position. The comparison method `operate` (line 20) has to be implemented by the specific operator classes. The tag of an operator is the tag of his first child. An operator can return its children to be able to return its primitives and a reverse polish notation of its tree structure. The union operator class `gLevelsetUnion` (line 28) is then shown. The function `operate` returns the minimum of the levelset values to take the union of the levelsets.

A simple way to write the relation between the different combined levelset functions is to use the reverse polish notation. Every operator keeps in memory the levelset functions it is performed on. The reverse polish notation is a list of levelsets where each operator acts on the previous two levelsets, that can be an operator. The last operator of the list gives the final value of the levelset. As an example, the union of the two circles in figure 1.11 minus their intersection can be expressed as : $\bigcup \ \backslash \ \phi_1 \ \phi_2 \ \backslash \ \phi_2 \ \phi_1$ or $-\bigcup \ \bigcap \ \phi_1 \ \phi_2 \ \bigcap \ -\phi_1 \ -\phi_2$.

```
1  class gLevelsetOperator : public gLevelset
2  {
3  protected:
4    std::vector<gLevelset *> children;
5   public:
6    gLevelsetOperator(const std::vector<gLevelset *> &p)
7    {
8      children = p;
9    }
10   double operator()(double x, double y, double z) const
11   {
12     double d = (*children[0])(x, y, z);
13     for(int i = 1; i < (int)children.size(); i++){
14       double dt = (*children[i])(x, y, z);
15       d = operate(d, dt);
16     }
17     return d;
18   }
19   std::vector<gLevelset *> getChildren() const {...}
20   virtual double operate(double d1, double d2) const = 0;
21   bool isPrimitive() const
22   {
23     if(children.size() != 1) return false;
24     return children[0]->isPrimitive();
25   }
26 };
27
28 class gLevelsetUnion : public gLevelsetOperator
29 {
30  public:
31   gLevelsetUnion(std::vector<gLevelset *> p)
32     : gLevelsetOperator(p) {}
33   double operate(double d1, double d2) const
34   {
35     return (d1 < d2) ? d1 : d2; // minimum of d1 and d2
36   }
37 };
```

Listing 1.2 – C++ code of the operators data structure.

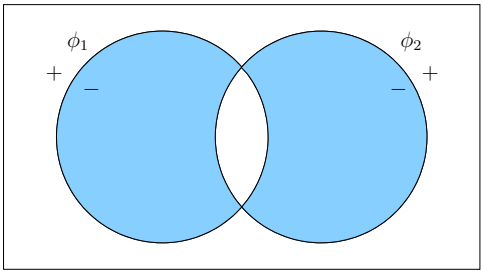


Figure 1.11 – Boolean operations on levelsets in a 2D domain. Union of two circles minus their intersection.

As a last example, we can reproduce the solid constructed on the Wikipedia® page of the constructive solid geometry (CSG) [59]. It consists of the intersection of a cubic box and a sphere with the same center cut by three cylinders in the three axis directions. On figure 1.12, we can see on the left the binary tree of the

CSG with primitives at the leafs and operations at the nodes and on the right a reproduction of the solid with our levelset software. To reproduce the solid, the mesh of a cubic box has been created with more than 15.000 tetrahedra. The listing 1.3 shows the Python code used to create the solid. The mesh of the cube is loaded. Then the sphere and cylinder levelsets are created and combined to form a single cut levelset function. Finally the mesh is cut. The operation takes a bit more than one second to cut more than 6.000 tetrahedra.

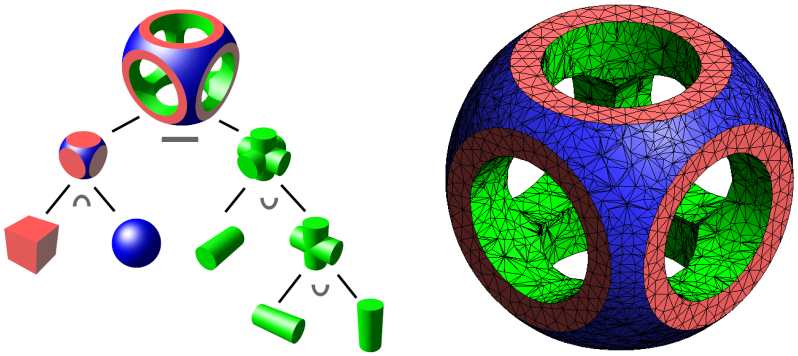


Figure 1.12 – 3D example of constructive solid geometry (left from Wikipedia and right reproduction).

```
1  g = GModel()
2  g.load("cube.msh")
3
4  ls1 = gLevelsetSphere(0.5,0.5,0.5,0.65,1)
5  ls2 = gLevelsetGenCylinder([0.5,0.5,0.5],[1,0,0],0.3,1)
6  ls3 = gLevelsetGenCylinder([0.5,0.5,0.5],[0,1,0],0.3,2)
7  ls4 = gLevelsetGenCylinder([0.5,0.5,0.5],[0,0,1],0.3,3)
8  ls5 = gLevelsetCut([ls1,ls2,ls3,ls4])
9
10 g2 = g.buildCutGModel(ls5)
11 g2.writeMSH("meshCut.msh")
```

Listing 1.3 – Pyhton code to create the wiki solid.

1.5 Mesh cutting along the iso-zero of a levelset

When dealing with finite element method and a levelset function describing the geometry, a mesh is created around the geometry, not necessarily conformed to its interface. The iso-zero of the levelset crosses some elements. The integration over these elements becomes then complicated. The extended finite element method can be used to solve the problem. The integration inside the crossed elements is executed differently on each side of the iso-zero. These elements have

to be cut into parts separated by the boundary of the domain.

To simplify the computation of the levelset inside the mesh, the levelset is first computed and stored at each node. To obtain the levelset value inside an element, the levelset function is interpolated with the values at the nodes. This enables to compute only once at the nodes the levelset function that can be time consuming.

Different cases can occur depending on the pattern of the cut. For linear triangular elements crossed by the iso-zero of a levelset, only two patterns can be obtained. The iso-zero can cross 2 edges or only one edge and the opposite vertex. The element is then cut into two parts. For the first pattern, the triangle is cut into a triangle and a quadrangle. The integration in the quadrangle has been chosen to be executed in two sub-triangles. The triangular element is then cut into 3 sub-triangles. For the second pattern, the triangular element is cut into two sub-triangles, as can be seen in figure 1.13.

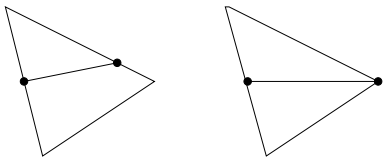


Figure 1.13 – The two different patterns for a levelset crossing a linear triangle.

An issue with this method has been pointed out by Nicolas Moës in his seminal paper [40]. When the iso-zero of a levelset passes very close to a vertex, sub-elements can be very small compared to the element. This could lead to numerical instabilities. Therefore, the levelset value at a node is set to zero if it is smaller than a arbitrary predefined threshold. The real geometry is then slightly distorted as the interface is imposed to pass through the node, but this modification is not significant if the chosen threshold is small enough. Other solutions would be to move the vertex exactly on the iso-zero of the levelset to keep the real geometry or to move the vertex far from the iso-zero. This leads to well shaped sub-elements but we chose to avoid local mesh modifications because it can create invalid tangled elements with negative Jacobian.

For linear quadrangular elements, there are more patterns. The iso-zero of the levelset can cross two opposite edges, two adjacent edges, pass through two opposite nodes or by a node and a non adjacent edge. To simplify the cutting, the quadrangle is first cut into 2 triangles as can be seen on figure 1.14. Then these two triangles are handled as triangular elements cut with an interface. Quadrangles which are not crossed by the interface are kept undivided and the integration points are mapped from the reference quadrangle.

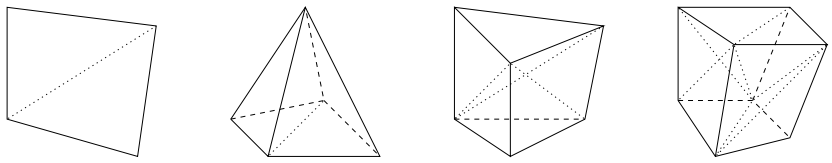


Figure 1.14 – The different patterns for the first cut of a quadrangle into 2 triangles, a pyramid into 2 tetrahedra, a prism into 3 tetrahedra and a hexahedron into 6 tetrahedra.

In 3 dimensions, for linear tetrahedral elements, there are four patterns depending on the number of edges crossed by the levelset, as can be seen on figure 1.15. The elements are cut into sub-tetrahedral parts in an equivalent manner described above for the triangles. For pyramids, prisms and hexahedra, these are first split into respectively 2, 3 and 6 tetrahedra, as can be seen on figure 1.14. These tetrahedra are then cut along the iso-zero of the levelset.

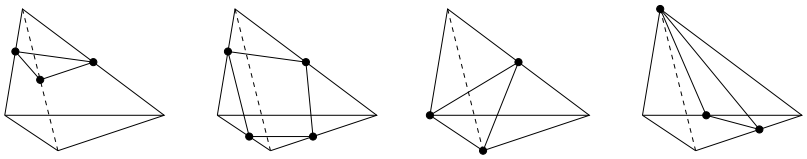


Figure 1.15 – The four different patterns for a levelset crossing a linear tetrahedron.

These sub-elements on each side of the interface are grouped into polygonal elements in 2D or polyhedral elements in 3D. These new elements keep a relation to the original element, called their parent. The interpolation in the polygonal elements is executed with the parent shape functions on the children interpolation points. Also, new border elements are created on the iso-zero, lines for 2D elements and surfaces for 3D elements. A relation is also kept with the parent element to interpolate on these interface elements.

When several primitive levelsets cross an element, the cutting is executed for each one, in the order of the reverse polish notation. The sub-elements are cut with the same technique. The resulting sub-elements are then classified inside or outside the domain to create the new polygonal/polyhedral elements. This has two advantages. It enables to tag the surfaces created with the different primitive levelsets as the tag of the cutting levelset is associated with the elements it creates. It also enables to obtain real corners inside the elements. If only the final levelset was taken into account, the elements crossed by several levelsets would crop the angles as the levelset inside the elements is computed

as the interpolation in these elements. That can be seen on figure 1.16.

These two advantages are shown on figure 1.17. It represents the surface of a cubic box created with a levelset function that is the combination of planes. A tetrahedral mesh has been created around the box and the tetrahedral elements have been cut with the levelset function. Sharp corners are observed and the different surfaces of the box are tagged differently. This enables for example to apply easily different boundary conditions on certain surfaces.

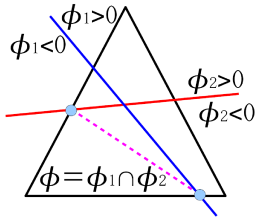


Figure 1.16 – Several levelsets cutting one element. Cropping the angle if the final levelset is computed.

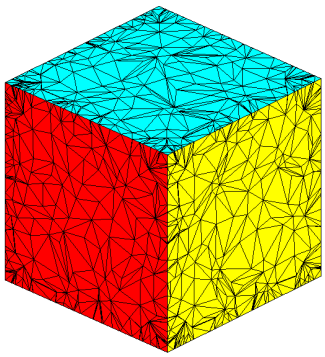


Figure 1.17 – Iso-zero of a box defined with six plane levelset functions with different tags on each surface.

Once the elements are cut, the integration points can be mapped on the subelements. Figure 1.18 shows a triangular element that is cut by a levelset. On the left is shown the element in the reference space and on the right the element in the real space. These triangles are cut into three subelements and the integration points for a second order Gauss integration are mapped in the subelements. To integrate on the interior of the domain, only the integration points in the subelements that belong to the interior of the domain are taken

into consideration.

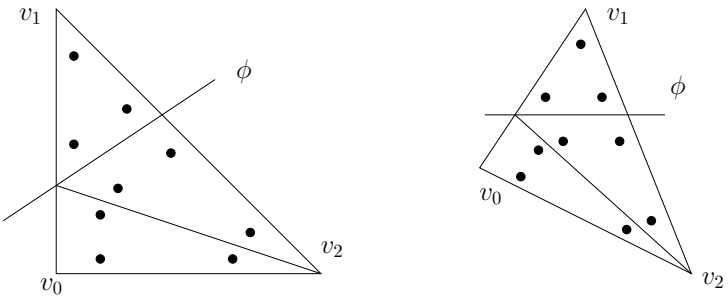


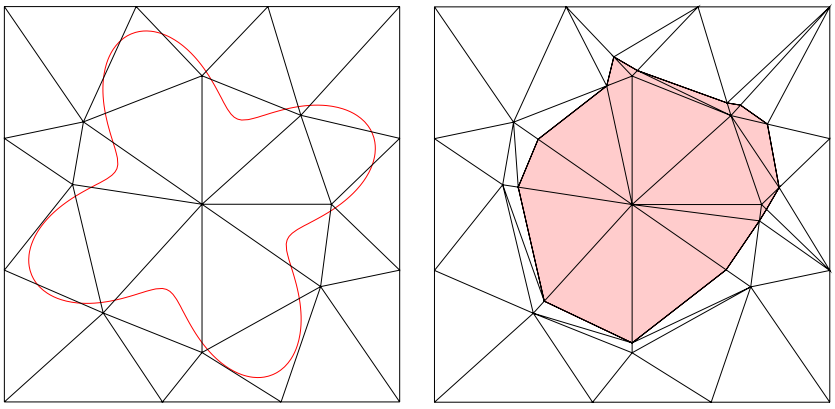
Figure 1.18 – Integration points in a triangle that is cut by a levelset (left : element in reference space, right : element in real space).

1.6 Recursive cut inside high order elements.

As the levelset is interpolated inside the elements, its representation is limited to the order of the elements. Inside linear elements, the iso-zero is reduced to lines in 2 dimensions and to planes in 3 dimensions. An integration error is then observed if the real levelset function is curved, as can be seen on figure 1.19(a). To reduce this error, higher order elements can be used to better represent the levelset function. The computation of the levelset then has to be executed at more points. To avoid curvilinear cuts inside the elements that could cause elements to collapse and result in negative Jacobians, a recursive cut is executed. The elements are cut into sub-elements along a predefined pattern. Triangular elements are cut into 4 sub-triangles and quadrangular elements are cut into 4 sub-quadrangles. The sub-elements can again be cut into smaller sub-elements following the same pattern. The order of recursion is chosen to be equal to the order of the mesh elements -1. Linear cuts are then performed into the smallest sub-elements.

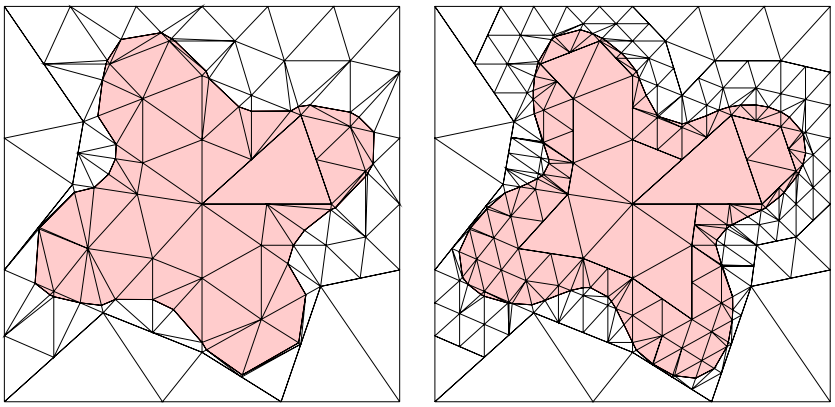
To capture cuts that occur twice in the edge of an element, with the same sign at its 2 nodes, every smallest sub-element has to be checked. To limit the number of sub-elements, intermediate-sized elements that are not crossed by the iso-zero are kept in one piece. To limit the computation, elements far from the iso-zero, with big levelset values at their nodes compared to their size, are not cut into sub-elements.

Let us consider a triangular mesh cut with the curvilinear iso-zero of a levelset function $R_0 - r + \frac{R_0}{3} sin(4\theta)$. As can be seen on the figures 1.19, for the same mesh, the iso-zero is better represented with higher order elements. For



(a) Triangular mesh and curvilinear iso-zero.

(b) Cut along the iso-zero in first order elements.



(c) Cut along the iso-zero in second order elements with 1 order of recursion.

(d) Cut along the iso-zero in third order elements with 2 orders of recursion.

Figure 1.19 – Recursive cut into high order elements to reduce integration error.

this example, a relative error of 29% is observed for the length of the iso-zero in linear elements, while this error decreases to 3.1% in second order elements and to 0.75% in third order elements. The relative error on the surface decreases from 23% for the linear elements to 0.76% for the second order elements and to 0.68% for the third order elements. See figure 1.20. The relative length error decreases smoothly. While the relative surface error decreases more irregularly. This can be explained by the concave shape of the geometry. The representation of the geometry with elements of low order removes parts of the surface on outward curves but adds surface on inward curves.

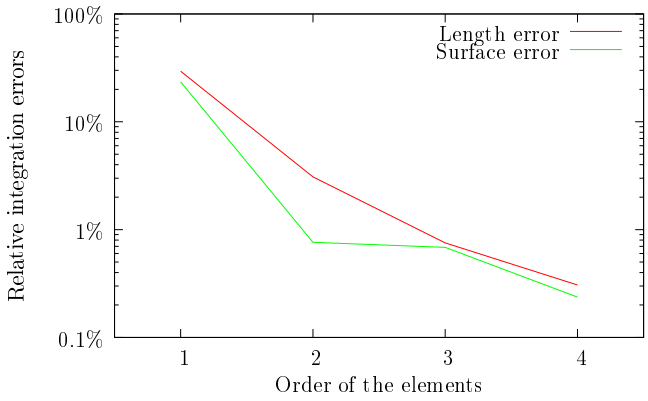


Figure 1.20 – Integration error on the surface and on the length for the recursive cut on elements of different orders.

1.7 Examples.

1.7.1 Analytical levelset : connecting rod

The first example is a connecting rod. Its levelset surface is composed only of analytical levelset functions : 8 planes and 4 cylinders. The two ends are hollow cylinders created with the intersection of two cylinder levelsets and two planes. The rod is made of the intersection of four planes and the two outer cylinders. The whole geometry is the union of the three parts.

A con rod levelset function can be created. With a few parameters as the radius of the cylinders, the length and width of the rod and the height of the ends, the function computes the equation of the cylinder and plane levelset functions and takes the union of their intersections.

An isotropic tetrahedral mesh has been created in a box around the geometry and the elements have been cut according to the method described before. The resulting surface can be seen on figure 1.21. Each surface has a different colour corresponding to its tag. This enables to apply boundary conditions on the interior surfaces of the cylinders in a later computation for example.

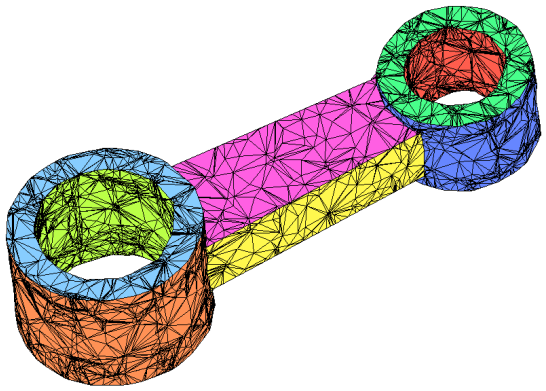


Figure 1.21 – Iso-zero of the levelset of a rod defined with several analytical levelset functions.

1.7.2 Distance to a triangulation : gear

The second example is a bevel gear. Its geometry has been found on the web in a STL format (see figure 1.22). STL format describes the surface geometry with a set of triangles. It is widely used for rapid prototyping and 3D printing. These triangles are however not aimed for a finite element computation because they have very bad aspect ratios. We can however use this triangulation to obtain a levelset function and use this levelset function to define the geometry for a finite element computation.

The STL triangulation is used to create the levelset function computed as the distance to the triangle mesh with the method described before. The problem with this triangulation is that a large number of triangles have a very low aspect ratio. The mean gamma aspect ratio (defined as the inscribed radius of a triangle divided by its circumscribed radius) is equal to 0.08. Therefore, the closest points do not necessarily belong to the closest triangle. The number of closest points returned by the ANN algorithm has been set to 100.

To limit the number of elements in the mesh of the bounding box surround-

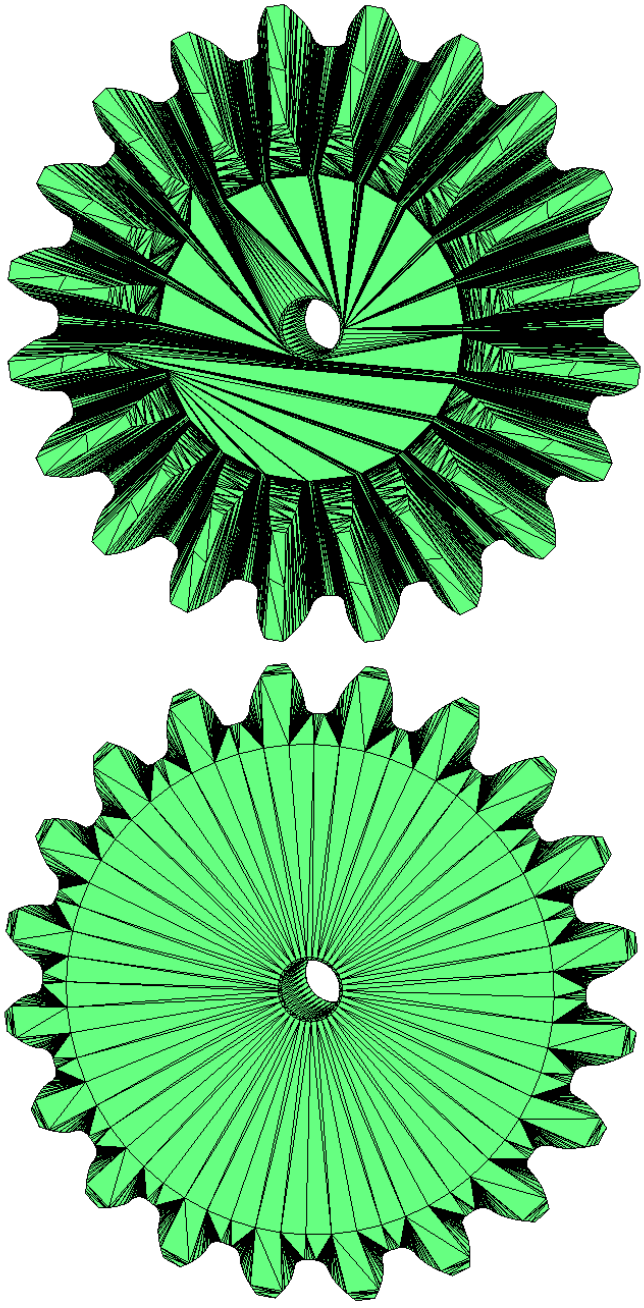


Figure 1.22 – Front and rear view of the STL triangulation of a bevel gear.

ing the gear, the mesh is created in two steps. First, an isotropic tetrahedral mesh is created in the bounding box and the levelset is computed at its nodes. This levelset is then used to create a second mesh of the bounding box where the size of the elements are specified by the levelset value, bounded by predefined minimal and maximal values. The elements close to the interface have a smaller size and the elements far from the interface have a larger size. The elements of the mesh are finally cut along the boundary of the geometry (see figure 1.23). As the levelset is unique, the surface has just one tag.

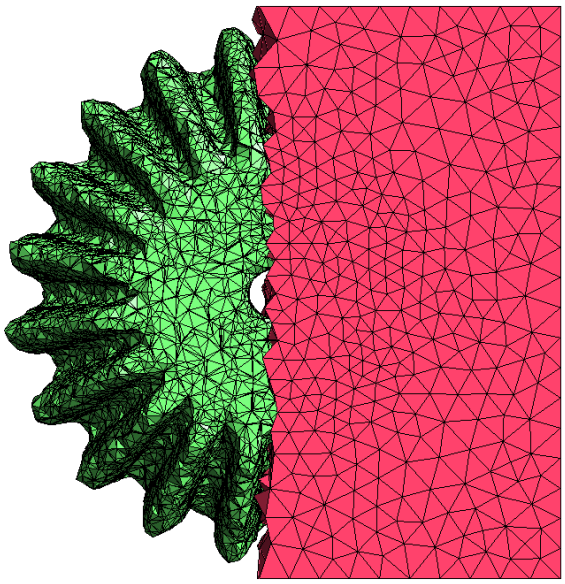


Figure 1.23 – Mesh of the bounding box (red) and boundary surface of the bevel gear (green).

1.7.3 Combination Operators : squirrel

The last example is the STL triangulation of the squirrel shown on figure 1.7 that is penetrated by four cylinders (see figure 1.24) and with a sphere appended on his tail. The cut operator is used to remove the cylinders from the squirrel geometry while the union operator is used to glue the sphere on the tail. Two cylinders intersect each other in the head of the animal (blue and green), one crosses the body (yellow) and one cuts the left foot (white).

This example shows that it is easy to modify a geometry afterwards without needing to remesh it. Some parts can be removed or added just by using operators with other levelsets that can be of any type. This is useful in shape

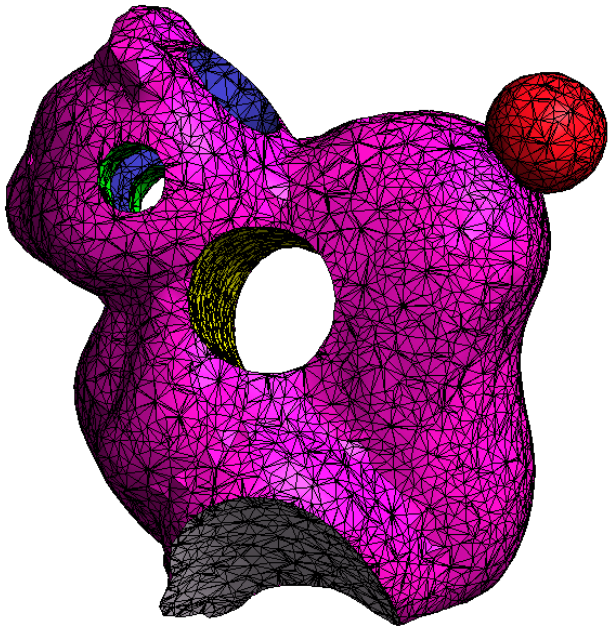


Figure 1.24 – STL mesh of a squirrel penetrated by cylinders and with a sphere appended.

optimization where medium is added to parts that need to be reinforced and removed where the stresses are low.

1.8 Conclusion

To conclude, we can say that we have obtained a tool that represents the geometry with the levelset method. The iso-zero of the levelset function represents the boundary of the geometry and its sign enables to determine the interior of the domain. This tool enables to use different types of levelset functions, analytical and distance to a triangulation being the most used. The distance to a triangulation has shown some difficulties to compute the sign of the levelset function. A method based on the normals of the triangulation is described.

To represent complex geometries, the tool also enables to combine several levelsets with boolean operators : unions, intersections or cut of a levelset by others. The domain can be described by the constructive solid geometry using several primitive levelset functions and operators.

These levelsets enable to cut a mesh surrounding the geometry along the boundary of the geometry. This makes it possible to integrate inside the elements crossed by the boundary.

Some improvements to represent better the geometry have been presented as the conservation of the primitive levelsets to avoid the crop in the elements where several levelsets cross each other and the recursive cut into higher order elements.

Examples are presented at the end of the chapter to illustrate the different possibilities that the tool can handle.

Chapter 2

Imposing Dirichlet boundary conditions on non conforming mesh

After having described the geometry with the levelset technique and enabled the integration inside finite elements cut by the boundary of the domain, we would like to perform a mechanical computation inside the domain. Therefore, we need a finite element formulation that solves elliptic partial differential equations on domains bounded by levelsets.

The standard finite element method does not allow to solve problems directly when the mesh is not conforming to the geometry. An alternative is to use the extended finite element method (X-FEM). It allows to tackle this problem by adding to the regular shape functions additional discontinuous shape functions by an enrichment procedure in the elements cut by the domain boundaries [42], [40]. The shape functions are then integrated separately in the different parts of the cut elements. If the embedded boundary is an external boundary, no enrichment is needed and the integration is simply executed in the part of the element belonging to the domain.

The X-FEM enables to solve mechanical problems with discontinuities where the classical finite element method fails because of meshing issues. A representative example is the crack propagation where the behaviour at the crack tip can be better represented with additional non linear shape functions and the discontinuity in the crack is modeled with Heaviside step functions.

2.1 Continuous formulation

We first consider the following boundary value problem. Let Ω be an open set of R^3 included in a triangular mesh $\mathcal{M}$ and $\Gamma = \Gamma_D + \Gamma_N$ its boundary of normal $\mathbf{n}$ composed of the two disjoints sets Γ_D and Γ_N. Let Γ_D be divided into a part conforming to the mesh Γ_D^C and a part non conforming to the mesh Γ_D^{NC} (see figure 2.1). We aim at finding u solution of the following Laplace problem :

$$\nabla \cdot k\nabla u = 0 \quad \text{in} \quad \Omega \tag{2.1}$$
$$u = \bar{u} \quad \text{on} \quad \Gamma_D \tag{2.2}$$
$$\mathbf{n} \cdot (k\nabla u) = \bar{f} \quad \text{on} \quad \Gamma_N \tag{2.3}$$

with k a positive constant that can be the thermal diffusivity if the problem represents a heat transfer, $\bar{u}$ the imposed solution on Γ_D and $\bar{f}$ the imposed normal gradient of the solution on Γ_N.

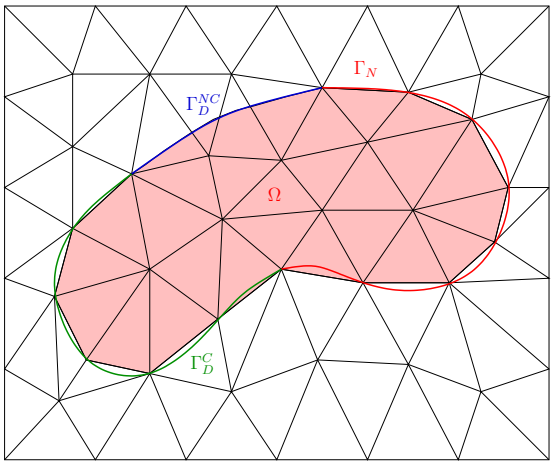


Figure 2.1 – Domain Ω included in a triangular mesh with boundary $\Gamma = \Gamma_N + \Gamma_D$ where Γ_D is composed of a conforming part Γ_D^C and a non conforming part Γ_D^{NC}.

2.2 Variational statements

Let us define subspaces of $H^1(\Omega)$, the Sobolev space of square integrable functions with square integrable first derivatives defined on Ω :

$$\mathcal{V}(\Omega) = \{u \in H^1(\Omega); u = \bar{u} \text{ on } \Gamma_D\}$$
$$\mathcal{V}_0(\Omega) = \{u \in H^1(\Omega); u = 0 \text{ on } \Gamma_D\}$$

A variational equivalent formulation of (2.1), (2.2) and (2.3) can be stated as : find $u \in \mathcal{V}(\Omega)$ such that :

$$\int_{\Omega} k(\nabla w) \cdot (\nabla u) d\Omega - \int_{\Gamma_D} k(\nabla u \cdot \mathbf{n}) w d\Gamma = \int_{\Gamma_N} \bar{f} w d\Gamma \tag{2.4}$$

for all $w \in \mathcal{V}_0(\Omega)$.

Another form of the variational formulation consists in minimizing the energy by stationarizing the following formulation, with $u \in \mathcal{V}(\Omega)$:

$$J_0 = \frac{1}{2} \int_{\Omega} k(\nabla u)^2 d\Omega - \int_{\Gamma_N} \bar{f} u d\Gamma \tag{2.5}$$

Defining such a space $\mathcal{V}(\Omega)$ allows to impose Dirichlet boundary conditions (2.2) "from the interior" i.e. *a priori*. In the standard finite element framework, it is easy to build a subspace of $\mathcal{V}(\Omega)$ based on a finite element mesh that is conforming to Γ_D when $\Gamma_D = \Gamma_D^C$. When Γ_D or a part of it is embedded in the mesh (i.e. $\Gamma_D^{NC} \neq \emptyset$), the imposition of Dirichlet boundary conditions becomes more complex.

The imposition of Neumann boundary conditions on Γ_N does not suffer this problem as the conditions are imposed weakly in the formulation and do not impose requirement on the space of the solution.

2.2.1 Existing methods to weakly impose Dirichlet boundary conditions on embedded interfaces

Different approaches have been developed to weakly impose Dirichlet boundary conditions on embedded interfaces, as penalty methods, Nitsche's methods or Lagrange multipliers methods. Lagrange multipliers methods encounter stability problems as it over-constraints the system [34]. To tackle this stability problem, two solutions exist. The first one consists in using a stable Lagrange multipliers space that enables to connect the two sides of the embedded discontinuity without over-constraining the system [41]. To build this stable space, some redundant Lagrange multipliers have to be removed from the space. The second one consists in stabilizing the formulation by adding stabilizing terms. Among stabilized methods, one can find methods that are consistent with the original formulation and others that violate consistency.

Penalty methods

The penalty methods are general concepts for the implementation of constraints in a variational problem [17]. In our formulation, it consists in adding

a surface integral to force the imposition of the boundary condition. The variational formulation (2.4) becomes :

$$\int_\Omega k(\nabla w)\cdot(\nabla u)d\Omega - \int_{\Gamma_D} k(\nabla u\cdot\mathbf{n})w d\Gamma = \int_{\Gamma_N}\bar{f}w d\Gamma + p\int_{\Gamma_D}(u-\bar{u})d\Gamma \quad (2.6)$$

with p a large positive parameter having the physical interpretation of a stiff spring constant. The additional term on the right hand side is an approximation of the constraining force to impose the boundary condition. The larger the value of p, the better the approximation. The constraint is satisfied as p tends to $+\infty$, corresponding to the spring connection becoming rigid.

The penalty term changes the properties of the functional and some problems of existence and unicity raise. Furthermore, the weak formulation is not consistent, in the sense that solutions to the weak formulation are not solutions to the original boundary value problem. This leads to a lack of optimality in the order of convergence [25]. This method is simple to implement but its main drawback is the difficulty to find an optimal value for the penalty parameter p due to to the lack of consistency of the formulation.

Nitsche methods

Nitsche presented in 1971 a stabilized method to account for Dirichlet boundary conditions in a weak sense without the use of Lagrange multipliers [43]. It transforms a Dirichlet boundary condition into a weak term similar to a Neumann boundary condition. However, the implementation of Nitsche's method requires an approximation of the corresponding Neumann term.

The variational form of equation (2.4) can be stated as : find $u \in H^1(\Omega)$ such that :

$$\begin{aligned}\int_\Omega k(\nabla w)\cdot(\nabla u)d\Omega - \int_{\Gamma_D} k(\nabla u\cdot\mathbf{n})w d\Gamma - \int_{\Gamma_D} u(\nabla w\cdot\mathbf{n})d\Gamma + \alpha\int_{\Gamma_D} wu\ d\Gamma \\ = \int_{\Gamma_N}\bar{f}w d\Gamma - \int_{\Gamma_D}\bar{u}(\nabla w\cdot\mathbf{n})d\Gamma + \alpha\int_{\Gamma_D} w\bar{u}\ d\Gamma \quad (2.7)\end{aligned}$$

for all $w \in H^1(\Omega)$, with α a scalar stabilizing coefficient.

As $u = \bar{u}$ on Γ_D, the formulation is consistent for any choice of the stabilization parameter. The surface integrals on Γ_D with the normal derivative on w are added to the formulation for the symmetry and ensures the property of adjoint consistency. The terms with the stabilizing coefficient α ensure a weak imposition of the Dirichlet boundary condition and are necessary to guarantee stability.

The stabilizing coefficient α depends on the mesh size. Nitsche proved that if *alpha* is taken as β/h, where h is the element size and β is a sufficiently large constant, then the discrete solution converges to the exact solution with optimal order in H^1 and L^2 [4]. The stability parameter can be identified by seeking the minimum value that guarantees coercivity of the bilinear form.

Practically, the stabilizing parameter is evaluated by solving a generalized eigenvalue problem at the Dirichlet boundary [22]. It can be estimated by solving a single global eigenvalue problem on the elements intersected by the Dirichlet boundary, yielding to a constant stabilization parameter along the entire Dirichlet boundary. Or it can be evaluated by solving multiple local small eigenvalue problems associated with every element intersected by the Dirichlet boundary, leading to piecewise-constant stabilization parameters along the boundary elements. With the local evaluation, acute increases in the magnitude of the stabilizing parameter can occur if the stabilizing parameter is evaluated on a non conforming element where the physical domain intersects only a small fraction of the boundary element. This can cause ill-conditioning of the global stiffness matrix.

A variant to Nitsche method is obtained by reversing the sign of the terms with surface integrals on Γ_D with the normal derivative on w. The resulting formulation is no longer symmetric but has a favorable coercivity property no matter the value of α. This method does not enjoy the adjoint consistency property. That will affect its L^2 convergence properties [4].

Nitsche type methods are attractive as they are simple to implement and benefit from the consistency of their formulation. Their drawbacks are the determination of the stabilization parameter, which is not an efficient strategy if required at every time step, and the difficulty to extend to other stiff boundary conditions as contact [60].

Lagrange multipliers

The Lagrange multipliers method is a general method to solve constrained minimization problems [51]. It allows to impose Dirichlet boundary conditions by adding new variables to obtain a mixed method described by the following saddle point problem :

$$\min_{u\in H^1(\Omega)} \max_{\lambda\in L^2(\Gamma_D)} \frac{1}{2}\int_\Omega k(\nabla u)^2 d\Omega - \int_{\Gamma_N} \bar{f} u d\Gamma + \int_{\Gamma_D} \lambda(u-\bar{u}) d\Gamma \qquad (2.8)$$

with λ is a field of Lagrange multipliers that may, for example, be defined on Γ_D only. The Euler-Lagrange equations relative to the first variation of (2.8)

with respect to u gives

$$\nabla^2 u = 0 \quad \text{in} \quad \Omega \tag{2.9}$$
$$\mathbf{n} \cdot k\nabla u = \lambda \quad \text{on} \quad \Gamma_D \tag{2.10}$$
$$\mathbf{n} \cdot k\nabla u = \bar{f} \quad \text{on} \quad \Gamma_N. \tag{2.11}$$

We can see that the Dirichlet-type boundary condition on Γ_D has been replaced by a Neumann-type boundary condition.

The annulation of the first variation of (2.8) with respect to λ gives

$$\int_{\Gamma_D} (u - \bar{u})\ \delta\lambda\ d\Gamma = 0 \quad \forall \delta\lambda \in L^2(\Gamma_D). \tag{2.12}$$

Equation (2.12) states that u is equal to $\bar{u}$ in the least square sense. In the continuous world, equation (2.12) imposes the Dirichlet boundary condition in the strong sense. Yet, in the finite element framework, with finite dimensional spaces, it is well known that a wrong choice for the Lagrange multipliers space leads to stability issues [34]. A naive approach to build the Lagrange multiplier space would be to take a linear continuous function along the embedded boundary. Yet this choice produces locking and leads to a suboptimal rate of convergence denoting an over-constrained problem [21].

To make the proper choice for the Lagrange multipliers space, the so-called LBB condition (or inf-sup condition) has to be verified [5], [8]. This condition is usually not possible to prove analytically. Chapelle and Bathe designed a robust numerical test that can be used instead [16]. It amounts to solve eigenvalue problems over meshes of increasing density.

A simple example to understand the over-constraint can be seen on figure 2.2. A meshed domain Ω is represented with a Dirichlet boundary condition applied on the boundary Γ_D not matching the mesh. The naive approach applies the constraints on the intersections between the boundary and the edges of the elements (9 constraints). If the geometry is slightly deformed or the mesh undergoes some modification so that the gap between Γ_D and the bottom nodes vanishes, the geometry will become conformed to the mesh and the number of constraints will decrease suddenly to 5, which is the right number.

This simple example shows that a naive construction of the Lagrange multipliers space is too rich and leads to an over-constrained problem that yields oscillatory multipliers on the boundary. Two approaches can be used to prevent this problem : shrinking the naive Lagrange multiplier space to the correct size or stabilize the formulation to remove the oscillations.

The first attempt to stabilize the unstable formulation with the naive Lagrange multipliers has been proposed by Barbosa and Hughes [8, 56]. It consists

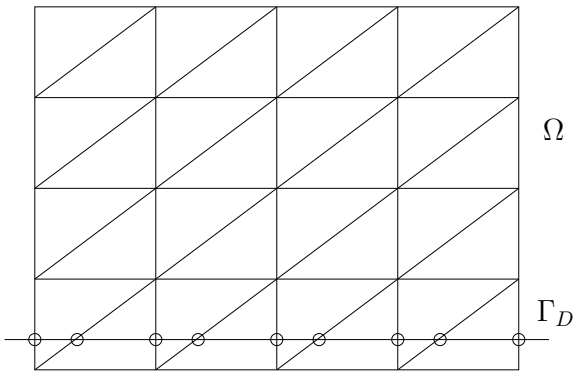


Figure 2.2 – Applying a Dirichlet boundary condition on a boundary embedded in the mesh.

in a least-square stabilization of the formulation. This enables to circumvent the inf-sup condition but adds a stabilizing parameter that has to be identified.

Winner nodes method

This method, presented by Moës and Béchet, consists in using Lagrange multipliers together with a stable Lagrange multipliers finite element space [41], [13]. The authors have developed an algorithm that selects a set of Lagrange multipliers to construct a stable space.

The Lagrange multipliers space is constructed in order to pass the numerical inf-sup test and to preserve reasonable rates of convergence. It is build by selecting a set of *winner nodes*. The nodes located on the boundary are first selected. Then the nodes of the edges cut by the boundary are investigated. The nodes belonging to more cut edges and close to the boundary are selected in priority, until every cut edge owns at least one winner node. See figure 2.3 for an example of winner nodes in a triangular mesh for an embedded circular boundary.

A piecewise-linear Lagrange Multiplier space is then built using the winner nodes. Lagrange multipliers are constant on some elements crossed by the interface that are connected to a single winner node, but this does not impact the global convergence of the method as the number of these cases remains low. An improvement has been later proposed. It uses the trace of the bulk shape functions on the interface to approximate the Lagrange multipliers. A global approach was proposed in [13]. A local approach followed in [29].

This method shows interesting results as no parameter is needed and its

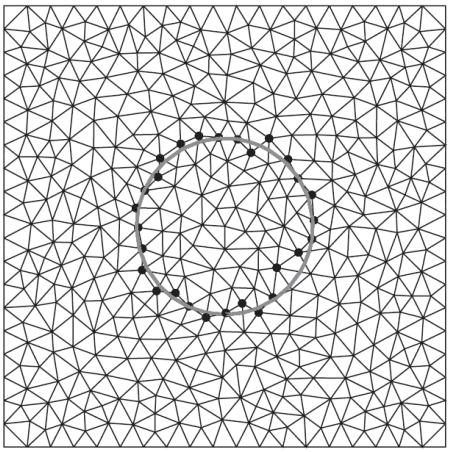


Figure 2.3 – Winner nodes for an embedded circular boundary (E. Béchet).

convergence has been demonstrated in [13]. But a generalization of the algorithm in dimension 3 or to high order elements is complex to implement.

Bubble-stabilized method

To stabilize the Lagrange multipliers, Dolbow et al. [39] developed a method using bubble functions. Bubble functions are high-order polynomial functions defined on the elements which vanish on their boundaries. The bulk field is decomposed into coarse and fine scales and a model is introduced for the fine-scale field with bubble functions. This allows to correct the lack of stability of the variational formulation of the problem. Using piecewise-constant Lagrange multipliers and piecewise-linear coarse-scale field leads to a form that is very similar to Nitsche's method with a stabilization parameter per element α^e that is obtained from the solution of the fine-scale problem in every element whose domain intersects the interface :

$$\int_\Omega k(\nabla w)\cdot(\nabla u)d\Omega - \int_{\Gamma_D} k(\nabla u\cdot\mathbf{n})w d\Gamma - \int_{\Gamma_D} u(\nabla w\cdot\mathbf{n})d\Gamma + \sum_e \alpha^e \int_{\Gamma_D} w\, d\Gamma \int_{\Gamma_D} u\, d\Gamma$$
$$= \int_{\Gamma_N} \bar{f} w d\Gamma - \int_{\Gamma_D} \bar{u}(\nabla w\cdot\mathbf{n})d\Gamma + \sum_e \alpha^e \int_{\Gamma_D} w\, d\Gamma \int_{\Gamma_D} \bar{u}\, d\Gamma \quad (2.13)$$

$$\alpha^e = \frac{\int_{\Omega^e} \nabla b^e \cdot \nabla b^e d\Omega}{(\int_{\Gamma^e} b^e d\Gamma)^2} \quad (2.14)$$

with b^e the bubble functions.

With appropriately chosen bubble functions, the equations lead to a symmetric positive-definite system. Yet error appears in elements crossed by an interface passing close to an edge. Residual-free bubble functions can be used to improve the stability and overall accuracy of the problem and simplify the computation of the stability parameter [20] :

$$\alpha^{e,rbf} = \frac{1}{\int_{\Gamma^e} b^e d\Gamma} \tag{2.15}$$

This method offers the advantage of computing automatically the stabilizing parameter but it comes with an increase in computational cost as a linear system has to be solved in every element crossed by the interface to determine the stabilization parameter.

2.2.2 Two stabilized formulations

We see that no existing solution is optimal. Stable formulations are not straightforward to extend to complex problems and stabilized formulations require to compute a stabilization parameter. We now present a new stabilized method using Lagrange multipliers that is not consistent with the original PDE. We compare it with an existing stabilization formulation that is consistent to analyse if it benefits from some advantages.

The new formulation we consider consists in a simple Laplace stabilization. Laplacian stabilization has been widely used in the context of the finite element resolution of the incompressible Navier-Stokes equations [11]. Consider the following functional :

$$J_1(u,\lambda) = \frac{1}{2}\int_\Omega k(\nabla u)^2 d\Omega - \int_{\Gamma_N} \bar{f} u d\Gamma + \int_{\Gamma_D} \lambda(u-\bar{u}) d\Gamma - \frac{1}{2}\int_{\Gamma_D} \tau(\nabla\lambda)^2 d\Gamma \tag{2.16}$$

with τ a stabilizing parameter.

Euler Lagrange equations of (2.16) are, in addition to (2.9), (2.10) and (2.11) :

$$u - \bar{u} = \nabla \cdot \tau \nabla\lambda \quad \text{on} \quad \Gamma_D \tag{2.17}$$

Let us call γ_D the boundary of Γ_D with its normal $\mathbf{t}$. Clearly, if Γ_D is closed, γ_D is empty. If not, Γ_D is connected to Γ_N i.e. γ_D is the intersection of Γ_D and Γ_N. Therefore we impose a Dirichlet boundary condition $\lambda = \bar{f}$ on γ_D.

If h is the mesh size and H is the problem size and if we assume a second order of convergence for u, i.e.

$$\|u - \bar{u}\|_2 = \mathcal{O}\left(h^2 \frac{\|u\|}{H^2}\right)$$

then (2.17) gives the following order of magnitude for τ

$$\tau = \tau_0 \frac{h^2 H}{k}$$

with τ_0 independent of h, H and k.

The second formulation that we consider consists in a least square stabilization. The method was first proposed by Hughes and Barbosa in [8]. It has then been used for imposing embedded Dirichlet boundary conditions in [28] and [12].

Consider the following functional

$$J_2(u,\lambda) = \frac{1}{2}\int_\Omega k\left|\nabla u\right|^2 d\Omega - \int_{\Gamma_N} \bar{f} u d\Gamma + \int_{\Gamma_D} \lambda(u-\bar{u}) d\Gamma - \frac{1}{2}\int_{\Gamma_D} \tau' \left|\lambda - k\partial_n u\right|^2 d\Gamma \tag{2.18}$$

with τ' a stabilizing parameter.

Euler Lagrange equations of (2.18) are, in addition to (2.9), (2.10) and (2.11),

$$u - \bar{u} = \tau'(\lambda - k\partial_n u) \quad \text{on} \quad \Gamma_D \tag{2.19}$$

We assume now that λ converges one order less than u i.e

$$\left\|\lambda - k\partial_n u\right\|_2 = \mathcal{O}\left(h\frac{k\|u\|}{H^2}\right)$$

Factor τ' has the following form

$$\tau' = \tau_0' \frac{h}{k}$$

with τ_0' independent of h, H and k.

2.3 Finite Element Formulation

To solve the problem with the finite element method, the continuous formulations have to be discretized. Assume the following finite expansions for u :

$$u(\mathbf{x}) \simeq u_h(\mathbf{x}) = \sum_{i\in I} u_i \phi_i(\mathbf{x}) \tag{2.20}$$

where I denotes the set of nodes of Ω_h which have some portion of their supports in Ω and where ϕ_i are the nodal shape functions associated to the nodes of I.

We then assume the following finite expansions for λ :

$$\lambda(\mathbf{x}) \simeq \lambda_h(\mathbf{x}) = \sum_{i \in B} \lambda_i \psi_i(\mathbf{x}) \tag{2.21}$$

where B denotes the set of nodes of $\Gamma_{D,h}$ and where ψ_i are the nodal shape functions associated to those nodes.

Thanks to expansions (2.20) and (2.21), a Galerkin formulation of (2.16) can be written as :

$$\sum_{i \in I} u_i \int_\Omega k \nabla \phi_i \cdot \nabla \phi_j d\Omega + \sum_{i \in B} \lambda_i \int_{\Gamma_D} \psi_i \phi_j d\Gamma = \int_{\Gamma_N} \bar{f} \phi_j d\Gamma \quad \forall j \in I \tag{2.22}$$

$$\sum_{i \in I} u_i \int_{\Gamma_D} \phi_i \psi_j d\Gamma - \sum_{i \in B} \lambda_i \int_{\Gamma_D} \tau \nabla \psi_i \cdot \nabla \psi_j d\Gamma = \int_{\Gamma_D} \bar{u} \psi_j d\Gamma \quad \forall j \in B \tag{2.23}$$

In matrix form, (2.22) and (2.23) can be written as

$$\begin{bmatrix} \mathbf{K} & \mathbf{C}^T \\ \mathbf{C} & -\mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{f}} \\ \bar{\mathbf{u}} \end{bmatrix} \tag{2.24}$$

where $\mathbf{K}$ is a matrix of size $\dim(I) \times \dim(I)$ of values

$$K_{ij} = \int_\Omega k \nabla \phi_i \cdot \nabla \phi_j d\Omega$$

$\mathbf{C}$ is a matrix of size $\dim(B) \times \dim(I)$ of values

$$C_{ij} = \int_{\Gamma_D} \phi_i \psi_j d\Gamma$$

and where $\mathbf{S}$ is a matrix of size $\dim(B) \times \dim(B)$ of values

$$S_{ij} = \int_{\Gamma_D} \tau \nabla \psi_i \cdot \nabla \psi_j d\Gamma$$

A Galerkin formulation of the second formulation (2.18) can be written as

$$\begin{aligned} \sum_{i \in I} u_i \left[\int_\Omega k \nabla \phi_i \cdot \nabla \phi_j d\Omega - \int_{\Gamma_D} \tau' k^2 \partial_n \phi_i \partial_n \phi_j d\Gamma \right] \\ + \sum_{i \in B} \lambda_i \left[\int_{\Gamma_D} \left(\psi_i \phi_j + \tau' k \psi_i \partial_n \phi_j \right) d\Gamma \right] \\ = \int_{\Gamma_N} \bar{f} \phi_j d\Gamma \quad \forall j \in I \end{aligned} \tag{2.25}$$

$$\sum_{i\in I} u_i \int_{\Gamma_D} (\phi_i\psi_j + \tau' k\partial_n\phi_i\psi_j)\, d\Gamma - \sum_{i\in B} \lambda_i \int_{\Gamma_D} \tau'\psi_i\psi_j d\Gamma = \int_{\Gamma_D} \bar{u}\psi_j d\Gamma \quad \forall j \in B \quad (2.26)$$

In matrix form, (2.25) and (2.26) can be written as

$$\begin{bmatrix} \mathbf{K}-\mathbf{A} & (\mathbf{C}+\mathbf{B})^T \\ (\mathbf{C}+\mathbf{B}) & -\mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{f}} \\ \bar{\mathbf{u}} \end{bmatrix} \quad (2.27)$$

where $\mathbf{A}$ is a matrix of size $\dim(I) \times \dim(I)$ of values

$$A_{ij} = \int_{\Gamma_D} \tau' k^2 \partial_n\phi_i \partial_n\phi_j d\Gamma$$

$\mathbf{B}$ is a matrix of size $\dim(B) \times \dim(I)$ of values

$$B_{ij} = \int_{\Gamma_D} \tau' k \partial_n\phi_i \psi_j d\Gamma$$

and where $\mathbf{M}$ is a matrix of size $\dim(B) \times \dim(B)$ of values

$$M_{ij} = \int_{\Gamma_D} \tau' \psi_i \psi_j d\Gamma$$

In this work, Γ is defined by means of the iso-zero of a levelset function. The boundary Γ intersects elements of the mesh and the aim of the procedure is to impose boundary conditions on Γ. As an example, consider the domain presented in figure 2.4. The domain of interest Ω bounded by two non conforming curves is coloured in green while the underlying uniform mesh of a surrounding square is in grey. Intersections of Γ with the underlying mesh are also represented as small spheres.

We now integrate numerically the formulation in the domain Ω and on its boundary $\Gamma_D + \Gamma_N$. In the elements crossed by the interface, the Gaussian integration points in the part belonging to Ω are obtained thanks to the cutting tool presented in chapter 1 that uses the levelset technique to represent the boundary.

To apply boundary conditions on the interface represented by the iso-zero of the levelset, the integration points on this surface are also needed. The interface is discretized with the triangular faces of the subtetrahedra with a zero levelset value on each vertex. The integration points are mapped from the reference triangle to these triangular faces.

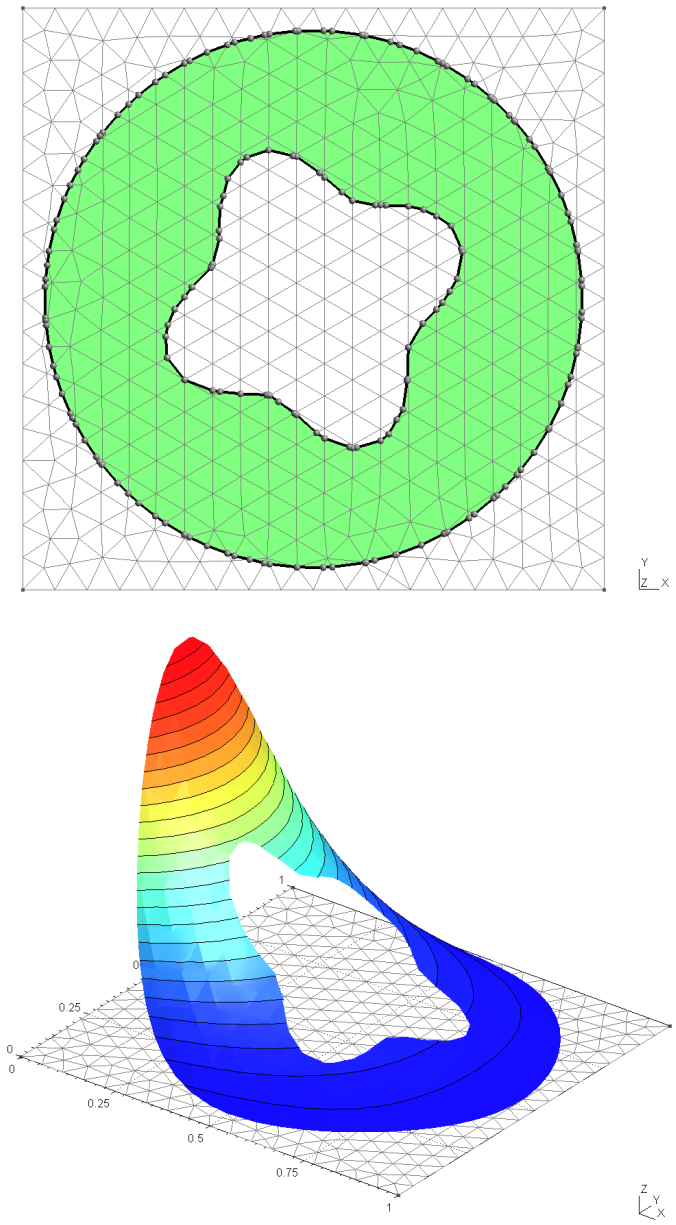


Figure 2.4 – Example of a domain (painted in green on the top picture) and the solution of a Laplace problem using embedded Dirichlet boundary conditions with Lagrange multipliers.

2.4 Implementation

The formulations have been implemented in the native solver of the `Gmsh` software. The solver is designed to solve different physical problems as heat transfer, elasticity, electromagnetism or fluid dynamics in a similar way. The code is written in C++ and a `Python` wrapper enables to write test cases easily.

```
1  def sol(x,y,z):
2   return sin (PI*y) * (cosh(PI*(-x+1.))-sinh(PI*(-x+1.))/tanh(PI))
3
4  ls1 = gLevelsetSphere (0.,0.,0.,0.9)
5  ls2 = gLevelsetMathEval("(.4-(x*x+y*y)^.5)+.4/5*sin(4*(atan(y/x)))")
6  li = [ls1, ls2]
7  ls = gLevelsetIntersection(li)
8
9  mySolver = laplaceSolver(1000)
10 mySolver.setMesh("mesh.msh")
11 mySolver.cutMesh(ls)
12 tau = 1.e-12
13 mySolver.setDomain(1000,1.)
14 mySolver.setLagrangeMultipliers(1, tau, 1, sol)
15 mySolver.setLagrangeMultipliers(405, tau, 2, sol)
16 mySolver.solve()
17 EU = mySolver.computeL2Norm(sol)
18 EU = mySolver.computeLagNorm(solLag)
```

Listing 2.1 – Pyhton code to solve a Laplace problem.

An example of a `Python` code to solve the Laplace problem described on figure 2.4 is presented in listing 2.1. First, the analytical solution of the problem (lines 1-2) and the levelset function representing the boundary of the domain (lines 4-7) are defined. Then the solver object is created (line 9). A triangular mesh of a box surrounding the domain is applied to the solver (line 10) and function spaces of the model dimension are created to represent the variables : a *scalarLagrangeFunctionSpace* to hold the variables of the solution field u and a *scalarLagrangeFunctionSpaceOfElement* for the Lagrange multipliers λ. These function spaces have predefined functions that return the shape functions and the gradient of the shape functions inside an element. The difference between the two classes is that the *scalarLagrangeFunctionSpaceOfElement* is adapted for border elements resulting from a cut, forcing to integrate inside the border elements and not inside the parent elements.

The mesh is then cut along the levelset function (line 11), creating several physical parts inside the mesh. The domain of computation is defined inside the appropriate parts of the cut mesh and the value of k is defined (line 13). Lagrange multipliers are added to the problem on the non conforming borders of the domain where Dirichlet boundary conditions are applied and the values for the stabilizing parameters are given (lines 14-15).

The problem is now well posed. The linear system can be initiated and assembled. First, the degrees of freedom of the problem are identified. The degrees of freedom that are fixed by Dirichlet boundary conditions on conforming bound-

aries are not added to the linear system. For our Laplace problem, the solution degrees of freedom are the nodes of the elements inside the domain or cut by the boundary and the Lagrange multipliers degrees of freedom are the nodes of the interfaces created by the cut. Then, the Neumann boundary conditions are added to the system, followed by the terms related to the Lagrange multipliers (cross terms, Laplace term and load term on the border) and finally the Laplace term related to the solution. The linear system is finally solved to obtain the values of the solution and of the Lagrange multipliers.

The linear system can be written in the form $Ax = b$ with A a sparse symmetric and positive definite matrix. To solve this system, there are two classes of methods : direct and iterative methods. A first direct method is the *Gaussian elimination* method. It uses a sequence of operations performed on the augmented matrix obtained by appending b to the right of A. Operations are swapping rows, multiplying rows by a scalar or adding rows. The goal is to obtain an upper triangular matrix with ones on the diagonal. The solution x can then be computed by backward substitution in each equation. It is a robust method but it requires to store the whole system in memory, what can be problematic for large problems. Another drawback is that this method can cause numerical round-off errors if pivots are too small. More rows swaps are necessary to avoid this problem.

The *LU factorization* decomposes matrix A into the product of a lower triangular matrix L and an upper triangular matrix U. The solution of the system x is obtained in two steps : a forward substitution to solve $Lz = b$ and then a backward substitution to solve $Ux = z$. The matrices L and U are obtained by the same procedure as for Gaussian elimination, with a similar number of operations. The advantage of this method is that the triangular matrices can be reused to solve the system with different right hand sides b, while the Gaussian elimination requires to restart from scratch.

A third direct method is the *Cholesky decomposition*. It consists in factorizing matrix A into the form LDL^T with D a diagonal matrix. For a symmetric and positive definite matrix, the number of operations is less than for a Gaussian elimination, Cholesky decomposition being twice as efficient.

Actually, matrices from finite element problems are often banded, with non-zero entries confined to a diagonal band. Applying a direct solver (LU decomposition) to such a matrix results in the band being filled in by many non-zero elements. However if the graph of the matrix is a planar graph (a graph that you can draw on a 2d plane), a well known result by Tarjan and Rose [50] shows that the complexity is $\mathcal{O}(N^{3/2})$ instead of $\mathcal{O}(N^3)$ for general algorithms. 2D finite element problems using linear elements are good candidate for direct methods. In 3D, the complexity grows to N^2 and direct solvers are way too expensive : iterative solvers have to be used for large 3D problems. Yet, they are less robust.

Iterative methods start from a first approximation of the solution. Several steps and search directions are computed to bring it closer to the solution. The advantage of these methods is to avoid to store matrix A in memory. Local residuals can be computed and assembled to obtain the global residual. A large amount of iterative methods exist.

The *conjugate gradient* method is quite efficient but can only solve linear systems whose matrix is symmetric and positive definite. Matrix A is used to define a scalar product. The search directions are chosen to be conjugate with respect to this scalar product. At each step, an optimal step is chosen and the new search direction is taken as a combination of the last one and the residual. The maximal number of iterations is proportional to the square root of the condition number of matrix A. To decrease the number of iterations, a preconditioner can be used to reduce the condition number of the system matrix. Yet the preconditioning should not be costly. Incomplete factorization enables to obtain an approximate factorization of the matrix and reduce the condition number.

A second iterative method, the *Generalized Minimal Residual* method (GMRES), is more general than the conjugate gradient method as it can solve non-symmetric systems of linear equations. The method approximates the solution by the vector in a Krylov subspace with minimal residual.

Several linear solvers are available in the `Gmsh` code to solve the system $Ax = b$:

- A first solver based on *fullVector* and *fullMatrix* classes of `Gmsh` uses a direct method. The values of the linear system are stored in a table of the dimension of the system. To solve the system, the `Lapack` library is used to perform a LU decomposition with partial pivoting and row interchanges. It factors A as $A = P * L * U$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. This solver is not optimized for linear systems of large dimensions.

- A second available solver is provided by the `Gmm` library that is suited for sparse, dense and skyline matrices. The values are stored in a sparse row matrix. To solve the system, an incomplete $LDLT$ preconditioner is first used. It gives an approximation of the decomposition of the matrix A into the product $A = L * D * L^t$ where L is a lower triangular matrix with ones on its diagonal and D is a diagonal matrix. Then the system is solved with the iterative method of the conjugate gradient.

- Another solver uses the `PETSc` library (*Portable Extensible Toolkit for Scientific Computation*). It uses an incomplete LU factorisation as precondi-

tioner and a GMRES method to solve the system.

- A last solver is provided by the `Taucs` library. It uses no preconditioner and a *LLT* factorisation (similar to the *LDLT* factorisation) to solve the system.

Direct methods are preferable for their robustness but require to store matrix A in memory, what can be not possible for large systems. For symmetric positive definite problems of moderate size, as linear elliptic problems in 2 dimensions, a Cholesky decomposition is recommended. If the system is too large, a conjugate gradient with an incomplete LU decomposition as preconditioner will be used. In the following problems, the direct solver using the `Lapack` library will be used for small 2D problems. When the number of degrees of freedom becomes too large, the solver provided by the `Gmm` library using an incomplete *LDLT* preconditioner and the conjugate gradient method is used.

To analyse the results, the solution and the Lagrange multipliers can be plotted on the mesh. The error can also be computed by evaluating the L^2-norm of the difference between the analytical solution and the computed solution :

$$\|e_u\| = \int_\Omega (u - u_h)^2 d\Omega$$

The error on the Lagrange multipliers can also be computed if the analytical solution of the normal derivative of the solution is available :

$$\|e_\lambda\| = \int_{\Gamma_D} (\lambda - \lambda_h)^2 d\Gamma$$

2.5 Stabilizing parameter

The remaining problem with the proposed methods is to determine the stabilizing parameter τ. It has been linked to an adimensional parameter $\tau_0 = \tau \frac{k}{h^2 H}$ for the first formulation and $\tau_0' = \tau' \frac{k}{h}$ for the second one. This relation has to be validated and the value of τ_0 and τ_0' have to be determined.

We can first analyse the evolution of the discretization error for the two formulations with respect to the value of the stabilizing parameter for different meshes.

Let us consider the Laplace problem described on figure 2.4. A domain is defined between two concentric curves. It is applied on a surrounded squared mesh not conforming to the boundary of the domain. A Laplace problem is computed inside the domain by imposing the solution as Dirichlet boundary conditions on the two curves :

$$u(x,y) = \sin(\pi y)\left(\cosh(\pi x) - \frac{\sinh(\pi x)}{\tanh(\pi)}\right) \tag{2.28}$$

The L^2-norm of the discretization error can be seen on figures 2.5 and 2.6 for the first and second formulation respectively, for different mesh sizes. The L^2-norm of the error on the Lagrange multipliers is shown on figures 2.7 and 2.8.

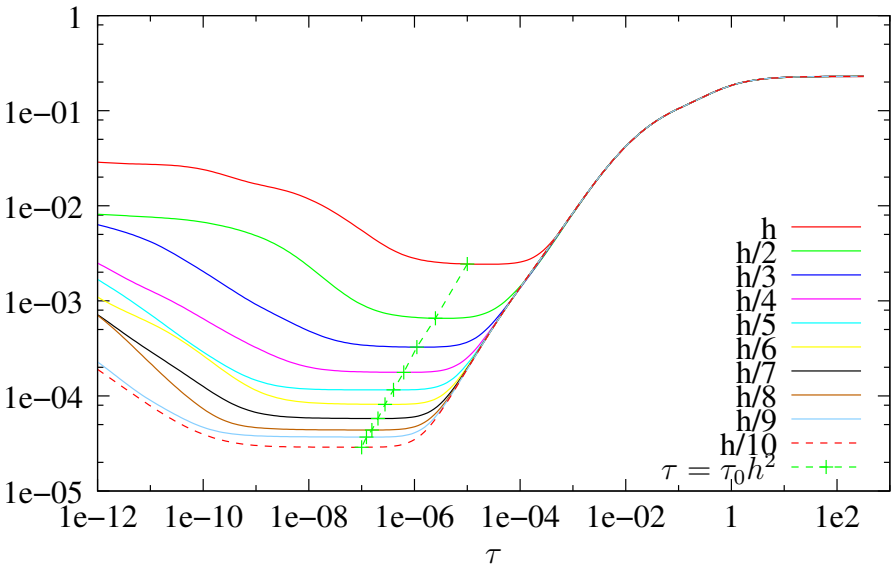


Figure 2.5 – L^2-norm of the discretization error as a function of τ for the first formulation.

As can be seen on figure 2.5, the L^2-norm of the error on the solution for the first formulation has a minimum value on a large interval. The error increases if the value of τ becomes too small. In this case, the stabilizing term of the stabilized formulation becomes insignificant compared to the other terms and does not stabilizes the formulation. If the value of τ becomes too large, the error also increases. In this case, the stabilizing term acts as a penalty and the other terms become insignificant. As the meshes are refined, the discretization error decreases, as expected, and the interval with minimal error is shifted to the left with smaller values of τ. On the right of the interval, with a large value of τ, the error is the same for all the meshes as the formulation is reduced to the stabilized term that does not depend on the volume mesh. If we take a value of τ located at the center of the interval of the coarse mesh, we can compute a value τ_0 and determine a value of τ for the refined meshes. We can see that the computed values are all located in the minimum error interval. That confirms the relation between τ and τ_0.

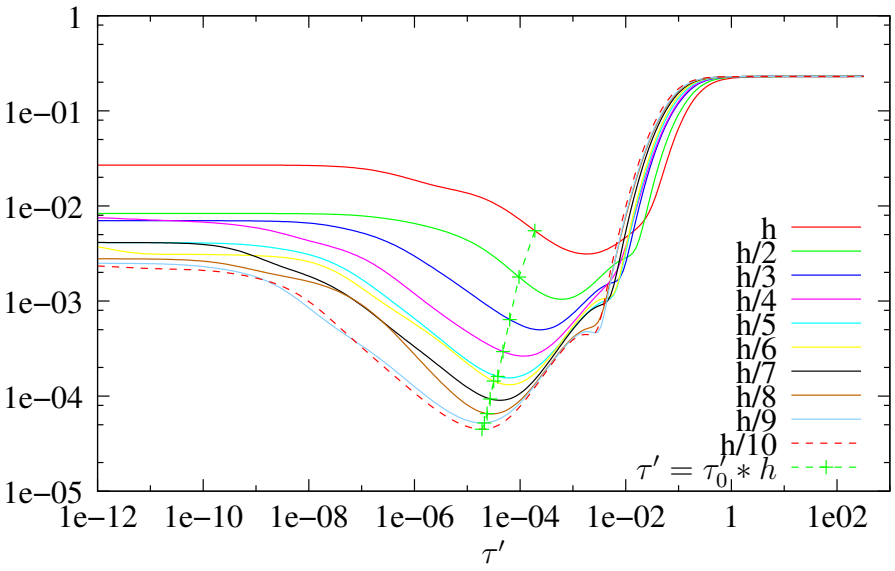


Figure 2.6 – L^2-norm of the discretization error as a function of τ' for the second formulation.

On figure 2.6, for the second formulation, the same phenomenon occurs but the interval with minimal value of the error is smaller. The dimensional analysis found that $\tau' = \tau'_0 \frac{h}{k}$. But if we take a value of τ' with the minimum error value, the value of τ'_0 does not give minimum values of the error for the other meshes. This second formulation looks less interesting.

Figure 2.7 shows the L^2-norm of the error on the Lagrange multipliers. As for the discretization error on the solution of the Laplace problem, it reaches a minimum value. For smaller values of τ than the minimum, the error increases continuously as τ decreases. For larger values of τ than the minimum, the error increases to a bounded value as τ increases. There is no large interval with minimal error as for the discretization error. When the mesh is refined, the minimal error decreases and is obtained for a smaller value of τ. Comparing the errors on the solution of the Laplace problem and on the Lagrange multipliers (figures 2.7 and 2.5) shows that the minimum error for the Lagrange multipliers is obtained close to the upper bound of the interval of the minimal discretization error. A good choice for τ seems to be around this value.

On figure 2.8, for the second formulation, the same phenomenon occurs. But the error decreases less as the mesh is refined. The second formulation is again less attractive.

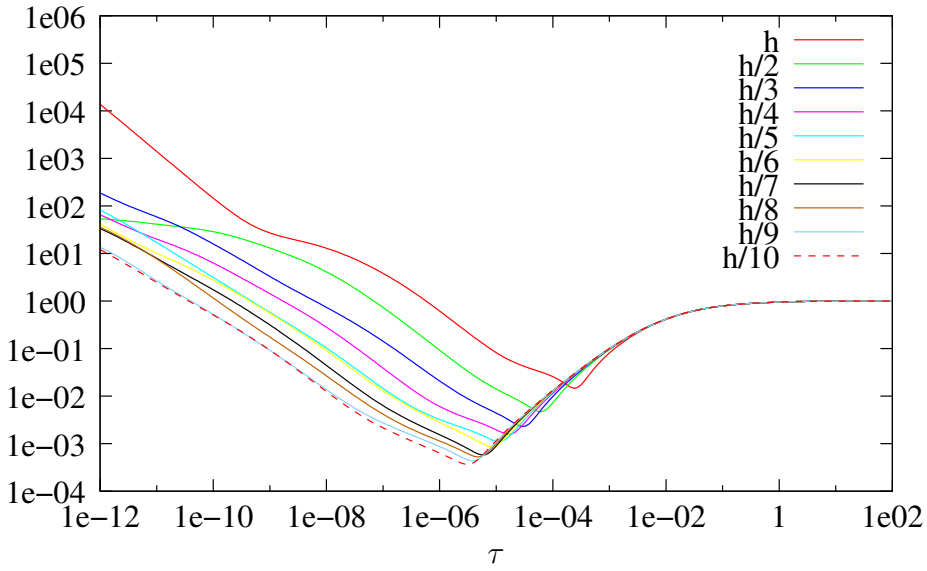


Figure 2.7 – L^2-norm of the error on the Lagrange Multiplier as a function of τ for the first formulation.

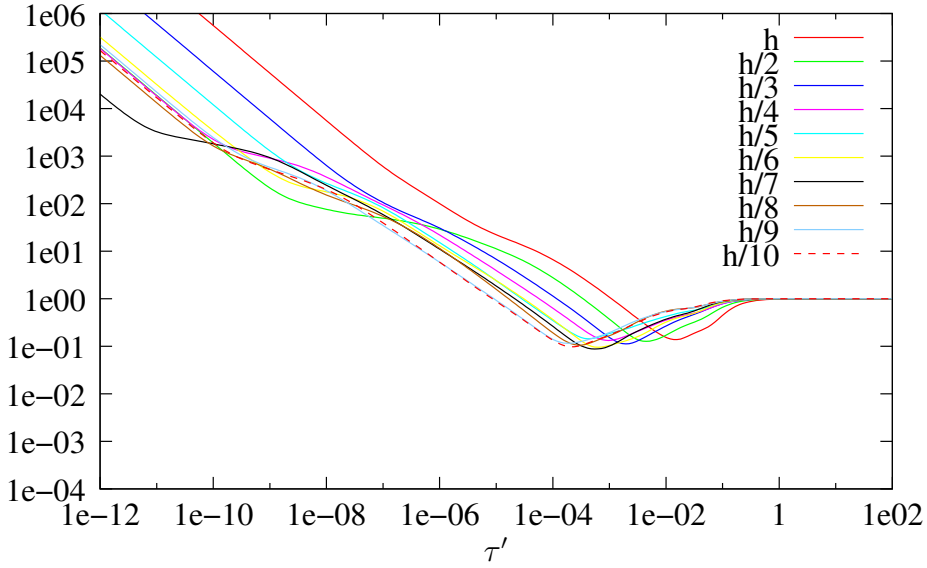


Figure 2.8 – L^2-norm of the error on the Lagrange Multiplier as a function of τ' for the second formulation.

To illustrate the stability problem, the Lagrange multipliers along the external border of the Laplace problem solved with the first formulation are plotted on figure 2.9 for three values of τ on the coarse mesh (mesh size h on curves 2.7). If the stabilizing parameter is too small ($\tau = 10^{-6}$), oscillations with small wavelength are present around the curve. When the stabilizing parameter becomes too large ($\tau = 10^{-2}$), curves with a wavelength equivalent to the problem size are also stabilized. The curve of the Lagrange multipliers is then flattened. For even larger values of τ, the Lagrange multipliers tend to be constant.

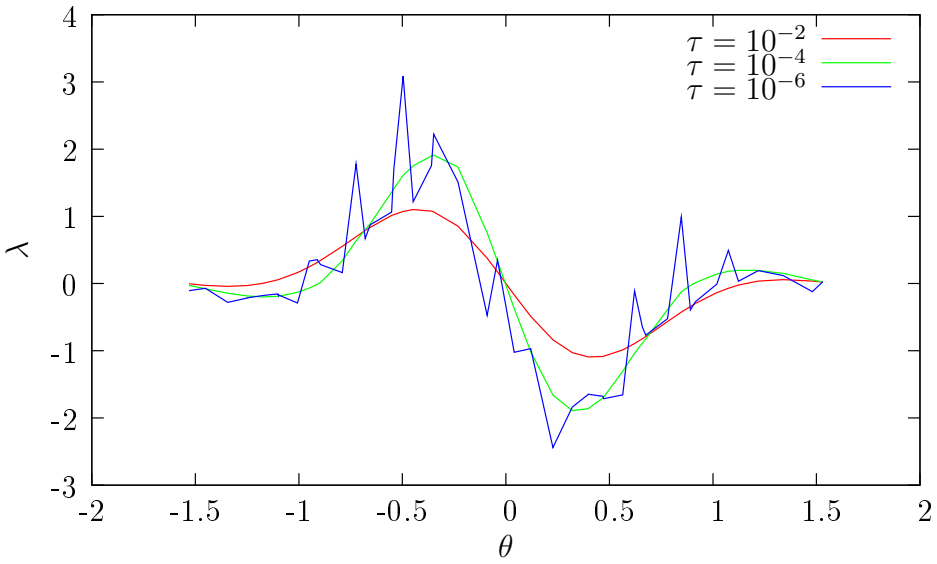


Figure 2.9 – Lagrange multipliers along the external embedded boundary of the Laplace problem for different values of the stabilizing parameter τ.

The rate of convergence of the methods can be analysed. Figure 2.10 shows the rate of convergence of the discretization error and of the error on the Lagrange Multipliers for both formulations. The values for the stabilizing parameters are taken as $\tau_0 = 0.04$ and $\tau_0' = 0.02$. The mean rates of convergence are computed and shown in table 2.1. The first formulation gives a rate of convergence close to optimal while the second formulation has a rate of convergence much worse, particularly for the Lagrange multipliers. This was predictable as the curves of the error on the Lagrange multipliers on figure 2.8 do not show a significant decrease of the minimum error when the mesh is refined. Again, the first formulation is advantageous regarding its rate of convergence.

To validate the value of the adimensional parameter for the first formulation

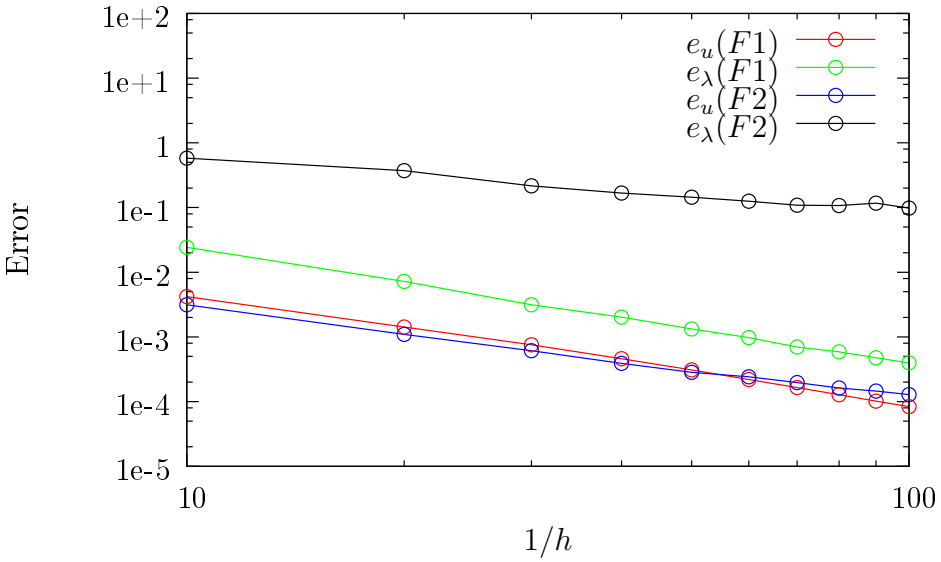


Figure 2.10 – Discretization error e_u and error on the Lagrange multipliers e_λ for the two presented formulations depending on the mesh size h.

Error	Rate of convergence
e_u (F_1)	-1.71
e_λ (F_1)	-1.79
e_u (F_2)	-1.38
e_λ (F_2)	-0.79

Table 2.1 – Mean rates of convergence for the discretization error and the error on the Lagrange multipliers for the two presented formulations.

$\tau_0 = \tau\frac{k}{h^2 H}$, different computations have been executed on two different geometries. The first one has already been described on figure 2.4. The second one is a linear cut in a rectangular domain (see figure 2.11). The mesh of a rectangle is cut to form a squared domain. A solution of the Laplace problem is imposed as Dirichlet boundary conditions on the 4 sides of the domain (see figure 2.12) :

$$u(x,y) = \sin(\pi y) * \left(\cosh(-\frac{\pi x}{2}) - \frac{\sinh(-\frac{\pi x}{2})}{\tanh(\pi)}\right) \tag{2.29}$$

The analytical solution of the Lagrange multipliers on the right edge can be computed analytically :

$$\lambda(x,y) = -\frac{\partial u}{\partial \mathbf{n}} = -\frac{\partial u}{\partial x} = \pi * \sin(\pi y) * \left(\sinh(-\frac{\pi x}{2}) - \frac{\cosh\frac{-\pi x}{2})}{\tanh(\pi)}\right) \tag{2.30}$$

The two geometries are also extended to 3 dimensions. For the first one, the computational domain is bounded by two concentric spheres surrounded by a cubic mesh. The solution is imposed on the two spheres. The second geometry is extruded perpendicularly to the plane of the 2 dimensional geometry. The solution of the problem is imposed on the 6 faces of the cubic domain.

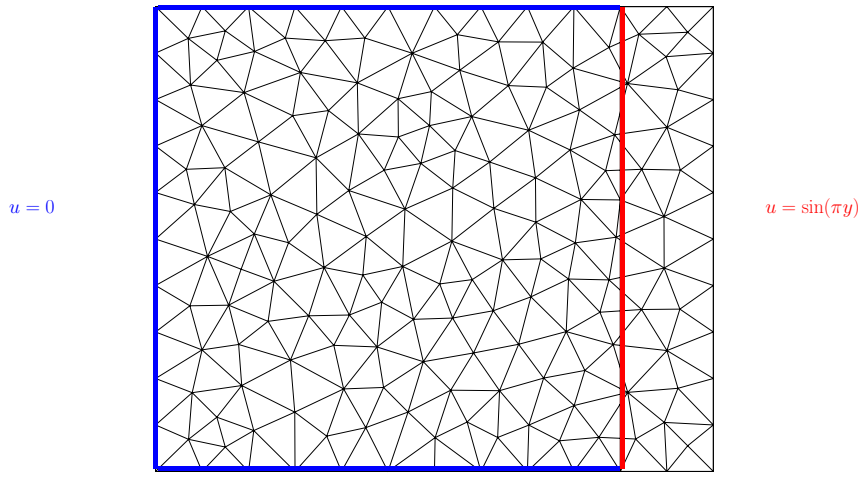


Figure 2.11 – Imposition of Dirichlet boundary condition on a straight cut not conforming to the mesh.

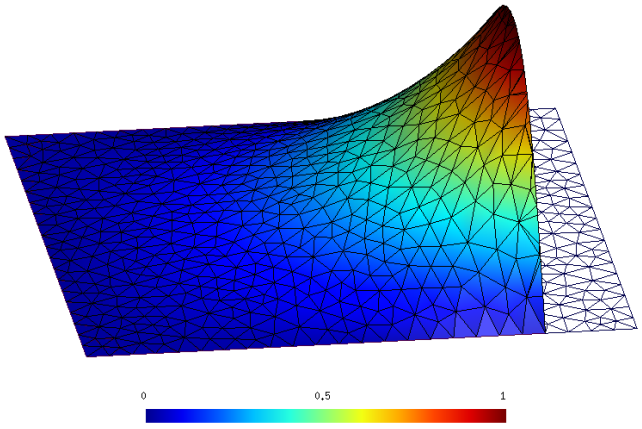


Figure 2.12 – Laplace solution imposed on the square domain.

Table 2.2 attempts to validate the value of τ_0. The two different simulations have been run on geometries of different sizes H with different element sizes h. The column $\tau_{min,U}$ shows the values of τ at the right bound of the interval of minimal discretization error ; $\tau_{min,\lambda}$ is the value of τ for the minimum value of the error on the Lagrange multipliers; the last two columns are computed as $\tau_0 = \frac{\tau}{h^2 H}$.

Geom.	Dim.	H	h	$\tau_{min,U}$	$\tau_{min,\lambda}$	$\tau_{0,U}$	$\tau_{0,\lambda}$
Concentric	2D	2	0.2	3.1×10^{-4}	7.5×10^{-4}	3.9×10^{-3}	9.3×10^{-3}
		2	0.1	8.0×10^{-5}	2.1×10^{-4}	4.0×10^{-3}	1.1×10^{-2}
		2	0.067	3.1×10^{-5}	9.9×10^{-5}	3.5×10^{-3}	1.1×10^{-2}
		2	0.05	1.8×10^{-5}	5.5×10^{-5}	3.6×10^{-3}	1.1×10^{-2}
		2	0.04	1.1×10^{-5}	3.6×10^{-5}	3.5×10^{-3}	1.1×10^{-2}
		2	0.033	8.0×10^{-6}	2.6×10^{-5}	3.6×10^{-3}	1.2×10^{-2}
		4	0.4	2.6×10^{-3}	6.0×10^{-3}	4.0×10^{-3}	9.4×10^{-3}
		6	0.6	7.4×10^{-3}	1.9×10^{-2}	3.4×10^{-3}	8.9×10^{-3}
		8	0.8	1.9×10^{-2}	4.5×10^{-2}	3.7×10^{-3}	8.8×10^{-3}
	3D	2	0.2	3.7×10^{-4}	8.5×10^{-4}	4.6×10^{-3}	1.1×10^{-2}
		2	0.13	2.1×10^{-4}	4.3×10^{-4}	5.8×10^{-3}	1.2×10^{-2}
		2	0.1	1.4×10^{-4}	2.0×10^{-4}	7.0×10^{-3}	9.8×10^{-3}
		2	0.08	1.2×10^{-4}	1.4×10^{-4}	9.1×10^{-3}	1.1×10^{-2}
Straight	2D	2	0.2	1.1×10^{-4}	5.6×10^{-5}	1.4×10^{-3}	6.9×10^{-4}
		2	0.1	2.8×10^{-5}	3.8×10^{-6}	1.4×10^{-3}	1.9×10^{-4}
		2	0.067	1.5×10^{-5}	1.4×10^{-6}	1.6×10^{-3}	1.5×10^{-4}
		2	0.05	1.0×10^{-5}	5.0×10^{-7}	2.1×10^{-3}	1.0×10^{-4}
		2	0.04	5.7×10^{-6}	2.9×10^{-7}	1.8×10^{-3}	9.0×10^{-5}
		2	0.033	4.3×10^{-6}	2.2×10^{-7}	1.9×10^{-3}	9.9×10^{-5}
	3D	2	0.2	3.1×10^{-4}	2.2×10^{-4}	3.8×10^{-3}	2.7×10^{-3}
		2	0.13	1.6×10^{-4}	1.6×10^{-4}	4.4×10^{-3}	4.4×10^{-3}
		2	0.1	1.1×10^{-4}	5.7×10^{-5}	5.6×10^{-3}	2.8×10^{-3}

Table 2.2 – Computation of τ_0 corresponding to minimal errors for different geometries with different dimensions, problem size (H) and elements size (h).

The first six lines show the results of a refined mesh. The value of τ_0 is almost constant, around 3.5×10^{-3} for the right bound of the minimal discretization error and around 1.1×10^{-2} for the minimal error of the Lagrange multipliers. As $\tau_{0,U} < \tau_{0,\lambda}$, we cannot have a minimal error for both the solution of the Laplace problem and the Lagrange multipliers. The results for the next three simulations representing the same geometry with constant number of elements but increasing size are still convincing with the same order of magnitude for the value of τ_0. The extension to 3 dimensions gives also good results but a slight

increase in $\tau_{0,U}$ can be seen, still in the same order of magnitude.

The results of the second simulation with a straight cut in a rectangular mesh also give values of τ_0 of the same order of magnitude for $\tau_{0,U}$. The value of $\tau_{0,\lambda}$ is constant for the simulations but is smaller than the values for the first simulation. This can be explained by the fact that the cut edge intersects the boundary of the rectangular mesh. At this position, the boundary condition is imposed weakly on the cut edge while it is imposed strictly on the boundary of the mesh. This affects the computation of the Lagrange multipliers.

To conclude the analysis of the graphs of the error depending on the value of the stabilizing parameter τ and the comparison of the different simulations, a good value of τ_0 seems to be around 10^{-3} to have a minimal error for the solution of the Laplace problem and to limit the error on the Lagrange multipliers.

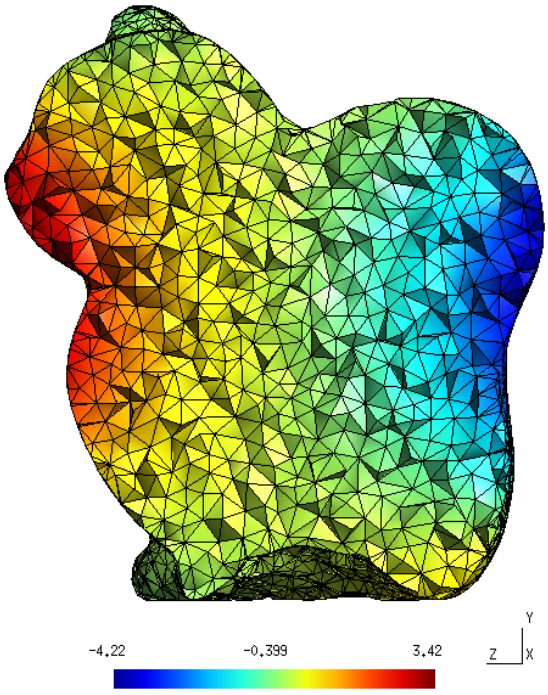


Figure 2.13 – Solution of the Laplace problem inside the geometry of a squirrel.

A last example of a complex 3D geometry is presented. The squirrel geometry used in the previous chapter is chosen (see figure 1.7). A box surrounding the geometry is meshed, not conforming to the geometry. The mesh is cut along the

levelset representation of the geometry and the following solution of the Laplace problem is imposed on the boundary of the geometry :

$$u(x, y, z) = \cos(0.3x)\cos(0.4y)\sinh(0.5z) \tag{2.31}$$

The value of the stabilizing parameter τ is computed with $\tau_0 = 10^{-3}$. The solution inside the geometry can be seen on figure 2.13. The L^2-norm of the discretization error inside the geometry is only 0.0688.

2.6 Conclusion

The stability problem that appears when trying to impose Dirichlet boundary conditions on embedded interfaces with Lagrange multipliers has been shown and existing methods to tackle this problem have been detailed. A new stabilized method have been proposed and compared to an existing one. The method uses a simple Laplace stabilization on the Lagrange multipliers with a scalar parameter. It has been shown to be more advantageous while not being consistent. It gives a wider range for the value of the stabilizing parameter with minimal error on the solution.

The stabilization parameter of the method has been adimensionalized for Laplace problems. This adimensionalized stabilization parameter has been calibrated on different geometries to minimize the displacement error. It has been fixed to the value 10^{-3}. To solve any Laplace problem, the stabilizing parameter can be computed easily, depending on the mesh size, the problem size and the material properties. Solving other kinds of problems requires to adimensionalize and calibrate the stabilizing parameter of the corresponding formulation.

Chapter 3

Adaptive mesh generation

After having described a technique to impose Dirichlet boundary conditions on a non conforming mesh by adding a stabilizing term, another technique to tackle the stability problem is described. It is based on the refinement of the mesh close to the embedded boundary. In this chapter, the refinement process is analysed to explain why an anisotropic refinement is necessary and to determine the required mesh size to obtain the optimal convergence for the solution of the problem and also for the geometry of the interface.

The results of this chapter have been published in papers [48] and [49].

3.1 Optimal *nearly* body-fitted mesh

Let us consider a mesh of a domain and an embedded interface Γ that is represented by the iso-zero of a levelset function $\phi(\mathbf{x})$ (see figure 3.1). To avoid the cutting of the mesh along the interface, which causes instability problems when imposing Dirichlet boundary conditions, the mesh can be split along the interface. To do so, all the elements that contain two vertices with opposite sign of the levelset function are included in the domain. This changes the geometry as the discrete approximation of the interface Γ^* is moved along the edges of the elements at the boundary of the domain. To reduce this modification of geometry, the size of the elements close to the interface should be reduced.

This way of treating the interface has obvious advantages. No enrichment is needed to account for intra-element features, as it is the case in X-FEM. Standard finite element formulations can therefore be used as is for solving a problem with an embedded interface. Yet, it is well known that this treatment of interfaces leads to a poor first order of convergence in finite element simulations [37]. We address this issue using anisotropic mesh adaptation near the interface Γ. The fact that mesh generators and finite element solvers are inde-

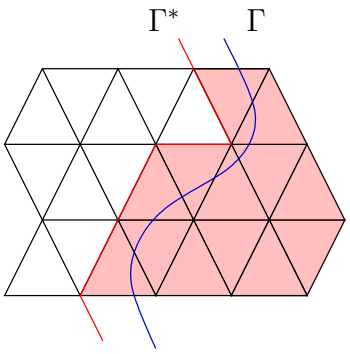


Figure 3.1 – A triangular mesh with the iso-zero of a levelset Γ. Stair-cased curve Γ^* is the discrete version of Γ.

pendent software components makes this approach appealing in practice [27].

3.1.1 Need for adaptive mesh refinement

Assume a mesh of elements with sizes h_{e_i}, $i = 1, \ldots, n_e$ and an associated finite element solution at order p, u_h, that is an approximation of the smooth exact solution u. The L_2 error on element e_i is of order:

$$\epsilon_i = \sqrt{\int_{e_i} \|u - u_h\|^2 d\mathbf{x}} = \mathcal{O}\left(h_{e_i}^k\right) \tag{3.1}$$

with $k = p + 1$ in smooth regions and $k = 1$ in the vicinity of the interface.

We first study the simple one-sided 2D Laplace problem :

$$\begin{aligned}
\Delta u &= 0, && \text{in } \Omega^+ : [0,1] \times [y^*, 1] && (3.2)\\
u &= \sin(\pi x)\hat{v}(y^*), && \text{at } \Gamma \ : y = y^* = 1/3 && (3.3)\\
u &= 0, && \text{at } \Gamma_D : y = 1 && (3.4)\\
\nabla u \cdot \mathbf{n} &= -\pi \hat{v}(y), && \text{at } \Gamma_N : x = 0;\ x = 1 \text{ and } y^* < y < 1 && (3.5)
\end{aligned}$$

The known analytical solution is $u(x, y) = \sin(\pi x)\hat{v}(y)$ where $\hat{v}(y) = \cosh(\pi y) - \coth(\pi)\sinh(\pi y)$.

The embedded planar surface Γ splits the domain $\Omega = [0, 1] \times [0, 1]$ into two parts Ω^+ and Ω^-. As discussed, the Dirichlet boundary condition on Γ is prescribed by imposing the value $\sin(\pi x)\hat{v}(y^*)$ at the nodes defining the discrete approximation Γ^* to Γ.

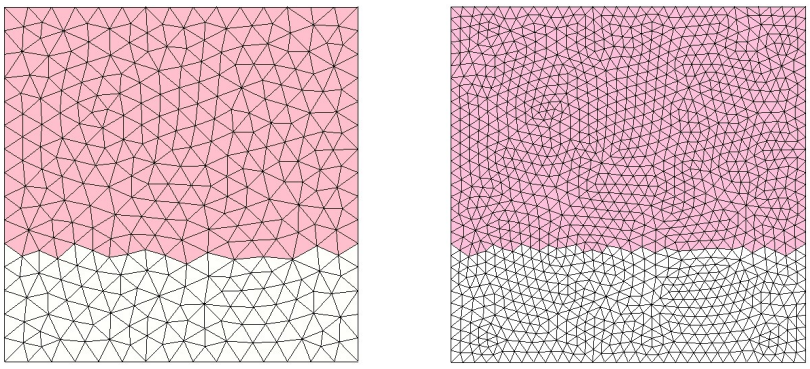
We perform a convergence study consisting in refining uniformly the mesh and measuring the global L_2 discretization error in the domain:

$$\mathcal{E}_{L_2}(u) = \sqrt{\sum_i \epsilon_i^2}. \tag{3.6}$$

Let us define a mesh refinement factor $r > 1$ and a refinement level l that determine for a *uniform mesh refinement* the mesh element size h as:

$$h^l = \frac{h_0}{r^l}, \quad l = 1, ..., l_n \tag{3.7}$$

where h_0 is an initial uniform mesh element size and l_n is the number of refinement levels. In this case, we solve equation (3.2) using linear finite elements ($p = 1$) on a sequence of five progressively uniform isotropic refined meshes ($l_n = 5$) with an initial mesh size of $h_0 = 0.1$ and a refinement factor of $r(\mathbf{x}) = 2$ (see figure 3.2). In this way, each linear triangular element is divided into four congruent triangles of half size and the new vertices of new elements lie exactly at the midpoint of the parent element's edges.



(a) Initial isotropic mesh (b) Uniformly refined mesh

Figure 3.2 – Coarsest isotropic mesh ($h_0 = 0.1$) and isotropic uniformly refined one ($p = 1$ and $r = 2$) for the one-sided problem (3.2).

The results of the convergence study are shown in figure 3.4, demonstrating a linear rate of convergence with uniform refinement, while the finite element method should yield second-order convergence for smooth problems.

It is thus clear that the poor first order rate of convergence shown for uniform refinement is due to the loss of accuracy at the interface. However, it is possible to obtain the optimal convergence by setting the mesh refinement factor $r(\mathbf{x})$

differently depending on whether an element is close to the embedded interface Γ or not. Indeed, as we have $\epsilon_i = \mathcal{O}(h_{e_i})$ near the interface and $\epsilon_i = \mathcal{O}(h_{e_i}^{p+1})$ in the bulk region, a global convergence order of $p+1$ can only be obtained if the refinement factor close to the interface is chosen as

$$r_\Gamma = r_b^{p+1} \Rightarrow h_\Gamma = \frac{h_0}{r_\Gamma}. \tag{3.8}$$

where r_b denotes the refinement factor in the bulk region. This means that the gap between the interface Γ and its nearly body-fitted version Γ^* has to decrease more rapidly than the bulk element size to ensure an optimal global rate of convergence [49]. An example of a mesh refined with isotropic adaptation for the 2D Laplace problem is shown in figure 3.3.

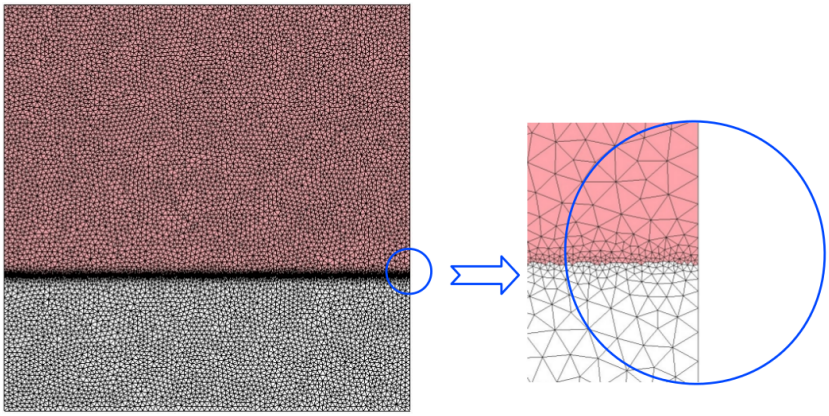


Figure 3.3 – Isotropic adaptive refined mesh for the 2D Laplace problem (3.2) for linear finite elements ($p = 1, h_0 = 0.1, r = r_b = 2, l = 4, E = 0.1$).

In practice, mesh adaptation is performed in the vicinity of the iso-zero of $\phi(\mathbf{x})$, i.e. in a band $\{\mathbf{x} \text{ s.t. } |\phi(\mathbf{x})| \leq E\}$ of thickness $2E$ around the interface. The mesh size for *isotropic adaptive mesh refinement* is computed by linearly interpolating between the minimal value $h_\Gamma^l = h_0/(r_\Gamma)^l$ at the interface Γ and a maximal value of $h_b^l = h_0/(r_b)^l$ in the bulk region:

$$h^l(\mathbf{x}) = h_\Gamma^l + \frac{h_b^l - h_\Gamma^l}{E}|\phi(\mathbf{x})|. \tag{3.9}$$

Here, r_b is a constant refinement factor in the bulk region.

Convergence results for the adaptive isotropic mesh refinement are shown in figure 3.4. It can be clearly seen that the optimal second-order convergence rate is recovered when applying isotropic adaptive refinement technique.

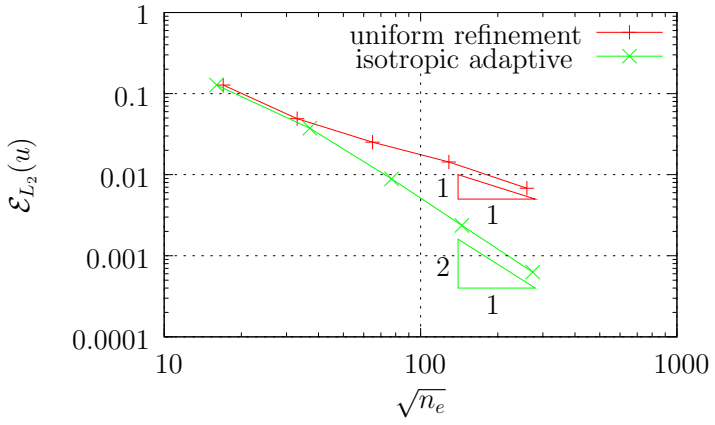


Figure 3.4 – Convergence study for the 2D Laplace problem (3.2)-(3.5): comparison between uniform and isotropic adaptive refinement techniques.

3.1.2 Need for anisotropic mesh refinement

Although the isotropic mesh refinement procedure provides the optimal convergence rate for the solution, it suffers from two problems: it does not enable geometrical convergence, and it still involves a significantly higher number of elements than body-fitted meshes [18].

First, the discrete representation Γ^* of the exact interface Γ is stair-cased even when employing very fine elements (see figure 3.3), so Γ^* does not converge uniformly towards Γ. More specifically, the measure of the interface (length in 2D or surface in 3D) does not converge to its exact value. This can be seen in figure 3.5 where the L_1 geometric error $\mathcal{E}_{geometry}$, measured as the error in the length of the interface, remains at a quasi constant level for isotropic adaptive refinement.

Such methods are thus severely limited when quantities of interest are integral values over the interface. In a CFD computation, for instance, it would lead to a large error in the evaluation of the lift and drag forces applied to the geometry by the flow. In a crack propagation simulation, the propagation path and velocity, that depend on quantities integrated along the crack front, would be strongly mispredicted.

Moreover, the isotropic mesh refinement implies a rapid growth in the number of elements in the band of thickness $2E$ around the interface. Even if the band is narrow compared to the size of the entire computational domain, this growth can lead to a significant increase in the global number of degrees of

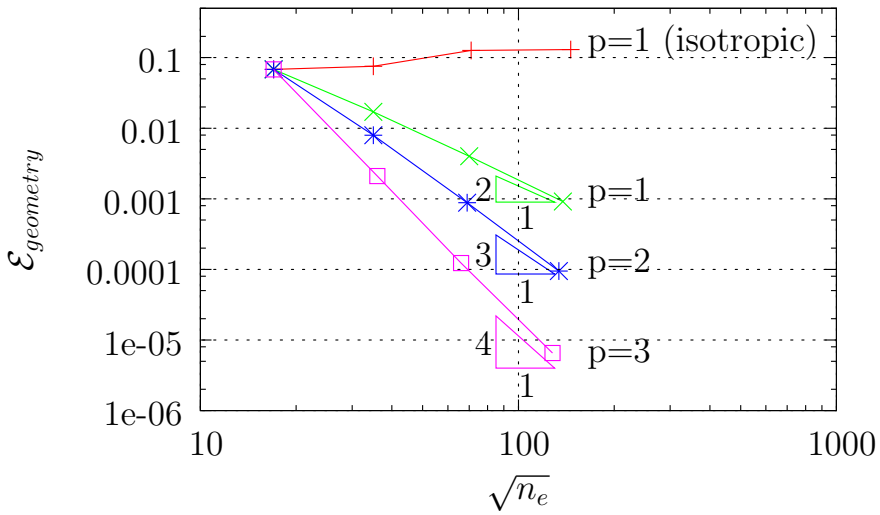


Figure 3.5 – Geometric error for the interface Γ^* of problem (3.2)-(3.5) analysis using high-order elements and anisotropic adaptive refinement technique. Comparison with the isotropic uniform refinement for $p = 1$.

freedom, which affects negatively the computational cost. A comparison of the number of elements n_e and the number of vertices n_v between uniform refined meshes and isotropic adaptive refined meshes for problem 3.2 is shown in table 3.1. Both refinement techniques started from an initial uniform mesh at the coarsest level corresponding to $h_0 = 0.1$. The uniform refinement factor and the bulk refinement factor are equal: $r = r_b = 2$ so that the refined meshes have the same mesh size $h^l = h_b^l$ in the bulk region for both refinement techniques. An average overhead of $\Delta n_e = 27.05\%$ in the number of elements n_e is observed for adaptively refined meshes compared to their uniformly refined counterparts.

A solution to both problems is to interpret r_Γ in formula (3.8) as a mesh refinement factor that should be applied only in the orthogonal (or normal) direction to the interface, so that only the mesh edges that cross the interface become very small. In the tangential direction, using a lower refinement factor should still yield the optimal rate of global convergence, as the solution along the interface is smooth.

In the case of the planar interface of problem 3.2, the refinement factor in the tangential direction can be the same as in the bulk region. However, in the more general case of a curved boundary, the error on the solution in the vicinity of the interface results both from the finite element approximation

l	h^l	n_e^{uniform}	h_Γ^l	n_e^{adapt}	$\Delta n_e(\%)$
1	0.1	275	0.1	275	0
2	0.05	1057	0.025	1402	32.64
3	0.025	4227	0.00625	5890	39.34
4	0.0125	16767	0.00156	20967	25.05
5	0.00625	67725	0.00039	75277	11.15

Table 3.1 – Mesh sizes and number of elements in refined meshes with uniform and adaptive refinement techniques for problem (3.2)-(3.5). We have $h_0 = 0.1, p = 1$ and $r = r_b = 2$ so that $r_\Gamma = 4$ and $h^l = h_b^l$.

and from the approximate representation of the geometry. Elements that are used in mesh generation are geometrically linear (i.e. straight sided), so the geometrical approximation, measured by the gap between the approximate and exact interfaces, converges at a second-order rate. Here, we assume that the part of the error in the solution that is due to the geometrical approximation decreases at the same rate as the geometrical error. The mesh refinement factor tangential to the interface is then computed as:

$$r_t = r_b^{(p+1)/2}. \tag{3.10}$$

Tangential mesh sizes are then also interpolated linearly in the band of thickness E.

In this manner, the geometry of the numerical approximation Γ^* of the interface Γ converges to the exact one. As seen in figure 3.6, the elements in the vicinity of the interface become increasingly stretched along Γ, which effectively controls the measure of Γ^*, preventing it from becoming "fractal-like" in the refinement process. We analyse the error in the 2D Laplace problem (3.2), using the proposed approach, with elements of different polynomial orders ($p = 1, 2, 3$). All the cases start with the same isotropic initial mesh ($h_0 = 0.1$). The refinement factors are $r_b = 2$ in the bulk region away from the interface and $r_\Gamma = 2^{p+1}$ in the fine region close to the interface. While the geometric error using isotropic mesh refinement does not decrease at all, the results of adaptive anisotropic refinement plotted in figure 3.5 show the advantage of this technique in terms of geometric convergence.

Moreover, the growth of the number of nodes in the mesh remains limited. A 2D example shows that an increase of only 17.8% in number of vertices with respect to uniformly refined meshes is sufficient for obtaining optimal convergence.

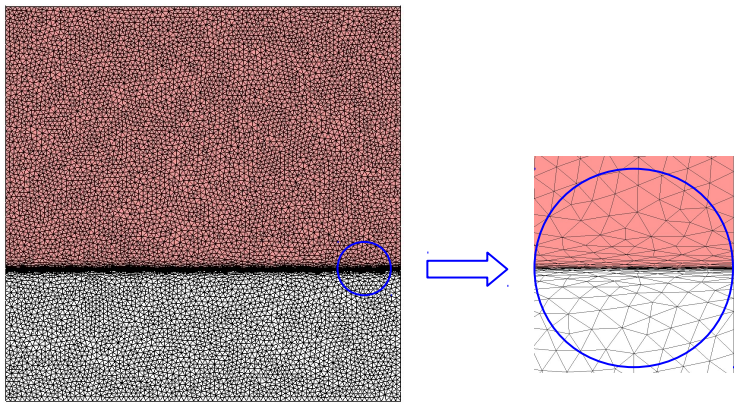


Figure 3.6 – Anisotropic adaptive refined mesh for the 2D Laplace problem (3.2)-(3.5): $p = 1, r = 2$ so that $r_\Gamma = 4$ and $r_t = 2$.

3.1.3 Mesh metric field construction in anisotropic adaptivity technique

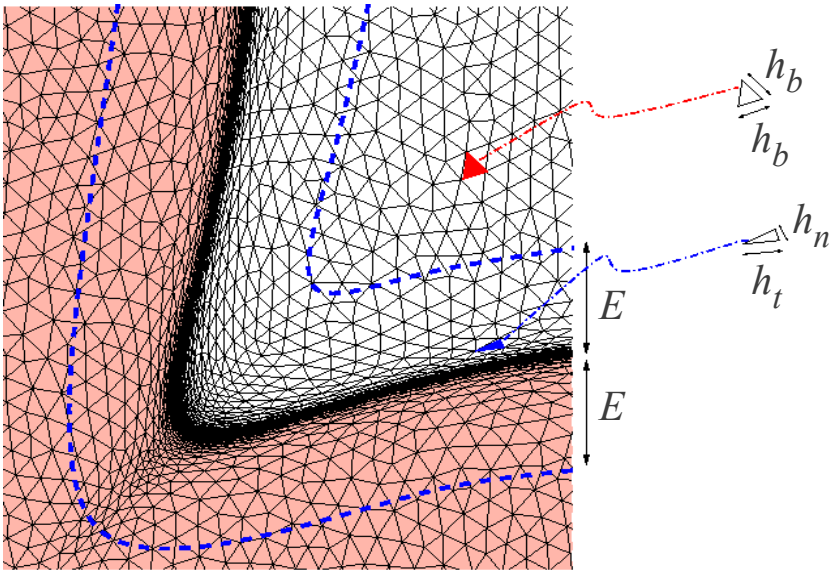


Figure 3.7 – Illustration for mesh size definition (h_n, h_t and h_b) in 2D anisotropic adaptive mesh

There exist open source mesh generators that allow to generate anisotropi-

cally adapted meshes based on metric maps, e.g. BAMG [30] in 2D and MMG3D [19] in 3D. Mesh adaptation in BAMG is based on a global constrained Delaunay kernel, while local mesh modifications are applied in MMG3D.

The aim of metric-based anisotropic mesh adaptation is to generate a uniform unit mesh [23] in a prescribed Riemannian metric space, in order to obtain an anisotropic adapted mesh in the Euclidean space. Anisotropic mesh adaptation is performed in the vicinity of the interface Γ described by the levelset function $\phi(\mathbf{x})$, i.e. in a band $\{\mathbf{x} \text{ s.t. } |\phi(\mathbf{x})| \leq E\}$ of thickness $2E$ around Γ.

With a linear discretization, the approximation error on the levelset function $\phi(\mathbf{x})$ is of second order. An appropriate metric field $\mathcal{M}$ can thus be constructed from the gradient vector $\nabla\phi(\mathbf{x}) = (\phi_x \ \phi_y \ \phi_z)^T$ and the Hessian matrix $\mathcal{H}(\phi(\mathbf{x}))$ of $\phi(\mathbf{x})$:

$$\mathcal{H}(\phi(\mathbf{x})) = \begin{pmatrix} \phi_{xx} & \phi_{xy} & \phi_{xz} \\ \phi_{yx} & \phi_{yy} & \phi_{yz} \\ \phi_{zx} & \phi_{zy} & \phi_{zz} \end{pmatrix} \tag{3.11}$$

Let us define an orthogonal basis $R^3 = \{\mathbf{n}, \mathbf{t}_1, \mathbf{t}_2\}$ at any point of the interface Γ, with $\mathbf{n}$ the unit normal vector to the interface and $\mathbf{t}_1, \mathbf{t}_2$ two unit tangent vectors in the principal directions of curvature of the surface defined by the iso-zero of the levelset. At any point in the band of thickness $2E$ around Γ, the Hessian matrix can undergo an eigenvalue decomposition:

$$\mathcal{H}(\phi(\mathbf{x})) = R^T \begin{pmatrix} \lambda_n & 0 & 0 \\ 0 & \lambda_{t_1} & 0 \\ 0 & 0 & \lambda_{t_2} \end{pmatrix} R. \tag{3.12}$$

If $\phi(\mathbf{x})$ is a distance function, the eigenvectors corresponding to the eigenvalues λ_n, λ_{t_1} and λ_{t_2} are proportional to $\mathbf{n}$, $\mathbf{t}_1$ and $\mathbf{t}_2$ respectively. Assuming that $h_n(\mathbf{x})$, $h_{t_1}(\mathbf{x})$ and $h_{t_2}(\mathbf{x})$ denote the element edge lengths in three principal directions (see in figure 3.7 for a detail illustration of notations), λ_n, λ_{t_1} and λ_{t_2} shall be inversely proportional to $h_n^2(\mathbf{x})$, $h_{t_1}^2(\mathbf{x})$ and $h_{t_2}^2(\mathbf{x})$, respectively.

In practice, the construction of the metric $\mathcal{M}$ at a given point of the band of thickness $2E$ around Γ requires thus the definition of the element edge lengths and the determination of the corresponding directions.

The normal direction $\mathbf{n}$ is obtained directly from the gradient $\nabla\phi(\mathbf{x})$. The associated element size h_n is computed by linearly interpolating between the minimal value $h_{n\Gamma}$ of elements located on the interface Γ and the maximal value h_b of isotropic elements in the bulk region:

$$h_n(\mathbf{x}) = h_{n\Gamma} + \frac{h_b - h_{n\Gamma}}{E} |\phi(\mathbf{x})| . \tag{3.13}$$

In the tangential directions $\mathbf{t_i}$ ($i = 1, 2$ for a general 3D case), the mesh size h_{t_i} is determined as:

$$h_{t_i} = \frac{2\pi}{\kappa_i N_p}, \tag{3.14}$$

where N_p is a user-specified parameter that represents the number of mesh points needed to discretize a whole circle and κ_i are principal curvatures corresponding to two directions $\mathbf{t}_i$. For two dimensional cases, the unique tangential direction is directly obtained as $\mathbf{t} = (-\phi_y \ \phi_x)^T$, and the curvature formula for implicit planar curves is given by [24] :

$$\kappa = \frac{\left|\mathbf{t}^T \mathcal{H}\, \mathbf{t}\right|}{|\nabla\phi(\mathbf{x})|^3} \tag{3.15}$$

For 3D cases, the gradient $\nabla\mathbf{n}$ of the unit normal vector $\mathbf{n}$ to the implicit embedded surface $\phi(\mathbf{x}) = 0$ is given by [9] :

$$\nabla\mathbf{n} = -\frac{1}{|\nabla\phi(\mathbf{x})|}\left(\mathcal{I} - \mathbf{n}\cdot\mathbf{n}^T\right)\mathcal{H}(\phi(\mathbf{x})) \tag{3.16}$$

where $\mathcal{I}$ is the identity matrix. The two non-zero eigenvalues of $\nabla\mathbf{n}$ give the two principal curvatures κ_i, and the corresponding directions $\mathbf{t}_i$ are the associated eigenvectors [9]. The formula can further be simplified for a distance function $\phi(\mathbf{x})$, as then $|\nabla\phi(\mathbf{x})| = 1$.

In practice, it is necessary to truncate the small and the large eigenvalues by imposing the maximal size as h_b and the minimal size as $h_{n\Gamma}$ in order to avoid singular metric case and to limit the local density of the adapted mesh [2]. Modified eigenvalues are then defined by:

$$\lambda'_n = \min\left(\max\left(\lambda_n, \frac{1}{h_b^2}\right), \frac{1}{h_{n\Gamma}^2}\right) \tag{3.17}$$

$$\lambda'_{t_i} = \min\left(\max\left(\lambda_{t_i}, \frac{1}{h_b^2}\right), \frac{1}{h_{n\Gamma}^2}\right) \tag{3.18}$$

The anisotropic mesh metric can finally be computed as:

$$\mathcal{M}(\mathbf{x}) = R'^T \begin{pmatrix} \lambda'_n & 0 & 0 \\ 0 & \lambda'_{t_1} & 0 \\ 0 & 0 & \lambda'_{t_2} \end{pmatrix} R', \tag{3.19}$$

where the matrix R' is made up of $\mathbf{n}$, $\mathbf{t}_1$ and $\mathbf{t}_2$. It is clear that the three unit vectors in the basis R^3 prescribe the orientation while the λ'_n, λ'_{t_1} and λ'_{t_2} control the size of the mesh elements along these directions. The meshing procedure gets the directional information of element shape and size from the metric tensor field $\mathcal{M}(\mathbf{x})$ and generates an adapted mesh. In the bulk region outside the transition band, the isotropic mesh is generated by prescribing an isotropic metric corresponding to a uniform mesh size h_b.

3.2 Application

To validate the results, the method is applied to a complex geometry. A woven composite material in which the yarn surfaces are described by a levelset function is taken into consideration. Its representative volume element is modeled. It is composed of cylindrical fibers with a sinusoidal axis organized perpendicularly with a phase difference to form a woven grid. Two layers of fibers are represented in the cubic cell. The geometry is inspired from the one studied in reference [37] where X-FEM is used to model complex geometries. The cube sides are equal to 2, the radius of the fibers is 0.1 and the sinus amplitude is 0.25. One analytical levelset represents each fiber surface.

Three refined meshes are generated using the proposed technique, with varying elements size in the layer around the fibers surface. Three parameters are given as input of the mesh adaptation procedure : the width E of the transition band, the mesh size $h_{n\Gamma}$ in the normal direction at the interface, and the mesh size h_b in the bulk region. $h_{n\Gamma}$ on the fiber surface and bulk element size h_b are changed to obtain finer meshes. The characteristic lengths of the meshes, as well as the resulting geometrical error $\mathcal{E}_{geo}$ on the surface of the fibers, are listed in table 3.2. As expected, the geometrical error decreases with increasing mesh density (decreasing elements size around the surfaces). The finest mesh is shown in figure 3.8.

A conformal mesh of high density is also created with a CAD modeler in order to compare the results of the adapted meshes (see figure 3.9). The geometry of the representative volume has been created using Spline curves to model the fibers. This conformal mesh is composed of 342 078 tetrahedra.

h_b	$h_{n\Gamma}$	n_e	$\mathcal{E}_{geo}$ (%)
0.10	0.015	170588	10.60
0.08	0.010	273966	7.88
0.06	0.005	615564	3.70

Table 3.2 – Mesh adaptation parameters and geometrical error for woven composite geometry.

The composite is submitted to loading test cases to determine its effective stiffness. Boron fibers and an aluminum matrix are considered. The material properties are: $E_f = 400GPa$, $\nu_f = 0.3$, $E_m = 72GPa$ and $\nu_m = 0.33$. Two test cases are carried out. The first one is a tensile test case. The face $y = 0$ of the representative volume is fixed and a displacement of 1% of the side length is imposed in the y-direction on the opposite face ($y = 2$). This test enables to compute the effective elastic modulus of the composite in the y-direction. In

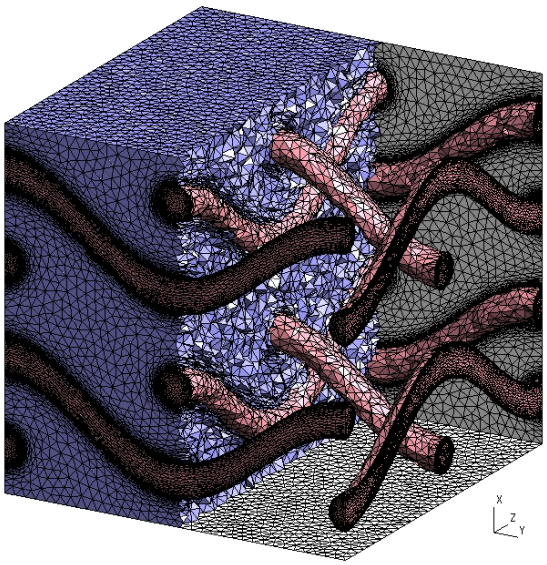


Figure 3.8 – Adapted mesh of woven composite.

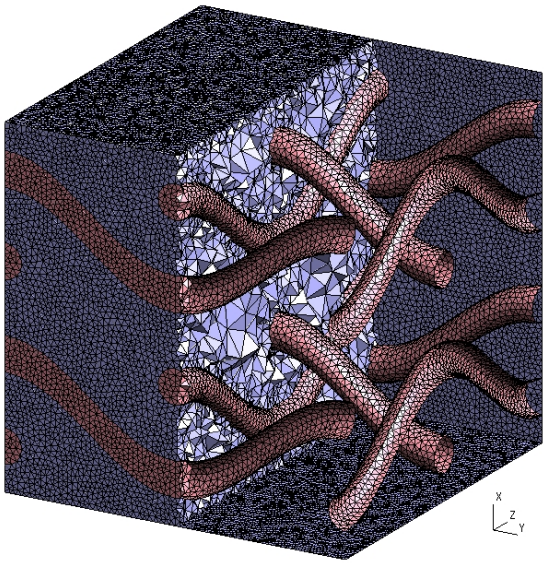


Figure 3.9 – Conformal mesh of woven composite.

the second test case, the face $y = 0$ is fixed and a displacement of 1% of the side length is imposed in the z-direction on the opposite face ($y = 2$). This test enables to determine the effective shear modulus in the yz-direction. First order finite elements are used to run these test cases.

The effective properties of the composite are obtained by integrating the stresses and the strains over the whole representative volume. The elastic modulus in the y-direction is obtained by the formula : $E_y = \frac{\bar{\sigma}_{yy}}{\bar{\epsilon}_{yy}}$, with $\bar{\sigma}_{yy}$ and $\bar{\epsilon}_{yy}$ being respectively the mean values of the stress and strain in the y-direction from the first test case. The shear modulus in the yz-direction is obtained by the formula : $\mu_{yz} = \frac{\bar{\sigma}_{yz}}{2\bar{\epsilon}_{yz}}$, with $\bar{\sigma}_{yz}$ and $\bar{\epsilon}_{yz}$ being respectively the mean values of the stress and strain in the yz-direction from the second test case.

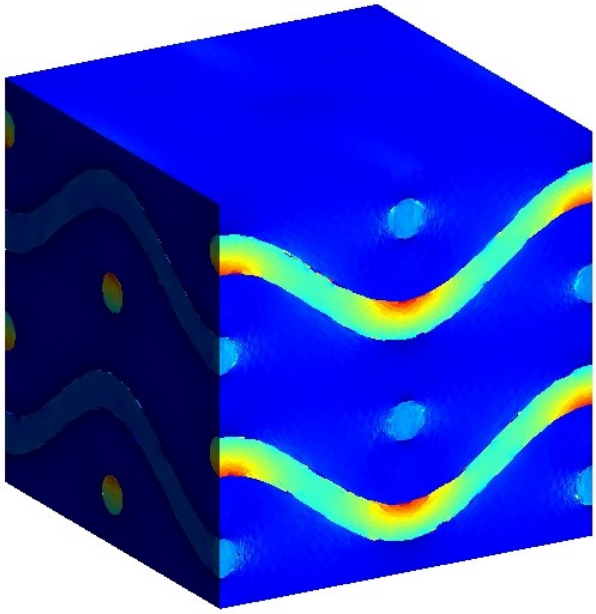


Figure 3.10 – Tensile normal stresses in composite material with anisotropic adapted mesh.

The conformal mesh gives an effective elasticity modulus of 82.2 GPa. It represents an increase of 14% compared to the matrix properties. The effective shear modulus is 29.8 GPa, compared to 27.1 GPa for the matrix. Figure 3.10 represents the normal stress distribution in the tensile test case along the y-axis with the finest adapted mesh. The relative error for the elasticity modulus and the shear modulus are computed for the 3 adapted meshes. The reference solution is calculated with a body-fitted mesh. The results are shown in figure 3.11.

The errors decrease as the adapted meshes contain more elements close to the surface of the fibers.

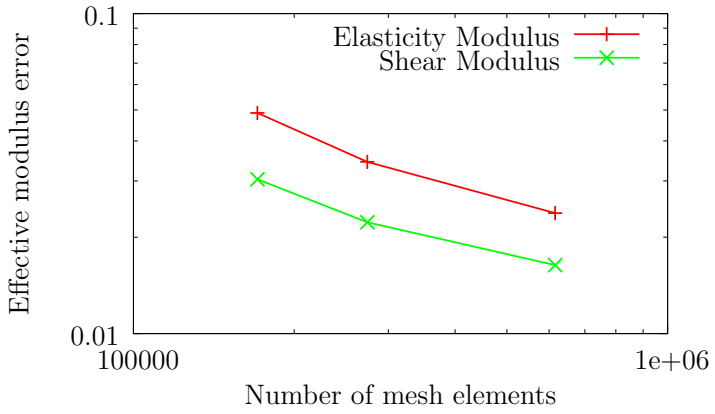


Figure 3.11 – Composite material with local anisotropic meshes to describe the fibers : errors on effective moduli for tensile and shear test cases.

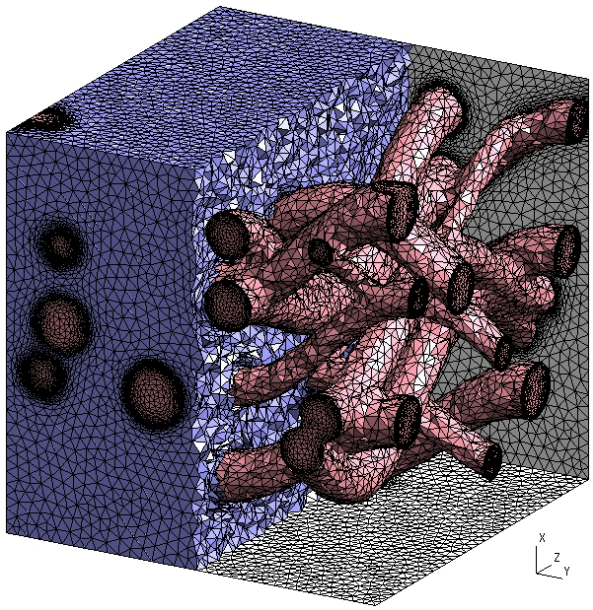


Figure 3.12 – Adapted mesh of intersecting fibers composite.

This test case illustrates the effectiveness of the approach presented above. The geometry is complex to model with a CAD modeler, while a levelset description is much simpler to set up. Another advantage of the method is that the levelset description allows fibers to intersect. The adaptation procedure only requires the distance function resulting from the union of intersecting fibers. An example of such a mesh is shown in figure 3.12.

3.3 Conclusion

This method to impose Dirichlet boundary conditions on embedded interfaces using local mesh adaptation around the interface is attractive as it enables to use the classical finite element formulation. The anisotropic adaptation needed to obtain a geometrical convergence can be executed easily thanks to mesh metrics. But it comes with the price of a mesh adaptation that has to be calibrated with several arbitrary geometric parameters and depends on external algorithms that can be time costly. Moreover, the mesh adaptation increases the number of degrees of freedom, even if anisotropic mesh adaptation is preferable than isotropic.

Chapter 4

Numerical examples

Up to now, the proposed methods have been applied to Laplace equations. These can be used to solve elliptic problems describing heat conduction, electrostatics or potential flow in a domain for example.

The methods will now be applied to different problems in linear elasticity to validate our results. The problem equations are different, so the relation to adimensionalize the stabilizing parameter has to be established in terms of the parameters of the elasticity problem.

4.1 Equations of linear elasticity

The equations of linear elasticity are :

$$\nabla \cdot \sigma + \mathbf{f} = \mathbf{0} \quad \text{in} \quad \Omega \tag{4.1}$$

$$\sigma = \frac{E}{1+\nu}\epsilon + \frac{E\nu}{(1+\nu)(1-2\nu)}\mathrm{tr}(\epsilon)\delta \tag{4.2}$$

$$\epsilon = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T) \tag{4.3}$$

$$\mathbf{u} \cdot \mathbf{e}_i = \bar{u}_i \quad \text{on} \quad \Gamma_D \ ; \ \forall i \in \{1, \dots, 3\} \tag{4.4}$$

$$\sigma \cdot \mathbf{n} = \bar{\mathbf{g}} \quad \text{on} \quad \Gamma_N \tag{4.5}$$

with σ the stress tensor, $\mathbf{f}$ the body force per unit volume, E the Young's modulus, ν the Poisson's ratio, ϵ the strain tensor, $\mathrm{tr}(\cdot)$ the trace operator, δ the identity tensor, $\mathbf{u}$ the displacement vector, $\mathbf{e}_i$ the base vector in the i-th direction, $\bar{u}_i$ the imposed displacement in the i-th direction on Γ_D, $\mathbf{n}$ the outward normal on Γ_N and $\bar{\mathbf{g}}$ the imposed normal forces on Γ_N.

The weak formulation is established by multiplying equation (4.1) by a test function $\hat{\mathbf{u}}$ and integrating over the domain Ω :

$$\int_\Omega \hat{\mathbf{u}} \cdot (\nabla \cdot \sigma + \mathbf{f})\ d\Omega = \mathbf{0} \tag{4.6}$$

Some developments lead to the following form :

$$\int_\Omega \epsilon(\hat{\mathbf{u}}) : \mathbf{C} : \epsilon(\mathbf{u})\ d\Omega = \int_\Omega \hat{\mathbf{u}} \cdot \mathbf{f}\ d\Omega + \int_{\Gamma_N} \hat{\mathbf{u}} \cdot \bar{\mathbf{g}}\ d\Gamma \tag{4.7}$$

where $\mathbf{C}$ is the stiffness tensor.

To impose weakly the Dirichlet boundary conditions, a term with several Lagrange multipliers $\lambda^{(i)}$ is added to this formulation. The problem becomes a saddle point problem :

$$\min_{\mathbf{u}\in H^1(\Omega)} \max_{\lambda\in L^2(\Gamma_D)} \frac{1}{2}\int_\Omega \epsilon : \mathbf{C} : \epsilon\ d\Omega - \int_\Omega \mathbf{u}\cdot\mathbf{f}\ d\Omega - \int_{\Gamma_N} \mathbf{u}\cdot\bar{\mathbf{g}}\ d\Gamma + \int_{\Gamma_D} \lambda^{(i)}(\mathbf{u}\cdot\mathbf{e}_i - \bar{u}_i)\ d\Gamma \tag{4.8}$$

The Euler-Lagrange equations relative to the first variation of (4.8) with respect to $\mathbf{u}$ gives :

$$\nabla \cdot \sigma + \mathbf{f} = \mathbf{0} \quad \text{in} \quad \Omega \tag{4.9}$$

$$\sigma \cdot \mathbf{n} = -\lambda^{(i)}\mathbf{e}_i \quad \text{on} \quad \Gamma_D \tag{4.10}$$

$$\sigma \cdot \mathbf{n} = \bar{\mathbf{g}} \quad \text{on} \quad \Gamma_N \tag{4.11}$$

The Dirichlet boundary condition on Γ_D becomes a Neumann boundary condition. The Lagrange multipliers are equivalent to forces per unit surface in each direction on Γ_D.

A second term is added to stabilize the formulation (4.8), similar to the term used in the first method of paragraph 2.2.2 consisting in a Laplace stabilization :

$$\min_{\mathbf{u}\in H^1(\Omega)} \max_{\lambda\in L^2(\Gamma_D)} \frac{1}{2}\int_\Omega \epsilon : \mathbf{C} : \epsilon d\Omega - \int_\Omega \mathbf{u}\cdot\mathbf{f} d\Omega - \int_{\Gamma_N} \mathbf{u}\cdot\bar{\mathbf{g}} d\Gamma + \int_{\Gamma_D} \lambda^{(i)}(\mathbf{u}\cdot\mathbf{e}_i - \bar{u}_i)\ d\Gamma - \sum_{i=1}^{3} \frac{1}{2}\int_{\Gamma_D} \tau(\nabla\lambda^{(i)})^2 d\Gamma \tag{4.12}$$

with τ a stabilizing parameter. As the Lagrange multipliers are forces per unit surface on the boundary Γ_D, the stabilizing term can be physically interpreted as a surface tension in the tangential direction on Γ_D.

The Euler-Lagrange equations of (4.12) are, in addition to (4.1), (4.4) and (4.5) :

$$(\mathbf{u}\cdot\mathbf{e}_i-\bar{u}_i)=\sum_{i=1}^{3}\nabla\cdot\tau\nabla\lambda^{(i)} \tag{4.13}$$

Assuming a second order of convergence for $\mathbf{u}$, i.e.

$$\|(\mathbf{u}\cdot\mathbf{e}_i-\bar{u}_i)\|_2=\mathcal{O}\left(h^2\frac{\|\mathbf{u}\|}{H^2}\right)$$

with h the mesh size and H the problem size, equation (4.13) gives the following order of magnitude for τ :

$$\tau=\tau_0\frac{h^2H(1+\nu)}{E} \tag{4.14}$$

with τ_0 independent of h, H, E and ν.

4.2 Implementation

The method is implemented in the elastic solver of the `Gmsh` software. The implementation is similar to the one realized in the Laplace solver (see paragraph 2.4).

An example of `Python` code to solve an elastic problem is shown on listing 4.1. The problem is a tensile stress test case on a 2D rectangular plate. The analytical displacement is first defined (lines 1-9). The *elasticitySolver* is then created (line 11) and the mesh is added to the solver (line 12), defining the vector Lagrange function space for the displacement according to the dimension of the problem. The mesh is cut by the levelset (line 15) to create the non-conforming boundary. The material properties are added to the parts of the mesh composing the domain of computation (line 17). The Lagrange multipliers are added to the model on the non-conforming boundary, defining the value of the stabilizing parameter τ, the direction and the value of the displacement (lines 19-20). The boundary conditions on the conforming edges are finally added to the model (lines 21-22). The linear system is then assembled and solved.

To assemble the system, the fixed degrees of freedom are first blocked then the variables are created for the displacements and the Lagrange multipliers. The load terms are assembled first then the terms related to the Lagrange multipliers (cross terms, Laplace term and load term on the border) and finally the elastic terms. To validate the result, the L^2-norm of the error $\|e_{\mathbf{u}}\|=\int_\Omega\left\|\mathbf{u}-\mathbf{u}^h\right\|^2d\Omega$ is computed.

```
1  def ux(x,y,z):
2      return sig/E*x
3  def uy(x,y,z):
4      return -nu*sig/E*y
5  def zero(x,y,z):
6      return 0.
7  dispX = simpleFunctionPython(ux)
8  dispY = simpleFunctionPython(uy)
9  dispZ = simpleFunctionPython(zero)
10
11 mySolver = elasticitySolver(100)
12 mySolver.setMesh("plate.msh")
13
14 ls = gLevelsetPlane(1., 0., 0., -1., 1)
15 mySolver.cutMesh(ls)
16
17 mySolver.setElasticDomain(dom,E,nu)
18 tau = 1.e-10
19 mySolver.setLagrangeMultipliers(line, tau, SVector3(1,0,0), dispX)
20 mySolver.setLagrangeMultipliers(line, tau, SVector3(0,1,0), dispY)
21 mySolver.setEdgeDisp(side, 0, dispX)
22 mySolver.setEdgeDisp(side, 1, dispY)
23 mySolver.solve()
24
25 Er = mySolver.computeL2Norm(dispX,dispY,dispZ)
```

Listing 4.1 – Pyhton code to solve an elastic problem using Lagrange multipliers.

4.3 Simple test case : plate with a hole

To determine a satisfying value for the stabilizing parameter τ for linear elasticity problems, a simple 2D test case is first analysed. It consists in a rectangular plate with a circular hole in its center submitted to a uniaxial tensile stress test (see figure 4.1). This test case, called the Kirsch problem, is studied to compute the stress concentration that appears around the hole.

The analytical solution of the stress and displacement fields is given in cylindrical coordinates (see [35]) :

$$
\begin{aligned}
\sigma_r &= \frac{\sigma}{2}\left[\left(1-\frac{R^2}{r^2}\right)+\left(1-4\frac{R^2}{r^2}+3\frac{R^4}{r^4}\right)\cos 2\theta\right] \\
\sigma_\theta &= \frac{\sigma}{2}\left[\left(1+\frac{R^2}{r^2}\right)-\left(1+3\frac{R^4}{r^4}\right)\cos 2\theta\right] \\
\sigma_{r\theta} &= -\frac{\sigma}{2}\left(1+2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right)\sin 2\theta \\
u_r &= \frac{\sigma r}{2E}\left[(1-\nu)+(1+\nu)\cos 2\theta+(1+\nu+4\cos 2\theta)\frac{R^2}{r^2}-(1+\nu)\cos 2\theta\frac{R^4}{r^4}\right] \\
u_\theta &= -\frac{\sigma r \sin 2\theta}{2E}\left[(1+\nu)+2(1-\nu)\frac{R^2}{r^2}+(1+\nu)\frac{R^4}{r^4}\right]
\end{aligned}
$$

with σ the load applied at the end of the plate and R the radius of the hole

centered at the origin. Around the hole ($r = R$), σ_r and $\sigma_{r\theta}$ vanish while $\sigma_\theta = \sigma(1 - 2cos2\theta)$, reaching a maximum value of 3σ at $\theta = \frac{\pi}{2}$.

A uniform triangular mesh is created in a rectangular plate. The hole at the center is created with a levelset function and the mesh is cut along the hole border (see figure 4.1). The displacement field is applied at the boundaries of the studied domain : the border of the rectangular plate and the circular hole at the center. The boundary conditions at the hole are applied using Lagrange multipliers. The advantage of our method in this case is that the size of the hole can be changed easily without needing to remesh the plate.

The plate is a rectangle of length equal to 4 and height equal to 2 with the radius of the hole $R = 0.2$. The material parameters are taken as $E = 200\ MPa$, $\nu = 0.3$ and the tensile stress $\sigma = 1\ MPa$. The deformed shape with the displacement norm can be seen on figure 4.2 with the original shape in black lines. The von Mises stress can be seen on figure 4.3. The stress value is overestimated at the border of the hole but is well represented anywhere else in the domain.

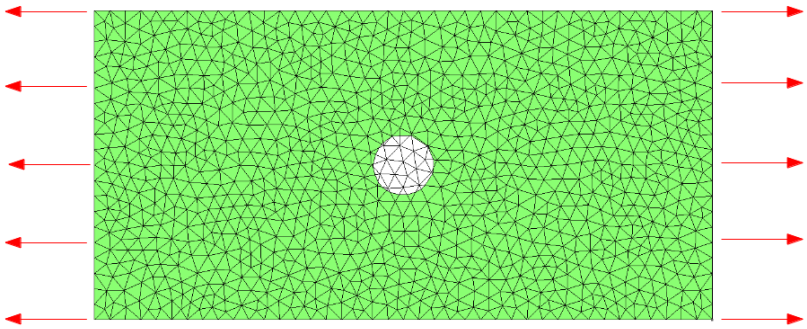


Figure 4.1 – Uniform triangular mesh of the rectangular plate with a not conforming circular hole (computational domain in green) and uniaxial tensile loading.

To determine a satisfying value for the stabilizing parameter τ, an error analysis is executed. The L^2-norm of the discretization error $\|e_{\mathbf{u}}\| = \int_\Omega \left\|\mathbf{u} - \mathbf{u}^h\right\|^2 d\Omega$ and of the Lagrange multiplier error $\|e_\lambda\| = \int_{\Gamma_D} \left\|\lambda - \lambda^h\right\|^2 d\Gamma$ are computed for different meshes with varying mesh size and for a large range of values for τ. Figure 4.4 shows the relative L^2-norm of the discretization error for 3 meshes and figure 4.5 shows L^2-norm of the error on the Lagrange multipliers in the x and y direction for the 3 meshes.

The evolution of the error according to the stabilizing parameter value looks like the one observed for the Laplace problem. The error is constant and signifi-

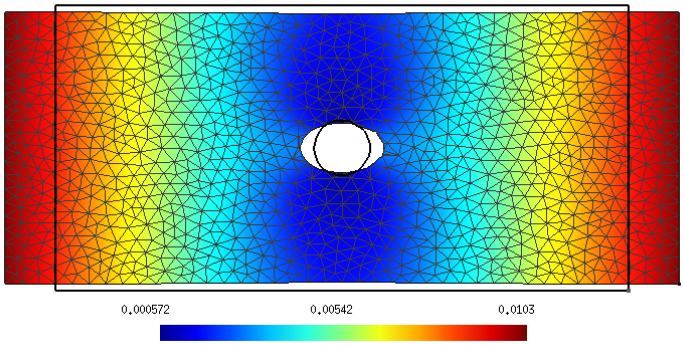


Figure 4.2 – Displacement norm of the deformed plate with a hole plot on the deformed shape (undeformed boundaries in black).

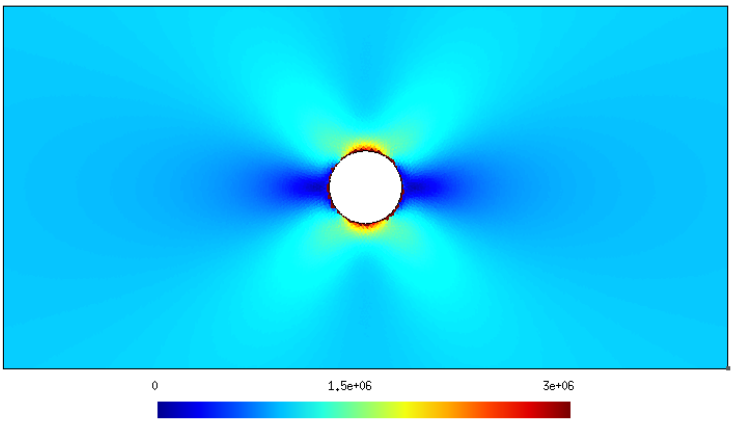


Figure 4.3 – von Mises stress inside the plate with a hole.

cant for very small values of τ as the stabilizing term is too small in comparison to the problem terms. Then the error decreases to a minimum range as τ reaches an adequate value. Finally the error increases when τ becomes too big and makes the stabilizing term too important compared to the problem terms. The range of τ with minimal error is shifted to the left as the size of the mesh elements decreases as τ depends on the square of the mesh size. The constant discretization error for large values of τ (where the Lagrange multipliers are flatten) is smaller than the one for small values of τ (where the Lagrange multipliers are unstable). This would mean that a constant value for the Lagrange multipliers returns satisfying results.

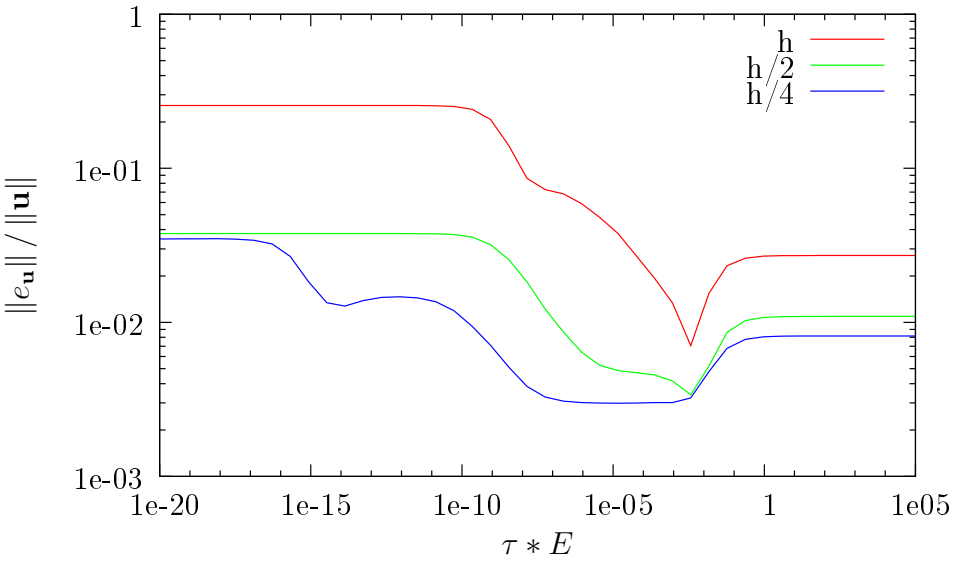


Figure 4.4 – Relative L^2-norm of the discretization error as a function of $\tau * E$ for the plate with a hole.

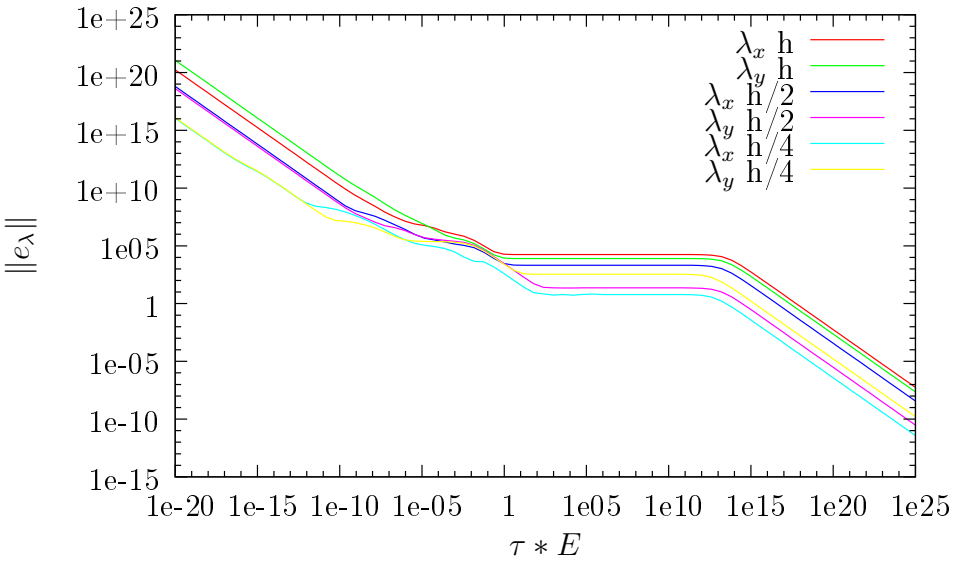


Figure 4.5 – L^2-norm of the error on the Lagrange multipliers in x and y direction as a function of $\tau * E$ for the plate with a hole.

The error on the Lagrange multipliers decreases as τ increases. It confirms the previous analysis on the discretization error. According to equation (4.10), the Lagrange multipliers are equivalent to the normal stress at the hole boundary : $\sigma \cdot \mathbf{n}|_{r=R} = \mathbf{0}$. The Lagrange multipliers should then be as close as possible to 0, but not too small to keep the term that imposes the Dirichlet boundary condition on Γ_D significant in formulation (4.12). The stabilizing parameter τ should then be as big as possible. The best value is at the right corner of the minimal level of the discretization error, corresponding to $\tau_0 = 10^{-2}$.

This analysis gives us a rule of good practice to impose Dirichlet boundary conditions on a non conforming interface by adding a stabilizing term with a predefined coefficient that depends on the size of the problem and the size of the mesh.

This Kirsch problem test case can also be used to realize a convergence analysis of the two presented methods. Three models are studied to compare their convergence depending on the mesh size. The first one is a conformal model of the problem. The second one uses the cutting of the elements with the stabilization term (see mesh on figure 4.1). And the third one uses the anisotropic mesh adaptation close to the circular hole (see mesh on figure 4.6). For these three models, four different meshes with different mesh sizes are generated. The prescribed mesh size for the model using anisotropic mesh adaptation is the mesh size inside the bulk while the anisotropic elements around the hole have the prescribed size in the tangential direction and a smaller size in the normal direction. The Kirsch problem is solved inside the domain by applying the boundary conditions on the edges of the rectangular plate and on the circular hole at the center. The elastic energy is computed for each problem and its relative error is obtained :

$$e_U = \frac{||U - U_{ref}||}{U_{ref}}$$

with U_{ref} a reference solution computed on the conformal mesh with elements of a much smaller size.

The results are presented on figure 4.7. An optimal second order of convergence is expected for linear finite elements. A linear regression gives the rate of convergence for each method. The model with the conformal mesh presents a mean convergence rate of 1.76, while the method using the cutting of the elements presents a rate of convergence of 2.07 and the method using anisotropic mesh adaptation presents a rate of convergence of 1.75. We obtain good rates of convergence for the three models. For this example, the method using mesh adaptation presents slightly better results but it requires around 20% more elements as the anisotropic elements around the hole have a smaller size in the normal direction.

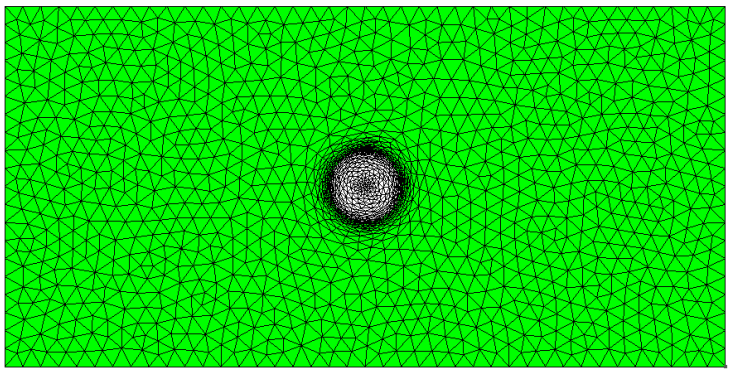


Figure 4.6 – Uniform triangular mesh of the rectangular plate with an anisotropic mesh adaptation around the circular hole (computational domain in green).

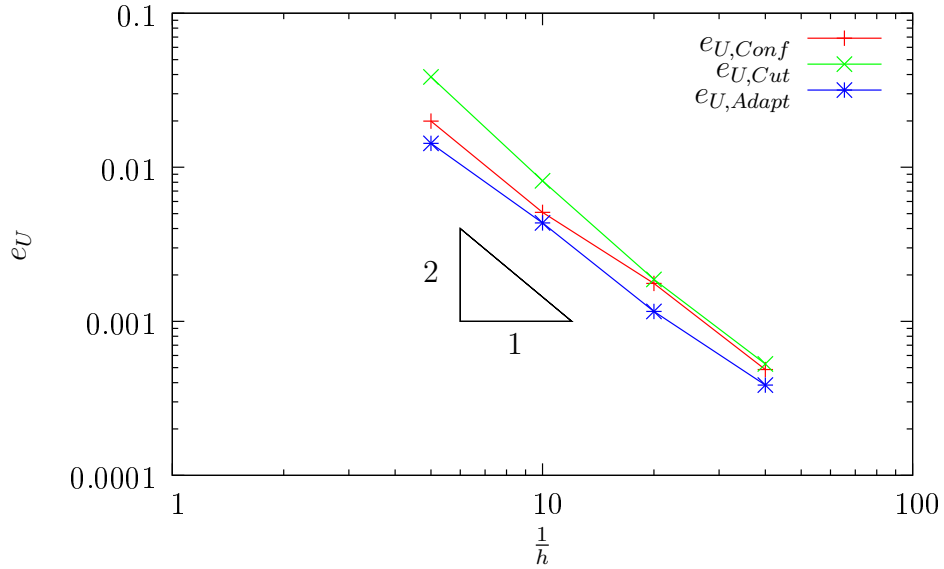


Figure 4.7 – Relative error on the elastic energy for the Kirsch problem depending on the mesh size using a conformal mesh, the method using the cutting of the elements and the method using anisotropic mesh adaptation.

4.4 Test case : gearing wheel

The second test case is a gearing wheel. Its geometry comes from a STL representation and has been presented in section 1.7.2.

The body is loaded to represent a real using case. The central hole is fixed to avoid displacement in any direction and a displacement is imposed on a side of each tooth in the positive radial direction, acting like the contact of a gear coupling to another wheel that drags the one studied. This loading creates a torque in the wheel.

Dirichlet boundary conditions are applied on the surface. The first one on the cylinder surface of the central hole and the other ones on a side of each tooth. These surfaces have to be delimited. A cylindrical levelset surrounding each tooth surface is created and the elements inside these levelsets are split from the whole surface thanks to the split method used in chapter 3. Figure 4.8 shows the delimited surfaces where the boundary conditions are applied on the cut mesh.

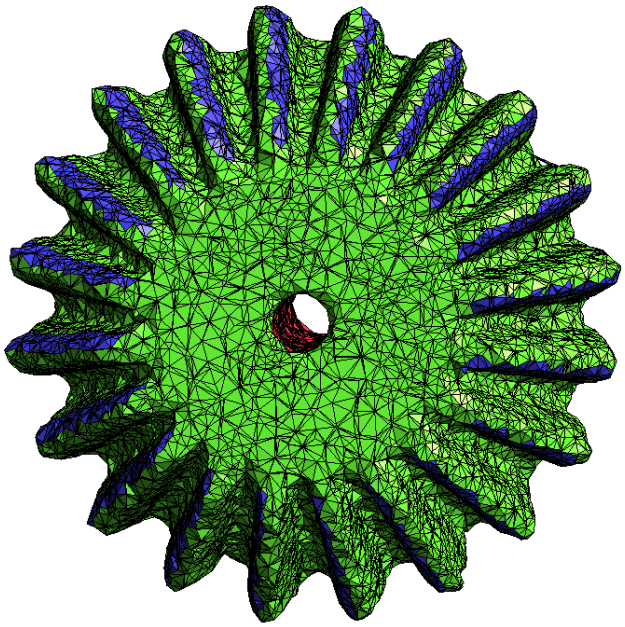


Figure 4.8 – Surfaces where the Dirichlet boundary conditions are applied on the cut mesh of the gearing wheel.

A displacement of 0.01 is imposed on the teeth in the positive radial di-

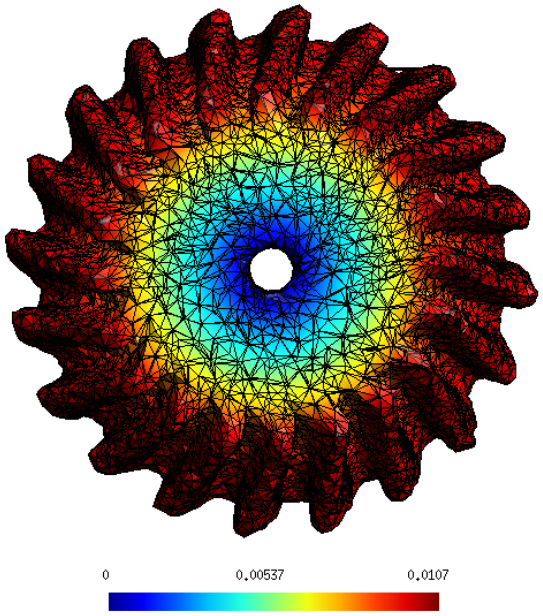


Figure 4.9 – Displacement norm on the deformed shape of the loaded gearing wheel using Lagrange multipliers.

rection. The material parameters are chosen : $E = 20\ MPa$ and $\nu = 0.3$. The Dirichlet boundary conditions are applied weakly with Lagrange multipliers with a stabilizing parameter $\tau_0 = 10^{-2}\frac{h^2 H(1+\nu)}{E}$. The resulting displacement norm on the wheel is shown on figure 4.9.

Two other meshes, with similar numbers of degrees of freedom, are created and loaded to compare the results. A second mesh is obtained using the anisotropic mesh adaptation with a normal characteristic length at the interface 20 times smaller than the bulk characteristic length (see figure 4.10). A third mesh is created using a CAD representation of the geometry in a STEP format (see figure 4.11). The STEP file describes the geometry with its edges as B-splines. The meshing of B-splines is complex but provides a conforming mesh of the geometry.

We can first compare the number of nodes of the geometries and the time needed to produce the meshes and to compute the loading on each mesh (see table 4.1). The numbers of nodes of the three meshes are different as the two first geometries are meshes of a box surrounding the wheel. Some nodes have fixed degrees of freedom when Dirichlet boundary conditions are applied strongly in the adapted and the conforming meshes, while degrees of freedom for the La-

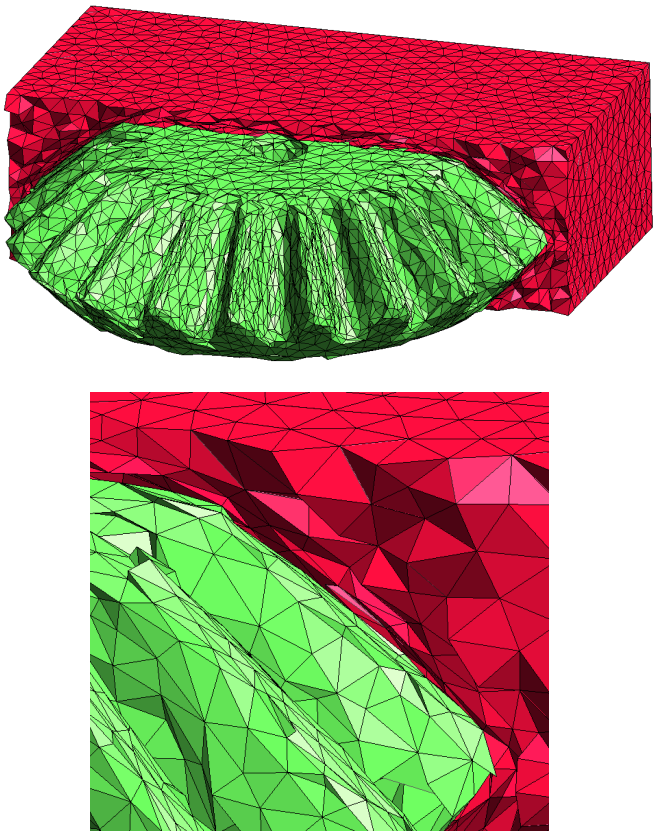


Figure 4.10 – Anisotropic adapted mesh of the gearing wheel.

grange multipliers are added for the cut mesh.

Mesh	Nodes	Free dofs	Fixed dofs	Meshing (s)	Loading (s)
Cut	29,113	36,755	0	129	118.8
Adapted	31,715	38,014	3,341	654.2	7.75
Conforming	11,097	30,863	2,704	46.4	3.9

Table 4.1 – Comparison of the three meshes of the gear : number of nodes of the mesh, number of free degrees of freedom (dofs), number of fixed degrees of freedom, time to create the mesh and time of computation of the loading (in seconds).

The meshing and loading times are also different for the three meshes (ex-

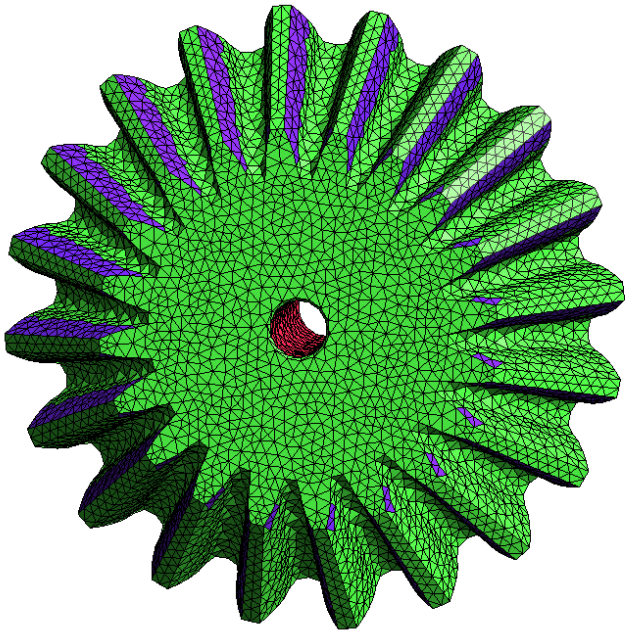


Figure 4.11 – Conforming mesh of the gearing wheel with boundary condition surfaces.

pressed in seconds). The first mesh is created significantly faster than the others as it is a simple uniform tetrahedral mesh of a box (2 s) that is cut to obtain the wheel surface (42.7 s) and then split with the 21 boundary surfaces (21 x 4 s). The adapted mesh needs 3 iterations to converge (346 s) then it is split to obtain the wheel surface (131 s) and the boundary surfaces (21 x 8 s). Finally, the conforming procedure creates directly the mesh of the wheel (5.2 s) from which the boundary surfaces are split (20 x 2 s).

The time of computation is equivalent for the adapted and conforming meshes as their formulation is the usual finite element formulation, while the cut mesh has a mixed formulation with Lagrange multipliers that makes the resolution of the linear system more complex. Trying different solvers, only the `PETSc` solver with LU factorisation converges while the solvers using Cholesky decomposition do not because of pivot being too small.

To conclude the time analysis, we can see that the mesh cut is much faster than the mesh adaptation to obtain a mesh of the geometry but slows down the computation of the loading by adding degrees of freedom of different order of magnitude that makes the resolution of the linear system more complex. In this example, the minimum total time is however obtained for this technique.

Yet if several different loadings have to be applied on the geometry, the mesh adaptation can be beneficial despite a longer meshing time thanks to a small loading computation time.

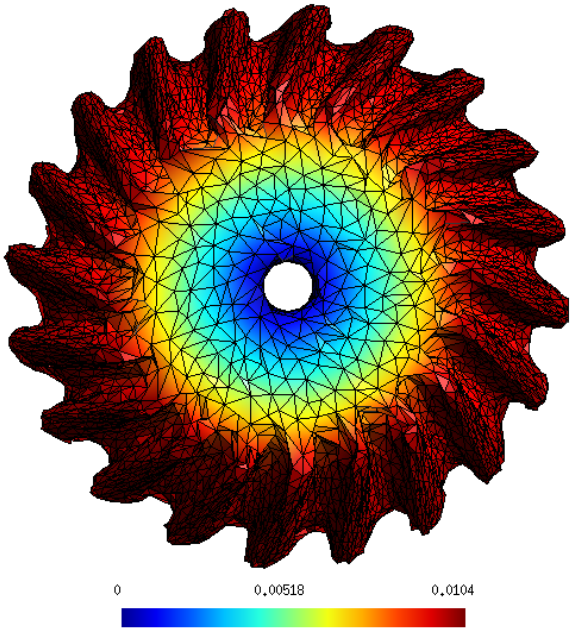


Figure 4.12 – Displacement norm on the deformed shape of the loaded gearing wheel represented with the anisotropic adapted mesh.

The solutions of the three loadings can also be compared. The displacement norm on the adapted and conforming meshes of the gear can be seen on figures 4.12 and 4.13. They look very similar to the displacement on the cut mesh on figure 4.9. Figure 4.14 shows the von Mises stress in the wheel on the conformal mesh. The stress concentrations at the tips of the teeth complicate the comparison. The results of the three loadings are compared in table 4.2 where the elastic energy $U = \int_\Omega \epsilon : \mathbf{C} : \epsilon \, d\Omega$ is computed for each mesh. Taking the conforming mesh as reference, the relative error $e_U = \frac{\|U - U_{conform}\|}{U_{conform}}$ is computed. The energy is slightly better computed for the adapted mesh but the method cannot be highlighted as the errors are similar.

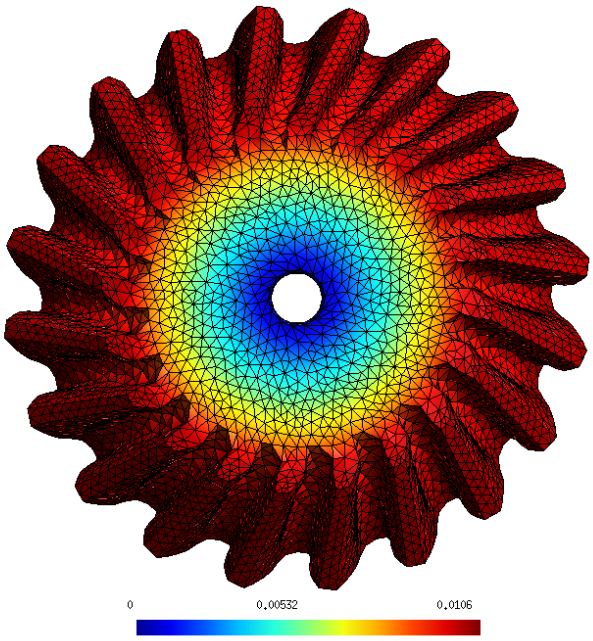


Figure 4.13 – Displacement norm on the deformed shape of the loaded gearing wheel represented with the conforming mesh.

Mesh	U	e_U
Cut	6,498.81	0.257
Adapted	6,825.03	0.219
Conforming	8,742.18	-

Table 4.2 – Elastic energy and error on elastic energy for the cut mesh, the anisotropic mesh and the conforming mesh of the gear.

4.5 Test case : engine bracket

A last test case is analysed. It consists in the loading of an engine bracket. The challenge with aircraft engine parts is to find a trade-off between performance requirements for strength and stiffness on one hand and size and weight on the other. This bracket has been optimized for both performance and weight. Additive manufacturing has recently emerged and gives the ability to produce parts of practically any shape, enabling weight savings compared to traditional manufacturing technologies. The geometry of the bracket is shown on figure 4.15 with an STL triangulation of its surface. As can be seen, the STL triangulation describes the surface of the geometry but contains low quality triangles not suited for a finite element computation. The loading case consists in fixing the

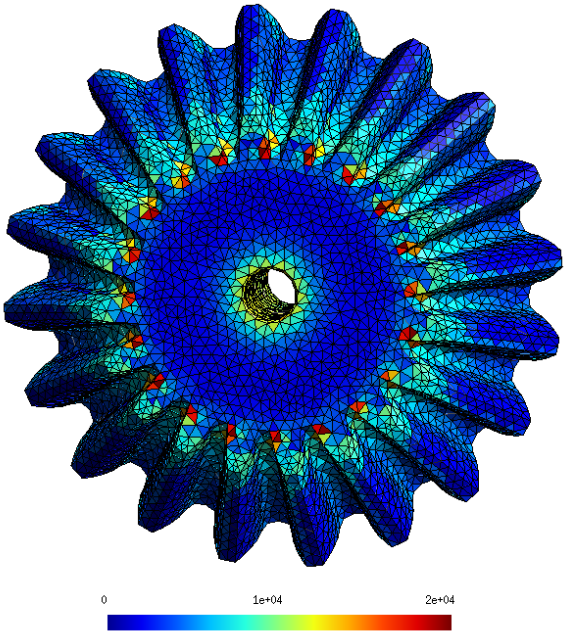


Figure 4.14 – Von Mises stress on the gearing wheel represented with the conforming mesh.

four holes by bolts and a vertical load is applied on the pins (red arrows).

The first step to analyse the test case is to obtain an implicit representation of the geometry. The levelset function of the geometry is the signed distance function to its surface. It is computed as the signed distance to the STL triangulation. Then a mesh is generated inside a box surrounding the geometry (see figure 4.16). An adaptation is performed to obtain smaller elements close to the surface. This enables to get a better representation of the geometry. This mesh is finally cut. The elements crossed by the interface are cut to create subelements and surface elements. The surface mesh can be seen on figure 4.17. To apply the boundary conditions on the pin and bolt surfaces, the surface elements corresponding to these surfaces have to be highlighted. Cylinder levelsets surrounding the holes are used to split the surface elements from the mesh.

Dirichlet boundary conditions are applied thanks to Lagrange multipliers on the four holes and the loading is finally applied on the pins. The elastic properties are applied on the computational domain. The problem is solved using the stabilized formulation with an iterative solver. The resulting displacements can be seen on figure 4.18.

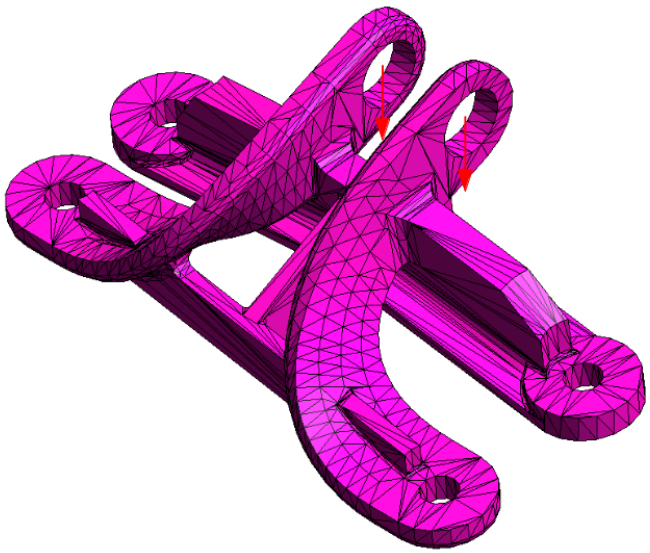


Figure 4.15 – STL triangulation of the engine bracket with loading on pins.

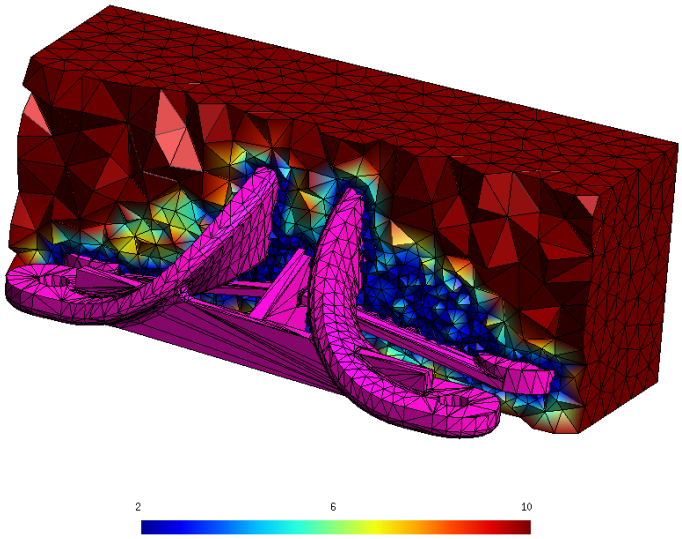


Figure 4.16 – Adapted mesh in a box surrounding the engine bracket and elements size.

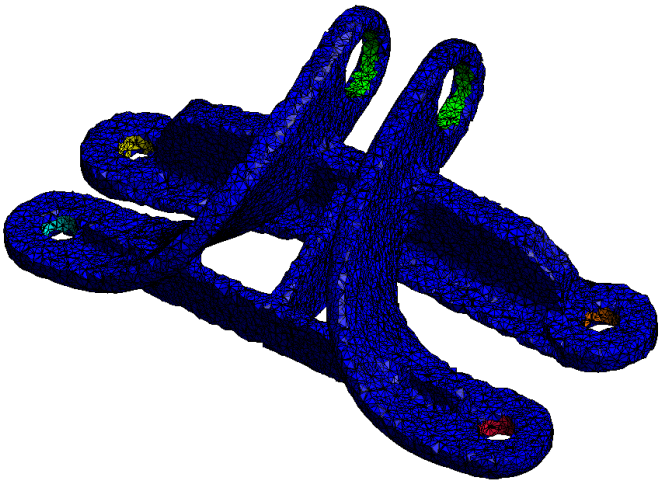


Figure 4.17 – Surface mesh of the bracket obtained from the cutting of the surrounding mesh and surfaces where the boundary conditions are applied.

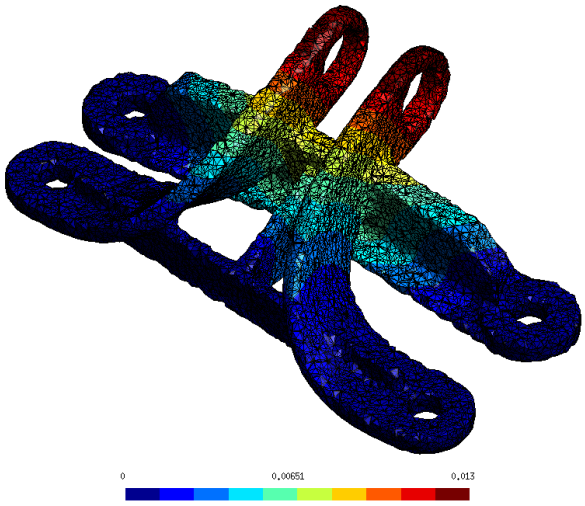


Figure 4.18 – Displacement of the bracket computed with Lagrange multipliers and the stabilized formulation.

The second method using anisotropic mesh adaptation close to the interface is used to perform a second computation based on the implicit representation of the bracket. An anisotropic mesh is generated in a box surrounding the bracket (see figure 4.19). The elements size in the normal direction to the surface is

prescribed as 1/10 of the bulk mesh size on the surface. The interior of the computational domain is then split by taking the elements connected to a vertex with negative sign. The surface of the computational domain can be seen on figure 4.20. To apply the boundary conditions, the surfaces of the pins and the bolts are needed. Cylinder levelsets surrounding these surfaces are used to split the elements from the surface mesh.

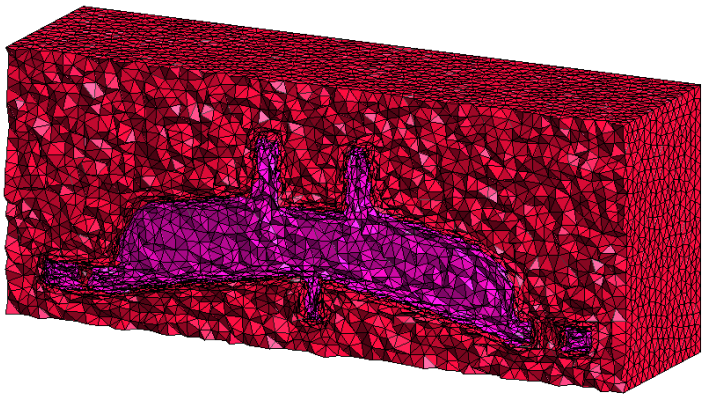


Figure 4.19 – Anisotropic adapted mesh close to the surface of the bracket in a surrounding box.

The computational domain is loaded. The Dirichlet boundary conditions are applied on the conformal surfaces and the standard finite element formulation is used to solve the problem with an iterative solver. The resulting displacements are shown on figure 4.21. The displacements are very similar to the ones obtained with the other method using the cutting of the elements and a stabilization.

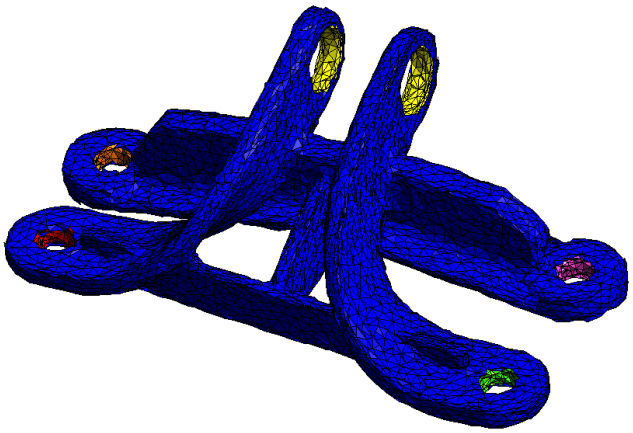


Figure 4.20 – Surface mesh of the computational domain obtained from the splitting of the anisotropic adapted mesh and surfaces where the boundary conditions are applied.

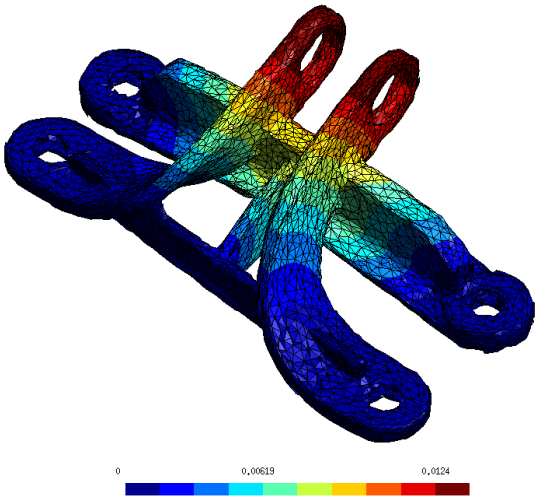


Figure 4.21 – Displacement of the bracket computed with the anisotropic adapted mesh.

4.6 Conclusion

To conclude, we can say that we have successfully extended the methods to linear elasticity. The stabilizing coefficient of the method based on a stabilizing term has been first adimensionalized. It has then been calibrated on a simple test case to obtain a rule of good practice that minimizes the discretization error. The extension to other problem formulations needs to calibrate again the stabilizing parameter. The second method based on anisotropic mesh adaptation does not depend on the problem formulation so its extension is straightforward. Both methods have shown optimal rates of convergence for a simple elastic problem. Yet the anisotropic adaptation leads to an increase in the number of elements of around 20% for an equivalent mesh size in the bulk.

Complex test cases of a gearing wheel and a bracket have been solved with both methods. The results obtained with both methods have been compared and produce similar results. An analysis of the meshing and computational time has been done. The method based on cut elements needs less time to create the mesh but is slower to compute the linear system because of the mixed formulation with the Lagrange multipliers. The best choice of method can be determined by the need to obtain a mesh rapidly or to solve multiple problems on the geometry.

Conclusion

Summary

In this thesis, a tool to describe the geometry using levelsets have first been presented. It enables to tackle the CAD validity problems by using signed distance functions to the surfaces of the geometry to describe the domain of computation. Distance functions can easily be obtained from different sources as analytical solutions or distances to a triangulation. This method has shown interesting results. Several levelsets can easily be combined with boolean operations. This enables to add or remove rapidly features to an existing geometry. This implicit representation is promising and could be used in general CAD softwares in a regular fashion in order to eliminate the problems of validity when trying to mesh the geometry.

Then two new approaches have been presented to handle the stability problem encountered when imposing Dirichlet boundary conditions on embedded interfaces. Both methods use the levelset representation of the geometry to benefit from its advantages.

The first method uses Lagrange multipliers and a stabilizing term. To obtain the geometry, the mesh is cut along the embedded interfaces. The integration in the cut elements is made more complex as it has to take into account only the part belonging to the computational domain. This method is intrusive as the finite element code has also to be modified to handle the Lagrange multipliers and the stabilizing term. A new stabilizing term has been presented and compared to an existing one. It has shown better results in terms of robustness and convergence, even though not being consistent. The stabilizing term contains a coefficient that has been adimensionalized and calibrated to minimize the discretization error for Laplace problems. The advantage of this first method is the simplicity of its implementation and a relative good robustness compared to other methods using Lagrange multipliers. Yet it still relies on a stabilizing term with a parameter that has to be well determined. If the stabilizing parameter is not large enough, the stabilizing term is insignificant compared to the problem terms and the problem remains unstable. While a stabilizing term with a too

large parameter overwhelms the problem terms and leads to a wrong solution.

The second method is based on a different approach. It uses a local mesh adaptation close to the embedded interfaces to obtain a nearly body-fitted mesh. The geometry is then obtained by splitting the adapted mesh along the interfaces. To retrieve an optimal rate of convergence, anisotropic mesh adaptation is needed with elements flattened in the normal direction to the interfaces. Anisotropic adaptation has the advantage of limiting the increase in the number of elements compared to isotropic adaptation. A positive point of this method is that the finite element formulation is not modified, enabling to use a standard finite element code to solve the problem. Yet it is based on arbitrary geometrical parameters to define the mesh adaptation and depends on an external mesh adapter that can be time costly to converge. Moreover, the anisotropic mesh containing elements with bad quality could not be fit to solve isotropic problems.

Both methods use mesh modifications in the vicinity of the embedded interfaces and add new degrees of freedom to the formulation. The method using Lagrange multipliers needs to cut the elements crossed by the embedded interfaces to integrate in the part belonging to the computational domain and adds Lagrange multipliers to the formulation. The method using mesh adaptation refines the mesh around the embedded interfaces, increasing the number of nodes. These mesh manipulations create slight modifications of the geometry. To cut the elements, the levelset function is interpolated inside the elements while the adapted mesh does not fit exactly the geometry. These geometry modifications were wanted to be avoided but are still present.

Finally, both methods are challenged on complex geometries in linear elasticity. A convergence study shows that both presented methods are optimally convergent with respect to the bulk mesh size. Yet the anisotropic mesh adaptation is subjected to an increase in the number of elements of around 20% in the vicinity of the interface. An analysis of the time of computation reveals that the mesh is created faster for the method using Lagrange multipliers but the linear system is solved faster for the method using mesh adaptation. The choice for the best method regarding the computational time depends on the need of a quick acquisition of the mesh or of the solving of several problems on the computational domain. A combination of both methods, using a slight isotropic mesh refinement close to the interface followed by a mesh cutting, results in a better representation of the geometry. Yet it still leads to a mixed formulation with Lagrange multipliers. Nitsche methods are nevertheless interesting as they lead to a robust formulation with a stabilizing parameter that has to be evaluated in advance by solving an eigenvalue problem or using bubble functions. A local computation is preferred as it can be called as a subroutine during the global assembly process.

Discussion and suggestions for future work

The method with a stabilizing term has been successfully applied to Laplace problems and linear elasticity. To solve *fluid dynamic* problems as advection-diffusion, the case of discontinuities at boundary layers has to be handled. A mesh refinement at the boundary is necessary. The method using mesh adaptation already refines the mesh in the normal direction to the boundary. But the bad quality of the elements at the interface leads to a wrong estimation of quantities such as the wall shear stress.

The methods have been applied to linear finite elements. *Higher order elements* are interesting as they enable to decrease the number of mesh elements. A part of the solution has been presented to cut high order elements using recursion in order to improve the integration. The method based on a stabilizing term can benefit from this approach as it reduces the number of elements to cut and improves the interpolation of the levelset function inside these elements. The method using mesh adaptation shows also benefits with high order elements as the anisotropy at the interface increases at high order.

As mentioned earlier, embedded interfaces are useful in *moving interface* problems to avoid remeshing at each step. Mesh adaptation can be advantageous in this case as the mesh adaptation is based on a local modification of the mesh. The adapted mesh at the next step can use the previous one as initial condition if the interface displacements are not too large. The method based on mesh cutting does not show any advantage in this case as the mesh has to be cut at each step.

To conclude, we can say that a conformal mesh is always preferred to avoid modifications of the finite element formulation or bad shaped elements. The presented methods attempt to simplify the mesh generation process by limiting human interventions. But the methods are still not robust enough to be used in commercial codes. Sensitive parameters are needed to perform the computation, geometrical parameters for the mesh adaptation and a stabilizing parameter for the stabilization method. As long as the robustness of these methods is not improved, the time devoted to mesh generation will remain an important part of the numerical analysis time. But we believe that these contributions are a building block for the path towards the full automation of mesh generation.

List of Figures

List of Tables

References

[1] Mark Ainsworth and J Tinsley Oden. *A posteriori error estimation in finite element analysis*, volume 37. John Wiley & Sons, 2011.

[2] F. Alauzet. Metric-based anisotropic mesh adaptation, 2010. `http://www-roc.inria.fr/gamma/Frederic.Alauzet/index.fr.html`.

[3] Grégoire Allaire, François Jouve, and Anca-Maria Toader. A level-set method for shape optimization. *Comptes Rendus Mathematique*, 334(12):1125–1130, 2002.

[4] 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.

[5] Ivo Babuška. *The finite element method with Lagrangian multipliers*, volume 20. 1973.

[6] J Andreas Bærentzen and Henrik Aanæs. Generating signed distance fields from triangle meshes. *Informatics and Mathematical Modeling, Technical University of Denmark, DTU*, 20, 2002.

[7] J Andreas Baerentzen and Henrik Aanaes. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, 2005.

[8] Helio J.C. Barbosa and Thomas J.R. Hughes. The finite element method with lagrange multipliers on the boundary: circumventing the babuska-brezzi condition. *Computer Methods in Applied Mechanics and Engineering*, 85(1):109–128, 1991.

[9] Pasko A.A. Belyaev A.G. and Kunii T.L. Ridges and ravines on implicit surfaces. In *Computer Graphics International 1998*, pages 530–535, 1998.

[10] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. ACM.

[11] F. Brezzi and J. Pitkäranta. On the stabilization of finite element approximations of the stokes equations. In Wolfgang Hackbusch, editor, *Efficient Solutions of Elliptic Systems*, volume 10 of *Notes on Numerical Fluid Mechanics*, pages 11–19. Vieweg+Teubner Verlag, 1984.

[12] Erik Burman and Peter Hansbo. Fictitious domain finite element methods using cut elements: I. a stabilized lagrange multiplier method. *Computer Methods in Applied Mechanics and Engineering*, 199(41–44):2680 – 2686, 2010.

[13] Éric Béchet, Nicolas Moës, and Barbara Wohlmuth. A stable lagrange multiplier space for stiff interface conditions within the extended finite element method. *International Journal for Numerical Methods in Engineering*, 78(8):931–954, 2009.

[14] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[15] Vicent Caselles, Francine Catté, Tomeu Coll, and Françoise Dibos. A geometric model for active contours in image processing. *Numerische mathematik*, 66(1):1–31, 1993.

[16] Dominique Chapelle and Klaus-Jürgen Bathe. The inf-sup test. *Computers & structures*, 47(4):537–545, 1993.

[17] Jack Chessa, Patrick Smolinski, and Ted Belytschko. The extended finite element method (xfem) for solidification problems. *International Journal for Numerical Methods in Engineering*, 53(8):1959–1977, 2002.

[18] Alexandra Claisse, Vincent Ducrot, and Pascal Frey. Levelsets and anisotropic mesh adaptation. 2008.

[19] C. Dobrzynski and P. Frey. Anisotropic Delaunay mesh adaptation for unsteady simulations. *Proceedings of the 17th International Meshing Roundtable*, pages 177–194, 2008.

[20] J.E. Dolbow and L.P. Franca. Residual-free bubbles for embedded dirichlet problems. *Computer Methods in Applied Mechanics and Engineering*, 197(45–48):3751 – 3759, 2008.

[21] John Dolbow, Nicolas Moës, and Ted Belytschko. An extended finite element method for modeling crack growth with frictional contact. *Computer Methods in Applied Mechanics and Engineering*, 190(51):6825–6846, 2001.

[22] Anand Embar, John Dolbow, and Isaac Harari. Imposing dirichlet boundary conditions with nitsche's method and spline-based finite elements. *International Journal for Numerical Methods in Engineering*, 83(7):877–898, 2010.

[23] P. J. Frey and P.-L. George. *Mesh Generation. Application to Finite Elements*. Hermès, Paris, 2000.

[24] R. Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(22):632–658, 2005.

[25] Michael Griebel and Marc Alexander Schweitzer. A particle-partition of unity method part v: boundary conditions. In *Geometric Analysis and Nonlinear Partial Differential Equations*, pages 519–542. Springer, 2003.

[26] A Guezlec. "meshsweeper": dynamic point-to-polygonal mesh distance and applications. *IEEE Transactions on visualization and computer graphics*, 7(1):47–61, 2001.

[27] Wagdi G Habashi, Julien Dompierre, Yves Bourgault, Djaffar Ait-Ali-Yahia, Michel Fortin, and Marie-Gabrielle Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent cfd. part i: general principles. *International Journal for Numerical Methods in Fluids*, 32(6):725–744, 2000.

[28] J. Haslinger and Y. Renard. A new fictitious domain approach inspired by the extended finite element method. *SIAM Journal on Numerical Analysis*, 47(2):1474–1499, 2009.

[29] M. Hautefeuille, C. Annavarapu, and J.E. Dolbow. Robust imposition of dirichlet boundary conditions on embedded surfaces. *International Journal for Numerical Methods in Engineering*, 90(1):40–64, 2011.

[30] F. Hecht. Bamg: Bidimensional anisotropic mesh generator, 2006. `http://www.freefem.org/ff++`.

[31] Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39):4135–4195, 2005.

[32] TJR Hughes, JA Cottrell, and Y Bazilevs. "consider a spherical cow"–conservation of geometry in analysis: Implications for computational methods in engineering. In *IMA Hot Topics Workshop: Compatible Spatial Discretizations for Partial Differential Equations*, 2004.

[33] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4):33, 2013.

[34] H Ji and JE Dolbow. On strategies for enforcing interfacial constraints and evaluating jump conditions with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 61(14):2508–2535, 2004.

[35] Gustav Kirsch. *Die theorie der elastizität und die bedürfnisse der festigkeitslehre*. Springer, 1898.

[36] Gerard Meurant. *Computer solution of large linear systems*, volume 28. Elsevier, 1999.

[37] N. Moës, M. Cloirec, P. Cartraud, and J.F. Remacle. A computational approach to handle complex microstructure geometries. *Computer Methods in Applied Mechanics and Engineering*, 192(28):3163–3177, 2003.

[38] David M Mount and Sunil Arya. Ann: Library for approximate nearest neighbour searching. 1998.

[39] Hashem M. Mourad, John Dolbow, and Isaac Harari. A bubble-stabilized finite element method for dirichlet constraints on embedded interfaces. *International Journal for Numerical Methods in Engineering*, 69(4):772–793, 2007.

[40] N. Moës, A. Gravouil, and T. Belytschko. Non-planar 3d crack growth by the extended finite element and level sets—part i: Mechanical model. *International Journal for Numerical Methods in Engineering*, 53(11):2549–2568, 2002.

[41] Nicolas Moës, Éric Béchet, and Matthieu Tourbier. Imposing dirichlet boundary conditions in the extended finite element method. *International Journal for Numerical Methods in Engineering*, 67(12):1641–1669, 2006.

[42] Nicolas Moës, John Dolbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999.

[43] J Nitsche. Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 36, pages 9–15. Springer, 1971.

[44] Elin Olsson and Gunilla Kreiss. A conservative level set method for two phase flow. *Journal of computational physics*, 210(1):225–246, 2005.

[45] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[46] Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Comput. Graph. Appl.*, 12(1):65–71, January 1992.

[47] Robert M Pinkebtom. The characteristics of 78 related airfoil sections from tests in the variable-density wind tunnel. 1933.

[48] Dieu-Linh Quan, Thomas Toulorge, Gaëtan Bricteux, Jean-François Remacle, and Emilie Marchandise. Anisotropic adaptive nearly body-fitted meshes for cfd. *Engineering with Computers*, 30(4):517–533, 2014.

[49] Dieu-Linh Quan, Thomas Toulorge, Emilie Marchandise, Jean-François Remacle, and Gaëtan Bricteux. Anisotropic mesh adaptation with optimal convergence for finite elements using embedded geometries. *Computer Methods in Applied Mechanics and Engineering*, 268:65–81, 2014.

[50] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[51] Jessica D Sanders, John E Dolbow, and Tod A Laursen. On methods for stabilizing constraints over enriched interfaces in elasticity. *International Journal for Numerical Methods in Engineering*, 78(9):1009, 2009.

[52] Carlo H Séquin. Procedural spline interpolation in unicubix. 1987.

[53] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[54] Jason F Shepherd and Chris R Johnson. Hexahedral mesh generation constraints. *Engineering with Computers*, 24(3):195–213, 2008.

[55] Bernard Sonon and Thierry J Massart. A level-set based representative volume element generator and xfem simulations for textile and 3d-reinforced composites. *Materials*, 6(12):5568–5592, 2013.

[56] Rolf Stenberg. On some techniques for approximating boundary conditions in the finite element method. *Journal of Computational and applied Mathematics*, 63(1):139–148, 1995.

[57] M Stolarska, DL Chopp, Nicolas Moës, and Ted Belytschko. Modelling crack growth by level sets in the extended finite element method. *International journal for numerical methods in Engineering*, 51(8):943–960, 2001.

[58] Bjarne Stroustrup. *The C++ programming language*. Pearson Education India, 1995.

[59] Wikipedia. Constructive solid geometry, November 2015. `https://en.wikipedia.org/wiki/Constructive_solid_geometry`.

[60] P Wriggers and Giorgio Zavarise. A formulation for frictionless contact problems using a weak form introduced by nitsche. *Computational Mechanics*, 41(3):407–420, 2008.