



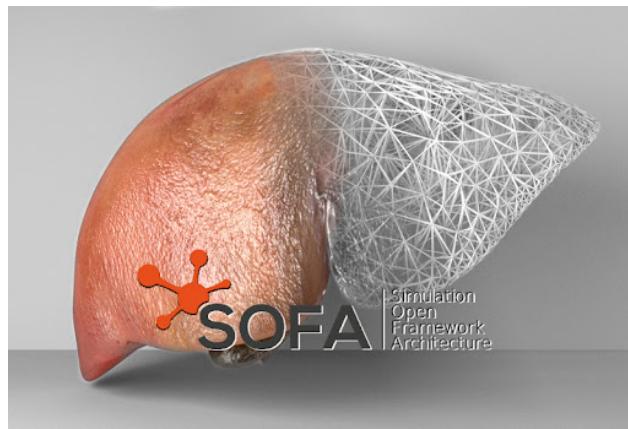
---

# Développement d'un plugin SOFA " $\phi$ -FEM" : une méthode éléments finis non-conforme adaptée à la chirurgie assistée par ordinateur

Killian Vuillemot

*Encadrants : Michel Duprez et Stéphane Cotin  
Enseignant référent : Franz Chouly*

---



*Du 11/04/2022 au 31/08/2022*

## Remerciements

Je tiens tout d'abord à remercier Michel Duprez<sup>1</sup> pour son accompagnement et son aide durant ce stage. Je le remercie également de m'avoir offert la possibilité de travailler sur ce sujet à ses côtés ainsi qu'aux côtés de Vanessa Lleras<sup>2</sup> et d'Alexei Lozinski<sup>3</sup>. J'ai une nouvelle fois pu apprendre énormément grâce à ce sujet de recherche.

Je remercie également Stéphane Cotin, pour son aide et ses conseils lorsque j'en ai eu besoin.

Je tiens également à remercier Franz Chouly<sup>4</sup>, notamment pour m'avoir mis en relation avec Michel Duprez l'année dernière, ce qui m'a permis de travailler sur des thèmes de recherches extrêmement enrichissants, mais également pour son accompagnement au cours de cette deuxième année de Master.

Je remercie enfin Sidaty El Hadramy<sup>5</sup> pour son aide mais aussi pour sa gentillesse et son accompagnement lors de nos journées passées à chercher des solutions à nos différents problèmes.

- 
1. <http://mduprez.perso.math.cnrs.fr/>
  2. <https://vanessalleras.wixsite.com/lleras>
  3. <https://imb.univ-fcomte.fr/Lozinski-Alexei>
  4. <http://fchouly.perso.math.cnrs.fr/>
  5. <https://mimesis.inria.fr/members/sidaty-el-hadramy/>

## Table des matières

<b>1 Introduction</b>	<b>5</b>
<b>2 Contexte du stage</b>	<b>7</b>
2.1 Présentation du laboratoire . . . . .	7
2.2 Objectifs du stage . . . . .	7
<b>3 Méthodes éléments finis et <math>\phi</math>-FEM</b>	<b>9</b>
3.1 Le problème de Poisson . . . . .	10
3.1.1 Traitement avec une méthode standard . . . . .	10
3.1.2 Traitement avec $\phi$ -FEM : le cas de conditions de Dirichlet pures . . . . .	13
3.1.3 Le cas de conditions mixtes . . . . .	16
3.2 Problème de Poisson non-linéaire . . . . .	21
3.2.1 Méthode standard . . . . .	22
3.2.2 Traitement avec $\phi$ -FEM . . . . .	23
<b>4 Le cas de l'hyperélasticité</b>	<b>25</b>
4.1 Une introduction à l'élasticité linéaire . . . . .	25
4.1.1 Méthode éléments finis standard . . . . .	26
4.1.2 Le schéma $\phi$ -FEM pour l'élasticité linéaire . . . . .	27
4.2 Les déformations et tenseurs essentiels en hyperélasticité . . . . .	28
4.3 Matériaux hyperélastiques . . . . .	32
4.3.1 Modèle de Saint-Venant-Kirchhoff . . . . .	33
4.3.2 Modèle néo-hookéen . . . . .	33
4.3.3 Modèle d'Ogden . . . . .	33
4.3.4 Modèle de Gent . . . . .	34
4.4 Résolution numérique . . . . .	34
4.4.1 Méthode standard . . . . .	35
4.4.2 Construction des schémas $\phi$ -FEM . . . . .	36
4.5 Résultats numériques sur des domaines simples . . . . .	38
4.5.1 Le cas 2D . . . . .	38
4.5.2 Le cas 3D : une sphère . . . . .	40
4.6 Application à un problème réel : la déformation du foie . . . . .	41
4.6.1 Construction de la level-set . . . . .	42
4.6.2 Implémentation et résultats numériques . . . . .	46
<b>5 L'implémentation de <math>\phi</math>-FEM au sein de SOFA</b>	<b>47</b>
5.1 L'environnement utilisé . . . . .	47
5.2 Les débuts de l'implémentation . . . . .	48
<b>6 Conclusion</b>	<b>55</b>
<b>A Quelques résultats pour le problème de Poisson</b>	<b>57</b>
<b>B Codes <i>FEniCSX</i></b>	<b>59</b>
<b>C Codes modifiés dans Caribou</b>	<b>64</b>

## Liste des figures

1	Deux exemples de situations pour des méthodes non-conformes.	5
2	Membres des équipes MLMS et MIMESIS.	7
3	Représentation des normales extérieures unitaires sur une sphère.	10
4	Maillages d'ellipse et d'ellipsoïde.	11
5	Éléments de référence 2D.	12
6	Exemples de fonctions de base.	13
7	Représentation de la situation.	15
8	Deux situations de partitionnement de la frontière.	17
9	Représentation des ensembles $\mathcal{F}_h^{\Gamma_N}$ et $\mathcal{F}_h^{\Gamma_D}$ .	20
10	Problème de Poisson avec conditions mixtes.	22
11	Problème de Poisson non-linéaire.	25
12	Élasticité linéaire avec des conditions mixtes.	28
13	Déformations du maillage.	29
14	Différences entre la configuration initiale et la configuration déformée ([4]).	30
15	Hyperélasticité sur des domaines en 2 dimensions.	39
16	Déformation d'une ellipse.	40
17	Cas test d'une sphère hyperélastique.	41
18	Maillage surfacique de foie.	42
19	Temps de calcul pour la construction de $\phi$ .	43
20	Maillages de sphère reconstruits avec la fonction level-set.	44
21	Erreur de construction de $\phi$ .	44
22	Maillages de foie reconstruits avec la fonction level-set.	45
23	Cas test du foie.	46
24	Hyperélasticité pour le cas du foie.	47
25	Résultats obtenus pour la chute d'une balle élastique.	48
26	Résultats obtenus pour la chute d'un foie sur une poutre élastique.	49
27	Deux exemples de grilles fictives.	50
28	Cas considéré pour la validation des méthodes.	52
29	Différentes constructions à l'aide de la classe <code>LevelSet</code> .	54
30	Problème de Poisson Dirichlet.	57
31	Problème de Poisson Neumann.	57
32	Problème de Poisson avec conditions mixtes.	58

## Liste des programmes

1	Sorties obtenues pour le cas représenté à la figure28.	51
2	Bindings utilisés.	53
3	Sorties obtenues.	53
4	Sorties obtenues.	53
5	Modification de la fonction <code>locate_entities</code> .	59
6	Création de la fonction <code>locate_bc_entities</code> .	60
7	Création de la fonction <code>locate_facets</code> .	62
8	Fonction rentrant tous les cotés d'une cellule en deux dimensions.	64
9	Fonction rentrant toutes les faces d'une cellule en trois dimensions.	65
10	Fonction rentrant les indices des cellules en fonction du type.	65
11	Fonction rentrant les indices des faces d'une cellule.	66
12	Fonction permettant de construire $\mathcal{F}_h^{\Gamma}$ .	67
13	Fonction permettant d'évaluer la level-set en un noeud de la grille.	68
14	Fonction permettant d'évaluer la level-set à chaque noeud d'une cellule de la grille.	68
15	Fonction permettant d'évaluer la level-set à chaque noeud de la grille.	69
16	Code permettant de créer différentes fonctions level-set.	69

## Liste des algorithmes

1	Récupération des indices des faces (cas 3D).	51
2	Construction de $\mathcal{F}_h^{\Gamma}$ .	52

## Liste des tableaux

1	Table des tenseurs utilisés.	30
2	Table des changements de variables.	31

## 1 Introduction

Les modèles éléments finis sont de plus en plus utilisés dans le cadre médical afin de simuler les mécanismes des tissus mous et élastiques du corps humain (par exemple le foie). Les modèles déjà existants sont des outils très importants dans ce domaine. Ils permettent notamment de développer de nouveaux dispositifs médicaux et d'améliorer les stratégies de planification et d'assistance chirurgicale. L'idée est donc que ces méthodes puissent être utilisées par des logiciels de simulation en temps réel. Au cours de ce rapport, nous parlerons de SOFA<sup>6</sup> [13], logiciel très utilisé au sein de l'équipe MIMESIS<sup>7</sup> puisque développé par certains de ces membres.

Cependant, l'un des principaux problèmes des méthodes éléments finis standards est la prise en compte des géométries complexes. En effet, la génération de maillages suffisamment précis pour approcher parfaitement la géométrie de domaines tels que des organes (le foie ou la langue par exemple) est numériquement très coûteuse pour obtenir des résultats réalistes. Cela devient encore plus coûteux lorsque l'on considère le cas de domaines évoluant au cours du temps : les calculs nécessaires à la création de ces maillages sont bien trop complexes pour être utilisés lors de simulations en temps réel. Différentes méthodes éléments finis non conformes ont été développées afin d'éviter la génération de ces maillages. On retrouve par exemple les méthodes de « Frontières immersées » (Immersed Boundary Method) [17] ou « Domaines Fictifs » (Fictitious domains) [15].

L'un des défauts majeurs des premières méthodes développées est leur traitement des conditions de bord qui générait une perte de précision sur les résultats. Pour faire face à ces problèmes, de nouvelles méthodes non conformes ont été développées. Ces méthodes plus précises, telles que CutFEM [6] ou XFEM [18] reposent sur l'idée d'introduire un maillage de fond plus grand que le domaine réel puis de construire un sous-maillage contenant seulement les cellules internes et celles en intersection avec la frontière réelle du domaine, comme représenté à la figure 1(a) issue de [10]. Une autre approche connue sous le nom de « méthode de frontière décalée » (Shifted boundary method) (voir [2, 3]) consiste cette fois à réduire légèrement le domaine considéré, en sélectionnant seulement les cellules complètement internes au domaine. La frontière considérée est alors la frontière du maillage construit, comme représenté à la figure 1(b) issue de [2].

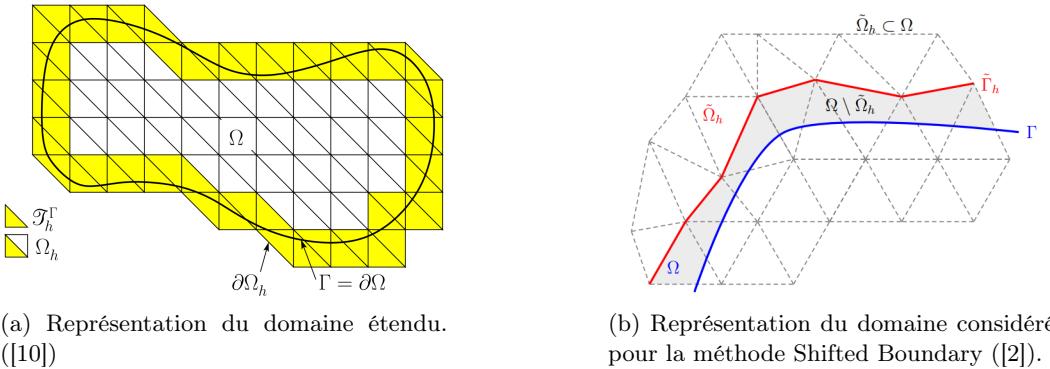


FIGURE 1 – Deux exemples de situations pour des méthodes non-conformes.

Nous nous intéresserons dans ce rapport à une autre méthode éléments finis non conforme, appelée  $\phi$ -FEM. Cette méthode, introduite dans [12, 8, 7] est appelée ainsi en raison de son utilisation d'une fonction level-set  $\phi$  pour définir le domaine physique  $\Omega$  et sa frontière  $\Gamma$ . L'intérêt de  $\phi$ -FEM est notamment de ne pas nécessiter des calculs trop complexes numériquement, puisqu'il n'y aura pas d'intégrales à calculer sur des éléments coupés, contrairement à CutFEM par exemple, tout en offrant un ordre de convergence optimal (i.e. même ordre de convergence qu'une méthode standard) et en étant relativement simple à implémenter. Cette méthode a été introduite premièrement dans le cadre de la résolution du problème de Poisson, pour des conditions de bord de type Dirichlet (voir [12]), avant d'être étendue à des conditions de Neumann (voir [8]). Par la suite, dans [7], la méthode a été présentée pour résoudre différents problèmes mécaniques, notamment des problèmes d'élasticité linéaire ainsi que des problèmes de transfert de chaleur. Enfin, dans [9], la méthode a été présentée pour la résolution du problème de Stokes.

6. <https://www.sofa-framework.org>

7. <https://mimesis.inria.fr/>

L'objectif principal de ce rapport sera de créer un lien entre SOFA la méthode  $\phi$ -FEM. En effet, l'un des objectifs sur le long terme est d'implémenter cette méthode au sein de SOFA, afin notamment de simuler des déformations de matériaux hyperélastiques. Nous allons donc au cours de ce rapport détailler les différentes étapes nécessaires et terminer ce dernier en présentant les premières étapes de l'implémentation de  $\phi$ -FEM dans SOFA. Pour cela, il sera nécessaire dans un premier temps d'introduire cette méthode ainsi que la résolution de problèmes à l'aide d'une méthode éléments finis standard. Nous considérerons tout d'abord deux problèmes déjà traités dans [12] et [8] : le problème de Poisson pour des conditions de Dirichlet puis pour des conditions de Neumann. Cela permettra ainsi de se familiariser avec les deux méthodes. Par la suite, nous présenterons un nouveau schéma  $\phi$ -FEM permettant de traiter le cas de conditions mixtes (des conditions de Dirichlet seront imposées sur une partie de la frontière et des conditions de Neumann sur une autre partie), une nouvelle fois pour le problème de Poisson.

Ensuite, nous introduirons une méthode  $\phi$ -FEM pour la résolution de **problèmes non-linéaires**. Pour cela, nous considérerons une nouvelle fois le cas de conditions de bord mixtes, sur une version légèrement modifiée du problème de Poisson, introduisant des non-linéarités dans l'idée du problème introduit dans [14]. Cela nous permettra alors de comparer la méthode avec le cadre linéaire et d'introduire le schéma  $\phi$ -FEM dans un premier temps sur un problème non-linéaire assez simple. Le schéma introduit ici sera essentiel. En effet, comme nous l'avons dit, l'objectif par la suite sera d'étendre la méthode  $\phi$ -FEM à la résolution de problèmes hyperélastiques et donc à la résolution de problèmes non-linéaires.

Une fois cela fait, nous nous intéresserons à la déformation de **matériaux hyperélastiques**. Pour cela, nous commencerons par présenter quelques aspects théoriques liés à l'élasticité linéaire, ainsi qu'une modification du schéma  $\phi$ -FEM pour les conditions mixtes, introduit dans [7]. Cela nous permettra alors d'introduire différents points essentiels permettant de comprendre plus aisément le cas plus complexe de l'hyperélasticité. Une fois le sujet de l'élasticité linéaire introduit, nous aborderons donc les grandes déformations en hyperélasticité. Pour cela, nous aurons besoin d'introduire différents aspects théoriques. Une fois la théorie présentée, nous verrons comment résoudre numériquement de tels problèmes avec une méthode standard ainsi qu'avec  $\phi$ -FEM. Enfin, nous appliquerons notamment la méthode  $\phi$ -FEM à un problème complexe : la déformation du foie.

La dernière partie sera consacrée au début de l'implémentation de  $\phi$ -FEM au sein de **SOFA**. Nous présenterons donc le logiciel et l'un des plugin que nous utiliserons à travers plusieurs exemples. Ensuite, nous décrirons brièvement les premières idées de cette implémentation ainsi que le travail déjà réalisé. Nous illustrerons alors le fonctionnement des méthodes créées avec quelques exemples.

Une autre partie de ce stage a été consacrée à l'analyse du schéma  $\phi$ -FEM pour résoudre l'équation de la chaleur, en collaboration avec Vanessa Lleras, Aleixei Lozinski et Michel Duprez. Ce travail n'a pas été inclus dans ce rapport mais une première version du document en cours est disponible à l'adresse <https://hal.archives-ouvertes.fr/hal-03685445>.



FIGURE 2 – Membres des équipes MLMS et MIMESIS.

## 2 Contexte du stage

### 2.1 Présentation du laboratoire

Ce stage a été réalisé au sein des équipes MIMESIS<sup>8</sup> et MLMS<sup>9</sup>. Présentons brièvement ces deux équipes. L'équipe Inria MIMESIS travaille sur différents problèmes dans les domaines du calcul scientifique, de l'acquisition de données, du machine learning et du contrôle dans un but : la création de jumeaux numériques d'organes en temps réel. Cela a notamment pour applications l'entraînement pré-opératoire ou bien le guidage en temps réel lors d'interventions chirurgicales complexes.

L'équipe MLMS (Machine Learning, Modélisation et Simulation) est une équipe du laboratoire ICube, rattaché à l'Université de Strasbourg. Elle s'intéresse aux données, modèles et simulations pour la science médicale et le mouvement humain. Les membres de cette équipe sont des neuroscientifiques, des bio-mécaniciens, des informaticiens ou encore des mathématiciens. Tous ses membres travaillent dans le but de développer des modèles avec comme application principale la chirurgie assistée par ordinateur. L'équipe MLMS accueille l'équipe-projet Inria MIMESIS, ce qui permet un travail en forte collaboration entre les équipes.

Les équipes MIMESIS et MLMS s'intéressent donc à la simulation en temps réel, avec pour objectif de lier la modélisation mathématique et la médecine clinique. L'idée est de faire évoluer la gestion de certains problèmes médicaux, notamment en utilisant des techniques de simulation en temps-réel, avec *SOFA*<sup>10</sup> ([13]) par exemple, ainsi que des modèles spécifiques au patient considéré à l'inverse de modèles "génériques". C'est dans ces deux domaines que la méthode  $\phi$ -FEM intervient. Les équipes MIMESIS et SOFA étant situées au même endroit, elles peuvent travailler en forte collaboration, l'équipe MIMESIS aidant à l'amélioration et au développement du logiciel ainsi qu'à l'amélioration ou à la création de plugins pour SOFA.

### 2.2 Objectifs du stage

Pour ce stage, différents objectifs étaient donnés. En particulier,

- apprendre à utiliser SOFA et SofaPython ,
- apprendre à utiliser Caribou ainsi que SofaCaribou ,
- comprendre le fonctionnement du code source de Caribou et de SofaCaribou ,
- travailler sur la théorie des problèmes hyperélastiques ,
- développer des schémas  $\phi$ -FEM pour :
  - i) le problème de Poisson avec des conditions mixtes ,
  - ii) des problèmes non-linéaires ,
  - iii) des problèmes hyperélastiques ,
- créer une fonction level-set pour tout domaine ,

8. <https://mimesis.inria.fr/>

9. <https://mlms.icube.unistra.fr/index.php/Presentation>

10. <https://www.sofa-framework.org/>

- débuter l'implémentation de  $\phi$ -FEM dans Caribou.

Toutes ces tâches ont fait intervenir de nombreux domaines de connaissances tels que

- i) les équations aux dérivées partielles : équation de Poisson, équation de la chaleur, équations élastiques, etc ,
- ii) l'analyse numérique : étude de convergence ,
- iii) le calcul scientifique : implémentation des différents schémas en python ,
- iv) la programmation : en C++, SOFA, python .

---

**Github repository.** L'ensemble des codes utilisés pour les simulations et représentations graphiques réalisées tout au long de ce rapport sont disponibles à l'adresse [https://github.com/KVuillemot/Stage\\_Master\\_2](https://github.com/KVuillemot/Stage_Master_2). Le code source modifié utilisé dans la dernière section est lui disponible à l'adresse [https://github.com/KVuillemot/caribou/tree/Phi\\_FEM](https://github.com/KVuillemot/caribou/tree/Phi_FEM).

#### Contributions sur $\phi$ -FEM :

- Stéphane Cotin, Michel Duprez, Vanessa Lleras, Alexei Lozinski and **Killian Vuillemot** :  $\phi$ -FEM : an efficient simulation tool using simple meshes for problems in structure mechanics and heat transfer (2021, accepté),
- Michel Duprez, Vanessa Lleras, Alexei Lozinski and **Killian Vuillemot** : An Immersed Boundary Method by Phi-FEM approach to solve the heat equation (soumis),
- Michel Duprez, Vanessa Lleras, Alexei Lozinski and **Killian Vuillemot** : A  $\phi$ -FEM approach to solve the Poisson problem with mixed boundary conditions (en préparation),
- Sidaty El Hadramy, **Killian Vuillemot**, Michel Duprez and Stéphane Cotin : Fast and accurate liver digital twin using  $\phi$ -FEM and Neural Networks (en préparation).

### 3 Méthodes éléments finis et $\phi$ -FEM

Avant d'introduire le sujet principal de ce rapport, il est essentiel de faire un bref rappel d'analyse fonctionnelle. Pour cela, nous donnerons ici trois définitions essentielles : les définitions des espaces et des normes qui seront majoritairement considérés durant ce rapport. Soit  $\Omega$  un ouvert de  $\mathbb{R}^d$  ( $d = 1, 2, 3$ ), de frontière  $\Gamma$ . Définissons premièrement les espaces de Lebesgue, notés  $L^p(\Omega)$ .

**Définition 3.1** (Espaces  $L^p(\Omega)$ ). L'espace  $L^p(\Omega)$  est défini par :

$$L^p(\Omega) := \left\{ u : \Omega \rightarrow \mathbb{R} \mid \int_{\Omega} u^p dv < +\infty \right\}.$$

En particulier, pour  $l = 2$ ,

$$L^2(\Omega) := \left\{ u : \Omega \rightarrow \mathbb{R} \mid \int_{\Omega} u^2 dv < +\infty \right\},$$

est l'espace des fonctions de carré intégrable.

**Définition 3.2** (Espaces de Sobolev). Soit  $p \in [1, +\infty]$  et soit  $k \in \mathbb{N}$ . L'espace de Sobolev  $W^{k,p}(\Omega)$  est défini par

$$W^{k,p}(\Omega) = \{u \in L^2(\Omega) \mid \forall \alpha \text{ tel que } |\alpha| \leq k, D^\alpha u \in L^p(\Omega)\}, \quad (1)$$

où  $L^p$  est un espace de Lebesgue, et  $D^\alpha u$  est une dérivée de  $u$  au sens des distributions.

Nous pouvons alors munir ces espaces de la norme  $\|\cdot\|_{W^{k,p}}$ , définie par

$$\|u\|_{W^{k,p}} = \begin{cases} \left( \sum_{|\alpha| \leq k} \|D^\alpha u\|_{L^p}^p \right)^{1/p} & \text{si } 1 \leq p < +\infty, \\ \max_{|\alpha| \leq k} \|D^\alpha u\|_{L^\infty} & \text{si } p = +\infty. \end{cases} \quad (2)$$

Un cas particulier d'espace de Sobolev qui nous intéressera particulièrement ici est l'espace fonctionnel linéaire suivant.

**Définition 3.3** (Espaces  $H^1(\Omega)$  et  $H_0^1(\Omega)$ ). L'espace fonctionnel linéaire  $H^1(\Omega)$  est défini par :

$$H^1(\Omega) := \{u \in L^2(\Omega) \mid \nabla u \in L^2(\Omega)^d\},$$

muni du produit scalaire  $\langle u, v \rangle_{H^1(\Omega)}$ , défini par :

$$\langle u, v \rangle_{H^1(\Omega)} = \int_{\Omega} uv + \nabla u \cdot \nabla v, \quad \forall u, v \in H^1(\Omega).$$

Soit également l'espace  $H_0^1(\Omega)$  par

$$H_0^1(\Omega) := \{u \in H^1(\Omega) \mid u = 0 \text{ sur } \Gamma\}.$$

*Remarque 3.1.* Ces définitions impliquent ainsi

$$\|u\|_{H^1(\Omega)}^2 = \|u\|_{L^2(\Omega)}^2 + \|\nabla u\|_{L^2(\Omega)}^2. \quad (3)$$

*Remarque 3.2.* Dorénavant, nous noterons  $\|\cdot\|_{1,\Omega}$  la norme  $H^1$  sur  $\Omega$ ,  $\|\cdot\|_{0,\Omega}$  la norme  $L^2$  sur  $\Omega$  et  $|\cdot|_{1,\Omega}$  la semi-norme  $H^1$  sur  $\Omega$  définie par  $|\cdot|_{1,\Omega} = \|\nabla \cdot\|_{0,\Omega}$ .

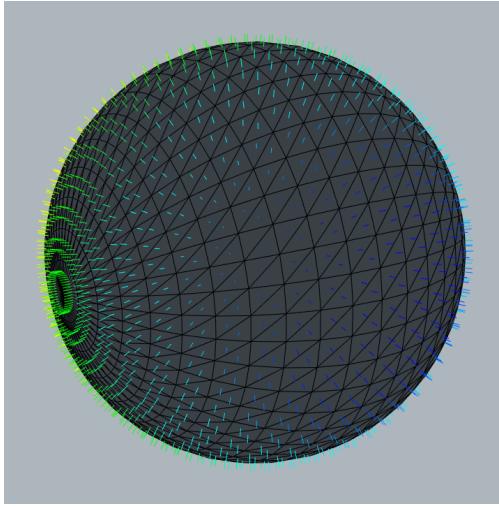


FIGURE 3 – Représentation des normales extérieures unitaires sur une sphère.

### 3.1 Le problème de Poisson

Nous allons dans un premier temps nous intéresser à la résolution numérique d'un problème très connu en analyse des équations aux dérivées partielles. Ce problème, assez simple en apparence sera ici utilisé afin d'introduire et de présenter la méthode éléments finis classique ainsi que la méthode  $\phi$ -FEM. Il semblait en effet important de présenter les deux méthodes de résolution qui nous intéresseront par la suite, en les appliquant premièrement à des problèmes plus simples que ceux que nous considérerons par la suite.

Soit  $\Omega$ , un ouvert de  $\mathbb{R}^d$ , avec  $d = 1, 2, 3$ , de frontière  $\Gamma$ . Soient  $\Gamma_N$  et  $\Gamma_D \neq \emptyset$  telles que  $\Gamma_N \cup \Gamma_D = \Gamma$  et  $\Gamma_N \cap \Gamma_D = \emptyset$ . Soient  $f \in L^2(\Omega)$  la force appliquée à  $\Omega$  et  $n$ , la normale unitaire extérieure au domaine  $\Omega$  (voir figure 3).

La frontière  $\Gamma$  est ici décomposée en deux parties, correspondant à des conditions de bord différentes. Sur la première partie,  $\Gamma_D$ , seront imposées des conditions de Dirichlet (dites essentielles). Les conditions de Dirichlet au bord permettent d'imposer un déplacement à au bord. Sur l'autre partie de la frontière,  $\Gamma_N$ , seront imposées des conditions de Neumann, dites conditions essentielles. Ces conditions font intervenir la dérivée normale de la solution, et imposer des conditions de Neumann revient à imposer une force externe à la solution.

Le problème est alors de trouver une fonction  $u$  telle que

$$\begin{cases} -\Delta u &= f, & \text{dans } \Omega, \\ u &= u_D, & \text{sur } \Gamma_D, \\ \nabla u \cdot n &= g, & \text{sur } \Gamma_N, \end{cases} \quad (4)$$

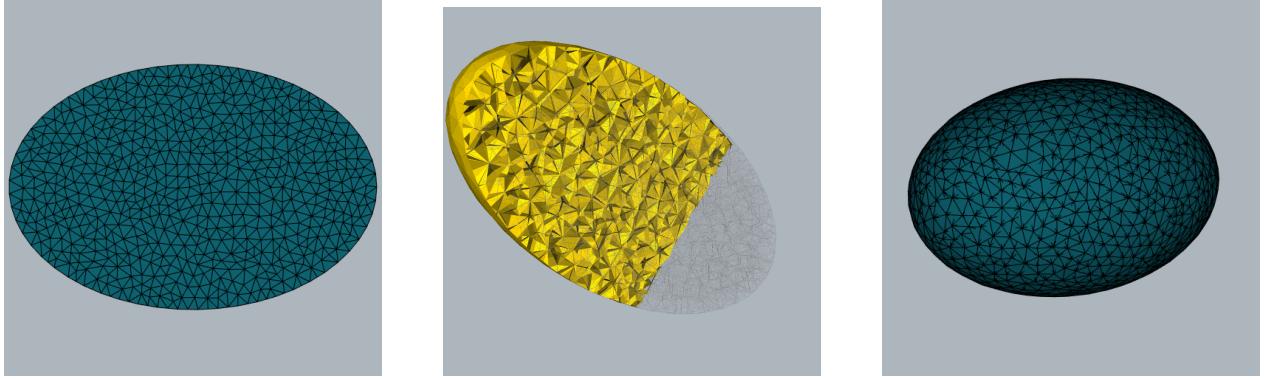
où  $\Delta u = \sum_{i=1}^d \frac{\partial^2 u}{\partial x_i^2}$ .

*Remarque 3.3.* Une condition importante ici est d'imposer  $\Gamma_D \neq \emptyset$ . En effet, si cette condition n'est pas vérifiée, le problème sera mal posé, c'est-à-dire qu'il admettra une infinité de solutions.

L'idée des méthodes éléments finis est de résoudre des équations aux dérivées partielles, numériquement, en discrétisant le domaine  $\Omega_h$ , sa frontière ainsi que les différentes fonctions  $f$ ,  $g$ ,  $u$  et  $u_D$ . Nous allons ainsi présenter ici deux de ces méthodes.

#### 3.1.1 Traitement avec une méthode standard

Commençons par présenter la résolution numérique du problème (4) avec une méthode éléments finis classique. Soient  $\Omega$  le domaine considéré,  $\Gamma$  sa frontière et  $v \in H_0^1(\Omega)$ .



(a) Maillage triangulaire sur une ellipse.

(b) Coupe de maillage tétraédrique sur une ellipsoïde.

(c) Maillage tétraédrique sur une ellipsoïde.

FIGURE 4 – Exemples de maillages d’ellipse et d’ellipsoïde centrées respectivement en  $(0, 0)$  et  $(0, 0, 0)$ , de paramètres  $(1/2, 1/3)$  et  $(1/2, 1/3, 1/3)$ .

L’idée première est de multiplier le problème 4 par cette fonction  $v$ , appelée fonction test, puis d’intégrer le tout par parties. Cela nous donne formellement

$$-\Delta u \cdot v = fv, \quad \forall v \text{ dans } \Omega \quad (5)$$

$$\iff \int_{\Omega} -\Delta u \cdot v = \int_{\Omega} fv, \quad \forall v \text{ dans } \Omega \quad (6)$$

$$\iff \int_{\Omega} \nabla u \cdot \nabla v - \int_{\Gamma} (\nabla u \cdot n)v = \int_{\Omega} fv, \quad \forall v \text{ dans } \Omega. \quad (7)$$

En reprenant les conditions aux limites imposées sur  $\Gamma_D$  et sur  $\Gamma_N$ , cela peut alors s’écrire

$$\int_{\Omega} \nabla u \cdot \nabla v - \overbrace{\int_{\Gamma_D} (\nabla u \cdot n)v}^{=0} = \int_{\Omega} fv + \int_{\Gamma_N} gv, \quad \text{dans } \Omega. \quad (8)$$

Le problème est ainsi de trouver une fonction  $u \in H^1(\Omega)$  telle que

$$a(u, v) = l(v)$$

pour tout  $v \in H_0^1(\Omega)$ , où  $a$  et  $l$  sont données par (8) et sont respectivement la forme bilinéaire et la forme linéaire du problème, c’est-à-dire,

$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v, \quad \text{et} \quad l(v) = \int_{\Omega} fv + \int_{\Gamma_N} gv.$$

L’idée de la méthode éléments finis va alors être de discréteriser l’équation (8). Pour cela, nous allons procéder en plusieurs étapes.

Tout d’abord, pour discréteriser le problème, la première étape consiste à discréteriser le domaine  $\Omega$ . Pour cela, différentes discréterisations sont possibles. Nous en verrons plusieurs au cours de ce rapport : en deux dimensions, les domaines seront généralement discréterisés en maillages triangulaires (figure 4(a)), en trois dimensions, il sera commun de considérer des maillages tétraédriques (figure 4(b) et 4(c)) ou hexaédriques.

*Remarque 3.4 (Condition de Ciarlet.).* Soit  $(\mathcal{T}_h)_{h>0}$  une suite de maillages de  $\Omega$ . Un maillage vérifiant la condition de Ciarlet est tel qu’il existe une constante  $c_0$ , telle que pour tout  $h > 0$  et pour tout élément  $K$  du maillage  $\mathcal{T}_h$  :

$$\frac{\text{diam}(K)}{\rho(K)} \leq c_0,$$

où  $\text{diam}(K) = \max_{x,y \in K} \|x - y\|$  et  $\rho(K) = \max_{B_r \subset K} 2r$ . Cette condition permet ainsi d’empêcher les éléments du maillage d’être trop aplatis.

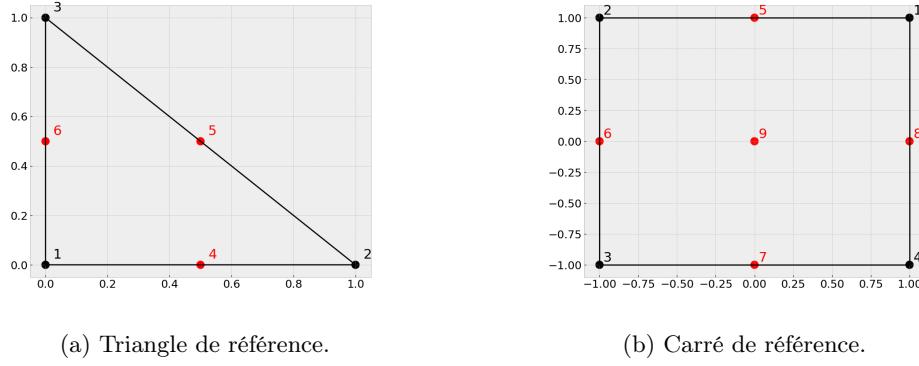


FIGURE 5 – Éléments de référence 2D.

Soit  $\mathcal{T}_h$  le maillage considéré sur le domaine  $\Omega$ . Posons  $V_h^{(k)}$  l'espace éléments finis de degré  $k \geq 1$ , défini par

$$V_h^{(k)} = \{v_h \in H^1(\Omega) : v_{h|T} \in \mathbb{P}^k(T) \forall T \text{ in } \mathcal{T}_h\}, \quad (9)$$

où  $\mathbb{P}^k(T)$  est l'espace des polynômes (en  $d$  variables) de degré inférieur ou égal à  $k$ .

Il est maintenant intéressant de réaliser plusieurs représentations graphiques. Commençons par représenter les éléments appelés éléments de référence. Pour cela, nous représentons deux éléments de référence dans le cas bidimensionnel : l'élément de référence triangulaire et celui quadrangulaire aux figures 5(a) et 5(b). Sur chacune de ces figures, les points noirs représentent les noeuds utilisés pour des éléments  $\mathbb{P}^1$  pour les triangles et  $\mathbb{Q}^1$  pour les quadrilatères et les points rouges représentent les noeuds ajoutés lors de l'utilisation d'éléments  $\mathbb{P}^2$  et  $\mathbb{Q}^2$ .

*Remarque 3.5.* Dans le cas unidimensionnel, l'élément de référence est le segment  $[0, 1]$  et dans le cas tridimensionnel, les éléments de référence sont un tétraèdre et un hexaèdre.

Ces éléments de référence ont un rôle central lors de la résolution numérique avec les éléments finis. En effet, l'idée est d'appliquer une transformation à chacun des éléments du maillage afin de se rapporter à l'un des éléments de référence présentés précédemment. Cela permet ainsi de réaliser les différents calculs sur un même élément et ainsi de les simplifier. Nous ne rentrerons pas dans les détails des différentes transformations utilisées mais ces dernières sont présentées très clairement dans [14].

Soit  $N$  le nombre de noeuds du maillage  $\mathcal{T}_h$ . Il est alors possible d'introduire une base de l'espace  $V_h^{(k)}$ , de dimension  $N$ . Cette base est donnée par  $(\phi_i)_{i=1,\dots,N}$  où les  $\phi_i$ , appelées fonctions de base, sont définies par

$$\forall i, j = 1, \dots, N, \quad \phi_i(s_j) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon,} \end{cases}$$

où les  $s_j$  sont les sommets de  $\mathcal{T}_h$ .

Plusieurs exemples de fonctions de base de l'espace sont représentées à la figure 6. Les figures 6(a) et 6(b) représentent les fonctions de base utilisées dans le cas unidimensionnel, sur l'élément de référence. Par définition des fonctions  $\phi_i$ , en considérant un maillage entier, la fonction  $\phi_i$  sera nulle presque partout sur le maillage, ce qui se voit bien sur la représentation bidimensionnelle, à la figure 6(c).

Ces fonctions de base sont utilisées tout au long des calculs lors de l'utilisation d'une méthode éléments finis. En effet, c'est grâce à ces dernières que les fonctions considérées pour notre problème vont être discrétisées sur chacun des éléments du maillage. Soient donc  $\phi_1, \dots, \phi_N$  les  $N$  fonctions de base du maillage  $\mathcal{T}_h$ , où  $N$  est le nombre de noeuds du maillage. Il est alors possible de construire une approximation  $u_h$  de  $u$  dans la base  $(\phi_1, \dots, \phi_N)$ , définie par

$$u_h := \sum_{j=1}^N U_j \phi_j, \quad (10)$$

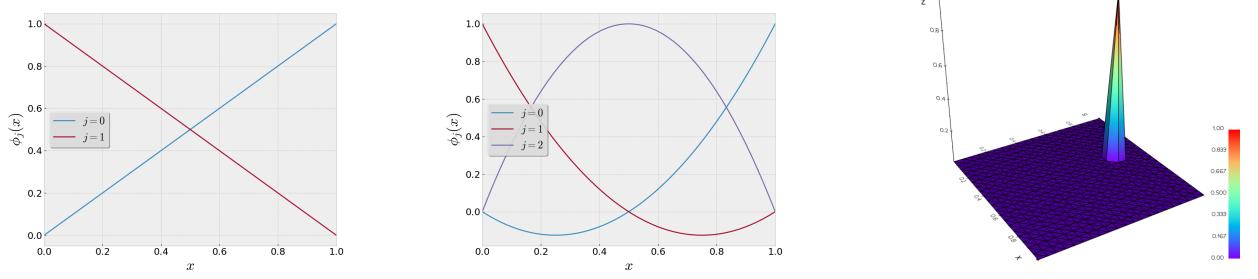
(a) Fonction de base  $\mathbb{P}^1$  en dimension 1 sur l'élément de référence.(b) Fonction de base  $\mathbb{P}^2$  en dimension 1 sur l'élément de référence.(c) Fonction de base  $\mathbb{P}^1$  en dimension 2.

FIGURE 6 – Exemples de fonctions de base.

où  $U_j = u_h(x_j)$ . Cela permet alors de transformer le problème initial, continu, en un problème discret. En effet, pour trouver la solution, il suffit maintenant de déterminer les valeurs du vecteur  $U_j$ , ce qui pourra par exemple dans le cas du problème de Poisson se faire simplement en résolvant un système linéaire de la forme  $AU = B$ . Finalement, les fonctions test  $v_h$  devant appartenir à l'espace  $V_h$ , nous pouvons en particulier prendre  $w_h = \phi_i$  et ainsi transformer le problème variationnel (8) sous la forme :

$$\begin{cases} \text{Trouver } u_h \in V_h \text{ telle que} \\ \sum_{j=1}^N a(\phi_j, \phi_i) u_h(s_j) = l(\phi_i) \quad \forall i = 1, \dots, N. \end{cases} \quad (11)$$

Ce problème peut finalement s'écrire :

$$\text{Trouver } U_h \in \mathbb{R}^N \text{ tel que : } A_h U_h = B_h, \quad (12)$$

avec :

$$\begin{cases} A_h &= (a(\phi_j, \phi_i))_{i,j=1,\dots,N}, \\ U_h &= (u_j)_{j=1,\dots,N} = (u_h(s_j))_{j=1,\dots,N}, \\ B_h &= (l(\phi_i))_{i=1,\dots,N}. \end{cases}$$

*Remarque 3.6.* La résolution vue précédemment peut également être vue comme un problème d'optimisation. Cette méthode pourra notamment être utilisée pour la résolution de problèmes non-linéaires. En effet, trouver la solution du problème 4 revient à trouver une fonction  $u \in H^1(\Omega)$ , minimisant la fonctionnelle  $\mathcal{J}$  où

$$\mathcal{J}(v) := \frac{1}{2} a(v, v) - l(v). \quad (13)$$

### 3.1.2 Traitement avec $\phi$ -FEM : le cas de conditions de Dirichlet pures

L'idée est maintenant d'introduire brièvement les idées et le raisonnement nécessaires à la compréhension de la résolution d'un problème éléments finis avec la méthode  $\phi$ -FEM. Nous ne présenterons donc aucun résultat théorique. Pour plus de détails, le lecteur pourra se référer à [12, 8, 10] ou bien [19], où sont notamment détaillées les hypothèses faites sur le maillage ainsi que les résultats de convergence et de conditionnement de la matrice éléments finis. Commençons par présenter l'idée de la méthode pour un problème avec des conditions de Dirichlet pures. Cela permettra de mieux appréhender la méthode qui sera introduite par la suite, pour des conditions de Neumann puis pour des conditions mixtes.

Considérons donc le problème suivant, dit de Poisson-Dirichlet

$$\begin{cases} -\Delta u &= f, \text{ dans } \Omega, \\ u &= 0, \quad \text{sur } \Gamma, \end{cases} \quad (14)$$

où le domaine  $\Omega$  est un ouvert de  $\mathbb{R}^d$  de frontière  $\Gamma$ , avec  $d = 2$  ou  $d = 3$ . La méthode  $\phi$ -FEM repose sur le principe suivant : le domaine et sa frontière sont définis par une fonction  $\phi$  dite "level-set", telle que

$$\Omega := \{\phi < 0\} \text{ et } \Gamma := \{\phi = 0\}. \quad (15)$$

*Remarque 3.7.* Dans tout ce qui suivra, nous considérerons seulement des éléments finis triangulaires ou tétraédriques pour la présentation de  $\phi$ -FEM. Cependant, il est important de noter que la méthode peut également être utilisée pour des éléments finis hexaédriques.

**Définitions des espaces.** Tout d'abord, nous allons construire un maillage sur le domaine  $\Omega$ . Pour cela, le domaine  $\Omega$  est supposé inclus dans une boîte  $\mathcal{D}$  de  $\mathbb{R}^d$  (par exemple, en dimension 2, un carré). L'idée est alors d'introduire un maillage régulier de cette boîte, noté  $\mathcal{T}_h^\mathcal{D}$ . La situation considérée ici est représentée à la figure 7(a) où  $\mathcal{D}$  est le carré  $[-0.7, 0.7] \times [-0.7, 0.7]$  et  $\Omega$  est l'ellipse centrée en  $(0, 0)$  de paramètres  $(0.5, 1/3)$ . Dans ce cas, la fonction level-set  $\phi$  utilisée est connue explicitement : il suffit de prendre l'équation de l'ellipse, ce qui nous donne

$$\phi(x, y) := -1 + 4x^2 + 9y^2. \quad (16)$$

La fonction level-set va ainsi nous permettre de construire un maillage sur lequel seront faits les calculs par la suite.

*Remarque 3.8.* Un des avantages de la méthode  $\phi$ -FEM est ainsi immédiat à remarquer. En effet, contrairement à la méthode standard décrite précédemment, le maillage construit est un maillage régulier. Ainsi, toutes les cellules du maillage ont la même taille, ce qui est très intéressant pour les calculs.

Maintenant que nous avons défini le maillage  $\mathcal{T}_h^\mathcal{D}$ , la prochaine étape va être de construire le maillage sur lequel seront faits les calculs. Pour cela, définissons un espace éléments finis sur  $\mathcal{T}_h^\mathcal{D}$ ,

$$V_h^{\mathcal{D},(k)} = \{v \in H^1(\mathcal{D}) \mid v|_T \in \mathbb{P}^k(T) \quad \forall T \in \mathcal{T}_h^\mathcal{D}\}.$$

Posons alors  $\phi_h := I_h \phi$  où  $I_h$  est l'opérateur d'interpolation de Lagrange sur  $V_h^{\mathcal{D},(k)}$ . A partir de cette interpolation, nous allons pouvoir sélectionner les cellules qui formeront le maillage « computationnel », utilisé pour les calculs. Ce maillage, que nous noterons  $\mathcal{T}_h$ , est défini par

$$\mathcal{T}_h := \{T \in \mathcal{T}_h^\mathcal{D} : T \cap \{\phi_h < 0\} \neq \emptyset\}. \quad (17)$$

Cela permet alors de définir un domaine discréte, construit sur ce maillage, défini par  $\Omega_h := (\cup_{T \in \mathcal{T}_h} T)^\circ$ , de frontière  $\partial\Omega_h$ . Soit maintenant l'espace fonctionnel  $V_h^{(k)}$ , espace des fonctions  $\mathbb{P}^k$ -Lagrange sur  $\Omega_h$ , défini par

$$V_h^{(k)} = \{v_h \in H^1(\Omega_h) : v_h|_T \in \mathbb{P}^k(T) \quad \forall T \in \mathcal{T}_h\}.$$

*Remarque 3.9.* Il est important de noter que  $\partial\Omega_h \neq \Gamma_h$ . En effet, ici  $\partial\Omega_h$  est le bord du domaine étendu, quand  $\Gamma_h = \{\phi_h = 0\}$ .

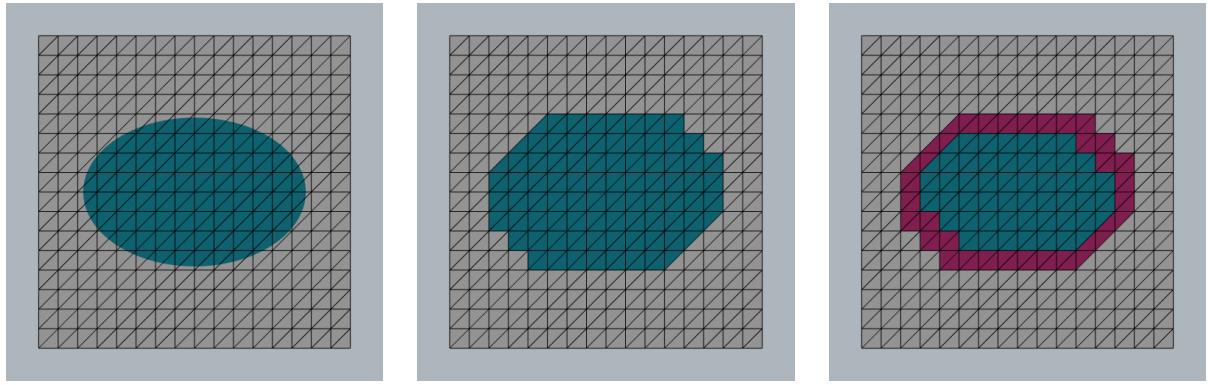
Soient enfin  $\mathcal{T}_h^\Gamma$  l'ensemble des éléments du maillage en intersection avec la frontière discrète,

$$\mathcal{T}_h^\Gamma := \{T \in \mathcal{T}_h : T \cap \Gamma_h \neq \emptyset\}, \quad (18)$$

$\Omega_h^\Gamma$  le domaine occupé par  $\mathcal{T}_h^\Gamma$  et  $\mathcal{F}_h^\Gamma$ , l'ensemble collectant toutes les facettes internes du maillage  $\mathcal{T}_h^\Gamma$ ,

$$\mathcal{F}_h^\Gamma := \{F : \exists T \in \mathcal{T}_h : T \cap \Gamma_h \neq \emptyset \text{ et } F \in \partial T\}. \quad (19)$$

Les représentations de ces différents espaces sont faites à la figure 7. Tout d'abord, nous représentons le domaine physique  $\Omega$  sur le maillage de fond  $\mathcal{T}_h^\mathcal{D}$ , à la figure 7(a). Ensuite, nous représentons le maillage  $\mathcal{T}_h$ , également sur  $\mathcal{T}_h^\mathcal{D}$ . Enfin, nous ajoutons à la figure 7(c), les cellules du maillage  $\mathcal{T}_h^\Gamma$ , en violet. Cette dernière représentation permet notamment de préciser les éléments de  $\mathcal{F}_h^\Gamma$ . En effet, les faces  $F$  de cet ensemble sont toutes les faces de la partie violette, à l'exception des faces communes à la partie violette et à la partie grise.



(a) Maillage régulier de  $\mathcal{D}$ , contenant le domaine  $\Omega$ .  
(b) Maillage sélectionné.  
(c) Situation finale avec les cellules de  $\mathcal{T}_h^\Gamma$  en violet.

FIGURE 7 – Représentation de la situation.

**Description de la méthode.** L'idée de la méthode va être de chercher une solution  $w$  telle que  $\phi w = u$ , avec  $u$  la solution du problème initial. Pour cela, il est nécessaire de supposer que la solution  $u$  peut être étendue du domaine  $\Omega$  au domaine  $\Omega_h$ , comme solution de la même équation.

Pour construire le schéma  $\phi$ -FEM, la première étape est de multiplier le problème par  $\phi$  (en supposant  $\phi$  connue et suffisamment régulière) puis d'intégrer par parties avec comme fonction test  $\phi v$ . Cela nous donne alors le problème

$$\begin{cases} \text{Trouver } w \in H^1(\Omega) \text{ telle que ,} \\ \int_{\Omega} \nabla(\phi w) \cdot \nabla(\phi v) - \int_{\partial\Omega} \frac{\partial}{\partial n}(\phi w)\phi v = \int_{\Omega} f\phi v, \quad \forall v \in H^1(\Omega). \end{cases} \quad (20)$$

Maintenant que le nouveau problème est introduit, il suffit de le discréteriser sur les espaces introduits précédemment, afin d'obtenir le problème

$$\begin{cases} \text{Trouver } w_h \in V_h^{(k)} \text{ telle que,} \\ a(w_h, v_h) = l(v_h), \quad \forall v_h \in V_h^{(k)}, \end{cases} \quad (21)$$

où

$$a(w_h, v_h) := \int_{\Omega_h} \nabla(\phi_h w_h) \cdot \nabla(\phi_h v_h) - \int_{\partial\Omega_h} \frac{\partial}{\partial n}(\phi_h w_h)\phi_h v_h + G_h(w_h, v_h) \quad (22)$$

$$l(v_h) := \int_{\Omega_h} f_h \phi_h v_h + G_h^{rhs}(v_h), \quad (23)$$

avec

$$G_h(w_h, v_h) := \sigma h \sum_{F \in \mathcal{F}_h^\Gamma} \int_F \left[ \frac{\partial}{\partial n}(\phi_h w_h) \right] \left[ \frac{\partial}{\partial n}(\phi_h v_h) \right] + \sigma h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T \Delta(\phi_h w_h) \Delta(\phi_h v_h) \quad (24)$$

$$G_h^{rhs}(v_h) := -\sigma h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T f \Delta(\phi_h v_h), \quad (25)$$

où  $\sigma > 0$  est le paramètre de stabilisation.

*Remarque 3.10.* Dans (22), le terme de bord ne s'annule pas, contrairement au schéma introduit pour la méthode standard dans (8). En effet, ici, le bord  $\partial\Omega_h$  ne correspond pas au bord  $\partial\Omega$ , il est donc impossible d'annuler le terme de bord en appliquant le même principe que pour la méthode standard.

*Remarque 3.11.* Dans (24), les crochets  $([\cdot])$  représentent le saut sur les facettes  $E$  de  $\mathcal{F}_h^\Gamma$  c'est-à-dire

$$\left[ \frac{\partial}{\partial n} (\phi_h w) \right] = (\nabla(\phi_h w)^+ - \nabla(\phi_h w)^-) \cdot n,$$

où  $n$  est le vecteur normal unitaire extérieur à la face  $F$ .

**Explication des termes de stabilisation.** Précisons maintenant le sens des différents termes introduits dans  $G_h(w, v)$  et  $G_h^{rhs}(v)$ .

1. Stabilisation d'ordre 1,

$$\sigma h \sum_{F \in \mathcal{F}_h^\Gamma} \int_F \left[ \frac{\partial}{\partial n} (\phi_h w_h) \right] \left[ \frac{\partial}{\partial n} (\phi_h v_h) \right]_{(1)}.$$

Nous introduisons ici un terme de stabilisation sur l'ensemble de facettes  $\mathcal{F}_h^\Gamma$ , dans l'idée de la pénalité fantôme (ghost penalty) introduite dans [5] pour des éléments finis  $\mathbb{P}^1$ . Ce terme pénalise ainsi les dérivées premières de la solution. Ajouter ce terme au schéma permet ainsi de pénaliser les sauts de gradients sur les facettes. Ce terme a ainsi pour objectif de minimiser la différence du gradient de la solution entre deux cellules, ce qui a pour conséquence d'assurer une continuité de la solution entre deux cellules.

2. Stabilisations d'ordre 2,

$$\sigma h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T \Delta(\phi_h w_h) \Delta(\phi_h v_h) + \sigma h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T f \Delta(\phi_h v_h)$$

Dans cette expression, nous reconnaissions immédiatement la forme forte du problème, c'est-à-dire l'équation (14). Ce terme revient à pénaliser le schéma en imposant la formulation forte sur  $\Omega_h^\Gamma$ . Il est d'ailleurs important de remarquer que si  $\phi_h w_h = u$  est la solution exacte, ce terme s'annule puisque  $-\Delta u = f$ .

*Remarque 3.12* (Conditions non-homogènes). Une dernière remarque essentielle pour la présentation de cette méthode est le traitement de conditions non-homogènes au bord, c'est-à-dire lorsque  $u = u_D$  sur  $\Gamma$ . Dans ce cas, nous ne chercherons plus une solution  $w$  telle que  $u = \phi w$  sur  $\Omega$ , mais telle que  $u = \phi w + u_D$ . Une nouvelle fois, par définition de  $\phi$ , cela implique que  $u$  satisfait directement  $u = u_D$  sur  $\Gamma$ . Les équations (22), (23), (24) et (25) doivent ainsi être modifiées en conséquence, puisque tous les termes en  $\phi_h w_h$  seront modifiés par des termes en  $\phi_h w_h + u_D$ .

*Remarque 3.13.* Nous avons en annexe A réalisé les représentations graphiques des résultats obtenus pour des conditions de Dirichlet pures et des conditions de Neumann pures, aux figures 30 et 31. Pour plus de représentations et notamment l'évolution du conditionnement de la matrice éléments finis ou l'influence des paramètres sur les différents schémas, nous renvoyons le lecteur aux publications [12, 8].

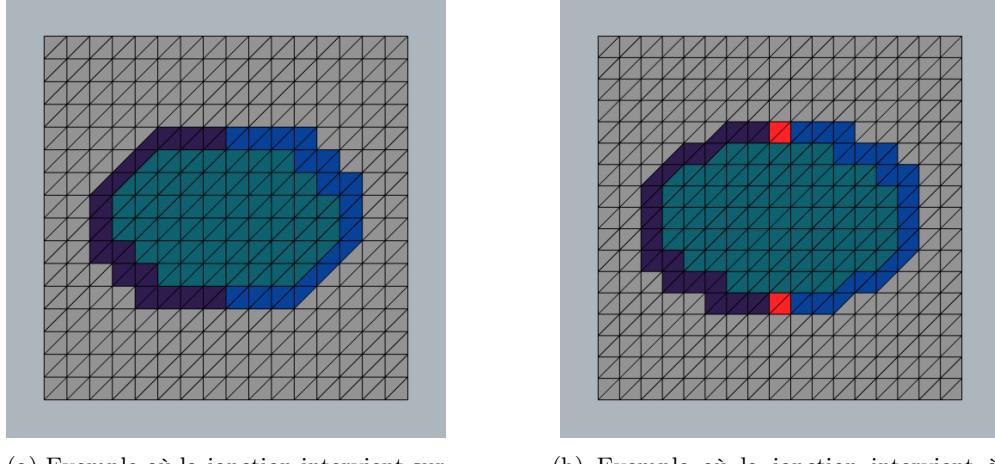
### 3.1.3 Le cas de conditions mixtes

Considérons maintenant le cas du problème

$$\begin{cases} -\Delta u &= f, & \text{dans } \Omega, \\ u &= u_D, & \text{sur } \Gamma_D, \\ \nabla u \cdot n &= g, & \text{sur } \Gamma_N. \end{cases} \quad (26)$$

Pour résoudre ce problème avec  $\phi$ -FEM, la méthode sera différente de la méthode introduite précédemment. L'idée ici sera tout d'abord la même que précédemment : le domaine  $\Omega$  sera étendu à un domaine  $\Omega_h$  et le problème sera résolu sur  $\Omega_h$ . Cependant, ici, sur tout le domaine  $\Omega_h$ , le problème sera résolu comme avec la méthode standard. C'est-à-dire que nous résoudrons

$$\int_{\Omega_h} \nabla u_h \cdot \nabla v_h - \int_{\partial\Omega_h} (\nabla u_h \cdot n) \cdot v_h = \int_{\Omega_h} f v_h \quad \forall v \in V_h^{(k)}, \quad (27)$$



(a) Exemple où la jonction intervient sur une face du maillage. En bleu,  $\Omega_h^{\Gamma_D}$  et en violet  $\Omega_h^{\Gamma_N}$ .

(b) Exemple où la jonction intervient à l'intérieur d'une cellule. En bleu,  $\Omega_h^{\Gamma_D}$ , en violet  $\Omega_h^{\Gamma_N}$  et en rouge les cellules à l'interface.

FIGURE 8 – Deux situations de partitionnement de la frontière.

où  $V_h^{(k)}$  est l'espace éléments finis  $\mathbb{P}^k$  Lagrange sur  $\Omega_h$ .

Ainsi, contrairement à la méthode introduite précédemment, les conditions de bord ne sont pas directement satisfaites par la solution. Pour cela, nous allons imposer nos conditions de bord en introduisant une formulation différente. En effet, les conditions de bord seront imposées grâce à des variables auxiliaires. Nous allons ainsi avoir besoin de définir quelques espaces supplémentaires. Détailons premièrement les points importants avant de construire le schéma  $\phi$ -FEM pour ce problème.

Le cas des conditions mixtes est un cas vraiment complexe pour les méthodes éléments finis non conformes, telle que  $\phi$ -FEM. En effet, La jonction entre la partie Neumann et la partie Dirichlet de la frontière risque d'intervenir à l'intérieur d'une cellule du maillage de la frontière, ce qui peut entraîner de traitement de ces conditions. Pour adapter notre schéma à problème, nous suivons l'approche choisie dans [7] : si la jonction entre les parties de la frontière intervient à l'intérieur d'une cellule, alors cette cellule ne sera soumise à aucune condition de bord. De plus, pour décomposer la frontière, nous supposons que la partie Neumann et la partie Dirichlet peuvent être définies par une seconde fonction level-set,  $\psi$ , telle que

$$\Gamma_D = \Gamma \cap \{\psi < 0\}, \quad \Gamma_N = \Gamma \cap \{\psi > 0\}.$$

Nous introduisons le maillage  $\mathcal{T}_h$  ainsi que le maillage  $\mathcal{T}_h^\Gamma$  comme précédemment, puis séparons ce dernier en deux parties :  $\mathcal{T}_h^{\Gamma_D}$ , autour de la frontière  $\Gamma_D$ , partie sur laquelle nous imposerons les conditions de Dirichlet et  $\mathcal{T}_h^{\Gamma_N}$  autour de  $\Gamma_N$  où nous imposerons les conditions de Neumann. Ces deux maillages sont donc définis par

$$\mathcal{T}_h^{\Gamma_D} := \{T \in \mathcal{T}_h^\Gamma : \psi \leq 0 \text{ sur } T\} \quad \text{et} \quad \mathcal{T}_h^{\Gamma_N} := \{T \in \mathcal{T}_h^\Gamma : \psi \geq 0 \text{ sur } T\}. \quad (28)$$

Nous pouvons alors noter  $\Omega_h$ ,  $\Omega_h^\Gamma$ ,  $\Omega_h^{\Gamma_D}$  et  $\Omega_h^{\Gamma_N}$  les domaines occupés respectivement par les maillages  $\mathcal{T}_h$ ,  $\mathcal{T}_h^\Gamma$ ,  $\mathcal{T}_h^{\Gamma_D}$ ,  $\mathcal{T}_h^{\Gamma_N}$ .

Cependant, un nouveau problème avec ces définitions, est le fait que certaines cellules de la frontière puissent être sélectionnées pour aucun des deux maillages. En effet si la jonction entre les frontières se fait à l'intérieur d'une cellule, le signe de la level-set  $\psi$  change. Comme nous l'avons dit précédemment nous n'imposerons pas de conditions de bord à ces cellules, mais ces dernières doivent tout de même être prises en compte dans notre schéma. Il est donc important de garder en mémoire que ce genre de situation peut se produire, comme par exemple à la figure 8. Ici, nous avons encore considéré l'ellipse centrée en  $(0, 0)$ , de paramètres  $(0.5, 1/3)$  et la level-set  $\psi$  est définie par  $\psi(x, y) = -x$ . Ainsi, la jonction entre les deux parties de la frontière intervient aux points de la frontière vérifiant  $x = 0$ . Les domaines  $\Omega_h^{\Gamma_D}$  et  $\Omega_h^{\Gamma_N}$  sont représentés respectivement en bleu et en violet et la partie interface est elle représentée en rouge.

Supposons maintenant que la solution  $u$  du problème (26) peut être étendue du domaine  $\Omega$  au domaine  $\Omega_h$ , comme solution de la même équation. Nous allons alors pouvoir introduire un schéma  $\phi$ -FEM en utilisant une formulation duale, ce qui permettra d'imposer les conditions de bord de façon indirecte. Cette formulation a été proposée dans un premier temps dans le cas de conditions de Neumann pures, dans [8]. Cette dernière a ensuite été étendue et adaptée au cas de conditions mixtes pour l'élasticité linéaire dans [7]. Pour cette méthode, nous allons donc résoudre le problème sur  $\Omega_h$ , et introduire différentes variables auxiliaires pour imposer les conditions de bord.

Introduisons premièrement les équations et les différentes variables auxiliaires. Nous définissons d'abord une variable auxiliaire  $p_D$  afin d'appliquer les conditions de Dirichlet, telle que

$$u = \phi p_D + u_D \quad \text{sur } \Omega_h^{\Gamma_D}. \quad (29)$$

Ainsi,  $u = u_D$  sur  $\Gamma_D$ .

*Remarque 3.14.* Ici, nous reconnaissions l'imposition des conditions de Dirichlet utilisée dans la version pour des conditions de Dirichlet pures. En effet, dans le cas précédent, sur  $\Omega_h$ , nous cherchions à déterminer  $w$  telle que  $u = \phi w + u_D$ , ce qui correspond à l'équation (29).

Pour les conditions de Neumann, nous allons introduire deux variables auxiliaires  $y$  et  $p_N$ , telles que

$$y = \nabla u, \quad \text{sur } \Omega_h^{\Gamma_N}, \quad (30)$$

$$y \cdot n + \phi p_N = g, \quad \text{sur } \Omega_h^{\Gamma_N}. \quad (31)$$

Ainsi,  $y \cdot n = g$  sur  $\Gamma_N$ . L'équation (30) permet de réécrire le problème original sur  $\Omega_h^{\Gamma_N}$  sous la forme

$$-\operatorname{div} y = f, \quad \text{sur } \Omega_h^{\Gamma_N}, \quad (32)$$

formulation que nous retrouverons lors de l'introduction du schéma  $\phi$ -FEM.

L'équation (31) peut quant à elle être réécrite sous une forme légèrement différente,

$$y \cdot \nabla \phi - p_N \phi = -g |\nabla \phi|, \quad \text{sur } \Omega_h^{\Gamma_N}. \quad (33)$$

En effet, sur  $\Omega_h^{\Gamma_N}$ , imposer  $y \cdot n = -g$  revient à imposer  $y \cdot \nabla \phi = -g |\nabla \phi|$  puisque la normale unitaire extérieure  $n$  est donnée par  $\nabla \phi / |\nabla \phi|$ , par définition de  $\Omega$ . Nous allons maintenant pouvoir introduire les différents espaces éléments finis. Rappelons premièrement la définition de  $V_h^{(k)}$ ,

$$V_h^{(k)} := \left\{ \mathbf{v}_h : \Omega_h \rightarrow \mathbb{R} : \mathbf{v}_{h|T} \in \mathbb{P}^k(T) \quad \forall T \in \mathcal{T}_h, \mathbf{v}_h \text{ continue sur } \Omega_h \right\}. \quad (34)$$

Soit maintenant  $\mathcal{M}_h$ , un sous-maillage de  $\mathcal{T}_h$ . Nous introduisons un espace de vecteurs de dimension  $d$ , défini par

$$Z_h(\mathcal{M}_h) := \left\{ \mathbf{z}_h : \mathcal{M}_h \rightarrow \mathbb{R}^d : \mathbf{z}_{h|T} \in \mathbb{P}^k(T)^{(d)} \quad \forall T \in \mathcal{M}_h, \mathbf{z}_h \text{ continue sur } \mathcal{M}_h \right\}, \quad (35)$$

ainsi que l'espace

$$Q_h^l(\mathcal{M}_h) := \left\{ q_h : \mathcal{M}_h \rightarrow \mathbb{R} : q_{h|T} \in \mathbb{P}^l(T) \quad \forall T \in \mathcal{M}_h, q_h \text{ continue sur } \mathcal{M}_h \text{ si } l \geq 0 \right\}, \quad (36)$$

version locale de l'espace éléments finis  $V_h^{(k)}$ .

**Premier schéma.** Soit  $W_h^{(k)} = V_h^{(k)} \times Q_h^{(k)}(\mathcal{T}_h^{\Gamma_D}) \times Z_h(\mathcal{T}_h^{\Gamma_N}) \times Q_h^{(k-1)}(\mathcal{T}_h^{\Gamma_N})$ . Nous allons ainsi chercher  $(u_h, p_{h,D}, y_h, p_{h,N}) \in W_h^{(k)}$  telles que

$$\begin{aligned}
& \underbrace{\int_{\Omega_h} \nabla u_h : \nabla v_h}_{\text{I}} - \underbrace{\int_{\partial\Omega_h \setminus \partial\Omega_{h,N}} \nabla u_h \cdot n \cdot v_h}_{\text{II}} + \underbrace{\int_{\partial\Omega_{h,N}} (y_h \cdot n) \cdot v_h}_{\text{III}} \\
& + \underbrace{\gamma_u \int_{\Omega_h^{\Gamma_N}} (y_h + \nabla u_h) : (z_h + \nabla v_h)}_{\text{IV}} + \underbrace{\frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \left( y_h \nabla \phi_h + \frac{1}{h} p_{h,N} \phi_h \right) \cdot \left( z_h \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h \right)}_{\text{V}} \\
& + \underbrace{\frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (u_h - \frac{1}{h} \phi_h p_{h,D}) \cdot (v_h - \frac{1}{h} \phi_h q_{h,D})}_{\text{VI}} + G_h(u_h, v_h) + J_h^{lhs,D}(u_h, v_h) + J_h^{rhs,N}(y_h, z_h) \\
& = \underbrace{\int_{\Omega_h} f \cdot v_h}_{\text{VII}} + \underbrace{\frac{\gamma}{h^2} \int_{\Omega_h^D} u_{D,h} \cdot (v_h - \frac{1}{h} \phi_h q_{h,D})}_{\text{VIII}} - \underbrace{\frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} g \cdot |\nabla \phi_h| (z_h \cdot \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h)}_{\text{IX}} \\
& \quad + J_h^{rhs,D}(v_h) + J_h^{rhs,N}(z_h), \\
& \forall (v_h, q_{h,D}, z_h, q_{h,N}) \in W_h^{(k)}, \quad (37)
\end{aligned}$$

où

$$\begin{aligned}
J_h^{lhs,D}(u, v) &:= \sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \Delta u \cdot \Delta v, \quad J_h^{rhs,D}(v) := -\sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T f \cdot \Delta v, \\
J_h^{lhs,N}(y, z) &:= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \operatorname{div} y \cdot \operatorname{div} z, \quad J_h^{rhs,N}(z) := \gamma_{div} \int_{\Omega_h^{\Gamma_N}} f \cdot \operatorname{div} z,
\end{aligned}$$

et

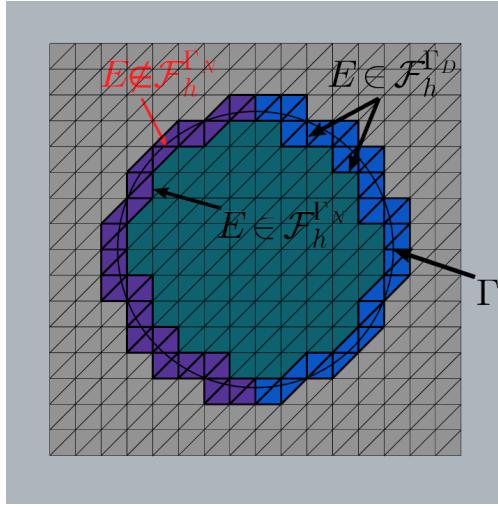
$$G_h(u, v) = \sigma_N h \sum_{F \in \mathcal{F}_h^{\Gamma_N}} \int_F \left[ \frac{\partial}{\partial n} (\phi_h w_h) \right] \left[ \frac{\partial}{\partial n} (\phi_h v_h) \right] + \sigma_D h \sum_{F \in \mathcal{F}_h^{\Gamma_D}} \int_F \left[ \frac{\partial}{\partial n} (\phi_h w_h) \right] \left[ \frac{\partial}{\partial n} (\phi_h v_h) \right].$$

Dans ce schéma, nous reconnaissions tout d'abord les termes apparaissant dans le schéma (7). En effet, en prenant seulement les termes **I**, **II**, **III** et **VII**, nous retrouvons exactement la même formulation, puisque  $y_h = \nabla u_h$  sur  $\partial\Omega_{h,N}$ . Nous retrouvons également les équations (29) dans les termes **VI** et **VIII**, (30) dans le terme **IV** et (33) dans les termes **V** et **IX**.

Les termes  $J_h^{lhs,D}$ ,  $J_h^{rhs,D}$ ,  $J_h^{lhs,N}$ ,  $J_h^{rhs,N}$  et  $G_h$  sont eux construits dans la même idée que les termes de stabilisation introduits dans (24) et (25).

*Remarque 3.15.* Dans le schéma (37), nous avons introduit pour  $G_h$  l'espace  $\mathcal{F}_h^{\Gamma_D}$  et l'espace  $\mathcal{F}_h^{\Gamma_N}$ . Le premier,  $\mathcal{F}_h^{\Gamma_D}$  est construit exactement comme  $\mathcal{F}_h^{\Gamma}$ , en ne considérant que les facettes appartenant à des cellules de  $\mathcal{T}_h^{\Gamma_D}$ . Le deuxième, est construit légèrement différemment. En effet, pour  $\mathcal{F}_h^{\Gamma_N}$ , ne sont considérées que les facettes en intersection entre  $\mathcal{T}_h^{\Gamma_N}$  et  $\mathcal{T}_h$ . Les facettes appartenant à ces deux ensembles sont représentées pour le cas d'un cercle à la figure 9.

Les premiers résultats numériques de ce schéma étaient très encourageants lors de la résolution de problèmes en 2 dimensions (dans le cadre de l'élasticité linéaire [7] notamment). Cependant, lorsque nous avons voulu étendre ce schéma à la résolution de problèmes en 3 dimensions, nous avons été confrontés à différents problèmes. En effet, numériquement, le schéma semblait avoir un défaut : la matrice éléments finis était très creuse, avec souvent, de nombreuses lignes de 0, ce qui avait pour conséquence une non inversibilité de cette matrice. Ainsi, même si ces défauts étaient masqués lors de l'utilisation d'un solveur linéaire avec *FEniCS*, dès le passage au cadre non-linéaire (sur lequel nous reviendrons plus en détails dans la prochaine section) ainsi qu'aux problèmes en 3 dimensions, nous nous sommes aperçus que le solveur non-linéaire ne pouvait converger, en raison d'une divergence du solveur linéaire utilisé pour la résolution de ce problème. Il a donc été nécessaire de comprendre les raisons de la présence d'un tel nombre de 0 dans la matrice éléments finis ainsi que de trouver une solution pour contrer ce problème.

FIGURE 9 – Représentation des ensembles  $\mathcal{F}_h^{\Gamma_N}$  et  $\mathcal{F}_h^{\Gamma_D}$ .

**Second schéma.** Il a ainsi été nécessaire de déterminer la raison menant à ce problème. L’explication que nous avons alors trouvée était la suivante : dans le schéma présenté en (37), nous avons introduit la variable auxiliaire  $p_D$  sur le domaine  $\Omega_h^{\Gamma_D}$ . Cependant, l’équation gouvernant cette variable auxiliaire, (29), semblait laisser la variable trop libre.

Pour corriger ce problème, la première étape a été de modifier la définition de l’un des espaces : plutôt que de considérer un espace continu, nous avons considéré un espace discontinu par maille donné par

$$Q_h^l(\mathcal{M}_h) := \{q_h : \mathcal{M}_h \rightarrow \mathbb{R} : q_{h|T} \in \mathbb{P}^l(T) \quad \forall T \in \mathcal{M}_h, \quad q_h \text{ discontinue sur } \mathcal{M}_h \text{ si } l \geq 0\}. \quad (38)$$

De plus nous avons légèrement modifié le schéma précédemment établi en (37). En particulier, il a été nécessaire de pénaliser les sauts des sauts de  $p_D$  sur les facettes internes de  $\Omega_h^{\Gamma_D}$ . Le nouveau schéma est ainsi donné par : chercher  $(u_h, p_{h,D}, y_h, p_{h,N}) \in W_h^{(k)}$  telles que

$$\begin{aligned} & \int_{\Omega_h} \nabla u_h : \nabla v_h - \int_{\partial\Omega_h \setminus \partial\Omega_{h,N}} \nabla u_h \cdot n \cdot v_h + \int_{\partial\Omega_{h,N}} (y_h \cdot n) \cdot v_h \\ & + \gamma_u \int_{\Omega_h^{\Gamma_N}} (y_h + \nabla u_h) : (z_h + \nabla v_h) + \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_D}} \left( y_h \nabla \phi_h + \frac{1}{h} p_{h,N} \phi_h \right) \cdot \left( z_h \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h \right) \\ & + \sigma_p h^{-1} \int_{\mathcal{F}_h^{\Gamma_D}} [\phi_h p_{h,D}] [\phi_h q_{h,D}] + \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (u_h - \phi_h p_{h,D}) \cdot (v_h - \phi_h q_{h,D}) \\ & + G_h(u_h, v_h) + J_h^{lhs,D}(u_h, v_h) + J_h^{lhs,N}(y_h, z_h) = \int_{\Omega_h} f \cdot v_h + \frac{\gamma}{h^2} \int_{\Omega_h^D} u_{D,h} \cdot (v_h - \phi_h q_{h,D}) \\ & - \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} g \cdot |\nabla \phi_h| (z_h \cdot \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h) + J_h^{rhs,D}(v_h) + J_h^{rhs,N}(z_h), \\ & \forall (v_h, q_{h,D}, z_h, q_{h,N}) \in W_h^{(k)}, \quad (39) \end{aligned}$$

où  $\sigma_p > 0$ ,  $\mathcal{F}_h^{\Gamma_D}$  est l’ensemble des facettes internes de  $\mathcal{T}_h^{\Gamma_D}$  et  $G_h$ ,  $J_h^{lhs,D}$ ,  $J_h^{lhs,N}$ ,  $J_h^{rhs,D}$  et  $J_h^{rhs,N}$  sont définis comme en (37).

*Remarque 3.16.* Une deuxième solution possible était de considérer pour la variable  $p_D$  un espace discontinu de degré  $k - 1$  sans appliquer de stabilisation sur les sauts.

De plus, dans un premier temps, nous avons fait le choix pour les tests réalisés, de ne considérer que des situations où la jonction entre les deux parties de la frontière n’intervient pas dans une cellule. Cela permet ainsi de ne pas avoir de cellules d’interface à gérer.

**Résultats numériques.** Nous allons maintenant nous intéresser à la convergence numérique de cette méthode. Pour cela, nous considérerons le cas test suivant

- le domaine  $\Omega$  est le cercle de centre  $(0.5, 0.5)$  et de rayon  $\sqrt{2}/4$ ,
- la fonction level-set  $\phi$  est ainsi l'équation de ce cercle, donnée par

$$\phi(x, y) := -\frac{1}{8} + (x - 0.5)^2 + (y - 0.5)^2, \quad (40)$$

- les bords  $\Gamma_N$  et  $\Gamma_D$  sont donnés par la fonction level-set  $\psi$  telle que  $\psi(x, y) = -x - 0.5$ ,
- la solution manufacturée est donnée par

$$u_{ex} = \sin(x) \times \exp(y),$$

- par définition,

$$f(x, y) = -\Delta u_{ex} = 0,$$

- les conditions de bord sont données par

$$u_D = (1 + \phi) \times u_{ex}, \quad \text{et} \quad g = \frac{(\nabla u_{ex}, \nabla \phi)}{|\nabla \phi|} + \phi u_{ex}.$$

*Remarque 3.17* (Normes utilisées lors des simulations numériques). Par la suite, lors des simulations numériques, nous utiliserons toujours les mêmes normes pour illustrer la convergence des méthodes. En l'occurrence, nous calculerons la norme  $L^2$  et la semi-norme  $H^1$  et les erreurs représentées seront les erreurs relatives. Ainsi, dans les différentes représentations graphiques, lorsque nous parlerons d'erreur  $L^2$ , cette dernière sera donnée par

$$\frac{\|u_{ex} - u_h\|_{0,\Omega_h}}{\|u_{ex} - u_h\|_{0,\Omega_h}} = \frac{\sqrt{\int_{\Omega_h} (u_{ex} - u_h)^2}}{\sqrt{\int_{\Omega_h} u_{ex}^2}}. \quad (41)$$

L'erreur  $H^1$  sera elle donnée par

$$\frac{|u_{ex} - u_h|_{1,\Omega_h}}{|u_{ex} - u_h|_{1,\Omega_h}} = \frac{\sqrt{\int_{\Omega_h} (\nabla(u_{ex} - u_h))^2}}{\sqrt{\int_{\Omega_h} (\nabla u_{ex})^2}}, \quad (42)$$

où  $u_{ex}$  est la solution exacte considérée et  $u_h$  la solution approchée obtenue à l'aide des différentes méthodes.

Enfin, sur les différents graphiques nous représenterons ces erreurs en fonction de  $h$ , où  $h$  représente le diamètre maximal des cellules (constant pour un maillage régulier).

Les résultats obtenus avec  $\phi$ -FEM et la méthode standard sont représentés à la figure 10. Dans les deux situations présentées, où nous avons utilisé des éléments finis  $\mathbb{P}^1$  et  $\mathbb{P}^2$ , numériquement, l'ordre de convergence  $L^2$  pour  $\phi$ -FEM correspond à une convergence de l'ordre de  $h^{k+1}$ , ce qui suit l'ordre de convergence théorique annoncé pour des conditions de Dirichlet pures ou de Neumann pures dans [12, 8].

*Remarque 3.18.* Afin d'illustrer l'importance des paramètres dans ce schéma, nous avons représenté en annexe A, à la figure 32 les résultats obtenus pour le même cas test, avec des éléments finis  $\mathbb{P}^1$ , pour différentes combinaisons de paramètres. Chacun des graphiques est réalisé en fixant tous les paramètres sauf un, représenté sur l'axe des abscisses.

## 3.2 Problème de Poisson non-linéaire

Nous allons maintenant étudier un problème légèrement différent en apparence, mais entraînant de grandes différences dans la résolution numérique. En effet, jusqu'à maintenant, le problème considéré était linéaire. Cependant, dans le cadre qui nous intéressera par la suite pour la résolution de problèmes hyperélastiques, nous aurons besoin de résoudre des problèmes non-linéaires. Pour cela, nous faisons le choix de présenter

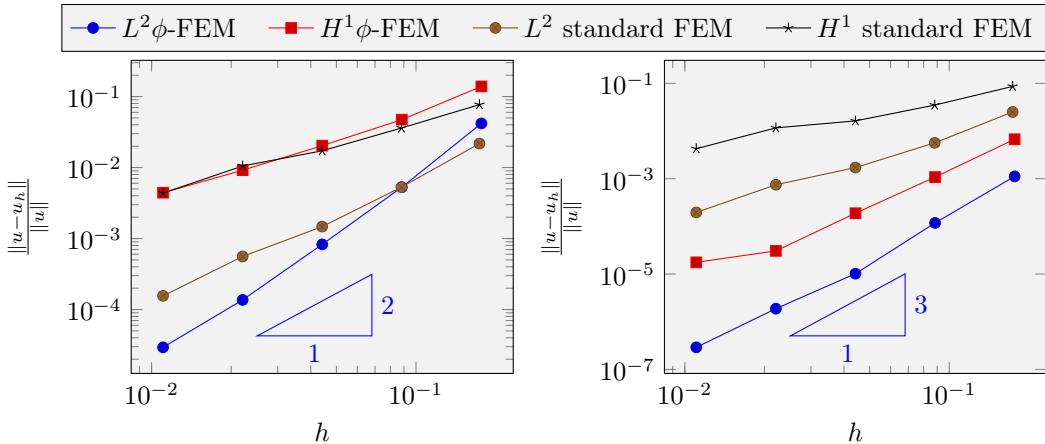


FIGURE 10 – A gauche : éléments finis  $\mathbb{P}^1$ ,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 1.0, 1.0, 0.01, 0.01, 2000.0, 0.2, 0.0001$ . A droite : éléments finis  $\mathbb{P}^2$ ,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 10.0, 1.0, 0.01, 0.01, 20000.0, 20, 0.0001$ .

premièrement une version modifiée du problème considéré précédemment : un problème de Poisson non-linéaire. Nous allons ainsi résoudre ce problème à l'aide des méthodes introduites précédemment. Nous ne redéfinirons donc pas tous les espaces nécessaires mais allons présenter la nouvelle méthode de résolution ainsi que les différents schémas considérés.

Le problème à résoudre sera alors le suivant : trouver  $u$  telle que

$$\begin{cases} -\nabla \cdot (q(u)\nabla u) = f, & \text{dans } \Omega, \\ u = u_D, & \text{sur } \partial\Omega, \end{cases} \quad (43)$$

où  $q(u)$  est une forme supposée non-constante, permettant de rendre le problème non-linéaire, par exemple  $q(u) = 1 + u^2$ .

### 3.2.1 Méthode standard

Le problème (43) se traduit sous la forme, trouver  $u \in V$  telle que

$$F(u; v) = 0 \quad \forall v \in \hat{V}, \quad (44)$$

où

$$F(u; v) = \int_{\Omega} (q(u)\nabla u \cdot \nabla v - fv) dx, \quad (45)$$

et

$$V = \{v \in H^1(\Omega) | v = u_D \text{ sur } \partial\Omega\}, \quad (46)$$

$$\hat{V} = \{v \in H^1(\Omega) | v = 0 \text{ sur } \partial\Omega\}. \quad (47)$$

Il suffit de discréteriser le problème en considérant les espaces  $V_h$  et  $\hat{V}_h$ , versions discréétisées de  $V$  et  $\hat{V}$  respectivement. Le but est alors de trouver  $u_h \in V_h$  tel que

$$F(u_h, v) = 0 \quad \forall v \in \hat{V}_h, \quad (48)$$

où l'on exprime  $u_h$  en fonction des « shape functions » (fonctions de base, dont des exemples sont représentés à la figure 6), sous la forme  $u_h = \sum_{j=1}^N U_j \psi_j$  avec  $\psi_j$  la  $j$ -ième fonction de forme (ou de base),  $U_j$  la valeur de  $U$  au  $j$ -ième noeud du maillage et  $N$  le nombre total de noeuds.

Par définition du problème, la forme  $F$  est évidemment non-linéaire en la variable  $u$ , la résolution du problème revient ainsi à la résolution d'un système non linéaire. Pour résoudre ce problème il sera donc nécessaire de faire appel à un solveur non-linéaire.

Présentons brièvement la résolution de ce genre de problèmes avec une méthode de Newton. Soit  $J$  le jacobien de  $F$  ( $J(x) = F'(x)$ ) et soit  $x_0$  une solution initiale. Pour déterminer la solution  $x_n$  du système

$$F(x) = 0, \quad (49)$$

la forme générale de la méthode Newton est donnée par

$$x_{k+1} = x_k - J^{-1}(x_k)F(x_k), \quad \text{pour } k = 0, 1, 2, \dots \quad (50)$$

Pour éviter d'inverser la matrice  $J$ , calcul qui peut être très lourd, nous déterminons une direction de Newton  $\Delta x_k$ , solution de

$$J(x_k)\Delta x_k = -F(x_k), \quad (51)$$

puis calculons simplement  $x_{k+1} = x_k + \Delta x_k$ .

*Remarque 3.19.* Un problème fréquent à cette méthode est sa convergence pour des problèmes avec des forces trop élevées. En effet, le solveur peut diverger lors de l'application de telles forces, la raison étant une convergence vers un minimum local. Ainsi, il est courant de trouver des codes où les forces sont appliquées par incrément, ce qui permet de fixer une solution initiale lorsque les forces deviennent élevées, et ainsi d'éviter au solveur de converger vers un minimum local.

### 3.2.2 Traitement avec $\phi$ -FEM

Maintenant que nous avons introduit brièvement la méthode standard pour résoudre des problèmes non-linéaires, la prochaine étape va être d'étendre la méthode  $\phi$ -FEM à la résolution de problèmes non-linéaires. En effet, pour l'instant, la méthode n'a été présentée que dans le cadre de problèmes linéaires : problème de Poisson dans [12, 8, 10], équation de la chaleur ou élasticité linéaire dans [7, 11] ou encore les équations de Stokes dans [9]. Nous allons pour cela considérer le même problème que précédemment, en ajoutant également des conditions de Neumann, donné par

$$\begin{cases} -\nabla \cdot (q(u)\nabla u) &= f, & \text{dans } \Omega, \\ u &= u_D, & \text{sur } \Gamma_D, \\ q(u)\nabla u \cdot n &= g, & \text{sur } \Gamma_N, \end{cases} \quad (52)$$

où  $q(u)$  est une forme supposée non-constante, permettant de rendre le problème non-linéaire.

Dans un premier temps, nous construisons les espaces comme précédemment. Ainsi,  $\Omega_h$ ,  $\Omega_h^\Gamma$ ,  $\Omega_h^{\Gamma_D}$  et  $\Omega_h^{\Gamma_N}$  sont définis respectivement comme les domaines occupés par (17), (18) et (28). De plus,  $V_h^{(k)}$ ,  $Z_h^{(k)}$  et  $Q_h^{(l)}$  sont eux donnés respectivement par (34), (35) et (38).

Nous allons définir des variables auxiliaires dans l'idée de celles introduites dans (29), (30) et (33). Ces équations seront légèrement modifiées et les nouvelles variables auxiliaires devront vérifier les équations

$$u = \phi p_D + u_D, \quad \text{sur } \Omega_h^{\Gamma_D}, \quad (53)$$

$$y = q(u)\nabla u, \quad \text{sur } \Omega_h^{\Gamma_N}, \quad (54)$$

$$y \cdot \nabla \phi - \phi p_N = -g|\nabla \phi|, \quad \text{sur } \Omega_h^{\Gamma_N}. \quad (55)$$

En discrétilisant le problème de la même façon que précédemment et en introduisant les équations (53),

(54) et (55), nous obtenons ainsi le schéma suivant : trouver  $(u_h, p_{h,D}, y_h, p_{h,N}) \in W_h^{(k)}$  telles que

$$\begin{aligned} & \int_{\Omega_h} q(u_h) \nabla u_h \cdot \nabla v_h - \int_{\partial\Omega_h \setminus \partial\Omega_{h,N}} q(u_h) (\nabla u_h) n \cdot v_h + \int_{\partial\Omega_{h,N}} y_h \cdot n \cdot v_h \\ & + \gamma_u \int_{\Omega_h^{\Gamma_N}} (y_h + q(u_h) \nabla u_h) \cdot (z_h + q(u_h) \nabla v_h) + \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \left( y_h \nabla \phi_h + \frac{1}{h} p_{h,N} \phi_h \right) \cdot \left( z_h \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h \right) \\ & + \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (u_h - \phi_h p_{h,D}) \cdot (v_h - \phi_h q_{h,D}) + \sigma_p h^{-1} \int_{\mathcal{F}_h^{\Gamma_D}} [\phi_h p_{h,D}] [\phi_h q_{h,D}] \\ & + G_h(u_h, v_h) + J_h^{lhs,D}(u_h, v_h) + J_h^{lhs,N}(y_h, z_h) = \int_{\Omega_h} f \cdot v_h \\ & + \frac{\gamma}{h^2} \int_{\Omega_h^D} u_{D,h} \cdot (v_h - \phi_h q_{h,D}) - \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} g \cdot |\nabla \phi_h| (z_h \cdot \nabla \phi_h + \frac{1}{h} q_{h,N} \phi_h) \\ & \forall (v_h, q_{h,D}, z_h, q_{h,N}) \in W_h^k, \quad (56) \end{aligned}$$

où

$$J_h^{lhs,D}(u, v) := \sigma_D h^2 \int_{\Omega_h^{\Gamma_D}} (\operatorname{div}(q(u) \nabla u) + f) \cdot \operatorname{div}(q(v) \nabla v), \quad (57)$$

$$J_h^{lhs,N}(y, z) := \gamma_{div} \int_{\Omega_h^{\Gamma_N}} (\operatorname{div}(y) - f) \cdot \operatorname{div} z \quad (58)$$

et

$$G_h(u, v) := \sigma_D h \sum_{E \in \mathcal{F}_h^{\Gamma_D}} \int_E [q(u) \nabla u \cdot n] \cdot [q(v) \nabla v \cdot n] + \sigma_N h \sum_{E \in \mathcal{F}_h^{\Gamma_N}} \int_E [q(u) \nabla u \cdot n] \cdot [q(v) \nabla v \cdot n]. \quad (59)$$

Il reste alors à passer tous les termes présents dans le second membre, dans le membre de gauche, afin d'obtenir un problème sous la forme

$$F(u, v) = 0,$$

que nous pouvons résoudre avec une méthode de Newton.

**Résultats numériques.** Analysons maintenant les résultats obtenus avec ces deux méthodes. Pour cela, nous considérerons une solution manufacturée afin de calculer l'erreur. Les différents paramètres de simulations sont ici les suivants :

- $\Omega$  est l'ellipse représentée à la figure 7(a) ou bien le cercle de centre  $(0.5, 0.5)$ , de rayon  $\sqrt{2}/4$ ,
- la fonction level-set  $\phi$  est ainsi l'équation de l'ellipse, donnée par

$$\phi(x, y) := -1 + 4x^2 + 9y^2, \quad (60)$$

ou bien l'équation du cercle,

$$\phi(x, y) := -\frac{1}{8} + (x - 0.5)^2 + (y - 0.5)^2, \quad (61)$$

- pour le cas de l'ellipse, les bords  $\Gamma_N$  et  $\Gamma_D$  sont donnés par la fonction level-set  $\psi$  telle que  $\psi(x, y) = -x$ . Nous considérons donc la situation représentée à la figure 8(a). Pour le cas du cercle nous considérerons  $\psi(x, y) = -x - 0.5$ ,
- la forme  $q$  est définie par  $q(u) = 1 + u^2$ ,
- par définitions de  $u_{ex}$  et de  $q(u)$ ,

$$f(x, y) = -10 - 10x - 20y,$$

- les conditions de bord sont données par

$$u_D = (1 + \phi) \times u_{ex}, \quad \text{et} \quad g = \frac{(1 + q(u_{ex}) \nabla u_{ex}, \nabla \phi)}{|\nabla \phi|} + \phi u_{ex},$$

Les résultats obtenus sont représentés à la figure 11. Sur ces représentations, nous pouvons remarquer que la méthode suit une convergence numériquement satisfaisante. En effet, les résultats obtenus sont meilleurs que pour une méthode éléments finis standard et l'ordre de convergence semble suivre la théorie annoncée dans le cas du problème de Poisson dans [12, 8]. Cependant, il est important de noter comme dans le cas vu précédemment que les paramètres ont une grande importance et que les déterminer peut s'avérer très complexe.

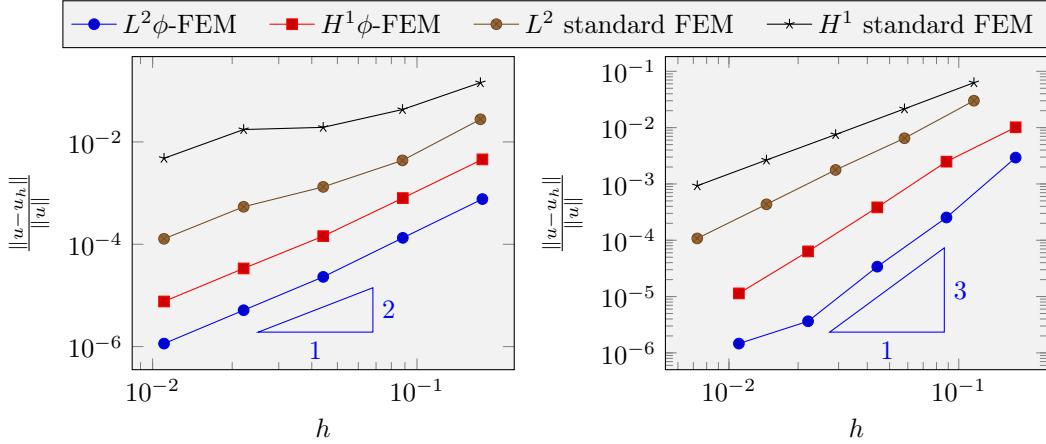


FIGURE 11 – A gauche : éléments finis  $\mathbb{P}^1$ , avec  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 1.0, 0.1, 10, 0.01, 10000, 1, 1.0,$  sur le cercle. A droite : éléments finis  $\mathbb{P}^2$  avec  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 1.0, 0.01, 1, 0.01, 10.0, 1.0, 0.0001,$  sur l'ellipse.

## 4 Le cas de l'hyperélasticité

### 4.1 Une introduction à l'élasticité linéaire

Nous allons maintenant aborder le sujet de l'élasticité linéaire. En effet, avant d'introduire la notion d'hyperélasticité, il est essentiel de considérer dans un premier temps le cas de l'élasticité linéaire, afin d'introduire quelques notions essentielles. La différence avec le cadre vu précédemment est que nous chercherons une solution sous forme vectorielle : le but est maintenant de déterminer le déplacement d'un point  $x$  selon chacun des axes de l'espace. L'idée est la suivante : prendre un point  $x = (x_1, \dots, x_d)$  de l'espace à  $d$  dimensions et lui appliquer le vecteur force :

$$\mathbf{r}(x) = (r_1(x_1, \dots, x_d), \dots, r_d(x_1, \dots, x_d)).$$

Nous cherchons alors à déterminer le déplacement  $u$  du point  $x$ . Pour cela, il est nécessaire d'introduire le tenseur  $\boldsymbol{\sigma}$ , symétrique appelé tenseur des contraintes (de Cauchy). Il est alors alors possible d'écrire les équations dites d'équilibre avec la notation tensorielle et d'ainsi obtenir l'équation

$$-\nabla \cdot \boldsymbol{\sigma} = \mathbf{r}. \tag{62}$$

*Remarque 4.1.* Il est également possible d'écrire ces équations sans utiliser la notation tensorielle, cependant cela implique un système de  $d$  équations, moins lisible et plus lourd à écrire. De plus, en utilisant la notation  $\text{div}(\cdot) = \nabla \cdot (\cdot)$ , l'équation (62) s'écrira également sous la forme

$$-\text{div } \boldsymbol{\sigma} = \mathbf{r}. \tag{63}$$

Introduisons maintenant deux notations, (écrites ici dans le cas où  $d = 3$ ) :

$$\nabla \mathbf{u} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} & \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} \end{bmatrix} \quad \gamma(\mathbf{u}) = \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^t}{2}.$$

Le tenseur  $\nabla \mathbf{u}$  est appelé tenseur gradient de déformation et  $\gamma(\mathbf{u})$  est le tenseur de déformation.

Maintenant que nous avons introduit ces notations, nous allons pouvoir définir le problème d'élasticité linéaire auquel nous nous intéresserons. Soit  $\Omega \subset \mathbb{R}^d$  de frontière  $\Gamma$ . Soient  $\Gamma_N$  et  $\Gamma_D$  tels que  $\Gamma_N \cup \Gamma_D = \Gamma$  et  $\Gamma_D \cap \Gamma_N = \emptyset$ . Nous allons chercher une solution  $\mathbf{u} \in H^1(\Omega)^d$  au problème

$$\begin{cases} -\operatorname{div} \boldsymbol{\sigma}(\mathbf{u}) = \mathbf{r}, & \text{sur } \Omega, \\ \boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n} = \mathbf{g}, & \text{sur } \Gamma_N, \\ \mathbf{u} = \mathbf{u}_D, & \text{sur } \Gamma_D. \end{cases} \quad (64)$$

#### 4.1.1 Méthode éléments finis standard

Nous allons maintenant aborder brièvement la résolution d'un tel problème avec une méthode éléments finis standard. Pour cela, nous considérons une fonction test  $\mathbf{w} = (w_1, \dots, w_d)$  appartenant à  $(H^1(\Omega))^d$ . Comme précédemment, il suffit alors de multiplier le problème (64) par cette fonction test. Le problème devient ainsi de trouver  $\mathbf{u} \in H^1(\Omega)^d$  tel que

$$\begin{aligned} \int_{\Omega} -\operatorname{div} \boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{w} &= \int_{\Omega} \mathbf{r} \cdot \mathbf{w}, \quad \forall \mathbf{w} \in H_0^1(\Omega)^d, \\ \iff \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \nabla \mathbf{w} - \int_{\Gamma_N} (\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{w} &= \int_{\Omega} \mathbf{r} \cdot \mathbf{w}, \quad \forall \mathbf{w} \in H_0^1(\Omega)^d. \end{aligned}$$

De plus, puisque  $\boldsymbol{\sigma}$  est symétrique, nous pouvons utiliser le fait que le produit d'un tenseur symétrique avec un tenseur antisymétrique est nul. Ainsi,

$$\begin{aligned} \boldsymbol{\sigma}(\mathbf{u}) : \nabla \mathbf{w} &= \boldsymbol{\sigma}(\mathbf{u}) : \underbrace{\frac{(\nabla \mathbf{w} + (\nabla \mathbf{w})^t)}{2}}_{\text{partie antisymétrique de } \nabla \mathbf{w}} + \boldsymbol{\sigma}(\mathbf{u}) : \underbrace{\frac{(\nabla \mathbf{w} + (\nabla \mathbf{w})^t)}{2}}_{\text{partie symétrique de } \nabla \mathbf{w}} \\ &= 0 + \boldsymbol{\sigma}(\mathbf{u}) : \frac{(\nabla \mathbf{w} + (\nabla \mathbf{w})^t)}{2} = \boldsymbol{\sigma}(\mathbf{u}) : \gamma(\mathbf{w}). \end{aligned}$$

Ce qui transforme le problème en, chercher  $\mathbf{u}$  tel que

$$\int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \gamma(\mathbf{w}) - \int_{\Gamma_N} (\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{w} = \int_{\Omega} \mathbf{r} \cdot \mathbf{w}, \quad \forall \mathbf{w} \in H_0^1(\Omega)^d.$$

En introduisant le tenseur d'élasticité permettant de relier  $\boldsymbol{\sigma}(\mathbf{u})$  et  $\gamma(\mathbf{u})$  et vérifiant :

$$\boldsymbol{\sigma}(\mathbf{u}) = \mathbf{C} : \gamma(\mathbf{u}).$$

Finalement, chercher une solution au problème (64) revient à chercher un vecteur  $\mathbf{u} \in H^1(\Omega)^d$  tel que :

$$\int_{\Omega} (\mathbf{C} : \gamma(\mathbf{u})) : \gamma \mathbf{w} - \int_{\Gamma_N} (\boldsymbol{\sigma} \cdot \mathbf{n}) \cdot \mathbf{w} = \int_{\Omega} \mathbf{r} \cdot \mathbf{w}, \quad \forall \mathbf{w} \in H_0^1(\Omega)^d. \quad (65)$$

Dans le cas qui nous intéressera ici, c'est-à-dire le cas d'un matériau linéaire élastique isotrope, le tenseur d'élasticité  $\mathbf{C}$  est défini par

$$C_{ijkl} = \lambda I_{ij} I_{kl} + \mu (I_{ik} I_{jl} + I_{il} I_{jk}), \quad (66)$$

avec  $I$  le tenseur identité et  $\lambda$  et  $\mu$  les constantes de Lamé.

Ainsi, le problème peut finalement être réécrit sous la forme, trouver  $\mathbf{u} \in H^1(\Omega)^d$ , tel que :

$$\int_{\Omega} \lambda (\operatorname{div} \mathbf{u}) (\operatorname{div} \mathbf{w}) + 2\mu \gamma(\mathbf{u}) : \gamma(\mathbf{w}) = \int_{\Omega} \mathbf{r} \cdot \mathbf{w} + \int_{\Gamma_N} \mathbf{g} \cdot \mathbf{w}, \quad \forall \mathbf{w} \in H_0^1(\Omega)^d, \quad (67)$$

où le vecteur  $\mathbf{g}$  est le vecteur de traction, tel que  $\mathbf{g} = \boldsymbol{\sigma} \cdot \mathbf{n} = \lambda (\operatorname{div} \mathbf{u}) \mathbf{n} + 2\mu \gamma(\mathbf{u}) \cdot \mathbf{n}$ .

### 4.1.2 Le schéma $\phi$ -FEM pour l'élasticité linéaire

Le schéma précédent a ainsi été adapté à la méthode  $\phi$ -FEM dans [7]. Nous allons le présenter ici assez brièvement, en utilisant les espaces définis en (34), (35), (38). Cependant, ici ces derniers seront légèrement modifiés. En effet, jusqu'à présent il s'agissait respectivement d'espaces de fonctions, de vecteurs et de fonctions. Cependant, par définition du nouveau problème, les variables auxiliaires qui seront introduites, seront des tenseurs ou des vecteurs. En effet, les équations (30), (33) et (29) seront légèrement différentes. Les nouvelles équations sont ainsi

$$\mathbf{u} = \phi \mathbf{p}_D + \mathbf{u}_D, \quad \text{sur } \Omega_h^{\Gamma_D}, \quad (68)$$

$$\mathbf{y} = \boldsymbol{\sigma}(\mathbf{u}), \quad \text{sur } \Omega_h^{\Gamma_N}, \quad (69)$$

$$\mathbf{y} \cdot \nabla \phi - \mathbf{p}_N \phi = -\mathbf{g} |\nabla \phi|, \quad \text{sur } \Omega_h^{\Gamma_N}. \quad (70)$$

Ainsi, les variables auxiliaires  $p_D$  et  $p_N$  sont vectorielles et la variable  $y$  est elle tensorielle. Les nouveaux espaces sont donc définis par

$$\begin{aligned} V_h^{(k)} &:= \left\{ \mathbf{v}_h : \Omega_h \rightarrow \mathbb{R}^d : \mathbf{v}_{h|T} \in \mathbb{P}^k(T)^d \quad \forall T \in \mathcal{T}_h, \mathbf{v}_h \text{ continue sur } \Omega_h \right\}, \\ Z_h(\mathcal{M}_h) &:= \left\{ \mathbf{z}_h : \mathcal{M}_h \rightarrow \mathbb{R}^{(d \times d)} : \mathbf{z}_{h|T} \in \mathbb{P}^k(T)^{(d \times d)} \quad \forall T \in \mathcal{M}_h, \mathbf{z}_h \text{ continue sur } \mathcal{M}_h \right\}, \\ Q_h^l(\mathcal{M}_h) &:= \left\{ \mathbf{q}_h : \mathcal{M}_h \rightarrow \mathbb{R}^d : \mathbf{q}_{h|T} \in \mathbb{P}^l(T)^d \quad \forall T \in \mathcal{M}_h, \mathbf{q}_h \text{ discontinue sur } \mathcal{M}_h \text{ si } l \geq 0 \right\}. \end{aligned}$$

Finalement, nous pouvons introduire le schéma  $\phi$ -FEM consistant à trouver  $\mathbf{u}_h \in V_h$ ,  $\mathbf{p}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D})$ ,  $\mathbf{y}_h \in Z_h(\Omega_h^{\Gamma_N})$  et  $\mathbf{p}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N})$  tels que

$$\begin{aligned} &\int_{\Omega_h} \boldsymbol{\sigma}(\mathbf{u}_h) : \nabla \mathbf{v}_h - \int_{\partial \Omega_h \setminus \partial \Omega_{h,N}} \boldsymbol{\sigma}(\mathbf{u}_h) \mathbf{n} \cdot \mathbf{v}_h + \int_{\partial \Omega_{h,N}} \mathbf{y}_h \mathbf{n} \cdot \mathbf{v}_h \\ &+ \gamma_u \int_{\Omega_h^{\Gamma_N}} (\mathbf{y}_h + \boldsymbol{\sigma}(\mathbf{u}_h)) : (\mathbf{z}_h + \boldsymbol{\sigma}(\mathbf{v}_h)) + \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \left( \mathbf{y}_h \nabla \phi_h + \frac{1}{h} \mathbf{p}_{h,N} \phi_h \right) \cdot \left( \mathbf{z}_h \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h \right) \\ &+ \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (\mathbf{u}_h - \phi_h \mathbf{p}_{h,D}) \cdot (\mathbf{v}_h - \phi_h \mathbf{q}_{h,D}) + \sigma_p h^{-1} \int_{\mathcal{F}_h^{\Gamma_D}} [\phi_h \mathbf{p}_{h,D}] [\phi_h \mathbf{q}_{h,D}] \\ &+ G_h(\mathbf{u}_h, \mathbf{v}_h) + J_h^{lhs,D}(\mathbf{u}_h, \mathbf{v}_h) + J_h^{lhs,N}(\mathbf{y}_h, \mathbf{z}_h) = \int_{\Omega_h} \mathbf{f} \cdot \mathbf{v}_h \\ &+ \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} \mathbf{u}_h^q \cdot (\mathbf{v}_h - \frac{1}{h} \phi_h \mathbf{q}_{h,D}) - \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \mathbf{g} \cdot |\nabla \phi_h| (\mathbf{z}_h \cdot \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h) \\ &+ J_h^{rhs,D}(\mathbf{v}_h) + J_h^{rhs,N}(\mathbf{z}_h), \\ &\forall \mathbf{v}_h \in V_h, \mathbf{q}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D}), \mathbf{z}_h \in Z_h(\Omega_h^{\Gamma_N}), \mathbf{q}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N}). \quad (71) \end{aligned}$$

avec

$$\begin{aligned} J_h^{lhs,D}(\mathbf{u}, \mathbf{v}) &:= \sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \operatorname{div} \boldsymbol{\sigma}(\mathbf{u}) \cdot \operatorname{div} \boldsymbol{\sigma}(\mathbf{v}), \quad J_h^{rhs,D}(\mathbf{v}) := -\sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \mathbf{f} \cdot \operatorname{div} \boldsymbol{\sigma}(\mathbf{v}), \\ J_h^{lhs,N}(\mathbf{y}, \mathbf{z}) &:= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \operatorname{div} \mathbf{y} \cdot \operatorname{div} \mathbf{z}, \quad J_h^{rhs,N}(\mathbf{z}) := \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \mathbf{f} \cdot \operatorname{div} \mathbf{z}, \end{aligned}$$

et

$$G_h(u, v) := \sigma_D h \sum_{E \in \mathcal{F}_h^{\Gamma_D}} \int_E [\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n}] \cdot [\boldsymbol{\sigma}(\mathbf{v}) \cdot \mathbf{n}] + \sigma_N h \sum_{E \in \mathcal{F}_h^{\Gamma_N}} \int_E [\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n}] \cdot [\boldsymbol{\sigma}(\mathbf{v}) \cdot \mathbf{n}].$$

**Résultats numériques.** Une nouvelle fois, il est maintenant important de représenter graphiquement l'erreur des deux méthodes sur un même problème, afin d'étudier l'ordre de convergence numérique de ces dernières. Pour cela, nous allons considérer deux cas tests, en modifiant seulement le maillage considéré. Les cas test sont donc

- $\Omega$  est l'ellipse représentée à la figure 7(a) ou bien le cercle de centre  $(0.5, 0.5)$ , de rayon  $\sqrt{2}/4$ ,
- la fonction level-set  $\phi$  est ainsi l'équation de l'ellipse, donnée par

$$\phi(x, y) := -1 + 4x^2 + 9y^2, \quad (72)$$

ou bien l'équation du cercle,

$$\phi(x, y) := -\frac{1}{8} + (x - 0.5)^2 + (y - 0.5)^2, \quad (73)$$

- pour le cas de l'ellipse, les bords  $\Gamma_N$  et  $\Gamma_D$  sont donnés par la fonction level-set  $\psi$  telle que  $\psi(x, y) = -x$ . Nous considérons donc la situation représentée à la figure 8(a). Pour le cas du cercle nous considérerons  $\psi(x, y) = -x - 0.5$ ,
- la solution manufaturée est donnée par

$$u_{ex} = (\cos(x) \sin(y), \cos(x) \sin(y)),$$

- par définition,

$$f = (2 \sin(y) \cos(x), 2 \sin(y) \cos(x)),$$

- les conditions de bord sont données par

$$u_D = (1 + \phi) \times u_{ex}, \quad \text{et} \quad g = \frac{(\nabla u_{ex}, \nabla \phi)}{|\nabla \phi|} + \phi u_{ex}.$$

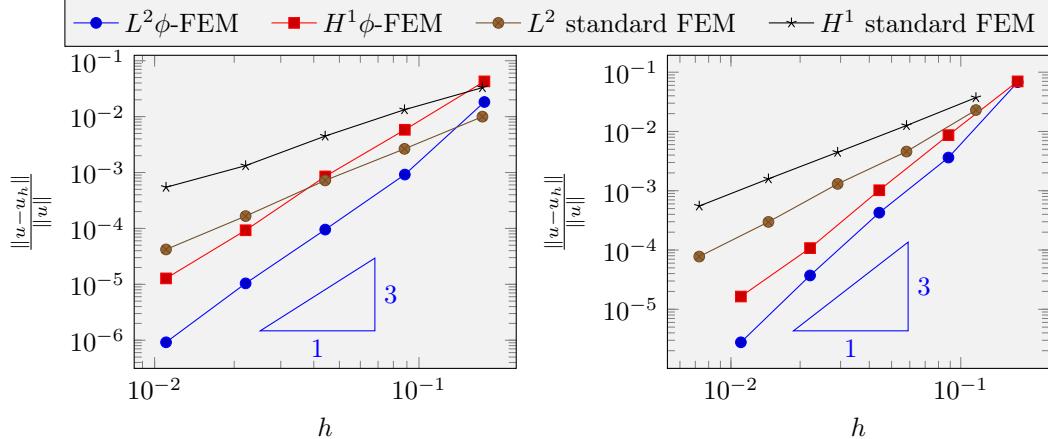


FIGURE 12 – A gauche : éléments finis  $\mathbb{P}^2$ ,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 10.0, 1.0, 0.01, 0.01, 200000.0, 0.02, 0.001$  sur le cercle. A droite : éléments finis  $\mathbb{P}^2$  sur l'ellipse,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 10.0, 1.0, 0.01, 0.01, 200000.0, 0.02, 0.001$ .

Pour illustrer les déformations subies par le domaine sur un problème d'élasticité linéaire, nous avons représenté à la figure 13 le domaine initial, le domaine subissant le déplacement exact

$$u_{ex} = (0.3 \cos(x) \sin(y), 0.3 \cos(x) \sin(y)),$$

et le domaine subissant le déplacement obtenu après résolution avec  $\phi$ -FEM.

## 4.2 Les déformations et tenseurs essentiels en hyperélasticité

Maintenant que nous avons introduit les notions essentielles, nous allons nous intéresser à l'extension de la méthode  $\phi$  au cadre de l'élasticité non-linéaire (autrement appelée hyperélasticité). Ici, nous allons ainsi utiliser tout ce qui a été introduit précédemment : en effet, le but va être de développer un schéma  $\phi$ -FEM

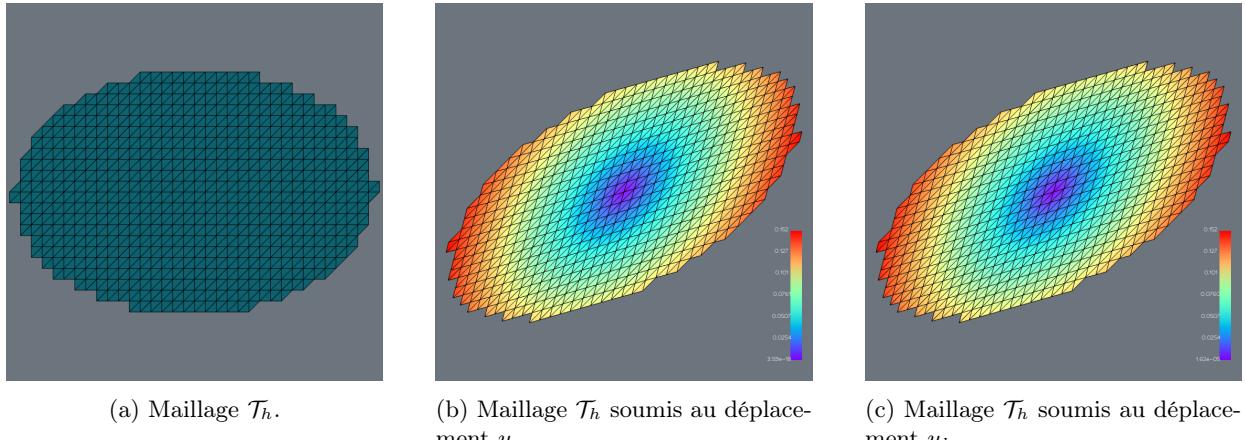


FIGURE 13 – Déformations du maillage.

pour l'élasticité, en considérant des déformations non-linéaires et il sera ainsi nécessaire des résoudre des systèmes non-linéaires.

De plus, même si nous nous contenterons dans ce rapport d'étudier numériquement dans un premier temps des problèmes statiques, il est important de préciser un point : en élasticité non-linéaire, les déformations et déplacements subis par les matériaux sont plus importants que dans le cadre de l'élasticité linéaire. Ainsi, nous aurons besoin de définir différentes lois de comportement pour modéliser ces déformations. En raison de ces déformations, le domaine  $\Omega$  varie au fil du temps pour un problème dynamique. Il est donc important d'introduire quelques aspects théoriques tenant compte du temps.

Soit  $t_0$  le temps initial, et soit  $\Omega^0$  le domaine au temps  $t_0$ . Le principe de la méthode sera, à chaque temps considéré de se ramener par un changement de variables au domaine  $\Omega^0$ . À chaque temps  $t$  le nouveau domaine sera noté  $\Omega^t$  et les points de ce domaine seront notés  $x(X, t) = (x_1(X, t), x_2(X, t), x_3(X, t))^t$  où  $X$  est un point de la configuration initiale  $\Omega^0$  (ici considérée de dimension 3).

Soit  $\mathbf{u}$  le déplacement appliqué au point  $X$ . Ce déplacement peut s'exprimer sous la forme

$$\mathbf{u}(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t) - \mathbf{X}. \quad (74)$$

Il est alors possible d'exprimer  $\mathbf{x}(\mathbf{X}, t)$  sous forme vectorielle,

$$\begin{bmatrix} x_1(\mathbf{X}, t) \\ x_2(\mathbf{X}, t) \\ x_3(\mathbf{X}, t) \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} u_1(\mathbf{X}, t) \\ u_2(\mathbf{X}, t) \\ u_3(\mathbf{X}, t) \end{bmatrix}. \quad (75)$$

Introduisons une convention de notation qui sera utilisée tout au long de cette section. Comme dit précédemment, nous aurons besoin de faire des changements de variables. Nous noterons donc désormais les mesures sur la configuration initiale avec des lettres majuscules ( $dX$ ) et des lettres minuscules pour les configurations déformées ( $dx$ ). Ces transformations de mesures seront réalisées grâce aux différents tenseurs que nous allons maintenant introduire.

Un tenseur primordial pour ces transformations est le tenseur gradient de déformation, noté  $F$  et défini par

$$F = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix}. \quad (76)$$

Nous supposerons ici  $F$  toujours inversible et d'inverse :

$$F^{-1} = \begin{bmatrix} \frac{\partial X_1}{\partial x_1} & \frac{\partial X_1}{\partial x_2} & \frac{\partial X_1}{\partial x_3} \\ \frac{\partial X_2}{\partial x_1} & \frac{\partial X_2}{\partial x_2} & \frac{\partial X_2}{\partial x_3} \\ \frac{\partial X_3}{\partial x_1} & \frac{\partial X_3}{\partial x_2} & \frac{\partial X_3}{\partial x_3} \end{bmatrix}. \quad (77)$$

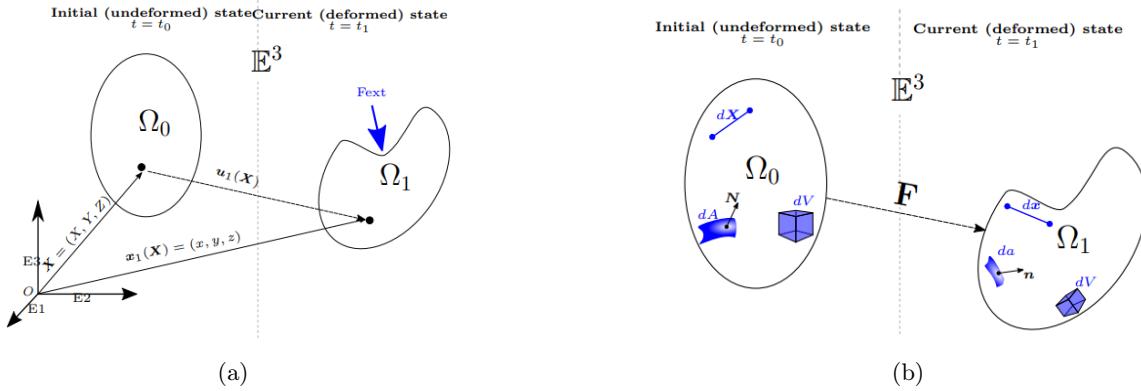


FIGURE 14 – Différences entre la configuration initiale et la configuration déformée ([4]).

Les composantes de  $F$  seront notées  $F_{ij}$  et son déterminant sera noté  $J$ . Introduisons maintenant les formules permettant de relier les différents tenseurs, dans le tableau 1. Nous avons également dans ce tableau donné les relations avec une partie des tenseurs qui seront utilisés par la suite. Ici seulement une expression de chaque tenseur est donnée et lorsque cela sera nécessaire, (pour certaines caractérisations notamment), d'autres expressions ou définitions seront données. Il est également intéressant de présenter les transformations utilisées

Définition du tenseur	Notation	Expression
Tenseur gradient de déplacement	$\nabla_{\mathbf{X}} \mathbf{u}$	$(\nabla_{\mathbf{X}} \mathbf{u})_{ij} = \frac{\partial u_i}{\partial X_j}$
Tenseur gradient de déformation	$F$	$F = I + \nabla_{\mathbf{X}} \mathbf{u}$
Partie symétrique de $F$	$\gamma_{\mathbf{X}} \mathbf{u}$	$\gamma_{\mathbf{X}} \mathbf{u} = \frac{\nabla_{\mathbf{X}} \mathbf{u} + \nabla_{\mathbf{X}} \mathbf{u}^t}{2}$
Tenseur de Cauchy-Green	$C$	$C = F^t \cdot F$
Tenseur de Green-Lagrange	$E$	$E = \frac{1}{2}(C - I)$

TABLE 1 – Table des tenseurs utilisés.

pour passer de la configuration initiale  $\Omega_0$  à une configuration déformée  $\Omega_t$ . Pour cela, nous introduisons ces différentes transformations dans le tableau 2. Les différences sont également représentées à la figure 14, issue de [4].

**Définition 4.1** (Jacobien surfacique). Le rapport entre l'aire d'un élément de surface déformée et un élément de surface déformée est appelé le jacobien surfacique. Il est donné par

$$J_S = \frac{da}{dA} = J \|F^{-t} \cdot N\| = J \sqrt{(C^{-1} \cdot \mathbf{N}) \cdot \mathbf{N}} = \frac{J}{\sqrt{(F \cdot F^{-t} \cdot \mathbf{n}) \cdot \mathbf{n}}} \quad (78)$$

où  $N$  et  $n$  sont respectivement la normale à la configuration initiale et la normale à la configuration déformée.

La normale à la configuration déformée peut ainsi s'exprimer en fonction de la normale à la configuration initiale via la formule

$$n = J F^{-t} \cdot N \frac{dA}{da} = \frac{J}{J_S} (F^{-t} \cdot N). \quad (79)$$

**Définition 4.2** (Tenseur de Cauchy). Soit  $df$  un élément de force agissant sur un élément de surface  $da$  de

Configuration initiale	Configuration déformée	Transformation utilisée
Élément de longueur $dX = (dX_1, dX_2, dX_3)^t$	Élément de longueur $dx = (dx_1, dx_2, dx_3)^t$	$dx = F \cdot dX$
Longueur $dl$ de $dX$	Longueur $(dl)^2$ de $dx$	$(dl)^2 = (\mathbf{C} \cdot dX) \cdot dX$
Élément de volume $dV$	Élément de volume $dv$	$dv = \det F dV = J dV$
Élément d'aire orientée $dA$	Élément d'aire orientée $da$	$da = J(\mathbf{X}) F^{-t} \cdot dA$
Divergence de $W$	Divergence de $w$	$\operatorname{div} w = \frac{\nabla_{\mathbf{x}} \cdot W}{J} = \frac{\nabla_{\mathbf{x}} \cdot (J F^{-1} \cdot w)}{J}$
Divergence de $T_0$	Divergence de $T$	$\operatorname{div} T = \frac{\nabla_{\mathbf{x}} \cdot T_0}{J} = \frac{\nabla_{\mathbf{x}} \cdot (J T \cdot F^{-t})}{J}$
Masse volumique $\rho_0$	Masse volumique $\rho$	$\rho = \frac{\rho_0}{J}$

TABLE 2 – Table des changements de variables.

la configuration déformée  $\Omega^t$ , de normale  $\mathbf{n}$ . Le tenseur de Cauchy  $\boldsymbol{\sigma}$  vérifie

$$df = (\boldsymbol{\sigma} \cdot \mathbf{n}) da \text{ ou de façon équivalente } \frac{df}{da} = \boldsymbol{\sigma} \cdot \mathbf{n}. \quad (80)$$

**Définition 4.3** (Premier tenseur de Piola-Kirchhoff). Soit  $df$  un élément de force agissant sur un élément de surface  $da$  de la configuration déformée  $\Omega^t$ , de normale  $\mathbf{n}$ . Le premier tenseur de Piola-Kirchhoff  $\boldsymbol{\Pi}$  est défini par

$$df = \boldsymbol{\Pi} \cdot \mathbf{N} da. \quad (81)$$

**Définition 4.4** (Deuxième tenseur de Piola-Kirchhoff). Soit  $df$  un élément de force agissant sur un élément de surface  $da$  de la configuration déformée  $\Omega^t$ , de normale  $\mathbf{n}$ . Soit  $df_0 = F^{-1} \cdot df$  un élément de force sur un élément d'aire  $dA$  de la configuration initiale. Le second tenseur de Piola-Kirchhoff noté  $\mathbf{S}$  vérifie

$$\mathbf{S} \cdot \mathbf{N} da = df_0. \quad (82)$$

Ce qui va nous intéresser par la suite est la relation entre ces différents tenseurs. En effet, il est possible d'exprimer ces tenseurs les uns en fonction des autres, comme nous l'avons déjà vu pour certains avec le tableau 1. Nous allons maintenant introduire les dernières relations dont nous aurons besoin.

**Lemme 4.1** (Relations entre les tenseurs). *Soient  $\boldsymbol{\Pi}$ ,  $\mathbf{S}$ ,  $F$  et  $\boldsymbol{\sigma}$  les tenseurs définis précédemment. Ils vérifient alors,*

$$\boldsymbol{\Pi} = F \cdot \mathbf{S}, \quad (83)$$

$$\boldsymbol{\sigma} = \frac{1}{J} F \cdot \mathbf{S} \cdot F^t, \quad (84)$$

et

$$\boldsymbol{\sigma} = \frac{1}{J} \boldsymbol{\Pi} \cdot F^t. \quad (85)$$

*Démonstration.* 1) Par définition,

$$df = \boldsymbol{\Pi} \cdot \mathbf{N} da \text{ et } \mathbf{S} \cdot \mathbf{N} da = F^{-1} df.$$

Alors,

$$\mathbf{S} \cdot \mathbf{N}dA = F^{-1}df \Leftrightarrow df = F \cdot \mathbf{S} \cdot \mathbf{N}dA.$$

En combinant les deux relations,

$$\mathbf{\Pi} \cdot \mathbf{N}dA = F \cdot \mathbf{S} \cdot \mathbf{N}dA \Leftrightarrow \mathbf{\Pi} = F \cdot \mathbf{S}.$$

2) Premièrement,

$$(\boldsymbol{\sigma} \cdot \mathbf{n})da = df = Fdf_0 = F\mathbf{S}\mathbf{N}dA.$$

Or, par changement de variable entre la configuration originale et la configuration déformée,

$$\boldsymbol{\sigma} \cdot (\mathbf{n}da) = \boldsymbol{\sigma} JF^{-t} \cdot \mathbf{N}dA.$$

D'où :

$$\begin{aligned} \boldsymbol{\sigma} JF^{-t} \cdot \mathbf{N}dA &= F\mathbf{S}\mathbf{N}dA \\ \Leftrightarrow \boldsymbol{\sigma} &= \frac{1}{J}FSF^t. \end{aligned}$$

3) La troisième relation découle directement des deux précédentes : il suffit de remplacer  $F \cdot \mathbf{S}$  (d'après la première expression), dans la deuxième expression pour obtenir le résultat souhaité.  $\square$

### 4.3 Matériaux hyperélastiques

Maintenant que nous avons introduit les premiers points importants de cette section, nous allons pouvoir parler des matériaux hyperélastiques. Pour cela, nous allons introduire plusieurs modèles courants (ou lois de comportement). Ces modèles, tels que les modèles néo-hookéen, de Saint-Venant-Kirchhoff ou d'Ogden par exemple sont caractérisés par l'expression du potentiel considérée. En effet, dans le cas hyperélastique, le deuxième tenseur de Piola-Kirchhoff est exprimé en fonction d'un potentiel d'énergie de déformation, sous la forme

$$\mathbf{S} = \frac{\partial \Psi}{\partial E} = 2 \frac{\partial \Psi}{\partial \mathbf{C}}, \quad (86)$$

où  $\mathbf{C}$  est le tenseur de Cauchy-Green et  $E$  est le tenseur de Green-Lagrange.

Le potentiel peut être exprimé en fonction des invariants du tenseur de Cauchy-green, donnés par :

$$\begin{cases} I_1 &= \text{tr}(\mathbf{C}), \\ I_2 &= \frac{1}{2}(I_1^2 - \text{tr}(\mathbf{C} \cdot \mathbf{C})), \\ I_3 &= \det(\mathbf{C}). \end{cases} \quad (87)$$

En supposant une relation pour exprimer  $\Psi$  de la forme :

$$\Psi = \Psi(I_1, I_2, I_3),$$

cela nous donne finalement une expression du deuxième tenseur de Piola-Kirchhoff, donnée par :

$$\mathbf{S} = 2 \left( \frac{\partial \Psi}{\partial I_1} I + \frac{\partial \Psi}{\partial I_2} (I_1 I - \mathbf{C}) + \frac{\partial \Psi}{\partial I_3} I_3 \mathbf{C}^{-1} \right). \quad (88)$$

*Remarque 4.2.* Cette formulation est obtenue en utilisant les dérivées des invariants définis par (87), données par

$$\begin{cases} \frac{\partial I_1}{\partial \mathbf{C}} &= I, \\ \frac{\partial I_2}{\partial \mathbf{C}} &= I_1 I - \mathbf{C}, \\ \frac{\partial I_3}{\partial \mathbf{C}} &= I_3 \mathbf{C}^{-t}. \end{cases} \quad (89)$$

*Remarque 4.3.* Dans [14] et [4] notamment, plus de détails théoriques sur la notion d'hyperélasticité sont donnés, nous nous contenterons dans ce rapport d'introduire les points importants et nécessaires à la construction des schémas de résolution qui nous intéresseront par la suite.

Nous allons maintenant présenter quelques modèles très utilisés en hyperélasticité. Un des objectifs sur le long terme pour  $\phi$ -FEM sera d'implémenter ces différents modèles sous *FEniCS* afin de comparer les résultats obtenus pour ces différents modèles.

### 4.3.1 Modèle de Saint-Venant-Kirchhoff

Pour ce modèle, le potentiel d'énergie sera donné par

$$\Psi = \frac{1}{2}\lambda(\text{tr } E)^2 + \mu(E : E). \quad (90)$$

Ainsi, le deuxième tenseur de Piola-Kirchoff est donné par :

$$S = \frac{\partial \Psi}{\partial E} = \lambda \text{tr}(E)I + 2\mu E. \quad (91)$$

Ce modèle est en fait une généralisation aux matériaux hyperélastiques, du modèle utilisé en élasticité linéaire. Cela se remarque notamment lors du calcul du tenseur d'élasticité  $\mathbf{C} = \frac{\partial S}{\partial E}$  :

$$C_{ijkl} = \lambda I_{ij}I_{kl} + \mu(I_{ik}I_{jl} + I_{il}I_{jk}), \quad (92)$$

qui correspond au tenseur défini à l'équation (66) page 26.

### 4.3.2 Modèle néo-hookéen

Définissons maintenant un modèle légèrement plus complexe mais faisant toujours intervenir seulement les invariants de  $\mathbf{C}$ , le modèle néo-hookéen. Ici, le potentiel d'énergie s'écrit sous la forme

$$\Psi = \frac{\mu}{2}(\text{tr } \mathbf{C} - 3) + \frac{\lambda}{2}(\ln J)^2 - \mu \ln J = \frac{\mu}{2}(I_1 - 3) + \frac{\lambda}{8}(\ln I_3)^2 - \frac{\mu}{2} \ln I_3. \quad (93)$$

Ainsi le second tenseur de Piola-Kirchhoff s'écrit  $S$  :

$$S = 2 \frac{\partial \Psi}{\partial \mathbf{C}} = \mu(I - \mathbf{C}^{-1}) + \frac{\lambda}{2}(\ln I_3)\mathbf{C}^{-1}. \quad (94)$$

### 4.3.3 Modèle d'Ogden

Nous allons maintenant introduire un modèle plus complexe que les précédents. Dans ce modèle, le potentiel  $\Psi$  ne sera pas exprimé en fonction des invariants mais des valeurs propres de  $\mathbf{C}$ . Le potentiel s'écrit :

$$\Psi = \sum_{m=1}^N \frac{\mu_m}{\alpha_m} \left( L_1^{\frac{\alpha_m}{2}} + L_2^{\frac{\alpha_m}{2}} + L_3^{\frac{\alpha_m}{2}} - 3 \right), \quad (95)$$

où  $L_i$  sont les valeurs propres des invariants,  $N = 3$  généralement et  $\mu_m$  ainsi que  $\alpha_m$  sont des paramètres dépendant du matériau considéré.

Il est nécessaire, pour dériver le potentiel par rapport aux invariants, d'exprimer ces derniers en fonction des valeurs propres de  $\mathbf{C}$  :

$$\begin{cases} I_1 = \text{tr}(\mathbf{C}) \\ I_2 = \frac{1}{2}(I_1^2 - \text{tr}(\mathbf{C} \cdot \mathbf{C})) \\ I_3 = \det \mathbf{C} \end{cases} = \begin{aligned} &= L_1 + L_2 + L_3 \\ &= L_1 L_2 + L_1 L_3 + L_2 L_3 \\ &= L_1 L_2 L_3. \end{aligned} \quad (96)$$

Le tenseur  $S$  peut être exprimé dans une base orthonormale de vecteurs propres  $N_i$ , tout comme le tenseur identité, le tenseur  $\mathbf{C}$  et son inverse,

$$I = \sum_{i=1}^3 N_i \otimes N_i, \quad (97)$$

$$\mathbf{C} = \sum_{i=1}^3 L_i N_i \otimes N_i, \quad (98)$$

$$\mathbf{C}^{-1} = \sum_{i=1}^3 \frac{1}{L_i} N_i \otimes N_i. \quad (99)$$

En remplaçant dans l'expression (88),

$$\mathbf{S} = 2 \sum_{i=1}^3 \left( \frac{\partial \Psi}{\partial I_1} + \frac{\partial \Psi}{\partial I_2} (I_1 - L_i) + \frac{\partial \Psi}{\partial I_3} \frac{I_3}{L_1} \right) N_i \otimes N_i.$$

Puisque  $\frac{\partial I_1}{\partial L_i} = 1$ ,  $\frac{\partial I_2}{\partial L_i} = I_1 - L_i$  et  $\frac{\partial I_3}{\partial L_i} = \frac{I_3}{L_i}$  pour  $i = 1, 2, 3$ , on a alors :

$$\begin{aligned} \mathbf{S} &= 2 \sum_{i=1}^3 \left( \frac{\partial \Psi}{\partial I_1} \frac{\partial I_1}{\partial L_i} + \frac{\partial \Psi}{\partial I_2} \frac{\partial I_2}{\partial L_i} + \frac{\partial \Psi}{\partial I_3} \frac{\partial I_3}{\partial L_i} \right) N_i \otimes N_i \\ &= 2 \sum_{i=1}^3 \frac{\partial \Psi}{\partial L_i} N_i \otimes N_i. \end{aligned}$$

#### 4.3.4 Modèle de Gent

Nous allons maintenant définir le dernier modèle auquel nous nous intéresserons. Ce modèle est également plus complexe que le modèle de Saint-Venant-Kirchhoff et est un modèle fréquemment utilisé en élasticité pour les matériaux caoutchouteux. Pour ce modèle, le potentiel est donné par

$$\Psi = (-E/6) J_m \ln \left( 1 - \frac{J_1 - 3}{J_m} \right),$$

où

$$J_1 = I_1 J^{-2/3},$$

$E$  est le module de Young et  $J_m = I_m - 3$  où  $I_m$  est la valeur limite du premier invariant de  $\mathbf{C}$ .

*Remarque 4.4.*

- Le modèle de Gent, contrairement notamment au modèle Néo-Hookeen, ne montre aucune linéarité lors de grandes déformations. Il modélise ainsi très précisément les grandes déformations non-linéaires.
- Il est intéressant de noter que le cas limite où  $I_m \rightarrow +\infty$ , revient à considérer le modèle Néo-Hookeen précédemment introduit.

*Remarque 4.5.* Nous ajouterons parfois dans les lois de comportement le terme

$$c \times (J - 1)^2,$$

afin de pénaliser le déterminant de  $F$  et donc la perte de volume, de façon plus ou moins élevée, dépendant de la constante  $c$ .

*Remarque 4.6.* Ces différents modèles ont été présentés car le but est ensuite d'adapter la méthode  $\phi$ -FEM et de vérifier les résultats pour chacun d'entre eux. Pour cela, il faudra notamment déterminer les paramètres qui permettent d'obtenir une bonne convergence. Malheureusement, nous n'avons pour l'instant pas eu le temps de réaliser ces études numériques.

### 4.4 Résolution numérique

Maintenant que nous avons introduit les points importants, nous allons pouvoir définir le problème considéré. Pour cela, nous allons ici parler seulement du cas statique, le cas dynamique étant dans un premier temps trop complexe à considérer. Le but sera ainsi de déterminer le déplacement d'un domaine lorsqu'une force lui sera appliquée. Soient  $\lambda$  et  $\mu$  des constantes dépendant du matériau considéré (constantes de Lamé). Soient  $F$ ,  $\mathbf{C}$ ,  $J$  et  $I$  les différents tenseurs introduits précédemment et définis par

$$\begin{cases} F &= I + \nabla u, \\ \mathbf{C} &= F^T F, \\ J &= \det F, \\ I &= \text{tr} (\mathbf{C}). \end{cases} \quad (100)$$

Considérons dans un premier temps une loi de comportement pour un matériau Néo-Hookeen, définie par

$$\Psi = \frac{\mu}{2}(I - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2. \quad (101)$$

Rappelons que le premier tenseur de Piola-Kirchhoff s'écrit

$$\mathbf{\Pi} = \frac{\partial \Psi}{\partial F}.$$

Soient une nouvelle fois  $\Omega \subset \mathbb{R}^d$ , de frontière  $\Gamma = \Gamma_N \cup \Gamma_D$  vérifiant  $\mathring{\Gamma}_D \cap \mathring{\Gamma}_N = \emptyset$ . Le problème hyperélastique à résoudre est alors donné par

$$\begin{cases} -\operatorname{div} \mathbf{\Pi}(\mathbf{u}) = \mathbf{f}, & \text{dans } \Omega, \\ \mathbf{u} = \mathbf{u}_D, & \text{sur } \Gamma_D, \\ \mathbf{\Pi}(\mathbf{u}) \cdot \mathbf{n} = \mathbf{T}, & \text{sur } \Gamma_N. \end{cases} \quad (102)$$

Précisons le sens physique des différentes variables  $\mathbf{\Pi}$ ,  $\mathbf{f}$ ,  $\mathbf{T}$  et  $\mathbf{u}_D$ . Premièrement, dans ce problème, le premier tenseur de Piola-Kirchhoff (abrégé parfois PK1), représentera ici l'ensemble des forces internes dans  $\Omega$ , quant  $\mathbf{f}$  représentera l'ensemble des forces externes. Les termes  $\mathbf{u}_D$  et  $\mathbf{T}$  sont eux respectivement la contrainte de déplacement et la contrainte de traction.

Maintenant que nous avons défini le problème, nous allons pouvoir introduire la méthode utilisée pour résoudre de tels problèmes. Un point important est à noter : comme nous l'avons dit précédemment, dans le cadre de l'hyperélasticité, les déformations subies ne sont pas linéaires. Il sera donc nécessaire de résoudre les problèmes avec une méthode non-linéaire. C'est d'ailleurs la raison pour laquelle nous avons présenté la méthode standard et la méthode  $\phi$ -FEM pour le problème de Poisson non-linéaire, en 3.2.

#### 4.4.1 Méthode standard

Commençons par introduire comme précédemment la méthode standard, avant de présenter la méthode  $\phi$ -FEM utilisée pour résoudre le problème (102). Pour cela, nous utilisons ici la technique habituelle : multiplier par une fonction test  $\mathbf{v}$  puis intégrer par parties. Soit ainsi  $\mathbf{v} \in V$  une fonction test, avec

$$V = \{\mathbf{v} \in H^1(\Omega)^d \mid \mathbf{v} = 0 \text{ sur } \Gamma_D\}.$$

Alors, multiplier l'équation par cette fonction test transforme le problème, qui devient : chercher  $\mathbf{u} \in H^1(\Omega)^d$  telle que

$$\int_{\Omega} -\operatorname{div} \mathbf{\Pi}(\mathbf{u}) \cdot \mathbf{v} = \int_{\Omega} \mathbf{f} \cdot \mathbf{v}, \quad \forall \mathbf{v} \in V,$$

ce qui après une intégration par parties et l'utilisation des conditions de bord, revient à chercher  $\mathbf{u} \in H^1(\Omega)^d$  telle que

$$\int_{\Omega} \mathbf{\Pi}(\mathbf{u}) : \nabla \mathbf{v} = \int \mathbf{f} \cdot \mathbf{v} - \underbrace{\int_{\Gamma_D} (\mathbf{\Pi}(\mathbf{u}) \cdot \mathbf{n}) \cdot \mathbf{v}}_{=0} + \int_{\Gamma_N} \mathbf{T} \cdot \mathbf{v}, \quad \forall \mathbf{v} \in V.$$

Il suffit alors de réécrire le problème en mettant tous les termes dans le membre de gauche puis de résoudre

$$F(\mathbf{u}; \mathbf{v}) = 0,$$

où

$$F(\mathbf{u}; \mathbf{v}) = \int_{\Omega} \mathbf{\Pi}(\mathbf{u}) : \nabla \mathbf{v} - \int \mathbf{f} \cdot \mathbf{v} - \int_{\Gamma_N} \mathbf{T} \cdot \mathbf{v}.$$

*Remarque 4.7* (Deuxième méthode équivalente). Cette méthode peut également s'appliquer sous une forme légèrement différente. Pour cette méthode, nous chercherons à minimiser l'énergie totale, sans utiliser le premier tenseur de Piola-Kirchhoff. A la place, nous ferons appel directement à la loi de comportement

utilisée. Le problème revient alors à trouver le déplacement  $\mathbf{u} \in H^1(\Omega)^d$  minimisant  $\mathbf{P}$ , où  $\mathbf{P}$  est l'énergie potentielle totale, c'est-à-dire,

$$\mathbf{P} = \int_{\Omega} \Psi(\mathbf{u}) - \int_{\Omega} \mathbf{f} \cdot \mathbf{u} - \int_{\Gamma_N} \mathbf{T} \cdot \mathbf{u}, \quad (103)$$

avec  $\Psi$  la loi de comportement.

Pour minimiser cette expression il suffit alors de dériver ce potentiel par rapport à la fonction  $\mathbf{u}$ , dans la direction d'une fonction test  $\mathbf{v} \in V$ , ce qui nous permet de définir  $L$ , donnée par

$$L(\mathbf{u}; \mathbf{v}) = D_{\mathbf{v}} \mathbf{P}.$$

Il suffit alors de chercher à annuler cette dérivée afin de trouver le minimum, c'est-à-dire résoudre

$$L(\mathbf{u}; \mathbf{v}) = 0, \quad \forall \mathbf{v} \in V.$$

#### 4.4.2 Construction des schémas $\phi$ -FEM

Dans cette partie, nous allons présenter les premiers résultats obtenus avec la méthode  $\phi$ -FEM pour résoudre des problèmes hyperélastiques. Ces résultats sont le fruit de plusieurs semaines de travail intense, en collaboration avec Sidiaty El Hadramy<sup>11 12</sup>.

Nous allons donc chercher à construire un schéma  $\phi$ -FEM pour résoudre des problèmes hyperélastiques. Pour cela, nous allons procéder en utilisant les idées introduites dans 3.2 et 4.1.2. Rappelons premièrement que nous cherchons à déterminer un déplacement  $\mathbf{u} \in H^1(\Omega)^d$  tel que

$$\begin{cases} -\operatorname{div} \Pi(\mathbf{u}) = \mathbf{f}, & \text{dans } \Omega, \\ \mathbf{u} = \mathbf{u}_D, & \text{sur } \Gamma_D, \\ \Pi(\mathbf{u}) \cdot \mathbf{n} = \mathbf{g}, & \text{sur } \Gamma_N, \end{cases} \quad (104)$$

où  $\Gamma = \Gamma_N \cup \Gamma_D$  (avec  $\Gamma_N \cap \Gamma_D = \emptyset$ ) est la frontière de  $\Omega$ .

Comme pour les problèmes de Poisson ou d'élasticité linéaire, nous allons tout d'abord introduire des variables auxiliaires afin d'imposer les conditions de bord. Il est ainsi nécessaire dans un premier temps d'introduire une variable auxiliaire pour imposer les conditions de Dirichlet sur  $\Omega_h^{\Gamma_D}$ . Soit donc  $\mathbf{p}_D$ , telle que

$$\mathbf{u} = \phi \mathbf{p}_D + \mathbf{u}_D, \quad \text{sur } \Omega_h^{\Gamma_D}. \quad (105)$$

Soient maintenant  $\mathbf{y}$  et  $\mathbf{p}_N$  telles que

$$\mathbf{y} + \Pi(\mathbf{u}) = 0, \quad \text{sur } \Omega_h^{\Gamma_N}, \quad (106a)$$

$$\mathbf{y} \nabla \phi + \mathbf{p}_N \phi = -\mathbf{g} |\nabla \phi|, \quad \text{sur } \Omega_h^{\Gamma_N}. \quad (106b)$$

Comme dans le cas de l'élasticité linéaire ces variables sont donc tensorielles et vectorielles. Nous rappelons donc les différents espaces considérés puisqu'ils sont définis comme dans le cadre de l'élasticité linéaire, c'est-à-dire,

$$\begin{aligned} V_h^{(k)} &:= \left\{ \mathbf{v}_h : \Omega_h \rightarrow \mathbb{R}^d : \mathbf{v}_{h|T} \in \mathbb{P}^k(T)^d \quad \forall T \in \mathcal{T}_h, \mathbf{v}_h \text{ continue sur } \Omega_h \right\}, \\ Z_h(\mathcal{M}_h) &:= \left\{ \mathbf{z}_h : \mathcal{M}_h \rightarrow \mathbb{R}^{(d \times d)} : \mathbf{z}_{h|T} \in \mathbb{P}^k(T)^{(d \times d)} \quad \forall T \in \mathcal{M}_h, \mathbf{z}_h \text{ continue sur } \mathcal{M}_h \right\}, \\ Q_h^l(\mathcal{M}_h) &:= \left\{ \mathbf{q}_h : \mathcal{M}_h \rightarrow \mathbb{R}^d : \mathbf{q}_{h|T} \in \mathbb{P}^l(T)^d \quad \forall T \in \mathcal{M}_h, \mathbf{q}_h \text{ discontinue sur } \mathcal{M}_h \text{ si } l \geq 0 \right\}. \end{aligned}$$

Nous pouvons alors introduire les équations (105), (106a) et (106b) comme nous l'avons fait pour les schémas précédents. Cela nous donne alors le problème suivant : trouver  $\mathbf{u}_h \in V_h$ ,  $\mathbf{p}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D})$ ,  $\mathbf{y}_h \in Z_h(\Omega_h^{\Gamma_N})$  et  $\mathbf{p}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N})$  tels que

11. <https://mimesis.inria.fr/members/sidaty-el-hadramy/>

12. <https://github.com/Sidaty1>

$$\begin{aligned}
& \int_{\Omega_h} \boldsymbol{\Pi}(\mathbf{u}_h) : \nabla \mathbf{v}_h - \int_{\partial\Omega_h \setminus \partial\Omega_{h,N}} \boldsymbol{\Pi}(\mathbf{u}_h) \mathbf{n} \cdot \mathbf{v}_h + \int_{\partial\Omega_{h,N}} \mathbf{y}_h \mathbf{n} \cdot \mathbf{v}_h \\
& + \gamma_u \int_{\Omega_h^{\Gamma_N}} (\mathbf{y}_h + \boldsymbol{\Pi}(\mathbf{u}_h)) : (\mathbf{z}_h + \textcolor{red}{\boldsymbol{\Pi}(\mathbf{v}_h)}) + \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \left( \mathbf{y}_h \nabla \phi_h + \frac{1}{h} \mathbf{p}_{h,N} \phi_h \right) \cdot \left( \mathbf{z}_h \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h \right) \\
& + \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (\mathbf{u}_h - \phi_h \mathbf{p}_{h,D}) \cdot (\mathbf{v}_h - \phi_h \mathbf{q}_{h,D}) + \sigma_p h^{-1} \int_{\mathcal{F}_h^{\Gamma_D}} [\phi_h \mathbf{p}_{h,D}] [\phi_h \mathbf{q}_{h,D}] \\
& + G_h(\mathbf{u}_h, \mathbf{v}_h) + J_h^{lhs,D}(\mathbf{u}_h, \mathbf{v}_h) + J_h^{lhs,N}(\mathbf{y}_h, \mathbf{z}_h) = \int_{\Omega_h} \mathbf{f} \cdot \mathbf{v}_h \\
& + \frac{\gamma}{h^2} \int_{\Omega_h^D} \mathbf{u}_h^g \cdot (\mathbf{v}_h - \phi_h \mathbf{q}_{h,D}) - \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \mathbf{g} \cdot |\nabla \phi_h| (\mathbf{z}_h \cdot \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h) \\
& + J_h^{rhs,D}(\mathbf{v}_h) + J_h^{rhs,N}(\mathbf{z}_h), \\
& \forall \mathbf{v}_h \in V_h, \mathbf{q}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D}), \mathbf{z}_h \in Z_h(\Omega_h^{\Gamma_N}), \mathbf{q}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N}). \quad (107)
\end{aligned}$$

où

$$\begin{aligned}
J_h^{lhs,D}(\mathbf{u}, \mathbf{v}) &:= \sigma_D h^2 \sum_{T \in \mathcal{T}_h^\Gamma \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \operatorname{div} \boldsymbol{\Pi}(\mathbf{u}) \cdot \operatorname{div} \textcolor{red}{\boldsymbol{\Pi}(\mathbf{v})}, \\
J_h^{rhs,D}(\mathbf{v}) &:= -\sigma_D h^2 \sum_{T \in \mathcal{T}_h^\Gamma \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \mathbf{f} \cdot \operatorname{div} \textcolor{red}{\boldsymbol{\Pi}(\mathbf{v})}, \\
J_h^{lhs,N}(\mathbf{y}, \mathbf{z}) &= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \operatorname{div} \mathbf{y} \cdot \operatorname{div} \mathbf{z}, & J_h^{rhs,N}(\mathbf{z}) &= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \mathbf{f} \cdot \operatorname{div} \mathbf{z}, \quad (108)
\end{aligned}$$

et

$$G_h(u, v) := \sigma_D h \sum_{E \in \mathcal{F}_h^{\Gamma_D}} \int_E [\boldsymbol{\Pi}(u) \cdot \mathbf{n}] \cdot [\textcolor{red}{\boldsymbol{\Pi}(v)} \cdot \mathbf{n}] + \sigma_N h \sum_{E \in \mathcal{F}_h^{\Gamma_N}} \int_E [\boldsymbol{\Pi}(u) \cdot \mathbf{n}] \cdot [\textcolor{red}{\boldsymbol{\Pi}(v)} \cdot \mathbf{n}].$$

Cependant, en construisant le schéma de cette manière un problème se pose : en effet, ici par définition du premier tenseur de Piola-Kirchhoff, les termes  $\boldsymbol{\Pi}(\mathbf{v})$  ne sont pas linéaires en  $\mathbf{v}$ . Or, pour établir le schéma φ-FEM, nous avons besoin que la forme considérée soit linéaire en la variable  $\mathbf{v}$ . Il est donc nécessaire de linéariser cette formulation. Pour cela, nous allons remplacer  $\boldsymbol{\Pi}(\mathbf{v})$  par

$$D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v} = \left. \frac{d\boldsymbol{\Pi}(\mathbf{u} + \varepsilon \mathbf{v})}{d\varepsilon} \right|_{\varepsilon=0},$$

la dérivée de  $\boldsymbol{\Pi}$  en  $\mathbf{u}$  dans la direction  $\mathbf{v}$ .

En introduisant alors les équations comme précédemment, et en remplaçant  $\boldsymbol{\Pi}(\mathbf{v})$  par  $D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v}$ , le schéma

devient alors, trouver  $\mathbf{u}_h \in V_h$ ,  $\mathbf{p}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D})$ ,  $\mathbf{y}_h \in Z_h(\Omega_h^{\Gamma_N})$  et  $\mathbf{p}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N})$  tels que

$$\begin{aligned} & \int_{\Omega_h} \boldsymbol{\Pi}(\mathbf{u}_h) : \nabla \mathbf{v}_h - \int_{\partial \Omega_h \setminus \partial \Omega_{h,N}} \boldsymbol{\Pi}(\mathbf{u}_h) \mathbf{n} \cdot \mathbf{v}_h + \int_{\partial \Omega_{h,N}} \mathbf{y}_h \mathbf{n} \cdot \mathbf{v}_h \\ & + \gamma_u \int_{\Omega_h^{\Gamma_N}} (\mathbf{y}_h + \boldsymbol{\Pi}(\mathbf{u}_h)) : (\mathbf{z}_h + D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}_h) \mathbf{v}_h) + \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \left( \mathbf{y}_h \nabla \phi_h + \frac{1}{h} \mathbf{p}_{h,N} \phi_h \right) \cdot \left( \mathbf{z}_h \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h \right) \\ & + \frac{\gamma}{h^2} \int_{\Omega_h^{\Gamma_D}} (\mathbf{u}_h - \phi_h \mathbf{p}_{h,D}) \cdot (\mathbf{v}_h - \phi_h \mathbf{q}_{h,D}) + \sigma_p h^{-1} \int_{\mathcal{F}_h^{\Gamma_D}} [\phi_h \mathbf{p}_{h,D}] [\phi_h \mathbf{q}_{h,D}] \\ & + G_h(\mathbf{u}_h, \mathbf{v}_h) + J_h^{lhs,D}(\mathbf{u}_h, \mathbf{v}_h) + J_h^{rhs,N}(\mathbf{y}_h, \mathbf{z}_h) = \int_{\Omega_h} \mathbf{f} \cdot \mathbf{v}_h \\ & + \frac{\gamma}{h^2} \int_{\Omega_h^D} \mathbf{u}_h^g \cdot (\mathbf{v}_h - \phi_h \mathbf{q}_{h,D}) - \frac{\gamma_p}{h^2} \int_{\Omega_h^{\Gamma_N}} \mathbf{g} \cdot |\nabla \phi_h| (\mathbf{z}_h \cdot \nabla \phi_h + \frac{1}{h} \mathbf{q}_{h,N} \phi_h) \\ & + J_h^{rhs,D}(\mathbf{v}_h) + J_h^{rhs,N}(\mathbf{z}_h), \\ & \forall \mathbf{v}_h \in V_h, \mathbf{q}_{h,D} \in Q_h^k(\Omega_h^{\Gamma_D}), \mathbf{z}_h \in Z_h(\Omega_h^{\Gamma_N}), \mathbf{q}_{h,N} \in Q_h^{k-1}(\Omega_h^{\Gamma_N}), \quad (109) \end{aligned}$$

où

$$\begin{aligned} J_h^{lhs,D}(\mathbf{u}, \mathbf{v}) &:= \sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \operatorname{div} \boldsymbol{\Pi}(\mathbf{u}) \cdot \operatorname{div}(D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v}), \\ J_h^{rhs,D}(\mathbf{v}) &:= -\sigma_D h^2 \sum_{T \in \mathcal{T}_h^{\Gamma} \setminus \mathcal{T}_h^{\Gamma_N}} \int_T \mathbf{f} \cdot \operatorname{div}(D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v}), \\ J_h^{lhs,N}(\mathbf{y}, \mathbf{z}) &= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \operatorname{div} \mathbf{y} \cdot \operatorname{div} \mathbf{z}, & J_h^{rhs,N}(\mathbf{z}) &= \gamma_{div} \int_{\Omega_h^{\Gamma_N}} \mathbf{f} \cdot \operatorname{div} \mathbf{z}, \quad (110) \end{aligned}$$

et

$$G_h(\mathbf{u}, \mathbf{v}) := \sigma_D h \sum_{E \in \mathcal{F}_h^{\Gamma_D}} \int_E [\boldsymbol{\Pi}(\mathbf{u}) \cdot \mathbf{n}] \cdot [D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v} \cdot \mathbf{n}] + \sigma_N h \sum_{E \in \mathcal{F}_h^{\Gamma_N}} \int_E [\boldsymbol{\Pi}(\mathbf{u}) \cdot \mathbf{n}] \cdot [D_{\mathbf{u}} \boldsymbol{\Pi}(\mathbf{u}) \mathbf{v} \mathbf{n}] . \quad (111)$$

Maintenant que nous avons défini notre schéma, il ne reste plus qu'à le mettre sous la forme vue précédemment, afin de résoudre le problème

$$F(\mathbf{u}, \mathbf{v}) = 0.$$

Il suffit alors de passer tous les termes du second membre dans le membre de gauche, et nous obtenons ainsi la forme souhaitée.

*Remarque 4.8.* Le schéma (109), comme le schéma (56) ont tous deux été introduits dans un premier temps sous la forme habituelle pour les problèmes linéaires, c'est-à-dire sous la forme

$$a(\mathbf{u}, \mathbf{v}) = l(\mathbf{v}).$$

Cela permet de comparer ces schémas avec les autres schémas introduits pour les problèmes linéaires. Cependant, il est important de noter qu'en pratique, ces formulations ne sont pas utilisables. En effet, dans ces deux schémas, le membre de gauche n'est pas une forme bilinéaire, puisque non-linéaire en  $\mathbf{u}$ . C'est d'ailleurs la raison pour laquelle il est nécessaire par la suite de résoudre le problème en utilisant la forme  $F(\mathbf{u}, \mathbf{v}) = 0$ , à l'aide d'un solveur non-linéaire. Les schémas (109) et (56) ont donc été introduits sous cette forme, par analogie avec tous ceux présentés précédemment.

## 4.5 Résultats numériques sur des domaines simples

### 4.5.1 Le cas 2D

Nous allons maintenant nous intéresser à la convergence numérique de ce nouveau schéma. Pour cela, nous considérerons le cas test :

- $\Omega$  est le cercle de centre  $(0.5, 0.5)$  et de rayon  $\sqrt{2}/4$ ,
- la fonction level-set  $\phi$  est donnée par

$$\phi(x, y) := -\frac{1}{8} + (x - 0.5)^2 + (y - 0.5)^2, \quad (112)$$

- les bords  $\Gamma_N$  et  $\Gamma_D$  sont donnés par  $\psi(x, y) = -x - 0.5$ ,
- la solution manufacturée est donnée par

$$\mathbf{u}_{ex} = (3 \cos(x/90) \times \sin(y/90), 3 \cos(x/90) \times \sin(y/90)),$$

- les conditions de bord sont données par

$$\mathbf{u}_D = (1 + \phi) \times \mathbf{u}_{ex}, \quad \text{et} \quad \mathbf{g} = \frac{(\Pi(\mathbf{u}_{ex}), \nabla \phi)}{|\nabla \phi|} + \phi \mathbf{u}_{ex}.$$

Les résultats obtenus pour ce cas test avec l'utilisation d'une méthode standard et de la méthode  $\phi$ -FEM sont ainsi représentés à la figure 15. Nous avons également représenté l'erreur obtenue pour le même cas test en considérant cette fois l'ellipse au lieu du cercle. Dans les deux cas, nous pouvons remarquer que, numériquement la convergence semble suivre la convergence théorique annoncée pour le problème de Poisson avec des conditions de Dirichlet ou des conditions de Neumann.

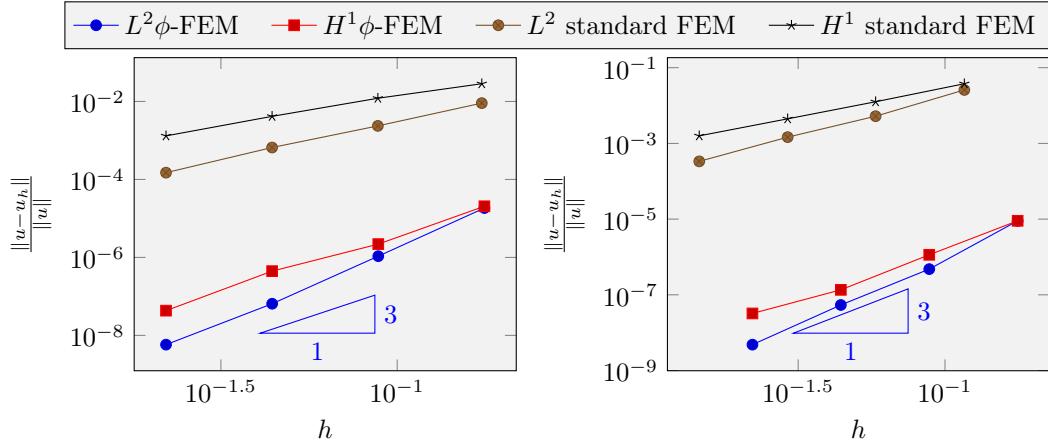


FIGURE 15 – A gauche : éléments finis  $\mathbb{P}^2$ ,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 1.0, 1.0, 20.01, 0.01, 20000.0, 0.2, 0.01$  sur le cercle. A droite : éléments finis  $\mathbb{P}^2$  sur l'ellipse,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 10.0, 1.0, 0.01, 0.01, 200000.0, 0.2, 0.001$ .

Nous avons également choisi de considérer un second cas. En effet, maintenant que nous avons pu valider numériquement la convergence de la méthode, il semblait intéressant d'appliquer cette dernière à un cas sans solution manufacturée. Pour ce faire, nous avons considéré le cas de l'ellipse présentée à la figure 7(a), pour laquelle nous avons choisi d'appliquer

- une force constante sur tout le domaine,  $f = (0, 0)$ ,
- un déplacement nul sur la frontière  $\Gamma_D = \Gamma \cap \{x > 0.3\} : u_D = (0, 0)$ ,
- une traction sur la frontière  $\Gamma_N = \Gamma \cap \{x < 0.3\} : g = (-200, 0)$ .

Pour cela, nous avons choisi de considérer un matériau Néo-Hookéen de module de Young  $E = 5000 Pa$  et de coefficient de Poisson  $\nu = 0.4$ . Le maillage déformé est ainsi représenté à la figure 16, ainsi que le maillage non déformé.

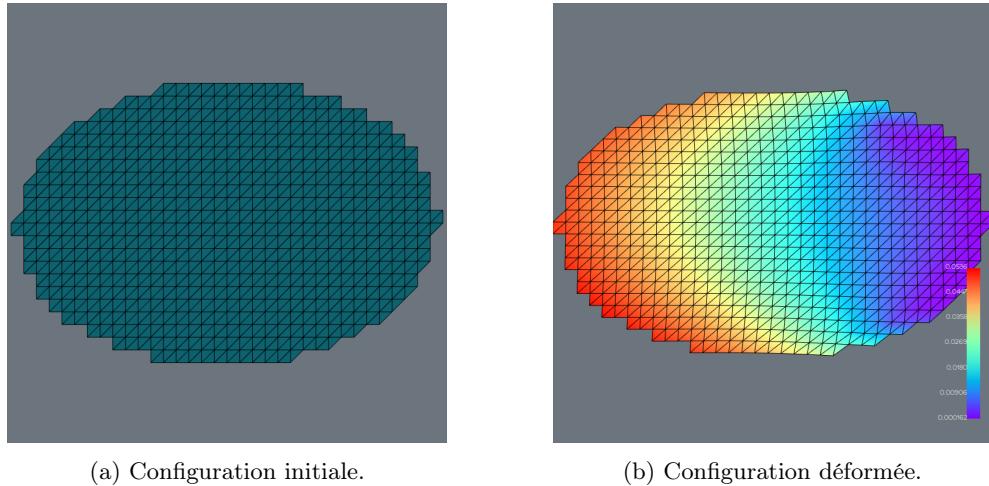


FIGURE 16 – Déformation d'une ellipse.

#### 4.5.2 Le cas 3D : une sphère

Nous allons maintenant étendre ces résultats numériques au cas 3D. Pour cela, nous considérons le cas test suivant, qui est seulement une extension du cas test vu précédemment :

- $\Omega$  est la sphère de centre  $(0.5, 0.5, 0.5)$  et de rayon  $\sqrt{2}/4$ ,
- la fonction level-set  $\phi$  est donnée par

$$\phi(x, y) := -\frac{1}{8} + (x - 0.5)^2 + (y - 0.5)^2 + (z - 0.5)^2, \quad (113)$$

- les bords  $\Gamma_N$  et  $\Gamma_D$  sont donnés par  $\psi(x, y, z) = -x - 0.5$ ,
- la solution manufacturée est donnée par

$$\mathbf{u}_{ex} = \begin{pmatrix} 3 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \\ 3 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \\ 3 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \end{pmatrix},$$

- les conditions de bord sont données par

$$\mathbf{u}_D = (1 + \phi) \times \mathbf{u}_{ex}, \quad \text{et} \quad \mathbf{g} = \frac{(\mathbf{\Pi}(\mathbf{u}_{ex}), \nabla \phi)}{|\nabla \phi|} + \phi \mathbf{u}_{ex}.$$

Nous représentons les résultats à la figure 17 où nous avons également représenté la déformation du maillage en lui appliquant le déplacement exact, ainsi que les forces appliquées pour obtenir ce déplacement. Une nouvelle fois, les résultats semblent suivre la théorie annoncée, en tenant du compte du choix de paramètres qui n'est pas forcément idéal ici. En effet, par manque de temps, nous n'avons pas pu déterminer les meilleurs paramètres.

**Difficultés rencontrées.** Il est important de noter que l'implémentation de cette méthode a généré énormément de difficultés. C'est notamment à ce moment que nous avons constaté des problèmes dans le schéma présenté pour les conditions mixtes, et avons corrigé ces erreurs en modifiant le schéma comme présenté dans (39) ou (104). Malheureusement, trouver l'erreur a pris énormément de temps puisque d'autres problèmes lors de l'implémentation numériques cachaient le réel problème. Ainsi, après avoir cherché une solution avec *FEniCS*, nous avons eu l'idée d'essayer d'utiliser *FEniCSX*<sup>13</sup> en pensant que cela permettrait possiblement de résoudre les problèmes que nous rencontrions. Cependant nous avons à ce moment fait face à un nouveau

13. <https://fenicsproject.org/>

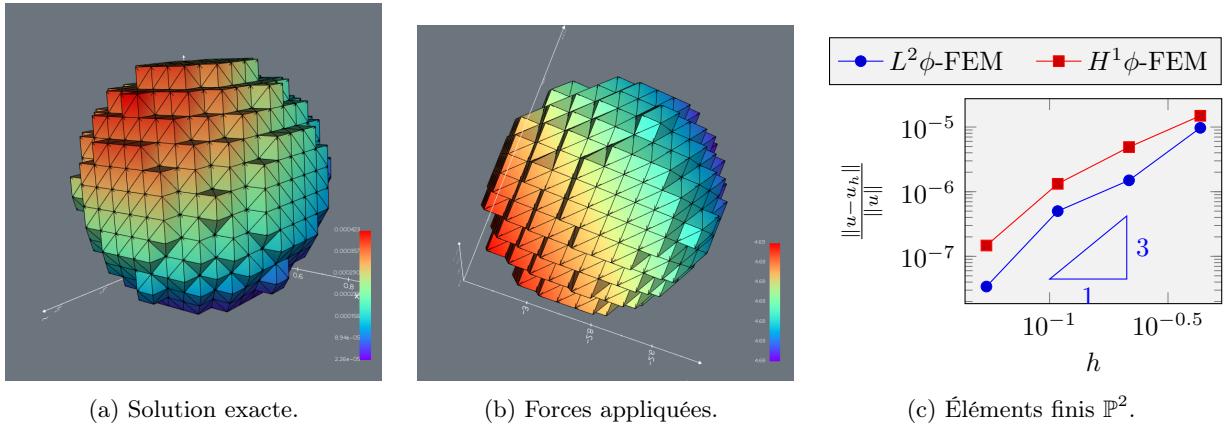


FIGURE 17 – Cas test d'une sphère hyperélastique.

problème : adapter le code  $\phi$ -FEM que nous avions afin de l'utiliser avec *FEniCSX* était plus complexe que prévu. En particulier, différentes fonctions utilisées dans les codes *FEniCS* n'existent plus avec *FEniCSX*. Nous avons ainsi cherché à modifier le code source disponible, afin d'ajouter les fonctions nécessaires.

Les fonctions qu'il était nécessaire de recréer étaient les fonctions de sélection de cellules, afin de créer dans un premier temps le maillage  $\mathcal{T}_h$  puis les maillages  $\mathcal{T}_h^{\Gamma_N}$  et  $\mathcal{T}_h^{\Gamma_D}$ . En effet, le premier problème survenait lors de la construction du maillage  $\phi$ -FEM : lorsque nous construisons ce maillage, toutes les cellules dont l'un des noeuds  $x$  vérifie  $\phi(x) \leq 0$  est sélectionnée pour être dans le maillage. Cependant, dans la nouvelle version, la construction de sous-maillage se faisait différemment : d'après les recherches que nous avons pu faire, il n'était plus possible de créer un sous-maillage de cette façon. Dans cette nouvelle version, lors de la création de sous-maillages, afin de sélectionner une cellule, tous les noeuds de cette dernière doivent vérifier une condition. Nous avons alors dû créer une fonction permettant de sélectionner les cellules comme nous en avions besoin. Le code créé à cette occasion est disponible en annexe B, à la figure 5.

Ce code nous a ainsi permis de créer le maillage  $\phi$ -FEM comme souhaité. Cependant, deux autres problèmes étaient toujours présents : une fois le maillage créé, il reste à créer les restrictions  $\mathcal{T}_h^{\Gamma_N}$  et  $\mathcal{T}_h^{\Gamma_D}$  et à sélectionner les cellules et faces où seront appliqués les termes de stabilisation du schéma. Pour localiser les cellules de la frontière ( $\mathcal{T}_h^{\Gamma_N}$  et  $\mathcal{T}_h^{\Gamma_D}$ ) il a ainsi été nécessaire de créer une autre méthode, dont le code est disponible en annexe B, 6. Enfin, il restait maintenant à sélectionner les facettes sur lesquelles appliquer les termes de saut, et donc pour cela il était nécessaire de déterminer si les facettes étaient dans la partie Dirichlet  $\Gamma_D$  ou dans la partie  $\Gamma_N$  et surtout il était important de sélectionner les facettes en intersection entre l'intérieur du maillage et les cellules de bord, ce qui n'était pas possible dans un premier temps.

Pour régler ce problème, une première idée a été de créer la méthode 7, cependant une solution plus simple, n'utilisant pas cette méthode a été choisie. Malheureusement, malgré tout cela, nous avons fini par retrouver le problème que nous avions avec *FEniCS*.

Nous avons alors compris à ce moment que le problème ne venait ni de *FEniCS*, ni de *FEniCSX* mais semblait venir du schéma utilisé. Nous avons alors décidé d'utiliser à nouveau *FEniCS* avec ce nouveau schéma pour éviter de modifier le code source. Suite aux modifications du schéma, nous avons pu réaliser différentes simulations numériques où nous nous sommes rendus compte de l'importance des paramètres de ce dernier. De plus, le choix de ces paramètres n'est pas aisné. En effet, certains permettront de fortement pénaliser des termes et donc d'obtenir parfois une très faible erreur mais en contrepartie, cela nuira à l'ordre de convergence et générera parfois des sauts de l'erreur (voir par exemple en annexe A, figure 32 pour le problème de Poisson).

## 4.6 Application à un problème réel : la déformation du foie

Nous nous plaçons maintenant dans un cadre plus concret : nous allons chercher à déterminer les déformations d'un foie, lorsque ce dernier est soumis à différentes forces. Pour cela, nous considérerons le maillage surfacique de foie représenté à la figure 18.

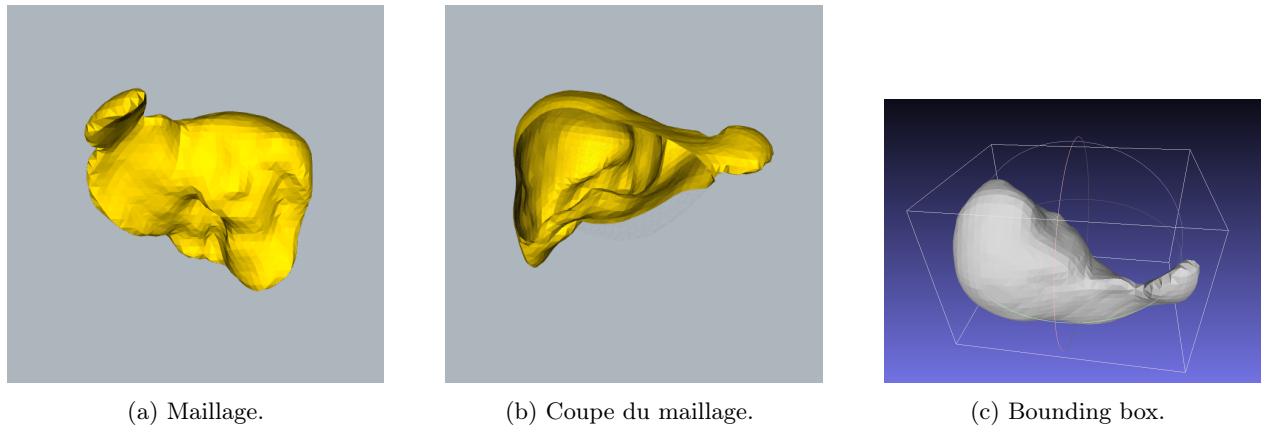


FIGURE 18 – Maillage surfacique de foie.

#### 4.6.1 Construction de la level-set

La première étape afin d'adapter notre méthode  $\phi$ -FEM a ainsi été d'améliorer une méthode introduite dans [19]. En effet, il a été nécessaire de construire une fonction level-set pour tout maillage donné. Pour cela nous avons donc repris l'idée d'utiliser la distance signée, qui semble tout à fait naturelle pour construire une telle fonction. Le but sera donc ici de déterminer une fonction level-set pour tout maillage surfacique, afin notamment de construire le maillage  $\phi$ -FEM. Le code de cette méthode est disponible à l'adresse : <https://github.com/Sidaty1/LevelSet.git>.

Pour cela, nous construisons tout d'abord une « bounding box » (voir figure 18(c)), afin de déterminer les dimensions du maillage qui sera le maillage de fond utilisé pour  $\phi$ -FEM. À partir de cette bounding box, nous pouvons ainsi créer un maillage régulier tétraédrique qui contient le maillage surfacique du foie. L'idée est alors d'évaluer la distance de chaque noeud du maillage de fond au maillage surfacique du foie. Pour cela, différentes étapes sont nécessaires : pour chaque noeud  $x$  du maillage de fond, il suffit de déterminer la distance à tous les noeuds du maillage surfacique et de garder la plus petite de ces distances. Cependant, il reste maintenant à déterminer le signe de cette distance : le point  $x$  du maillage de fond est-il à l'intérieur ou à l'extérieur du foie ? Pour cela, nous récupérons dans un premier temps la cellule du maillage du foie la plus proche de  $x$ . Cela permet alors de déterminer le vecteur normal unitaire extérieur à cette cellule afin d'ensuite simplement projeter le point  $x$  sur cette même cellule. Il suffit alors de calculer le vecteur  $xp_x$  où  $p_x$  est le projeté de  $x$  sur la cellule puis de calculer le produit scalaire entre ce vecteur et le vecteur normal. Ainsi, si ce produit scalaire est négatif, cela signifie que  $x$  est à l'intérieur du foie. Sinon, il est à l'extérieur.

*Remarque 4.9.* Un aspect négatif ici est le temps de calcul. En effet, ces différents calculs sont certes très simples, mais ils prennent beaucoup de temps : le nombre d'opérations à effectuer est très élevé. Cela engendre ainsi un temps de calcul très élevé puisque pour un maillage de fond avec  $N$  noeuds et un maillage surfacique avec  $M$  noeuds, la complexité sera de l'ordre de  $N \times M$ . Il était ainsi important de déterminer le temps de calcul des différentes étapes nécessaires à la création du maillage  $\phi$ -FEM. Pour cela, nous avons considéré différentes situations, en considérant un maillage surfacique de sphère. Pour la première situation, nous considérons différents maillages surfacique, en conservant le même maillage de fond. Le but ici est ainsi de déterminer l'évolution du temps de calcul lorsque le nombre de noeuds du maillage surfacique augmente. Pour cela, nous avons décomposé le temps de calcul global en 3 étapes :

- Initialisation de la classe `LevelSet`;
- Interpolation de la fonction sur le maillage de fond ;
- Création du sous-maillage  $\mathcal{T}_h$ , en parcourant toutes les cellules et en calculant  $\phi(x, y, z)$  en chaque noeud du maillage.

Les résultats indiquent donc que les temps de calcul sont relativement élevés. Cependant, il devrait être possible par la suite de réaliser ces constructions plus rapidement, notamment en parallélisant les calculs.

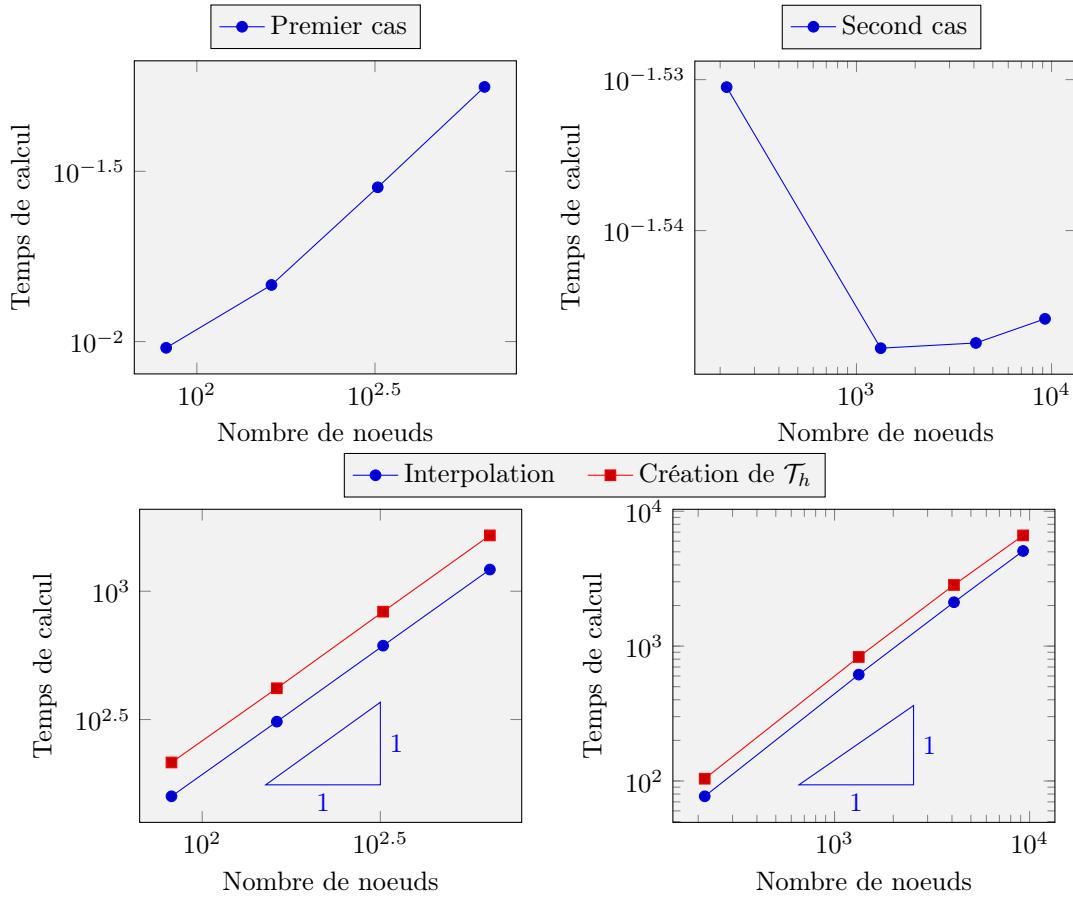


FIGURE 19 – Temps de calcul pour les différentes étapes. À gauche : les représentations pour le premier cas. À droite : les représentations pour le second cas.

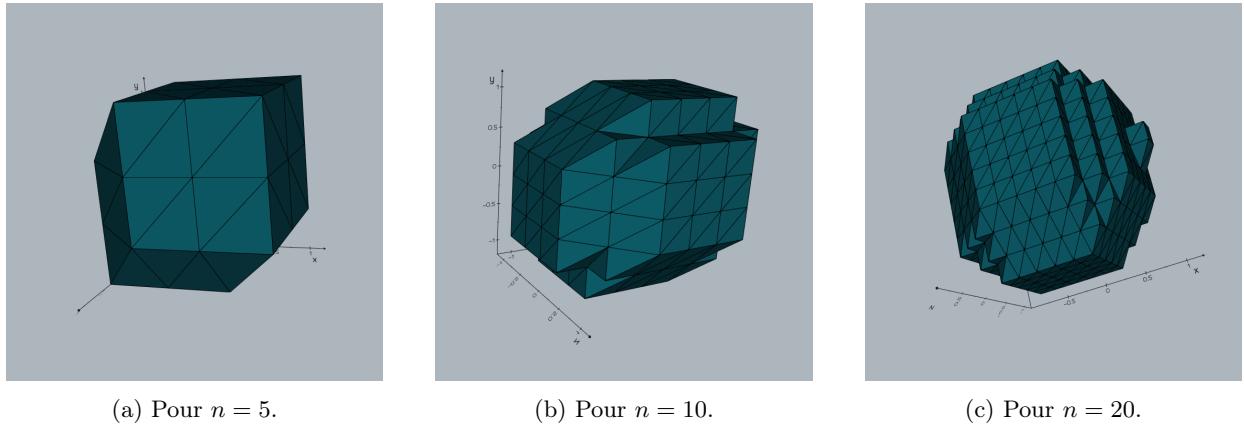


FIGURE 20 – Maillages de sphère reconstruits avec la fonction level-set, avec  $n$  le nombre d'hexaèdres dans chaque direction de l'espace.

Nous avons ainsi testé cette méthode dans deux cas : le premier, un cas simple sur un maillage surfacique de sphère. Cela nous a permis de valider la construction de la fonction level-set ainsi que la création du maillage utilisé pour  $\phi$ -FEM et de réaliser les représentations des figures 19 et 21. Différents exemples de maillages créés pour la sphère, pour plusieurs tailles de maillages de fond, sont représentés à la figure 20.

Enfin, il semblait également important de valider la création de cette fonction level-set en analysant l'erreur. Pour cela, nous avons considéré le cas du maillage surfacique d'une sphère, et nous avons calculé l'erreur à chaque noeud du maillage de fond entre l'expression exacte et l'expression recréée, avec comme expression exacte,

$$\phi_{ex}(x, y, z) = -1 + \sqrt{x^2 + y^2 + z^2}.$$

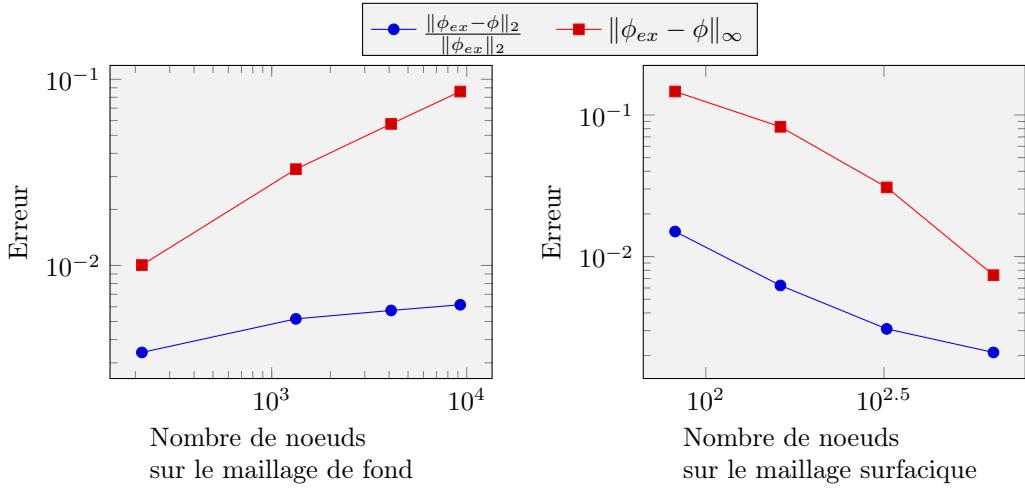


FIGURE 21 – À gauche : erreur de construction de  $\phi$ , pour différentes tailles de maillage de fond. À droite : erreur de construction de  $\phi$ , pour différentes tailles du maillage surfacique.

Après avoir testé cette méthode sur la création d'une sphère, nous avons donc pu l'étendre au cas du foie et l'appliquer au maillage de foie représenté à la figure 18. Cela nous a ainsi permis de créer les différents maillages représentés à la figure 22. Pour créer ces maillages, nous avons considéré le même maillage surfacique puis avons construit la boîte délimitée par les points  $(70, 117, -415)$  et  $(341, 301, -216)$ . Cette boîte a alors été divisée en  $n = 5, 10, 15, 20$  et 25 hexaèdres dans chacune des dimensions, eux-mêmes divisés en tétraèdres.

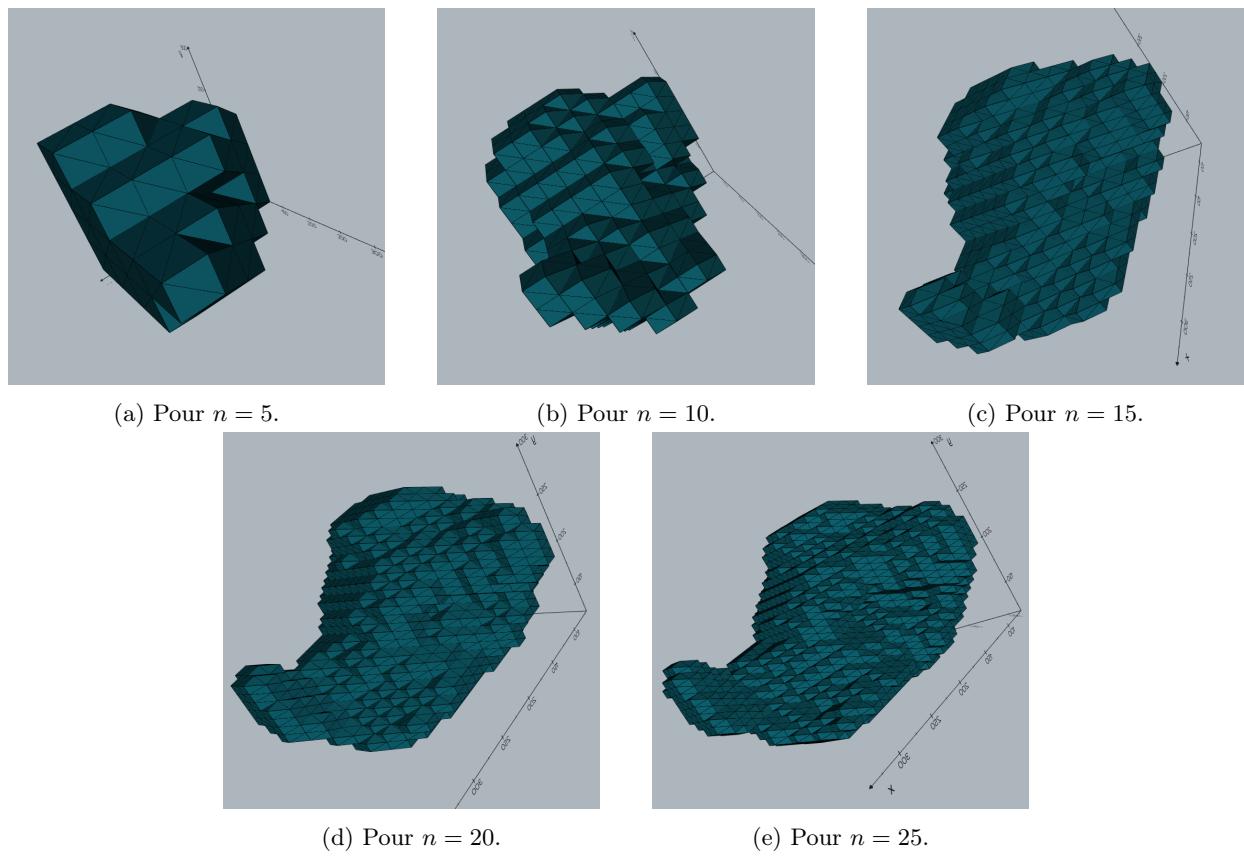


FIGURE 22 – Maillages de foie reconstruits avec la fonction level-set, avec  $n$  le nombre d'hexaèdres dans chaque direction de l'espace.

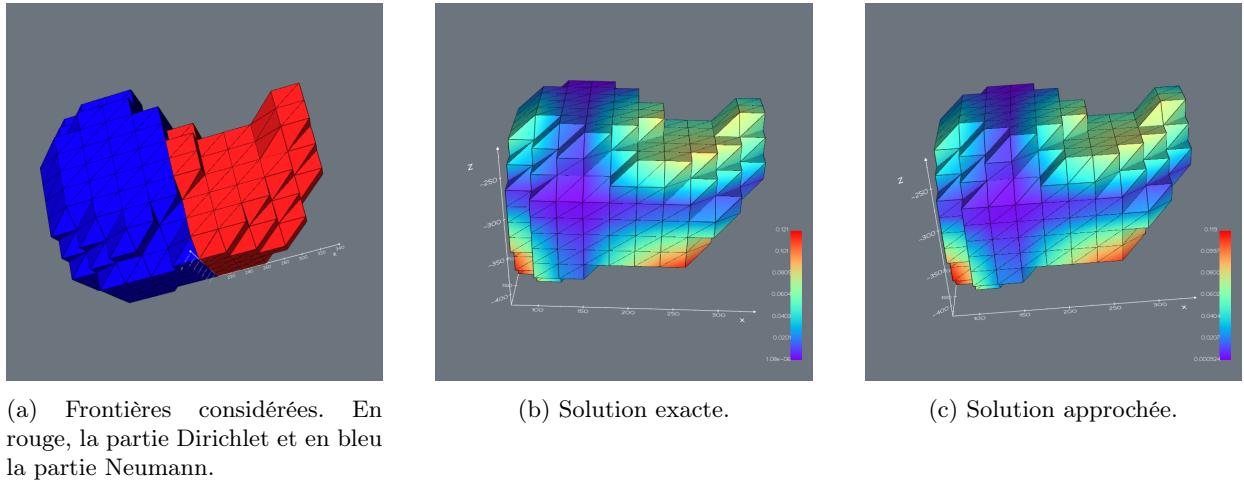


FIGURE 23 – Cas test du foie.

#### 4.6.2 Implémentation et résultats numériques

Le but a ensuite été d'utiliser les différents maillages précédemment créés afin de simuler la déformation de ces derniers, en utilisant la méthode  $\phi$ -FEM. Pour cela, il a été nécessaire de fixer les paramètres d'élasticité, la loi de comportement à considérer et les conditions de bord.

Nous avons ainsi considéré un module de Young de  $5000\text{Pa}$ , d'après [16], ainsi qu'un coefficient de Poisson de 0.4. Ensuite, nous avons choisi de considérer une loi de comportement Neo-Hookeenne, loi correspondant le mieux à la déformation du foie.

Enfin, il a fallu déterminer les sections du maillage à fixer (conditions de Dirichlet). Pour cela, nous avons fait le choix de fixer un ensemble de cellules (ce qui pour l'instant est un choix peu réaliste) : nous appliquerons des conditions de Dirichlet à toutes les cellules dont l'un des noeuds de coordonnées ( $x, y, z$ ) vérifie

$$201 < x .$$

Toutes les autres cellules seront des cellules du bord  $\Gamma_N$  (voir figure 23(a), où la partie Dirichlet est représentée en rouge et la partie Neumann en bleu). Enfin, pour ce cas test, nous avons choisi de considérer un petit déplacement comme solution exacte, donné par

$$u_{ex} = \begin{pmatrix} 0.1 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \\ 0.1 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \\ 0.1 \cos(x/90) \times \sin(y/90) \times \sin(z/90) \end{pmatrix} .$$

Nous avons alors obtenu les résultats représentés à la figure 24. Ces derniers ne sont pour l'instant pas aussi satisfaisants que ceux obtenus dans les cas précédents. Cependant, il est important de noter que ces résultats sont parmi les premiers obtenus sur ce cas test. En effet, par manque de temps, nous n'avons pas encore eu la possibilité de déterminer les meilleurs paramètres pour ce cas. Ainsi, bien que ces résultats ne soient pas encore réellement satisfaisants, ils restent tout de même très encourageants puisque l'ordre de convergence numérique semble tendre vers la pente attendue pour l'erreur  $L^2$  comme pour l'erreur  $H^1$ .

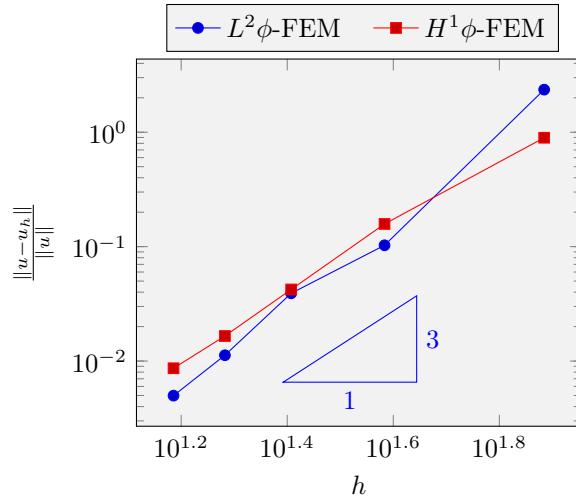


FIGURE 24 – Éléments finis  $\mathbb{P}^2$ ,  $\gamma_{div}, \gamma_u, \gamma_p, \sigma_N, \gamma_D, \sigma_D, \sigma_p = 10.0, 1.0, 10.01, 0.01, 200000.0, 0.02, 0.001$ .

## 5 L'implémentation de $\phi$ -FEM au sein de SOFA

Nous allons maintenant aborder le sujet de l'implémentation de la méthode  $\phi$ -FEM. Ce projet est un projet sur le long terme et les travaux qui vont être présentés par la suite ne sont qu'une ébauche de ce qui nous l'espérons sera d'ici quelques années une méthode complètement intégrée et utilisable aisément, avec des résultats précis et rapides. Nous allons donc dans un premier temps présenter l'environnement utilisé pour l'implémentation de cette méthode. Pour cela, nous présenterons rapidement l'historique et les évolutions du logiciel SOFA<sup>14</sup> ainsi que l'utilité de ce dernier. Nous présenterons ensuite le plugin utilisé pour l'implémentation de  $\phi$ -FEM : Caribou<sup>15</sup>. Nous en profiterons également pour présenter quelques simulations réalisées avec le plugin SofaPython3<sup>16</sup>, permettant de créer des scènes SOFA en python3.

### 5.1 L'environnement utilisé

Le logiciel SOFA est un logiciel open-source, créé afin de réaliser des simulations physiques. L'idée de ce logiciel date de 2000 et elle a été mise en oeuvre afin de permettre une première simulation en 2005. Depuis, le logiciel n'a cessé d'évoluer. Il a notamment été décrit précisément dans [1, 13] et depuis, la communauté travaillant sur SOFA ne cesse de grandir. En effet, de nombreuses publications utilisant des simulations SOFA ont été acceptées avec un avantage non-négligeable : il est possible de créer ses propres plugin afin de les utiliser avec SOFA. Il est ainsi possible de simuler énormément de problèmes : simulations de mécanique des solides (cerveau, coeur, foie notamment), dynamique des fluides (circulation sanguine par exemple) ainsi que des problèmes de thermodynamique. De plus, grâce notamment à différents plugin qui ont été développés, énormément de possibilités sont offertes par SOFA. Il existe par exemple un plugin permettant de simuler la découpe d'un maillage<sup>17</sup> ou bien des plugin permettant de lier les simulations SOFA avec des interfaces haptiques. De nombreux autres plugin ont ainsi été développés<sup>18</sup>, comme le plugin Caribou auquel nous allons nous intéresser par la suite.

Enfin, un point essentiel permettant de simplifier l'accès à SOFA est le plugin SofaPython3 (successeur de SofaPython pour le langage python2), qui offre la possibilité de créer des scènes en utilisant le langage python. C'est d'ailleurs cette méthode que nous utiliserons pour les quelques scènes qui seront présentées ultérieurement.

Présentons premièrement un exemple de scène SOFA, avec les résultats obtenus lors de la simulation. Pour ce premier exemple, nous allons considérer le cas d'une balle élastique chutant sur un support rigide.

14. <https://www.sofa-framework.org/>

15. <https://github.com/mimesis-inria/caribou>

16. [https://github.com/sofa-framework/sofapython3/](https://github.com/sofa-framework/sofapython3)

17. <https://www.sofa-framework.org/applications/marketplace/cutting-mesh-refinement/>

18. <https://www.sofa-framework.org/applications/marketplace/>

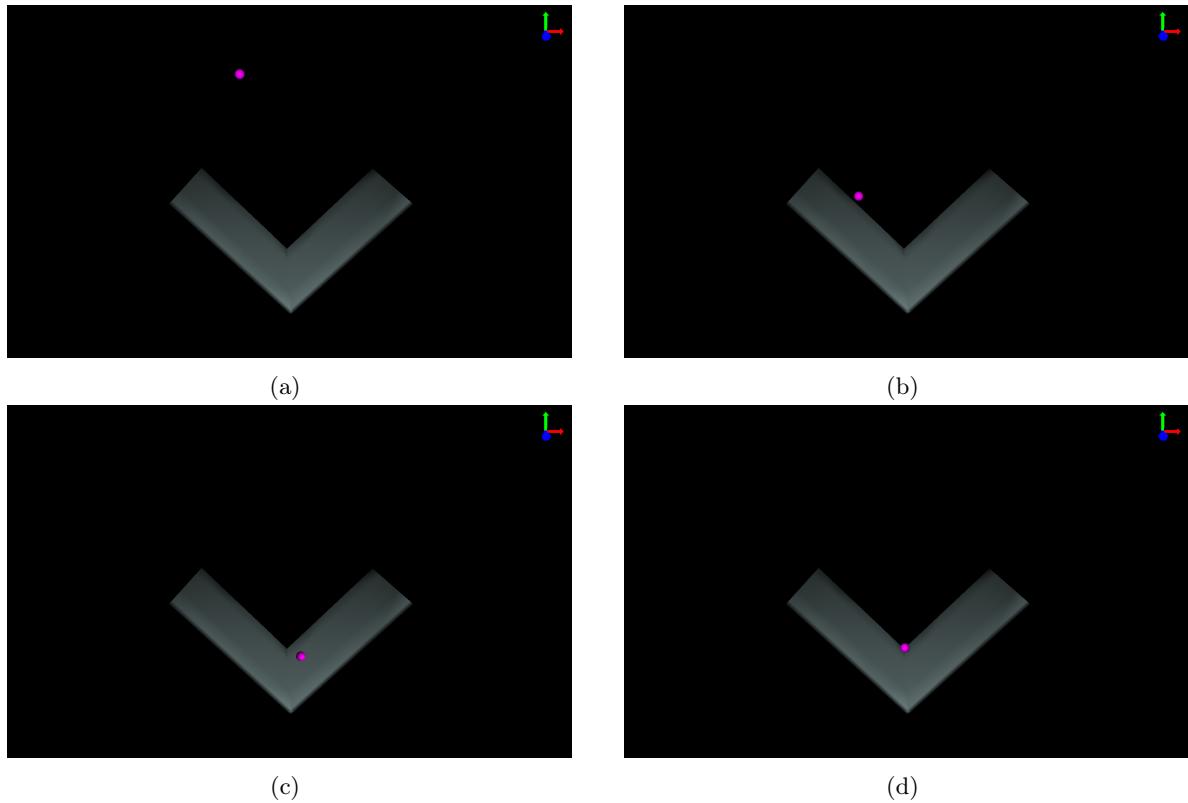


FIGURE 25 – Résultats obtenus pour la chute d'une balle élastique.

Nous représentons les résultats obtenus à différents pas de temps à la figure 25 (une vidéo de la simulation ainsi que le code utilisé sont disponibles à l'adresse [https://github.com/KVuillemot/Stage\\_Master\\_2](https://github.com/KVuillemot/Stage_Master_2)).

Maintenant que nous avons présenté brièvement SOFA, nous allons pouvoir également présenter le plugin qui nous intéressera par la suite. Ce plugin, Caribou (avec SofaCaribou) a été développé par Jean-Nicolas Brunet<sup>19</sup> au cours de sa thèse ([4]). Le projet Caribou est séparé en 2 parties :

- Caribou : partie indépendante de SOFA, permet de définir des géométries (triangles, carrés, tétraèdres ou bien hexaèdres) ainsi que des outils topologiques permettant notamment de créer des maillages ,
- SofaCaribou : utilise des éléments de Caribou ainsi que des composants de SOFA, apporte de nouveaux éléments : **grilles fictives**, **IsoSurfaces**, des « ForceFields », des solveurs linéaires et des solveurs d'EDO ainsi que différents matériaux.

Nous allons maintenant faire comme précédemment et illustrer certaines possibilités offertes par Caribou, en créant une scène utilisant à la fois SOFA et SofaCaribou. L'idée est la suivante sur la scène présentée : nous considérons un objet élastique qui chute. Cette fois, l'objet tombe sur une poutre également élastique, fixée à un mur. L'objet considéré sera cette fois-ci un foie et différentes étapes de la simulation sont représentées à la figure 26. Une vidéo de la simulation est également disponible à l'adresse [https://github.com/KVuillemot/Stage\\_Master\\_2](https://github.com/KVuillemot/Stage_Master_2).

## 5.2 Les débuts de l'implémentation

Maintenant que nous avons présenté rapidement les outils que nous allons utiliser pour ce projet, nous allons pouvoir parler de ce qui a été mis en place jusqu'à maintenant. En effet, ce projet étant à ses débuts, ce qui a été fait jusqu'à maintenant n'est qu'une petite partie réalisée à la suite de premières idées. Cependant, il est important de noter que tout cela est voué à évoluer et à être modifié.

19. <https://www.jnbrunet.com/>

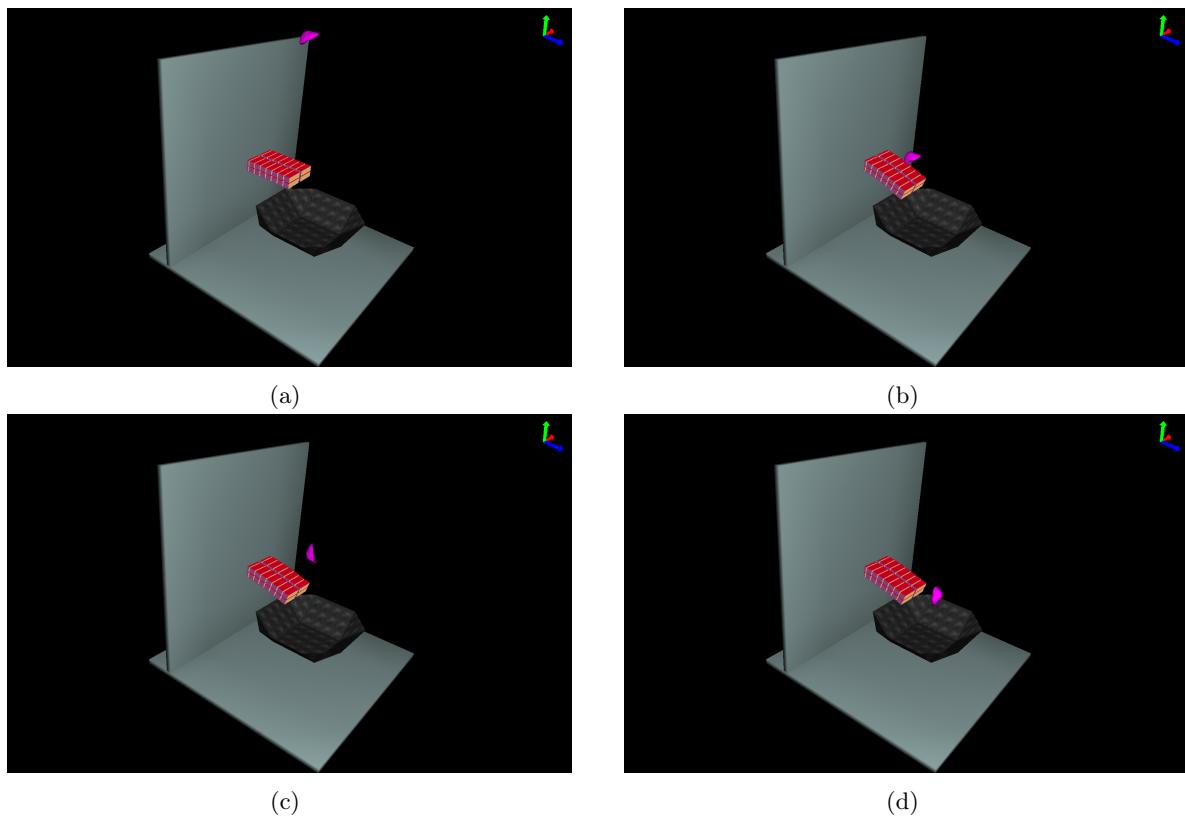
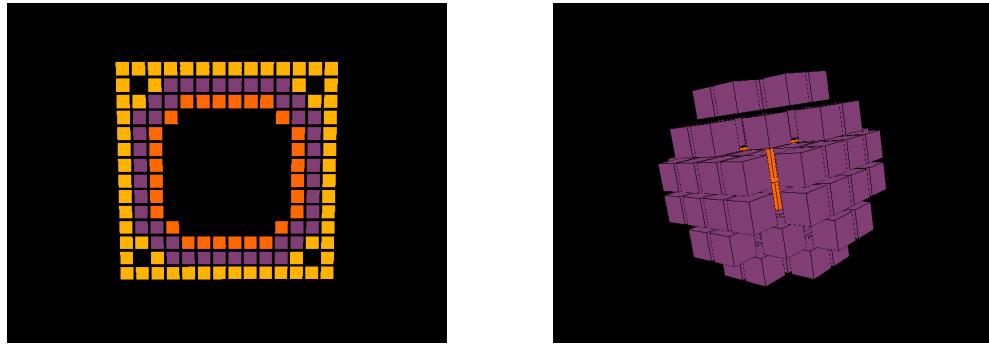


FIGURE 26 – Résultats obtenus pour la chute d'un foie sur une poutre élastique.



(a) Exemple en 2D.

(b) Exemple en 3D.

FIGURE 27 – Deux exemples de grilles fictives.

*Remarque 5.1.* Tous les codes qui seront présentés dans cette partie et en annexe C sont disponibles à l'adresse [https://github.com/KVuillemot/caribou/tree/Phi\\_FEM](https://github.com/KVuillemot/caribou/tree/Phi_FEM).

Comme nous l'avons dit précédemment, nous allons utiliser pour l'implémentation le plugin SofaCaribou. En effet, un point essentiel présent dans le plugin est l'implémentation déjà réalisée des « FictitiousGrid », que nous allons utiliser.

Cette fonctionnalité permet en effet de créer une grille fictive sur le même principe que le maillage utilisé pour le maillage  $T_h$  utilisé pour  $\phi$ -FEM. Cela nous permettra en effet de créer une grille quadrangulaire en 2D ou une grille hexaédrique en 3D, comme représentées à la figure 27. Pour la représentation 27(a), nous avons construit une grille pour le cercle de centre  $(0, 0)$ , de rayon 5 dans le carré  $[-6, 6] \times [-6, 6]$ . En orange, nous avons représenté les cellules internes de cette grille, dont l'une des faces est en intersection avec une cellule de la frontière. Les cellules violettes sont les cellules en intersection avec la frontière et les cellules jaunes sont une partie des cellules externes de la grille : soit elles ont une face en intersection avec une cellule violette, soit une face en intersection avec la frontière de la boîte considérée. Nous avons procédé de la même façon sur un exemple 3D, en considérant cette fois une sphère pour laquelle nous n'avons pas représenté les cellules externes, à la figure 27(b).

*Remarque 5.2 (Visualisation des maillages.).* Dans les différentes représentations, par exemple pour les figures 27(a) ou 28, il est important de noter que les cellules en couleur représentent uniquement le bord de chaque domaine. En particulier, c'est la raison pour laquelle nous avons des cellules noires à l'intérieur des cellules jaunes ou des cellules oranges. Ces cellules sont bien prises en compte dans la construction du maillage ou dans les calculs mais elles ne sont pas représentées.

L'idée pour utiliser notre méthode  $\phi$ -FEM est donc de considérer une grille fictive comme pour ces deux exemples.

**Indices des cellules.** Tout d'abord, pour utiliser ces grilles fictives, il va être nécessaire de connaître le type de chaque cellule, ce qui est déjà possible avec SofaCaribou. En effet, lors de la construction de la grille fictive, chaque élément est catégorisé en élément de bord, élément intérieur ou élément extérieur. Cependant, pour notre méthode, il est également nécessaire de récupérer tous les indices des éléments intérieurs et des éléments de bord. En effet, comme nous l'avons vu lors de la construction des différents schémas  $\phi$ -FEM, nous aurons besoin d'appliquer des termes de stabilisation à des endroits précis. Il est donc important de localiser les différentes cellules où nous devrons appliquer ces termes de stabilisation. Ainsi, pour déterminer les ensembles de cellules, nous avons dû créer la méthode « `boundary_cells_indices()` », dont le code est disponible en annexe C, 10. Ainsi, lors de la création d'une grille, cela permettra de connaître la liste des cellules de chaque type ainsi que d'afficher à l'utilisateur le nombre de cellules internes, externes ou à la frontière.

**Faces de chaque cellule.** Cependant, une fois ces éléments récupérés, nous nous sommes aperçus d'un autre point manquant dans Caribou. En effet, les termes de stabilisation des schémas  $\phi$ -FEM ne s'appliquent pas seulement sur les cellules : les différents termes de saut sont appliqués sur les facettes des éléments. Il

était donc nécessaire d'avoir plusieurs informations supplémentaires : en particulier, dans un premier temps, il était nécessaire de récupérer les indices des faces de chaque cellule. Il a donc fallu créer une méthode, permettant de déterminer pour chaque cellule, les faces de cette dernière. Pour cela, nous avons créé deux fonctions : une dans le cas 2D et une dans le cas 3D, toutes deux disponibles en annexe C, respectivement au programme 8 et 9. Il est important de noter que ces méthodes ne sont pour l'instant pas optimales. En effet, le temps de calcul est ici très important dès que la grille considérée contient beaucoup de cellules. Il sera donc par la suite essentiel de trouver une solution plus rapide, puisque pour l'instant l'algorithme est construit suivant l'algorithme 1, qui est une méthode où chaque face de la grille est parcourue. Ainsi, lorsque nous cherchons à déterminer les faces de chaque cellule, le nombre d'opérations devient vite très élevé.

```

Données :  $i$  : indice de la cellule considérée
Résultat : edges : liste avec les indices des faces de la cellule d'indice  $i$ 
Initialisation : edges = (0, 0, 0, 0, 0, 0),  $j = 0$ 
Pour chaque face de la grille Faire
    noeuds = noeuds de la face d'indice  $f$ 
    nombre de noeuds communs = 0
    Pour chaque noeud de la face Faire
        Pour chaque noeud de la cellule d'indice  $i$  Faire
            Si le noeud de la cellule et le noeud de la face correspondent Alors
                | nombre de noeuds communs = nombre de noeuds communs +1
            Fin
        Fin
    Fin
    Si le nombre de noeuds communs vaut 4 Alors
        | edges( $i$ ) =  $f$ 
        |  $i = i + 1$ 
    Fin
Fin
```

**Algorithme 1** : Récupération des indices des faces (cas 3D).

**Faces de  $\mathcal{F}_h^{\Gamma}$ .** Maintenant que nous avons la possibilité de déterminer les faces appartenant à une cellule, nous allons pouvoir déterminer les faces appartenant à l'ensemble  $\mathcal{F}_h^{\Gamma}$ . Rappelons que cet espace contient toutes les facettes internes du maillage du bord, c'est-à-dire par exemple pour le cas de la figure 27(a), toutes les faces des cellules violettes, sauf celles communes à un élément violet et à un élément jaune. Pour cela, nous avons construit la méthode « `get_boundary_faces()` » selon l'algorithme 2.

Il était alors important de tester les différentes fonctions implémentées afin de valider ou non le fonctionnement de ces dernières. Pour cela, nous avons considéré la grille représentée à la figure 28, qui permettait de compter facilement le nombre de cellules et de faces.

Programme 1 – Sorties obtenues pour le cas représenté à la figure 28.

[INFO]	[ FictitiousGrid(grid) ] There are 16 cells on the boundary
[INFO]	[ FictitiousGrid(grid) ] There are 9 cells inside
[INFO]	[ FictitiousGrid(grid) ] There are 56 cells outside
[INFO]	[ FictitiousGrid(grid) ] Number of edges on the grid 180
[INFO]	[ FictitiousGrid(grid) ] Number of edges on the boundary cells 43

**Analyse des résultats.** Malheureusement, à la lecture de ces résultats, il semble qu'il y ait des problèmes. Notons dans un premier temps que le nombre de cellules internes, externes et sur la frontière correspond bien à ce qui est observé. Cependant, le nombre de faces sur la frontière ne semble pas correspondre à ce qui est attendu. Pour vérifier cela, nous allons utiliser les différents *bindings* créés (liens entre le code C++ et le Python pour récupérer différentes informations sur la grille créée). Le code utilisé ainsi que les sorties obtenues sont disponibles aux programmes 2 et 3.

**Résultat :** faces : liste avec les indices des faces de  $\mathcal{F}_h^\Gamma$

**Initialisation :**

frontière = liste des indices des cellules de la frontière

extérieur = liste des indices des cellules externes

faces\_boundary = liste vide

**Pour** chaque cellule de la frontière **Faire**

    récupérer les faces de la cellule

    ajouter les indices de ces faces à la liste faces\_boundary

**Fin**

**/\* La liste faces\_boundary contient maintenant toutes les faces de la frontière, y compris celles communes à la partie extérieure. \*/**

**Pour** chaque cellule de la frontière **Faire**

    récupérer les cellules voisines

**Pour** chaque cellule voisine **Faire**

**Si** cette cellule est une cellule extérieure **Alors**

            faces\_voisin = récupérer les indices des faces de cette cellule

            supprimer de faces\_boundary les éléments communs à faces\_voisins

**Fin**

**Fin**

**Fin**

**Algorithme 2 :** Construction de  $\mathcal{F}_h^\Gamma$ .

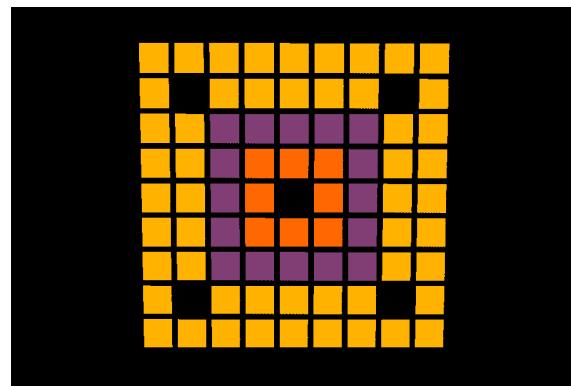


FIGURE 28 – Cas considéré pour la validation des méthodes.

Programme 2 – Bindings utilisés.

```
print(f'{root.grid.number_of_cells()=}')
print(f'{root.grid.number_of_nodes()=}')
print(f'{root.grid.boundary_cells_indices()=}')
print(f'{root.grid.face_cell_indices(3)=}')
print(f'{root.grid.boundary_faces()=}')
print(f'{len(root.grid.boundary_faces())=}')
```

Programme 3 – Sorties obtenues.

```
root.grid.number_of_cells()=25
root.grid.number_of_nodes()=36
root.grid.boundary_cells_indices()=
    ([20, 21, 22, 23, 24, 29, 33, 38, 42, 47, 51, 56, 57, 58, 59, 60],
     [30, 31, 32, 39, 40, 41, 48, 49, 50],
     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
      18, 19, 25, 26, 27, 28, 34, 35, 36, 37, 43, 44, 45, 46, 52, 53,
      54, 55, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
      73, 74, 75, 76, 77, 78, 79, 80])
root.grid.face_cell_indices(3)=[3, 12, 13, 22]
root.grid.boundary_faces()=[]
len(root.grid.boundary_faces())=0
```

Malheureusement, ces sorties ont pu confirmer qu'une erreur était présente : il semble n'y avoir aucune face sélectionnée à la frontière. Cependant il restait à trouver d'où venait l'erreur. Pour cela, nous avons une nouvelle fois utilisé les différents bindings python pour obtenir différentes informations sur la grille. Grâce à cela, nous avons pu appliquer l'algorithme 2 directement en python. En appliquant cet algorithme, nous avons pu obtenir le résultat souhaité : il y a 40 faces sur la maillage de la frontière, dont 28 dans  $\mathcal{F}_h^\Gamma$ . L'erreur se trouve donc dans la méthode `get_boundary_faces()`.

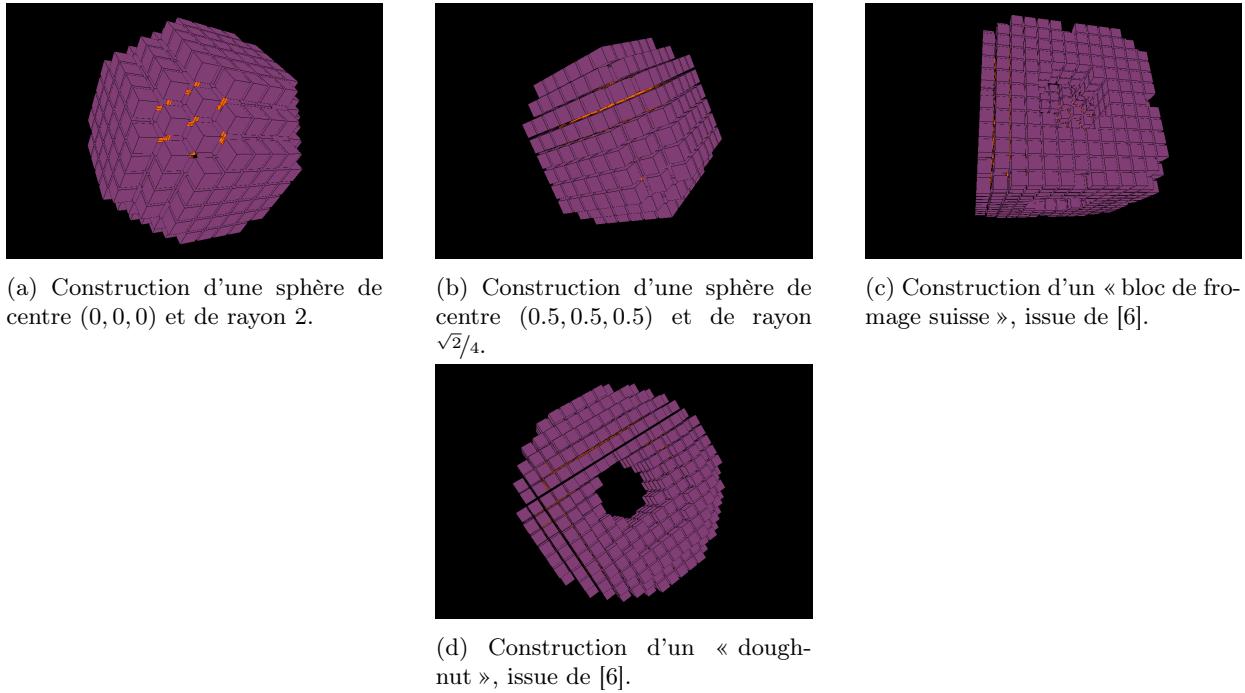
**Nouvelle version.** Une fois cette erreur localisée, nous avons pu modifier le code utilisé dans la méthode `get_boundary_faces`. Ce code est disponible en annexe C, 12. Nous avons alors testé une nouvelle fois la méthode sur le même cas que précédemment, en comparant les résultats avec la liste obtenue à l'aide du code python. Ces résultats sont représentés au programme 4, où nous retrouvons les mêmes résultats, correspondant au nombre de facettes attendu.

Programme 4 – Sorties obtenues.

```
faces_python = [50, 51, 52, 53, 59, 60, 61, 62, 63, 69,
                72, 78, 82, 88, 91, 97, 101, 107, 110,
                116, 117, 118, 119, 120, 126, 127, 128, 129]
len(faces_python) = 28
faces_sofa = [50, 51, 52, 53, 59, 60, 61, 62, 63, 69,
              72, 78, 82, 88, 91, 97, 101, 107, 110,
              116, 117, 118, 119, 120, 126, 127, 128, 129]
len(faces_sofa) = 28
```

**La fonction level-set.** Comme nous l'avons vu précédemment lors de la présentation de la méthode  $\phi$ -FEM, il est nécessaire de construire une fonction level-set. En particulier, il est nécessaire de construire la grille fictive à partir de cette fonction. Cela est déjà possible dans différents cas particuliers avec Caribou,

- un cercle : `cicrcleIsoSurface`;
- une sphère : `sphereIsoSurface`;
- un cylindre : `cylinderIsoSurface`.

FIGURE 29 – Différentes constructions à l'aide de la classe **LevelSet**.

et permet d'obtenir par exemple les résultats représentés aux figures 27(a) et 27(b).

Cependant, il sera important par la suite de pouvoir construire d'autres domaines, à l'aide de la fonction `distance` signée, en suivant l'idée présentée dans 4.6.1. Dans un premier temps, nous avons choisi de créer une classe **LevelSet** permettant de définir quelques domaines supplémentaires, dont les résultats sont représentés à la figure 29

Cependant, pour  $\phi$ -FEM nous n'avons pas besoin que de construire le maillage avec la fonction level-set. En effet, elle est utilisée dans les calculs et il est donc nécessaire de déterminer sa fonction à chaque noeud du maillage. Pour cela, nous avons créé différentes fonctions permettant d'évaluer  $\phi$  en un noeud, puis sur une cellule puis sur tout le maillage. Ces différentes fonctions sont disponibles en annexe C, aux programmes 13, 14 et 15.

## 6 Conclusion

Nous avons au cours de ce rapport présenté la méthode  $\phi$ -FEM en développant de nouveaux schémas dans différents cadres : problèmes aux conditions mixtes, problèmes non-linéaires ou problèmes élastiques. La convergence numérique de ces schémas a été illustré dans différents cas, nous montrant l'efficacité de cette technique. Cependant, pour l'instant, nous n'avons pas eu le temps d'aborder l'aspect plus théorique de ces différents schémas. L'étude théorique de tels schémas pouvant se révéler relativement complexe, elle est souvent assez longue.

Ainsi, il sera important par la suite de réaliser une telle étude, notamment afin de s'assurer de la convergence théorique de ces schémas mais aussi par exemple du conditionnement des matrices éléments finis considérées. Cependant, avant de travailler sur l'aspect théorique des schémas introduits dans ce rapport, il sera également important de considérer le schéma introduit pour des conditions mixtes, avec des conditions de Dirichlet pures. En effet, si la théorie à déjà été traitée dans [12] pour le schéma introduit pour des conditions de Dirichlet, le cas de ce nouveau schéma n'a pas encore été traité théoriquement.

De plus, il reste encore beaucoup de travail à réaliser sur ces différents schémas, au niveau numérique. Il sera notamment intéressant de développer d'autres cas tests ou encore comme cela est prévu de lier la méthode  $\phi$ -FEM et les réseaux de neurones. Cela permettra notamment d'améliorer le temps de calcul de la méthode qui pour l'instant reste élevé lorsque les domaines considérés sont complexes.

Enfin, il reste également beaucoup de travail à réaliser afin de lier Caribou et  $\phi$ -FEM. En effet, il sera nécessaire dans un premier de sélectionner correctement les faces de la frontière puis, ensuite il restera à développer la partie permettant de réaliser les différents calculs.

Cependant, malgré tout le travail restant à accomplir, nous avons déjà pu obtenir des premiers résultats aussi bien pour quelques méthodes implémentées au sein de Caribou que des résultats pour la résolution de problèmes complexes avec  $\phi$ -FEM, permettant d'illustrer numériquement l'efficacité de cette méthode.

## Références

- [1] Jérémie Allard, Stéphane Cotin, François Faure, Pierre-Jean Bensoussan, François Poyer, Christian Duriez, Hervé Delingette, and Laurent Grisoni. SOFA - an Open Source Framework for Medical Simulation. In *MMVR 15 - Medicine Meets Virtual Reality*, volume 125 of *Studies in Health Technology and Informatics*, pages 13–18, Palm Beach, United States, February 2007. IOP Press.
- [2] Nabil M. Atallah, Claudio Canuto, and Guglielmo Scovazzi. Analysis of the shifted boundary method for the poisson problem in general domains. *ArXiv*, abs/2006.00872, 2021.
- [3] Nabil M. Atallah, Claudio Canuto, and Guglielmo Scovazzi. The shifted boundary method for solid mechanics. *International Journal for Numerical Methods in Engineering*, 122(20) :5935–5970, 2021.
- [4] Jean-Nicolas Brunet. *Exploring new numerical methods for the simulation of soft tissue deformations in surgery assistance*. Theses, Université de Strasbourg, November 2020.
- [5] E. Burman. Ghost penalty. *C. R. Math. Acad. Sci. Paris*, 348(21-22) :1217–1220, 2010.
- [6] Erik Burman, Susanne Claus, Peter Hansbo, Mats G. Larson, and André Massing. Cutfem : Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7) :472–501, 2015.
- [7] Stéphane Cotin, Michel Duprez, Vanessa Lleras, Alexei Lozinski, and Killian Vuillemot.  $\phi$ -FEM : an efficient simulation tool using simple meshes for problems in structure mechanics and heat transfer. In *Partition of Unity Methods (Wiley Series in Computational Mechanics) 1st Edition*. Wiley, November 2022.
- [8] Michel Duprez, Vanessa Lleras, and Alexei Lozinski. A new  $\phi$ -FEM approach for problems with natural boundary conditions. *Numerical Methods for Partial Differential Equations*, 2022.
- [9] Michel Duprez, Vanessa Lleras, and Alexei Lozinski.  $\phi$ -FEM : an optimally convergent and easily implementable immersed boundary method for particulate flows and Stokes equations. working paper or preprint, February 2022.
- [10] Michel Duprez, Vanessa Lleras, and Alexei Lozinski.  $\phi$ -FEM : A fictitious domain method for finite element methods on domains defined by level-sets. 23/03/2021.
- [11] Michel Duprez, Vanessa Lleras, Alexei Lozinski, and Killian Vuillemot. An Immersed Boundary Method by Phi-FEM approach to solve the heat equation. working paper or preprint, June 2022.
- [12] Michel Duprez and Alexei Lozinski.  $\phi$ -FEM : a finite element method on domains defined by level-sets. *SIAM J. Numer. Anal.*, 58(2) :1008–1028, 2020.
- [13] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, and Stéphane Cotin. SOFA : A Multi-Model Framework for Interactive Physical Simulation. In Yohan Payan, editor, *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, volume 11 of *Studies in Mechanobiology, Tissue Engineering and Biomaterials*, pages 283–321. Springer, June 2012.
- [14] André Fortin and André Garon. Les éléments finis : de la théorie à la pratique. [https://giref.ulaval.ca/afortin/elements\\_finis.pdf](https://giref.ulaval.ca/afortin/elements_finis.pdf), 1999.
- [15] R. Glowinski, T. Pan, and J. Periaux. A fictitious domain method for Dirichlet problem and applications. *Computer Methods in Applied Mechanics and Engineering*, 111(3-4) :283–303, 1994.
- [16] Andrea Mendizabal, Pablo Márquez-Neila, and Stéphane Cotin. Simulation of hyperelastic materials in real-time using deep learning. *Medical Image Analysis*, 59 :101569, 2019.
- [17] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37 :239–261, 2005.
- [18] Nicolas Moës and Ted Belytschko. X-FEM, de nouvelles frontières pour les éléments finis. In *5e Colloque national en calcul des structures*, volume 11, pages 305 – 318, Giens, France, May 2001. CSMA.
- [19] Killian Vuillemot. Développement de méthodes éléments finis non-conformes adaptées à la chirurgie assistée par ordinateur. [https://github.com/KVuillemot/Stage\\_M1\\_Phi\\_FEM/blob/main/Rapport\\_Stage\\_M1.pdf](https://github.com/KVuillemot/Stage_M1_Phi_FEM/blob/main/Rapport_Stage_M1.pdf), 2021.

## A Quelques résultats pour le problème de Poisson

Il semblait intéressant de présenter quelques résultats numériques pour les problèmes de Poisson Dirichlet et de Poisson Neumann. Pour la cas Dirichlet, nous avons résolu numériquement le problème présenté en 3.1.3, page 21, considérant  $\Gamma_N = \emptyset$  et  $\Gamma_D = \Gamma$ . Ces résultats sont représentés à la figure 30

Pour la problème de Neumann, nous avons considéré une version légèrement modifiée de ce problème, où nous avons cette fois cherché la solution de

$$-\Delta u + u = f,$$

avec les conditions de bord introduites dans 3.1.3, en considérant  $\Gamma_D = \emptyset$  et  $\Gamma_N = \Gamma$ . Les résultats sont représentés à la figure 31

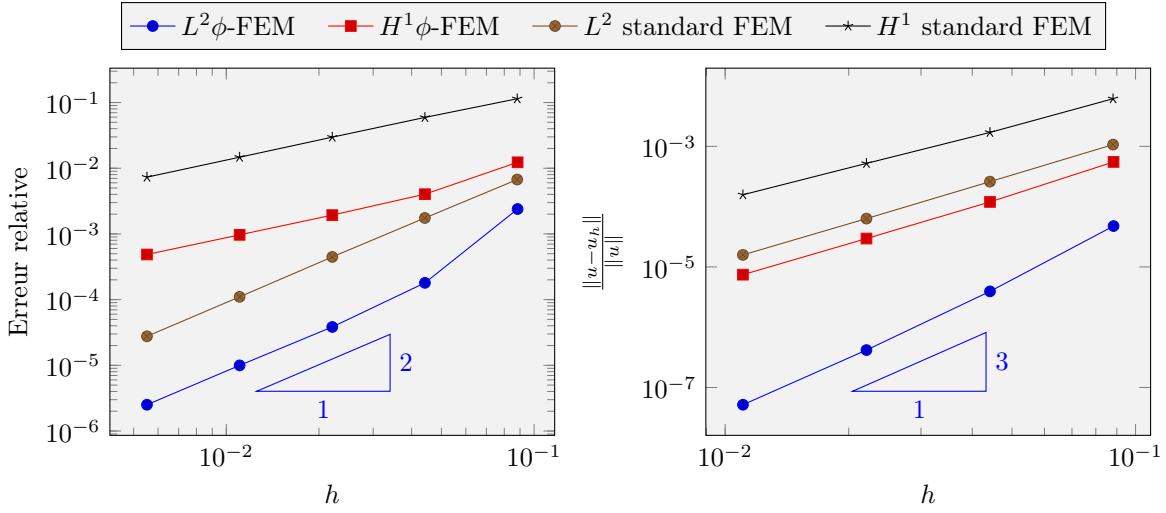


FIGURE 30 – Conditions de Dirichlet pures, à gauche éléments finis  $\mathbb{P}^1$  et à droite  $\mathbb{P}^2$ .

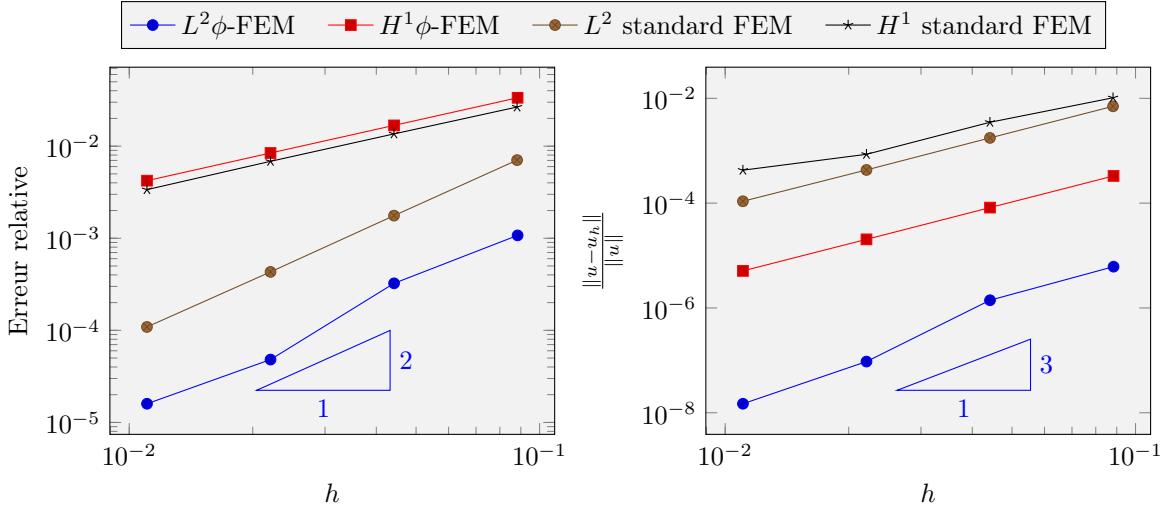


FIGURE 31 – Conditions de Neumann pures, à gauche éléments finis  $\mathbb{P}^1$  et à droite  $\mathbb{P}^2$ .

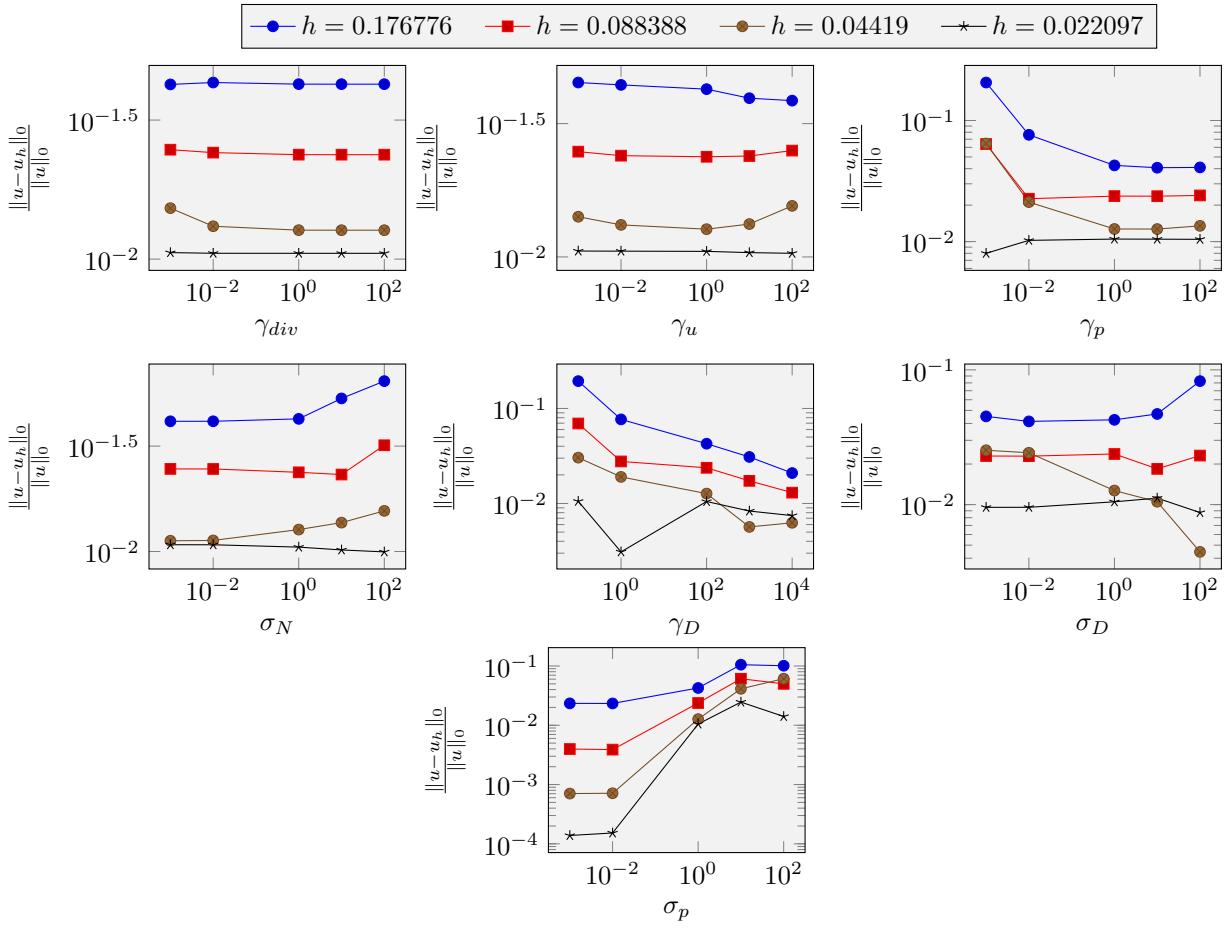


FIGURE 32 – Conditions mixtes, avec des éléments finis  $\mathbb{P}^1$  en faisant varier les paramètres : les paramètres sont tous fixés sauf 1. De gauche à droite :  $\gamma_{div}$ ,  $\gamma_u$ ,  $\gamma_p$ ,  $\sigma_N$ ,  $\gamma_D$ ,  $\sigma_D$  et  $\sigma_p$ .

## B Codes *FEniCSX*

Programme 5 – Modification de la fonction `locate_entities`.

```
std::vector<std::int32_t> mesh::locate_entities_new(
    const Mesh& mesh, int dim,
    const std::function<xt::xtensor<bool, 1>
        (const xt::xtensor<double, 2>&) & marker)
{
    const Topology& topology = mesh.topology();
    const int tdim = topology.dim();

    // Create entities and connectivities
    mesh.topologyMutable().create_entities(dim);
    mesh.topologyMutable().create_connectivity(tdim, 0);
    if (dim < tdim)
        mesh.topologyMutable().create_connectivity(dim, 0);

    // Get all vertex 'node' indices
    const graph::AdjacencyList<std::int32_t>& x_dofmap =
        mesh.geometry().dofmap();
    const std::int32_t num_vertices = topology.index_map(0)->size_local()
        + topology.index_map(0)->num_ghosts();
    auto c_to_v = topology.connectivity(tdim, 0);
    assert(c_to_v);
    std::vector<std::int32_t> vertex_to_node(num_vertices);
    for (int c = 0; c < c_to_v->num_nodes(); ++c)
    {
        auto x_dofs = x_dofmap.links(c);
        auto vertices = c_to_v->links(c);
        for (std::size_t i = 0; i < vertices.size(); ++i)
            vertex_to_node[vertices[i]] = x_dofs[i];
    }

    // Pack coordinates of vertices
    xtl::span<const double> x_nodes = mesh.geometry().x();
    xt::xtensor<double, 2> x_vertices({3, vertex_to_node.size()});
    for (std::size_t i = 0; i < vertex_to_node.size(); ++i)
    {
        const int pos = 3 * vertex_to_node[i];
        for (std::size_t j = 0; j < 3; ++j)
            x_vertices(j, i) = x_nodes[pos + j];
    }

    // Run marker function on vertex coordinates
    const xt::xtensor<bool, 1> marked = marker(x_vertices);
    if (marked.shape(0) != x_vertices.shape(1))
        throw std::runtime_error("Length of array of markers is wrong.");

    // Iterate over entities to build vector of marked entities
    auto e_to_v = topology.connectivity(dim, 0);
    assert(e_to_v);
    std::vector<std::int32_t> entities;
    for (int e = 0; e < e_to_v->num_nodes(); ++e)
```

```

{
    // Iterate over entity vertices
    bool all_vertices_marked = true;
    for (std::int32_t v : e_to_v->links(e))
    {
        if(marked[v]) {
            all_vertices_marked = true;
            break;
        } else {
            all_vertices_marked = false;
        }
    }
    if (all_vertices_marked)
        entities.push_back(e);
}
return entities;
}

```

Programme 6 – Création de la fonction `locate_bc_entities`.

```

std::vector<std::int32_t> mesh::locate_bc_entities(
    const Mesh& mesh, int dim, bool is_facets, double epsilon,
    const std::function<xt::xtensor<double, 1>
    (const xt::xtensor<double, 2>&)>& marker_phi_boundary,
    // Checks if we are in the level set function boundary
    const std::function<xt::xtensor<bool, 1>
    (const xt::xtensor<double, 2>&)>& marker_boundary_type)
    // Checks if it's Neuman or Dirichlet

{
    const Topology& topology = mesh.topology();
    const int tdim = topology.dim();

    // Create entities and connectivities
    mesh.topology Mutable().create_entities(dim);
    mesh.topology Mutable().create_connectivity(tdim, 0);
    if (dim < tdim)
        mesh.topology Mutable().create_connectivity(dim, 0);

    // Get all vertex 'node' indices
    const graph::AdjacencyList<std::int32_t>& x_dofmap =
        mesh.geometry().dofmap();
    const std::int32_t num_vertices = topology.index_map(0)->size_local()
                                    + topology.index_map(0)->num_ghosts();
    auto c_to_v = topology.connectivity(tdim, 0);
    assert(c_to_v);
    std::vector<std::int32_t> vertex_to_node(num_vertices);
    for (int c = 0; c < c_to_v->num_nodes(); ++c)
    {
        auto x_dofs = x_dofmap.links(c);
        auto vertices = c_to_v->links(c);
        for (std::size_t i = 0; i < vertices.size(); ++i)
            vertex_to_node[vertices[i]] = x_dofs[i];
    }
}

```

```

// Pack coordinates of vertices
xtl::span<const double> x_nodes = mesh.geometry().x();
xt::xtensor<double, 2> x_vertices({3, vertex_to_node.size()});
for (std::size_t i = 0; i < vertex_to_node.size(); ++i)
{
    const int pos = 3 * vertex_to_node[i];
    for (std::size_t j = 0; j < 3; ++j)
        x_vertices(j, i) = x_nodes[pos + j];
}

// Run marker function on vertex coordinates
const xt::xtensor<double, 1> marked_phi =
    marker_phi_boundary(x_vertices);
const xt::xtensor<bool, 1> marked_bc_type =
    marker_boundary_type(x_vertices);
if (marked_phi.shape(0) != x_vertices.shape(1)
    && marked_bc_type.shape(0) != x_vertices.shape(1) )
    throw std::runtime_error("Length of array of markers is wrong.");

// Iterate over entities to build vector of marked entities
auto e_to_v = topology.connectivity(dim, 0);
assert(e_to_v);
std::vector<std::int32_t> entities;
bool all_vertices_marked = false;
for (int e = 0; e < e_to_v->num_nodes(); ++e)
{
    std::int32_t v1 = e_to_v->links(e)[0];
    std::int32_t v2 = e_to_v->links(e)[1];

    bool marked_v1_type = marked_bc_type[v1];
    bool marked_v2_type = marked_bc_type[v2];

    double marked_v1_phi = marked_phi[v1];
    double marked_v2_phi = marked_phi[v2];

    if(!is_facets) {
        std::int32_t v3 = e_to_v->links(e)[2];
        double marked_v3_phi = marked_phi[v3];
        bool marked_v3_type = marked_bc_type[v3];
        if( marked_v1_phi*marked_v2_phi <= epsilon ||
            marked_v1_phi*marked_v3_phi <= epsilon ||
            marked_v2_phi*marked_v3_phi <= epsilon) {
            if( marked_v1_type && marked_v2_type && marked_v3_type) {
                all_vertices_marked = true;
            }
        }
    }
} else {
    if(marked_v1_phi*marked_v2_phi <= epsilon) {
        if( marked_v1_type && marked_v2_type ) {
            all_vertices_marked = true;
        }
    }
}
}
}

```

```

    // Iterate over entity vertices
    if (all_vertices_marked)
        entities.push_back(e);
    all_vertices_marked = false;
}
return entities;
}

```

Programme 7 – Création de la fonction `locate_facets`.

```

std::vector<std::int32_t> mesh::locate_facets(
    const Mesh& mesh, int dim, bool is_facets, double epsilon,
    const std::function<xt::xtensor<double, 1>
        (const xt::xtensor<double, 2>&)>& marker_phi_boundary,
    // Checks if we are in the level set function boundary
    const std::function<xt::xtensor<bool, 1>
        (const xt::xtensor<double, 2>&)>& marker_boundary_type )
    // Checks if it's Neuman or Dirichlet
{
    const Topology& topology = mesh.topology();
    const int tdim = topology.dim();

    // Create entities and connectivities
    mesh.topologyMutable().create_entities(dim);
    if (dim < tdim)
        mesh.topologyMutable().create_connectivity(tdim, 0);

    mesh.topologyMutable().create_connectivity(dim, 0);
    mesh.topologyMutable().create_connectivity(dim, dim-1);
    mesh.topologyMutable().create_connectivity(dim-1, 0);

    // Get all vertex 'node' indices
    const graph::AdjacencyList<std::int32_t>& x_dofmap =
        mesh.geometry().dofmap();
    const std::int32_t num_vertices = topology.index_map(0)->size_local()
        + topology.index_map(0)->num_ghosts();
    auto c_to_v = topology.connectivity(tdim, 0);
    assert(c_to_v);
    std::vector<std::int32_t> vertex_to_node(num_vertices);
    for (int c = 0; c < c_to_v->num_nodes(); ++c)
    {
        auto x_dofs = x_dofmap.links(c);
        auto vertices = c_to_v->links(c);
        for (std::size_t i = 0; i < vertices.size(); ++i)
            vertex_to_node[vertices[i]] = x_dofs[i];
    }

    // Pack coordinates of vertices
    xtl::span<const double> x_nodes = mesh.geometry().x();
    xt::xtensor<double, 2> x_vertices({3, vertex_to_node.size()});
    for (std::size_t i = 0; i < vertex_to_node.size(); ++i)
    {
        const int pos = 3 * vertex_to_node[i];
        for (std::size_t j = 0; j < 3; ++j)
            x_vertices(j, i) = x_nodes[pos + j];
    }
}

```

```

}

// Run marker function on vertex coordinates
const xt::xtensor<double, 1> marked_phi =
    marker_phi_boundary(x_vertices);
const xt::xtensor<bool, 1> marked_bc_type =
    marker_boundary_type(x_vertices);
if ( marked_phi.shape(0) != x_vertices.shape(1)
&& marked_bc_type.shape(0) != x_vertices.shape(1) )
    throw std::runtime_error("Length of array of markers is wrong.");

// Iterate over entities to build vector of marked entities
auto e_to_v = topology.connectivity(dim, 0);
auto e_to_f = topology.connectivity(dim, dim-1);
auto f_to_v = topology.connectivity(1, 0);
assert(e_to_v);
assert(e_to_f);
assert(f_to_v);

std::vector<std::int32_t> entities;
bool all_vertices_marked = false;
for(int e = 0; e < e_to_v->num_nodes(); e++) {
    std::int32_t v1 = e_to_v->links(e)[0];
    std::int32_t v2 = e_to_v->links(e)[1];
    std::int32_t v3 = e_to_v->links(e)[2];

    bool marked_v1_type = marked_bc_type[v1];
    bool marked_v2_type = marked_bc_type[v2];
    bool marked_v3_type = marked_bc_type[v3];

    double marked_v1_phi = marked_phi[v1];
    double marked_v2_phi = marked_phi[v2];
    double marked_v3_phi = marked_phi[v3];

    if( marked_v1_phi*marked_v2_phi <= epsilon ||
        marked_v1_phi*marked_v3_phi <= epsilon ||
        marked_v2_phi*marked_v3_phi <= epsilon) {
        if( marked_v1_type && marked_v2_type && marked_v3_type) {
            for(int ii=0; ii < 3; ii++) {
                auto facet = e_to_f->links(e)[ii];
                bool good_facet = false;
                for(int f=0; f < f_to_v->num_nodes(); f++) {
                    auto curr_facet = f_to_v->links(e)[f];
                    if(curr_facet == facet && !good_facet ) {
                        good_facet = true;
                        entities.push_back(f);
                    }
                }
            }
        }
    }
}

auto it = unique(entities.begin(), entities.end());

```

```

entities.resize(distance(entities.begin(), it));

return entities;
}

```

## C Codes modifiés dans Caribou

Programme 8 – Fonction retournant tous les cotés d'une cellule en deux dimensions.

```

/** Return all the edges of a cell */
[[nodiscard]] inline auto
contained_edges(const CellIndex & index) const
-> std::vector<UNSIGNED_INTEGER_TYPE>{

    std::vector<UNSIGNED_INTEGER_TYPE> edges(4, 0); // output vector

    // first : get nodes of the cell
    // then use the nodes to determine all the faces
    // that have two nodes of the list

    const auto & cell_nodes = node_indices_of(index);
    // get the nodes of the cell

    int node_index_in_cell = 0;

    for (UNSIGNED_INTEGER_TYPE edge_index = 0;
        edge_index < number_of_edges(); ++edge_index ){

        // for each edge, we check if all the nodes are in cell_nodes
        const auto & edge_nodes = edge(edge_index);
        int nbr_common_nodes = 0;
        for(UNSIGNED_INTEGER_TYPE node_edge_index = 0;
            node_edge_index < 2; ++node_edge_index){

            for(UNSIGNED_INTEGER_TYPE node_cell_index=0;
                node_cell_index < 4; ++node_cell_index){
                // check the 4 nodes of the cell
                if (edge_nodes[node_edge_index] ==
                    cell_nodes[node_cell_index]){
                    ++nbr_common_nodes;
                }
            }
        }
        if (nbr_common_nodes == 2){
            // if there are two common nodes, then the edge is in the cell
            edges[node_index_in_cell] = edge_index;
            ++node_index_in_cell;
        }
    }
    return edges;
}

```

Programme 9 – Fonction retournant toutes les faces d'une cellule en trois dimensions.

```
/** Return all the faces that are in a cell */
[[nodiscard]] inline auto
contained_faces(const CellIndex & index)
    const -> std::vector<UNSIGNED_INTEGER_TYPE>{

    std::vector<UNSIGNED_INTEGER_TYPE> faces(6, 0); // output vector
    // first : get nodes of the cell
    // then use the nodes to determine all the faces
    // that have two nodes of the list

    const auto & cell_nodes = node_indices_of(index);
    // get the 8 nodes of the cell
    int i = 0;

    for (UNSIGNED_INTEGER_TYPE face_index = 0;
        face_index < number_of_faces(); ++face_index ){
        // check all the faces of the grid

        const auto & face_nodes = face(face_index);
        int nbr_common_nodes = 0;
        for(UNSIGNED_INTEGER_TYPE node_face_index = 0;
            node_face_index < 4; ++node_face_index){
            // check all the nodes of the face

            for(UNSIGNED_INTEGER_TYPE node_cell_index=0;
                node_cell_index < 8; ++node_cell_index){
                // check all the nodes of the cell

                if (face_nodes[node_face_index] ==
                    cell_nodes[node_cell_index]){
                    ++nbr_common_nodes;
                }
            }
        }
        if (nbr_common_nodes == 4){
            // if there are 4 common nodes, then the face is on the cell
            faces[i] = face_index;
            ++i;
        }
    }
    return faces;
}

};
```

Programme 10 – Fonction retournant les indices des cellules en fonction du type.

```
template <typename DataTypes>
auto FictitiousGrid<DataTypes>::boundary_cells_indices()
    const -> std::tuple< std::vector<UNSIGNED_INTEGER_TYPE>,
        std::vector<UNSIGNED_INTEGER_TYPE>,
        std::vector<UNSIGNED_INTEGER_TYPE>> {
```

```

const auto number_of_cells = p_grid->number_of_cells();
UNSIGNED_INTEGER_TYPE number_of_boundary_cells = 0,
                      number_of_inside_cells = 0,
                      number_of_outside_cells = 0;

std::vector<UNSIGNED_INTEGER_TYPE> inside_cells_indices;
std::vector<UNSIGNED_INTEGER_TYPE> boundary_cells_indices;
std::vector<UNSIGNED_INTEGER_TYPE> outside_cells_indices;

for (UNSIGNED_INTEGER_TYPE index = 0;
      index < number_of_cells; ++index) {
    if(p_cells_types[index] == Type::Inside) {
        inside_cells_indices.push_back(index);
    }
    else if(p_cells_types[index] == Type::Boundary) {
        boundary_cells_indices.push_back(index);
    }
    else {
        outside_cells_indices.push_back(index);
    }
}
number_of_boundary_cells = boundary_cells_indices.size();
number_of_inside_cells = inside_cells_indices.size();
number_of_outside_cells = outside_cells_indices.size();

msg_info() << "There are " << number_of_boundary_cells
            << " cells on the boundary";
msg_info() << "There are " << number_of_inside_cells
            << " cells inside";
msg_info() << "There are " << number_of_outside_cells
            << " cells outside";

return {boundary_cells_indices, inside_cells_indices,
        outside_cells_indices};
}

```

Programme 11 – Fonction retournant les indices des faces d'une cellule.

```

// to get the indices of all the faces of one cell
template <typename DataTypes>
auto FictitiousGrid<DataTypes>::get_faces_on_cell(
    const CellIndex & cellIndex)
    const -> std::vector<UNSIGNED_INTEGER_TYPE>{

    std::vector<UNSIGNED_INTEGER_TYPE> face_on_cell;

    if constexpr (Dimension == 2){
        face_on_cell = p_grid->contained_edges(cellIndex);
    }

    if constexpr (Dimension == 3){
        face_on_cell = p_grid->contained_faces(cellIndex);
    }
}

```

```

    return face_on_cell;
}

```

Programme 12 – Fonction permettant de construire  $\mathcal{F}_h^{\Gamma}$ .

```

// to get the faces on which ones we apply the ghost penalty
template <typename DataTypes>
auto FictitiousGrid<DataTypes>::get_boundary_faces()
    const -> std::vector<UNSIGNED_INTEGER_TYPE>{

    std::vector<UNSIGNED_INTEGER_TYPE> boundary_faces;

    const auto cells = boundary_cells_indices();
    const auto boundary = std::get<0>(cells);
    const auto inside = std::get<1>(cells);
    const auto outside = std::get<2>(cells);

    if constexpr (Dimension == 3 || Dimension == 2) {
        for(UNSIGNED_INTEGER_TYPE index = 0;
            index < boundary.size(); index++){
            // build a vector with all the faces that are on boundary_cells
            const auto faces = get_faces_on_cell(boundary[index]);
            boundary_faces.insert(boundary_faces.end(),
                                  faces.begin(),
                                  faces.end());
    }

    // delete the duplicate indices
    std::sort(boundary_faces.begin(), boundary_faces.end());
    auto last = std::unique(boundary_faces.begin(),
                           boundary_faces.end());
    boundary_faces.erase(last, boundary_faces.end());

    msg_info() << "Number of faces on the boundary cells "
                << boundary_faces.size();

    std::vector<UNSIGNED_INTEGER_TYPE> neighbors_indices;
    std::vector<UNSIGNED_INTEGER_TYPE> outside_faces;

    for (UNSIGNED_INTEGER_TYPE index = 0;
         index < boundary.size(); index++){
        const auto neighbors =
            get_neighbors(&p_cells[boundary[index]]);

        for (UNSIGNED_INTEGER_TYPE index_neighbor = 0;
             index_neighbor < neighbors.size(); index_neighbor++){
            neighbors_indices.push_back(
                neighbors[index_neighbor]->index);
        }
    }

    std::sort(neighbors_indices.begin(), neighbors_indices.end());
    auto last_neighbors = std::unique(neighbors_indices.begin(),
                                     neighbors_indices.end());
    neighbors_indices.erase(last_neighbors,
                           neighbors_indices.end());
}

```

```

// get indices of the neighbors that are outside
for (UNSIGNED_INTEGER_TYPE index = 0;
     index < neighbors_indices.size(); index++){

    if(p_cells_types[neighbors_indices[index]]
       == Type::Outside){
        const auto faces =
            get_faces_on_cell(neighbors_indices[index]);
        outside_faces.insert(outside_faces.end(),
                             faces.begin(),
                             faces.end());
    }
}

std::sort(outside_faces.begin(), outside_faces.end());
auto last_outside_faces = std::unique(outside_faces.begin(),
                                       outside_faces.end());
outside_faces.erase(last_outside_faces, outside_faces.end());

std::vector<int> v(outside_faces.size()
                  + boundary_faces.size());
std::vector<int>::iterator it, st;

it = std::set_intersection(boundary_faces.begin(),
                           boundary_faces.end(),
                           outside_faces.begin(),
                           outside_faces.end(),
                           v.begin());
for (auto st = v.begin(); st != it; ++st){
    boundary_faces.erase(std::remove(boundary_faces.begin(),
                                    boundary_faces.end(),
                                    *st),
                         boundary_faces.end());
}
neighbors_indices.clear();
outside_faces.clear();
}

return boundary_faces;
}

```

Programme 13 – Fonction permettant d'évaluer la level-set en un noeud de la grille.

```

template <typename DataTypes>
auto FictitiousGrid<DataTypes>::evaluate_level_set(
    const NodeIndex & nodeIndex)
const -> float {
    return (float) d_iso_surface->iso_value(p_grid->node(nodeIndex));
}

```

Programme 14 – Fonction permettant d'évaluer la level-set à chaque noeud d'une cellule de la grille.

```
template <typename DataTypes>
auto FictitiousGrid<DataTypes>::evaluate_level_set_cell(const
    CellIndex & cellIndex) const -> std::vector<float> {
    auto cell_nodes = p_grid->node_indices_of(cellIndex);
    std::vector<float> values(cell_nodes.size(), 0);

    for (UNSIGNED_INTEGER_TYPE i = 0; i < cell_nodes.size(); i++) {
        values[i] = evaluate_level_set(cell_nodes[i]);
    }
    return values;
}
```

Programme 15 – Fonction permettant d'évaluer la level-set à chaque noeud de la grille.

```
template <typename DataTypes>
auto FictitiousGrid<DataTypes>::evaluate_level_set_everywhere() const
-> std::vector<float>{

    const auto number_of_nodes = p_grid->number_of_nodes();

    std::vector<float> values(number_of_nodes, 0);
    for (UNSIGNED_INTEGER_TYPE i = 0; i < number_of_nodes; ++i) {
        values[i] = d_iso_surface->iso_value(p_grid->node(i));
    }
    return values;
}
```

Programme 16 – Code permettant de créer différentes fonctions level-set.

```
namespace SofaCaribou::topology {

class LevelSet : public IsoSurface<sofa::defaulttype::Vec3Types> {
    template < class T = void* >
    using Data = sofa::core::objectmodel::Data<T>;
    using Base = IsoSurface<sofa::defaulttype::Vec3Types>;
    using Base::Coord;
    using Base::Real;
public:

    LevelSet()
        : p_mesh_type(initData(&p_mesh_type, 1,
                               "mesh_type", "Type of the mesh."))
    {}

    inline Real iso_value(const Eigen::Matrix<Real, 3, 1> & X)
        const final {
        Real x = X[0];
        Real y = X[1];
        Real z = X[2];
        const auto mesh_type = p_mesh_type.getValue();

        /** objective for this part :
         * finally compute the signed distance between X
         * and the real boundary of the domain
    }}
```

```

        * to generate a fictitious grid for every complex domains
        */

Real value(0);
// Some examples of level-set functions

// sphere (center (0,0,0) radius 2):
if (mesh_type == 1.){
    value = x*x + y*y + z*z - 4.;}

// another sphere
else if (mesh_type == 2) {
    value = (x-0.5)*(x-0.5) + (y-0.5)*(y-0.5)
            + (z-0.5)*(z-0.5) - 1./8.;

}

// "Swiss-cheese block"
else if (mesh_type == 3){
    value = (x*x+y*y-4)*(x*x+y*y-4) + (z*z-1)*(z*z-1)
            + (y*y+z*z-4)*(y*y+z*z-4) + (x*x-1)*(x*x-1)
            + (z*z+x*x-4)*(z*z+x*x-4) + (y*y-1)*(y*y-1) -15;
}

// "doughnut"
else {
    value = (1.8- sqrt(x*x + y*y))*(1.8 - sqrt(x*x + y*y))
            + z*z - 0.8*0.8;
}
return value;
}

private:
Data<int> p_mesh_type;
};

}

```