INGÉNIEURS SUP GALILÉE - UNIVERSITÉ PARIS NORD

INSTITUT DE RADIOPROTECTION ET DE SÛRETÉ NUCLÉAIRE

RAPPORT DE STAGE DE FIN D'ÉTUDES

Simulation d'Écoulements Multiphasiques en Milieu Poreux avec Résolution par la Méthode Galerkin Discontinue

Stagiaire :
Maria Adela PUSCAS

Encadrant :
Magdalena DYMITROWSKA

Paris, 24 septembre 2011

Remerciements

Je tiens à remercier dans un premier temps madame Magdalena DYMITROWSKA de m'avoir offert la chance d'effectuer mon stage de fin d'étude au sein de l'IRSN. Je tiens tout particulièrement à la remercier de m'avoir apporté les explications nécessitant une intuition physique indispensable à la bonne compréhension du phénomène modélisé, et de m'avoir fait confiance lors des différentes initiatives que j'ai pu prendre dans mon travail de stagiaire.

Je remercie également Farid Smaï pour les différents échanges que nous avons eus à l'occasion de l'utilisation de la librairie libMesh.

Je suis très reconnaissante à Mathieu Gregory pour son aide lors de mes soucis ponctuels dans l'utilisation de l'interface graphique sous le cluster "CURIUM" de l'IRSN.

Je remercie également Hanen Amor pour sa bonne humeur et ses encouragements.

Je salue aussi mes collègues de bureau pour l'agréable ambiance qu'ils ont installé tout au long de mon stage, mais aussi à tout le service pour leur bon accueil.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

J'ai effectué mon stage de fin d'études au sein de l'Institut de Radioprotection et de Sûreté Nucléaire (IRSN), dans le service DSU / SSIAD / BERIS sous l'encadrement de madame Magdalena DYMITROWSKA.

Le stage a débuté le premier avril 2011 pour une période de 6 mois, et a pour objectif la résolution par la méthode Galerkin Discontinu d'un modèle d'écoulement eau-gaz dans un milieu poreux. L'un des objectifs du stage est l'implémentation de cette méthode numérique pour une intégration dans un code de simulation des écoulements multiphasiques miscibles dans des milieux poreux. Les résultats obtenus devront s'appliquer à la modélisation du transfert d'hydrogène à proximité d'une installation de stockage géologique de déchets radioactifs. Dans un premier temps, nous allons présenter l'IRSN et le service dans lequel s'est déroulé le stage. Puis nous mettrons l'accent sur les motivations du stage et ses objectifs. Nous présenterons dans la section 2 un modèle mathématique d'écoulement multiphasique dans un milieux poreux. La méthode de résolution utilisée étant Galerkin Discontinu (GD), nous avons fait le choix de la présenter dans un premier temps dans le cas simple du laplacien (section 3) en rappelant préalablement quelques résultats fondamentaux d'analyse numérique. Nous terminerons cette dernière section par une étude de convergence de la méthode GD. Une fois traité le cas simple du laplacien, nous appliquerons dans un deuxième temps la méthode GD (section 4) dans le cas du système d'équations modélisant les écoulements multiphasiques miscibles dans des milieux poreux. Les sections 5 et 6 seront consacrées à la présentation de l'implémentation du schéma numérique obtenu en C++ en passant par la librairie libMesh. Les résultats numériques obtenus feront l'objet de la section 7.

Nous renvoyons en annexe les résultats sur le raffinement de maillage, ainsi que le code C++ assemblant la matrice du système et son second membre. Enfin nous présentons un tableau récapitulant les grandeurs physiques intervenants dans notre modèle.

1.1 IRSN

L'IRSN est l'expert public en matière de recherche et d'expertise sur les risques nucléaires et radiologiques.
L'IRSN est un établissement public à caractère industriel et commercial placé sous la tutelle conjointe : Ministère de l'Ecologie, du Développement durable, des Transports et du Logement ; Ministère de l'Economie, des Finances et de l'Industrie ; Ministère de l'Enseignement supérieur et de la Recherche ; Ministère de la Défense ; Ministère du Travail, de l'Emploi et de la Santé.
Le champ de compétences de l'IRSN couvre l'ensemble des risques liés aux rayonnements ionisants, utilisés dans l'industrie ou la médecine, ou encore les rayonnements naturels. Plus précisément, l'IRSN exerce ses missions d'expertise et de recherche dans les domaines suivants :

- Surveillance radiologique de l'environnement et intervention en situation d'urgence radiologique.
- Radioprotection de l'homme.
- Prévention des accidents majeurs dans les installations nucléaires.
- Sûreté des réacteurs.
- Sûreté des usines, des laboratoires, des transports et des déchets.

– Expertise nucléaire de défense.

DSU / SSIAD / BERIS

Les équipes de recherche à l'IRSN sont constituées de laboratoires, regroupés en services, eux-mêmes répartis dans six directions opérationnelles.
La Direction de la Sûreté des Usines, Laboratoires, Transports et Déchets (DSU) est chargée des activités d'expertise et d'études tournées vers la maîtrise des risques présentés par les différents éléments du cycle du combustible, y compris les déchets et les transports.
Le Service de sûreté des irradiateurs, des accélérateurs et de la gestion des déchets (SSIAD) est chargé d'examiner du point de vue de la sûreté :

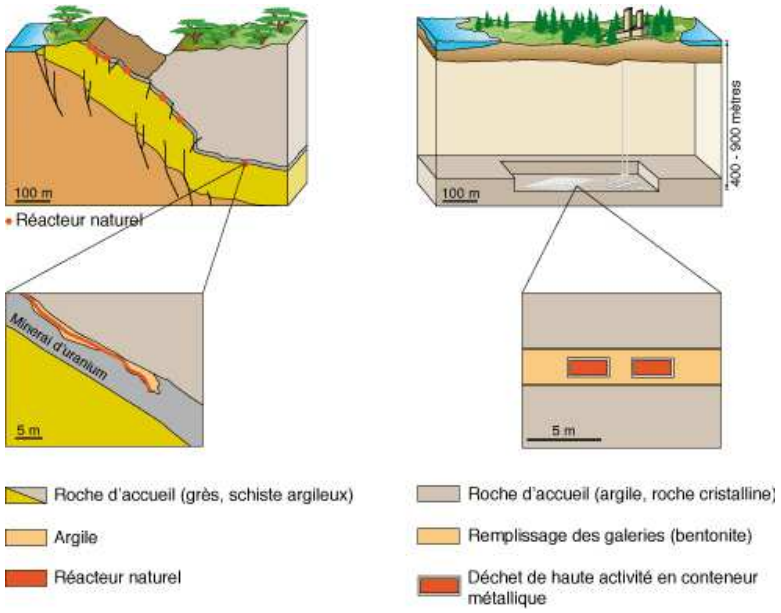
– les irradiateurs
– les accélérateurs de particules
– les laboratoires nucléaires du nord de la France
– les réacteurs nucléaires arrêtés définitivement, jusqu'à leur déclassement, ainsi que des installations de traitement et d'entreposage de déchets issus des réacteurs nucléaires
– et les installations de traitement et d'entreposage d'effluents et de déchets des sites du CEA, de leur création jusqu'à leur déclassement
– les colis de déchets, qu'ils soient destinés à être entreposés ou stockés, et les installations de stockage des déchets radioactifs.

Le BERIS est chargé de l'expertise de sûreté des stockages de déchets radioactifs de surface et des projets de stockage de déchets radioactifs en subsurface ou en couches géologiques profondes. Ce bureau assure également l'évaluation des capacités de confinement des différentes barrières des stockages de déchets radioactifs (barrière géologique et composants ouvragés). Ces travaux font intervenir une équipe pluridisciplinaire : hydrogéologues, géo-mécaniciens, géologues, géochimistes, ingénieurs de calcul, développeurs, etc.

1.2 Contexte de l'étude

Un déchet radioactif est une matière radioactive classifiée comme déchet. Les déchets radioactifs sont essentiellement issus de l'utilisation de l'énergie nucléaire : médecine nucléaire, production d'énergie, propulsion navale ou fabrication d'armes atomiques. La plus grande partie des déchets radioactifs proviennent de l'industrie électronucléaire qui utilise et génère des matières radioactives dans les différentes étapes du cycle du combustible nucléaire. La stratégie de cycle diffère selon les pays et les périodes : le combustible irradié (dont uranium et plutonium) est soit considéré comme une matière valorisable (recyclage partiel des isotopes fissiles) soit comme un déchet (stockage direct).
En France, selon la définition de la loi, un déchet radioactif est une matière radioactive ne pouvant être réutilisée ou retraitée (dans les conditions techniques et économiques du moment).
Ils sont classés notamment selon les deux critères suivants : la durée de leur activité radioactive et le niveau de radioactivité. Suivant leur taux de décroissance radioactive, les déchets sont appelés à "vie longue" ou à "vie courte". Les déchets de haute activité (HA) et les déchets de moyenne activité et à vie longue (MAVL) ce sont principalement les déchets issus du cœur du réacteur, hautement radioactifs pendant des centaines de milliers, voire millions d'années.

En France, après 15 ans de recherche organisée en 1991 par la loi Bataille, la solution de référence pour la gestion des déchets radioactifs de haute activité et de moyenne activité à vie longue est le stockage géologique. Cette solution reste débattue du point de vue technique (sûreté du concept par rapport à un entreposage notamment) et politique (processus décisionnel, choix du site pour l'éventuel centre de stockage).
Le **stockage en couche géologique profonde** est un mode de gestion des déchets radioactifs envisagé dans plusieurs pays pour les déchets de haute et moyenne activité à vie longue. Il consiste à conditionner ces déchets puis à les placer dans une formation géologique stable en interposant des barrières naturelles et artificielles entre les déchets et l'environnement. Ce mode de gestion repose sur l'hypothèse que la rétention des déchets peut atteindre une durée suffisante pour assurer leur décroissance radioactive.
Le stockage géologique est conçu pour retarder le relâchement et la migration des radioéléments sur une échelle de temps compatible avec leur période de décroissance. Il repose sur une conception multi-barrières dont le niveau le plus élevé est la formation géologique en elle-même. Les autres barrières mises en place sont le colis de déchets en lui-même, le colis de stockage ainsi que d'éventuels dispositifs de confinement telle une barrière ouvragée en bentonite.



L'eau est le principal facteur d'altération des colis de déchets radioactifs d'une part, et le principal vecteur des éléments radioactifs éventuellement relâchés dans la barrière géologique d'autre part. Cependant, plusieurs phénomènes peuvent conduire à une formation de gaz et même à l'apparition d'une phase gazeuse dans le système. En effet, selon le type de déchets, on peut retrouver dans les conteneurs de colis des métaux en acier non allié. Au contact de l'eau, on assisterait alors sur le long terme à un processus de transformation du métal en un état supérieur d'oxydation. On parle de corrosion des métaux qui s'accompagne d'une production d'hydrogène sous forme gazeuse. Vu la quantité importante de l'hydrogéné qui pourrait être produit on s'attend à l'apparition d'une phase gazeuse qui sera à l'origine d'une perturbation hydrique, mécanique et chimique de l'ensemble des éléments du stockage.
Compte tenu de l'ordre de grandeur de plusieurs centaines de milliers d'années à la quelle

on s'intéresse, le recours à la simulation numérique est inévitable puisqu'il est impossible de mener des expériences sur des durées comparables. Aussi les éléments de réponse qu'apporteraient la simulation numérique doivent prendre effet dans un cadre le plus réaliste possible pour mieux rendre compte des phénomènes recensés.
Ainsi, on s'intéresse dans le prochain chapitre à un modèle mathématique d'écoulement à deux phases dans un milieu poreux, l'eau principalement issue de la resaturation du milieu et le gaz produit par corrosion.

2 Modèle d'écoulement Liquide-Gaz à N-Gaz

En vue de la simulation du comportement d'un site profond de stockage et de son voisinage lors de la génération d'une grande quantité d'hydrogène en son sein, on présente ici une modélisation mathématique des écoulements diphasiques liquide-gaz en milieux poreux d'un mélange de N+1 composants (1 solvant et N composants gaz).
Ce modèle prendra en compte l'apparition, l'évolution et la disparition d'une phase gazeuse dans un milieu poreux. Les effets capillaires sont pris en compte ainsi que les échanges de matière entre les phases régis par l'équilibre thermodynamique local.
Dans le cadre du stockage, le solvant considéré sera l'eau et l'on pourra ainsi modéliser l'écoulement eau/hydrogène (N=1) mais aussi des situations plus complexes (N>1) telles que eau/hydrogène/air.
Ces informations peuvent être retrouvées de manière plus détaillées dans [11].

2.1 Description d'un mélange de 2 phases à N+1 composants

On présente ici les éléments constitutifs de la modélisation physique de l'écoulement d'un mélange eau/hydrogène dans un milieux poreux.
2 phases : liquide et gaz. On notera avec les indices l et g les quantités relatives respectivement à la phase liquide et à la phase gazeuse.
On note respectivement P_α, S_α, ρ_α et c_α la pression, la saturation, la densité massique et la concentration molaire (ou molarité) de la phase $\alpha \in \{g, l\}$ (l'ensemble des notations utilisées est synthétisé dans l'annexe).
N+1 composants : 1 solvant et N "gaz". On notera avec l'exposant $i \in \{0, \cdots, N\}$ les quantités relatives aux composants. On adoptera de plus la convention suivante : la valeur de l'exposant $i = 0$ sera dédiée au solvant tandis que $i = 1 \cdots N$ désignera le $i^{\text{ème}}$ composant gazeux.
On note M^i la masse molaire du composant $i \in \{0, \cdots, N\}$, et l'on note respectivement ρ_α^i, c_α^i et x_α^i la densité massique, la concentration molaire et la fraction molaire du composant $i \in \{0, \cdots, N\}$ dans la phase $\alpha \in \{g, l\}$.
Ces différentes quantités sont reliées par les relations suivantes :

$$\rho_\alpha^i = M^i c_\alpha^i \quad ; \quad x_\alpha^i = \frac{c_\alpha^i}{c_\alpha} \quad ; \quad \rho_\alpha = \sum_{k=0}^{N} \rho_\alpha^k \quad ; \quad c_\alpha = \sum_{k=0}^{N} c_\alpha^k \quad ; \tag{1}$$

On définit ϕ^i le *flux massique* du composant i par

$$\phi^i = \rho_l^i \mathbf{q}_l + \rho_g^i \mathbf{q}_g + \mathbf{j}_l^i + \mathbf{j}_g^i \ ; \ i \in \{0, \cdots, N\} \tag{2}$$

avec $\mathbf{j}_\alpha^i$ le flux massique de diffusion du composant i dans la phase α et $\mathbf{q}_\alpha$ la vitesse de Darcy de la phase α. On définit de plus X^i, la *densité massique* moyenne dans le mélange de phases du composant i par

$$X^i = S_l \rho_l^i + S_g \rho_g^i \ ; \ i \in \{0, \cdots, N\} \tag{3}$$

Étant donné un milieu de porosité Φ, l'équation de **conservation de la masse** pour chaque composant s'écrit :

$$\frac{\partial}{\partial t}\left(\Phi X^i\right) + div\left(\phi^i\right) = \mathsf{F}^i \ ; \ i \in \{0, \cdots, N\} \tag{4}$$

où F^i est la source volumique du composant i.
La mise en équation d'écoulements en milieux poreux repose sur le modèle de Darcy qui décrit la vitesse d'écoulement d'un fluide proportionnellement à sa viscosité et à la perméabilité du milieu. Cette vitesse que nous noterons q_α est appelée vitesse de Darcy et a pour expression :

$$\mathbf{q}_\alpha = -\mathbb{K}\frac{k_{r,\alpha}(S_\alpha)}{\mu_\alpha}\left(\nabla P_\alpha - \rho_\alpha \mathbf{g}\right) \tag{5}$$

où :
- $\mathbb{K}$ est le tenseur de perméabilité absolue
- $k_{r,\alpha}$ est la perméabilité relative de la phase α
- μ_α est la viscosité dynamique de la phase α
- **g** est l'accélération gravitationnelle

Le **flux massique** de diffusion du composant i dans la phase α est donné par

$$\mathbf{j}_\alpha^i = -\Phi M^i S_\alpha c_\alpha D_\alpha^i \nabla x_\alpha^i \ ; \ i \in \{0, \cdots, N\} \ , \ \alpha \in \{g, l\} \tag{6}$$

où D_α^i est le coefficient de diffusion moléculaire du composant i dans la phase α.
Ce modèle de base doit être complété par des différentes lois physiques que nous allons énumérer :

- **Équilibre mécanique et loi de pression capillaire** On suppose que les phases sont localement à l'équilibre mécanique. Du fait de la capillarité dans les pores, les pressions des phases liquides et gazeuses ne sont pas nécessairement égales et l'on définit la pression capillaire comme :
$$P_c = P_g - P_l \tag{7}$$
- **Équilibre thermique** On suppose que les phases liquide et gazeuse sont localement à la même température T.
- **Loi des gaz parfaits et loi de Dalton pour le mélange de gaz** On suppose également que le mélange gazeux se comporte comme un mélange de gaz parfait, ce qui permet d'écrire :
$$P_g^i = x_g^i P_g \quad \text{et} \quad P_g^i = \frac{\rho_g^i}{M^i} RT \ , \ i \in \{0, \cdots, N\} \tag{8}$$
avec P_g^i la pression partielle du composant i dans le gaz ; R la constante universelle des gaz parfaits et T la température du gaz.

- **Équilibre thermodynamique** on démontre que cette propriété revient à écrire :
$$x_g^i P_g = K^i x_l^i \quad , \ i \in \{0, \cdots, N\} \tag{9}$$
Ici les K^i dépendent des caractéristiques du liquide (température, pression et composition).

- **L'incompressibilité de l'eau** :
$$\rho_l^0 = 0$$

Grandeurs physiques invoquées :

– Les variables sont notées par :

$$S_\alpha \ , \ P_\alpha \ , \ \rho_\alpha \ , \ c_\alpha \ , \ \rho_\alpha^i \ , \ c_\alpha^i \ , \ x_\alpha^i \ , \ \mathbf{q}_\alpha \ , \ \mathbf{j}_\alpha^i \tag{10}$$

– Les données par :

$$T \ , \ \mathbf{g} \ , \ \Phi(\mathbf{x}) \ , \ \mathbb{K}(\mathbf{x}) \ , \ k_{r,\alpha}(\cdot, \mathbf{x}) \ ,$$
$$P_c(\cdot, \mathbf{x}) \ , \ \mathsf{F}^i(\mathbf{x}, t) \ , \ \mu_\alpha \ , \ D_\alpha^i \ , \ M^i$$

Ces données dépendant explicitement de la position **x**, du temps t, de la température T et/ou la pression de référence P_{ref}.

2.2 Choix des variables principales

La formulation mathématique du modèle physique présenté doit intégrer aussi "naturellement" que possible : la gestion de l'apparition/disparition de la phase de gaz et la gestion des milieux poreux hétérogènes.
Un dénombrement des équations et des variables physiques exposées précédemment nous permet de conclure que le problème est décrit par un système de $N+1$ équations aux dérivées partielles à $N+1$ inconnues.
Il s'agit donc de choisir $N+1$ variables principales permettant d'exprimer l'ensemble de variables (10), et le cas échéant leur gradient.
Parmi les variables énumérées ci-dessus seules certaines sont "naturellement" continues à l'interfaces d'une hétérogénéité. C'est le cas en particulier des pressions P_α mais aussi des fractions molaires x_α^i : ceci est dû au fait que ces grandeurs participent à des équilibres physiques et que leurs discontinuités induiraient un déséquilibre local en contradiction avec les hypothèses constitutives du modèle physique étudié.
Nous suivons le choix proposé par F. Smaï, A. Bourgeat et M. Dymitrowska à savoir **le jeu de variables** $(P_l, (x_l^i)_{i=1\cdots N})$ qui présente plusieurs avantages. Comme mentionné ci-dessus les variables sont continues à travers les frontières des discontinuités, sont physiquement définies en présence de liquide et peut être prolonger analytiquement quand le liquide disparaît. On va donc, exprimer les variables secondaires en fonction de $(P_l, (x_l^i)_{i=1\cdots N})$.

Rappelons que notre problème est décrit par le système de $N+1$ équations aux dérivées partielles :

$$\frac{\partial}{\partial t}\left(\Phi X^i\right) + div\left(\phi^i\right) = \mathsf{F}^i \ \ ; \ i \in \{0, \cdots, N\}$$

Les X^i sont explicités dans les variables principales. Les flux ϕ^i sont données par

$$\phi^i = \rho_l^i \mathbf{q}_l + \rho_g^i \mathbf{q}_g + \mathbf{j}_l^i + \mathbf{j}_g^i \ \ ; \ i \in \{0, \cdots, N\} \tag{11}$$

Expressions des flux
Il s'agit donc dans un premier temps d'expliciter chaque terme des ϕ^i dans le jeu de variables principales $\mathbf{u} = (u_i)_{i=0\cdots N} = (P_l, (x_l^i)_{i=1,\ldots,N})$.

Les flux de diffusion dans le liquide

$$\begin{cases} \mathbf{j}_l^0 = + \Phi M^0 S_l c_l D_l^0 \sum_{k=1}^{N} \nabla u_k \\ \mathbf{j}_l^i = - \Phi M^i S_l c_l D_l^i \nabla u_i \quad , \ i = 1 \cdots N \end{cases} \tag{12}$$

Les flux de diffusion dans le gaz

$$\mathbf{j}_g^i = -\Phi M^i S_g c_g D_g^i \sum_{k=0}^{N} \frac{\partial x_g^i}{\partial u_k} \nabla u_k \quad , \ i = 0 \cdots N \tag{13}$$

Les flux de convection dans le liquide

$$\begin{cases} \rho_l^0 \mathbf{q}_l = - \rho_l^0 \dfrac{k_{r,l}}{\mu_l} \left(\mathbb{K} \nabla u_0 - \rho_l \mathbb{K} \mathbf{g} \right) \\ \rho_l^i \mathbf{q}_l = - u_i \ M^i c_l \dfrac{k_{r,l}}{\mu_l} \left(\mathbb{K} \nabla u_0 - \rho_l \mathbb{K} \mathbf{g} \right) \quad , \ i = 1 \cdots N \end{cases} \tag{14}$$

Les flux de convection dans le gaz

$$\begin{cases} \rho_g^0 \mathbf{q}_g = - \rho_g^0 \dfrac{k_{r,g}}{\mu_g} \left(\displaystyle\sum_{k=0}^{N} \frac{\partial P_g}{\partial u_k} \mathbb{K} \nabla u_k - \rho_g \mathbb{K} \mathbf{g} \right) \\ \rho_l^i \mathbf{q}_l = - M^i K^i \dfrac{c_g}{P_g} \dfrac{k_{r,g}}{\mu_g} \left(\displaystyle\sum_{k=0}^{N} u_i \frac{\partial P_g}{\partial u_k} \mathbb{K} \nabla u_k - u_i \rho_g \mathbb{K} \mathbf{g} \right) \quad , \ i = 1 \cdots N \end{cases} \tag{15}$$

Forme canonique – Termes "elliptiques" et "hyperboliques"
On peut mettre notre problème sous la forme canonique suivante :

$$\Phi \frac{\partial X^i}{\partial t} - \text{div} \left[\sum_{k=0}^{N} A_{i,k} \nabla u_k + A_{i,g} \mathbf{g} + u_i \left(\sum_{k=0}^{N} C_{i,k} \nabla u_k + C_{i,g} \mathbf{g} \right) \right] = \mathcal{F}^i \quad , \ i = 0 \cdots N \tag{16}$$

Avec pour $i \in \{0, ..., N\}$ et $k \in \{0, ..., N, g\}$

$$A_{i,k} = \begin{cases} +\rho_l^0 \frac{k_{r,l}}{\mu_l} \mathbb{K} + \rho_g^0 \frac{k_{r,g}}{\mu_g} \frac{\partial P_g}{\partial u_k} \mathbb{K} & \text{si } i = 0 \text{ et } k = 0 \\ -\rho_l^0 \rho_l \frac{k_{r,l}}{\mu_l} \mathbb{K} - \rho_g^0 \frac{k_{r,g}}{\mu_g} \rho_g \mathbb{K} & \text{si } i = 0 \text{ et } k = g \\ -\Phi M^0 S_l c_l D_l^0 + \rho_g^0 \frac{k_{r,g}}{\mu_g} \frac{\partial P_g}{\partial u_k} \mathbb{K} & \text{si } i = 0 \text{ et } k > 0 \\ +\Phi M^i S_l c_l D_l^i & \text{si } i > 0 \text{ et } k = i \\ +\Phi M^i S_g c_g D_g^i \frac{\partial x_g^i}{\partial u_k} & \text{si } i = 0, ..., N \text{ et } k = 0, ..., N \end{cases} \tag{17}$$

et

$$C_{i,k} = \begin{cases} +M^i c_l \frac{k_{r,l}}{\mu_l} \mathbb{K} & \text{si } i > 0 \text{ et } k = 0 \\ -M^i K^i \frac{c_g}{P_g} \frac{k_{r,g}}{\mu_g} \mathbb{K} & \text{si } i > 0 \text{ et } k = i \\ -M^i c_l \rho_l \frac{k_{r,l}}{\mu_l} \mathbb{K} - M^i K^i \frac{c_g}{P_g} \frac{k_{r,g}}{\mu_g} \rho_g \mathbb{K} & \text{si } i > 0 \text{ et } k = g \\ +M^i K^i \frac{c_g}{P_g} \frac{k_{r,g}}{\mu_g} \frac{\partial P_g}{\partial u_k} \mathbb{K} & \text{si } i > 0 \text{ et } k \neq i \end{cases} \tag{18}$$

Les $A_{i,k}$ et $A_{i,g}$ sont appelés tenseurs "elliptiques" ; les $C_{i,k}$ ainsi que $C_{i,g}$ tenseurs "hyperboliques".

2.3 Formulation mathématique

Pour le problème étudié, nous allons considérer des conditions aux limites de types Neumann et Dirichlet, partant d'une condition initiale donnée.

Soit $T > 0$ un temps final donné, Ω un domaine borné de $\mathbb{R}^d$ de frontière $\partial\Omega$ et $N+1$ partitions $\Gamma_i^D \cup \Gamma_i^N = \partial\Omega$ ($i = 0, ..., N$) de la frontière du domaine. Ceci conduit au système d'EDP suivant :

$$\begin{cases} \dfrac{\partial X^i(\mathbf{u})}{\partial t} - div(\phi^i(\mathbf{u})) = F^i, \quad \text{dans }]0,T[\times\Omega, \text{ et } i = 0, ..., N \\ u_i(t=0) = u_i^0, \quad \text{dans } \Omega, \text{ et } i = 0, ..., N \\ u_i = u_i^D, \quad \text{sur }]0,T[\times\Gamma_i^D, \text{ et } i = 0, ..., N \\ \phi^i(\mathbf{u}) \cdot \nu = \phi^{i,N}, \quad \text{sur }]0,T[\times\Gamma_i^N, \text{ et } i = 0, ..., N \end{cases} \tag{19}$$

Avec :

- $\phi^i(\mathbf{u}) = \sum_{k=0}^N A_{i,k}(u)\nabla u_k + A_{i,g}(u)g + u_i \left(\sum_{k=0}^N C_{i,k}(u)\nabla u_k + C_{i,g}(u)g\right)$
- $(u_i^0)_{i=0\cdots N}$ les conditions initiales
- $(u_i^D)_{i=0\cdots N}$ les conditions de type Dirichlet
- $(\phi^{i,N}\cdots\nu)_{i=0\cdots N}$ les conditions de type Neumann
- ν la normale unitaire sortante du domaine

La discrétisation en temps s'appuie sur la méthode "Euler implicite". Le problème discret est résolu de manière itérative par une méthode de type "quasi Newton".
Étant donnée $0 = t^0 < t^1 < ... < t^{N_T}$ et $\Delta t^n = t^n - t^{n-1}$, la solution approchée au temps t^n qu'on notera u^n, est obtenue en calculant de manière itérative la suite $(u_q^n)_q$ définie par :

- étant donnée $u_0^n = u^{n-1}$
- $q \geq 0$, u_{q+1}^n est la solution du problème linéaire suivant :

$$\begin{cases} J_{X^i}(\mathbf{u}_q^n)\ \left(\dfrac{\mathbf{u}_{q+1}^n - \mathbf{u}_q^n}{\Delta t^n}\right) - \text{div}(\phi^i(\mathbf{u}_{q+1}^n, \mathbf{u}_q^n)) = F^i(t^n) - \dfrac{X^i(\mathbf{u}_q^n) - X^i(\mathbf{u}^{n-1})}{\Delta t^n}, \quad \text{dans } \Omega, \text{ et } i = 0, ..., N \\ u_{i,\ q+1}^n = u_i^D(t^n), \quad \text{sur } \Gamma_i^D, \text{ et } i = 0, ..., N \\ \phi^i(\mathbf{u}_{q+1}^n) \cdot \nu = \phi^{i,N}(t^n), \quad \text{sur } \Gamma_i^N, \text{ et } i = 0, ..., N \end{cases} \tag{20}$$

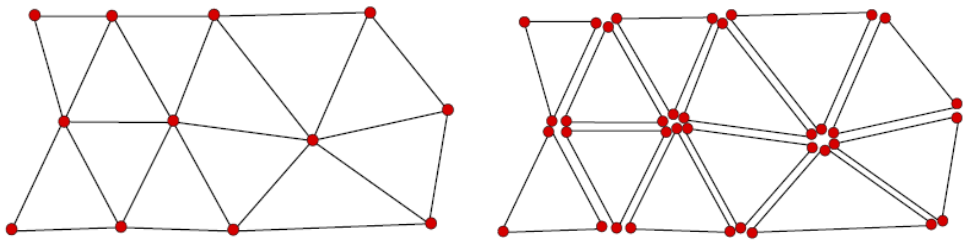
où $J_{X^i}(\mathbf{u}) = \left(\dfrac{\partial X^i}{\partial u_j}(\mathbf{u})\right)_{j=0,...,N}^T$ - la jacobienne de $X^i(\mathbf{u})$

et

$\phi^i(w,v) = \sum_{k=0}^N A_{i,k}(v)\nabla w_k + A_{i,g}(v)g + w_i \left(\sum_{k=0}^N C_{i,k}(v)\nabla v_k + C_{i,g}(v)g\right)$

3 Méthode de Galerkin Discontinue (GD)

Comme leur nom le laisse entendre, les méthodes de Galerkin discontinu sont basées sur la théorie des éléments finis de Galerkin à la différence près qu'elles autorisent la solution numérique à être discontinue. Chaque élément du maillage possède donc ses propres degrés de liberté et ne les partage pas avec d'autres. Les interactions entre éléments sont modélisées par des flux numériques qui peuvent parfois être complexes. C'est aujourd'hui une famille de méthodes numériques très populaire et prometteuse. Leur principal intérêt est la généralisation facile et compacte des méthodes à des formulations d'ordre élevé. Cela est dû au fait que la représentation polynômiale d'ordre élevé de la solution n'est pas reconstruite mais définie sur chaque élément du maillage.
Néanmoins, le coût de l'approche discontinue est particulièrement élevé. Les degrés de liberté nécessaires pour un calcul éléments finis et pour un calcul de Galerkin discontinu sont présentés dans la Figure 1. Il est clair que la discrétisation Galerkin discontinu demande beaucoup plus de degrés de liberté.



(a) Eléments finis (b) Galerkin discontinu

FIGURE 1 – Degrés de liberté nécessaires pour chacune des méthodes

3.1 Résultats fondamentaux

3.1.1 Rappels sur les espaces de Sobolev

Pour m entier positif, $1 \leq p \leq \infty$ et Ω un domaine ouvert de $\mathbb{R}^d$ on définit

$$W^{m,p}(\Omega) := \{v \in L^p(\Omega); D^\alpha v \in L^p(\Omega), |\alpha| \leq m\},$$

les dérivées $D^\alpha v = \dfrac{\partial^\alpha v}{\partial x_1^{\alpha_1} ... \partial x_d^{\alpha_d}}$ étant prises au sens des distributions avec la notation usuelle $|\alpha| = \alpha_1 + ... + \alpha_d$.
Pour $m = 0$ on pose $W^{0,p} = L^p$. On munit $W^{m,p}(\Omega)$ de la norme

$$\|v\|_{W^{m,p}} := \left(\sum_{|\alpha| \leq m} \|D^\alpha v\|_{L^p}^p \right)^{1/p}$$

et on définit aussi la semi-norme par :

$$|v|_{W^{m,p}} := \left(\sum_{|\alpha|=m} \|D^\alpha v\|_{L^p}^p \right)^{1/p}$$

de sorte que $\|v\|_{W^{m,p}} = \left(\|v\|_{W^{m-1,p}}^p + |v|_{W^{m,p}}^p \right)^{1/p}$
Dans le cas $p = \infty$ ces définitions sont modifiées en remplaçant les normes ℓ^p par des max. En partant de ces définitions et en utilisant le fait que $L^p(\Omega)$ est complet, on démontre facilement le résultat suivant.

Théorème 3.1. *$W^{m,p}(\Omega)$ muni de sa norme est un espace de Banach.*

Dans le cas $p = 2$ qui nous intéresse plus particulièrement, on pose $H^m(\Omega) := W^{m,2}(\Omega)$. La norme H^m dérive du produit scalaire

$$\langle v, w \rangle_{H^m} := \sum_{|\alpha| \leq m} \langle D^\alpha v, D^\alpha w \rangle_{L^2}$$

et par conséquent H^m est un espace de Hilbert.
Rappelons à présent les résultats concernant la restriction au bord ou trace des fonctions des espaces de Sobolev : si Ω est un ouvert lipschitzien, alors l'opérateur γ_0 qui à une fonction régulière u définie sur Ω associe sa restriction au bord $\partial\Omega$, $\gamma_0 : u \mapsto u|_{\partial\Omega}$ vérifie l'estimation :

$$\|\gamma_0 u\|_{L^2(\partial\Omega)} \leq C \|u\|_{H^1(\Omega)}$$

Par un argument de densité ceci montre que la trace γ_0 définit un opérateur continu de $H^1(\Omega)$ dans $L^2(\Omega)$. La continuité de l'opérateur trace permet de définir l'espace

$$H_0^1 := \left\{ v \in H^1(\Omega); \gamma_0 v = 0 \right\}$$

qui est un sous-espace hilbertien de $H^1(\Omega)$.
Inégalité de Poincaré : si Ω est un domaine lipschitzien borné, il existe une constante C_P telle que pour toute fonction v de H_0^1 on a :

$$\|v\|_{L^2} \leq C_P \|\nabla v\|_{L^2}$$

Cette inégalité nous indique que la semi-norme $\|\nabla v\|_{L^2}$ est une norme équivalente à la norme H^1 sur H_0^1. On la désigne parfois comme "norme H_0^1" et on note donc :

$$\|v\|_{H_0^1} = |v|_{H^1} = \|\nabla v\|_{L^2}$$

Nous renvoyons à [3] pour les développements et démonstration des résultats rapidement évoque dans ce paragraphe.

3.1.2 Formule de la divergence et de Green

Soit Ω un ouvert borné de $\mathbb{R}^d$ $(d = 2, 3)$, et n sa normale extérieure. Soit u et v deux fonctions régulières, w un champ de vecteurs définis sur Ω. Alors

$$\int_\Omega div(w)\, dx = \int_{\partial\Omega} w \cdot n\, d\sigma \quad \text{(formule de la divergence)}$$

$$\int_\Omega \Delta u\, v\, dx = -\int_\Omega \nabla u \cdot \nabla v\, dx + \int_{\partial\Omega} \frac{\partial u}{\partial n}\, v\, d\sigma \quad \text{(formule de Green)}$$

où $\frac{\partial u}{\partial n} = \nabla u \cdot n$ est la dérivée normale de u, avec n le vecteur unitaire normal extérieur au bord $\partial\Omega$.

3.1.3 L'inégalité de Cauchy-Schwarz et l'inégalité de Young

Inégalité de Cauchy - Schwarz :

$$\forall\, f, g \in L^2(\Omega),\ |(f,g)_{L^2(\Omega)}| \leq \|f\|_{L^2(\Omega)}\, \|g\|_{L^2(\Omega)}$$

Inégalité de Young :

$$\forall\, \varepsilon > 0, \forall\, a, b \in \mathbb{R}, \quad ab \leq \frac{\varepsilon}{2}\, a^2 + \frac{1}{2\,\varepsilon}\, b^2$$

3.1.4 Propriétés d'approximation

Dans cette section, nous rappelons des résultats d'approximation dans l'espace des polynômes de degré k. Pour des plus amples détails voir [8].

Théorème 3.2. *Soit T un triangle ou un parallélogramme en $2d$ ou un tétraèdre ou hexaèdre en $3d$ et h_T son diamètre. Soit $v \in H^s(T)$ avec $s \geq 1$ et $k \geq 0$ un entier. Alors, il existe une constante C indépendante de v et h_T, et une fonction $\tilde{v} \in P^k(T)$ telle que :*

$$\forall\, 0 \leq q \leq s, \quad \|v - \tilde{v}\|_{H^q(T)} \leq C\, h_T^{\min(k+1,s)-q}\, |v|_{H^s(T)}$$

Remarque : en particulier pour $v \in H^{k+1}(T)$, il existe une constante C indépendante de v et h_T, et une fonction $\tilde{v} \in P^k(T)$ telle que :

$$\|v - \tilde{v}\|_{L^2(T)} \leq C\, h_T^{k+1}\ \|v\|_{H^{k+1}(T)}$$

et

$$\|\nabla(v - \tilde{v})\|_{L^2(T)} \leq C\, h_T^{k}\ \|v\|_{H^{k+1}(T)}$$

3.2 Espace de Sobolev brisé

Les espaces de Sobolev brisés sont des espaces naturels pour travailler avec les méthodes de Galerkin discontinue. Ces espaces dépendent fortement de la partition du domaine. Soit $(\mathcal{T}_h)_{h>0}$ une famille de maillages non-structurés de Ω. Les maillages peuvent être non-coïncidents et sont réguliers au sens de Ciarlet (exemple Figure 2).

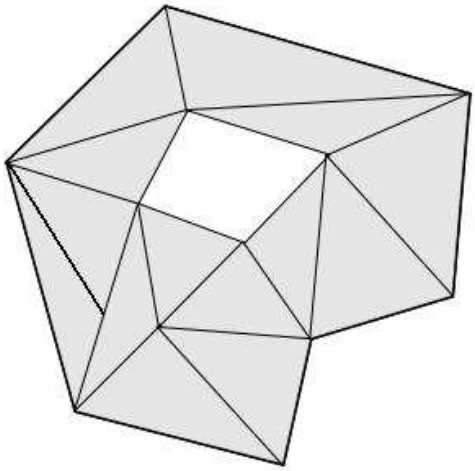


FIGURE 2 – Maillage non-structuré et non-coïncident

Cela signifie que si h_T désigne le diamètre de T et ρ_T désigne le diamètre maximal de la boule inscrite dans T, il existe une constante $\rho > 0$ tel que :

$$\forall\, T \in \mathcal{T}_h, \quad \frac{h_T}{\rho_T} \leq \rho$$

On introduit l'espace de Sobolev brisé pour un réel s,

$$H^s(\mathcal{T}_h) = \left\{ v \in L^2(\Omega) : \forall\, T\ \in \mathcal{T}_h, v|_T \in H^s(T) \right\}$$

muni de la norme de Sobolev brisé :

$$|\!|\!| v |\!|\!|_{H^s(\mathcal{T}_h)} = \left(\sum_{T \in \mathcal{T}_h} \|v\|^2_{H^s(T)} \right)^{1/2}$$

En particulier, on utilise la semi-norme brisée :

$$|\!|\!| \nabla v |\!|\!|_{H^0(\mathcal{T}_h)} = \left(\sum_{T \in \mathcal{T}_h} \|\nabla v\|^2_{L^2(T)} \right)^{1/2}$$

Clairement, on a :

$$H^s(\Omega) \subset H^s(\mathcal{T}_h) \quad \text{et} \quad H^{s+1}(\mathcal{T}_h) \subset H^s(\mathcal{T}_h)$$

L'opérateur "gradient brisé" $\nabla_h : H^1(\mathcal{T}_h) \to L^2(\Omega)^d$ est tel que :
pour une fonction $v \in H^1(\mathcal{T}_h)$, $\quad \nabla_h\, v|_T = \nabla(v|_T),\ \forall\, T \in \mathcal{T}_h$ (voir [4]).

3.3 Moyenne et saut

Pour un élément $T \in \mathcal{T}_h$, ∂T représente sa frontière et n_T désigne sa normale unitaire sortante.
L'ensemble $\mathcal{F}_h$ des faces du maillage est partitionné en $\mathcal{F}_h^i \cup \mathcal{F}_h^D \cup \mathcal{F}_h^N$, où $\mathcal{F}_h^i$ est l'ensemble des faces internes, $\mathcal{F}_h^D$ regroupe l'ensemble des faces sur lesquelles est imposée une condition de Dirichlet et $\mathcal{F}_h^N$ désigne l'ensemble des faces sur lesquelles est imposée une condition de Neumann.
Pour une face interne $F \in \mathcal{F}_h^i$, il existe T_1 et T_2 dans $\mathcal{T}_h$ tels que $F = \partial T_1 \cap \partial T_2$. Nous désignons par n_F la normale unitaire à F dirigée de T_1 vers T_2 (voir Figure 3).

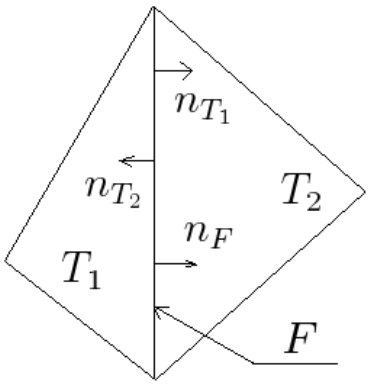


FIGURE 3 – Interface entre deux éléments : définition et notation

Nous définissons l'opérateur de moyenne $\{\}_F$ et l'opérateur de saut $[\![]\!]_F$ de la façon suivante : pour une fonction v pouvant prendre deux valeurs sur F,

$$\{v\}_F \overset{def}{=} \tfrac{1}{2}(v_1 + v_2) \quad \text{et} \quad [\![v]\!]_F \overset{def}{=} v_1 - v_2$$

avec $v_i = v|_{T_i},\ i = 1, 2$.
On étend ces définitions aux faces du bord :

$$\{v\}_F = [\![v]\!]_F = v|_T, \quad \forall\ F = \partial T \cap \partial\Omega$$

Pour une fonction vectorielle, les opérateurs de moyenne et de saut sont définis composante par composante.
Notons que le signe du saut est arbitraire mais sans conséquence pour la suite puisque les sauts seront toujours multipliés par la normale n_F.

3.4 Espace d'éléments finis discontinus

L'espace d'éléments finis discontinus V_h est défini par :

$$V_h \overset{def}{=} \mathbb{P}_d^k(\mathcal{T}_h) = \left\{ v \in L^2(\Omega), \forall\, T \in \mathcal{T}_h,\ v|_T \in \mathbb{P}_d^k(T) \right\}$$

dans lequel $\mathbb{P}_d^k(T)$ est l'ensemble des polynômes de degré inférieur ou égale à k sur un élément quelconque T. Les fonctions de l'espace V_h n'étant pas continues, cela permet de choisir des fonctions de base dont le support correspond à un élément du maillage (voir [12]).
Dans la Figure 4 sont présentés les degrés de liberté nécessaires pour un calcul Galerkin discontinu en fonctions de différents valeurs de $k = 0, 1, 2$.

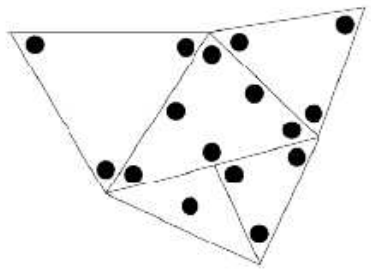


FIGURE 4 – Degrés de liberté nécessaires pour GD

3.5 Inégalité inverse

Lemme 3.1. *Soit $(\mathcal{S}_h)_{h>0}$ une famille de maillages non-structurés de Ω, coïncident et régulières au sens de Ciarlet, alors $\exists\ C_{inv}$ tel que $\forall\ h, \forall\ S \in \mathcal{S}_h$ et $\forall\ v_h \in \mathbb{P}_d^k(\mathcal{S}_h)$ on a :*

$$\|\nabla v_h\|_{L^2(S)^d} \leq C_{inv}\ h_S^{-1}\ \|v_h\|_{L^2(S)}$$

Pour une famille de maillages généraux non-coïncident et régulière au sens de Ciarlet, $(\mathcal{T}_h)_{h>0}$, sous l'hypothèse que $\exists\ \mathcal{S}_h$ un sous - maillage de $\mathcal{T}_h$ tel que :

1. $\mathcal{S}_h$ est composé de simplex coïncidente et régulière au sens de Ciarlet (voir 3.2), tel que $\forall\ S \in \mathcal{S}_h, \exists\,!\ T \in \mathcal{T}_h$ tel que $S \subset T$
2. $\exists\ \rho > 0$ tel que $\forall\ S' \in \mathcal{S}_T$ on a : $h_{S'} \geq\ \rho h_T$
   où $\mathcal{S}_T := \{S' \in \mathcal{S}_h, S' \subset T\}$ et h_T désigne le diamètre de T.
3. $\forall\ F' \in \mathfrak{F}_h, \exists\,!\ F \in \mathcal{F}_h$ tel que $F' \subset F$
   $\mathfrak{F}_h$ désigne l'ensemble des faces du $\mathcal{S}_h$.

on a le résultat suivant (cf. [6]) :

Lemme 3.2. *$\exists\ C_{inv}$ tel que $\forall\ h, \forall\ T \in \mathcal{T}_h$ et $\forall\ v_h \in \mathbb{P}_d^k(\mathcal{T}_h)$ on a :*

$$\|\nabla v_h\|_{L^2(T)^d} \leq C_{inv}\ h_T^{-1}\ \|v_h\|_{L^2(T)}$$

3.6 Problème modèle

Nous allons illustrer la méthode de Galerkin discontinue sur l'exemple simple mais pertinent du problème du laplacien (ou équation de Poisson) : on cherche une fonction u telle que :

$$\begin{cases} -\Delta u = f, & \text{dans } \Omega \\ u = g, & \text{sur } \Gamma = \partial\Omega \end{cases} \tag{21}$$

où Ω désigne un ouvert borné de $\mathbb{R}^d$ $(d = 2, 3)$, $\partial\Omega$ désigne la frontière de Ω, f est une fonction définie sur Ω, $f \in L^2(\Omega)$ et $g \in L^2(\partial\Omega)$. Par la suite, nous allons considérer le cas $g \equiv 0$.
La formulation variationelle du problème (21) est la suivante :

$$(\Pi) : \begin{cases} \text{Trouver } u \in V := H_0^1(\Omega) \text{ tel que :} \\ a(u,v) := \int_\Omega \nabla u \cdot \nabla v = \int_\Omega f\ v := L(v), \quad \forall\, v \in V \end{cases}$$

L'existence de la solution du Π va découler du théorème de **Lax-Milgram** suivant.
On considère un problème général pouvant se mettre sous la forme variationnelle

$$(V) : \begin{cases} \text{Trouver } u \in X(\Omega) \text{ tel que :} \\ a(u,v) = L(v), \quad \forall\, v \in X \end{cases}$$

Théorème 3.3. *Lax - Milgram*
On suppose que X est un espace de Hilbert et que les formes a et L vérifient les hypothèses suivantes :

1. *Continuité de L :*
   il existe une constante C_L telle que $|L(v)| \leq C_L \|v\|_X$ pour tout $u, v \in X$.
2. *Continuité de a :*
   il existe une constante C_a telle que $|a(u,v)| \leq C_a \|u\|_X \ \|v\|_X$ pour tout $u, v \in X$.
3. *Coercivité de a : $a(u,u) \geq \alpha \ \|u\|_X^2$, pour tout $u \in X$, avec $\alpha > 0$.*

Alors il existe une solution unique u au problème (V) qui vérifie l'estimation a priori

$$\|u\|_X \leq \frac{\|L\|_{X'}}{\alpha}$$

avec $\|L\|_{X'} = \sup_{\|v\|_X = 1} |L(v)|$ la norme de L dans le dual X' de X.

Proposition 3.1. *Une fonction $v \in V \cap H^1(\mathcal{T}_h)$ si et seulement si, pour tout F face du maillage $\mathcal{T}_h$, le saut de la fonction est nul sur F $(\llbracket v \rrbracket_F = 0, \forall\ F \in \mathcal{F}_h)$.*

Preuve : Soit $v \in H^1(\mathcal{T}_h)$ et Φ une fonction test à support compact dans Ω : $\Phi \in \mathcal{C}_0^\infty(\Omega)$.
On rappelle que le gradient brisé $\nabla_h v$ est telle que : $\nabla_h v|_T = \nabla v|_T,\ \forall\, T \in \mathcal{T}_h$.
On multiplie $\nabla_h v$ par la fonction test Φ, on intègre sur Ω, ensuite on décompose l'intégrale sur Ω, sur les éléments du maillage et on intègre par partie (formule de Green) :

$$\int_\Omega \nabla_h v \Phi = \sum_{T \in \mathcal{T}_h} \int_T \nabla v \Phi = -\int_\Omega v \nabla \Phi + \sum_{T \in \mathcal{T}_h} \int_{\partial T} \Phi\ n_T v \tag{22}$$

On rappelle que $\mathcal{F}_T$ représente l'ensemble des faces de l'élément T, en conséquence on a :

$$\sum_{T\in\mathcal{T}_h}\int_{\partial T}\Phi\ n_T v = \sum_{T\in\mathcal{T}_h}\sum_{F\in\mathcal{F}_T}\int_F \Phi\ n_T v$$

La double somme est en fait égale à la somme sur les faces du maillage

$$= \sum_{F\in\mathcal{F}_h^b}\int_F \Phi\ n_T v + \sum_{F\in\mathcal{F}_h^i}\int_F (\Phi|_{T_1}\ n_{T_1} v|_{T_1} + \Phi|_{T_2}\ n_{T_2} v|_{T_2})$$

Soit $n_F = n_{T_1} = -n_{T_2}$, d'après la définition de l'opérateur de saut on a :

$$= \sum_{F\in\mathcal{F}_h^b}\int_F \Phi\ n_F v + \sum_{F\in\mathcal{F}_h^i}\int_F [\![\Phi\ n_F v]\!]$$

Comme $[\![a\ b]\!] = [\![a]\!]\ \{b\} + [\![b]\!]\ \{a\}$

$$= \sum_{F\in\mathcal{F}_h^b}\int_F \Phi\ n_F v + \sum_{F\in\mathcal{F}_h^i}\int_F [\![v]\!]\ n_F\{\Phi\} + [\![\Phi]\!]\ n_F\{v\}$$

Φ étant une fonction test quelconque, on suppose $\{\Phi\} = \Phi$ et $[\![\Phi]\!] = 0$, on obtient :

$$\sum_{T\in\mathcal{T}_h}\int_{\partial T}\Phi\ n_T v = \sum_{F\in\mathcal{F}_h^b}\int_F \Phi\ n_F v + \sum_{F\in\mathcal{F}_h^i}\int_F [\![v]\!]\ n_F\ \Phi = \sum_{F\in\mathcal{F}_h}\int_F [\![v]\!]\ n_F\ \Phi$$

On injecte ce résultat dans l'équation (22) :

$$\int_\Omega \nabla_h v\Phi = -\int_\Omega v\nabla\Phi + \sum_{F\in\mathcal{F}_h}\int_F [\![v]\!]\ n_F\ \Phi$$

" $\Rightarrow$ " Si $v \in H^1(\Omega)$ alors $\nabla_h v = \nabla v$, cela implique : $\forall\ \Phi \in \mathcal{C}_0^\infty(\Omega),\ \sum_{F\in\mathcal{F}_h}\int_F [\![v]\!]\ n_F\ \Phi = 0$, soit $[\![v]\!] = 0, \forall\ F \in \mathcal{F}_h$

" $\Leftarrow$ " Si $v \in H^1(\mathcal{T}_h)$ et $[\![v]\!] = 0, \forall\ F \in \mathcal{F}_h$ alors on a :

$\int_\Omega \nabla_h v\Phi = -\int_\Omega v\nabla\Phi,\ \forall\ \Phi \in \mathcal{C}_0^\infty$ d'où $v \in H^1(\Omega)$.

Proposition 3.2. *Si u est solution de Π alors $[\![\nabla u]\!]\ n_F = 0,\ \forall\ F \in \mathcal{F}_h^i$*

Preuve : On multiplie (21) par φ, fonction test à support compact dans Ω, on intègre sur Ω, on décompose l'intégrale sur les éléments du maillage :

$$\int_\Omega f\varphi = -\int_\Omega \Delta u\varphi = -\sum_{T\in\mathcal{T}_h}\int_T \Delta u\varphi$$

On fait une intégration par partie (F. Green)

$$= \sum_{T\in\mathcal{T}_h}\int_T \nabla u\cdot\nabla\varphi - \sum_{T\in\mathcal{T}_h}\int_{\partial T}\nabla u\, n_T\,\varphi$$

Comme précédemment, le deuxième terme s'écrit comme la somme sur les faces du maillage

$$= \int_\Omega \nabla u\cdot\nabla\varphi - \sum_{F\in\mathcal{F}_h^i}\int_F [\![\nabla u]\!]\, n_F\varphi$$

Or u est solution de Π

$$= \int_\Omega f\varphi - \sum_{F\in\mathcal{F}_h^i}\int_F [\![\nabla u]\!]\, n_F\varphi$$

Cela implique que $\sum_{T\in\mathcal{F}_h^i}\int_F [\![\nabla u]\!]\, n_F\varphi = 0,\ \forall\,\varphi\in\mathcal{C}_0^\infty,$ par conséquent

$$[\![\nabla u]\!]\, n_F = 0,\ \forall\, F\in\mathcal{F}_h^i.$$

Proposition 3.3. $\forall\, v_h\in V_h := \mathbb{P}_d^k, k\geq 1, \exists\,\sigma>0$ *tel que* $\|v_h\|_{L^2(\Omega)}\leq\sigma\ \|v_h\|_{GD}$ *, où*

$$\|v_h\|_{GD} := \left(\|\nabla_h v_h\|^2_{L^2(\Omega)^d} + |v_h|_J^2\right)^{1/2}$$

$$\textit{et}\quad |v_h|_J^2 := \sum_{F\in\mathcal{F}_h}\frac{1}{h_F}\int_F \|[\![v_h]\!]\|^2_{L^2(F)}$$

Remarque : Si $v\in H^1(\mathcal{T}_h)$ on peut également définir la norme GD de v :

$$\|v\|_{GD} := \left(\|\nabla v\|^2_{L^2(\Omega)^d} + |v|_J^2\right)^{1/2} \text{ et } |v|_J^2 := \textstyle\sum_{F\in\mathcal{F}_h}\frac{1}{h_F}\|[\![v]\!]\|^2_{L^2(F)}$$

Remarque : Si $v\in W^{2,1}(\Omega)$ alors v a une trace dans $L^1(\partial T), \forall\, T\in\mathcal{T}_h$.

Hypothèse : On fait l'hypothèse suivante sur la solution de Π : $u\in H_0^1(\Omega)\cap W^{2,1}(\Omega) := V_*$

On va chercher une forme bilinéaire $a_h\in(V_{*h}\times V_h;\ \mathbb{R})$ où $V_{*h} = V_h + V_*$ de façon a satisfaire les trois propriétés suivantes :

1. **Consistance** : cette propriété réalise la liaison entre le problème modèle et la formulation variationnelle : une forme bilinéaire a_h est consistante si :

$$a_h(u, v_h) = \int_\Omega f v_h, \forall\, v_h\in V_h$$

où u représente la solution exacte.

2. **Stabilité** : une forme bilinéaire a_h est stable si $\exists$ une constante positive C_{stab} telle que :

$$\forall\, v_h\in V_h,\quad a_h(v_h, v_h)\geq C_{stab}\,\|v_h\|^2_{GD}$$

3. **Continuité** : une forme bilinéaire a_h est continue si $\exists$ une constante positive C_{bud} telle que :
$$\forall\,(w,v_h)\in V_{*h}\times V_h,\quad a_h(w,v_h)\leq C_{bud}\,\|w\|_{GD,*}\,\|v_h\|_{GD}$$

Théorème 3.4. *Si les propriétés de stabilité, consistance et continuité sont satisfaite, alors on a l'estimation d'erreur a priori suivante :*

$$\|u-u_h\|_{GD}\leq\left(1+\frac{C_{bud}}{C_{stab}}\right)\inf_{y_h\in V_h}\|u-y_h\|_{GD,*}$$

Preuve : Soit $y_h\in V_h$, d'après la propriété de stabilité on a :

$$C_{stab}\ \|v_h-y_h\|_{GD}\leq a_h(v_h-y_h,v_h-y_h)\leq\sup_{w_h\in V_h}\frac{a_h(v_h-y_h,w_h)}{\|w_h\|_{GD}}$$

En utilisant la propriété de consistance on obtient :

$$C_{stab}\ \|v_h-y_h\|_{GD}\leq\sup_{w_h\in V_h}\frac{a_h(u-y_h,w_h)}{\|w_h\|_{GD}}$$

Or d'après la propriété de continuité on a :

$$C_{stab}\ \|v_h-y_h\|_{GD}\leq C_{bud}\,\|u-y_h\|_{GD,*}$$

En utilisant ce résultat et l'inégalité de Cauchy-Schwarz, on obtient le résultat énoncé :

$$\begin{aligned}\|u-u_h\|_{GD}&\leq\|u-y_h\|_{GD}+\|u_h-y_h\|_{GD}\\&\leq\|u-u_h\|_{GD,*}+\frac{C_{bud}}{C_{stab}}\|u-y_h\|_{GD,*}\\&\leq\left(1+\frac{C_{bud}}{C_{stab}}\right)\inf_{y_h\in V_h}\|u-y_h\|_{GD,*}\end{aligned}$$

On propose d'abord la forme bilinéaire suivante :

$$a_h^0(w,v_h)=\int_\Omega\nabla_h w\cdot\nabla_h v_h$$

On décompose l'intégrale sur les éléments du $\mathcal{T}_h$ et on fait une intégration par partie :

$$a_h^0(w,v_h)=\sum_{T\in\mathcal{T}_h}\int_T\nabla w\cdot\nabla v_h=-\sum_{T\in\mathcal{T}_h}\int_T\Delta w\ v_h+\sum_{T\in\mathcal{T}_h}\int_{\partial T}\nabla w\ n_T\ v_h$$

Comme on l'à déjà vu dans la proposition (3.1), on peut l'écrire $\sum_{T\in\mathcal{T}_h}\int_{\partial T}$ comme une somme sur les faces du maillage :

$$\begin{aligned}\sum_{T\in\mathcal{T}_h}\int_{\partial T}\nabla w\ n_T\ v_h&=\sum_{F\in\mathcal{F}_h^b}\int_F\nabla w\ n_F\ v_h+\sum_{F\in\mathcal{F}_h^i}\int_F[\![\nabla w v_h]\!]\ n_F\\&=\sum_{F\in\mathcal{F}_h^b}\int_F\nabla w\ n_F\ v_h+\sum_{F\in\mathcal{F}_h^i}\int_F[\![\nabla w]\!]\ n_F\{v_h\}+[\![v_h]\!]\ n_F\{\nabla w\}\end{aligned}$$

Par conséquent :

$$a_h^0(w,v_h) = -\sum_{T\in\mathcal{T}_h}\int_T \Delta\, w\, v_h + \sum_{F\in\mathcal{F}_h^b}\int_F \nabla w\, n_F\, v_h + \sum_{F\in\mathcal{F}_h^i}\int_F [\![\nabla w]\!]\, n_F\, \{v_h\} + [\![v_h]\!]\, n_F\, \{\nabla w\}$$

Nous allons vérifier les trois propriétés énoncées précédemment :

1 - Consistance : en remplaçant w par la solution exacte u, et en utilisant le résultat de la proposition (3.2), on a :

$$a_h^0(u,v_h) = \int_\Omega f v_h + \sum_{F\in\mathcal{F}_h}\int_F [\![v_h]\!]\, n_F\, \{\nabla u\}$$

a_h^0 est donc non-consistante, en conséquence on va modifier a_h^0 de façon à avoir une forme bilinéaire consistante. On pose :

$$a_h^1(w,v_h) = a_h^0(w,v_h) - \sum_{F\in\mathcal{F}_h}\int_F [\![v_h]\!]\, n_F\, \{\nabla w\}$$

Cette modification nous conduit à une forme bilinéaire consistante, car :

$$a_h^1(u,v_h) = \int_\Omega f v_h,\ \forall\, v_h \in V_h$$

On peut encore modifier a_h^1 de façon a avoir une forme bilinéaire symétrique en posant :

$$a_h^{cs}(w,v_h) = a_h^1(w,v_h) - \sum_{F\in\mathcal{F}_h}\int_F [\![w]\!]\, n_F\, \{\nabla v_h\}$$

La consistance est conservée car d'après la proposition (3.1) : $[\![u]\!]_F = 0, \forall\, F \in \mathcal{F}_h$

2 - Stabilité : Soit w égale à v_h, alors a_h^{cs} devient :

$$a_h^{cs}(v_h,v_h) = \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - 2\sum_{F\in\mathcal{F}_h}\int_F [\![v_h]\!]\, n_F \{\nabla v_h\}, \quad \forall\, v_h \in V_h \tag{23}$$

L'étude de la stabilité va s'appuier sur le résultat suivant :

Lemme 3.3.

$$\left|\sum_{F\in\mathcal{F}_h}\int_F [\![v_h]\!]\, n_F \{\nabla w\}\right| \le C_{tr}\, N_\partial^{1/2}\, \|\nabla_h w\|_{L^2(\Omega)^d}\ |v_h|_J, \quad \forall\, (w,v_h) \in V_{*h}\times V_h$$

Preuve : Par Cauchy Schwarz on a :

$$\begin{aligned}\left|\sum_{F\in\mathcal{F}_h}\int_F [\![v_h]\!]\, n_F \{\nabla w\}\right| &\le \left(\sum_{F\in\mathcal{F}_h}\int_F h_F|\{\nabla w\}n_F|^2\right)^{1/2}\ \left(\sum_{F\in\mathcal{F}_h}\int_F h_F^{-1}|[\![v_h]\!]n_F|^2\right)^{1/2}\\ &= \left(\sum_{F\in\mathcal{F}_h}\int_F h_F|\{\nabla w\}n_F|^2\right)^{1/2}\ |v_h|_J\end{aligned}$$

Par définition de l'opérateur de moyenne on a :

$$\{\nabla w\} = \frac{1}{card(\mathcal{F}_T)} \sum_{F \in \mathcal{F}_T} \int_F \nabla w|_T$$

Par conséquent

$$\int_F h_F |\{\nabla w\} n_F|^2 \leq \sum_{F \in \mathcal{F}_T} \int_F |\nabla w|_T|^2$$

Grâce au théorème de trace

$$\sum_{F \in \mathcal{F}_T} \int_F |\nabla w|_T|^2 \leq C_{tr}^2 \sum_{F \in \mathcal{F}_T} \int_F h_F^{-1} \|\nabla w\|^2_{L^2(T)}$$

Finalement, on trouve :

$$\sum_{F \in \mathcal{F}_h} \int_F h_F |\{\nabla w\} n_F|^2 \leq C_{tr}^2 \, N_\partial \|\nabla_h w\|^2_{L^2(\Omega)^d},$$

où $N_\partial = \max_{T \in \mathcal{T}_h} card(\mathcal{F}_T)$. Ce qui nous conduit au résultat annonce :

$$\left| \sum_{F \in \mathcal{F}_h} \int_F [\![v_h]\!] \, n_F \{\nabla w\} \right| \leq C_{tr} \, N_\partial^{1/2} \, \|\nabla_h w\|_{L^2(\Omega)^d} \; |v_h|_J$$

On injecte ce résultat, pour $w = v_h$, dans (23), on obtient :

$$a_h^{cs}(v_h, v_h) \geq \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - 2C_{tr} \, N_\partial^{1/2} \, \|\nabla_h v_h\|_{L^2(\Omega)^d} \; |v_h|_J$$

On utilise l'inégalité de Young avec $a = \|\nabla_h v_h\|_{L^2(\Omega)^d}$ et $b = C_{tr} \, N_\partial^{1/2} \, |v_h|_J$, cela conduit à :

$$a_h^{cs}(v_h, v_h) \geq \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - \varepsilon \, \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - \frac{C_{tr}^2 \, N_\partial}{\varepsilon} |v_h|_J^2$$

Pour avoir la stabilité il faut contrôler $\frac{C_{tr}^2 \, N_\partial}{\varepsilon}$. Pour ce faire on va introduire un terme dit de pénalisation :

$$\sum_{F \in \mathcal{F}_h} \frac{\eta}{h_F} \int_F [\![w]\!][\![v_h]\!],$$

où η est telle que $\eta \geq C_{tr}^2 \, N_\partial$.
On définit la forme bilinéaire a_h^{SIPG}, en partant de a_h^{cs} et en lui ajoutant le terme de pénalisation :

$$a_h^{SIPG} = a_h^{cs} + \sum_{F \in \mathcal{F}_h} \frac{\eta}{h_F} \int_F [\![w]\!][\![v_h]\!]$$

Cela conduit à une forme bilinéaire stable car :

$$a_h^{SIPG} \geq \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - \varepsilon \, \|\nabla_h v_h\|^2_{L^2(\Omega)^d} - \frac{C_{tr}^2 \, N_\partial}{\varepsilon} |v_h|_J^2 + \eta |v_h|_J^2 \geq C_{stab} \, \|v_h\|^2_{GD}$$

Remarque : la consistance est conservée car $[\![u]\!] = 0$.

3 - Continuité : Par Cauchy Schwarz on a :

$$\int_\Omega \nabla_h w \, \nabla_h v_h \leq \|\nabla_h w\|^2_{L^2(\Omega)^d} \, \|\nabla_h v_h\|^2_{L^2(\Omega)^d} \leq \|w\|_{GD} \; \|v_h\|_{GD}$$

et

$$\sum_{F\in\mathcal{F}_h} \int_F [\![w]\!] \, n_F \{\nabla v_h\} \leq |v_h|_J \left(\sum_{\mathcal{F}_h} h_F \int_F |\{\nabla w_h\} \, n_F|^2 \right)^{1/2}$$
$$\leq |v_h|_J \left(\sum_{\mathcal{T}_h} h_F \int_{\partial T} |\{\nabla w_h\} \, n_F|^2 \right)^{1/2}$$

On définit la norme "$GD, *$" :

$$\|w\|^2_{GD,*} := \|w\|^2_{GD} + \sum_{T\in\mathcal{T}_h} h_F \int_{\partial T} |\{\nabla w_h\} \, n_F|^2$$

Par conséquent,

$$\sum_{F\in\mathcal{F}_h} \int_F [\![w]\!] \, n_F \{\nabla v_h\} \leq \|w\|_{GD,*} \, \|v_h\|_{GD}$$

et

$$\sum_{F\in\mathcal{F}_h} \frac{\eta}{h_F} \int_F [\![w]\!][\![v_h]\!] \leq \|w\|_{GD,*} \, \|v_h\|_{GD}$$

En effet,

$$a_h^{SIPG}(w, v_h) \leq C_{bud} \, \|w\|_{GD,*} \, \|v_h\|_{GD}$$

ce qui prouve que a_h^{SIPG} est continue.
Pour résumer, la formulation GD associée au problème (21) est la suivante :

$$(\Pi) : \begin{cases} \text{Trouver } u_h \in V(\mathcal{T}_h) \text{ tel que :} \\ a_h^{SIPG}(u_h, v_h) = L(v_h), \quad \forall \, v_h \in V(\mathcal{T}_h) \end{cases}$$

où ,

$$a_h^{SIPG}(w, v) = \int_\Omega \nabla_h w \cdot \nabla_h v - \sum_{F\in\mathcal{F}_h} \int_F [\![v]\!] \, n_F \, \{\nabla w\}$$
$$- \sum_{F\in\mathcal{F}_h} \int_F [\![w]\!] \, n_F \, \{\nabla v\} + \sum_{F\in\mathcal{F}_h} \frac{\eta}{h_F} \int_F [\![w]\!][\![v]\!]$$

et

$$L(v) = \int_\Omega f \, v$$

Plus généralement, la méthode des éléments finis GD associée au problème (21) est comme suit :

$$(\Pi) : \begin{cases} \text{Trouver } u_h \in V(\mathcal{T}_h) \text{ tel que :} \\ a_{h,\epsilon}(u_h, v_h) = L(v_h), \quad \forall \, v_h \in V(\mathcal{T}_h) \end{cases}$$

où ,

$$a_{h,\epsilon}(w,v) = \int_\Omega \nabla_h w \cdot \nabla_h v - \sum_{F\in\mathcal{F}_h} \int_F [\![v]\!]\, n_F\, \{\nabla w\} + \epsilon \sum_{F\in\mathcal{F}_h} \int_F [\![w]\!]\, n_F\, \{\nabla v\} + \sum_{F\in\mathcal{F}_h} \frac{\eta}{h_F} \int_F [\![w]\!][\![v]\!]$$

En fonction du paramètre ϵ on obtient quelques variantes de la méthode Galerkin discontinue qui sont apparues dans la littérature à différents moments :

- Si $\epsilon = -1$ la méthode est appelée "symmetric interieur penalty Galerkin" (**SIPG**)
- Si $\epsilon = 1$ la méthode est appelée "nonsymmetric interieur penalty Galerkin" (**NIPG**)
- Si $\epsilon = 0$ la méthode est appelée "incomplete interieur penalty Galerkin " (**IIPG**)

Proposition 3.4. *Soit $v \in V_* \cap H^{k+1}(\Omega)$, alors $\exists$ une constante C_{app} indépendante de h et une fonction $\tilde{v} \in P^k(\Omega)$ telle que :*

$$\|v - \tilde{v}\|_{GD,*} \leq C_{app}\, h^k\ \|v\|_{H^{k+1}(\Omega)}$$

Preuve : La preuve est immédiate et utilise les résultats des approximations présenté dans la section 3.1.4.

Proposition 3.5. *Soit $u \in H^{k+1}(\Omega) \cap V$ et u_h la solution de Π, alors $\exists$ une constante $C > 0$ telle que :*

$$\|u - u_h\|_{L^2(\Omega)} \leq C h^k\ \|v\|_{H^{k+1}(\Omega)}$$

Preuve :
D'après la proposition 3.3 on sait qu'il existe une constante σ telle que :

$$\frac{1}{\sigma}\|u - u_h\|_{L^2(\Omega)} \leq \|u - u_h\|_{GD}$$

En utilisant le résultat du théorème 3.4 on a :

$$\|u - u_h\|_{GD} \leq \left(1 + \frac{C_{bud}}{C_{stab}}\right) \inf_{y_h \in V_h} \|u - y_h\|_{GD,*}$$

En utilisant le résultat de la proposition 3.4 on a :

$$\|u - u_h\|_{GD} \leq \left(1 + \frac{C_{bud}}{C_{stab}}\right) C_{app}\, h^k\ \|v\|_{H^{k+1}(\Omega)}$$

Donc,

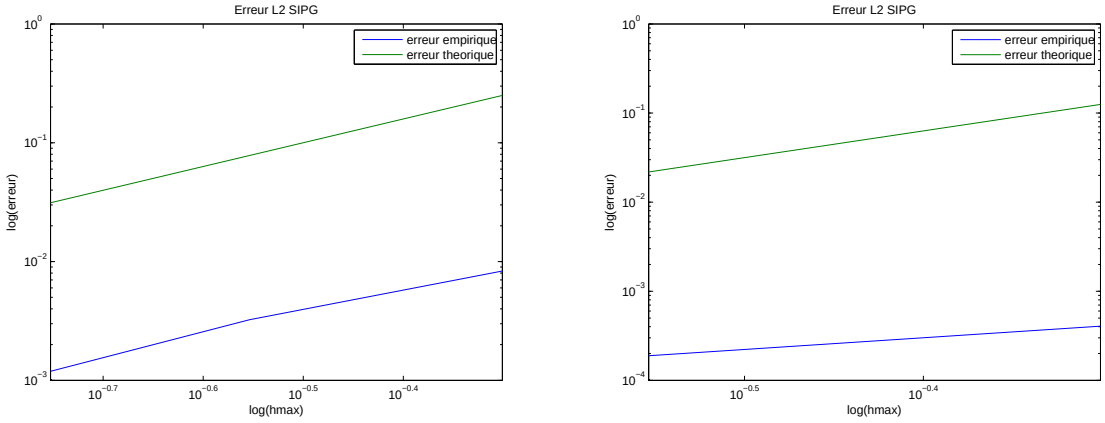
$$\|u - u_h\|_{L^2(\Omega)} \leq \underbrace{\sigma \left(1 + \frac{C_{bud}}{C_{stab}}\right) C_{app}}_{C}\ h^k\ \|v\|_{H^{k+1}(\Omega)}$$

Remarque : En introduisant les opérateurs de lifting et en utilisant le théorème d'Aubin Netsche on peut montrer que :

$$\|u - u_h\|_{L^2(\Omega)} \leq C\, h^{k+1}\ \|v\|_{H^{k+1}(\Omega)}$$

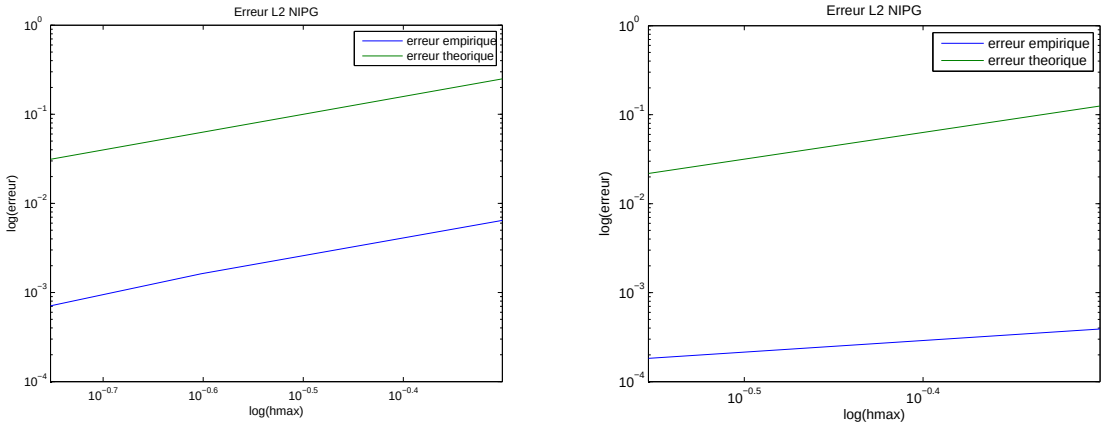
3.7 Convergence

Nous avons tracé les courbes d'erreurs donnant l'ordre de convergence au sens de l'erreur L^2 pour différentes valeurs de ϵ et de k, et ceci pour la solution exacte $u(x, y) = x^2 + y^2$ dans (21).
Ci-dessous, les courbes tracées donnant le logarithme de l'erreur en fonction du logarithme de la distance caractéristique maximale du maillage notée $hmax$, montrent un ordre de convergence égal à 2 dans le cas où l'espace des approximation est composé de fonctions linéaires par morceaux (P^1), et un ordre de convergence égal à 3 pour des fonctions quadratiques par morceaux (P^2).



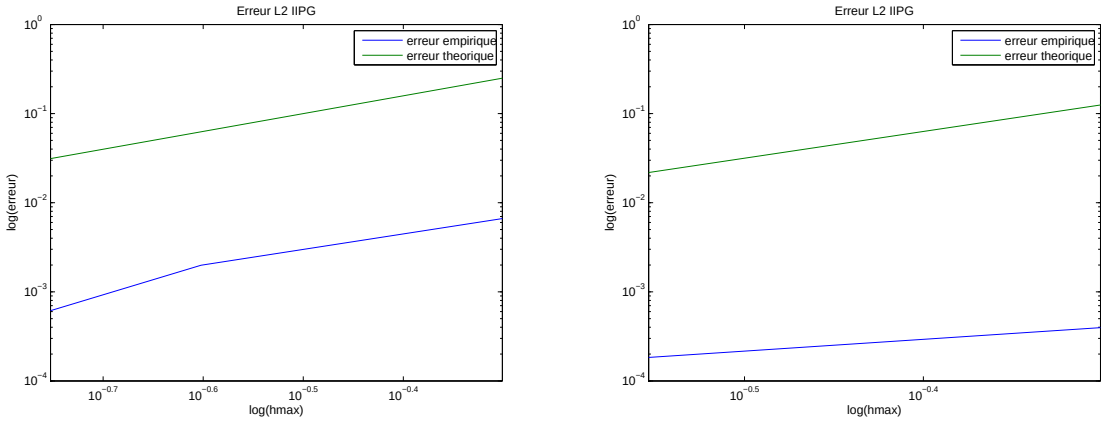
(a) ordre de convergence = 2 si $k = 1$ (b) ordre de convergence = 3 si $k = 2$

FIGURE 5 – SIPG - ordre de convergence



(a) ordre de convergence = 2 si $k = 1$ (b) ordre de convergence = 3 si $k = 2$

FIGURE 6 – NIPG - ordre de convergence



(a) ordre de convergence = 2 si $k = 1$ (b) ordre de convergence = 3 si $k = 2$

FIGURE 7 – IIPG - ordre de convergence

4 Discrétisation en espace

On cherche $\mathbf{u} = (u_i)_{i=0,...,N} = (P_l, (x_l^i)_{i=1,...,N})$ solution de :

$$\begin{cases} -div(\phi^i(w,v)) = F^i, & \text{dans } \Omega, \text{ et } i = 0, ..., N \\ w_i = u_i^D, & \text{sur } \Gamma_i^D, \text{ et } i = 0, ..., N \\ \phi^i(w,v) \cdot \nu = \phi^{i,N}, & \text{sur } \Gamma_i^N, \text{ et } i = 0, ..., N \end{cases} \tag{24}$$

Avec :

$$\phi^i(w,v) = \sum_{k=0}^N A_{i,k}(v)\nabla w_k + A_{i,g}(v)g + w_i \left(\sum_{k=0}^N C_{i,k}(v)\nabla v_k + C_{i,g}(v)g \right)$$

où $w = \mathbf{u}_{q+1}^n$ et $v = \mathbf{u}_q^n$ (voir 2.3).

4.1 GD formulation variationnelle

Le problème (24) est discrétisé en espace par la méthode Galerkin discontinue. Le domaine Ω est subdivisé en éléments (voir les notations introduites dans la section 3.2).
Premièrement, les termes de diffusion "$\nabla.(A_{i,k}\nabla u_k)$" sont discrétisés par la forme habituelle bilinéaire a_ϵ semblable à celle donnée dans la section 3.6 :

a_ϵ est la forme bilinéaire définie sur : $H^s(\mathcal{T}_h) \times H^s(\mathcal{T}_h) \to \mathbb{R}$

$$
\begin{aligned}
a_\epsilon(w,z) = & \sum_{T\in\mathcal{T}_h}\sum_{k=0}^{N}\int_T A_{i,k}(v)\ \nabla w_k\ \nabla z \\
& - \sum_{F\in\mathcal{F}_h^i\cup\mathcal{F}_D}\sum_{k=0}^{N}\int_F \{A_{i,k}(v)\ \nabla w_k\} n_F\ [\![z]\!] \\
& + \epsilon\sum_{F\in\mathcal{F}_h^i}\sum_{k=0}^{N}\int_F \{A_{i,k}(v)\ \nabla z\} n_F\ [\![w_k]\!] + \epsilon\sum_{F\in\mathcal{F}_D}\int_F (A_{i,i}(v)\ \nabla z\cdot n_F)\ w_i \\
& + \sum_{F\in\mathcal{F}_h^i}\sum_{k=0}^{N}\frac{\eta}{h_F}\int_F [\![w_k]\!]\ [\![z]\!] + \sum_{F\in\mathcal{F}_D}\frac{\eta}{h_F}\int_F w_i\ z
\end{aligned}
\tag{25}
$$

Remarque :

1. L'inégalité de Cauchy-Schwarz et l'inégalité de trace impliquent que tous les termes intégrales définis précédemment ont un sens si les fonctions appartiennent à $H^s(\mathcal{T}_h),\ s \geq 3/2$.
2. La forme bilinéaire a_ϵ contient le paramètre ϵ qui peut prendre les valeurs $-1, 0, 1$.

Cette forme bilinéaire induit la norme GD suivante : pour $w \in H^s(\mathcal{T}_h)^{N+1},\ s \geq 3/2$,

$$\|w\|_{GD}^2 = \sum_{i=0}^{N}\|w_i\|_{GD}^2$$

où :

$$\|w_i\|_{GD}^2 = \left(\|\nabla_h w_i\|_{L^2(\Omega)^d}^2 + |w_i|_J^2\right)^{1/2}$$

$$|w_i|_J^2 := \sum_{F\in\mathcal{F}_h}\frac{1}{h_F}\|[\![w_i]\!]\|_{L^2(F)}^2$$

Deuxièmement, les termes convectifs "$\nabla.(u_i\ C_{i,k}\ \nabla u_k)$ " sont discrétisés par la forme bilinéaire $c : H^s(\mathcal{T}_h)\times H^s(\mathcal{T}_h)\to\mathbb{R}$ définie par :

$$c(w_i,z) = \sum_{T\in\mathcal{T}_h}\int_T \xi_i\cdot\nabla z - \sum_{F\in\mathcal{F}_h^i}\int_F \tilde{\xi}_{\mathbf{i}}\cdot n_F\ [\![z]\!] \tag{26}$$

où :

$$\xi_i := w_i\ \zeta_i,\quad \zeta_i := \left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)g\right)$$

et $\tilde{\xi}_{\mathbf{i}}\cdot n_F$ désigne le flux numérique. Nous allons considérer les deux cas suivants :

- **flux centre :**

$$\tilde{\xi}_{\mathbf{i}}\cdot n_F = \{\xi_i\}\cdot n_F$$

- **flux upwind :**

$$\tilde{\xi}_{\mathbf{i}}\cdot n_F = \xi_{\mathbf{i}}^{\mathbf{up}}\cdot n_F = \begin{cases}\xi_i|_{T_1}\cdot n_F, & \text{si}\quad \{\zeta_i\}\cdot n_F > 0\\ \xi_i|_{T_2}\cdot n_F, & \text{si}\quad \{\zeta_i\}\cdot n_F \leq 0\end{cases}$$

Et $L : H^s(\mathcal{T}_h) \to \mathbb{R}$ est la forme linéaire définie par :

$$\begin{aligned} L(z) = &+ \sum_{T\in\mathcal{T}_h} \int_T F^i z + \sum_{F\in\mathcal{F}_N} \int_F \phi^{i,N} z \\ &- \sum_{T\in\mathcal{T}_h} \int_T A_{i,g}(v)\, g\, \nabla z + \sum_{F\in\mathcal{F}_D} \int_F (A_{i,g}(v)\, g\, n_F)\, z \\ &+ \sum_{F\in\mathcal{F}_D} \int_F \left(\epsilon A_{i,i}(v)\, n_F\, \nabla z + \frac{\eta}{h_F}\, z \right) u_i^D \\ &+ \sum_{F\in\mathcal{F}_D} \int_F \left(\sum_{k=0}^{N} C_{i,k}(v)\, \nabla v_k\, n_F\, z + C_{i,g}(v)\, g\, n_F\, z \right) u_i^D \end{aligned} \tag{27}$$

La formulation variationnelle GD du (24) est la suivante :

$$(\mathrm{II}) : \begin{cases} \text{Trouver } w_i \in H^s(\mathcal{T}_h) \text{ tel que :} \\ a_\epsilon(w,z) + c(w_i,z) = L(z), \quad \forall\, z \in H^s(\mathcal{T}_h), s \geq 3/2,\ i = 0, ..., N \end{cases} \tag{28}$$

Remarque :
Le problème (28) est dépendant du choix de n_F. Par conséquent, soit n_F une face telle que $F = \partial T_i \cup \partial T_j$, soit n_{ij} la normale unitaire dirigée de T_i vers T_j. Si n_F coïncide avec n_{ij} alors on a :

$$\{\nabla v\, n_F\}[\![w]\!] = \{\nabla v\, n_{ij}\}(w|_{T_i} - w|_{T_j})$$

Si n_F coïncide avec n_{ji} le saut de $[\![w]\!]$ a un signe différent :

$$\{\nabla v\, n_F\}[\![w]\!] = \{\nabla v\, n_{ji}\}(w|_{T_j} - w|_{T_i}) = \{\nabla v\, (-n_{ij})\}(w|_{T_j} - w|_{T_i}) = \{\nabla v\, n_{ij}\}(w|_{T_i} - w|_{T_j})$$

on obtient donc la même expression que précédemment, ce qui montre que (28) est indépendant du choix du n_F.

4.2 Propriétés

4.2.1 Consistance

D'après la formule de Green on a :

$$\sum_{T\in\mathcal{T}_h} \sum_{k=0}^{N} \int_T A_{i,k}(v)\, \nabla u_k \cdot \nabla z = \sum_{T\in\mathcal{T}_h} \int_T -div \left(\sum_{k=0}^{N} A_{i,k}(v)\, \nabla u_k \right) z + \int_{\partial T} \sum_{k=0}^{N} A_{i,k}(v)\, \nabla u_k\, n_T z$$

$\mathcal{F}_T$ désigne l'ensemble des faces de l'élément T, par conséquent on a :

$$\sum_{T\in\mathcal{T}_h}\int_{\partial T}\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k\ n_T z=\sum_{T\in\mathcal{T}_h}\sum_{F\in\mathcal{F}_T}\int_F\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k\ n_T z$$

La double somme est en fait égale à la somme sur les faces du maillage

$$=\sum_{F\in\mathcal{F}_h^b}\sum_{k=0}^{N}\int_F(A_{i,k}(v)\ \nabla u_k)n_F\ z$$

$$+\sum_{F\in\mathcal{F}_h^i}\int_F\{\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k\}n_F\ [\![z]\!]+\int_F\underbrace{[\![\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k]\!]n_F\ \{z\}}_{=0,\text{ car }[\![\nabla u_k]\!]\ n_F=0}$$

Or $\mathcal{F}_h^b=\mathcal{F}_h^D\cup\mathcal{F}_h^N$

$$=\sum_{F\in\mathcal{F}_h^D}\sum_{k=0}^{N}\int_F(A_{i,k}(v)\ \nabla u_k)n_F\ z+\sum_{F\in\mathcal{F}_h^N}\sum_{k=0}^{N}\int_F(A_{i,k}(v)\ \nabla u_k)n_F\ z$$

$$+\sum_{F\in\mathcal{F}_h^i}\int_F\{\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k\}n_F\ [\![z]\!]$$

Par conséquent,

$$a_{h,\epsilon}(u,z)=\sum_{T\in\mathcal{T}_h}\int_T-div\left(\sum_{k=0}^{N}A_{i,k}(v)\ \nabla u_k\right)z$$

$$+\sum_{F\in\mathcal{F}_h^N}\sum_{k=0}^{N}\int_F(A_{i,k}(v)\ \nabla u_k)n_F\ z$$

$$+\sum_{F\in\mathcal{F}_D}\int_F\left(\epsilon A_{i,i}(v)\ n_F\ \nabla z+\frac{\eta}{h_F}\ z\right)u_i$$

Ensuite, on intègre par partie le premier terme du c_h :

$$\sum_{T\in\mathcal{T}_h}\int_T\sum_{k=0}^{N}(u_i\ C_{i,k}(v)\nabla v_k)\ \nabla z+\int_T(u_i\ C_{i,g}(v)\ g)\nabla z$$

$$=\sum_{T\in\mathcal{T}_h}\int_T-div\left[u_i\left(\sum_{k=0}^{N}\ C_{i,k}(v)\nabla v_k+C_{i,g}(v)\ g\right)\right]z+\int_{\partial T}u_i\left(\sum_{k=0}^{N}\ C_{i,k}(v)\nabla v_k+C_{i,g}(v)\ g\right)\ n_T\ z$$

Comme précédemment, on réécrit les intégrales sur les bords de T comme une somme des

intégrales sur les faces de T, ce qui conduit à :

$$\sum_{T\in\mathcal{T}_h}\int_{\partial T} u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\ n_T\ z =$$
$$= \sum_{F\in\mathcal{F}_h^D}\int_F u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\ n_F\ z + \sum_{F\in\mathcal{F}_h^N}\int_F u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\ n_F\ z$$
$$+ \sum_{F\in\mathcal{F}_h^i}\int_F \{u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\}\ n_F\ [\![z]\!] + \underbrace{[\![u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)]\!]}_{=0,\text{ car }\ [\![u_i]\!]=0}\ n_F\ \{z\}$$

En utilisant ce résultat, la forme bilinéaire c_h devient :

$$\begin{aligned} c_h(u_i,z) = &\sum_{T\in\mathcal{T}_h}\int_T -div\left[u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\right] z \\ &+ \sum_{F\in\mathcal{F}_h^N}\int_F u_i\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\ n_F\ z \\ &+ \sum_{F\in\mathcal{F}_h^D}\int_F u_i^D\left(\sum_{k=0}^{N} C_{i,k}(v)\nabla v_k + C_{i,g}(v)\ g\right)\ n_F\ z \end{aligned}$$

En sommant $a_{h,\epsilon}$ et c_h on trouve :

$$\begin{aligned} a_{h,\epsilon}(u,z) + c_h(u_i,z) = &\sum_{T\in\mathcal{T}_h}\int_T F^i z + \sum_{F\in\mathcal{F}_N}\int_F \underbrace{\phi^i(u,v)\cdot\ n_F\ z}_{\phi^{i,N}z} \\ &- \sum_{T\in\mathcal{T}_h}\int_T A_{i,g}(v)\ g\ \nabla z + \sum_{F\in\mathcal{F}_D}\int_F (A_{i,g}(v)\ g\ n_F)\ z \\ &+ \sum_{F\in\mathcal{F}_D}\int_F \left(\epsilon A_{i,i}(v)\ n_F\ \nabla z + \frac{\eta}{h_F}\ z\right) u_i^D \\ &+ \sum_{F\in\mathcal{F}_D}\int_F \left(\sum_{k=0}^{N} C_{i,k}(v)\ \nabla v_k\ n_F\ z + C_{i,g}(v)\ g\ n_F\ z\right) u_i^D \\ &= L(z) \end{aligned}$$

Ce qui prouve la consistance de Π

4.2.2 Stabilité

Le schéma est stable par construction, dû au terme de pénalisation η qui doit être choisi convenablement. Nous reviendrons sur ce point *très important* au paragraphe (7.2).

4.2.3 Continuité

L'étude de la continuité est similaire à celle effectuée à la section 3.6. Il n'est pas difficile de vérifier que :

$$a_\epsilon(w,z) + c(w,z) \leq C_{bud} \|w\|_{GD,*} \|z\|_{GD}$$

4.3 Schéma Galerkin discontinue

La méthode des éléments finis GD est comme suit :

$$(\Pi)_h : \begin{cases} \text{Trouver } U_i \ \in V_h := \mathbb{P}_d^k(\mathcal{T}_h) \text{ tel que :} \\ a_{h,\epsilon}(U,z) + c_h(U_i,z) = L(z), \quad \forall\, z \in \mathbb{P}_d^k(\mathcal{T}_h), i = 0, ..., N \end{cases} \tag{29}$$

Selon le choix de ϵ, nous obtenons les **SIPG**, **NIPG** et **IIPG** variations des méthodes GD(voir section 3.6).

5 Implémentation de la méthode GD

5.1 L'élément de référence

Lors de l'implémentation de la méthode GD, il faut calculer des intégrales sur les volumes (comme triangles ou quadrilatères en 2d, tétraèdres ou hexaèdres en 3d) et les faces (telles que les arrêtes en 2d, des triangles ou quadrilatères en 3d). Il serait trop coûteux pour calculer les intégrales sur chaque élément physique dans la maille. Une approche plus économique et plus efficace est d'utiliser un changement de variable afin d'obtenir une intégrale sur un élément fixe, appelé l'élément de référence.

L'élément de référence triangulaire : il se compose d'un triangle $\hat{T}$ de sommets $\hat{a}_1(0,0)$, $\hat{a}_2(1,0)$, et $\hat{a}_3(0,1)$. Pour un élément T physique donné de sommets $a_i(x_i, y_i)$, il existe une unique transformation affine F_T qui envoie $\hat{a}_i$ sur a_i pour tout $i = 1, 2, 3$ (voir Figure 8), définie par :

$$F_T\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = \begin{pmatrix}x\\ y\end{pmatrix}, \quad x = \sum_{i=1}^{3} x_i\,\hat{\phi}_i(\hat{x},\hat{y}), \quad y = \sum_{i=1}^{3} y_i\,\hat{\phi}_i(\hat{x},\hat{y})$$

où

$$\hat{\phi}_1(\hat{x},\hat{y}) = 1-\hat{x}-\hat{y}, \quad \hat{\phi}_2(\hat{x},\hat{y}) = \hat{x}, \quad \hat{\phi}_3(\hat{x},\hat{y}) = \hat{y}$$

Nous pouvons réécrire la transformation :

$$\begin{pmatrix}x\\ y\end{pmatrix} = F_T\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = B_T\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} + b_T$$

où B_T est une matrice 2×2 et b_T un vecteur. Il est facile de montrer que :

$$B_T = \begin{pmatrix}x_2-x_1 & x_3-x_1\\ y_2-y_1 & y_3-y_1\end{pmatrix}, \quad b_T = \begin{pmatrix}x_1\\ y_1\end{pmatrix}$$

Nous avons : $det(B_T) = 2|T|$, où $|T|$ désigne l'aire de T.
Si v est une fonction définie sur T, on définit $\hat{v}$ sur $\hat{T}$ par :

$$\hat{v}(\hat{x}) = v(x) \Leftrightarrow \hat{v} = v \circ F_T$$

On remarque que si $\hat{v} = v \circ F_T$ alors on a :

$$\nabla\hat{v}(\hat{x}) = B_T^t \nabla v(x),$$

où B_T^t est la transposée de B_T.
De manière générale, on définit le simplexe de référence $\hat{T}$ dont les sommets sont donnés par l'origine $\hat{a}_0 = (0, ..., 0)$ et les points $\hat{a}_i = (0, ..., 1, ..., 0)$ dont toutes les coordonnées sont nulles sauf la i-ème qui vaut 1. On note F_T l'unique transformation affine qui envoie $\hat{a}_i$ sur a_i pour tout $i = 0, ..., d$. On a :

$$F_T(\hat{x}) = B_T(\hat{x}) + b_T$$

où B_T est une matrice $d \times d$ dont la i-ème colonne est donnée par les coordonnées de $a_i - a_0$. Si le simplexe T est non-dégénère, F_T est une bijection qui envoie $\hat{T}$ sur T et la matrice B_T

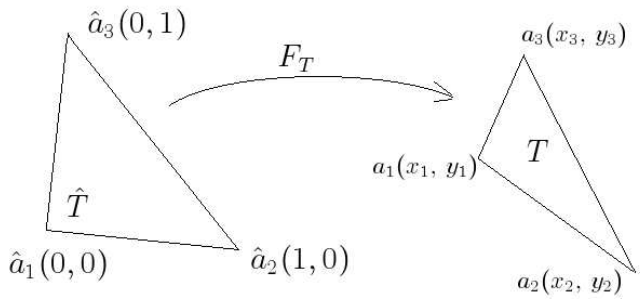


FIGURE 8 – Élément de référence versus élément physique

est inversible. Nous renvoyons à [9] pour les développements et démonstration des résultats rapidement évoque dans ce paragraphe.

5.2 Fonctions de base

En raison de l'absence de contraintes de continuité entre les éléments du maillage pour les fonctions test, les fonctions de base de $\mathbb{P}_d^k(\mathcal{T}_h)$ ont un support contenu dans un élément. On écrit :

$$\mathbb{P}_d^k(\mathcal{T}_h) = Vect\left\{\phi_i^T,\ 1 \leq i \leq N_{loc},\ T \in \mathcal{T}_h\right\}$$

avec

$$\phi_i^T(x) = \begin{cases} \hat{\phi}_i^T \circ F_T, & \text{si } x \in T \\ 0, & \text{si } x \notin T \end{cases}$$

Les fonctions base locales $\left(\hat{\phi}_i^T\right)_{1\leq i\leq N_{loc}}$ sont définies sur l'élément de référence. Il suffit d'utiliser les fonctions monômes. Par exemple, en $2d$, nous avons :

$$\hat{\phi}_i^T(\hat{x},\hat{y}) = \hat{x}^I\ \hat{y}^J, \quad I+J=i, \quad 0 \leq i \leq k$$

Cela donne la dimension locale :

$$N_{loc} = \frac{(k+1)(k+2)}{2}$$

Par exemple, nous avons pour les fonctions linéaires par morceaux $(k=1)$:

$$\hat{\phi}_0^T(\hat{x},\hat{y}) = 1, \quad \hat{\phi}_1^T(\hat{x},\hat{y}) = \hat{x}, \quad \hat{\phi}_2^T(\hat{x},\hat{y}) = \hat{y}$$

et pour les fonctions quadratiques par morceaux $(k=2)$:

$$\hat{\phi}_0^T(\hat{x},\hat{y}) = 1, \quad \hat{\phi}_1^T(\hat{x},\hat{y}) = \hat{x}, \quad \hat{\phi}_2^T(\hat{x},\hat{y}) = \hat{y}$$
$$\hat{\phi}_3^T(\hat{x},\hat{y}) = \hat{x}^2, \quad \hat{\phi}_4^T(\hat{x},\hat{y}) = \hat{x}\hat{y}, \quad \hat{\phi}_5^T(\hat{x},\hat{y}) = \hat{y}^2$$

De même, en $3d$, nous définissons

$$\hat{\phi}_i^T(\hat{x},\hat{y}) = \hat{x}^I\ \hat{y}^J\ \hat{z}^K, \quad I+J+K=i, \quad 0 \leq i \leq k$$

Cela donne la dimension locale :

$$N_{loc} = \frac{(k+1)(k+2)(k+3)}{6}$$

Remarque : la flexibilité des méthodes de GD nous permet de changer facilement les fonctions de base. Pour des plus amples détails sur ce paragraphe, nous renvoyons à [9].

5.3 Formules de quadrature

L'intégrale d'une fonction $\hat{v}$ définie sur l'élément de référence $\hat{T}$ peut être calculée en utilisant une formule de quadrature avec N_q points :

$$\int_{\hat{T}} \hat{v} \approx \sum_{q=1}^{N_q} w_q \, \hat{v}(\xi_q) \tag{30}$$

Cette relation s'appelle formule de quadrature de Gauss ; les points ξ_q sont les points de quadrature et les réels w_q les poids de la formule de quadrature (30). Lorsque pour une fonction $\hat{v}$ particulière, on a égalité dans l'expression (30), on dit que la formule de quadrature est exacte pour $\hat{v}$ (voir [7]).
Soit T un triangle ou un tétraèdre. La fonction $F_T : \hat{T} \to T$ est affine, et nous avons :

$$\int_T v = \int_{\hat{T}} v \circ F_T \; det(B_T) = 2|T| \int_{\hat{T}} \hat{v}$$

Cette intégrale est alors approchée par :

$$\int_T v \approx 2|T| \sum_{q=1}^{N_q} w_q \, \hat{v}(\xi_q)$$

De même, si l'intégrale implique le gradient de v et z, nous avons :

$$\int_T \nabla v \cdot \nabla z \approx 2|T| \sum_{q=1}^{N_q} w_q \, (B_T^t)^{-1} \hat{\nabla} \hat{v}(\xi_q) \cdot (B_T^t)^{-1} \hat{\nabla} \hat{z}(\xi_q)$$

5.4 Calcul de matrices locales et second membre du système

On revient sur le problème modèle présenté dans la section 3.6 et on est intéresse au calcul numérique du problème II.
Il existe deux types de matrices locales en fonction du domaine d'intégration. Premièrement, nous calculons la matrice M_T résultante de l'intégrale de volume sur un élément fixe T. Nous rappelons (voir section 5.2) que les fonctions de base locales $\phi_{i,T}$ sont obtenus à partir du changement des fonctions monôme $\hat{\phi}_i$ de l'élément $\hat{T}$ de référence sur l'élément T :

$$\phi_{i,T} = \hat{\phi}_i \circ F_T^{-1}$$

Ensuite, nous avons :

$$1 \leq i, j \leq N_{loc}, \quad (M_T)_{i,j} = \int_T \nabla \phi_{i,T} \cdot \nabla \phi_{j,T}$$

En utilisant le changement de variable, on peut calculer l'intégrale sur l'élément de référence :

$$(M_T)_{i,j} = 2|T| \int_{\hat{T}} (B_T^t)^{-1} \hat{\nabla} \hat{\phi}_{i,T} \cdot (B_T^t)^{-1} \hat{\nabla} \hat{\phi}_{j,T}$$

Les contributions volumiques du second membre local sont :

$$(L_T)_i = \int_T f \phi_{i,T} = 2|T| \int_{\hat{T}} f \hat{\phi}_i$$

Nous allons maintenant calculer les matrices locales correspondant à des intégrales sur une face fixe F. Si F est une face intérieure, notons T_1 et T_2 les éléments qui partagent cette face. On rappelle ci-dessous les termes impliquant des intégrales sur F :

$$A_F = -\int_F \{\nabla w \cdot n_F\} [\![v]\!] - \int_F \{\nabla v \cdot n_F\} [\![w]\!] + \frac{\eta}{h_F} \int_F [\![w]\!][\![v]\!]$$

En désignant par w_i et v_i les restrictions du w et de v à l'élément T_i et en utilisant la définition des moyennes et des sauts, nous avons :

$$A_F = a_F^{11} + a_F^{12} + a_F^{21} + a_F^{22}$$

où

$$a_F^{11} = -\frac{1}{2} \int_F \nabla w_1 \cdot n_F \, v_1 - \frac{1}{2} \int_F \nabla v_1 \cdot n_F \, w_1 + \frac{\eta}{h_F} \int_F w_1 \, v_1$$

$$a_F^{22} = \frac{1}{2} \int_F \nabla w_2 \cdot n_F \, v_2 + \frac{1}{2} \int_F \nabla v_2 \cdot n_F \, w_2 + \frac{\eta}{h_F} \int_F w_2 \, v_2$$

$$a_F^{12} = \frac{1}{2} \int_F \nabla w_1 \cdot n_F \, v_2 - \frac{1}{2} \int_F \nabla v_2 \cdot n_F \, w_1 - \frac{\eta}{h_F} \int_F w_1 \, v_2$$

$$a_F^{21} = -\frac{1}{2} \int_F \nabla w_2 \cdot n_F \, v_1 + \frac{1}{2} \int_F \nabla v_1 \cdot n_F \, w_2 - \frac{\eta}{h_F} \int_F w_2 \, v_1$$

De ces quatre termes vont résulter quatre matrices de taille $N_{loc} \times N_{loc}$ définies ci-dessous :

$$(A_F^{11})_{i,j} = -\frac{1}{2} \int_F \nabla \phi_{i,T_1} \cdot n_F \, \phi_{j,T_1} - \frac{1}{2} \int_F \nabla \phi_{j,T_1} \cdot n_F \, \phi_{i,T_1} + \frac{\eta}{h_F} \int_F \phi_{i,T_1} \, \phi_{j,T_1}$$

$$(A_F^{22})_{i,j} = \frac{1}{2} \int_F \nabla \phi_{i,T_2} \cdot n_F \, \phi_{j,T_2} + \frac{1}{2} \int_F \nabla \phi_{j,T_2} \cdot n_F \, \phi_{i,T_2} + \frac{\eta}{h_F} \int_F \phi_{i,T_2} \, \phi_{j,T_2}$$

$$(A_F^{12})_{i,j} = \frac{1}{2} \int_F \nabla \phi_{i,T_1} \cdot n_F \, \phi_{j,T_2} - \frac{1}{2} \int_F \nabla \phi_{j,T_2} \cdot n_F \, \phi_{i,T_1} - \frac{\eta}{h_F} \int_F \phi_{i,T_1} \, \phi_{j,T_2}$$

$$(A_F^{21})_{i,j} = -\frac{1}{2} \int_F \nabla \phi_{i,T_2} \cdot n_F \, \phi_{j,T_1} + \frac{1}{2} \int_F \nabla \phi_{j,T_1} \cdot n_F \, \phi_{i,T_2} - \frac{\eta}{h_F} \int_F \phi_{i,T_2} \, \phi_{j,T_1}$$

Ensuite, si F est une face de bord, nous désignerons également par T_1 l'élément auquel il appartient, la matrice locale $(A_F^{11})_{i,j}$ est créé :

$$(A_F^{11})_{i,j} = -\int_F \nabla \phi_{i,T_1} \cdot n_F \, \phi_{j,T_1} - \int_F \nabla \phi_{j,T_1} \cdot n_F \, \phi_{i,T_1} + \frac{\eta}{h_F} \int_F \phi_{i,T_1} \, \phi_{j,T_1}$$

Comme précédemment, toutes les intégrales sur l'élément physique sont transformées en intégrales sur l'élément de référence dans l'espace $\mathbb{R}^{d-1}$.

6 libMesh

6.1 Introduction

libMesh (voir [1]) est une bibliothèque C++ open-source qui fournit un cadre pour la **résolution** numérique des **équations aux dérivées partielles**, sur des maillages non structurées en 1d, 2d et 3d. Un des principaux objectifs de la bibliothèque est de fournir un appui pour le **raffinement adaptatif** (h, p et hp raffinement) de maillage (voir section 9.1) et le **calcul parallèle**.
La bibliothèque libMesh a été activement mis au point à l'Université du Texas à Austin dans le CFDLab depuis Mars 2002. Les principales contributions proviennent de développeurs à l'Institut Technische Universität Hamburg-Harburg de Modélisation et calcul. Des contributions ont été apporteés récemment par les diplômés CFDLab au Sandia National Laboratories et la NASA Lyndon B. Johnson Space Center.
La bibliothèque écrite en C++, utilise la programmation orientée objet qui a permit aux développeurs d'écrire leur code autour de classe abstraite regroupant les outils tels que : structure des données - maillage, fonction de base et leurs dérivées, passage entre l'élément de référence et l'élément physique, formule de quadrature, assemblage de matrice, etc...
La bibliothèque offre un certain nombre de "familles" d'éléments finis qui peuvent être utilisé dans une simulation. Les espaces des éléments finis classique de Lagrange et Hermite sont supportés. Les espaces d'éléments finis discontinus sont également proposés ; la bibliothèque offre la base des éléments finis monôme pour ces espaces.
La bibliothèque génère seulement des maillages réguliers sur un segment, un rectangle ou un pavé droit. libMesh prend en charge la lecture et l'écriture d'un certain nombre de formats de maillage non structuré, y compris le format UCD, VNU, GMSH, TetGen, Tecplot, GMV, etc.
La bibliothèque utilise des librairies existantes : PETSc est utilisé pour la résolution de systèmes linéaires sur les deux plate-formes en série et en parallèle, et LASPack est inclu avec la bibliothèque pour fournir un solveur linéaire sur les machines en série. La bibliothèque utilise à la fois METIS et ParMETIS pour la décomposition de domaine, voir Figure 9 pour les dépendances entre les principales bibliothèques utilisées par libMesh.

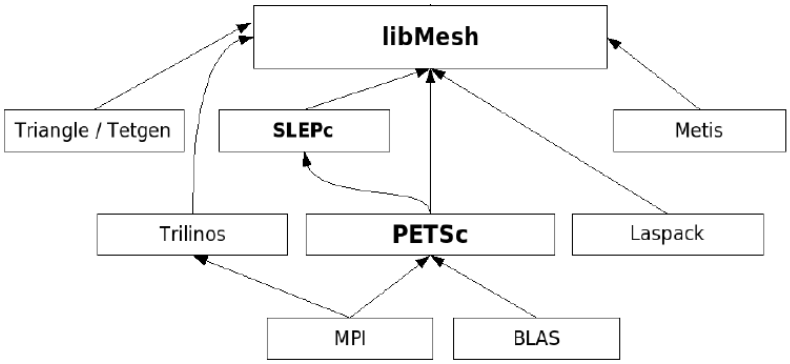


FIGURE 9 – Les dépendances entre les principales bibliothèques utilisées par libMesh

Les principales classes dans libMesh sont :
- Mesh : fournit une description discrète d'un objet dans l'espace d-dimensionnel, où d est 1, 2, ou 3. La classe **Mesh** est centrale, elle fournit tous les concepts géométriques nécessaires dont nous aurons besoin.
Un maillage est principalement composé d'éléments et de nœuds. Ces entités ont évidemment leurs propres classes :
- Elem : définit l'interface pour un élément géométrique. C'est une classe importante qui encapsule toutes les notions abstraites qui sont liées au concept "élément" (comme le nombre de nœuds, son volume, ses voisins, etc.). Pour chaque élément spécifique il y a des classes dérivées qui ajoutent des fonctionnalités. L'implémentation de tous les types d'éléments géométriques utilisés dans l'analyse par éléments finis, y compris les quadrilatères, triangles, hexaèdres, tétraèdres, prismes et les pyramides, ainsi qu'une collection d'éléments infinis, est fournie dans libMesh (voir Figure 10 et Figure 11 pour la diagramme d'héritage de la classe Elem).
- Node : chaque objet de la classe Node stocke la localisation dans l'espace des sommets : (x, y, z), ainsi que des informations d'état supplémentaires, y compris un numéro d'identification global unique (ID).
Remarque : Nous pouvons voir la classe Mesh comme un conteneur qui peut contenir des objets de type Elem et Node.
- BoundaryInfo : est la classe qui gère les informations liées à la frontière, y compris les conditions de bords. Nous utilisons cette information pour appliquer les bonnes conditions aux limites.
La classe Mesh a un objet de type BoundaryInfo comme un attribut. Lorsque le maillage est généré (ou chargées à partir d'un fichier) toutes les informations liées à la frontière sont crées et stockés dans cet objet BoundaryInfo.
- FEBase : fournit l'interface générique pour toutes les familles d'éléments finis tels que Lagrange, Hermite ou Galerkin discontinu. La classe FEBase fournit des données essentielles pour les routines d'assemblage de matrices telles que les valeurs des fonctions de base et leurs gradients, la jacobienne, l'emplacement des points de quadrature dans l'espace physique, etc.
- DofObject : gère les différents types de degrés de liberté de façon générique.
- System : correspond à un système d'EDP d'une ou plusieurs équations qui doit être résolu sur un maillage donné. libMesh fournit plusieurs implémentations concrètes du système pour différents types de problèmes (linéaire, non linéaire, des problèmes d'évolution, des problèmes de valeurs propres, etc.). Un objet du système contient des structures de données pertinentes pour le problème, comme une matrice creuse.
Autre classes : SparseMatrix, NumericVector, LinearSolver, etc...

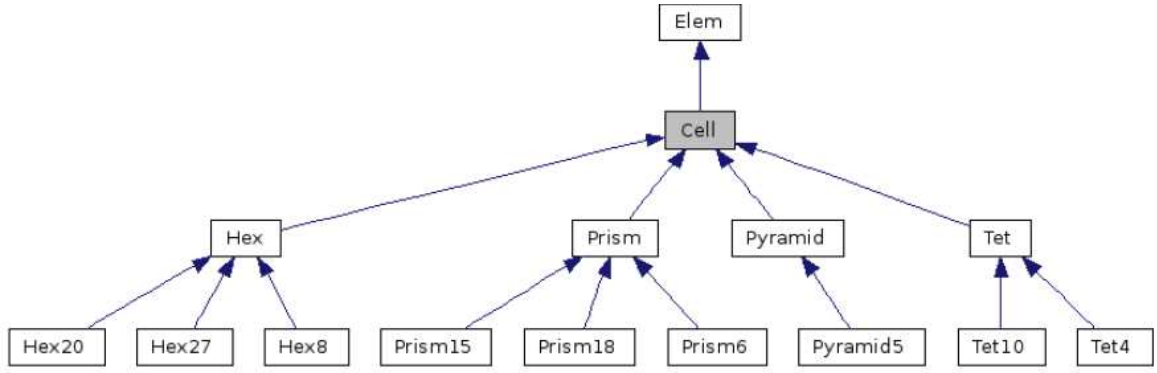


FIGURE 10 – Diagramme d'héritage de la classe Elem (la branche Cell)

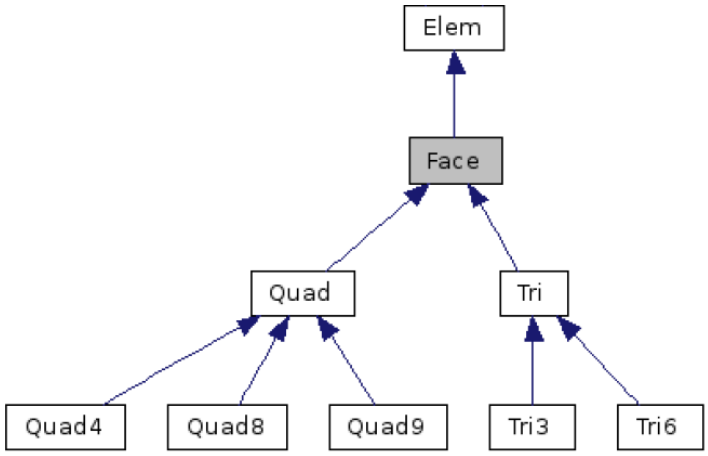


FIGURE 11 – Diagramme d'héritage de la classe Elem (la branche Face)

libMesh constitue donc un outil pour l'implémentation des différentes méthodes d'éléments finis, en particulier Galerkin discontinu, et c'est l'outil qui a été retenu pour le développement et l'implémentation de notre méthode numérique. Signalons que l'intégration de la méthode GD dans la librairie libMesh est plutôt récente, et que de fait les outils developpés ne sont pas aussi complet que dans le cas des méthodes Galekin Continue (l'outil de visualisation ne permet la lecture qu'à partir du format 'gmv', qui de plus est payant).
Remarque : la difficulté principale dans l'utilisation de libMesh est l'inexistence d'un manuel d'utilisation. Parmis les nombreux exemples illustrant l'utilisation de la librairie, un seul exemple concerne la méthode GD.

6.2 Calcul de matrices locales et second membre

libMesh utilise pour le calcul de l'intégrale de volume sur un élément fixe T le passage sur l'élément référence (voir section 5.4) et les formules de quadrature à N_q points ξ_q associés aux poids w_q (voir section 5.3) :

$$(M_T)_{i,j} = \int_T \nabla\phi_i \cdot \nabla\phi_j = \int_{\hat{T}} \hat{\nabla}_\xi\phi_i \cdot \hat{\nabla}_\xi\phi_j \, |J| \, d\xi \approx \sum_{q=1}^{N_q} \hat{\nabla}_\xi\phi_i(\xi_q) \cdot \hat{\nabla}_\xi\phi_j(\xi_q) \, |J(\xi_q)| \, w_q$$

et

$$(L_T)_i = \int_T f \, \phi_i = \int_{\hat{T}} f \, \phi_i \, |J| \, d\xi \approx \sum_{q=1}^{N_q} f(x(\xi_q))\phi_i(\xi_q) \, |J(\xi_q)| \, w_q$$

LibMesh fournit les variables suivantes à chaque point de quadrature q :

Code	Math	Description
`JxW[q]`	$\|J(\xi_q)\|w_q$	la Jacobienne fois le poids
`phi[i][q]`	$\phi_i(\xi_q)$	valeur de la $i^{\text{ème}}$ fonction base
`dphi[i][q]`	$\hat{\nabla}_\xi\phi_j(\xi_q)$	valeur du gradient du $i^{\text{ème}}$ fonction base
`xyz[q]`	$x(\xi_q)$	localisation de ξ_q dans l'espace physique

L'assemblage sous LibMesh de M_T et L_T est similaire à l'énoncé mathématique :

Algorithm 1 Assemblage M_T et L_T

```
for (q=0; q<Nq; q++)
  {
    for (i=0; i<Nloc; i++)
    {
      for (j=0; j<Nloc; j++)
      {
         M_T(i,j) += JxW[q]*(dphi[i][q]*dphi[j][q]);
      }
      L_T(i) += JxW[q] * f(xyz[q]) * phi[i][q];
    }
  }
```

L'assemblage des matrices $A_F^{11}, A_F^{22}, A_F^{12}$ et A_F^{21} correspondant aux intégrales sur les faces est similaire à celui pour les matrice M_T et L_T :

Algorithm 2 Assemblage $A_F^{11}, A_F^{22}, A_F^{12}$ et A_F^{21}

```
for (q=0; q<Nq; q++){
    for (i=0; i<Nloc; i++){
        for (j=0; j<Nloc; j++){
            A_11(i,j) -= 0.5 * JxW_face[q] *
                        (phi_face[j][q]*(qface_normals[q]*dphi_face[i][q]) +
                         phi_face[i][q]*(qface_normals[q]*dphi_face[j][q]));
            // consistence
            A_11(i,j) += JxW_face[qp] * penalty/h *
                         phi_face[j][qp]*phi_face[i][qp];
            // stabilité
        }
    }
    for (i=0; i<Nloc; i++){
        for (j=0; j<Nloc; j++){
            A_22(i,j) += 0.5 * JxW_face[q] *
                        (phi_neighbor_face[j][q]*
                        (qface_normals[q]*dphi_neighbor_face[i][q]) +
                         phi_neighbor_face[i][q]*
                        (qface_normals[q]*dphi_neighbor_face[j][q]));
            // consistence
            A_22(i,j) += JxW_face[q] * penalty/h *
                         phi_neighbor_face[j][q]*phi_neighbor_face[i][q];
            // stabilité
             }
           }

      for (i=0; i<Nloc; i++){
        for (j=0; j<Nloc; j++){
            A_21(i,j) += 0.5 * JxW_face[q] *
                        (phi_neighbor_face[i][q]*(qface_normals[q]*dphi_face[j][q]) -
                         phi_face[j][q]*(qface_normals[q]*dphi_neighbor_face[i][q]));
             // consistence
            A_21(i,j) -= JxW_face[q] * penalty/h *
                         phi_face[j][q]*phi_neighbor_face[i][q];
            // stabilité
             }
           }

      for (i=0; i<Nloc; i++){
        for (j=0; j<Nloc; j++){
            A_12(i,j) += 0.5 * JxW_face[q] *(phi_neighbor_face[j][q]*
                        (qface_normals[q]*dphi_face[i][q]) -
                         phi_face[i][q]*(qface_normals[q]*dphi_neighbor_face[j][q]));
            // consistence
            A_12(i,j) -= JxW_face[q] * penalty/h *
                        phi_face[i][q]*phi_neighbor_face[j][q];
            // stabilité
             }
           }
     }
```

Soit $u(r,\theta) = r^{\frac{2}{3}}\sin(\frac{2}{3}\theta)$ (en coordonnées polaires) la solution exacte du problème de Poisson (21). Ici f est identiquement nulle. Sur la Figure 13 est représentée u dans le L-domaine et sur la Figure 12 est tracée la solution numérique obtenu en utilisant le schéma SIPG.

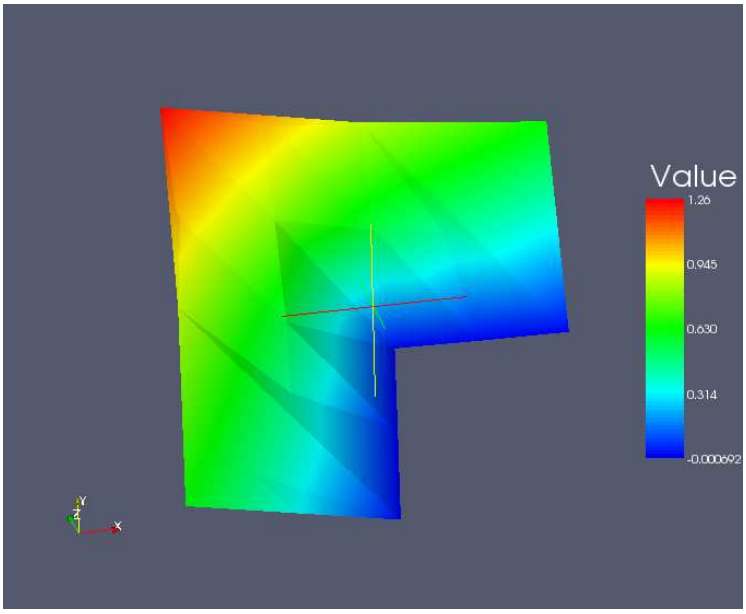


FIGURE 12 – Solution SIPG

SIPG $(\varepsilon = -1)$ et $\eta = 10$
Number of elements : 30
System has : 240 degrees of freedom
L2-Error is : 0.00353314

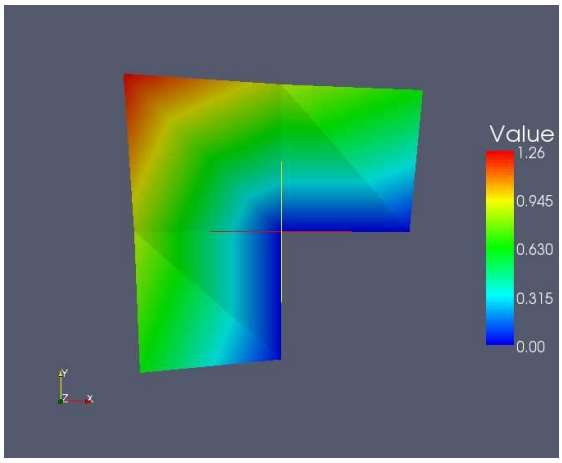


FIGURE 13 – Solution exacte

7 Résultats numériques

Afin de valider les résultats numériques dans le cas de notre système (19) dont l'inconnu est un vecteur à $N+1$ dimension, nous avons d'abord traité le cas $N=0$ (section 7.1 et 7.2), en comparant le résultat obtenu dans le cadre de notre méthode, avec un code de calcul existant à l'IRSN qui utilise la méthode des éléments finis Lagrange $P1$ (EF). Nous avons par la suite considéré et validé le cas $N=1$ sur un cas test simple (section 7.3). Enfin, nous avons mis en place un cas test proposée par MoMaS (section 7.4).

7.1 Cas avec un composant

Afin de valider le code de calculs implémenté dans le cas $N=0$ du problème d'évolution (19), nous avons considéré le cas test suivant : Soit le domaine rectangulaire (Figure 14) avec quatre types de frontières : Γ_1, Γ_2 ,Γ_3 et Γ_4.

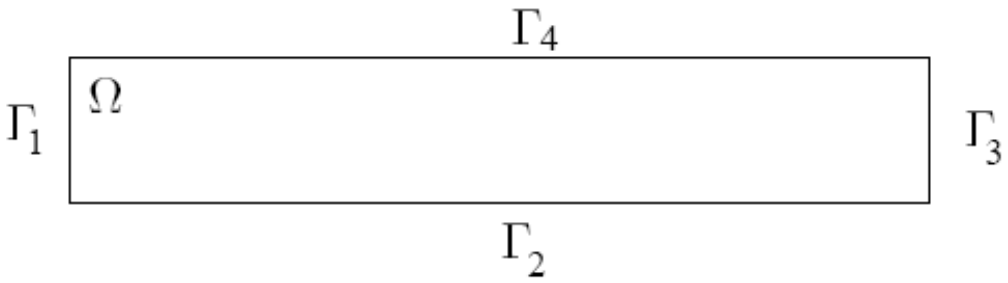


FIGURE 14 – Domaine

On impose les conditions aux limites suivantes :

- des conditions de Dirichlet sur le bord Γ_1
  $\Gamma_1 : P_l = 10^6\, Pa$

- une injection d'eau pur sur la frontière Γ_2
  $\Gamma_2 : \phi^w \cdot \nu = 5.57\, 10^{-5}\, kg/m^2/ans$

- des valeurs négatives de flux sur la frontière Γ_3
  $\Gamma_3 : \phi^w \cdot \nu = -3\, 10^{-1}\, kg/m^2/ans$

- des valeurs négatives de flux sur la frontière Γ_4
  $\Gamma_4 : \phi^w \cdot \nu = -9\, 10^{-1}\, kg/m^2/ans$
  ν désigne la normale sortante du domaine

Avec une condition initiale vérifiant sur tout les domaine : $P_l^0 = 10^6$
Le terme source volumique F^w est fixé à zéro sur tout le domaine. Le milieu poreux considéré est isotrope homogène, le tenseur de perméabilité absolue est de la forme $\mathbb{K} = k\,\mathbb{I}$, k scalaire. Le Tableau 1 synthétise les valeurs des paramètres décrivant le milieu poreux et les caractéristiques

du fluide. Ce cas test a été inspiré par un cas test proposée par MoMaS, voir section 7.4 pour la descriptions de l'exercice original.

Paramètres	Symbole	Valeur
Milieu poreux	$\mathbf{k}$	$\mathbf{5\ 10^{-20}}\ m^2$
	$\mathbf{\Phi}$	$\mathbf{0.15}$
	P_r	$1,6\ 10^6\ Pa$
	n	1.49
	S_{lr}	0.4
	S_{gr}	0
Caractéristiques des fluides	D_l^h	$3\ 10^{-9}\ m^2/s$
	$\mu_\mathbf{l}$	$\mathbf{1\ 10^{-3}}\ Pa.s$
	μ_g	$9\ 10^{-6}\ Pa.s$
	H	$7,65\ 10^{-6}\ mol/Pa/m^3$
	$\mathbf{M_l}$	$\mathbf{10^{-2}}\ kg/mol$
	M_g	$2\ 10^{-3}\ kg/mol$
	ρ_l^{std}	$10^3\ kg/m^3$
	ρ_g^{std}	$8\ 10^{-2}\ kg/m^3$

TABLE 1 – Paramètres : milieux poreux + caractéristiques des fluides. En gras les paramètres pertinent pour le cas $N = 0$

Nous avons lancé des simulations en prenant une durée d'observation sur 3000 ans avec une température fixée à $T = 303K$. Les courbes obtenues (Figure 15 et 16 pour le cas **1d et IIPG**, puis Figure 17 et 18 pour le cas **2d et SIPG** et enfin Figure 19 et 20 pour le cas **3d et NIPG**) permettent une comparaison des résultats dans le cas GD avec ceux fournis par le code de calcul de l'IRSN utilisant la méthode EF. Nous obtenu les même résultats, ce qui valide à la fois la partie du code consacrée à la discrétisation en espace mais aussi celle consacrée à la discrétisation en temps.

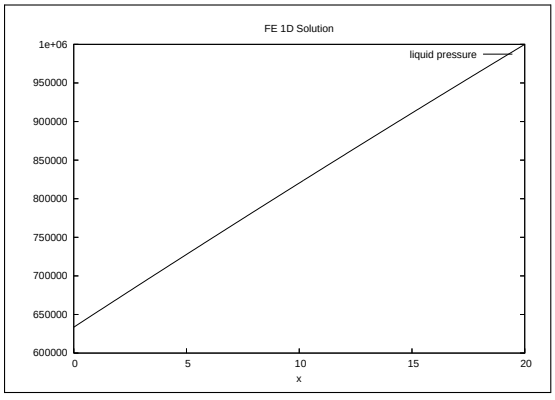


FIGURE 15 – Solution EF 1d

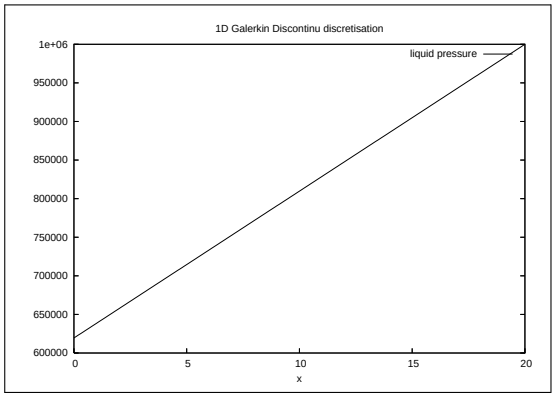


FIGURE 16 – Solution GD 1d IIPG ($\varepsilon = 0$) et $\eta = 1e - 6$

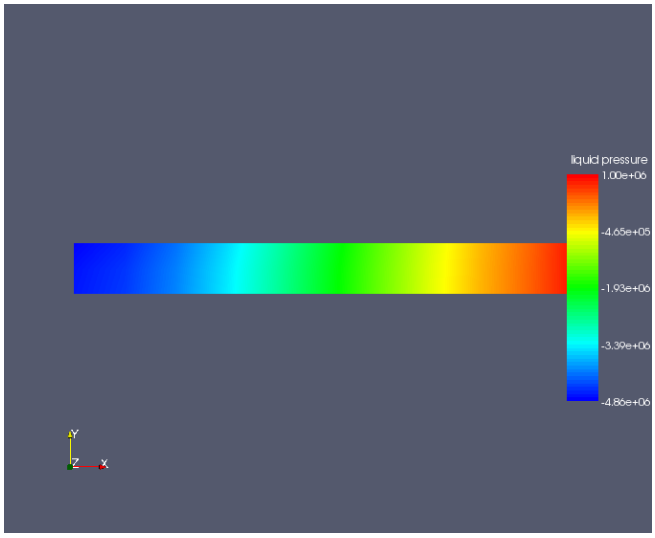


FIGURE 17 – Solution EF 2d

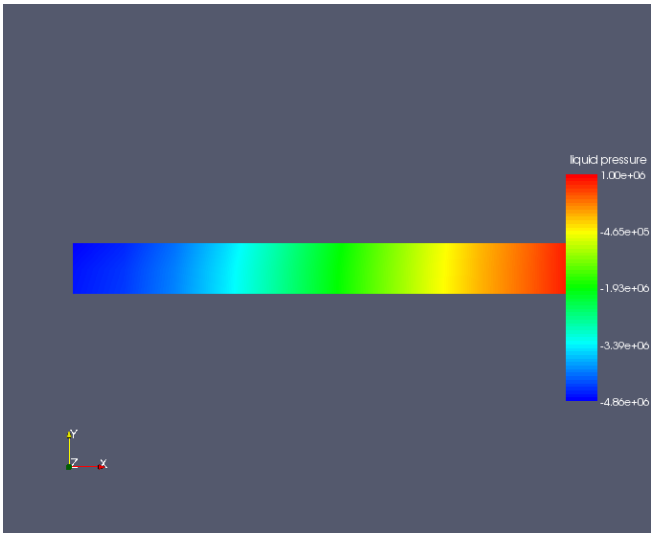


FIGURE 18 – Solution GD 2d SIPG $(\varepsilon = -1)$ et $\eta = 1e - 6$

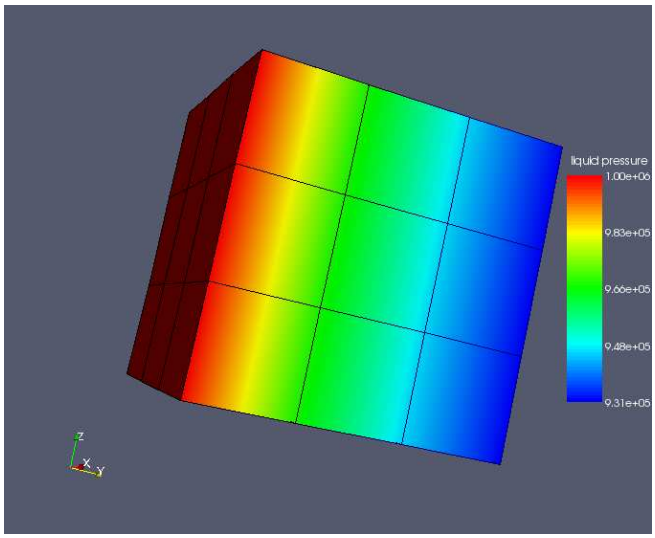


FIGURE 19 – Solution EF 3d

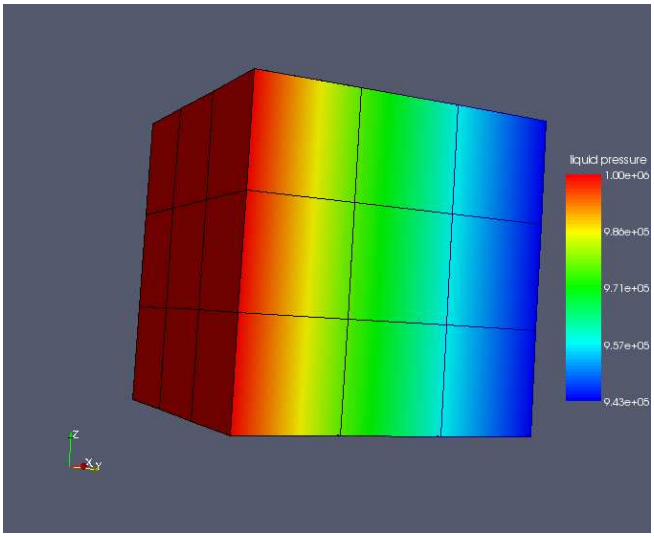


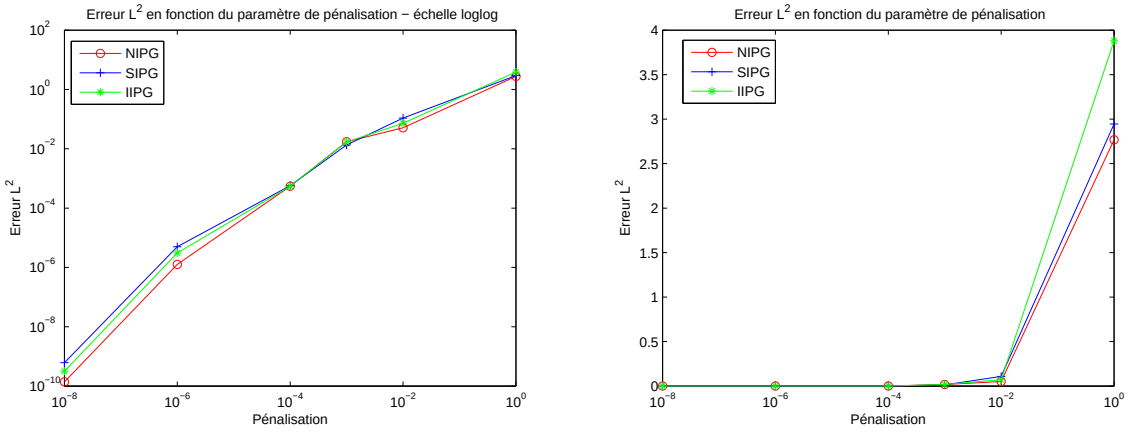
FIGURE 20 – Solution GD 3d NIPG $(\varepsilon = +1)$ et $\eta = 1e - 5$

7.2 L'effet de la valeur de pénalisation

Etudions maintenant l'effet du paramètre de pénalisation η sur la solution obtenue.
Soit un maillage fixé, et des fonctions linéaires par morceaux (polynômes par morceaux de degré un) comme fonctions de base. Nous avons fait varier le paramètre de pénalisation η entre 10^{-8} et 1.
La Figure 21 montre l'évolution de la norme L^2 de l'erreur pour NIPG, SIPG et IIPG en fonction du paramètre de pénalisation :

- si $\eta < 10^{-3}$, les trois méthodes donnent des solutions numériques correctes.

- si on teste des valeurs de plus en plus grandes de η, l'erreur numérique augmente. La méthode ne converge pas si la valeur de pénalisation est $\eta > 10^{-1}$. Ceci s'explique par le fait que les termes de saut dominent les termes de flux calculés sur les bords.



(a) (b)

FIGURE 21 – Les variations de la norme L^2 de l'erreur numérique par rapport à la valeur de la pénalisation pour NIPG (ligne rouge), SIPG (ligne bleu) et IIPG (ligne verte). (a) échelle logarithmique (b) échelle usuelle.

Remarque : La sensibilité du schéma par rapport au terme de pénalisation a été également mise en évidence dans le cas de l'équation de Navier-Stokes (voir [9] pages 140-141, Figure 7.2).

7.3 Cas avec deux composants

On considère un domaine rectangulaire (Figure 22) où l'on distingue 3 types de frontières : Γ_1 - bord entrant, Γ_2 - bord sortant et Γ_3 - bord imperméable.

On impose les conditions aux limites suivantes :

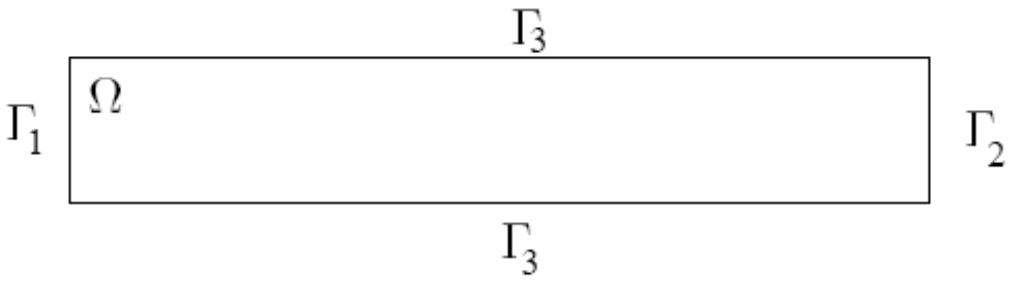
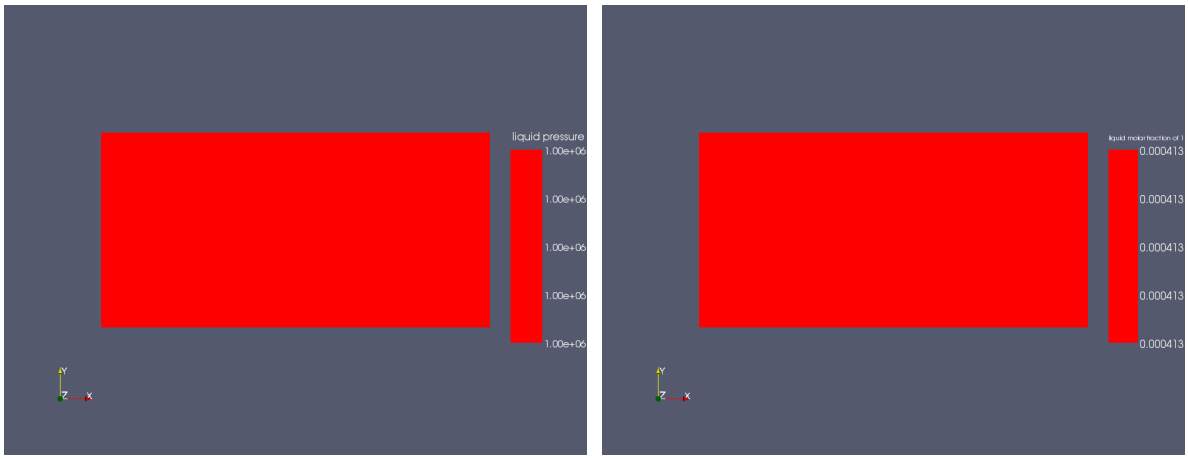


FIGURE 22 – Domaine

- des conditions de Dirichlet sur le bord entrant
  Γ_1 et Γ_2 : $P_l = 10^6$ Pa et $x_l^1 = 41.3\ 10^{-5}$

- des conditions de flux nuls sur la frontière imperméable
  Γ_3 : $\phi^w \cdot \nu = 0$ et $\phi^h \cdot \nu = 0$
  ν désigne la normale sortante du domaine.

Les termes sources volumiques F^w et F^h sont fixés à zéros sur tout le domaine.
Ce cas test a été traité en stationnaire. Nous avons considéré les mêmes paramètres physiques que ceux présentées dans le cas avec un seul composant (Tableau 1).
La difficulté principale de cette partie a été le choix de la valeur de η qui apparaît dans le schéma numérique à résoudre. Or nous nous sommes rendu compte qu'on devait associer à chaque équation un paramètre de pénalisation différent. Par exemple le choix de $\eta = (\eta_{P_l}, \eta_{x_l^1}) = (10^{-13}, 10^{-19})$ fournit une *solution satisfaisante* (voir Figure 23). Nous avons testé plusieurs valeurs de η avec $\eta_{P_l} = \eta_{x_l^1}$, mais aucune n'a fournit la solution numérique escomptée. Ceci nous a conduit à penser que ce paramètre devait dépendre non seulement de la géométrie du maillage et de la constante de trace, mais également des opérateurs à discrétiser, et de "la physique du problème" (par exemple les fonctions $A_{i,k}$ et $C_{i,k}$). De fait, résoudre le système 19 reviendrai à trouver une suite optimale $(\eta_i)_{i\in\{0,N\}}$ de paramètres de pénalisation. Au départ, nous n'avons pas prévu cette difficulté et pas eu le temps de trouver des réponses théoriques à ce choix optimal du paramètre. Nous avons simplement constaté de façon empirique que le couple de valeurs choisies convient.



(a) P_l (b) x_l^1

FIGURE 23 – IIPG $\eta_{P_l} = 10^{-13}$ et $\eta_{x_l^1} = 10^{-19}$

7.4 MoMaS

Cet exercice a été proposé au sein du groupe national de recherche MOMAS (Modélisations Mathématiques et Simulations Numériques liées aux problèmes de gestion des déchets nucléaires), résultat de la collaboration de diverses entreprises et laboratoires de recherche dont l'IRSN fait partie. Pour plus d'informations sur les activités du GNR MOMAS, on pourra consulter le site internet `http://www.gdrmomas.org`.
On considère un domaine rectangulaire (Figure 24) où l'on distingue 3 types de frontières : Γ_{in} - bord entrant, Γ_{out} - bord sortant et Γ_{imp} - bord imperméable.

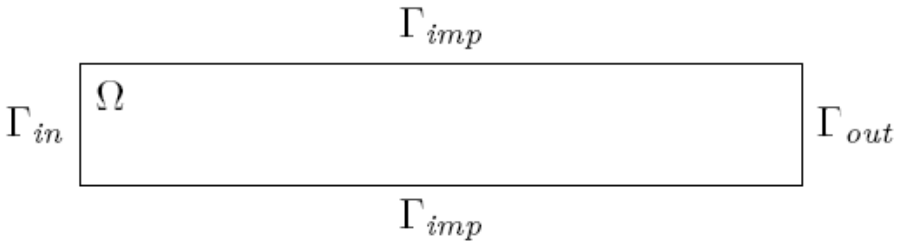


FIGURE 24 – Domaine

On impose les conditions aux limites suivantes :

- des conditions de Dirichlet sur le bord sortant Γ_{out} : $P_l = 10^6\ Pa$ et $P_g = 1.1\,10^6\ Pa$
- une injection d'hydrogène pur sur le bord entrant Γ_{in} : $\phi^w \cdot \nu = 0$ et $\phi^h \cdot \nu = 5.57\,10^{-5}\ kg/m^2/ans$
- des conditions de flux nuls sur la frontière imperméable

Γ_{imp} : $\phi^w \cdot \nu = 0$ et $\phi^h \cdot \nu = 0$

ν désigne la normale sortante du domaine.

Les conditions initiales considérées sont de conditions uniformes vérifiant sur tout le domaine :

$$P_l = 10^6, \quad P_g = 1.110^6$$

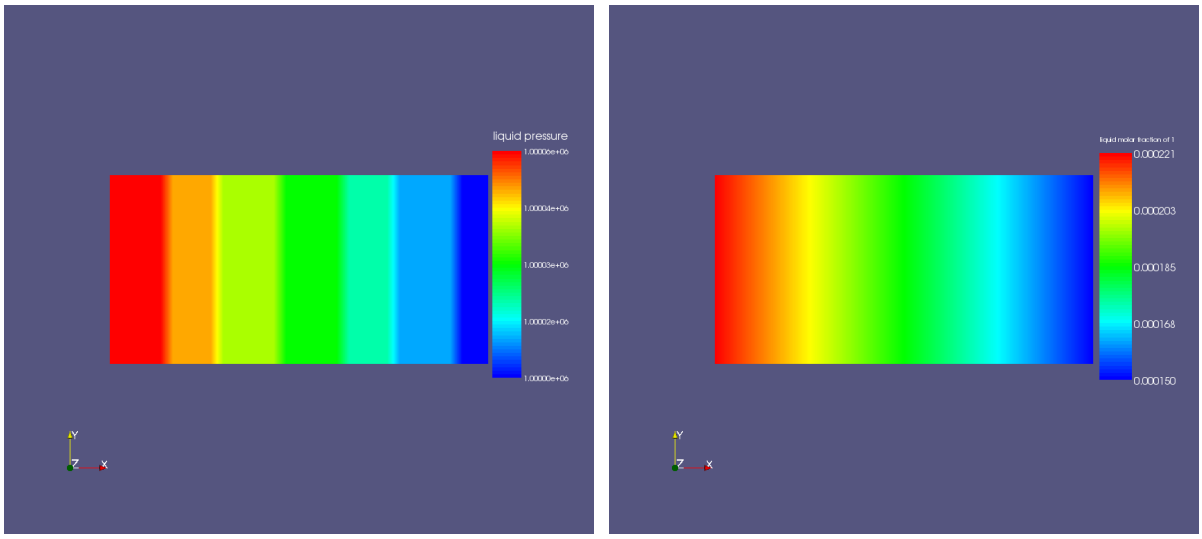
Les termes sources volumiques F^w et F^h sont fixés à zéros sur tout le domaine. Le milieu poreux considèré est isotrope homogène. Le Tableau 2 synthétise les valeurs des paramètres décrivant le milieu poreux et les caractéristiques des fluides. La température est fixée à $T = 303K$. On peut trouver la définition complète de cet exercice à l'adresse :
`http://momas.univ-lyon1.fr/cas_test/sature_2.pdf`
ainsi que les résultats de la simulation à l'adresse :
`http://momas.univ-lyon1.fr/cas_test/resultats/sature_2.pdf`

Paramètres	Symbole	Valeur
Milieu poreux	k	$5\ 10^{-20}\, m^2$
	Φ	0.15
	P_r	$2\ 10^6\, Pa$
	n	1.49
	S_{lr}	0.4
	S_{gr}	0
Caractéristiques des fluides	D_l^h	$3\ 10^{-9}\, m^2/s$
	μ_l	$1\ 10^{-3}\, Pa.s$
	μ_g	$9\ 10^{-6}\, Pa.s$
	H	$7,65\ 10^{-6}\, mol/Pa/m^3$
	M_l	$10^{-2}\, kg/mol$
	M_g	$2\ 10^{-3}\, kg/mol$
	ρ_l^{std}	$10^3\, kg/m^3$
	ρ_g^{std}	$8\ 10^{-2}\, kg/m^3$

TABLE 2 – Paramètres : milieux poreux + caractéristiques des fluides MoMaS

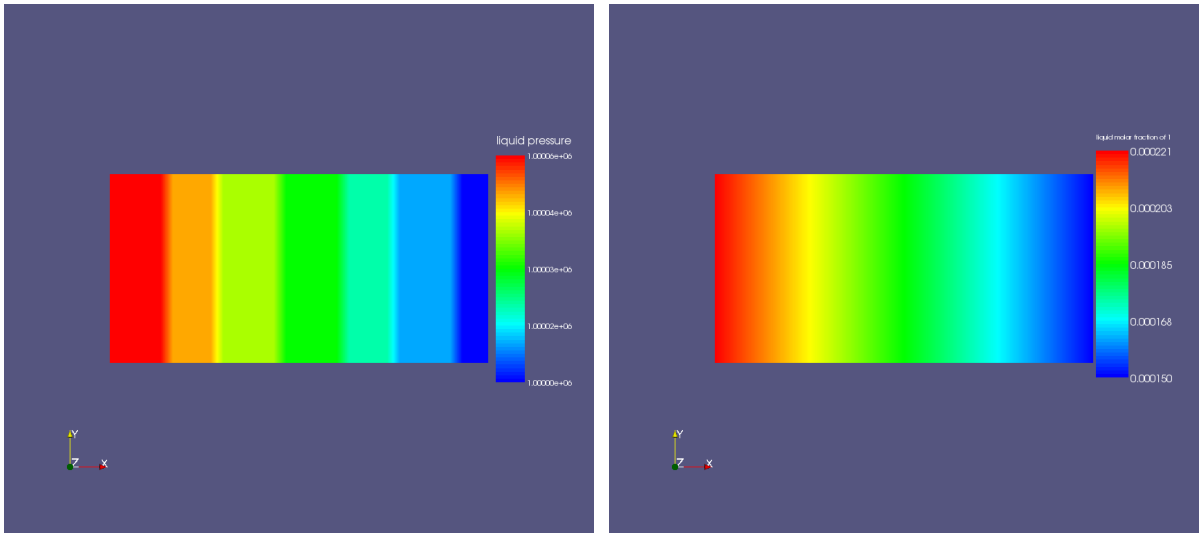
Nous avons commencé par considérer *le cas stationnaire*. Nous avons tracé les graphiques Figure 25 et 26, qui montrent que le jeu de paramètres de pénalisation $\eta_{P_l} = 10^{-10}$ et $\eta_{x_l^1} = 10^{-16}$ permettent d'avoir la solution numérique attendu.

Ce cas test a échoué dans *le cas instationnaire* du fait que nous n'avons pas pu trouver de façon empirique, à chaque pas de temps, les couples de paramètres de pénalisation assurant la stabilité du schéma numérique. Signalons que dans ce cas test, les termes $A_{i,k}$ et $C_{i,k}$ varient fortement *à chaque pas de temps*. Il convient dès lors de définir ces paramètres d'un point de vu théorique à partir de grandeurs propres à l'écoulement telles que la vitesse, la densité, la pression et les grandeurs de la discrétisation numérique, c'est-à-dire le degré polynomiale des fonctions de base, *le pas de temps* ainsi que la géométrie du domaine (constante de trace, inégalité de Poincaré).



(a) P_l (b) x_l^1

FIGURE 25 – IIPG $\eta_{P_l} = 10^{-10}$ et $\eta_{x_l^1} = 10^{-16}$



(a) P_l (b) x_l^1

FIGURE 26 – EF

8 Conclusions et perspectives

Dans le cadre du stage réalisé chez l'IRSN, j'ai pu approfondir mes connaissances en analyse numérique et élargir mes compétences en programmation C++. J'ai eu l'opportunité de mettre en œuvre la récente méthode de résolution des EDP intitulée Galerkin Discontinu. Ce stage a été l'occasion pour moi d'avoir une initiation au travail de recherche, dans lequel je compte orienter ma carrière.

Ce stage avait pour ambition l'application de cette méthode numérique au cas d'un système d'équations modélisant l'écoulement multiphasique en milieux poreux, qui intéresse les chercheurs de l'IRSN. Cela m'a conduit à proposer trois schémas numériques dont il a été question dans ce rapport : NIPG, SIPG et IIPG, et à réaliser une implémentation en C++ utilisant la librairie libMesh.

Les résultats obtenus sont très satisfaisants dans le cas où le système est limité à une seule équation dont l'inconnue est la pression (dans les cas stationnaire/instationnaire). Cela a permis de valider le code implémenté dans le cadre de ces trois méthodes proposées jusqu'à la dimension 3 en espace.

Après avoir étendu le nombre d'équations du système à deux équations, je me suis rendu compte qu'il fallait définir une constante de pénalisations pour chaque inconnu. J'ai pu trouver de façon empirique les valeurs des paramètres de pénalisation correspondant à chaque inconnue, dans tous les cas tests traitant des problèmes stationnaire. Le passage au cas instationnaire nécessite à chaque pas de temps la connaissance de ce type de paramètres difficiles à trouver de façon empirique. De fait, le schéma diverge dans le cas du système instationnaires à deux équations.

Avant d'aller plus loin dans la démarche, je pense qu'il est impératif de définir de façon théorique le jeu de paramètres de pénalisation permettant d'assurer la stabilité du schéma numérique. Signalons que pour les problèmes de diffusion non linéaire, il est nécessaire de mettre en œuvre des limiteurs de pente. Ensuite il convient de définir un meilleur paramètre d'estimateur d'erreur *a posteriori* qui prendrait en compte l'aspect physique du modèle. Enfin, nous remarquons que la méthode GD se prête bien au calcul parallèle, ce qui permettrait d'accélérer la vitesse de résolution du système.

Références

[1] Site du libmesh : http ://libmesh.sourceforge.net/.

[2] Peter BASTIAN. Higher order discontinuous galerkin methods for flow and transport in porous media. *Technical Report*, (33), 2002.

[3] Haim BREZIS. *Analyse Fonctionnelle*. Dunod, 1995.

[4] Pierre TARDIF D'HAMONVILLE. *Modélisation et simulation du transport advectif et diffusif en milieu poreux monophasique et diphasique*. PhD thesis, Ecole Nationale des Ponts et Chaussées, 2006.

[5] A. ERN, I. MOZOLEVSKI, and L. SCHUH. Discontinuous galerkin approximation of two-phase flows in heterogeneous porous media with discontinuous capillary pressures. *Comput. Methods Appl. Mech. Engrg*, 2009.

[6] Alexandre ERN and Jean-Luc GUERMOND. *Theory and Practice of Finite Elements*. Springer, 2004.

[7] Brigitte LUCQUIN and Olivier PIRONNEAU. *Introduction au calcul scientifique*. Masson, 1996.

[8] Pierre-Arnaud RAVIART and Jean-Marie THOMAS. *Introduction à l'Analyse Numérique des Equations aux Dérivées Partielles*. Dunod, 1998.

[9] Béatrice RIVIERE. *Discontinuous Galerkin Methods for Solving Elliptic and Parabolic Equations*. Siam, 2008.

[10] Béatrice RIVIERE and Peter BASTIAN. Discontinuous galerkin methods for two-phase flow in porous media. *Technical Report*, (28), 2004.

[11] Farid SMAI. Développement d'un code de calculs 3d pour modéliser l'écoulement biphasique avec échanges entre les phases dans un milieu poreux. *Rapport du post-doc IRSN*, 2010.

[12] Pierre SOCHALA. *Méthodes numériques pour les écoulements souterrains et couplage avec le ruissellement*. PhD thesis, Ecole Nationale des Ponts et Chaussées, 2008.

9 Annexe

9.1 Comparaison h, p et hp raffinement

Soit $u(r,\theta) = r^{\frac{2}{3}}\sin(\frac{2}{3}\theta)$ la solution exacte du problème (21).
Ici $\|\nabla u\| \underset{r\to 0}{\longrightarrow} \infty$, ce qui signifie qu'au voisinage de l'origine la surface représentée par u est très raîde. Il peut alors être intéressant d'utiliser des maillages qui ne sont pas uniformes, mais localement adaptés au voisinage de cette singularité.
C'est l'objectif des techniques d'estimation *a postériori* qui fournissent des indicateurs d'erreur visant à approcher $|u - u_h|_{H^1(T)}$. On peut alors adapter les éléments dont les indicateurs sont les plus grand afin d'obtenir un nouveau maillage mieux adapté à la solution.
Le maillage optimal est celui qui réalise la précision prescrite par l'utilisateur à moindre coût.
On distingue essentiellement les méthodes d'adaptation de maillage suivantes : la r-adaptation (relocalisation des nœuds), la p-adaptation (augmentation du degré des fonctions de forme) et la h-adaptation (modification de la taille élémentaire) ainsi que les combinaisons de celles-ci, principalement la hp-adaptation.
libMesh fournit des outils pour les h, p et hp- adaptation. libMesh a mis l'accent sur les indicateurs d'erreur locaux qui sont essentiellement indépendants de la physique et fournit également des indicateurs de saut de flux, ceux-ci uniquement dans le cas où il y a une discontinuité de flux à travers le bord des éléments.

9.1.1 h raffinement

La h-adaptation travaille par modification de la taille des éléments. Cette modification peut se faire avec régénération complète du maillage ou par raffinement du maillage initial. Sous libMesh le maillage initial est raffiné.

Number of elements : 6
System has : 18 degrees of freedom
Linear solver converged at step : 6
final residual : 2.2472e-13
L2-Error is : 0.0621953

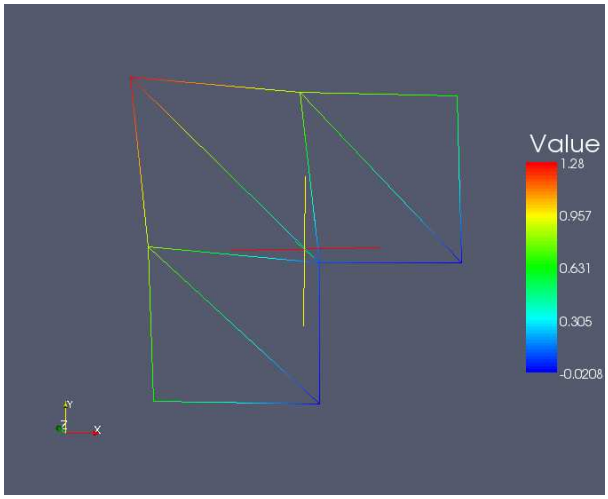


FIGURE 27 – h raffinement

Number of elements : 86
System has : 198 degrees of freedom
Linear solver converged at step : 64
final residual : 4.66553e-12
L2-Error is : 0.0141262

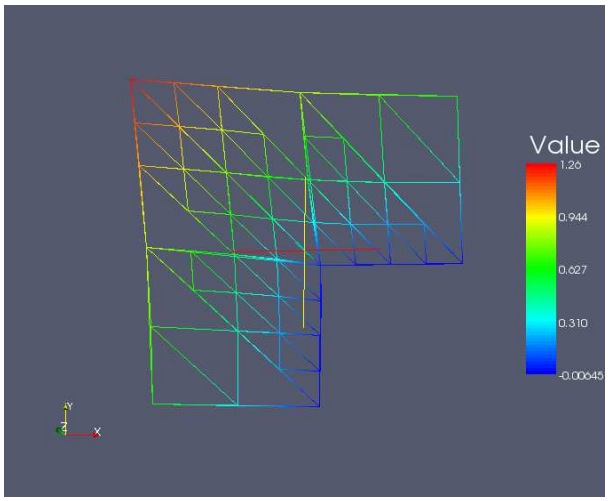


FIGURE 28 – h raffinement

Number of elements : 414
System has : 936 degrees of freedom
Linear solver converged at step : 142
final residual : 6.92728e-12
L2-Error is : 0.00414844

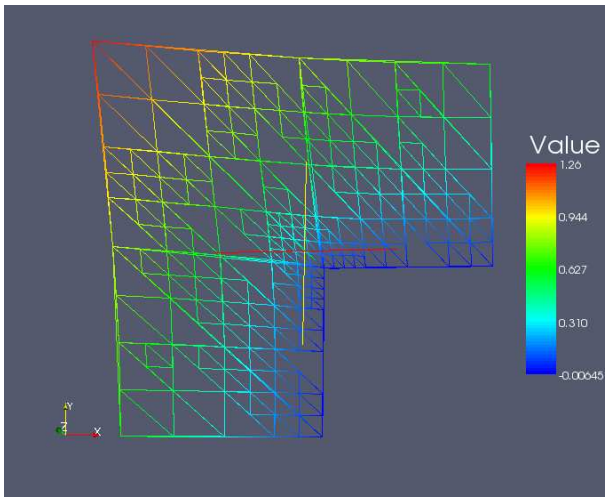


FIGURE 29 – h raffinement

Number of elements : 766
System has : 1728 degrees of freedom
Linear solver converged at step : 229
final residual : 7.39169e-12
L2-Error is : 0.00161582

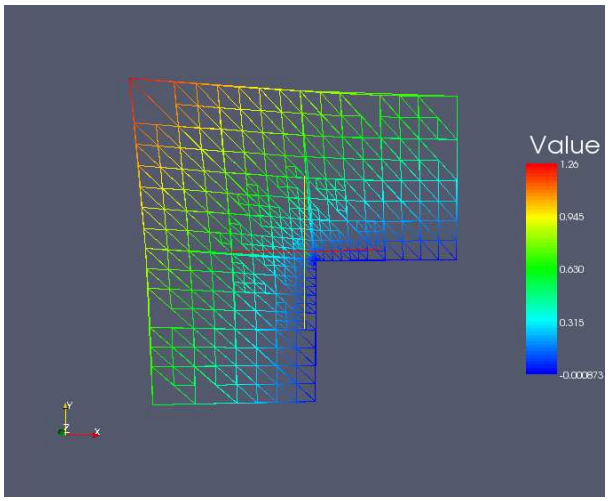


FIGURE 30 – h raffinement

9.1.2 p raffinement

La p-adaptation consiste à enrichir le degré des fonctions d'interpolation.

Number of elements : 6
System has : 52 degrees of freedom
Linear solver converged at step : 9
final residual : 2.56967e-14
L2-Error is : 0.010687

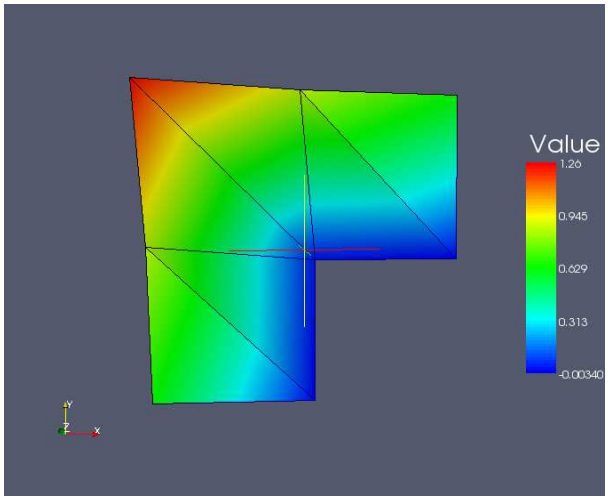


FIGURE 31 – p raffinement

Number of elements : 6
System has : 114 degrees of freedom
Linear solver converged at step : 10
final residual : 2.11391e-11
L2-Error is : 0.00308608

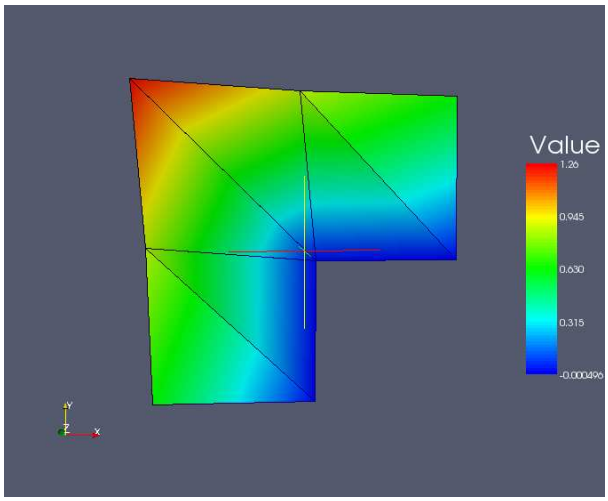


FIGURE 32 – p raffinement

9.1.3 hp raffinement

La hp-adaptation consiste à enrichir le degré des fonctions d'interpolation plus modification de la taille des éléments.

Number of elements : 30
System has : 144 degrees of freedom
Linear solver converged at step : 30
final residual : 8.03423e-13
L2-Error is : 0.00856701

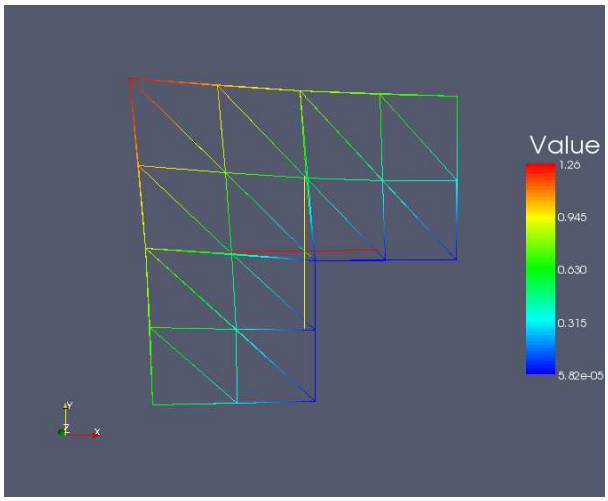


FIGURE 33 – hp raffinement

Il existe une abondante littérature consacrée à l'obtention des estimateurs *a postériori*, plus fiable avec des indicateurs d'erreur qui sont plus étroitement liées aux opérateurs et équations régissant l'application.

9.2 L'assemblage de la matrice et du second membre du système traduit en C++

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4
5  // Libmesh includes
6  #include "mesh.h"
7  #include "mesh_generation.h"
8  #include "equation_systems.h"
9  #include "fe.h"
10 #include "quadrature_gauss.h"
11 #include "sparse_matrix.h"
12 #include "dof_map.h"
13 #include "numeric_vector.h"
14 #include "dense_matrix.h"
15 #include "dense_vector.h"
16 #include "boundary_info.h"
17 #include "elem.h"
18 #include "fe_interface.h"
19
20 // Pour les systèmes d'équations DenseSubMatrix
21 // et DenseSubVector fournir des moyens pratiques pour
22 // l'assemblage de la matrice et seconde membre
23
24 #include "dense_submatrix.h"
25 #include "dense_subvector.h"
26
27 #ifdef LIBMESH_HAVE_PETSC
28 #include <petsc.h>
29 #endif
30
31 #include "mesh_base.h"
32 #include "mesh_output.h"
33 #include "libmesh_logging.h"
34
35 // DiphPoM includes
36
37 #include "assemble_functions.h"
38 #include "math_formulation.h"
39 #include "transient_quasi_linear_implicit_system.h"
40 #include "boundary_condition_set.h"
41 #include "models.h"
42
43 using namespace std;
44
45 void DiphPoM::AssembleFunctions::GD_test
46 (EquationSystems& es, const std::string& system_name)
```

```
47 {
48   // Obtenir certains paramètres dont
49   //nous avons besoin lors de l'assemblage
50
51   Real Temperature =  es.parameters.get<Real>("Temperature");
52   Real epsi =  es.parameters.get<Real>("Epsilon");
53   MathFormulation* mf = &es.parameters.set
54                         <MathFormulation>("MathFormulation");
55   const BoundaryConditionSet& bcs = es.parameters.get
56       <BoundaryConditionSet>("BoundaryConditionSet") ;
57   const Models& models = es.parameters.get<Models>("models");
58   vector<Real> eq_renorm = es.parameters.get
59                            <vector<Real>>("eq_renorm") ;
60   vector<Real> var_renorm = es.parameters.get
61                             <vector<Real>>("var_renorm") ;
62   Real idt = es.parameters.get<Real>("time step inverse") ;
63   RealGradient gravity = es.parameters.get<RealGradient>("gravity");
64   Real current_time = es.parameters.get<Real>("current time") ;
65   vector<Real> eta = es.parameters.get<vector<Real>>("penalty") ;
66
67   // Math-Formulation
68   mf->init(models);
69   EllipticTensorId A ;
70   HyperbolicTensorId C ;
71   ConservativeTermId X ;
72   ConservativeTermJacobianId J ;
73
74   // Obtenir une référence du System que nous résolvons
75   TransientQuasiLinearImplicitSystem& system =
76       es.get_system<TransientQuasiLinearImplicitSystem>(0);
77
78    // Obtenir une référence pour l'objet mesh.
79   const MeshBase& mesh = es.get_mesh();
80
81    // La dimension
82   const unsigned int dim = mesh.mesh_dimension();
83
84   // le nombre de composants
85   const unsigned int Nvar = system.n_vars();
86
87    // Obtenir une référence constante
88    // vers le type des éléments finis
89   FEType fe_type = system.variable_type(0);
90
91   // Construire un objet EF du type spécifié
92   AutoPtr<FEBase> fe  (FEBase::build(dim, fe_type));
93   AutoPtr<FEBase> fe_elem_face(FEBase::build(dim, fe_type));
94   AutoPtr<FEBase> fe_neighbor_face(FEBase::build(dim, fe_type));
95
```

```
96   // Règle de quadrature pour l'intégration numérique.
97
98   #ifdef QORDER
99       QGauss qrule (dim, QORDER);
100  #else
101      QGauss qrule (dim, fe_type.default_quadrature_order());
102  #endif
103      fe->attach_quadrature_rule (&qrule);
104
105  #ifdef QORDER
106      QGauss qface(dim-1, QORDER);
107  #else
108      QGauss qface(dim-1, fe_type.default_quadrature_order());
109  #endif
110
111  // Dire à l'objet FE d'utiliser notre formule de quadrature.
112  fe_elem_face->attach_quadrature_rule(&qface);
113  fe_neighbor_face->attach_quadrature_rule(&qface);
114
115  // Données pour les intégrales de  volume intérieur
116  const std::vector<Real>& JxW = fe->get_JxW();
117  const std::vector<std::vector<Real> >& phi = fe->get_phi();
118  const std::vector<std::vector<RealGradient> >& dphi = fe->get_dphi
       ();
119
120  // Données pour les intégrales de surface sur la frontière
121  const std::vector<std::vector<Real> >&  phi_face
122                                          = fe_elem_face->get_phi();
123  const std::vector<std::vector<RealGradient> >& dphi_face
124                                          = fe_elem_face->get_dphi();
125  const std::vector<Real>& JxW_face = fe_elem_face->get_JxW();
126  const std::vector<Point>& qface_normals = fe_elem_face->
       get_normals();
127  const std::vector<Point>& qface_points = fe_elem_face->get_xyz();
128  const std::vector<Point>& q_points = fe->get_xyz();
129
130  // Données pour les intégrales de surface sur la frontière voisine
131  const std::vector<std::vector<Real> >&  phi_neighbor_face
132                                        = fe_neighbor_face->get_phi();
133  const std::vector<std::vector<RealGradient> >& dphi_neighbor_face
134                                      = fe_neighbor_face->get_dphi();
135
136  // Une référence à l'objet DofMap pour ce système.
137  // Le DofMap objet gère la traduction d'index des numéros
138  // de noeuds et de l'élément vers le numéros de degré de liberté.
139  const DofMap& dof_map = system.get_dof_map();
140
141  // Ce vecteur contiendra les degrés de liberté pour l'élément
142  // Ceux-ci définissent où dans le système global
```

```
143   // l'élément va contribue.
144   vector<unsigned int> dof_indices;
145
146   vector<unsigned int>* dof_indices_var
147                         = new vector<unsigned int>[Nvar];
148
149    // Définir les structures de données pour contenir
150    // la matrice intérieure et le seconde membre.
151    // que nous noterons  "Ke" et "Fe".
152   DenseMatrix<Number> Ke;
153   DenseVector<Number> Fe;
154   vector<DenseSubMatrix<Number>>
155                         _col_K(Nvar, DenseSubMatrix<Number>(Ke));
156
157   vector<vector<DenseSubMatrix<Number>>> subK(Nvar, _col_K);
158   vector<DenseSubVector<Number>>
159                         subF(Nvar, DenseSubVector<Number>(Fe));
160
161   // Les structures de données pour contenir les contributions
162   //sur les faces : élément - voisin
163   DenseMatrix<Number> Kne;
164   DenseMatrix<Number> Ken;
165   DenseMatrix<Number> Kee;
166   DenseMatrix<Number> Knn;
167
168   vector<DenseSubMatrix<Number>>
169                         _col_Kee(Nvar, DenseSubMatrix<Number>(Kee));
170   vector<DenseSubMatrix<Number>>
171                         _col_Knn(Nvar, DenseSubMatrix<Number>(Knn));
172   vector<DenseSubMatrix<Number>>
173                         _col_Kne(Nvar, DenseSubMatrix<Number>(Kne));
174   vector<DenseSubMatrix<Number>>
175                         _col_Ken(Nvar, DenseSubMatrix<Number>(Ken));
176
177   vector<vector<DenseSubMatrix<Number>>> subKne(Nvar, _col_Kne);
178   vector<vector<DenseSubMatrix<Number>>> subKen(Nvar, _col_Ken);
179   vector<vector<DenseSubMatrix<Number>>> subKnn(Nvar, _col_Knn);
180   vector<vector<DenseSubMatrix<Number>>> subKee(Nvar, _col_Kee);
181
182   // Maintenant, nous allons boucler sur tous les éléments du
        maillage.
183   // Nous allons calculer d'abord la matrice intérieure et
184   // le seconde membre et ensuite les contributions sur les faces.
185   MeshBase::const_element_iterator el
186                               = mesh.active_local_elements_begin();
187   const MeshBase::const_element_iterator el_end
188                                  = mesh.active_local_elements_end();
189   // Boucle sur les elements
190   for( ; el != el_end ; ++el)
```

```
191   {
192     // Stocker un pointeur vers l'élément
193     //dont nous travaillons actuellement.
194     // Cela permet une syntaxe plus agréable plus tard.
195     const Elem* elem = *el;
196
197     // Obtenir le degré de liberté pour le élément courant
198     // Ceux-ci définissent où dans la matrice intérieure
199     // et le seconde membre cet élément contribuera.
200     dof_map.dof_indices(elem,dof_indices);
201     int n_dofs = dof_indices.size();
202     vector<int> n_dofs_var(Nvar);
203     for(int var=0 ; var<Nvar ; ++var)
204     {
205       dof_map.dof_indices(elem,dof_indices_var[var],var);
206       n_dofs_var[var] = dof_indices_var[var].size();
207     }
208
209      // Calcul des données spécifiques  pour l'élément  courant
210      // Cela implique  la localisation du
211      // points de quadrature (q_point) et les fonctions de forme
212      // (phi, dphi) pour l'élément courant.
213
214      fe->reinit(elem);
215
216      // Mettre  à zero la matrice et le seconde membre avant de
217      // les additionner. Nous utilisons le membre "resize" ici
218      // parce que le nombre de degrés de liberté pourrait avoir
219      // changé par rapport au dernier élément.
220     Ke.resize(n_dofs, n_dofs);
221     Fe.resize(n_dofs);
222
223     // Repositionner les sous-matrices ... L'idée est la suivante:
224     //
225     //          _            _          _  _
226     //         | Kv_1v_2 |          | Fv_1 |
227     //    Ke = | Kv_2v_1 |;   Fe = | Fv_2 |
228     //          -            -          -  -
229     //
230     // La DenseSubMatrix.repostition membre prend la
231     // (Row_offset, column_offset, row_size, COLUMN_SIZE).
232     //
233     // De même, le DenseSubVector.reposition membre
234     // prend la (row_offset, row_size)
235
236     for(int v_1=0 ; v_1<Nvar ; ++v_1)
237     {
238       subF[v_1].reposition(v_1*n_dofs_var[v_1],n_dofs_var[v_1]);
239
```

```
240       for(int v_2=0 ; v_2<Nvar ; ++v_2)
241       {
242         subK[v_1][v_2].reposition (v_1*n_dofs_var[v_1],
243                                    v_2*n_dofs_var[v_2],
244                                    n_dofs_var[v_1],
245                                    n_dofs_var[v_2]);
246       }
247     }
248
249     //Calcul des intergrales sur l'element
250     for(unsigned int qp=0; qp<qrule.n_points(); ++qp)
251     {
252       //solution aux temps précédent : u^n_q
253       vector<Real> u_old(Nvar+1,0.);
254       u_old[Nvar] = Temperature ;
255
256       //solution à l'itération antérieure
257       vector<Real> u(Nvar+1,0.);
258       u[Nvar] = Temperature ;
259
260       for(int v = 0 ; v < Nvar ; ++v)
261       {
262         for(unsigned int i=0 ; i<phi.size(); ++i)
263         {
264           u_old[v]+= phi[i][qp] * var_renorm[v]
265                      * system.old_solution(dof_indices_var[v][i]);
266                       //u^n-1
267
268           u[v]+= phi[i][qp] * var_renorm[v]
269                  * system.prev_solution(dof_indices_var[v][i]);
270                 //u^n_q
271         }
272       }
273       mf->init(elem->subdomain_id(),u_old);
274
275       // X(u^n-1)
276       vector<Real> X_old (Nvar,0.) ;
277
278       for(int v=0 ; v<Nvar ; ++v)
279       {
280         X_old[v] = (*mf)(X(v)) ;
281       }
282
283       mf->init(elem->subdomain_id(),u);
284
285       for(int v_1=0 ; v_1<Nvar ; ++v_1)
286       {
287         for(unsigned int i=0; i<phi.size(); ++i)
288         {
```

```
289             subF[v_1](i) -= eq_renorm[v_1] * idt
290                             * JxW[qp]
291                             * ( (*mf)(X(v_1)) -X_old[v_1])
292                             * phi[i][qp];
293
294             subF[v_1](i) -= eq_renorm[v_1]
295                             * JxW[qp]
296                             * ((*mf)(A(v_1,'g'))
297                             * gravity)
298                             * dphi[i][qp];
299
300             for(unsigned int j=0; j<phi.size(); ++j)
301             {
302               subK[v_1][v_1](i,j) += eq_renorm[v_1]*var_renorm[v_1]
303                                      * JxW[qp]
304                                      * ((*mf)(C(v_1,'g'))
305                                      * gravity
306                                      * dphi[i][qp])
307                                      * phi[j][qp];
308             }
309           }
310
311           for(int v_2=0 ; v_2<Nvar ; ++v_2)
312           {
313             // Cv_1v_2 * grad u^n_q
314             RealGradient G_i = 0. ;
315
316             for(unsigned int k=0; k<phi.size(); ++k)
317             {
318               G_i += system.prev_solution(dof_indices_var[v_2][k])
319                      * ( (*mf)(C(v_1,v_2)) * dphi[k][qp] );
320
321             }
322             for(unsigned int i=0; i<phi.size(); ++i)
323             {
324               subF[v_1](i) += eq_renorm[v_1] * idt
325                               * JxW[qp]
326                               * (*mf)(J(v_1,v_2))
327                               * u[v_2] * phi[i][qp];
328
329               for(unsigned int j=0; j<phi.size(); ++j)
330               {
331                 subK[v_1][v_2](i,j) += eq_renorm[v_1]* var_renorm[v_2]
332                                        * JxW[qp]* (((*mf)(A(v_1,v_2))
333                                        * dphi[j][qp])* dphi[i][qp]);
334
335                 subK[v_1][v_1](i,j) += eq_renorm[v_1]* var_renorm[v_1]
336                                        * var_renorm[v_2]* JxW[qp]
337                                        * (G_i *dphi[i][qp])*phi[j][qp];
```

```
338
339                 subK[v_1][v_2](i,j) += eq_renorm[v_1] *var_renorm[v_2]
340                                        * idt * JxW[qp]
341                                        * phi[i][qp] * phi[j][qp]
342                                        * (*mf)(J(v_1,v_2));
343               }
344             }
345           } // fin boucle v_2
346         } // fin boucle v_1
347       } // fin boucle points de quadratures
348
349       // Boucle sur les faces de l'élément
350       for (unsigned int side=0; side<elem->n_sides(); side++)
351       {
352        // Maintenant nous imposons les conditions aux limites.
353        // Si l'élément n'a pas de voisin sur un côté =>
354        // la face est située sur la frontière du domaine.
355         if (elem->neighbor(side) == NULL)
356         {
357           // Pointeur vers la face de l'élément
358           fe_elem_face->reinit(elem, side);
359           AutoPtr<Elem> elem_side (elem->build_side(side));
360
361              // h pour calculer le paramètre de pénalisation
                    intérieure
362           const unsigned int elem_b_order = static_cast<unsigned int>
363                                             (fe_elem_face->get_order());
364           const double h_elem = elem->volume()/elem_side->volume() *
365                                 1./pow(elem_b_order, 2.);
366           for (unsigned int qp=0; qp<qface.n_points(); qp++)
367           {
368             vector<Real> u_BC(Nvar+1,0.);
369             u_BC[Nvar] = Temperature;
370
371             for(int v=0 ; v<Nvar ; ++v)
372             {
373               for(unsigned int i=0; i<phi_face.size(); ++i)
374               {
375                 u_BC[v]+= phi[i][qp]*var_renorm[v]
376                           * system.prev_solution(dof_indices_var[v][i])
                              ;
377               }
378             }
379             mf->init(elem->subdomain_id(),u_BC);
380
381             for(int v_1=0 ; v_1<Nvar ; ++v_1)
382             {
383               if( bcs.has("Dirichlet",v_1,mesh.boundary_info->
                     boundary_id(elem,side)))
```

```
384             {
385               for (unsigned int i=0; i<phi_face.size(); i++)
386               {
387                 subF[v_1](i) += eq_renorm[v_1] * epsi*JxW_face[qp]*
388                                 (((*mf)(A(v_1,v_1))
389                                 * dphi_face[i][qp])
390                                 * qface_normals[qp])
391                                 * bcs("Dirichlet",v_1,
392                                       mesh.boundary_info->
393                                       boundary_id(elem,side),
394                                       current_time,elem->point(i));
395                                   // consistence
396                 subF[v_1](i) += eq_renorm[v_1] * JxW_face[qp]
397                                  * eta[v_1]/h_elem
398                                  * phi_face[i][qp]
399                                  * bcs("Dirichlet",v_1,
400                                        mesh.boundary_info->
401                                        boundary_id(elem,side),
402                                        current_time,elem->point(i));
403                                   // stabilité
404
405                 subF[v_1](i) += eq_renorm[v_1]* JxW[qp]
406                                  * ((*mf)(A(v_1,'g'))
407                                  * gravity
408                                  * qface_normals[qp]) * phi_face[i][
                                      qp] ;
409
410                 subF[v_1](i) += eq_renorm[v_1]*JxW[qp]
411                                  * ((*mf)(C(v_1,'g'))
412                                  * gravity *qface_normals[qp])
413                                  * phi_face[i][qp]
414                                  * bcs("Dirichlet",v_1,
415                                        mesh.boundary_info->
416                                        boundary_id(elem,side),
417                                        current_time,elem->point(i));
418
419                 for (unsigned int j=0; j<phi_face.size(); j++)
420                 {
421                   subK[v_1][v_1](i,j) += epsi*eq_renorm[v_1]
422                                          * var_renorm[v_1]
423                                          * JxW_face[qp]
424                                          * phi_face[j][qp]
425                                          *( ((*mf)(A(v_1,v_1))
426                                          * dphi_face[i][qp])
427                                          * qface_normals[qp]);
428                                          // consistence
429
430                   subK[v_1][v_1](i,j) += eq_renorm[v_1]
431                                          * var_renorm[v_1]
```

```
                                        * JxW_face[qp]
                                        * eta[v_1]/h_elem
                                        * phi_face[i][qp]
                                        * phi_face[j][qp];
                                        // stabilité

            }
          }
          for(int v_2=0 ; v_2<Nvar ; ++v_2)
          {
            RealGradient G_bc = 0. ;
            for(unsigned int k=0; k<phi.size(); ++k)
            {
              G_bc += system.prev_solution
                      (dof_indices_var[v_2][k])
                      *((*mf)(C(v_1,v_2))* dphi[k][qp]);

            }
            for (unsigned int i=0; i<phi_face.size(); i++)
            {
              for (unsigned int j=0; j<phi_face.size(); j++)
              {
                subK[v_1][v_2](i,j) -= eq_renorm[v_1]
                                       * var_renorm[v_2]
                                       * JxW_face[qp]
                                       * phi_face[i][qp]
                                       * (((*mf)(A(v_1,v_2))
                                       * dphi_face[j][qp])
                                       * qface_normals[qp]);
                                       // consistence
              }
              subF[v_1](i) += eq_renorm[v_1]
                              * var_renorm[v_1]
                              * var_renorm[v_2]
                              * JxW_face[qp]
                              * phi_face[i][qp]
                              * (G_bc*qface_normals[qp])
                              * bcs("Dirichlet",v_1,
                                    mesh.boundary_info->
                                    boundary_id(elem,side),
                                    current_time,
                                    elem->point(i));
                                 // C
            }
          } // fin boucle v_2

        } // face Dirichlet
        else
        {
```

```
481                for (unsigned int i=0; i<phi_face.size(); i++)
482                {
483                  subF[v_1](i) += eq_renorm[v_1]
484                                  * JxW_face[qp]
485                                  * phi_face[i][qp]
486                                  * bcs("Neumann",v_1,
487                                        mesh.boundary_info->
488                                        boundary_id(elem,side),
489                                        current_time,
490                                        qface_points[qp]);
491                }
492              } // face Neumann
493            } // fin boucle sur v_1
494          } // fin boucle points de quadratures
495        }  // fin neighbor(side) == NULL
496
497         // Si l'élément n'est pas sur le bord du domaine
498        // On boucle sur ses voisins pour calculer
499        // les contributions élément - voisin sur les faces
500        else
501        {
502
503        // Stocker un pointeur sur le voisin dont
504        //nous travaillons  actuellement
505          const Elem* neighbor = elem->neighbor(side);
506
507        // Obtenir l'ID global de l'élément et son voisinr
508          const unsigned int elem_id = elem->id();
509          const unsigned int neighbor_id = neighbor->id();
510
511          if ((neighbor->active() && (neighbor->level()
512                                      == elem->level())
513                                  && (elem_id < neighbor_id))
514                                  || (neighbor->level()
515                                      < elem->level()))
516          {
517          // Pointeur sur la face élément
518            AutoPtr<Elem> elem_side (elem->build_side(side));
519          // h pour calculer le paramètre de pénalisation intérieure
520            const unsigned int elem_b_order =
521                                  static_cast <unsigned int>
522                                    (fe_elem_face->get_order());
523            const unsigned int neighbor_b_order =
524                                  static_cast <unsigned int>
525                                    (fe_neighbor_face->get_order());
526            const double side_order = (elem_b_order
527                                       + neighbor_b_order)/2.;
528            const double h_elem = (elem->volume()/elem_side->volume())
529                                  * 1./pow(side_order,2.);
```

```
530
531         // le point de qudrature sur la face voisine
532           std::vector<Point> qface_neighbor_point;
533
534         // le point de qudrature sur la face
535           std::vector<Point > qface_point;
536
537           // Réinitialisation de la fonctions de forme
538           //sur la face de l'élément
539           fe_elem_face->reinit(elem, side);
540
541         //Obtenir les emplacements physiques des points de
              quadrature élémen
542           qface_point = fe_elem_face->get_xyz();
543
544         // Find their locations on the neighbor
545           FEInterface::inverse_map (elem->dim(), fe->get_fe_type(),
546                                     neighbor, qface_point,
547                                     qface_neighbor_point);
548         // Réinitialisation de la fonctions de forme
549           //sur la face voisine
550           fe_neighbor_face->reinit(neighbor, &qface_neighbor_point);
551
552             // Obtenir le degré de liberté pour le voisin
553            // Ceux-ci définissent où dans matrice globale
554            // ce voisin contribuera.
555           std::vector<unsigned int> neighbor_dof_indices;
556           dof_map.dof_indices (neighbor, neighbor_dof_indices);
557           const unsigned int n_neighbor_dofs
558                              = neighbor_dof_indices.size();
559
560           // Mettre  à zero la matrice et le seconde membre avant de
561           // les additionner.
562           // Nous utilisons le membre redimensionner ici
563           // parce que le nombre de degrés de liberté pourrait avoir
564           // changé par rapport au dernier élément ou un voisin.
565           // Kne et Ken ne sont pas des matrices carrées,
566           // si le voisin et l'élément ont
567           // un niveau différent de "p" (du au p-raffinemnet)
568           Kne.resize (n_neighbor_dofs, n_dofs);
569           Ken.resize (n_dofs, n_neighbor_dofs);
570           Kee.resize (n_dofs, n_dofs);
571           Knn.resize (n_neighbor_dofs, n_neighbor_dofs);
572
573         // Repositionner les sous-matrices ...
574         // La DenseSubMatrix.repostition membre prend la
575         // (Row_offset, column_offset, row_size, COLUMN_SIZE).
576
577           for(int v_1=0 ; v_1<Nvar ; ++v_1)
```

```
578             {
579               for(int v_2=0 ; v_2<Nvar ; ++v_2)
580               {
581                 subKee[v_1][v_2].reposition (v_1*n_dofs_var[v_1],
582                                              v_2*n_dofs_var[v_2],
583                                              n_dofs_var[v_1],
584                                              n_dofs_var[v_2]);
585                 subKnn[v_1][v_2].reposition (v_1*n_dofs_var[v_1],
586                                              v_2*n_dofs_var[v_2],
587                                              n_dofs_var[v_1],
588                                              n_dofs_var[v_2]);
589                 subKen[v_1][v_2].reposition (v_1*n_dofs_var[v_1],
590                                              v_2*n_dofs_var[v_2],
591                                              n_dofs_var[v_1],
592                                              n_dofs_var[v_2]);
593                 subKne[v_1][v_2].reposition (v_1*n_dofs_var[v_1],
594                                              v_2*n_dofs_var[v_2],
595                                              n_dofs_var[v_1],
596                                              n_dofs_var[v_2]);
597               }
598             }
599
600
601             for (unsigned int qp=0; qp<qface.n_points(); qp++)
602             {
603               vector<Real> u_face(Nvar+1,0.);
604               u_face[Nvar] = Temperature ;
605               for(int v_1=0 ; v_1<Nvar ; ++v_1)
606               {
607                 for(unsigned int i=0; i<phi_face.size(); ++i)
608                 {
609                   u_face[v_1]+= phi[i][qp]* var_renorm[v_1]
610                                 * system.prev_solution(dof_indices_var
                                      [v_1][i]);
611                 }
612               }
613               mf->init(elem->subdomain_id(),u_face);
614               for(int v_1=0 ; v_1<Nvar ; ++v_1)
615               {
616                 for(unsigned int i=0; i<phi_face.size(); ++i)
617                 {
618                   subF[v_1](i) += eq_renorm[v_1]* JxW[qp]
619                                   * ((*mf)(C(v_1,'g'))
620                                  * gravity *qface_normals[qp])
621                                  * phi_face[i][qp];
622                 }
623                 for (unsigned int i=0; i<phi_neighbor_face.size(); i
                      ++)
624                 {
```

```
625                 subF[v_1](i) -= eq_renorm[v_1]* JxW[qp]
626                                 * ((*mf)(C(v_1,'g'))
627                                 * gravity *qface_normals[qp])
628                                 * phi_neighbor_face[i][qp];
629               }
630
631               for(int v_2=0 ; v_2<Nvar ; ++v_2)
632               {
633                 RealGradient G_f = 0. ;
634                 for(unsigned int k=0; k<phi_face.size(); ++k)
635                 {
636                   G_f += system.prev_solution
637                          ( dof_indices_var[v_2][k])
638                          * ((*mf)(C(v_1,v_2)) * dphi[k][qp] );
639                 }
640
641                 for (unsigned int i=0; i<phi_face.size(); i++)
642                 {
643                   for (unsigned int j=0; j<phi_face.size(); j++)
644                   {
645                     subKee[v_1][v_2](i,j) += 0.5 * eq_renorm[v_1]
646                                              * var_renorm[v_2]*
647                                              JxW_face[qp]
648                                              * (epsi*phi_face[j][qp]
649                                              * (( (*mf)(A(v_1,v_2))
650                                              * dphi_face[i][qp] )
651                                              * qface_normals[qp])
652                                              - phi_face[i][qp]*
653                                              (((*mf)(A(v_1,v_2))
654                                              * dphi_face[j][qp])
655                                              * qface_normals[qp]));
656                                              // consistence
657                     subKee[v_1][v_2](i,j) += eq_renorm[v_1]
658                                              * var_renorm[v_2]
659                                              * JxW_face[qp]
660                                              * eta[v_1]/h_elem
661                                              * phi_face[j][qp]
662                                              * phi_face[i][qp];
663                                              // stabilité
664
665                     subKee[v_1][v_1](i,j) += 0.5 *eq_renorm[v_1]
666                                              * var_renorm[v_1]
667                                              * var_renorm[v_2]
668                                              * JxW_face[qp]
669                                              * (G_f*qface_normals[qp
                                                 ])
670                                              * phi_face[i][qp]
671                                              * phi_face[j][qp];
672                                            // terme convectif
```

```
673                     }
674                 }
675
676                 for (unsigned int i=0; i<phi_neighbor_face.size(); i
                        ++)
677                 {
678                   for (unsigned int j=0; j<phi_neighbor_face.size();
                          j++)
679                   {
680
681                     subKnn[v_1][v_2](i,j) -= 0.5 * eq_renorm[v_1]
682                                              * var_renorm[v_2]
683                                              * JxW_face[qp]*
684                                              (epsi*phi_neighbor_face
685                                                          [j][qp]
686                                              * (((*mf)(A(v_1,v_2))
687                                              * dphi_neighbor_face[i
                                                   ][qp])
688                                              * qface_normals[qp])
689                                              - phi_neighbor_face
690                                                          [i][qp]
691                                              * (((*mf)(A(v_1,v_2))
692                                              * dphi_neighbor_face
693                                                          [j][qp])
694                                              * qface_normals[qp]));
695                                            // consistence
696                     subKnn[v_1][v_2](i,j) += eq_renorm[v_1]
697                                              * var_renorm[v_2]
698                                              * JxW_face[qp]
699                                              * eta[v_1]/h_elem
700                                              * phi_neighbor_face
701                                                          [j][qp]
702                                              * phi_neighbor_face
703                                                          [i][qp];
704                                              // stabilité
705
706                     subKnn[v_1][v_1](i,j) -= 0.5 *eq_renorm[v_1]
707                                              * var_renorm[v_1]
708                                              * var_renorm[v_2]
709                                              * JxW_face[qp]
710                                              * (G_f*qface_normals[qp
                                                   ])
711                                              * phi_neighbor_face
712                                                          [i][qp]
713                                              * phi_neighbor_face
714                                                          [j][qp];
715                                            // terme convectif
716                   }
717                 }
```

```
718
719                 for (unsigned int i=0; i<phi_neighbor_face.size(); i
                        ++)
720                 {
721                   for (unsigned int j=0; j<phi_face.size(); j++)
722                   {
723                     subKne[v_1][v_2](i,j) += 0.5 * eq_renorm[v_1]
724                                              * var_renorm[v_2]
725                                              * JxW_face[qp]
726                                              * (phi_neighbor_face
727                                                                [i][qp]
728                                              * (((*mf)(A(v_1,v_2))
729                                              * dphi_face[j][qp])
730                                              * qface_normals[qp])
731                                              + epsi*phi_face[j][qp]
732                                              * (((*mf)(A(v_1,v_2))
733                                              * dphi_neighbor_face
734                                                                [i][qp])
735                                              * qface_normals[qp]));
736                                             // consistence
737                     subKne[v_1][v_2](i,j) -= eq_renorm[v_1]
738                                              * var_renorm[v_2]
739                                              * JxW_face[qp]
740                                              * eta[v_1]/h_elem
741                                              * phi_face[j][qp]
742                                              * phi_neighbor_face
743                                                                [i][qp];
744                                              // stabilité
745                     subKne[v_1][v_1](i,j) -= 0.5 * eq_renorm[v_1]
746                                              * var_renorm[v_1]
747                                              * var_renorm[v_2]
748                                              * JxW_face[qp]
749                                              * (G_f*qface_normals[qp
                                                    ])
750                                              * phi_face[j][qp]
751                                              * phi_neighbor_face
752                                                                [i][qp];
753                                             // terme convectif
754                   }
755                 }
756
757                 for (unsigned int i=0; i<phi_face.size(); i++)
758                 {
759                   for (unsigned int j=0; j<phi_neighbor_face.size();
                          j++)
760                   {
761                     subKen[v_1][v_2](i,j) -= 0.5 *eq_renorm[v_1]
762                                              * var_renorm[v_2]
763                                              * JxW_face[qp]* (epsi
```

```
764                                                 * phi_neighbor_face
765                                                         [j][qp]
766                                                 * (((*mf)(A(v_1,v_2))
767                                                 * dphi_face[i][qp])
768                                                 * qface_normals[qp])
769                                                 + phi_face[i][qp]
770                                                 * (((*mf)(A(v_1,v_2))
771                                                 * dphi_neighbor_face
772                                                         [j][qp])
773                                                 * qface_normals[qp]));
774                                                 // consistence
775                         subKen[v_1][v_2](i,j) -= eq_renorm[v_1]
776                                                 * var_renorm[v_2]
777                                                 * JxW_face[qp]
778                                                 * eta[v_1]/h_elem
779                                                 * phi_face[i][qp]
780                                                 *phi_neighbor_face
781                                                         [j][qp];
782                                                 // stabilité
783                         subKen[v_1][v_1](i,j) += 0.5 * eq_renorm[v_1]
784                                                 * var_renorm[v_1]
785                                                 * var_renorm[v_2]
786                                                 * JxW_face[qp]
787                                                 * (G_f*qface_normals[qp
                                                        ])
788                                                 * phi_face[i][qp]
789                                                 * phi_neighbor_face
790                                                         [j][qp];
791                                                 // terme convectif
792                             }
793                         }
794
795                     } //fin boucle sur  v_2
796
797                   }// fin boucle sur v_1
798
799                 } // fin boucle sur les points de quadratures
800
801                   // les contribution sont maintenant construits
802                  // pour ce face, Ajoutons-les à la matrice globale
803                  // Le SparseMatrix:: les membres add_matrix() le font
                        pour nous.
804                 system.matrix->add_matrix(Kne,neighbor_dof_indices,
                        dof_indices);
805                 system.matrix->add_matrix(Ken,dof_indices,
                        neighbor_dof_indices);
806                 system.matrix->add_matrix(Kee,dof_indices);
807                 system.matrix->add_matrix(Knn,neighbor_dof_indices);
808               } // ->if element est activ
```

```
809         } //fin boucle sur les face internes  (else)
810       }  // fin boucle sur les faces
811
812       // La matrice intérieure et le seconde membre sont désormais
              construits
813       // pour cet élément. Ajoutons-les à la matrice globale et
814       // le vecteur Rhs en utilisant SparseMatrix:: add_matrix()
815       // et NumericVector:: add_vector()
816       system.matrix->add_matrix(Ke, dof_indices);
817       system.rhs->add_vector(Fe, dof_indices);
818     } // fin boucle sur les elementes
819     system.matrix->close();
820     delete[] dof_indices_var ;
821 } //fin
```

9.3 Grandeurs physique

Symbole	Signification	Unité
G	Enthalpie libre, énergie libre de Gibbs	J
G_α	Enthalpie libre de la phase α	J
H^i	Constante de Henry	$mol.J^{-1}$
K_H^i	Constante de Henry du composant i	Pa
M^i	Masse molaire du composant i	$kg.mol^{-1}$
N_α^i	Nombre de moles du composant i dans la phase α	mol
P_α	Pression de la phase α	Pa
P_c	Pression capillaire	Pa
P^0	Pression de référence	Pa
P_v^s	Pression de vapeur saturante du solvant	Pa
R	Constante des gaz parfaits x	$J.mol^{-1}.K^{-1}$
T	Température	K
V_α	Volume de la phase α	m^3
$V_\alpha^{i,*}$	Volume du composant i dans la phase α dans l'état pur de référence	m^3
v_α	Volume molaire de la phase α	$m^3.mol^{-1}$
v_α^i	Volume molaire partiel du composant i dans la phase α	$m^3.mol^{-1}$
$v_\alpha^{i,*}$	Volume molaire partiel du composant i dans la phase α dans l'état pur de référence	$m^3.mol^{-1}$
$v_\alpha^{i,\infty}$	Volume molaire partiel du composant i dans la phase α à dilution infinie	$m^3.mol^{-1}$
x_α^i	Fraction molaire du composant i dans la phase α	$-$
γ_α^i	Coefficient d'activité du composant i dans la phase α	$-$
$\rho_\alpha^{s,*}$	Masse volumique du solvant pur	$kg.m^{-3}$
μ_α^i	Potentiel chimique du composant i dans la phase α	$J.mol^{-1}$
$\mu_\alpha^{i,*}$	Potentiel chimique du composant i dans la phase α dans l'état pur de référence	$J.mol^{-1}$
$\mu_\alpha^{i,0}$	Potentiel chimique de référence du composant i dans la phase α	$J.mol^{-1}$