Projet
Réalisation d'un solveur discontinuous-Galerkin pour équation de Poisson 1d en C++

Présenté par :
Johan MARGUET et Soufian MECHOUAT

Encadré par :
Monsieur Fabien Marche

Table des matières

Introduction

Les méthodes de Galerkin discontinues sont une technique classique pour approximer les équations aux dérivées partielles de type elliptiques et hyperboliques. Une théorie unifiée a été développée récemment dans le cadre des systèmes de Friedrich [3]. Pour les équations aux dérivées partielles elliptiques, l'une des méthodes les plus populaires est la méthode Symmetric Interior Penalty (SIP) introduite par Baker [2] et Arnold [1].

Motivation

Nous nous appuierons tout au long de notre rapport sur le livre de Di Pietro [4] pour l'étude de la base théorique nécessaire à la réalisation de notre solveur 1D de l'équation de Poisson.
Le but principal de ce projet est de réaliser un solveur informatique dans le langage C++ afin dans un premier temps de calculer les termes de la matrice représentés par la forme bilinéaire discrète obtenue par la méthode de Galerkin Discontinue SIP. Puis dans un deuxième temps proposer une résolution de notre problème de Poisson via un solver automatique tel que Freefem ++.
Pourquoi le langage C++ ? Tout simplement parce qu'avec ce projet il s'agit d'une bonne occasion de faire le lien avec notre enseignement reçu dans l'UE de Programmation qui consistait à nous donner une base solide pour le calcul scientifique et donc nous montrer l'utilité forte de ce langage dans le monde de la simulation numérique.
Un travail théorique de compréhension est nécessaire avant toute étape de codage, qui là aussi nous permet de faire un lien entre l'UE d'Analyse Fonctionnelle du premier semestre et celle d'Analyse numérique 3 du second semestre.

1 Méthode de Galerkin discontinue

1.1 Principe

Cadre générale de la méthode de Galerkin discontinue

Tout comme la méthode continue de Galerkin, la méthode discontinue de Galerkin est une méthode d'éléments finis formulée par rapport à une formulation faible. Contrairement aux méthodes CG traditionnelles conformes, la méthode DG fonctionne sur un espace d'essai de fonctions qui ne sont continues que par morceaux, et comprennent donc souvent des espaces de fonctions plus inclusifs que les sous-espaces de produits internes de dimension finie utilisés dans les méthodes conformes. Il s'agit d'une approximation non conforme.
Contrairement à la formulation de Galerkin continue qui consiste en l'intégration sur tout le domaine puis la discrétisation, la formulation discontinue est une formulation locale. La solution est recherchée dans chaque élément séparément.

Nous nous intéresserons plus particulièrement à la méthode SIP dont nous détaillons le principe dans le paragraphe suivant.

1.2 Méthode à Pénalité Intérieure Symétrique (SIP)

La forme bilinéaire SIP est obtenue en intégrant par parties sur chaque élément du maillage, et en sommant sur tous les éléments. Dans notre forme bilinéaire discrète on ajoute un premier terme permettant d'avoir une forme bilinéaire consistante puis un second terme permettant de garder la symétrie de la forme bilinéaire d'origine. Ensuite un troisième terme est ajouté que l'on appelle terme de pénalité permettant d'avoir la coercivité discrète. Ce terme de pénalité impose une faible continuité de la solution numérique.

1.3 Avantages et inconvénients

La méthode de Galerkin discontinue à de nombreux avantages et désavantages, nous les avons listés ci-dessous :

Avantages :

- Les matrices intervenant dans le système linéaire à résoudre sont creuses et structurées par blocs de taille égale au nombre de degrés de liberté d'un élément du maillage. De plus avec la méthode SIP la matrice est globale définie positive permettant d'utiliser des solveurs plus rapides.

- Elle est adaptée pour un ordre élevé (supérieur à 2)

- La discontinuité de la solution entre les éléments permet de ne pas se soucier de la régularité des maillages. Cela permet donc de travailler avec des maillages non structurés et non conformes donc il est plus facile de représenter certaines géométries adaptées à l'industrie car bien souvent, on cherche à approcher un domaine compliqué.

Désavantages :

- Comme vu dans l'UE d'Analyse numérique 3, avec des discontinuités on peut observer une perte de précision de la solution numérique ainsi que l'apparition d'un **mode parasite. Un exemple est le problème de Stokes que l'on a vu en cours.** Mais on peut remédier à cela à l'aide de diverses techniques.

- Les coûts en encombrement mémoire et en temps CPU sont importants à cause des valeurs de la solution aux interfaces ce qui augmentent le nombre de degrés de liberté. Il faut mettre l'accent sur des techniques de simplification des calculs et d'accélération de convergence.

1.4 État de l'art en simulation numérique

La simulation numérique est devenue aujourd'hui incontournable dans l'innovation industrielle et plus généralement dans la recherche scientifique. Elle permet d'aller plus loin que la théorie qui atteint ces limites lorsque les calculs deviennent si complexes et que les modèles d'équations ne sont plus simplifiables.
Que reste-il ? L'expérience ? Oui mais là aussi il y a des limites, par exemple une entreprise peut-elle expérimenter à l'infini ? Non si vous prenez le secteur de l'aéronautique, on ne va pas construire plusieurs avions afin d'y réaliser des tests car il y a la une question de budget.
De manière générale la simulation numérique est un moteur indispensable à la compétitivité industrielle, permet de concevoir plus rapidement, plus sûrement et de réduire les coûts en limitant la fréquence des tests sur prototypes.
Bien évidemment la simulation numérique ne remplace pas l'expérience ou la théorie mais avec l'évolution rapide des capacités informatiques (CPU, stockage mémoire et de temps de calcul), elle prendra de plus en plus de place dans le monde de la recherche scientifique.
Bien sûr pour répondre à cela il faut choisir la bonne méthode numérique qui doit être un compromis entre précision de la solution numérique, temps de calcul nécessaire etc.

Plusieurs domaines sont concernés par la simulation numérique, parlons de la mécanique des fluides qui intervient dans le domaine de l'aéronautique où l'on étudie **les écoulements turbulents** et qui sont d'un enjeu capital pour leurs applications industrielles malgré leur complexité.
Ou bien les bâtiments et les villes lorsqu'on simule la circulation urbaine, des ambiances sonores et l'exposition aux ondes électromagnétiques des habitants.

La simulation intervient donc dans plusieurs domaines tous aussi variés tels que : l'agriculture, la géophysique et l'astrophysique, la météorologie et la climatologie, les énergies et la biomécanique, le nucléaire etc.

Pour conclure on peut affirmer que la simulation numérique est clairement une réponse aux enjeux scientifiques et industrielle de demain en continuant à améliorer et à développer les outils numériques.

2 Application de la méthode SIP au problème de Poisson

2.1 Description du problème

Notre sujet nous amène à travailler sur l'équation de Poisson à une dimension qui est une équation aux dérivées partielles elliptique du second ordre avec les conditions initiales de Dirichlet homogène suivante :

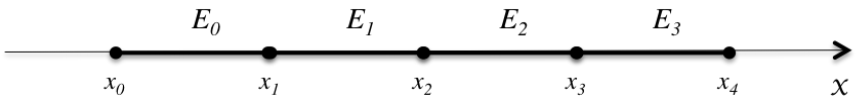
$$\begin{cases} -\Delta u(x) = f(x) \quad \text{sur } \Omega \\ u = 0 \quad \text{sur } \partial\Omega \end{cases} \qquad (1)$$

où Δ est l'opérateur laplacien et f une fonction appartenant à $L^2(\Omega)$.

En pratique cette équation peut amener à des problèmes importants dans différents domaines comme en mécanique des fluides, électrostatique selon le f que nous choisissons.

2.2 Discrétisation unidimensionnelle

Soit Ω = [0,1] notre domaine unidimensionnel selon l'espace x qui est maillé en un nombre fini de *n* éléments $E_j = [x_j, x_{j+1}]$. De plus E_j est maillé de façon uniforme de la manière suivante $x_j = a + jh$ où $j = 0,\ldots,N$ est l'index de l'élément.
On définit la taille de chaque élément par $h_j = x_{j+1} - x_j$ et h : = max h_E la taille d'une maille.



Exemple illustration de la discrétisation de Ω

Après avoir construit un maillage pour notre domaine Ω, nous aurons besoin d'énoncer quelques notions importantes qui jouent un rôle important pour la méthode de Galerkin discontinues SIP.
C'est l'objet de la section suivante.

2.3 Face de maillage, formule de saut et moyenne

Soit T un maillage du domaine Ω.
Pour tout T ∈ T, h_T représente le diamètre de T et la taille d'une maille est défini par le nombre réel h : = max h_T.

Définition 1 :

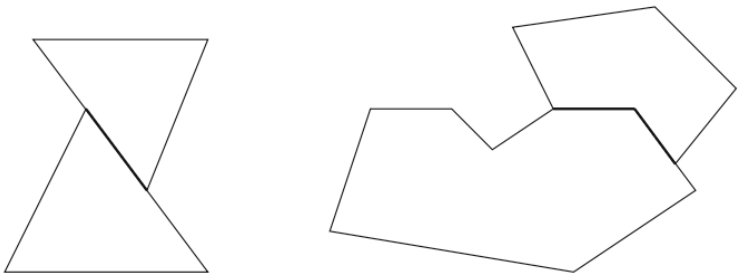
Soit T_h un maillage de Ω.

On dit qu'un sous-ensemble (fermé) F de E est une face du maillage si F a une mesure de Hausdorff positive (d – 1) de dimension (en dimension 1, cela signifie que F est non vide) et si l'une des deux conditions suivantes est satisfaite :

- Il existe des éléments de maillage distincts T_1 et T_2 tels que F = ∂T1 ∩ ∂T2 ; dans ce cas, F est appelée **une interface.**
- Il existe T ∈ Th tel que F = ∂T ∩ ∂Ω ; dans ce cas, F est appelée **face frontière.**

Les interfaces sont collectées dans l'ensemble F_h^i, et les frontières sont collectées dans l'ensemble F_h^b que nous regrouperons dans l'ensemble :

$$F_h \coloneqq F_h^i \cup F_h^b$$



Exemples d'interface pour une maille simple (à gauche) et une maille général (à droite)

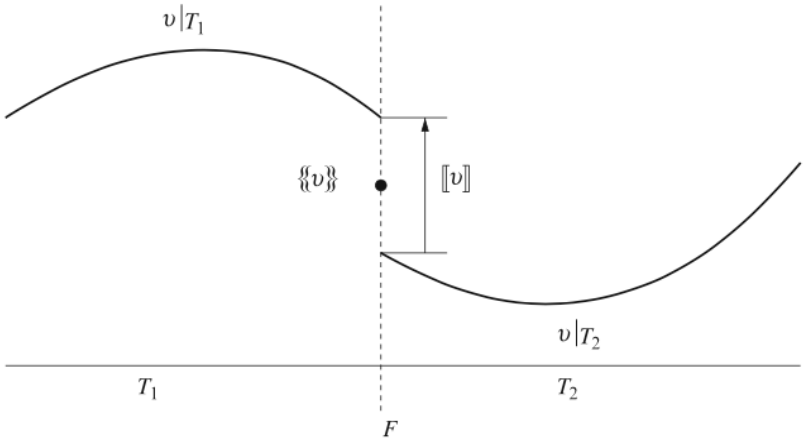
<u>Définition 2</u> : Formules de saut et moyenne

Pour tout F ∈ F_h^i et x ∈ F, la moyenne de v est définie par :

$$\{\{v\}\}_F(x) \coloneqq \frac{1}{2}\left(v_{|T_1}(x) + v_{|T_2}(x)\right)$$

Et la formule de saut est définir par :

$$[\![v]\!]_F(x) \coloneqq v_{|T_1}(x) - v_{|T_2}(x)$$



Exemple pour d = 1 des opérateurs de moyenne et de saut. L'interface F est réduite à un point séparant deux intervalles adjacents

Maintenant il faut que l'on choisisse un espace discret V_h où l'on prendra nos fonctions tests puisque aucune continuité n'est imposée aux interfaces de notre domaine.
Comme choix nous utiliserons celui de l'espace de polynômes par morceaux.
Cela nous permettra d'obtenir la formulation variationnelle et donc d'en tirer la forme bilinéaire issus de la méthode SIP que nous recherchons.

<u>Définition 3 :</u> (Espace des polynômes par morceaux $\mathbf{P_d^k}$)

$$\mathbf{P_d^k}\,(\mathrm{E_h}) := \{\, v \in L^2(\Omega) \mid \forall\, E \in \mathrm{E_h},\, v_{|E} \in \mathbf{P_d^k}\,(E) \,\}$$

La dimension de cet espace est donnée par : $\dim(\mathbf{P_d^k}) = \frac{(k+d)!}{k!d!}$

Les méthodes de Galerkin discontinues reposent sur ces espaces polynomiaux brisés.

2.4 Forme bilinéaire SIP

Vous trouverez en Annexe (voir plus bas) une explication plus détaillée de ce qui a été compris par nos soins pour l'obtention de la forme bilinéaire discrète SIP. Pour l'instant, dans cette partie nous nous concentrerons sur le principal afin de rendre la lecture le plus claire possible.

Dans la suite on utilisera la notation E pour le maillage et $\mathrm{E_h}$ pour une maille de taille h.
On considère dans ce qui suit la discrétisation introduite au paragraphe 3.1

<u>Étape 1</u> : Formulation faible pour le problème continu

On rappelle la formulation faible de notre problème (1) qui consiste à trouver :

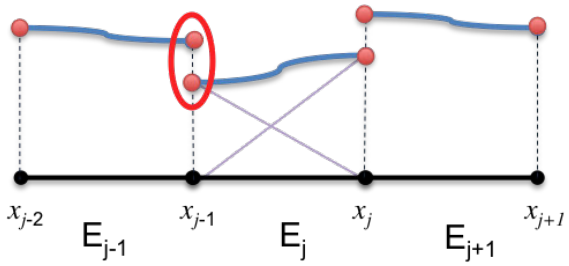
$$u \in V \text{ tel que } a(u,v) = \int_0^1 fv \quad \forall\, v \in V$$

Avec $V = H_0^1(\Omega) := \{\, v \in H^1(\Omega)\ tel\ que\ v_{\partial\Omega} = 0 \,\}$ et la forme bilinéaire qui sera donnée par (à l'aide d'une intégration par partie sur tout Ω) :

$$a(u,v) := \int_0^1 \nabla u \nabla v$$

<u>Étape 2</u> : Espace discret pour la méthode GD

Les fonctions tests engendrant l'espace où est recherchée la solution numérique sont supposées polynomiales mais sans condition de continuité aux interfaces. Donc, la solution numérique est naturellement discontinue aux interfaces entre les éléments.



Interface ou la solution est non continue

On choisira donc : $v_h \in V_h^p(E_h) = \{v \in L^2(\Omega) \mid \forall\, j \in \{1,\ldots,N\}, v_{|E_h} \in \mathbb{P}_p(E_h)\}$

Remarque 1 : Comme aucune continuité n'est imposée aux interfaces des éléments, la méthode GD constitue une méthode des éléments finis non conforme : V_h^p n'est pas inclus dans V.

<u>Étape 3</u> : Choix de la base

Pour le choix de la base plusieurs peuvent être possible, pour une question de simplification des calculs nous avons choisis une base qui sera valable pour toutes les mailles de notre domaine Ω. Voici celle que nous utiliserons pour la suite de nos calculs :

$$\varphi_1^i(x) = \begin{cases} \dfrac{x - x_i}{h} \;\; si \;\; x_{i-1/2} \leq x \leq \;\; x_{i+1/2} \\ 0 \; sinon \end{cases}$$

$$\varphi_2^i(x) = \begin{cases} 1 \;\; si \;\; x_{i-1/2} \leq x \leq \;\; x_{i+1/2} \\ 0 \; sinon \end{cases} \text{ pour tout } 1 \leq i \leq N.$$

Comme nous pouvons le voir notre base est de dimension deux. Cette base engendre bien évidemment notre espace discret V_h^p.

Remarque 2 : On aurait pu utiliser une autre base qui s'applique à une maille puis la modifier pour chaque maille de façon local. Un exemple aurait été : Vect(V_h^p) = {1,1-x}.
Une autre méthode aurait été de passer par une famille affine finie d'élément (transformation affine, élément de référence etc.)

<u>Étape 4</u> : Formulation variationelle de notre problème discret GD

Notre problème consiste à trouver $v_h \in V_h^p$ tel que :

$$a(u_h, v_h) = l(v_h) \quad \forall\ v_h \in V_h^p$$

Pour trouver la formulation variationnelle de notre problème discret, on multiplie l'équation (1) par une fonction test v discontinue et on intègre par parties (formule de Green) sur chaque élément E_j

$$a(u_h, v_h) := \int_0^1 \nabla u_h \nabla v_h \qquad \forall\ v_h \in V_h^p$$

<u>Étape</u> 5 : Méthode SIP

La méthode de symétrie à pénalité intérieure (SIP) consiste en l'intégration par parties du Laplacien, puis en la somme sur tous les éléments. Ensuite, des termes sont alors rajoutés à la formulation. Le premier est issus de l'intégration par partie, le deuxième, troisième terme et quatrième terme correspondent aux termes de consistance, symétrie et pénalité.

Pour la trouver nous nous sommes appuyés sur la formule bilinéaire présente dans le livre de Di Pietro qui traite les cas à une dimension, deux dimensions etc.
Nous avons dû l'adapter par rapport à notre problème ce qui nous a permis d'obtenir le résultat suivant et qui est à la base de notre projet. On rappelle leur utilité dans la partie 1.2

La formulation variationnelle correspondante au problème (1) est définie tel que $\forall\ v_h \in V_h^p$, on a :

<u>Cas général :</u>

$$\begin{aligned} a_h^{\mathrm{sip}}(v, w_h) = & -\sum_{T\in\mathcal{T}_h}\int_T (\triangle v) w_h + \sum_{F\in\mathcal{F}_h^i}\int_F [\![\nabla_h v]\!]\cdot \mathrm{n}_F \{\!\{w_h\}\!\} \\ & -\sum_{F\in\mathcal{F}_h}\int_F [\![v]\!]\{\!\{\nabla_h w_h\}\!\}\cdot \mathrm{n}_F + \sum_{F\in\mathcal{F}_h}\frac{\eta}{h_F}\int_F [\![v]\!][\![w_h]\!]. \end{aligned}$$

<u>Cas unidimensionnel :</u>

$$a_h^{sip}(u_h, v_h) = -\sum_{k=1}^{N+1} \int_{x_{k-1}}^{x_k} (\Delta v) v_h \; + \sum_{k=1}^{N} [\![\nabla_h v]\!].\{\{w_h\}\} - \sum_{k=1}^{N} \{\{\nabla_h v_h\}\}[\![v]\!] + \sum_{k=1}^{N} \frac{\eta}{h_F} [\![w_h]\!][\![v]\!] \quad \forall\, v_h \in V_h$$

Où $\eta > 0$ *est un paramètre* de pénalité que l'on choisit.

2.4 Matrice de rigidité

On se positionne dans le cas où on discrétise le segment [0,1] avec N+2 points.
On pose h = 1/N+1.
En prenant les fonctions de bases définies précédemment, la matrice de discrétisation est donc de cette forme :

$$A_h = \begin{pmatrix} a(\varphi_1^1, \varphi_1^1) & a(\varphi_1^1, \varphi_2^1) & \cdots & a(\varphi_1^1, \varphi_1^N) & a(\varphi_1^1, \varphi_2^N) \\ a(\varphi_2^1, \varphi_1^1) & a(\varphi_2^1, \varphi_2^1) & & a(\varphi_2^1, \varphi_1^N) & a(\varphi_2^1, \varphi_2^N) \\ \vdots & & \ddots & & \vdots \\ a(\varphi_1^N, \varphi_1^1) & a(\varphi_1^N, \varphi_2^1) & & a(\varphi_1^N, \varphi_1^N) & a(\varphi_1^N, \varphi_2^N) \\ a(\varphi_2^N, \varphi_1^1) & a(\varphi_2^N, \varphi_2^1) & \cdots & a(\varphi_2^N, \varphi_1^N) & a(\varphi_2^N, \varphi_2^N) \end{pmatrix}$$

Où,

$$a(\varphi_m^i, \varphi_n^j) = -\sum_{k=1}^{N+1} \int_{x_{k-1}}^{x_k} \varphi_m^{i\,\prime} \varphi_n^j + \sum_{k=1}^{N} [[\varphi_m^{i\,\prime}]]\{\{\varphi_n^j\}\} - \sum_{k=1}^{N} \left[\left[\varphi_m^i\right]\right]\left\{\left\{\varphi_n^{j\,\prime}\right\}\right\} + \sum_{k=1}^{N} \frac{\eta}{h} \left[\left[\varphi_m^i\right]\right]\left[\left[\varphi_n^j\right]\right] \quad \text{pour } 1 \le i,j \le N \text{ et m,n = 1 ou 2.}$$

On remarque ainsi que les trois dernières sommes ne dépendent pas des mailles sur lesquelles on travaille mais uniquement des fonctions de bases que l'on utilise.
Calculons donc $[[\varphi_1^i]]$, $\{\{\varphi_1^i\}\}$, $\left[\left[\varphi_2^i\right]\right]$, $\left\{\left\{\varphi_2^i\right\}\right\}$, $[[\varphi_1^{i\,\prime}]]$, $\{\{\varphi_1^{i\,\prime}\}\}$, $\left[\left[\varphi_2^{i\,\prime}\right]\right]$ et $\left\{\left\{\varphi_2^{i\,\prime}\right\}\right\}$:

- $[[\varphi_1^i]]$ = -1/2 -1/2 = -1
- $\{\{\varphi_1^i\}\} = \frac{\frac{1}{2}-\frac{1}{2}}{2} = 0$
- $\left[\left[\varphi_2^i\right]\right]$ = 1-1 = 0
- $\left\{\left\{\varphi_2^i\right\}\right\} = \frac{1+1}{2} = 1$
- $[[\varphi_1^{i\,\prime}]]$ = 1/h– 1/h = 0
- $\{\{\varphi_1^{i\,\prime}\}\} = \frac{\frac{1}{h}+\frac{1}{h}}{2} = 1/h$

- $\left[\left[\varphi_2^{i\,\prime}\right]\right] = 0$
- $\left\{\left\{\varphi_2^{i\,\prime}\right\}\right\} = 0$

Calculons dorénavant les coefficients de la matrice :

1. $a\left(\varphi_1^i, \varphi_1^j\right) = -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_1^j + \sum_{k=1}^{N}[[\varphi_1^{i\,\prime}]]\{\{\varphi_1^j\}\} - \sum_{k=1}^{N}\left[\left[\varphi_1^i\right]\right]\left\{\left\{\varphi_1^{j\,\prime}\right\}\right\} + \sum_{k=1}^{N}\frac{\eta}{h}\left[\left[\varphi_1^i\right]\right][\left[\varphi_1^j\right]]$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_1^j + \sum_{k=1}^{N} 0 * 0 - \sum_{k=1}^{N} -1 * \frac{1}{h} + \sum_{k=1}^{N}\frac{\eta}{h} * -1 * -1$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_1^j + \frac{N}{h} + \frac{\eta}{h} * N$

<u>Si i ≠ j</u>, $a\left(\varphi_1^i, \varphi_1^j\right) = \frac{N}{h}(1+\eta) = N(N+1)(1+\eta)$

<u>Si i = j</u>, $a\left(\varphi_1^i, \varphi_1^i\right) = \frac{N}{N+1}(1+\eta) - \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\frac{1}{h^2}(x - x_i) = \frac{N}{N+1}(1+\eta) - \frac{1}{h^2}\frac{\left(x_{i+\frac{1}{2}} - x_i + x_{i-\frac{1}{2}} - x_i\right)}{2}$
$= N(N+1)(1+\eta)$

2. $a\left(\varphi_1^i, \varphi_2^j\right) = -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_2^j + \sum_{k=1}^{N}[[\varphi_1^{i\,\prime}]]\{\{\varphi_2^j\}\} - \sum_{k=1}^{N}\left[\left[\varphi_1^i\right]\right]\left\{\left\{\varphi_2^{j\,\prime}\right\}\right\} + \sum_{k=1}^{N}\frac{\eta}{h}\left[\left[\varphi_1^i\right]\right][\left[\varphi_2^j\right]]$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_2^j + \sum_{k=1}^{N} 0 * 1 - \sum_{k=1}^{N} -1 * 0 + \sum_{k=1}^{N}\frac{\eta}{h} * -1 * 0$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_1^{i\,\prime}\varphi_2^j$

<u>Si i ≠ j</u>, $a\left(\varphi_1^i, \varphi_2^j\right) = 0$

<u>Si i = j</u>, $a\left(\varphi_1^i, \varphi_2^i\right) = -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}}\frac{1}{h} * 1 = -\frac{1}{h^2} = -(N+1)^2$

3. $a\left(\varphi_2^i, \varphi_1^j\right) = -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_2^{i\,\prime}\varphi_1^j + \sum_{k=1}^{N}[[\varphi_2^{i\,\prime}]]\{\{\varphi_1^j\}\} - \sum_{k=1}^{N}\left[\left[\varphi_2^i\right]\right]\left\{\left\{\varphi_1^{j\,\prime}\right\}\right\} + \sum_{k=1}^{N}\frac{\eta}{h}\left[\left[\varphi_2^i\right]\right]\left[\left[\varphi_1^j\right]\right]$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_2^{i\,\prime}\varphi_1^j + \sum_{k=1}^{N} 0 * 0 - \sum_{k=1}^{N} 0 * \frac{1}{h} + \sum_{k=1}^{N}\frac{\eta}{h} * 1 * 0$
$= -\sum_{k=1}^{N+1}\int_{x_{k-1}}^{x_k} \varphi_2^{i\,\prime}\varphi_1^j$

<u>Si i ≠ j</u>, $a\left(\varphi_2^i, \varphi_1^j\right) = 0$

<u>Si i = j</u>, $a\left(\varphi_2^i, \varphi_1^i\right) = -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} 0 * \frac{x\_x_i}{h} = 0$

4. $a(\varphi_2^i, \varphi_2^j) = -\sum_{k=1}^{N+1} \int_{x_{k-1}}^{x_k} \varphi_2^{i\prime} \varphi_2^j + \sum_{k=1}^{N} [[\varphi_2^{i\prime}]]\{\{\varphi_2^j\}\} - \sum_{k=1}^{N} \left[\left[\varphi_2^i\right]\right]\left\{\left\{\varphi_2^{j\prime}\right\}\right\} +$
$\sum_{k=1}^{N} \frac{\eta}{h} \left[\left[\varphi_2^i\right]\right]\left[\left[\varphi_2^j\right]\right]$
$= -\sum_{k=1}^{N+1} \int_{x_{k-1}}^{x_k} \varphi_2^{i\prime} \varphi_2^j + \sum_{k=1}^{N} 0 * 1 - \sum_{k=1}^{N} 0 * 0 + \sum_{k=1}^{N} \frac{\eta}{h} * 0 * 0$
$= -\sum_{k=1}^{N+1} \int_{x_{k-1}}^{x_k} \varphi_2^{i\prime} \varphi_2^j$

<u>Si i ≠ j</u>, $a(\varphi_2^i, \varphi_2^j) = 0$

<u>Si i = j,</u> $a(\varphi_2^i, \varphi_2^i) = -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} 0 * 1 \; = 0$

Ainsi, on a :

$$\boldsymbol{A_h} = (N+1) * \begin{pmatrix} N(1+\eta) & -(N+1) & N(1+\eta) & 0 & & N(1+\eta) & 0 & N(1+\eta) & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ N(1+\eta) & 0 & N(1+\eta) & -(N+1) & & N(1+\eta) & 0 & N(1+\eta) & 0 \\ 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ & \vdots & & & \ddots & & \vdots & & \\ N(1+\eta) & 0 & -(N+1) & 0 & & N(1+\eta) & -(N+1) & N(1+\eta) & 0 \\ 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ N(1+\eta) & 0 & -(N+1) & 0 & \cdots & N(1+\eta) & 0 & N(1+\eta) & -(N+1) \\ 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \end{pmatrix}$$

3. Partie code

3.1 Partie code pour la matrice de rigidité

Voici les lignes de code que nous avons écrites pour afficher la matrice de rigidité Ah

main.cpp

C: > Users > user > Documents > Projet M1 > main.cpp > main()

```cpp
#include <iostream>
#include <cmath>
#include "full.hpp"

using namespace std;

int main(){

    int N;
    double eta;

    cout<<"Entrer le nombre de points de discretisation voulus :"<<endl;
    cin>>N;
    N=N-2;

    double h=1./(N+1);

    cout<<"Entrer le coefficient de penalisation voulu :"<<endl;
    cin>>eta;

    FullMtx A(N*2);

    double moyenne_phi1 = 0;
    double moyenne_phiprime1 = 1/h;
    double moyenne_phi2 = 1;
    double moyenne_phiprime2 = 0;

    double saut_phi1 = -1;
    double saut_phiprime1 = 0;
    double saut_phi2 = 0;
    double saut_phiprime2 = 0;

    double A11 = N*saut_phiprime1*moyenne_phi1-N*saut_phi1*moyenne_phiprime1+N*(eta/h)*saut_phi1*saut_phi1;

    double trois_sommes12 = N*saut_phiprime1*moyenne_phi2-N*saut_phi1*moyenne_phiprime2+N*(eta/h)*saut_phi1*saut_phi2;
    double A12 = trois_sommes12-1/(pow(h,2));

    double trois_sommes21 = N*saut_phiprime2*moyenne_phi1-N*saut_phi2*moyenne_phiprime1+N*(eta/h)*saut_phi2*saut_phi1;

    double trois_sommes22 = N*saut_phiprime2*moyenne_phi2-N*saut_phi2*moyenne_phiprime2+N*(eta/h)*saut_phi2*saut_phi2;



    for (int ii=0; ii<2*N; ii++){
        for (int jj=0; jj<2*N; jj++){
            if (ii%2 == 0){
                if (jj%2 == 0){
                    A(ii,jj) = A11;
                }
                else {
                    if ((jj-1) == ii){
                        A(ii,jj) = A12;
                    }
                    else {
                        A(ii,jj) = trois_sommes12;
                    }
                }
            }
            else {
                if (jj%2 == 0){
                    A(ii,jj) = trois_sommes21;
                }
                else {
                    A(ii,jj) = trois_sommes22;
                }
            }
        }
    }

    cout<<"La matrice de discretisation Ah vaut :"<<endl;
```

```
68      }
69
70      cout<<"La matrice de discretisation Ah vaut :"<<endl;
71
72      for (int ii=0; ii<2*N; ii++){
73          for (int jj=0; jj<2*N; jj++){
74              cout<<A(ii,jj)<<"   ";
75          }
76          cout<<endl;
77      }
78  }
```

Ainsi, nous obtenons le résultat suivant dans le terminal :

```
PS C:\Users\user\Documents\Projet M1> g++ main.cpp full.cpp -o main
PS C:\Users\user\Documents\Projet M1> ./main
Entrer le nombre de points de discretisation voulus :
6
Entrer le coefficient de penalisation voulu :
2
La matrice de discretisation Ah vaut :
60   -25   60   0   60   0   60   0
0   0   0   0   0   0   0   0
60   0   60   -25   60   0   60   0
0   0   0   0   0   0   0   0
60   0   60   0   60   -25   60   0
0   0   0   0   0   0   0   0
60   0   60   0   60   0   60   -25
0   0   0   0   0   0   0   0
PS C:\Users\user\Documents\Projet M1>
```

3.2. Solveur pour le problème de Poisson Freefem++

3.2.1 Commandes pour Galerkin Discontinue

FreeFem++ permet d'utiliser des formulations de type Galerkin à l'aide des mots clés **jump**, **average**, **intalledges** et **nTonEdge**.

$$\textbf{intalledges(Th)(u)} = \sum_{T \in T_h} \int \boldsymbol{u}$$

Sur chaque arrête d'un triangle T ou bien lorsqu'on est sur un segment :

$$\textbf{jump(u)(x)} = \lim_{t \to 0+} \boldsymbol{u(x + tn) - u(x - tn)}$$

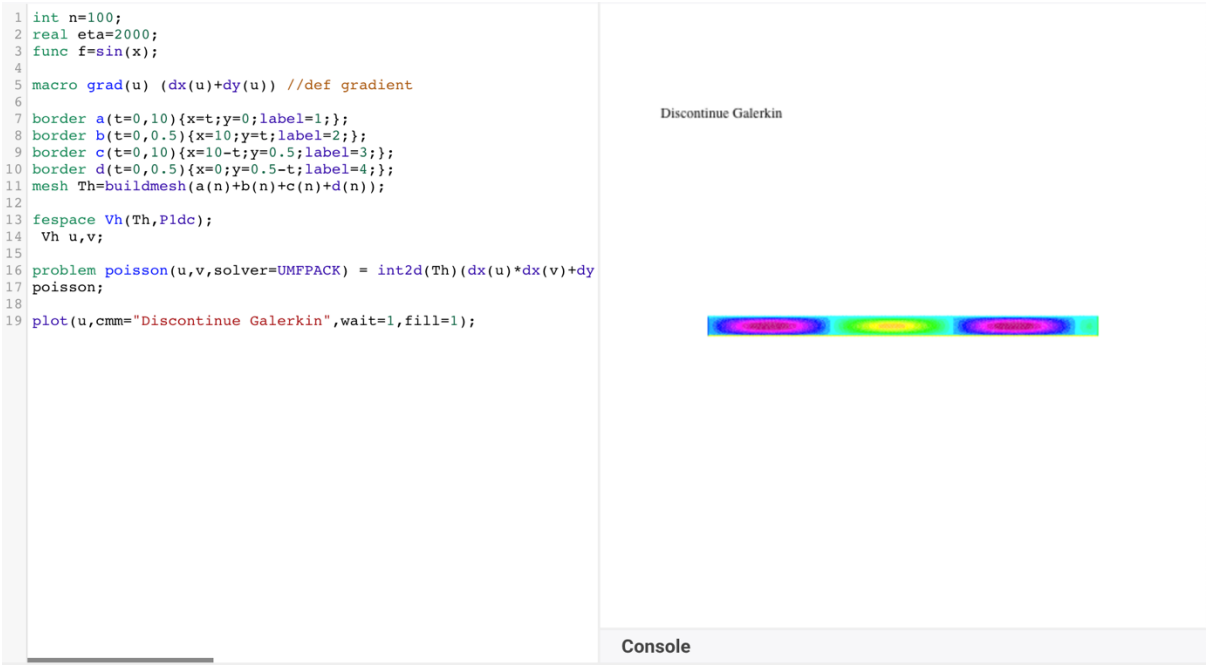
Où n est la normal extérieur au triangle T

$$\textbf{Average(u)(x)} = \lim_{t \to 0+} \big(\boldsymbol{u(x + tn) + u(x - tn)}\big)\textbf{/2}$$

nTonEdge = Nombre de triangles adjacents à l'arrête

3.2.2 Code

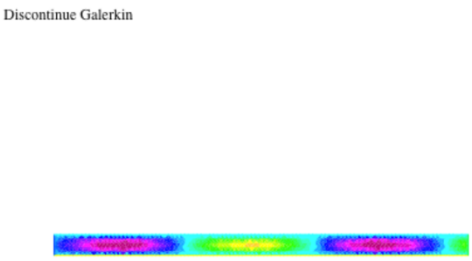
```
int n=100;
real eta=10000;
func f=sin(x);

macro grad(u) (dx(u)+dy(u));

border a(t=0,10){x=t;y=0;label=1;};
border b(t=0,0.5){x=10;y=t;label=2;};
border c(t=0,10){x=10-t;y=0.5;label=3;};
border d(t=0,0.5){x=0;y=0.5-t;label=4;};
mesh Th=buildmesh(a(n)+b(n)+c(n)+d(n));

fespace Vh(Th,P1dc);
 Vh u,v;

problem poisson(u,v,solver=UMFPACK) = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))+
intalledges(Th)(( jump(v)*mean(grad(u)) -  jump(u)*mean(grad(v)) + eta*jump(u)*jump(v) )
/ nTonEdge) - int2d(Th)(f*v);
poisson;

plot(u,cmm="Discontinue Galerkin",wait=1,fill=1);
```



Code du solveur et aspect de la solution

Remarque importante : Lorsque le coefficient de pénalisation est petit cela impacte la qualité de la solution, elle est moins régulière.

```
int n=100;
real eta=1;
func f=sin(x);

macro grad(u) (dx(u)+dy(u)) //def gradient

border a(t=0,10){x=t;y=0;label=1;};
border b(t=0,0.5){x=10;y=t;label=2;};
border c(t=0,10){x=10-t;y=0.5;label=3;};
border d(t=0,0.5){x=0;y=0.5-t;label=4;};
mesh Th=buildmesh(a(n)+b(n)+c(n)+d(n));

fespace Vh(Th,P1dc);
 Vh u,v;

problem poisson(u,v,solver=UMFPACK) = int2d(Th)(dx(u)*dx(v)+dy
poisson;

plot(u,cmm="Discontinue Galerkin",wait=1,fill=1);
```



Importance coefficient de pénalisation

Annexe

Cette annexe a pour but d'expliquer assez simplement mais de façon plus détaillé la méthode SIP introduite par Di Pietro dans son livre partie 4.2.

Pour commencer l'étude du problème de Poisson par la méthode de Galerkin discontinue SIP, on utilise une approximation non conforme puisque $V_h^p \subset P_d^k$. (notre espace discret n'est pas inclus dans notre de base).

Le but est d'approximer notre solution dans l'espace suivant P_d^k.
On impose une régularité forte sur notre solution afin d'avoir tous nos outils nécessaires sur les bons espaces telle que la convergence en nome ce qui nous permettre d'avoir si notre méthode est satisfaisante.
Ce jeu sur les espaces peut être traduit par une sorte d'équivalence entre problème continue, variationnelle et discret. Au final nous nous retrouverons avec un nouveau problème sur les bons espaces.

(PC) : Trouver u tel que $-\Delta u = f$
(PV) : Trouver $u \in V = H_0^1$ tel que $a(u,v)=l(v) \quad \forall v \in V$
(PD) : Trouver $u_h \in V_h$ tel que $a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h$

Nos nouveaux problème équivalent :

(PVbis) : Trouver $u \in V \cap H^2 = V_*$ tel que $a(u,v)=l(v) \ \forall v \in V_*$
(PDbis) : Trouver $u_h \in V_{*h}$ tel que $a(u_h, v_h) = l(v_h) \ \forall v_h \in V_{*h}$

Donc notre nouvelle forme bilinéaire est défini dans :

$$a_h : V_{*h} \times V_h \to \mathbb{R}$$
$$(u \ , \ v_h) \to a_h(u, \ v_h)$$

On veut que notre forme bilinéaire discrète soit consistante, dans le livre il est utilisé la technique appelé Heuristic Derivation. L'important à retenir ici est que l'on veut une forme consistante qui vérifie la définition (1.31) de livre que l'on rappelle ici :

> **Definition 1.31** (Consistency). We say that the discrete problem (1.26) is *consistent* if for the exact solution $u \in X_*$,
>
> $$a_h(u, w_h) = l_h(w_h) \qquad \forall w_h \in V_h. \tag{1.32}$$

Mais la forme bilinéaire à notre nouveau problème doit posséder les mêmes avantages que le problème original comme la symétrie de la forme bilinéaire discrète or le terme de consistance seul rompt la symétrie. Il faut aussi vérifier qu'il s'agit d'un problème bien posé. Le problème original l'était car le théorème de Lax-Milgram était vérifié ou l'on avait la coercivité, la continuité etc.
La méthode SIP permet de retrouver tous ces avantages, en ajoutant un terme symétrisant on conserve la symétrie de la forme originale et notre forme bilinéaire est toujours consistante. (notion très avantageuse car cela simplifie notre matrice et nos calculs à faire lors l'implémentation du code)

$$a_h^{\mathrm{cs}}(v,w_h) = -\sum_{T\in\mathcal{T}_h}\int_T(\triangle v)w_h + \sum_{F\in\mathcal{F}_h^i}\int_F[\![\nabla_h v]\!]\cdot\mathrm{n}_F\{\!\{w_h\}\!\} - \sum_{F\in\mathcal{F}_h}\int_F[\![v]\!]\{\!\{\nabla_h w_h\}\!\}\cdot\mathrm{n}_F.$$

On aimerait avoir la coercivité discrète sur notre espace de polynômes brisé par morceaux, pour cela on ajoute des termes de pénalité qu agissent comme un phénomène de compensation sur les formules de saut et moyenne sur les interfaces précédemment mises en place sur la forme bilinéaire consistante et symétrique.

$$a_h^{\mathrm{sip}}(v,w_h) := a_h^{\mathrm{cs}}(v,w_h) + s_h(v,w_h),$$

with the stabilization bilinear form

$$s_h(v,w_h) := \sum_{F\in\mathcal{F}_h}\frac{\eta}{h_F}\int_F[\![v]\!][\![w_h]\!],$$

Au final il reste à rajoute la partie consistante et symétrique à la partie stabilisée par le terme de pénalité.

$$a_h^{\mathrm{sip}}(v,w_h) = -\sum_{T\in\mathcal{T}_h}\int_T(\triangle v)w_h + \sum_{F\in\mathcal{F}_h^i}\int_F[\![\nabla_h v]\!]\cdot\mathrm{n}_F\{\!\{w_h\}\!\} - \sum_{F\in\mathcal{F}_h}\int_F[\![v]\!]\{\!\{\nabla_h w_h\}\!\}\cdot\mathrm{n}_F + \sum_{F\in\mathcal{F}_h}\frac{\eta}{h_F}\int_F[\![v]\!][\![w_h]\!].$$

Bibliographie :

1. [1] D.N. Arnold, An interior penalty finite element method with discontinuous elements, SIAM J. Numer. Anal. 19 (4) (1982) 742–760.
2. [2] G. Baker, Finite element methods for elliptic equations using nonconforming elements, Math. Comp. 31 (1977) 44–59.
3. [4] A. Ern, J.-L. Guermond, Discontinuous Galerkin methods for Friedrichs' systems. I. General theory, SIAM J. Numer. Anal. 44 (2) (2006)
4. D. A. Di Pietro and A. Ern, Mathematical Aspects of Discontinuous Galerkin Methods, Number 69 in Mathématiques & Applications, Springer, Berlin, 2011
5. Sophie Gérald. Méthode de Galerkin Discontinue et intégrations explicites-implicites en temps basées sur un découplage des degrés de liberté. Applications au système des équations de Navier-Stokes.. Mécanique des fluides [physics.class-ph]. Université Pierre et Marie Curie - Paris VI, 2013. Français.
6. http://www.cmap.polytechnique.fr/IMG/pdf/Setif-FreeFem_II.pdf, O. Pantz, M. Kallel