Institut Galilée : Mathématiques appliquées au calcul scientifique

Rapport de Stage

Troisième année

Méthode multigrille parallèle pour les éléments finis *hp*

Auteur:
Clément Maradei
Encadrant:
Zhaonan Dong
Martin Vohralík

Avril - Septembre 2023

Table des Matières

Liste des Figures

Remerciements

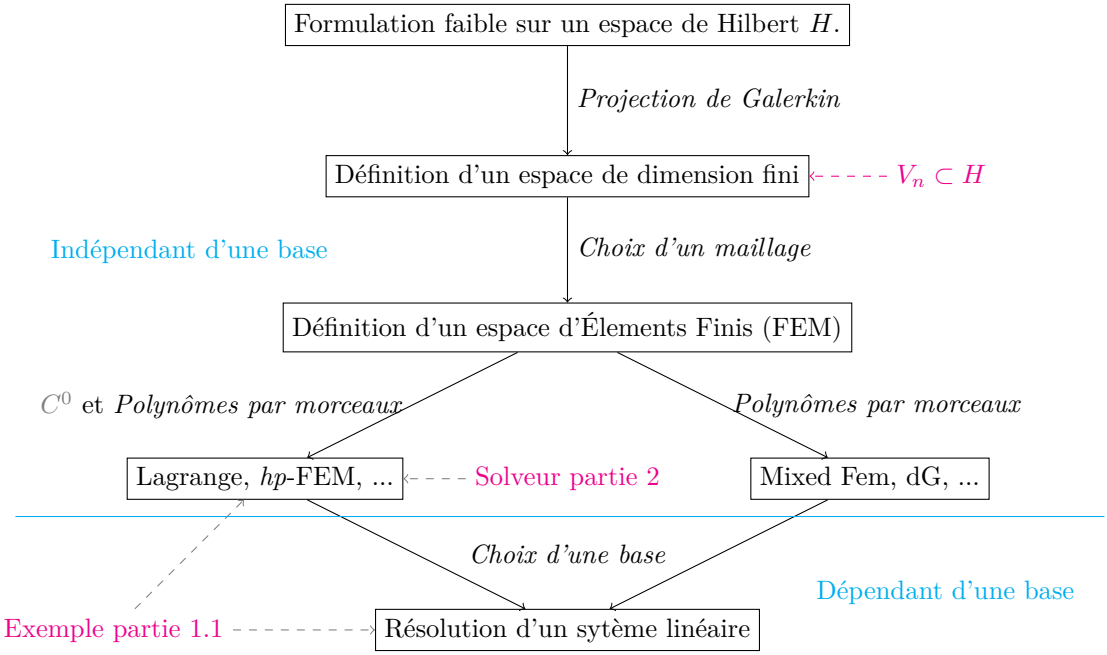
Plus tôt cette année, je réalisais mon premier stage dans le domaine de la recherche. J'ai eu la chance de rencontrer une équipe accueillante dans laquelle j'ai pu trouvé ma place. Cette expérience a été l'occasion pour moi de réaliser un premier travail de recherche, un domaine dans lequel je souhaite désormais poursuivre. Ce mémoire est le fruit de six mois de travail que j'ai réalisé sous la supervision de Zhaonan Dong et de Martin Vohralík, je les remercie pour leur implication, leurs encouragements ainsi que pour la confiance qu'ils m'accordent pour un projet de thèse. Ce travail achève également plusieurs années de formation durant lesquelles j'ai été encadré par une équipe enseignante investie et soucieuse de la réussite de chaque élève. Je souhaite remercier Marion Darbas, Hakim Boumaza et chaque membre de l'équipe pour leurs encouragements et leur bienveillance. Ils m'ont toujours soutenu et ont réussi à m'amener jusqu'au bout de ce projet d'études. Je leur témoigne toute ma reconnaissance.

Introduction

Les équations aux dérivées partielles sont omniprésentes dans les domaines scientifiques. Elles permettent de fournir une description mathématique à une multitude de phénomènes, qu'ils soient physiques ou qu'ils proviennent de considérations entièrement abstraites. Même pour les cas les plus simples, trouver une solution analytique pour de telles équations n'est généralement pas réalisable. En revanche, il est souvent possible d'en trouver une solution approchée à l'aide de méthodes numériques. Largement populaire, c'est dans cette démarche que s'inscrit la méthode des éléments finis. Quelle que soit la dimension d'espace considérée, son principe repose sur une décomposition du domaine en plusieurs éléments simples et l'introduction de fonctions de bases qui sont typiquement polynomiales. La solution approchée peut alors être exprimée comme une combinaison linéaire de ces fonctions, pondérée par des coefficients qu'il est nécessaire de déterminer, donnant lieu à la résolution d'un système linéaire. Dans l'optique de résoudre un tel système, il a été développé des méthodes appelées multigrilles, exploitant des informations issues d'une hiérarchie de maillage. C'est cette approche qu'on présente dans la partie 1. Dans la partie 2, on présente et motive une méthode de résolution introduite par M.Vohralík, J.Papež et A.Miraçi dans [6]. La partie 3 décrit le fonctionnement de notre code de calcul et la partie 4 présente les résultats numériques obtenus.

1 Méthodes de résolution numérique

Les méthodes numériques aboutissant au calcul d'une solution approché sont nombreuses, le diagramme non-exhaustif suivant contextualise certaines d'entre elles couramment utilisées.



Dans la suite de cette partie, on présente comment arriver à la résolution d'un système linéaire en suivant la méthode des Élements Finis de Lagrange, dans la partie 2 on présente cependant un solveur dont le principe s'applique pour n'importe quel espace d'Élements Finis dits conforme.

1.1 D'un problème continu à la résolution d'un système linéaire

La méthode des éléments finis permet de donner une solution approchée pour un grand nombre d'équation dont la solution analytique n'est pas connue. Il a été développé au fil des années des méthodes de calculs adaptées à la complexité de ces nombreux problèmes et de leurs spécificités intrinsèques. Dans cette partie, on se restreint à un cadre élémentaire pour mettre en lumière le principe fondamental de la méthode : le passage d'un problème continu à la résolution d'un système linéaire.

1.1.1 Problème de Poisson

Donnons dans un premier temps un exemple d'une équation aux dérivées partielles : le problème de Poisson. En considérant une fonction $f : \Omega \to \mathbb{R}$ donnée, on cherche la fonction $u : \Omega \to \mathbb{R}$ telle que

$$-\Delta u = f \quad \text{dans } \Omega \tag{1.1}$$
$$u = 0 \quad \text{sur } \partial\Omega \tag{1.2}$$

où Δ représente le Laplacien,

$$\Delta v := \sum_{i=1}^{d} \partial^2_{x_i} v.$$

1.1.2 Formulation variationnelle continue

Une formulation variationnelle pour ce problème est donné par :

$$\begin{cases} \text{Trouver } u \in H_0^1(\Omega) \text{ tel que} \\ \qquad\qquad \mathcal{A}(u,v) = \mathcal{L}(v) \quad \forall v \in H_0^1(\Omega) \end{cases} \tag{1.3}$$

Avec

$$\mathcal{A}(u,v) = \int_\Omega \nabla u(x) \nabla v(x) dx$$
$$\mathcal{L}(v) = \int_\Omega f(x) v(x) dx$$

où $\mathcal{A}$ est une forme bilinéaire continue et $\mathcal{L}$ une forme linéaire continue.

D'après le théorème de Représentation de Riesz, on sait qu'une unique solution faible existe cependant rien ne nous indique comment l'obtenir. En revanche, on peut établir une démarche simple afin d'en trouver une approximation.

1.1.3 Formulation variationnelle discrète

La méthode des éléments finis consiste tout d'abord à introduire une discrétisation de l'espace, un maillage. Considérons alors la triangulation du domaine suivante :

$$\Omega_h = \bigcup_{k=1}^{\mathrm{n_{me}}} K_k, \quad \text{avec } K_k \text{ un } d\text{-simplexe}$$

On suppose cette famille régulière, c'est-à-dire

$$\exists \sigma \geq 1 \text{ t.q } \max_{K \in \Omega_h} \frac{h_K}{\rho_K} \leq \sigma, \quad \forall h > 0.$$

où $h_K = \max\{|\mathbf{x} - \mathbf{y}|, (\mathbf{x}, \mathbf{y}) \in K^2\}$ et $\rho_k = \sup\{\mathrm{diam}(B), B \text{ une boule incluse dans } K\}$. Et on définit $h = \max_{K \in \Omega_h} h_K$.

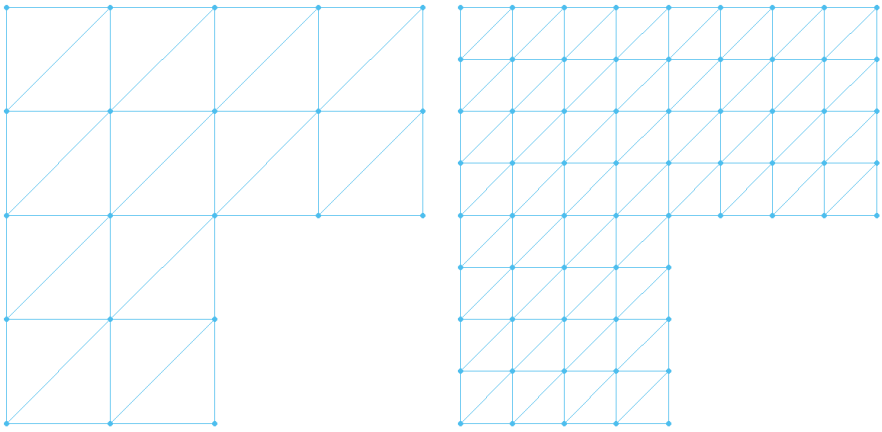


Figure 1: Maillage uniforme d'un domaine *L-shape domain*. À gauche 24 éléments, à droite 96.

Dès lors, on ne cherche plus u comme une solution analytique, on va se restreindre à déterminer une valeur approchée en chacun des noeuds du maillage, qu'on note $\mathbf{q^i}$ avec $i \in [1, \mathrm{n_q}]$. C'est sur ce nouveau domaine discret que nous allons pouvoir introduire l'espace des éléments finis. Pour le cas le plus simple, $\mathbb{P}_1$-Lagrange, il est défini comme :

$$V(\Omega_h) = \left\{ v \in \mathcal{C}^0(\bar{\Omega}) \text{ t.q } \forall k \in [1, \mathrm{n_{me}}], v_{|K_k} \in \mathbb{P}_1(K_k) \right\}.$$

On peut écrire $V(\Omega_h)$ à l'aide d'une base

$$V(\Omega_h) = \mathrm{Vect}(\phi_1, ..., \phi_{\mathrm{n_q}}).$$

Les fonctions ϕ_i sont les fonctions de bases de l'espace de discrétisation. Il est souhaitable qu'elles vérifient chacune la condition suivante :

$$\phi_i(\mathbf{q}^j) = \delta_{i,j}$$

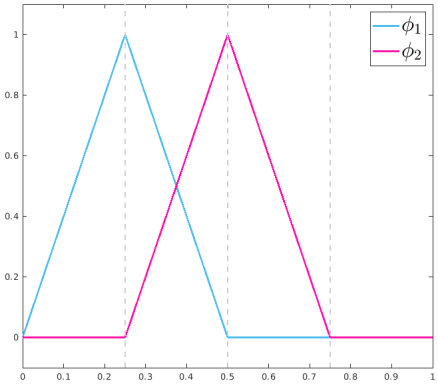


Figure 2: Exemples de fonctions de bases de l'espace V_h avec $d = 1$.

Il aurait été possible de faire un choix différent, seulement celui-ci permet de réduire grandement la complexité liée à la résolution du système linéaire introduit en 1.7.
Pour faire le lien entre l'espace continu et l'espace discret, il est nécessaire d'introduire un opérateur d'interpolation qu'on définit par :

Définition 1.1 (Opérateur d'interpolation)**.** *On peut définir l'opérateur π_h comme étant la fonction telle que :*

$$\begin{cases} \pi_h : \mathcal{C}^0(\bar{\Omega}) & \longrightarrow V(\Omega_h) \\ w & \longmapsto \pi_h(w) = \sum_{i=1}^{n_q} w(\mathbf{q}^i)\phi_i \end{cases}$$

qu'on peut restreindre au cadre de notre problème à :

$$\begin{cases} \pi_h : \mathcal{C}^0_*(\bar{\Omega}) & \longrightarrow V_h = V(\Omega_h) \cap H^1_0(\Omega) \\ w & \longmapsto \pi_h(w) = \sum_{i \in I} w(\mathbf{q}^i)\phi_i \end{cases}$$

Avec $I = \{i \in [1, n_q], \mathbf{q}^i \notin \partial\Omega\}$. $\mathcal{C}^0_*(\bar{\Omega})$ représente l'ensemble des fonctions de $\mathcal{C}^0(\bar{\Omega})$ nulles sur $\partial\Omega$.

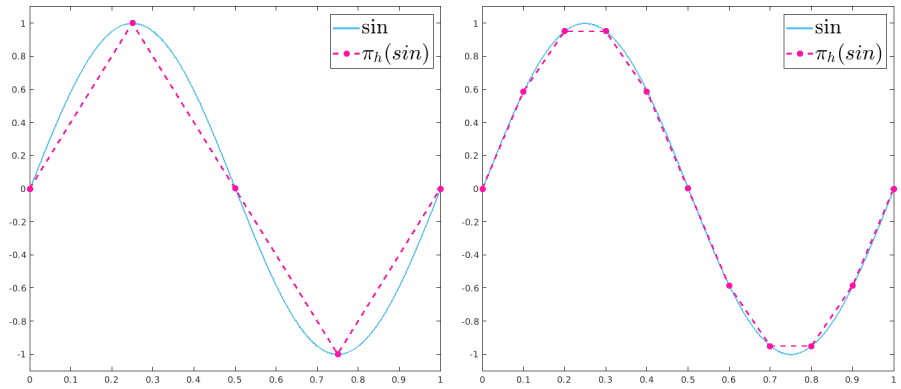


Figure 3: Représentation de la fonction sinus sur le domaine et de son interpolation. A gauche $n_q = 5$, à droite $n_q = 11$.

Cet interpolation est communément appelée l'interpolation de Lagrange. On considère des fonctions de $\mathcal{C}^0(\bar{\Omega})$. Il aurait été possible d'élargir le cadre en considérant des fonctions de $H^1(\Omega)$ en prenant non pas des valeurs aux sommets mais des moyennes sur les éléments (Interpolation de Clément).

En notant $u_h = \pi_h(u)$ et $v_h = \pi_h(v)$, on peut alors établir la formulation variationnelle discrète qui s'écrit :

$$\begin{cases} \text{Trouver } u_h \in V_h \text{ tel que} \\ \qquad \mathcal{A}_h(u_h, v_h) = \mathcal{L}_h(v_h) \quad \forall v_h \in V_h \end{cases} \tag{1.4}$$

Avec

$$\mathcal{A}_h(u_h, v_h) = \int_{\Omega_h} (\nabla u_h, \nabla v_h) dx$$
$$\mathcal{L}_h(v_h) = \int_{\Omega_h} f v_h dx$$

On note que l'existence et l'unicité de la solution s'obtient de la même manière que dans le cas continu, en utilisant le théorème de représentation de Riesz.

1.1.4 Convergence de la méthode

Dès lors que l'on cherche à résoudre un problème donnant une approximation de la solution, il est important de s'assurer que celle-ci est soit *suffisamment proche* de la solution exacte. Dans notre cas, les résultats suivants permettent d'établir ce caractère.

Lemme 1.1 (Lemme de Céa)**.** *Soit $u \in H_0^1(\Omega)$ et $u_h \in V_h$ respectivement les solutions de la formulation continue et discrète. Alors*

$$\|u - u_h\|_{H_0^1(\Omega)} = \min_{v_h \in V_h} \|u - v_h\|_{H_0^1(\Omega)}.$$

où la norme de $H_0^1(\Omega)$ est définie pour $v \in H_0^1(\Omega)$ par $\|v\|^2_{H_0^1(\Omega)} = \int_\Omega \nabla v(x)^2 dx$

Ce lemme valable pour toute dimension d'espace permet de donner le résultat suivant pour la méthode des éléments finis $\mathbb{P}_1$.

Théorème 1.1 (Convergence éléments finis $\mathbb{P}$)**.** *Soit $u \in H_0^1(\Omega)$ et $u_h \in V_h$ respectivement les solutions de la formulation continue et discrète.*
Alors il y a convergence, c'est-à-dire $\lim_{h\to 0} \|u - u_h\|_{H_0^1(\Omega)} = 0$. De plus on a si $u \in H^s(\Omega)$ pour $s \geq 1$,

$$\|u - u_h\|_{H_0^1(\Omega)} \leq C h^{\min\{p, s-1\}} \|u\|_{H^s(\Omega)}.$$

Une fois la convergence de la méthode établie, on cherche à en donner une formulation que l'on peut exploiter numériquement, c'est-à-dire sous la forme d'un système linéaire.

1.1.5 Construction d'un système linéaire

En considérant $(\phi_i)_{i\in I}$ les fonctions de base de l'espace V_h. Une formulation équivalente de (1.4) est donnée par :

$$\begin{cases} \text{Trouver } u_h \in V_h \text{ tel que} \\ \qquad \mathcal{A}_h(u_h, \phi_i) = \mathcal{L}_h(\phi_i) \quad \forall i \in I. \end{cases} \tag{1.5}$$

On peut alors écrire u_h comme sa décomposition dans la base de V_h, c'est-à-dire

$$u_h = \sum_{j\in I} u_j \phi_j$$

En remplaçant dans 1.5 et en utilisant la bilinéarité de $\mathcal{A}_h$, on obtient la relation suivante :

$$\mathcal{A}_h(u_h, \phi_i) = \mathcal{A}_h(\sum_{j\in I} u_j \phi_j, \phi_i) = \sum_{j\in I} \mathcal{A}_h(\phi_j, \phi_i) u_j.$$

En notant $A \in \mathcal{M}_{\text{card}(I)}$ la matrice définie par $A_{(i,j)} = \mathcal{A}_h(\phi_i, \phi_j)$ et $\mathbf{u} = (u_i)_{i\in I}$, on obtient l'égalité :

$$\mathcal{A}_h(u_h, \phi_i) = A_{(i,:)}\mathbf{u} \quad \text{où } A_{(i,:)} \text{ est la i-ème ligne de } A$$

La formulation variationnelle 1.5 s'écrit alors comme :

$$\begin{cases} \text{Trouver } \mathbf{u} \in \mathbb{R}^{\text{card}(I)} \text{ tel que} \\ \qquad A_{(i,:)}\mathbf{u} = \mathcal{L}_h(\phi_i) \quad \forall i \in I. \end{cases} \tag{1.6}$$

Et en définissant le second membre $\mathbf{f} \in \mathbb{R}^{\text{card}(I)}$ tel que $f_i = \mathcal{L}_h(\phi_i) \quad \forall i \in I$, on aboutit finalement à un problème qui consiste en la résolution d'un système linéaire :

$$\begin{cases} \text{Trouver } \mathbf{u} \in \mathbb{R}^{\text{card}(I)} \text{ tel que} \\ \qquad A\mathbf{u} = \mathbf{f}. \end{cases} \tag{1.7}$$

1.2 Méthodes de résolution de systèmes linéaires

Au vu des précédents résultats, il est légitime de se poser l'une des questions omniprésentes en analyse numérique. Comment résoudre ce système linéaire ? Dans notre cas et plus généralement pour les systèmes linéaires établies par la méthode des éléments finis, la matrice obtenue possède une propriété particulière : son caractère creux ou *sparse* pour certaines bases. Il est donc primordial de le prendre en compte et c'est d'ailleurs tout l'objet d'importants travaux de recherches depuis plus d'une cinquantaine d'années. On peut distinguer deux grandes familles de méthodes : les méthodes directs et les méthodes itératives. Les méthodes directs reposent sur le principe d'élimination de Gauss et permettent de déterminer la solution exacte d'un système (à la précision machine près). Elles reposent sur un nombre d'opérations arithmétiques fini et peuvent être extrêmement performantes dans des cadres restreints. Leur complexité en nombres d'inconnus s'expriment souvent en $O(n^2 \log n)$ pour des problèmes de taille $n \times n$. Les méthodes itératives ont quant à elles prouvé leur efficacité dans des cadres plus généraux. Leur principe repose sur un choix initial de solution qui s'améliore après chaque itération jusqu'à converger vers la solution exacte. L'utilisateur peut en principe choisir la tolérance qu'il souhaite imposer. Contrairement aux méthodes directes, la solution exacte du système n'est généralement pas obtenue.

1.2.1 Performances de solveurs communs pour l'équation de Poisson

Considérons alors le système linéaire obtenu en (1.7) par l'équation de Poisson en dimension 2 et regardons la performance (en temps) de plusieurs solveurs communément utilisés. Les résultats suivants sont obtenus en utilisant comme second membre la fonction f telle que $f(x) := \sin(2\pi x)\sin(2\pi y)$ sur le domaine $\Omega = [-1, 1]^2$.

Solveurs	**Directs**			**Itératifs**	
	LU	Cholesky	UMFPACK	CG	GMRES
Nombre d'éléments : 125000					
Erreur	6.69E-6	6.69E-6	6.69E-6	6.69E-6	6.70E-6
Temps d'exécution (s)	4.26	2.46	0.85	0.68	2.74
Nombre d'éléments : 500000					
Erreur	1.67E-6	1.67E-6	1.67E-6	1.68E-6	1.69E-6
Temps d'exécution (s)	65.21	34.53s	4.29	3.87	30.78
Nombre d'éléments : 2000000					
Erreur	4.18E-7	4.18E-7	4.18E-7	4.18E-7	4.48E-7
Temps d'exécution (s)	1215.73	558.91	26.282	20.33	183.35

Table 1: Comparaisons de différents solveurs pour l'équation de Poisson en dimension 2. L'erreur entre l'approximation u_h^i obtenue après convergence et la solution de l'EDP u est calculée avec la norme L^2. Les résultats sont obtenus avec FreeFem++.

On a représenté sur la gauche les résultats obtenus avec les solveurs directs et sur la droite les solveurs itératifs. UMFPACK est un ensemble de sous routines reposant sur le principe de méthodes multifrontales pour les problèmes non-symétriques. On peut observer une grande disparité quant à la performance de l'ensemble des solveurs. Il aurait été possible d'améliorer la performance de CG et GMRES en préconditionnant la matrice bien que l'idée principale reste de montrer que le choix de méthode de résolution joue un rôle plus qu'essentiel en termes

d'optimisation de performance. On a effectivement un gain considérable en utilisant la méthode du gradient conjugué et non une simple factorisation LU par exemple.

1.3 Solveurs itératifs

Les méthodes itératives proposent des résultats intéressants, dans cette partie on s'intéresse au fonctionnement de celles-ci en prenant comme exemple très simple la méthode de Jacobi. On considère un système linéaire issu de (1.7) de la forme :

$$A\mathbf{u} = \mathbf{f}$$

où $\mathbf{u}$ est la solution unique du système. On note $\mathbf{v}$ une approximation de $\mathbf{u}$.

1.3.1 Méthode de Jacobi

La méthode de Jacobi repose tout d'abord sur la décomposition de $\mathbb{A}$ telle que :

$$A = D - L - U$$

On écrit alors le système (1.7) comme

$$(D - L - U)\mathbf{u} = \mathbf{f} \tag{1.8}$$

qui s'écrit encore

$$D\mathbf{u} = (L + U)\mathbf{u} + \mathbf{f} \tag{1.9}$$

$$\mathbf{u} = D^{-1}(L + U)\mathbf{u} + D^{-1}\mathbf{f} \tag{1.10}$$

On définit alors la matrice d'itération de Jacobi par

$$R_J = D^{-1}(L + U)$$

Ce qui permet de caractériser chaque itération de la méthode comme la solution du système linéaire telle que

$$\mathbf{v}^{i+1} = R_J\mathbf{v}^i + D^{-1}\mathbf{f} \tag{1.11}$$

où l'on choisit comme vecteur initial $\mathbf{v}^0$. Il est intéressant d'écrire la méthode de Jacobi sous la forme

$$\mathbf{v}^{i+1} = \mathbf{v}^i + D^{-1}(\mathbf{f} - A\mathbf{v}^i) \tag{1.12}$$

Cette écriture permet d'expliciter de quelle manière est obtenue la mise à jour de la solution $\mathbf{v}^{i+1}$, on utilise l'itéré précédemment $\mathbf{v}^i$ auquel on ajoute une modification faite sur le résidu, ici l'application de D^{-1}. C'est avec cette formulation que l'on fera le lien dans la partie 2.

On peut alors mesurer l'erreur $e_i = \|\mathbf{v}^{i+1} - \mathbf{u}\|$. On dit que la méthode converge si pour tout vecteur initial $\mathbf{v}^0$, on a $\lim_{i\to\infty} \mathbf{e}_i = 0$ ce qui implique :

$$\mathbf{u} = D^{-1}\mathbf{f} + R_J\mathbf{u} \tag{1.13}$$

En utilisant (1.11) et (1.13), on obtient :

$$\mathbf{e}_i = \|\mathbf{v}^{i+1} - \mathbf{u}\| = \|R_j(\mathbf{v}^i - \mathbf{u})\| \tag{1.14}$$

Puis en remplaçant successivement de manière récursive, on obtient

$$\mathbf{e}_i = \|R_J^i(\mathbf{v}^0 - \mathbf{u})\| \tag{1.15}$$

On déduit de cette formule que la convergence n'a lieu que si $\lim_{i\to\infty} R_J^i$ donne la matrice nulle. On connaît une équivalence de cette propriété, plus simple à utiliser en pratique :

$$\lim_{i\to\infty} R_J = 0 \Leftrightarrow \rho(R_J) < 1, \tag{1.16}$$

avec ρ le rayon spectral. L'étude de la convergence peut alors se résumer à la détermination de valeurs propres.

1.3.2 Étude de la convergence sur un exemple simple

Considérons un système linéaire issu d'une méthode de différences finies pour le problème (1.1) avec un second membre nul. Ce choix est fait pour simplifier les résultats de cette partie, la solution d'un tel problème est clairement 0. On a la matrice de discrétisation $A \in \mathcal{M}_n$:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & -1 & 2 \end{bmatrix}$$

pour le système :

$$A\mathbf{u} = 0 \tag{1.17}$$

Avec $\mathbf{u} \in \mathbb{R}^n$. On propose de résoudre (1.17) avec la méthode de Jacobi modifiée qui est plus largement utilisée. Pour cela, on introduit un paramètre d'amortissement $\omega \in \mathbb{R}$, et il s'agit alors de trouver pour chaque itération :

$$\mathbf{v}^{i+1} = R_\omega \mathbf{v}^i + \omega D^{-1}\mathbf{f} \tag{1.18}$$

avec $R_\omega = (1-\omega)I + \omega R_J$. On note que le cas $\omega = 1$ correspond à la méthode classique, on se restreint par la suite au cadre $\omega = \frac{2}{3}$. De nombreuses méthodes itératives fonctionnent sur le même principe, c'est par exemple le cas de la méthode de Gauss-Seidel qui permet d'accroître la vitesse de convergence significativement. Dans notre cas la matrice R_ω s'écrit

$$R_\omega = I - \frac{\omega}{2}\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & -1 & 2 \end{bmatrix}$$

Et on a la relation suivante pour les valeurs propres de R_ω

$$\lambda(R_\omega) = 1 - \frac{\omega}{2}\lambda(A) \tag{1.19}$$

avec les valeurs propres de A

$$\lambda_k(A) = 4\sin^2\left(\frac{k\pi}{2n}\right), \qquad 1 \leq k \leq n-1 \tag{1.20}$$

On en déduit alors

$$\lambda(R_\omega) = 1 - 2\omega\sin^2\left(\frac{k\pi}{2n}\right), \qquad 1 \leq k \leq n-1 \tag{1.21}$$

Les vecteurs propres de R_ω sont donnés par

$$w_{k,j} = \sin\left(\frac{jk\pi}{n}\right), \qquad 1 \leq k \leq n-1, \quad 0 \leq j \leq n \tag{1.22}$$

Ils sont également appelés modes de Fourrier. On les classes en deux catégories, les modes à basse fréquence tels que $1 \leq k \leq \frac{n}{2}$ sont appelés modes réguliers et ceux à haute fréquence tels que $\frac{n}{2} \leq k \leq n$ sont appelés modes oscillatoires.

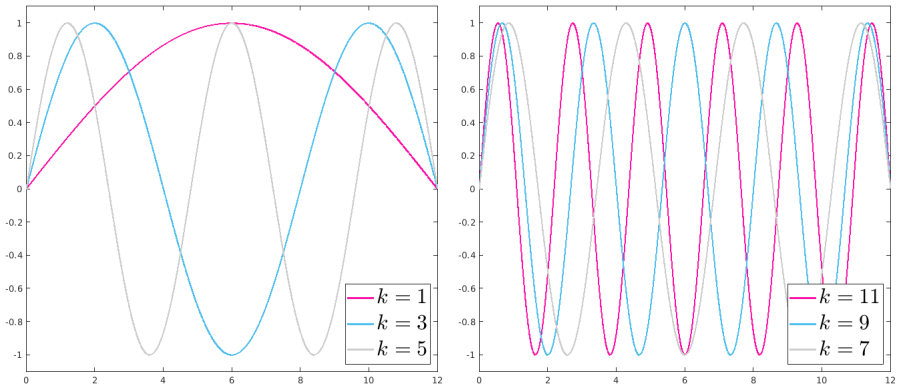


Figure 4: Représentation de différents modes de Fourrier. Pour $n = 12$, on a représenté à gauche des modes réguliers et à droite des modes oscillatoires.

Il est alors possible de décomposer n'importe quel vecteur comme une combinaison linéaire de ceux-ci. En particulier, l'erreur (1.14) initiale se décrit tel que :

$$\mathbf{e}^0 = \sum_{k=1}^{n-1} c_k \mathbf{w}_k, \tag{1.23}$$

où c_k sont les poids attribués pour chaque mode de Fourrier. Après m itérations, l'erreur s'écrit alors

$$\mathbf{e}^m = R_\omega^m \mathbf{e}^0. \tag{1.24}$$

En utilisant (1.23), on peut désormais établir une relation donnant des informations cruciales quant à l'erreur résiduelle après chaque itération. On a

$$\mathbf{e}^m = R_\omega^m \mathbf{e}^0 = \sum_{k=1}^{n-1} c_k R_\omega^m \mathbf{w}_k = \sum_{k=1}^{n-1} c_k \lambda_k^m(R_\omega)\mathbf{w}_k. \tag{1.25}$$

On voit ici qu'après m itérations la composante de l'erreur suivant le k-ème mode a été réduit d'un facteur $\lambda_k^m(R_\omega)$. Il est également important de noter que les modes associés aux valeurs propres les plus grandes donnent une composante de l'erreur plus importante. Désormais regardons comment réagit le solveur suivant les modes que comporte notre vecteur initial $\mathbf{u}^0$.

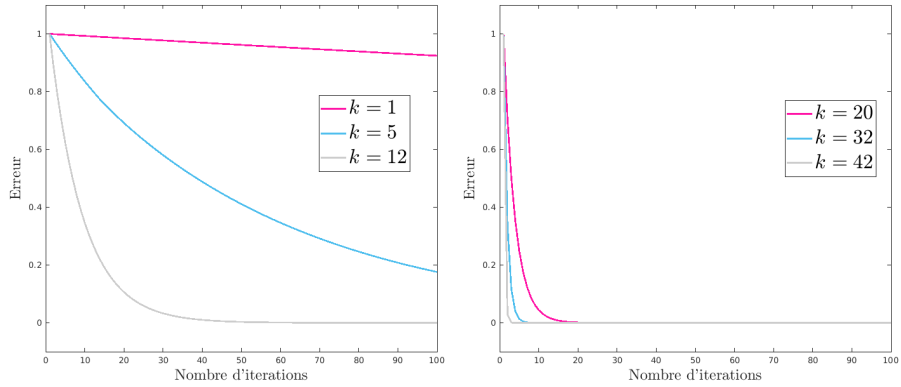


Figure 5: Représentation de l'erreur en fonction du nombre d'itérations pour le choix d'un vecteur initial $\mathbf{u}^0$ composé d'un mode k. Ici $n = 64$.

On observe sur la figure 5 un résultat important : Lorsque le vecteur $\mathbf{u}^0$ est composé de modes ayant un caractère oscillant plus fort, l'erreur atteint plus rapidement 0.
Il est important de comprendre qu'il a été fait un choix arbitraire de prendre $\mathbf{u}^0$ comme étant à chaque fois un seul mode distinct. En pratique et sur des cas plus complexes la décomposition de $\mathbf{u}^0$ et donc de l'erreur initiale n'est pas connue. La solution initiale supposée est en fait composée d'une multitude de modes différents.

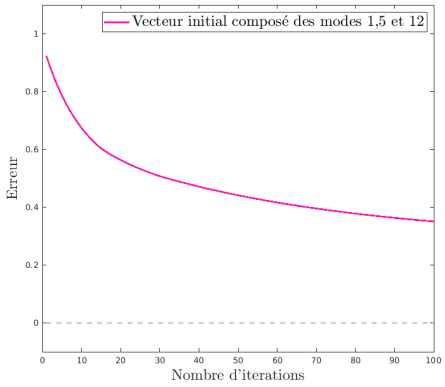


Figure 6: Représentation de l'erreur en fonction du nombre d'itérations pour le choix d'un vecteur initial $\mathbf{u}^0$ composé uniformément des modes $1, 5$ et 12. Ici $n = 64$.

On peut entrevoir une des limitations de la méthode. On sait d'après (1.25) que les modes dont la valeur propre associée est proche de 1 génèrent une composante de l'erreur importante. Or, d'après (1.19), on a l'expression suivante pour λ_1 :

$$\lambda_1 = 1 - 2\omega \sin^2\left(\frac{\pi}{2n}\right) = 1 - 2\omega \sin^2\left(\frac{\pi h}{2}\right) \approx 1 - \frac{\omega \pi^2 h^2}{2} \tag{1.26}$$

Il est clair d'après cette expression que la valeur propre associée au premier mode sera toujours proche de 1. Et on a de plus :

$$\lim_{h \to 0} \lambda_1(h) = 1. \tag{1.27}$$

On peut alors conclure de (1.27) que diminuer le pas de discrétisation dans une démarche d'approcher plus précisément la solution analytique va par la même occasion accroître l'erreur induite par la méthode de résolution numérique. Il en résulte que pour garder la même tolérance, le nombre d'itérations nécessaires, c'est-à-dire la quantité de ressources computationnelles, sera considérablement augmentée. C'est une limitation bien connue des solveurs itératifs classiques. L'un des intérêts principaux des solveurs multrigrilles réside dans le fait de pallier ce problème et obtenir un nombre d'itération pour une tolérance donnée indépendant du pas de discrétisation. On parle de *h-robustness*.

1.3.3 Solveurs multrigrilles

On a pu voir dans la partie 1.3.1 et sur la figure 5 que les méthodes itératives fonctionnaient bien pour atténuer les composantes de l'erreur issues des modes à haute fréquence (smoothing) mais étaient bien moins efficaces pour les modes à basse fréquence. Les composantes de l'erreur liées à ces modes sont plus difficiles à atténuer. Une piste pour améliorer la rapidité de convergence de la méthode serait d'avoir une idée "suffisamment bonne" de la solution et de s'en servir pour l'initialisation. Les solveurs multigrilles fonctionnent en introduisant une hiérarchie de maillage du domaine, allant du plus grossier au plus précis. L'idée est d'utiliser l'information acquise avec un plus faible coût de calcul sur des maillages plus *coarse* et de s'en servir par la suite sur le maillage le plus fin.
Regardons comment se comporte les modes dits réguliers sur des maillages plus grossiers. Pour ça, considérons Ω^{2h} un maillage comprenant seulement la moitié des sommets de Ω^h, par exemple les sommets pairs. On peut regarder comment se comporte le k-ème mode sur cette nouvelle grille. Pour $1 \leq k \leq \frac{1}{2}$, on a :

$$w^h_{k,2j} = \sin\left(\frac{2jk\pi}{n}\right) = \sin\left(\frac{jk\pi}{n/2}\right) = w^{2h}_{k,j}. \tag{1.28}$$

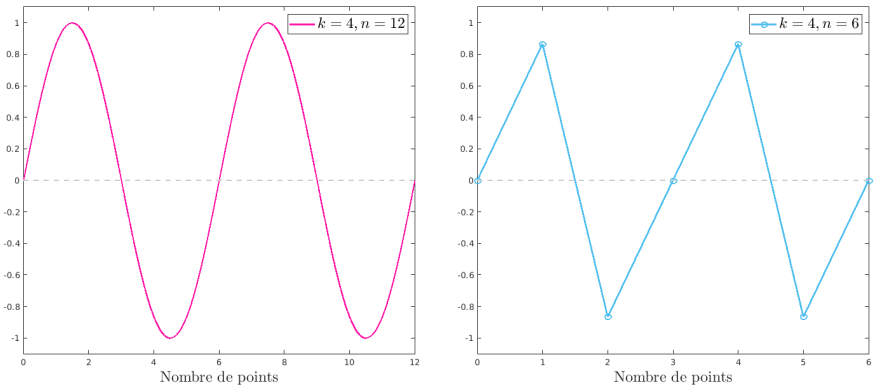


Figure 7: Représentation du mode $k = 4$ sur une grille où $n = 12$ à gauche et de son interpolation sur une grille où $n = 6$ à droite.

Sur la figure 7, on peut voir que lorsque $n = 6$, le mode apparaît plus oscillant que sur la grille où $n = 12$. Autrement formulé, un mode qui était auparavant régulier sur un maillage fin devient oscillatoire sur un maillage plus grossier. En considérant alors ce même maillage, on peut exploiter les résultats de la partie 1.3.1 et établir que notre méthode de résolution numérique sera plus performante en termes de nombre d'itérations dans ce cadre là. On peut alors mettre au point la stratégie dont le principe fonctionne de la manière suivante :

Algorithm 1: Méthode multigrille basique

Données: Une hiérarchie de maillage $\Omega_1, ..., \Omega_n$
Les matrices et vecteurs $A_1, ..., A_n$, $\mathbf{f}_1, ..., \mathbf{f}_n$
Un vecteur initial $\mathbf{v}_1$
Résultat: La solution approchée $\mathbf{u}_n$ sur le maillage Ω_n
for i *de* 1 *à* $n-1$ **do**
  Résoudre $A_i\mathbf{u}_i = \mathbf{f}_i$ sur Ω_i par une méthode numérique avec pour vecteur initial $\mathbf{v}_i$
  $\mathbf{v}_{i+1}$ devient l'interpolation de $\mathbf{u}_i$ sur Ω_{i+1}
end
Résoudre $A_n\mathbf{u}_n = \mathbf{f}_n$ sur Ω_n par une méthode numérique avec pour vecteur initial $\mathbf{v}_n$
return $\mathbf{u}_n$;

On a utilisé i comme indexe. $i = 1$ correspond au maillage le plus grossier et $i = n$ au maillage le plus fin. A_i et $\mathbf{f}_i$ sont respectivement la matrice de rigidité et le vecteur de second membre associés au maillages Ω_i.

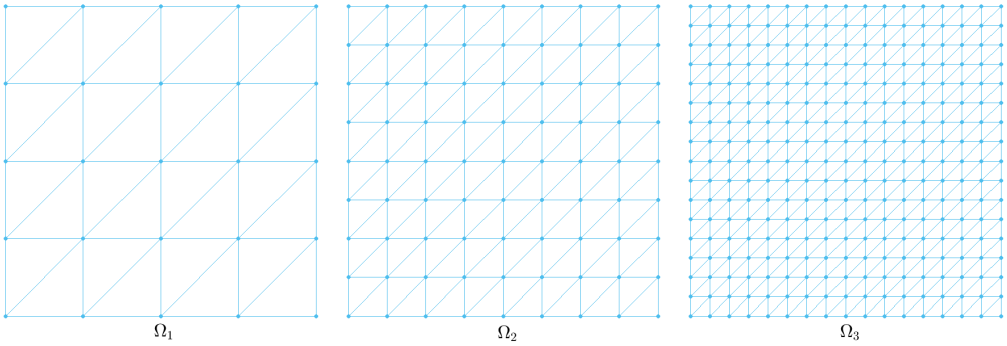


Figure 8: Exemple d'une hiérarchie de maillages uniformes pour un domaine carré. $d = 2$.

Algorithm 2: Méthode du schéma de correction sur deux grilles

Données: Les maillages Ω_1, Ω_2
Les matrices et vecteurs A_1, A_2, $\mathbf{f}_1, \mathbf{f}_2$
Un vecteur initial $\mathbf{v}_2$
Résultat: La solution approchée $\mathbf{u}_n$ sur le maillage Ω_n
Résoudre $A_2\mathbf{u}_2 = \mathbf{f}_2$ par une méthode itérative avec comme vecteur initial $\mathbf{v}_2$
Calculer le résidu $\mathbf{r_2} = \mathbf{f}_2 - A_2\mathbf{u}_2$
$\mathbf{r}_1$ devient la restriction de $\mathbf{r}_2$ sur Ω_1
    Résoudre l'équation résiduelle $\mathbf{A}_1\mathbf{e}_1 = \mathbf{r}_1$
    $\mathbf{e}_2$ devient l'interpolation de $\mathbf{e}_1$ sur Ω_2
$\mathbf{u}_2$ devient $\mathbf{u}_2 + \mathbf{e}_2$
Résoudre $A_2\mathbf{u}_2 = \mathbf{f}_2$ par une méthode itérative avec comme vecteur initial $\mathbf{u}_2$
return $\mathbf{u}_2$

La méthode de l'algorithme 1 propose de parcourir l'ensemble des maillages en allant du plus grossier au plus fin tandis que pour l'algorithme 2 on extrait tout d'abord une information sur le maillage le plus fin (le résidu) de façon à l'utiliser sur le maillage grossier (calcul de l'erreur). On revient ensuite sur le maillage le plus fin pour obtenir la dernière approximation. Ce même procédé peut être généralisé sur un nombre de grille arbitraire et est appelé *V-cycle* 1.3.4.

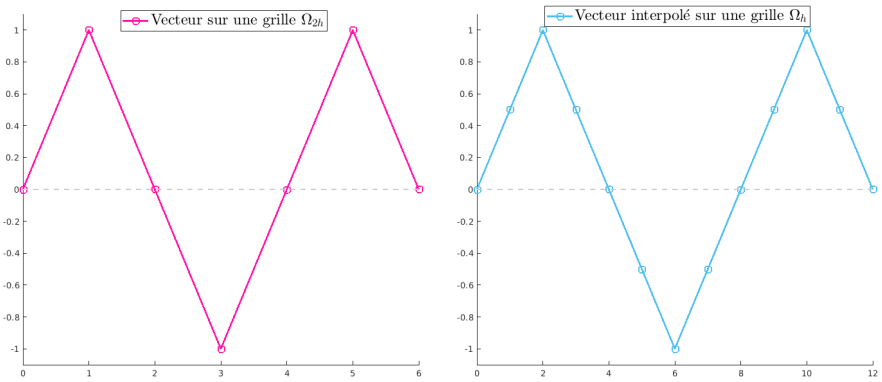


Figure 9: Représentation d'un vecteur sur une grille où $n = 6$ à gauche et de son interpolation sur une grille où $n = 12$ à droite. $d = 1$.

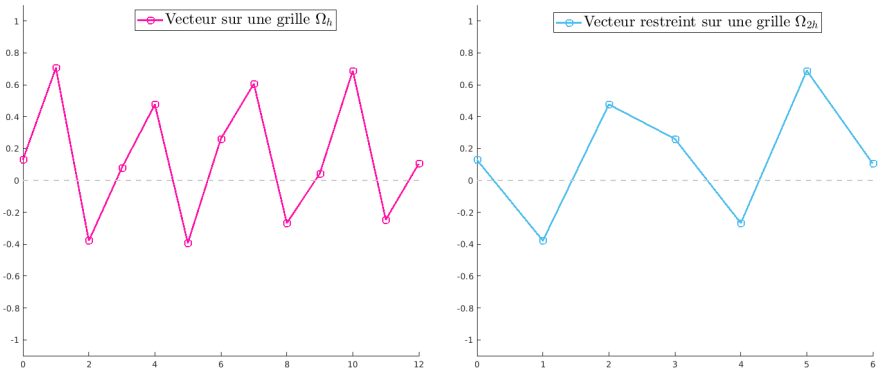


Figure 10: Représentation d'un vecteur sur une grille où $n = 12$ à gauche et de sa restriction sur une grille où $n = 6$ à droite. $d = 1$.

Sur la figure 9 et la figure 10, on a représenté des cas simples d'interpolation et de restriction. En dimension supérieure et pour des maillages issus de méthodes plus complexes, il est souvent compliqué d'implémenter de tels opérateurs.

1.3.4 V-Cycle

On peut voir le *V-cycle* comme la généralisation de la méthode présentée dans l'algorithme 2.

Algorithm 3: Algorithme *V-cycle*

Données: Une hiérarchie de maillage $\Omega_1, ..., \Omega_n$
Les matrices et vecteurs $A_1, ..., A_n$, $\mathbf{f}_1, ..., \mathbf{f}_n$
Les vecteurs initiaux $\mathbf{v}_1, ..., \mathbf{v}_n$
Les paramètres d'itérations ν_1, ν_2
Résultat: La solution approchée $\mathbf{u}_n$ sur le maillage Ω_n
for *i allant de n à 2* **do**
  Résoudre $A_i\mathbf{u_i} = \mathbf{f}_i$ sur Ω_i en itérant ν_1 fois avec comme vecteur initial $\mathbf{v}_i$
  Calculer le résidu $\mathbf{r}_i$
  $\mathbf{f}_{i-1}$ devient la restriction de $\mathbf{r}_i$ sur Ω_{i-1}
end
Résoudre de manière exacte $A_1\mathbf{u_1} = \mathbf{f}_1$
for *i allant de 1 à n-1* **do**
  $\mathbf{v}_i$ devient $\mathbf{u}_i$
  $I\mathbf{v}_{i+1}$ devient l'interpolation de $\mathbf{v}_i$ sur Ω_{i+1}
  $\mathbf{v}_{i+1}$ devient $\mathbf{v}_{i+1} + I\mathbf{v}_{i+1}$
  Résoudre $A_{i+1}\mathbf{u}_{i+1} = \mathbf{f}_{i+1}$ sur Ω_{i+1} en itérant ν_2 fois avec comme vecteur initial $\mathbf{v}_{i+1}$
end
return $\mathbf{u}_n$

L'algorithme 3 explique en quoi consiste un *V-cycle*. La première boucle for correspond au *pre-smoothing* tandis que la deuxième correspond au *post-smoothing*. On peut également donner une représentation schématique.

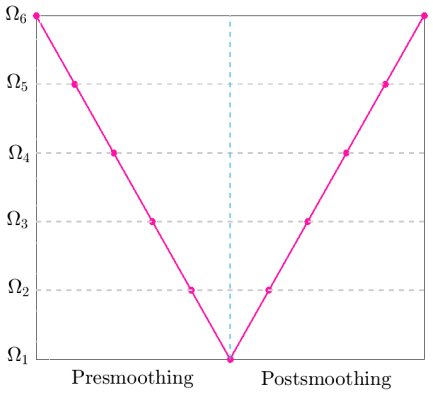


Figure 11: Schéma de fonctionnement du *V-cycle*.

2 Solveur multigrille *p-robust*

2.1 Principe de résolution performant et parallélisable

La méthode que nous introduisons s'inscrit dans la famille des méthodes multigrilles, elle a comme particularité de ne pas utiliser de presmoothing. Son principe est motivé par l'introduction d'une décomposition de l'erreur algébrique comme une somme orthogonale de ses composantes sur différents niveaux de maillages. Chaque itération repose alors sur le principe de recherche linéaire où pour chaque niveau de maillage la résolution d'un problème résiduel donne une direction de descente performante. On peut également établir un pas de descente λ pour lequel la convergence aura lieu plus rapidement. Dans un souci de performance, la résolution de chaque problème résiduel se fait à l'aide d'une décomposition par patch et la résolution de problèmes locaux entièrement parallélisables.

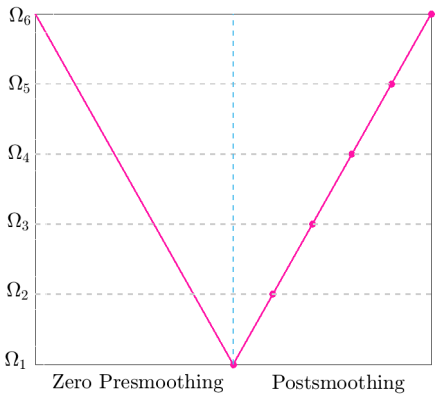


Figure 12: Schéma de fonctionnement du *V-cycle* sans *presmoothing*.

2.1.1 Le problème modèle

On considère un problème elliptique du deuxième ordre similaire à celui introduit dans la partie 1 sous une formulation différente. Ω désigne un ouvert de $\mathbb{R}^d, d \in \{1,2,3\}$. $f \in \mathrm{L}^2(\Omega)$ est le terme source, et $\mathcal{K} \in [\mathrm{L}^\infty(\Omega)]^{d\times d}$ est le coefficient de diffusion symétrique et défini positif. On s'intéresse à la solution $u \in H_0^1(\Omega)$ telle que

$$(\mathcal{K}\nabla u, \nabla v) = (f, v) \quad \forall v \in H_0^1(\Omega), \tag{2.1}$$

$(\cdot,\cdot)$ désigne le produit scalaire $\mathrm{L}^2(\Omega)$.
On introduit une discrétisation du problème en considérant $\mathcal{T}_J$ un maillage de Ω constitué de $d-$simplexes ainsi que l'espace V_J^p tel que

$$V_J^p = \mathbb{P}_p(\mathcal{T}_j) \cap H_0^1(\Omega), \tag{2.2}$$

avec $p \geq 1$ et $\mathbb{P}_p(\mathcal{T}_j) = \left\{v_J \in \mathrm{L}^2(\Omega), v_J|K \in \mathbb{P}_p(K) \quad \forall K \in \mathcal{T}_j\right\}$. Il s'agit alors de trouver $u_J \in V_J^p$ tel que

$$(\mathcal{K}\nabla u_J, \nabla v_J) = (f, v_J) \quad \forall v_J \in V_J^p. \tag{2.3}$$

On définit la fonctionnelle correspondant au résidu algébrique sur l'espace V_J^p du problème comme étant pour tout $u_J^i \in V_J^p$

$$v_J \mapsto (f, v_J) - \left(\mathcal{K}\nabla u_J^i, \nabla v_J\right) \in \mathbb{R}, \quad v_J \in V_J^p. \tag{2.4}$$

Cette description fonctionnelle du problème et non matricielle permet d'éviter l'introduction de bases pour les espaces d'éléments finis.

2.1.2 Structure du maillage et espaces de fonctions

On considère une hiérarchie de maillages $\{\mathcal{T}_j\}_{0\leq j\leq J}, J \geq 1$. $\mathcal{T}_J$ étant le maillage le plus fin, et $\mathcal{T}_0$ le maillage le plus grossier. On introduit également les espaces éléments finis associées à chacun des différents niveaux de raffinement. On établit alors pour $j \in \{0, ..., J\}, p_j$ le degré de polynôme associé au niveau j tel que $1 \leq p_0 \leq p_1 \leq ... \leq p_J = p$. Les espaces éléments finis sont alors définis tels que :

$$j = 0, \qquad V_0^1 = \mathbb{P}_1(\mathcal{T}_0) \cap H_0^1(\Omega), \tag{2.5}$$

$$1 \leq j \leq J, \qquad V_j^{p_j} = \mathbb{P}_{p_j}(\mathcal{T}_j) \cap H_0^1(\Omega). \tag{2.6}$$

où l'on a $\mathbb{P}_{p_j}(\mathcal{T}_j) = \left\{v_j \in \mathrm{L}^2(\Omega), v_j|K \in \mathbb{P}_{p_j}(k) \quad \forall K \in \mathcal{T}_j\right\}$.

On introduit également un concept clé pour la suite qui est la notion de patch. On en donne des illustrations sur la figure 13 et 14. Soit $V_j, 0 \leq j \leq J$, l'ensemble des sommets du maillage $\mathcal{T}_j$. Soit $\mathbf{a} \in V_J$, on note $\mathcal{T}_j^a$, tous les éléments $K \in \mathcal{T}_j$ qui ont comme sommet commun $\mathbf{a}$.

$$\mathcal{T}_j^a = \{K \in \mathcal{T}_j, \mathbf{a} \in V_K\} \tag{2.7}$$

Avec V_K l'ensemble des sommets de l'élément K. Le sous domaine du patch est noté $\omega_j^{\mathbf{a}}$. Les espaces éléments finis locaux sont alors notés :

$$V_j^{\mathbf{a}} = \mathbb{P}_{p_j}(\mathcal{T}_j) \cap H_0^1(\omega_j^{\mathbf{a}}). \tag{2.8}$$

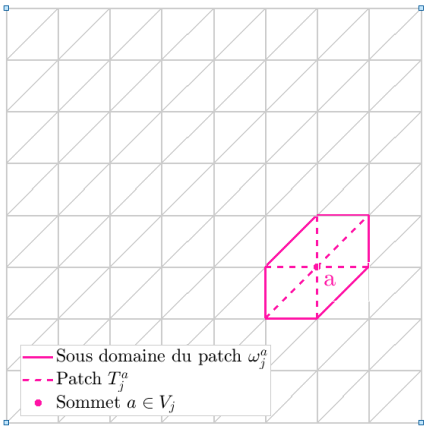


Figure 13: Représentation d'un patch sur un domaine carré $2d$ avec $p_j = 1$.

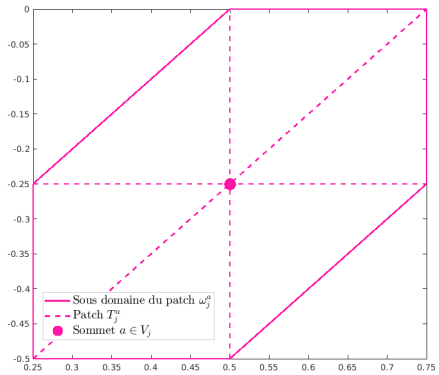


Figure 14: Représentation du patch de la figure 13.

2.1.3 Décomposition orthogonale de l'erreur suivant les niveaux du maillage

Dans cette section on propose une construction de l'erreur algébrique comme étant la décomposition orthogonale de ses composantes suivant les différents niveaux de maillages. C'est cette caractérisation qui inscrit notre approche dans la famille des méthodes multigrilles. L'idée étant de pouvoir travailler sur chacun des niveaux afin d'exploiter les informations liées à chaque composante.
Considérons $u_J^i \in V_J^p$ une approximation de la solution éléments finis $u_J \in V_J^p$ à l'itération i, on introduit la construction $\tilde{\rho}_{J,alg}^i \in V_J^p$ telle que :

$$\tilde{\rho}_{J,alg}^i = \rho_0^i + \sum_{j=1}^{J} \tilde{\rho}_j^i \tag{2.9}$$

où $\rho_0^i = \tilde{\rho}_0^i$ est la solution globale du problème résiduel sur le maillage le plus grossier, c'est-à-dire du problème :

$$\left(\mathcal{K}\nabla\rho_0^i, \nabla v_0\right) = (f, v_0) - \left(\mathcal{K}\nabla u_J^i, \nabla v_0\right), \quad \forall v_0 \in V_0^1 \tag{2.10}$$

De plus, on définit également $\tilde{\rho}_j^i \in V_j^{p_j}$, pour $1 \leq j \leq J$, comme étant la solution de

$$\left(\mathcal{K}\nabla\tilde{\rho}_j^i, \nabla v_j\right) = (f, v_j) - \left(\mathcal{K}\nabla u_J^i, \nabla v_j\right) - \sum_{k=0}^{j-1} \left(\mathcal{K}\nabla\tilde{\rho}_k^i, \nabla v_j\right), \quad \forall v_j \in V_j^{p_j}. \tag{2.11}$$

$\tilde{\rho}_{J,alg}^i$ peut alors s'exprimer comme étant $\tilde{\rho}_{J,alg}^i = u_J - u_J^i$, l'erreur entre u_J et u_J^i. De plus, en sommant (2.11) pour j de 0 à J, et en utilisant (2.1) pour le dernier terme de la somme on obtient alors la relation suivante :

$$u_J = u_J^i + \sum_{j=0}^{J} \tilde{\rho}_j^i \tag{2.12}$$

$\tilde{\rho}_{j,alg}^i$ satisfait alors la relation

$$\left(\mathcal{K}\nabla\tilde{\rho}_{J,alg}^i, \nabla v_J\right) = (f, v_J) - \left(\mathcal{K}u_J^i, \nabla v_J\right), \quad \forall v_J \in V_J^p. \tag{2.13}$$

$\tilde{\rho}^i_{J,alg}$ peut alors s'exprimer comme étant $\tilde{\rho}^i_{J,alg} = u_J - u^i_J$, l'erreur entre u_J et u^i_J. Pour finalement obtenir la décomposition souhaitée, il reste à montrer que $\left(\mathcal{K}\nabla\tilde{\rho}^i_j, \nabla\tilde{\rho}^i_k\right) = 0$ pour $0 \leq k, j \leq J$ avec $j \neq k$. En reprenant les résultats précédents on peut alors écrire :

$$\|\mathcal{K}^{\frac{1}{2}}\nabla\left(u_J - u^i_J\right)\|^2 = \|\mathcal{K}^{\frac{1}{2}}\nabla\tilde{\rho}^i_{J,alg}\|^2 = \sum_{j=0}^{J}\|\mathcal{K}^{\frac{1}{2}}\nabla\tilde{\rho}^i_j\|^2. \tag{2.14}$$

Ce résultat est crucial puisqu'il permet d'établir une démarche itérative **sur les niveaux** pour le calcul des composantes de l'erreur algébrique qui serviront de direction de descente pour notre méthode de résolution.

2.2 Solveur multigrille fonctionnant sur le principe de recherche linéaire

Pour réaliser une itération du solveur 4, on parcourt chacun des niveaux du maillage, puis on calcule une direction et un pas de descente de façon à mettre à jour la solution approchée. Il faut alors partir d'une solution initiale sur le maillage le plus grossier, chaque mise à jour de la solution sur le maillage j sera ensuite utilisée comme solution initiale sur le maillage $j+1$. L'intérêt de la décomposition de $\tilde{\rho}^i_{J,alg}$ introduite en 2.1.3 réside dans le fait de fournir une direction de descente performante pour chaque niveau : ρ^i_j. Cependant, il est important de noter que cette décomposition introduit la résolution de problèmes variationnels globaux pour le calcul de ρ^i_jsur chaque niveau de maillage. Pour $j = J$, résoudre (2.11) serait donc aussi coûteux que de déterminer u_J directement. Il est alors impératif de résoudre ces problèmes autrement. Pour cela, on décompose ρ^i_j suivant ses contributions sur chacun des patchs du maillage.

$$\rho^i_j := \sum_{\mathbf{a}\in V_j}\rho^i_{j,\mathbf{a}}. \tag{2.15}$$

où $\rho^i_{j,a}$ sont les solutions des problèmes

$$\left(\mathcal{K}\nabla\rho^i_{j,\mathbf{a}}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega^a_j} = \left(f, \nabla v^i_{j,\mathbf{a}}\right)_{\omega^a_j} - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega^a_j}, \quad \forall v_{j,\mathbf{a}} \in V^{\mathbf{a}}_j. \tag{2.16}$$

où $(\cdot,\cdot)_{\omega^a_j}$ représente le produit scalaire sur le domaine ω^a_j. Pour déterminer la direction de descente ρ^i_j, on résout alors plusieurs problèmes locaux et non un unique problème global. Le fonctionnement de la méthode est présenté dans l'algorithme suivant.

Algorithm 4: Solveur multigrille

1. Initialiser u_J^0 comme étant la fonction nulle dans l'espace V_J^p et i est égal à 0.

2. Effectuer les étapes (a)-(d)

(a) Déterminer ρ_0^i comme étant la solution du problème résiduel (2.11) :

$$\left(\mathcal{K}\nabla\rho_0^i, \nabla v_0\right) = (f, v_0) - \left(\mathcal{K}\nabla u_J^i, \nabla v_0\right) \quad \forall v_0 \in V_0^1.$$

λ_0^i est initialisé à 1 et on définit $u_{J,0}^i$ tel que :

$$u_{J,0}^i = u_J^i + \lambda_0^i \rho_0^i$$

(b) **Pour** j allant de **1** à **J** :

- Déterminer $\rho_{j,a}^i \in V_j^a$ comme solution des problèmes

$$\left(\mathcal{K}\nabla\rho_{j,\mathbf{a}}^i, \nabla v_{j,\mathbf{a}}^i\right)_{\omega_j^a} = \left(f, v_{j,\mathbf{a}}^i\right)_{\omega_j^a} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_{j,\mathbf{a}}^i\right)_{\omega_j^a} \quad \forall v_{j,\mathbf{a}} \in V_j^{\mathbf{a}}. \tag{2.17}$$

- Déterminer $\rho_j^i \in V_j^{p_j}$ tel que

$$\rho_j^i := \sum_{\mathbf{a}\in V_j} \rho_{j,\mathbf{a}}^i$$

- Déterminer λ_j^i le pas de descente optimal tel que

$$\lambda_j^i := \frac{\left(f, \rho_j^i\right) - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla\rho_j^i\right)}{\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2}. \tag{2.18}$$

- Déterminer $u_{J,j}^i$ tel que

$$u_{J,j}^i = u_{J,j-1}^i + \lambda_j^i \rho_j^i \tag{2.19}$$

Fin pour

(c) $u_J^{i+1} = u_{J,J}^i \in V_J^p$

(d) Si $u_J^{i+1} = u_J^i$, le solveur a convergé. Sinon $i = i + 1$ et on retourne à l'étape (a).

On propose de montrer que (2.18) est effectivement le choix de pas optimal. On souhaite obtenir à chaque itération une nouvelle solution approchée, la plus proche possible de la solution éléments finis u_J. Autrement dit, on cherche à ce que la quantité $\|\mathcal{K}(u_J - u_{J,j-1}^i)\|^2$ soit la plus petite possible. En utilisant (2.19), on a

$$\|\mathcal{K}\nabla(u_J - u_{J,j-1}^i)\|^2 = \|\mathcal{K}^{1/2}\nabla(u_J - (u_{J,j-1}^i + \lambda_j^i\rho_j^i))\|^2$$

. On peut alors écrire cette quantité comme étant une fonction de λ :

$$\begin{cases} J : \mathbb{R} & \longrightarrow \mathbb{R}^+, \\ \lambda & \longmapsto J(\lambda) := \|\mathcal{K}^{1/2}\nabla(u_J - (u_{J,j-1}^i + \lambda\rho_j^i))\|^2. \end{cases}$$

On cherche alors à minimiser cette fonction. On a $\|\mathcal{K}^{1/2}\nabla(u_J - (u_{J,j-1}^i + \lambda\rho_j^i))\|^2 = \|\mathcal{K}^{1/2}\nabla(u_J -$

$u_{J,j-1})\|^2 - 2\lambda\left(\mathcal{K}\nabla(u_J - u_{J,j-1}), \nabla\rho_j^i\right) + \lambda^2\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2$. La dérivée de la fonction J s'écrit telle que :

$$\begin{cases} J' : \mathbb{R} & \longrightarrow \mathbb{R}, \\ \lambda & \longmapsto J'(\lambda) := -2\left(\mathcal{K}\nabla(u_J - u^i_{J,j-1}), \nabla\rho_j^i\right) + 2\lambda\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2. \end{cases}$$

J étant une fonction convexe, elle réalise son unique minimum pour λ tel que $J'(\lambda) = 0$, c'est à dire pour

$$\lambda = \frac{\left(\mathcal{K}\nabla(u_J - u^i_{J,j-1}), \nabla\rho_j^i\right)}{\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2}$$
$$\lambda = \frac{\left(\mathcal{K}\nabla u_J, \nabla\rho_j^i\right) - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla\rho_j^i\right)}{\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2}$$

Et en utilisant (2.3), on trouve que l'unique minimum de J est réalisé pour

$$\lambda = \frac{\left(f, \rho_j^i\right) - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla\rho_j^i\right)}{\|\mathcal{K}^{1/2}\nabla\rho_j^i\|^2}.$$

2.2.1 Lissage des composantes hautes fréquences

Les problèmes (2.16) et l'obtention de ρ_j^i correspondent au lissage des modes à haute fréquence comme vu dans la partie 1.3.3. ρ_j^i, la correction apportée à u_j^i est obtenue en sommant ses contributions locales au patch. Cette obtention peut être vue comme une itération de la méthode de Jacobi pour $p_j = 1$ (généralisée avec block Jacobi pour $p_j > 1$). Pour $p = 1$, les degrés de liberté de $\rho_{j,a}$ sont des coefficients associés à des fonctions de bases nodales. Étant donné qu'un seul des sommets composant le patch n'est pas sur la frontière, le coefficient associé à celui ci sera le seul à être non nul.

2.2.2 *P*-robustesse du solveur multigrille

Le solveur multigrille présenté permet de choisir un degré polynomiale p_j de discrétisation éléments finis arbitraire, le résultat suivant fournit un caractère essentiel pour le solveur. Soit $u_J \in V_J^p$ la solution éléments finis satisfaisant (2.3), u_J^{i+1} et u_J^i les solution approchées à l'issue des l'itérations i et $i+1$ du solveur. En faisant une hypopthèse de régularité sur le maillage, c'est-à-dire, $\exists\kappa_\mathcal{T} > 0$ tel que

$$\max_{K\in\mathcal{T}_j} \frac{h_K}{\rho_K} \leq \kappa_\mathcal{T}, \quad \forall 0 \leq j \leq J$$

avec ρ_K le diamètre de la plus grand boule contenue dans K. On a alors :

$$\left\|\mathcal{K}^{\frac{1}{2}}\nabla\left(u_J - u_J^{i+1}\right)\right\| \leq \alpha\left\|\mathcal{K}^{\frac{1}{2}}\nabla\left(u_J - u_J^i\right)\right\|.$$

où $0 \leq \alpha \leq 1$ est un paramètre dépendant de la régularité du maillage $\kappa_\mathcal{T}$ et des valeurs propres du coefficients de diffusion $\mathcal{K}$. Ce résultat assure la p-robustesse du solveur, le solveur converge toujours aussi rapidement en augmentant la valeur de p_j. La précision de la solution et le nombre de degrés de liberté n'influent pas négativement le nombre d'itération nécessaire à la convergence.

2.2.3 Estimateur de l'erreur algébrique

L'erreur algébrique étant définit pour chaque itération par $\|\mathcal{K}^{\frac{1}{2}}(u_J - u_J^i)\|$. On définit η_{alg}^i, un estimateur de cette erreur par la formule

$$\eta_{\text{alg}}^i = \left(\sum_{j=0}^{J} (\lambda_j^i \|\mathcal{K}^{\frac{1}{2}} \nabla \rho_j^i\|)^2 \right). \tag{2.20}$$

On a le résultat suivant qui garantie l'efficacité de cet estimateur

$$\sqrt{1-\alpha^2}\|\mathcal{K}^{\frac{1}{2}}(u_J - u_J^i)\| \leq \eta_{\text{alg}}^i \leq \|\mathcal{K}^{\frac{1}{2}}(u_J - u_J^i)\|. \tag{2.21}$$

On a tout comme pour $\alpha \leq 1$, $\sqrt{1-\alpha^2} \leq 1$, ce qui assure la qualité de la borne inférieure comme de la borne supérieure. L'estimation n'est ni trop grande ou trop petite par rapport à la quantité désirée. En pratique il est important d'avoir ce résultat, il fournit une condition d'arrêt pour notre solveur. Effectivement, quand l'erreur algébrique devient plus petite que l'erreur de discrétisation il devient impossible d'améliorer la qualité de la solution obtenue. On l'illustre dans la partie 4.2.2.

3 Mise en pratique

3.1 Principe général

C'est donc sur la base de l'algorithme 4 que nous allons implémenter un code de calcul en utilisant Matlab. Son écriture étant encore loin des considérations pratiques, il est tout d'abord nécessaire d'interpréter cette méthode dans un langage se prêtant à cette optique. On peut tout d'abord établir le principe de fonctionnement suivant.

Algorithm 5: Principe de fonctionnement du code de calcul

Données: Terme source f.
Conditions au bords avec la fonction g.
Un maillage initial $\mathcal{T}_0$.
Une hiérarchie de maillage $\{\mathcal{T}_j\}_{1\leq j\leq J}$.
Ordre de la méthode p.
Résultat: Solution U sur le maillage $\mathcal{T}_J$.
for $j = 0 : J$ **do**
  Récupérer les informations de connectivités de $\mathcal{T}_j$
end
Initialiser U à 0.
while $\eta_{\text{alg}}^i <$ tol **do**
  Effectuer les étapes **2.**(a) à **2.**(d) de l'algorithme 4 pour obtenir U_{new}
    `// Correspond à une itération du solveur.`
  $\text{U} \leftarrow \text{U}_{new}$
end

On appelle de manière successive une fonction *solve*, laquelle prenant comme paramètres toutes les informations établies initialement ainsi que la solution à l'itération précédente.

Algorithm 6: Fonction *solve*

1. Résolution du problème résiduel sur $\mathcal{T}_0$.

• Assembler la matrice de masse et le vecteur de second membre correspondant au problème:

$$(\mathcal{K}\nabla\rho_0, \nabla v_0) = (f, v_0) - (\mathcal{K}\nabla u_J, \nabla v_0)\,, \quad \forall v_0 \in V_0^1$$

• Résoudre pour ρ_0 en utilisant la commande backslash de Matlab

• Projeter ρ_0 sur $\mathcal{T}_J$

• U_2 devient $\mathrm{U} + \rho_0$

2. Postsmoothing de la solution

Pour j allant de 1 à J faire

• Décomposer $\mathcal{T}_j$ comme un ensemble de patchs.

Pour k allant de 1 au nombre de patch faire

• Assembler la matrice et le vecteur de second membre correspondant à

$$\left(\mathcal{K}\nabla\rho^i_{j,\mathbf{a}}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a} = \left(f, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a} - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a}$$

• Résoudre pour $\rho_{j,a}$ en utilisant la commande backslash de Matlab

Fin pour

• ρ_j est obtenu en sommant toutes les contributions $\rho_{j,a}$.

• Projeter ρ_j sur $\mathcal{T}_J$

• Effectuer le calcul de λ_j par la formule

$$\lambda^i_j = \frac{\left(f, \rho^i_j\right) - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla\rho^i_j\right)}{\|\mathcal{K}^{1/2}\nabla\rho^i_j\|^2}$$

• Effectuer la mise à jour de la solution

$$\mathrm{U}_2 \to \mathrm{U}_2 + \lambda_j\rho_j$$

Fin pour

3.1.1 Représensation du vecteur solution

Considérons la solution du problème comme étant $u_h \in V_j^p$ qui peut s'écrire de la manière suivante :

$$u_h = \sum_{i=1}^{|V_j^p|} \alpha_i\phi_i \tag{3.1}$$

où $|V_j^p|$ est la dimension de V_j^p. On peut alors représenter la solution comme un vecteur de coefficients correspondants aux α_i. Pour les éléments finis d'ordre élevés on distingue les fonctions de bases ϕ_i suivant trois catégories : les fonctions *nodales*, *edge* et *modales*. On donne une représentation graphique des degrés de libertés pour un exemple simple où $p_j = 3$.

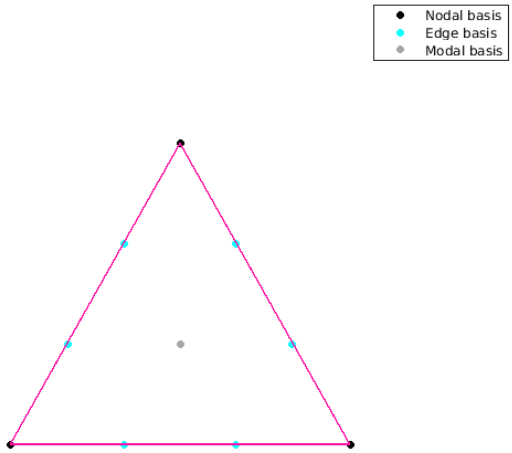


Figure 15: Degrés de liberté sur un élément pour $p_j = 3$

Structurellement, le vecteur solution stockera les coefficients correspondants aux degrés de libertés dans l'ordre nodal, edge puis modal.

3.1.2 Structures de données 2-D

Pour représenter les informations du maillage, on utilise deux matrices : $\texttt{node}(1:\texttt{N}, 2)$ et $\texttt{elem}(1:\texttt{NT}, 3)$, où N représente le nombre de sommets et NT représente le nombre de triangles. Chaque élément possède donc 3 sommets, représentés chacun par leur cordonnées carthésiennes dans le plan. Au sein d'un triangle, la numérotation des sommets est orienté dans le sens horaire contraire. On donne un exemple pour le domaine $\mathrm{L} = (-1,1) \times (-1,1) \backslash [0,1] \times [0,-1]$.

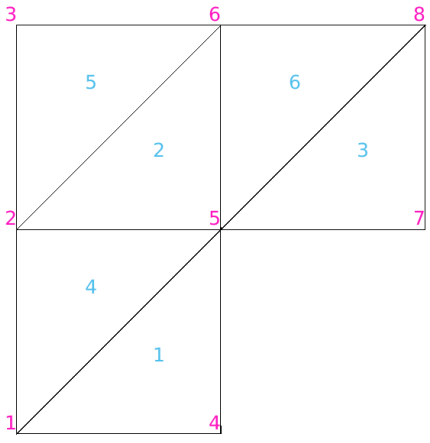


Figure 16: Représentation graphique du maillage pour le domaine L

-1	-1
-1	0
-1	1
0	-1
0	0
0	1
1	0
1	1

4	5	1
5	6	2
7	8	5
2	1	5
3	2	6
6	5	8

Table 2: Représentation des matrices `node` et `elem` pour le domaine L.

3.1.3 Assemblage de la matrice et du vecteur de second membre globaux pour le problème résiduel sur le maillage initial

Dans cette partie, on présente le code pour l'assemblage de la matrice et du vecteur de second membre pour le problème (2.10) :

$$\left(\mathcal{K}\nabla\rho_0^i, \nabla v_0\right) = (f, v_0) - \left(\mathcal{K}\nabla u_j^i, \nabla v_0\right) \quad \forall v_0 \in V_0^1. \tag{3.2}$$

```
% We build a C0 continuity FEM space: nodal + edge + modal

dim_elem = size(FEM_index,1);
NO_NodalBasis = size(Node,1);  NO_EdgeBasis = size(TotalEdge,1).*(Polydegree-1);
NO_ModalBasis = 0.5*(Polydegree-1)*(Polydegree-2).*NT;
dim_FEM = NO_NodalBasis+ NO_EdgeBasis +NO_ModalBasis;

%% Assembling for the global matrix and vector

i =zeros(dim_elem.^2,NT ); j =zeros(dim_elem.^2,NT); s =zeros(dim_elem.^2,NT );
i_vec =zeros(dim_elem,NT ); j_vec =ones(dim_elem,NT ); s_vec =zeros(dim_elem,NT
  );

for t =1:NT

  [local_elem] = Elem(t,:)';    [logl_index] = Checking_local_global_indicator(
  local_elem);
  local = localstiff(Node(local_elem,:),logl_index,Polydegree,Po,FEM_index,@(x)a
  (x));
  localvec = vect_forcing(Node(local_elem,:),logl_index,Polydegree,Po,FEM_index,
   @(x)f(x));
  nodal_index = local_elem;
  ind = findGlobalindex(nodal_index,Polydegree,TotalEdge,t,NO_NodalBasis,
  NO_EdgeBasis);

  %% Matrix

  i(:,t) = kron(ones(dim_elem,1),ind )  ;
  j(:,t) = kron(ind , ones(dim_elem,1)) ;
  s(:,t) = local(:);

  %% Forcing vector

  i_vec(:,t) = ind ;
  s_vec(:,t) = localvec;

end
```

```
B = sparse(i(:),j(:) ,s(:) ,dim_FEM,dim_FEM );
L = sparse(i_vec(:),j_vec(:) ,s_vec(:)  ,dim_FEM,1 );


% After the first iteration we build the second part of the right hand side.

if nbit > 1
  [rhs_second_part] = compute_rhs_second_part_J(Elem,Node,nodeMesh,NTMesh,
  bdEdgeMesh,intEdgeMesh,elemMesh,NT,Polydegree,FEM_index,U_previous_iteration_J
  ,J,0,Po);
  L2 = sparse(i_vec(:),j_vec(:), rhs_second_part(:),dim_FEM,1 );
  L = L - L2;
end
```

Listing 1: Assemblage de la matrice et du vecteur de second membre

La matrice globale est obtenue en assemblant les matrices locales correspondant à chaque élément. Les informations `dim_elem`, `NO_NodalBasis`, `NO_EdgeBasis`, `NO_NodalBasis`, `dim_FEM` sont générées en amont de la boucle `for`. Les matrices et vecteurs locaux sont obtenus respecti--vement avec les fonctions `localstiff` et `vect_forcing`. Étant donné qu'aucun des appels à ces fonctions n'ont de dépendances entre eux, la boucle `for` itérant sur les éléments peut être entièrement parallélisée. La matrice ainsi que le vecteur de second membre sont ensuite générés à l'aide de la commande `sparse` de Matlab pour un gain d'efficacité. Lors de la première itération du solveur, u_J^0 est initialisé à 0, il n'est alors pas nécessaire de faire appel à la fonction calculant la deuxième partie du second membre.

3.1.4 Résolution des problèmes résiduels sur les patchs pour chaque itération

Le code de résolution pour effectuer l'assemblage correspondant au problème (2.16), c'est à dire à l'équation

$$\left(\mathcal{K}\nabla\rho^i_{j,\mathbf{a}}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a} = \left(f, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a} - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a}, \quad \forall v_{j,\mathbf{a}} \in V_j^{\mathbf{a}}, \tag{3.3}$$

fonctionne de manière similaire au code établit précédemment. L'assemblage de ces matrices intervient pour chaque maillage $\mathcal{T}_j$, il est nécessaire de connaître $u^i_{J,j-1}$ pour assembler la partie correspondant à $\left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega_j^a}$.

```
for k = 1 : N_patch

  Star_cells_index = node2Star_Cell{k}; % Gives the indexes of the elements in
  patch k.
  NO_elem_patch = size(Star_cells_index,1); % Gives the number of element inside
   the patch k
  i = zeros(dim_elem_patch.^2, NO_elem_patch); j =zeros(dim_elem_patch.^2,
  NO_elem_patch); s =zeros(dim_elem_patch.^2, NO_elem_patch );
  i_vec =zeros(dim_elem_patch, NO_elem_patch ); j_vec =ones(dim_elem_patch,
  NO_elem_patch); s_vec =zeros(dim_elem_patch,NO_elem_patch);
  local_rhs_second_part = zeros(dim_elem_patch,NO_elem_patch);

  for t = 1 : NO_elem_patch

    [local_elem] = Elem(Star_cells_index(t),:)';
    [logl_index] = Checking_local_global_indicator(local_elem);
    local = localstiff(Node(local_elem,:),logl_index,Polydegree,Po,
    FEM_index,@(x)a(x));
    localvec = vect_forcing(Node(local_elem,:),logl_index,Polydegree,Po,
```

```
    FEM_index, @(x)f(x));
    nodal_index = local_elem;
    ind = findGlobalindex(nodal_index,Polydegree,TotalEdge,Star_cells_index(t),
    NO_NodalBasis,NO_EdgeBasis);
    [new_indexes_element] = old_to_new_map(Star_cells_index(t),:);
    nb_new_elements = length(new_indexes_element);

    %% Matrix

    i(:,t) = kron(ones(dim_elem,1),ind ) ;
    j(:,t) = kron(ind , ones(dim_elem,1)) ;
    s(:,t) = local(:);

    %% First part of the right hand side

    i_vec(:,t) = ind ;
    s_vec(:,t) = localvec;

    %% Second part of the right hand side

    for l=1:nb_new_elements
      new_elem = elemJ(new_indexes_element(l),:)';
      new_elem_nodes = nodeJ(new_elem',:);
      new_nodal_index = new_elem;
      [new_logl_index] = Checking_local_global_indicator(new_elem);
      new_ind = findGlobalindex(new_nodal_index,Polydegree,TotalEdge_J,
      new_indexes_element(l),NO_NodalBasis_J,NO_EdgeBasis_J);
      coef = U(new_ind);
      local_rhs_second_part(:,t) = local_rhs_second_part(:,t) +
      compute_inner_product_fine_element(Node(local_elem,:),new_elem_nodes,
      coef ,new_logl_index,logl_index, Polydegree,Po,FEM_index);
    end
  end

  B_patch = sparse(i(:),j(:) ,s(:) ,dim_FEM,dim_FEM);
  L_patch = sparse(i_vec(:),j_vec(:) ,s_vec(:)  ,dim_FEM,1 );
  L_patch2 = sparse(i_vec(:),j_vec(:),local_rhs_second_part(:),dim_FEM,1);
  L_patch = L_patch - L_patch2;
end
```

Listing 2: Assemblage de la matrice et du vecteur de second membre sur un patch

La résolution des problèmes sur chacun des patchs ne demandent pas d'informations préalables et sont donc indépendants entre eux. Il est alors possible de paralléliser la première boucle `for` itérant sur les patchs. La structure de donnée `node2Star_Cell` est générée en amont et donne l'indice de chaque élément au sein d'un même patch. C'est en parcourant ces éléments qu'il est possible d'assembler les contributions de chacun d'entre eux pour les matrices et la première partie de chacun des vecteurs locaux.

3.1.5 Calcul de λ pour chaque niveau

Après avoir déterminé ρ_j, pour effectuer la mise à jour de la solution, on propose de choisir λ donné par la formule vu précédemment (2.18) :

$$\lambda_j^i = \frac{(f, \rho_j^i) - (\mathcal{K}\nabla u_{J,j-1}^i, \nabla \rho_j^i)}{\|\mathcal{K}^{1/2}\nabla \rho_j^i\|^2} \tag{3.4}$$

```
numerator_vect = zeros(1,NT);
denominator_vect = zeros(1,NT);
for t=1:NT
    [new_indexes_element] = old_to_new_map(t,:);
    nb_new_elements = length(new_indexes_element);
    [old_elem] = Elem(t,:)';
    old_nodes = Node(old_elem,:);
    old_nodal_index = old_elem;
    [old_logl_index] = Checking_local_global_indicator(old_elem);
    old_ind = findGlobalindex(old_nodal_index,Polydegree,TotalEdge,t,
    NO_NodalBasis,NO_EdgeBasis);
    grad_uJ_grad_rho = 0;
    for i = 1:nb_new_elements
        new_elem = elemJ(new_indexes_element(i),:)';
        new_nodes = nodeJ(new_elem',:);
        new_nodal_index = new_elem;
        [new_logl_index] = Checking_local_global_indicator(new_elem);
        new_ind = findGlobalindex(new_nodal_index,Polydegree,TotalEdge_J,
        new_indexes_element(i),NO_NodalBasis_J,NO_EdgeBasis_J);
        U_coef = U(new_ind);
        rho_coef = Rho_global(new_ind);
        grad_uJ_grad_rho = grad_uJ_grad_rho +
        compute_grad_uJ_grad_rho(old_nodes,new_nodes,rho_coef,U_coef,
        old_logl_index,
        new_logl_index,Polydegree,Po,FEM_index,a);

    end
    rho_old_coef = Rho_global(old_ind);
    [local_num, local_dem] = compute_lambda_J(old_nodes,rho_old_coef,
    old_logl_index, Polydegree,Po,FEM_index,f,a,grad_uJ_grad_rho);
    numerator_vect(1,t) = local_num;
    denominator_vect(1,t) = local_dem;

end
lambda = sum(numerator_vect)/sum(denominator_vect);
```

Listing 3: Calcul de λ

Le calcul de λ se fait en itérant sur chacun des éléments du maillage et est décomposé en un calcul de numérateur et de dénominateur. Une nouvelle fois, la boucle `for` peut être parallélisée et la valeur de λ est finalement obtenue après celle-ci.

3.2 Amélioration des performances de calculs

Dans cette partie on propose une autre approche pour l'implémentation de l'algorithme 4. La méthode présentée précédemment fait intervenir la résolution de problèmes définis sur des patchs d'éléments. Dans un souci de performance et d'économie de temps de calcul et de mémoire, on se restreint à la résolution de problèmes sur les éléments.

3.2.1 Résolution de problèmes sur chaque élément du maillage

L'algorithme 4 fait donc intervenir sur chaque maillage $\mathcal{T}_j$ la résolution des problèmes : où l'on cherche $\rho^i_{j,a} \in V^a_j$ comme solution de

$$\left(\mathcal{K}\nabla\rho^i_{j,\mathbf{a}}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega^a_j} = \left(f, v^i_{j,\mathbf{a}}\right)_{\omega^a_j} - \left(\mathcal{K}\nabla u^i_{J,j-1}, \nabla v^i_{j,\mathbf{a}}\right)_{\omega^a_j}, \quad \forall v_{j,\mathbf{a}} \in V^{\mathbf{a}}_j. \tag{3.5}$$

Ce problème est bien posé, il admet une unique solution par le théorème de Lax-Milgram. La définiton de $V^{\mathbf{a}_j}$ comme étant $V^{\mathbf{a}}_j = \mathbb{P}_{p_j}(\mathcal{T}_j) \cap H^1_0(\omega_{j,\mathbf{a}})$ impose des conditions de Dirichlet homogènes sur les frontières du patch. On aimerait alors se restreindre à la résolution de problèmes sur un seul élément K. Pour celà, on va tout de même conserver la structure de patch et mettre au point une démarche séquentielle itérant sur chaque élément. Elle repose sur un ordre de résolution prédéfini ainsi que des conditions aux bords adaptées. Pour la mettre au point, il est nécessaire de distinguer les patchs en deux catégories : les patchs où $\mathbf{a}$ est situé à l'intérieur du maillage et les patchs où $\mathbf{a}$ est situé sur la frontière du domaine. Par la suite on désigne par V_n l'ensemble tel que

$$\begin{aligned}
V_1 &= \left\{v \in H^1(K_1), v_{|\partial K_1 \cap \partial\omega_a} = 0\right\} \\
V_n &= \left\{v \in H^1(K_n), v_{|\partial K_n \cap \partial\omega_a} = 0 \text{ et } v_{|\partial K_n \cap \partial K_{n-1}} = \rho^{i,n-1}_{j,|\partial K_n \cap \partial K_{n-1}}\right\} \\
V_N &= \left\{v \in H^1(K_N), v_{|\partial K_N \cap \partial\omega_a} = 0, v_{|\partial K_N \cap \partial K_{N-1}} = \rho^{i,N-1}_{j,|\partial K_N \cap \partial K_{N-1}}, v_{|\partial K_N \cap \partial K_1} = \rho^{i,1}_{j,|\partial K_N \cap \partial K_1}\right\}.
\end{aligned}$$

$\forall n, 2 \leq n \leq N-1$. Ce sont ces ensembles qui nous permettent d'écrire les différents problèmes presque sous forme variationnelle. La procédure est décrire par les algorithmes 7 et 8 de la page suivante.

Pour $\mathcal{T}_j^a$, un patch intérieur composé de N éléments on opère de la manière suivante :

Algorithm 7: Calcul de $\rho_{j,\mathbf{a}}^i$ pour un patch intérieur

1. Résolution sur le premier élément du patch .

• Résoudre pour $\rho_j^i \in \mathbb{P}_{p_j} \cap V_1$

$$\left(\mathcal{K}\nabla\rho_j^{1,i}, \nabla v_j^i\right)_{K_1} = \left(f, v_j^i\right)_{K_1} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_j^i\right)_{K_1}, \quad \forall v_j^i \in V_1. \tag{3.6}$$

2. Résolution jusqu'à l'avant dernier élément du patch

Pour n allant de 2 à $N-1$ faire

• Résoudre pour $\rho_j^i \in \mathbb{P}_{p_j} \cap V_n$

$$\left(\mathcal{K}\nabla\rho_j^{n,i}, \nabla v_j^i\right)_{K_n} = \left(f, v_j^i\right)_{K_n} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_j^i\right)_{K_n}, \quad \forall v_j^i \in V_n. \tag{3.7}$$

Fin pour

3. Résolution sur le dernier élément du patch .

• Résoudre pour $\rho_j^i \in \mathbb{P}_{p_j} \cap V_1$

$$\left(\mathcal{K}\nabla\rho_j^{N,i}, \nabla v_j^i\right)_{K_N} = \left(f, v_j^i\right)_{K_N} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_j^i\right)_{K_N}, \quad \forall v_j^i \in V_N. \tag{3.8}$$

Et de la même manière pour $\mathcal{T}_j^a$, un patch extérieur composé de N éléments :

Algorithm 8: Calcul de $\rho_{j,\mathbf{a}}^i$ pour un patch extérieur

1. Résolution jusqu'au dernier élément du patch.

Pour n allant de 1 à $N-1$ faire

• Résoudre pour $\rho_j^i \in \mathbb{P}_{p_j} \cap V_1$

$$\left(\mathcal{K}\nabla\rho_j^{n,i}, \nabla v_j^i\right)_{K_n} = \left(f, v_j^i\right)_{K_n} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_j^i\right)_{K_n}, \quad \forall v_j^i \in V_n. \tag{3.9}$$

Fin pour

2. Résolution pour le dernier élément du patch.

• Résoudre pour $\rho_j^i \in \mathbb{P}_{p_j} \cap V_N$

$$\left(\mathcal{K}\nabla\rho_j^{N,i}, \nabla v_j^i\right)_{K_N} = \left(f, v_j^i\right)_{K_N} - \left(\mathcal{K}\nabla u_{J,j-1}^i, \nabla v_j^i\right)_{K_N}, \quad \forall v_j^i \in V_N. \tag{3.10}$$

Les problèmes (3.6) et (3.9) font intervenir des conditions de Neumann sur les arêtes qui ne sont pas en contact avec les bords du patch. Pour le problème (3.7), on impose une condition de Neumann sur l'arête où la solution n'est pas encore connue et une condition de Dirichlet sur l'arête où la solution est obtenue antérieurement. Pour (3.8) et (3.10), on impose Dirichlet là ou la solution est connue. Pour chaque problème, on impose une condition de Dirichet homogène sur les arêtes communes à la frontière du patch. On peut obtenir les formulations fortes décrites par 9 et 10 qui sont valables dans le cas où ρ_j^i est une fonction polynomiale de $\mathcal{P}_{p_j}$.

Algorithm 9: Calcul de $\rho^i_{j,\mathbf{a}}$ pour un patch intérieur (Formulation forte)

1. **Résolution sur le premier élément du patch** .
   - Résoudre le problème suivant correspondant au premier élément.

$$\mathcal{K}\Delta\rho^i_j = f - \mathcal{K}\Delta u^i_{J,j-1} \quad \text{dans } K_1 \tag{3.11}$$
$$\rho^i_j = 0 \quad \text{sur } \partial K_1 \cap \partial\omega_{\mathbf{a}}. \tag{3.12}$$

   - Définir $g_1 = \rho^i_{j,|\partial K_1 \cap \partial K_2}$ et $g_N = \rho^i_{j,|\partial K_1 \cap \partial K_N}$.
2. **Résolution jusqu'à l'avant dernier élément du patch**
   Pour n allant de 2 à $N-1$ faire
   - Résoudre le problème suivant correspondant à l'élément n

$$\mathcal{K}\Delta\rho^i_j = f - \mathcal{K}\Delta u^i_{J,j-1} \quad \text{dans } K_n \tag{3.13}$$
$$\rho^i_j = 0 \quad \text{sur } \partial K_n \cap \partial\omega_{\mathbf{a}} \tag{3.14}$$
$$\rho^i_j = g_{n-1} \quad \text{sur } \partial K_n \cap \partial K_{n-1}. \tag{3.15}$$

   - Définir $g_n = \rho^i_{j,|\partial K_n \cap \partial K_{n+1}}$

   Fin pour
3. **Résolution sur le dernier élément du patch** .
   - Résoudre le problème suivant correspondant à l'élément n

$$\mathcal{K}\Delta\rho^i_j = f - \mathcal{K}\Delta u^i_{J,j-1} \quad \text{dans } K_N \tag{3.16}$$
$$\rho^i_j = 0 \quad \text{sur } \partial K_n \cap \partial\omega_{\mathbf{a}} \tag{3.17}$$
$$\rho^i_j = g_{N-1} \quad \text{sur } \partial K_N \cap \partial K_{N-1}. \tag{3.18}$$
$$\rho^i_j = g_N \quad \text{sur } \partial K_N \cap \partial K_1. \tag{3.19}$$

Algorithm 10: Calcul de $\rho^i_{j,\mathbf{a}}$ pour un patch extérieur (Formulation forte)

1. **Résolution jusqu'à l'avant dernier élément du patch**
   Pour n allant de 1 à $N-1$ faire
   - Résoudre le problème suivant correspondant à l'élément n

$$\mathcal{K}\Delta\rho^i_j = f - \mathcal{K}\Delta u^i_{J,j-1} \quad \text{dans } K_n \tag{3.20}$$
$$\rho^i_j = 0 \quad \text{sur } \partial K_n \cap \partial\omega_{\mathbf{a}} \tag{3.21}$$
$$\rho^i_j = g_{n-1} \quad \text{sur } \partial K_n \cap \partial K_{n-1}. \tag{3.22}$$

   - Définir $g_n = \rho^i_{j,|\partial K_n \cap \partial K_{n+1}}$

   Fin pour
2. **Résolution sur le dernier élément du patch** .
   - Résoudre le problème suivant correspondant à l'élément n

$$\mathcal{K}\Delta\rho^i_j = f - \mathcal{K}\Delta u^i_{J,j-1} \quad \text{dans } K_N \tag{3.23}$$
$$\rho^i_j = 0 \quad \text{sur } \partial K_n \cap \partial\omega_{\mathbf{a}} \tag{3.24}$$
$$\rho^i_j = g_{N-1} \quad \text{sur } \partial K_N \cap \partial K_{N-1}. \tag{3.25}$$

On propose d'illustrer les deux procédures sur les figures 17 et 18.

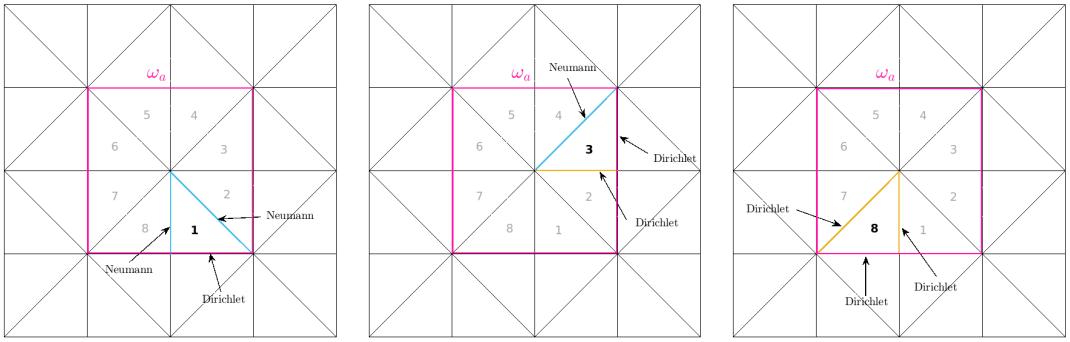


Figure 17: Illustration de la procédure menant au calcul de $\rho^i_{j,\mathbf{a}}$ sur un patch intérieur décrite par l'algorithme 7.

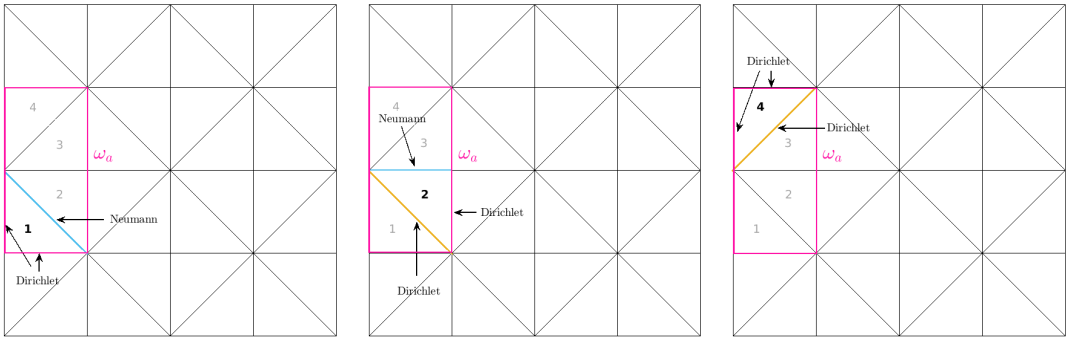


Figure 18: Illustration de la procédure menant au calcul de $\rho^i_{j,\mathbf{a}}$ sur un patch extérieur décrite par l'algorithme 8.

Les arêtes *magenta* représentent la frontière du patch $\omega_{\mathbf{a}}$. Pour les arêtes où sont imposées les conditions de Dirichet, celles en *cyan* représentent les conditions homogènes tandis que celles en *orange* représentent les conditions limites obtenues par la résolution d'un problème sur un élément antérieur.

4 Résultats numériques

4.1 Représentation de solutions

On propose dans cette partie de montrer les résultats obtenus en utilisant l'algorithme 4 sur différents exemples. On utilisera les exemples suivants avec différentes valeurs de J correspondant au nombre de niveaux et p_j le degré polynomiale correspondant au niveau j.

$$\text{Sine :} \quad u(x,y) := \sin(2\pi x)\sin(2\pi y), \quad \Omega := (-1,1)^2, \tag{4.1}$$

$$\text{Peak :} \quad u(x,y) := x(x-1)y(y-1)\exp^{-100((x-0.5)^2-(y-0.117))^2}, \quad \Omega := (0,1)^2. \tag{4.2}$$

$$\text{Poly :} \quad u(x,y) := (1-x^2)(1-y^2), \quad \Omega := (-1,1)^2. \tag{4.3}$$

Les raffinements de maillages $\mathcal{T}_j$ pour générés par raffinement uniforme à l'aide de la méthode de la bisection, on en donne des exemples sur la figure 19.

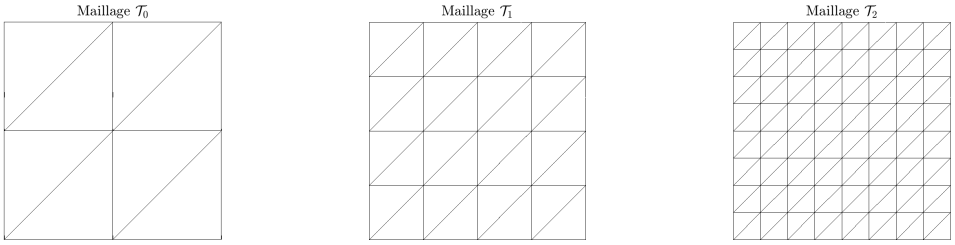


Figure 19: Représensation de $\mathcal{T}_0$ et de deux raffinements uniformes.

4.1.1 Code d'affichage

```
file_name = [ 'Mesh level ' num2str(J) ' triangle Elements FEM(P' num2str(
  Polydegree_level(J)) ').mat'];
  load(file_name);
  Polydegree = Polydegree_level(J);
  Dimension = 2;
  FEM_index = [1:(Polydegree+1).*(Polydegree+2)/2]';
  dim_elem = size(FEM_index,1); % Number of basis function for each element
  TotalEdge = [intEdge; bdEdge];
  NO_NodalBasis = size(Node,1);  NO_EdgeBasis = size(TotalEdge,1).*(Polydegree
  -1);
  NO_ModalBasis = 0.5*(Polydegree-1)*(Polydegree-2).*NT;
  dim_FEM = NO_NodalBasis+ NO_EdgeBasis +NO_ModalBasis;
  figure;
  hold on;
  for t =1:NT
     [local_elem] = Elem(t,:)';
      nodal_index = local_elem;[logl_index] = Checking_local_global_indicator(
      local_elem);
      ind = findGlobalindex(nodal_index,Polydegree,TotalEdge,t,NO_NodalBasis,
      NO_EdgeBasis);
      coef = U(ind ,1);
      node = Node(local_elem,:);
      shiftpoints = [-1,-1; 1,-1 ;-1,1 ];
      P = zeros(size(node,1) ,dim_elem);
      for i =1:dim_elem
         P(:,i)= H_basis2D(shiftpoints,FEM_index(i),Polydegree,logl_index);
```

```
    end
    u_FEM_val = P*coef;    % Finite Element solution
    c = u_FEM_val;
    fill3(node(:,1),node(:,2),u_FEM_val,c,'EdgeColor', 'none') ;
end
```

Listing 4: Code d'affichage

Les données relatives au maillage sont chargées en amont depuis un fichier `.mat`. Étant donné que les coefficients du vecteur de solution `U` sont stockés tels que

$$\mathtt{U} = \begin{bmatrix} \mathtt{nodal\_coefficients;} \\ \mathtt{edge\_coefficients;} \\ \mathtt{modal\_coefficients} \end{bmatrix},$$

il est nécessaire de parcourir l'ensemble des éléments du maillage de manière à attribuer les coefficients associés aux fonctions de base correspondants à chacun d'entre eux.

4.1.2 Représentation 2D de la solution approchée

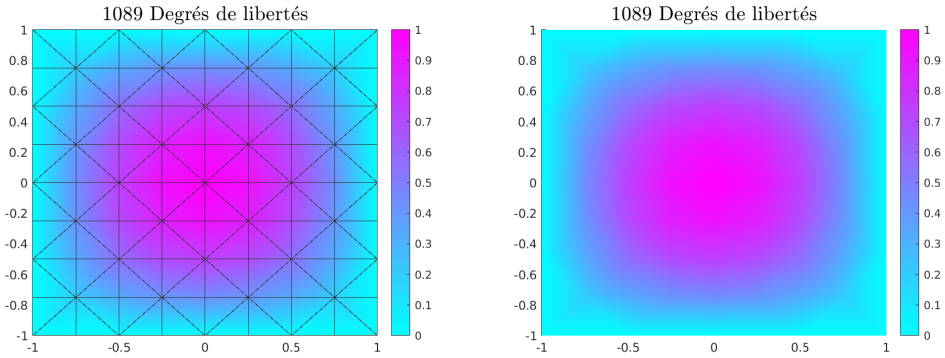


Figure 20: Représentation 2D de la solution calculée pour le problème Poly. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

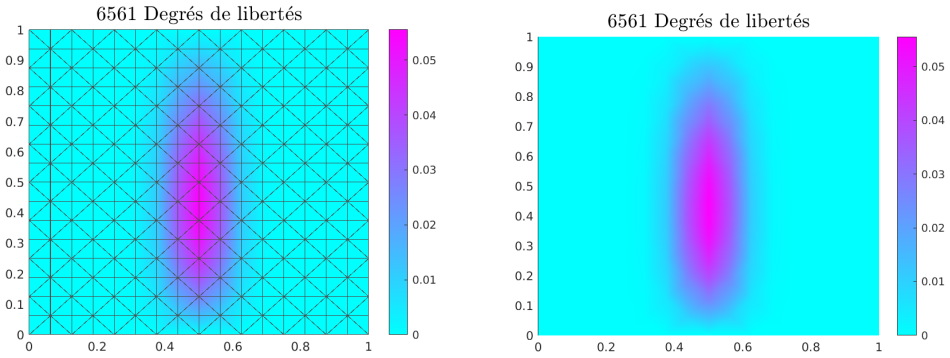


Figure 21: Représentation 2D de la solution calculée pour le problème Peak. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

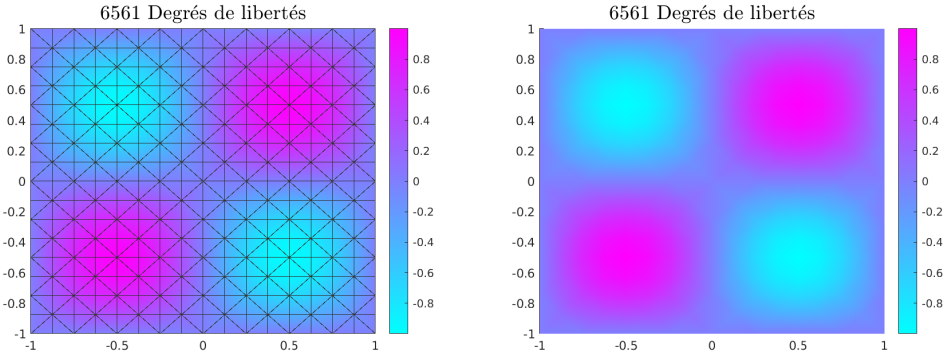


Figure 22: Représentation 2D de la solution calculée pour le problème Sine. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

4.1.3 Représentation 3D de la solution approchée

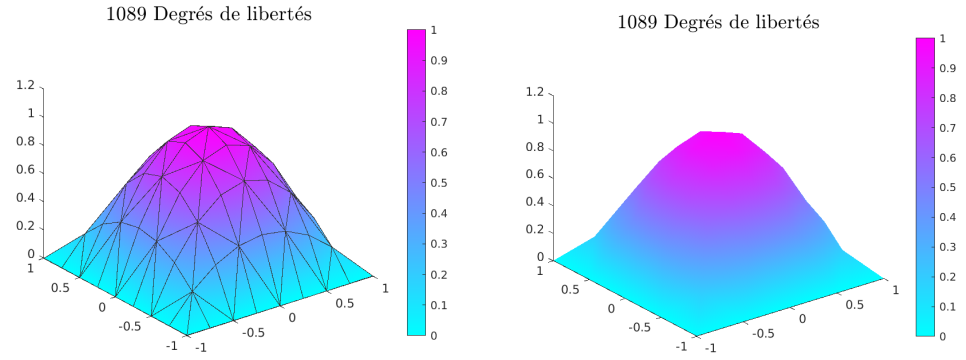


Figure 23: Représentation 3D de la solution calculée pour le problème Peak. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

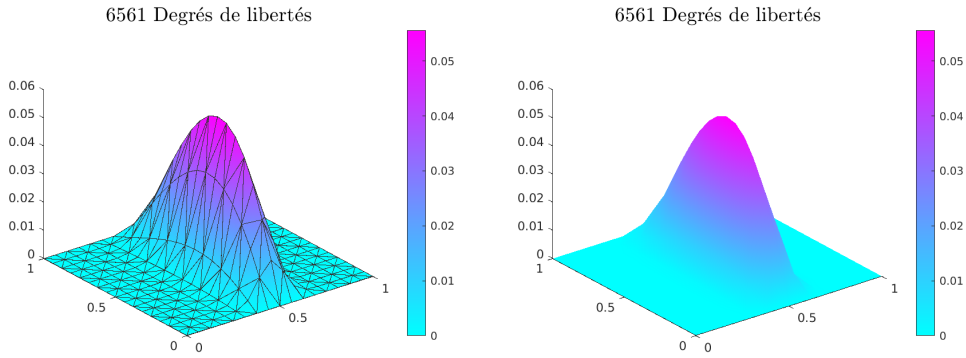


Figure 24: Représentation 3D de la solution calculée pour le problème Peak. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

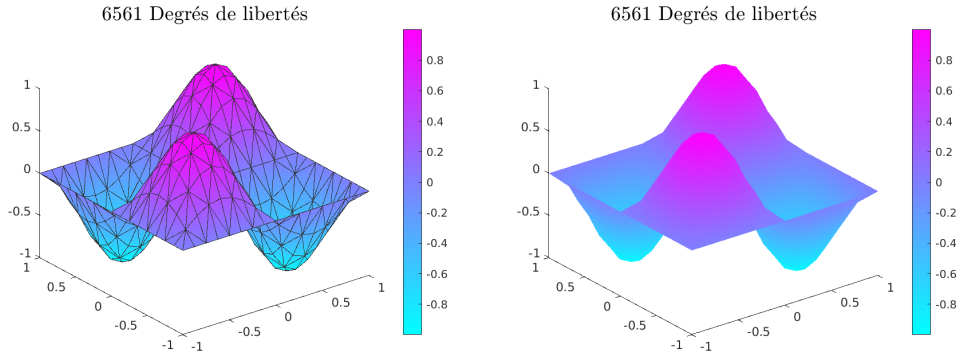


Figure 25: Représentation 3D de la solution calculée pour le problème Sine. Les axes x et y représentent le domaine sur lequel est calculée la solution approchée, la barre de couleur représente la valeur que prend la fonction sur le domaine. A droite la solution est représentée sans maillage.

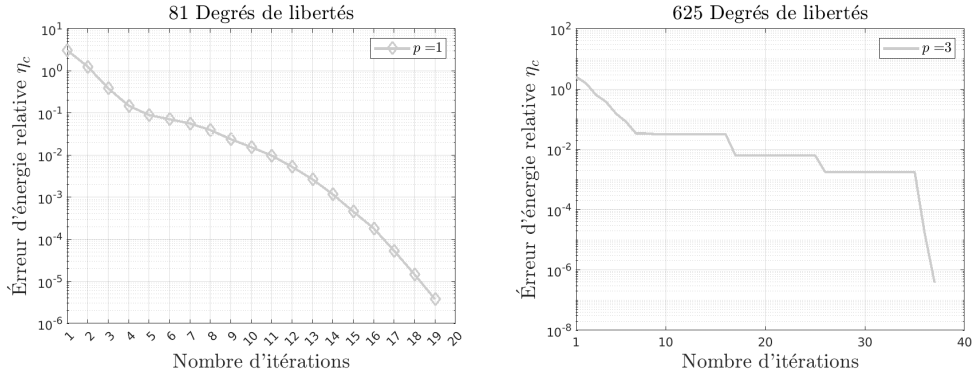
4.2 Validation du code

4.2.1 Vérification de la propriété de *p-robustesse*

Conformément au résultat établi dans la partie 2.2.2, on s'attent à pouvoir vérifier cette propriété numériquement. Pour cela, on introduit l'erreur d'énergie relative suivante :

$$\eta_c = \frac{\|\nabla(u_J - u_J^{i+1})\|}{\|\nabla(u_J)\|} \tag{4.4}$$

Lorsque η_c tend vers 0, la solution approchée tend également vers la solution éléments finis u_J. En pratique, on peut fixer une valeur (par exemple 10^{-5} dans le cas de la figure 27) servant de condition d'arrêt dans le cadre de la validation du code. Regardons dans un premier temps comment évolue le nombre d'itérations en utilisant la méthode du gradiant conjugué (**cg**).



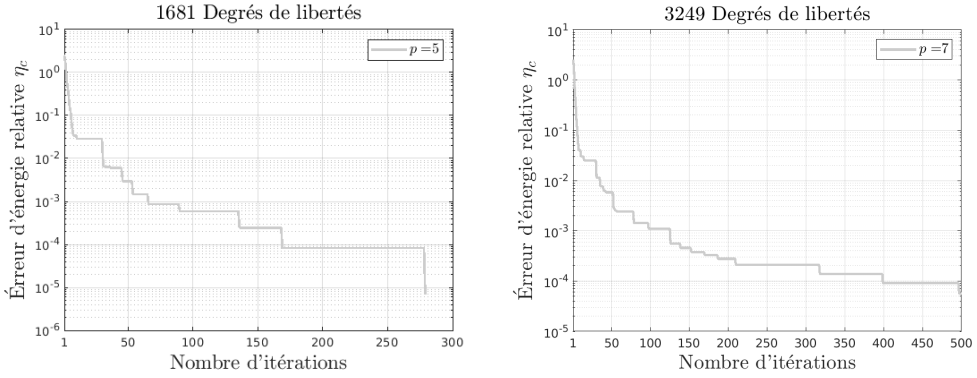
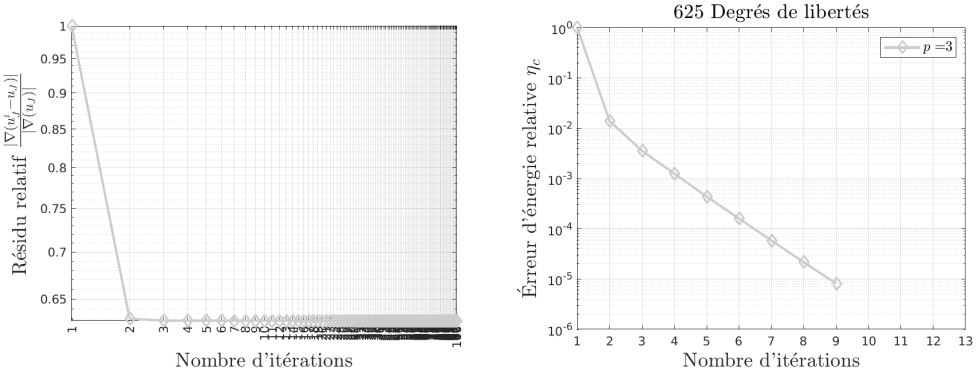


Figure 26: Évolution de η_c en fonction du nombre d'itérations pour le problème Sine avec $J = 2$. La fonction `pcg` de Matlab est utilisée sans précondionneur.

Le nombre d'itérations requis pour réduire l'erreur algébrique à 10^{-5} augmente considérablement lorsque p devient grand. Pour $p = 7$, après 500 itérations cette valeur n'est toujours pas atteinte. Pour cet exemple, la méthode du gradient conjugué a été utilisée sans préconditionneur ; la même tendance aurait pu être observée dans le cas ou il y en aurait un. Regardons désormais comment se comporte le solveur de l'algorithme 4 avec les mêmes conditions.



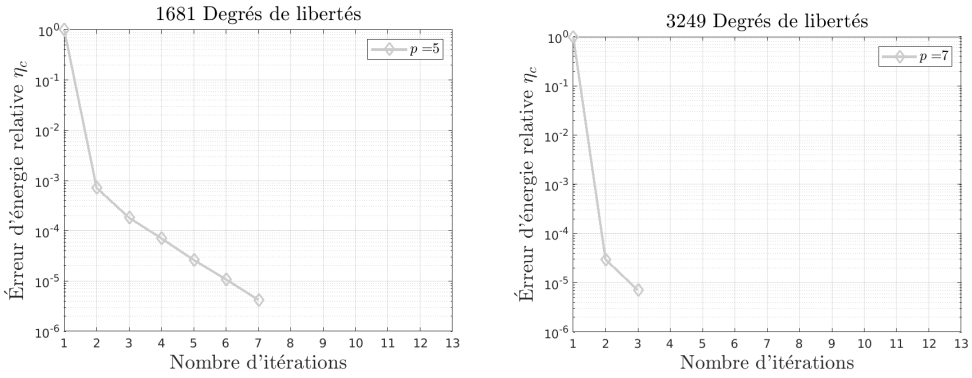


Figure 27: Évolution de η_c en fonction du nombre d'itérations pour le problème Sine avec $J = 2$ suivant différents degrés p avec $p_j = p, \forall j$. Les résultats sont obtenus en utilisant le solveur multigrille de la partie 2.

La figure 27 montre qu'augmenter p n'augmente pas le nombre d'itérations nécessaires à la convergence du solveur. On observe même que le nombre d'itérations requis décroit lorsque p augmente.

4.2.2 Convergence vers la solution éléments finis avec la norme H^1

La solution éléments finis est égale à la solution réelle à une erreur de discrétisation près qu'on peut évaluer avec la norme $H^1(\Omega)$. On s'attend à ce qu'après chaque itération du solveur, l'erreur de la solution se rapproche de l'erreur de discrétisation après chaque itération. On l'illustre sur la figure 28 et 30.

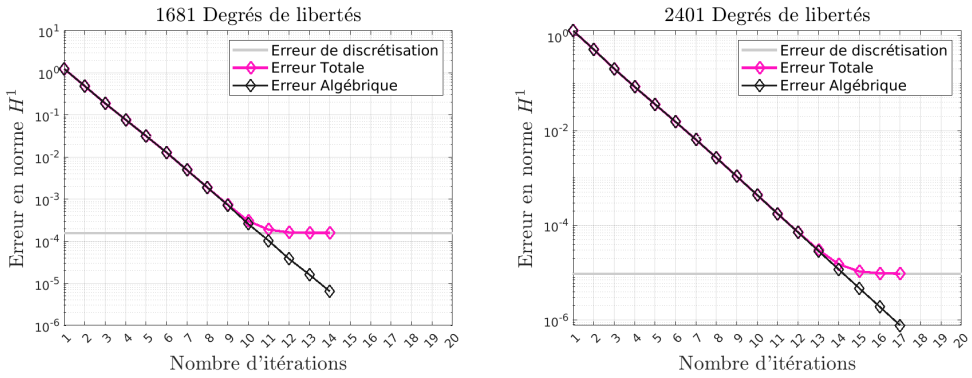


Figure 28: Évolution de l'erreur selon le nombre d'itérations. Problème Sine avec $J = 2$. À gauche $p_0 = 1$, $p_1 = 3$, $p_2 = 5$. À droite $p_0 = 1$, $p_1 = 4$, $p_2 = 6$.

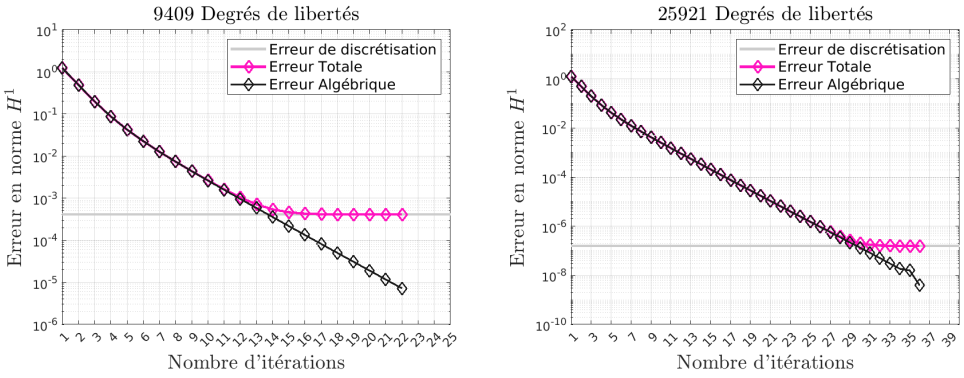


Figure 29: Évolution de l'erreur selon le nombre d'itérations. Problème Sine avec $J = 4$. $\forall j, 0 \leq j \leq J-1, p_j = 1$. À gauche $p_J = 3$. À droite $p_J = 5$.

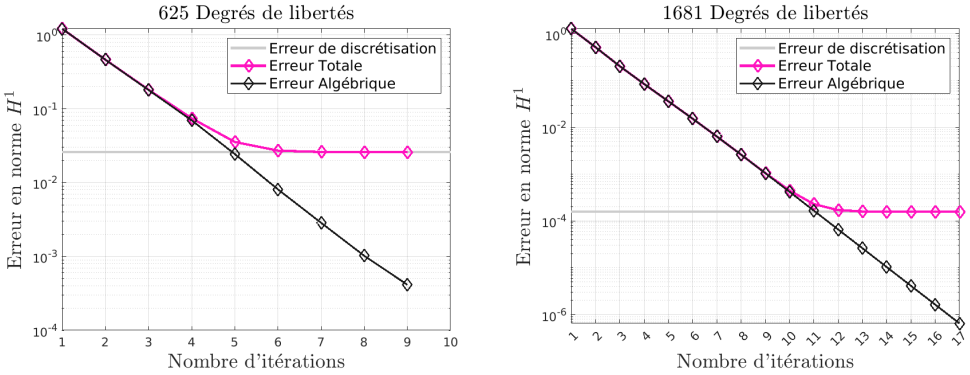


Figure 30: Évolution de l'erreur selon le nombre d'itérations. Problème Peak avec $J = 2$. À gauche $p_0 = 1$, $p_1 = 2$, $p_2 = 3$. À droite $p_0 = 1$, $p_1 = 4$, $p_2 = 5$.

L'erreur de discrétisation est définie comme étant l'erreur entre la solution éléments finis u_j et la solution réelle évaluée aux points de quadrature, elle est représentée en *gris* sur les figures 28 et 30. Notre solveur ne peut produire une solution meilleure que celle-ci. C'est pour cette raison que la courbe *magenta* représentant l'erreur entre la solution calculée à l'itération i, u_j^i et la solution réelle ne passera jamais au dessous de la courbe *grise*. La courbe *noire* représente l'erreur algébrique définie par $\|\nabla(u_J - u_J^{i+1})\|$. Lorsqu'elle est inférieure à l'erreur de discrétisation, il est souhaite d'arrêter le solveur puisqu'il n'est plus possible d'améliorer la qualité de la solution.

4.2.3 Influence du choix de λ sur la convergence

On a démonter que le choix optimal de λ s'obtenait par la formule (2.18). On propose de montrer quelle influence peut avoir un choix différent sur les figures 32 et 33, par la suite on note λ^* comme étant la valeur obtenue par (2.18) à chaque itération. On représente tout d'abord la fonction J

définie par (2.2) :

$$\begin{cases} J : \mathbb{R} & \longrightarrow \mathbb{R}^+, \\ \lambda & \longmapsto J(\lambda) := \|\mathcal{K}^{1/2}\nabla(u_J - (u^i_{J,j-1} + \lambda\rho^i_j))\|^2. \end{cases}$$

.

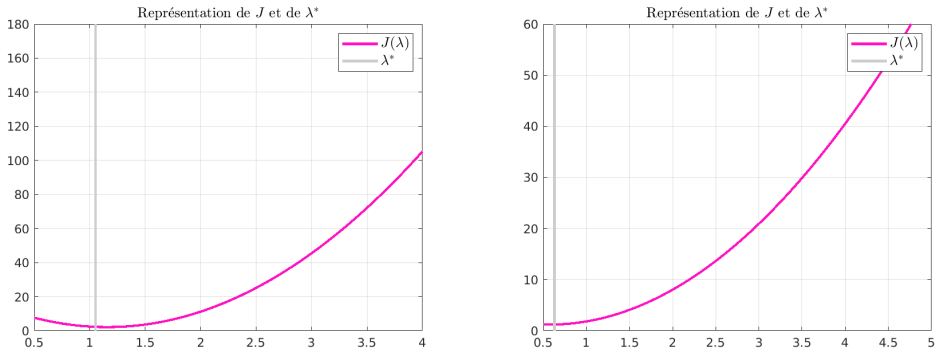


Figure 31: Représentation de J ainsi que de λ^* pour le problème Sine avec $J = 2, p_1 = 1, p_2 = 1$ pour la première itération. À gauche $j = 1$, à droite $j = 2$.

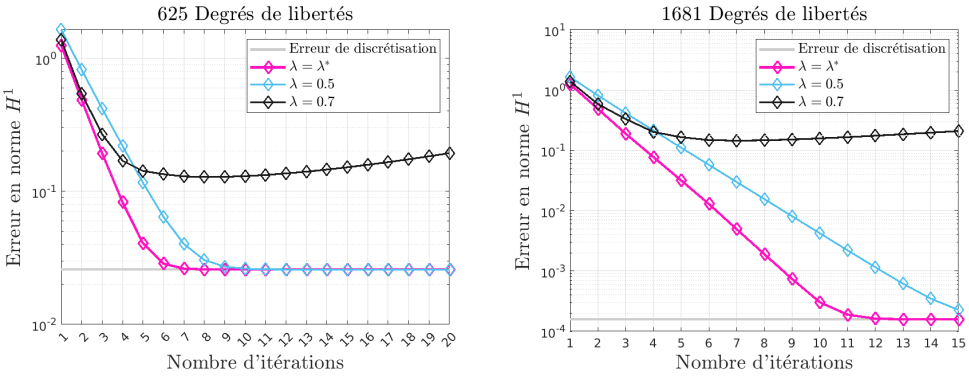


Figure 32: Évolution de l'erreur totale selon le nombre d'itérations avec différents choix pour λ pour le problème Sine. À gauche $p_0 = 1, p_1 = 3, p_2 = 3.$, à droite $p_0 = 1, p_1 = 3, p_2 = 5$.

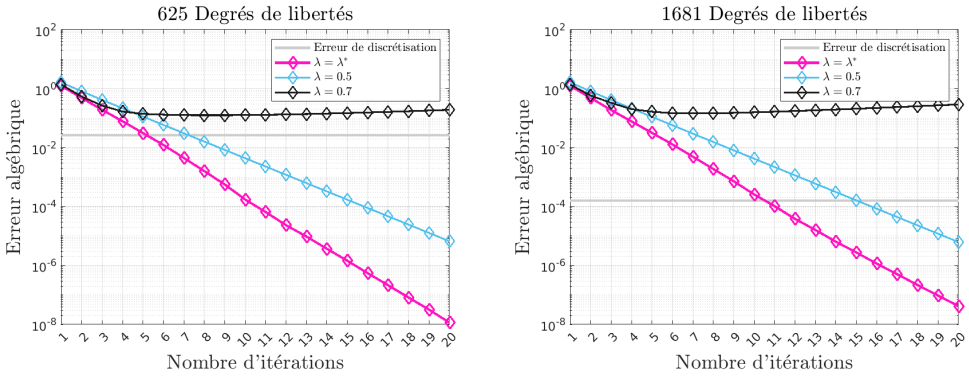


Figure 33: Évolution de l'erreur algébrique selon le nombre d'itérations avec différents choix pour λ pour le problème Sine. À gauche $p_0 = 1, p_1 = 3, p_2 = 3.$, à droite $p_0 = 1, p_1 = 3, p_2 = 5$.

La figure 32 permet de comparer le nombre d'itérations nécessaire à la convergence vers la solution éléments finis u_J. On observe par exemple que pour $\lambda = 0.7$, la convergence n'est pas assurée, l'erreur algébrique augmente et ne décroit pas (figure 33). $\lambda = 0.5$ assure la convergence mais est moins efficace que λ^*.

4.2.4 Efficacité du parallélisme

Pour mesurer l'efficacité du parallélisme, on utilise l'indice d'efficacité $\epsilon_i = T_{init}/T_i$. Dans notre cas, on utilise T_{init} le temps d'éxecution pour 4 processeurs. Les résultats sont représentés sur la figure 34.

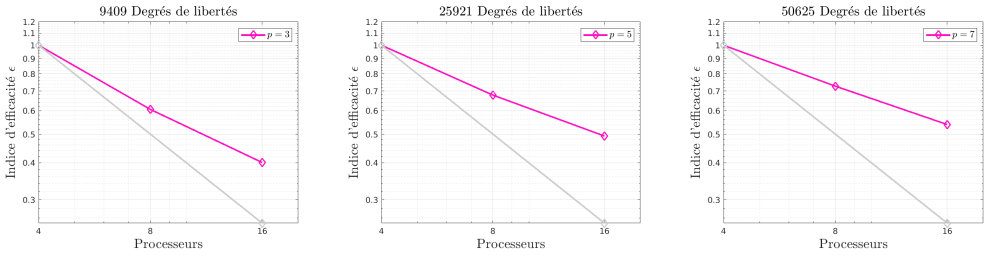


Figure 34: Représentation de ϵ pour différentes valeurs de p.

References

[1] William Briggs, Van Henson, and Steve McCormick. A multigrid tutorial, 2nd edition, 01 2000.

[2] François Cuvelier. Analyse numérique. https://www.math.univ-paris13.fr/~cuvelier, 2021. Cours pour les ingénieurs de première année, Institut Galilée.

[3] François Cuvelier. Introduction à la méthode des Élément finis. https://www.math.univ-paris13.fr/~cuvelier, 2022. Cours pour les ingénieurs de deuxième année, Institut Galilée.

[4] Marion Darbas. Équations aux dérivées partielles linéaires, 2022. Cours pour les ingénieurs de deuxième année, Institut Galilée.

[5] Michel Kern. Analyse numérique avancée, 2022. Cours pour les ingénieurs de deuxième année, Institut Galilée.

[6] Ani Miraçi, Jan Papež, and Martin Vohralík. A-posteriori-steered p-robust multigrid with optimal step-sizes and adaptive number of smoothing steps. *SIAM J. Sci. Comput.*, 43(5):S117–S145, 2021.

[7] Ari Rappaport. Exploration of black box multigrid for linear systems modeling resistive mhd. https://math.unm.edu/~jehanzeb/files/research/rappaport_honors_thesis.pdf, 2018.

[8] Martin Vohraík. A posteriori error estimates for efficiency and error control in numerical simulations. https://who.rocq.inria.fr/Martin.Vohralik/Enseig/APost/a_posteriori.pdf, 2018. Université Pierre et Marie Curie.