MASTER EN SCIENCES MATHÉMATIQUES

Comparaison de méthodes de multigrilles algébriques (type classique vs type agrégation) pour la résolution de systèmes linéaires issus de la discrétisation de problèmes continus

Detournay, Sylvie

Award date:
2006

FUNDP
NAMUR

Faculté des Sciences
Département de Mathématique

Rempart de la Vierge, 8
B - 5000 Namur (Belgique)

Comparaison de méthodes de multigrilles algébriques (type classique vs type agrégation) pour la résolution de systèmes linéaires issus de la discrétisation de problèmes continus

Mémoire présenté pour l'obtention
du grade de
Licencié en Sciences Mathématiques
par

Sylvie DETOURNAY

Promoteur : Annick SARTENAER

Année Académique 2005-2006

Comparaison de méthodes de multigrilles algébriques (type classique vs type agrégation) pour la résolution de systèmes linéaires issus de la discrétisation de problèmes continus

Résumé

Ce mémoire a pour objet la résolution de sytèmes linéaires issus de la discrétisation de problèmes continus par des méthodes de multigrilles algébriques. Nous expliquons tout d'abord la notion de multigrilles en introduisant la méthode de multigrilles géométriques. Nous présentons ensuite les méthodes de multigrilles algébriques et détaillons plus particulièrement deux de ces méthodes : l'une de type classique issue du livre de Briggs, Henson et McCormick [4], et l'autre de type agrégation proposée par Vanek, Mandel et Brezina dans [20]. Finalement, nous comparons les performances de ces méthodes sur base de résultats numériques obtenus à l'aide du programme implémenté dans le cadre de ce mémoire (basé sur la théorie de [4]), et de l'algorithme implémenté par Michal Kocvara (qui utilise la méthode de type agrégation de [20]).

Comparison of algebraic multigrid methods (classical vs aggregation) for the solution of linear systems arising from the discretization of continuous problems

Abstract

The aim of this work is to solve linear systems arising from the discretization of continuous problems by using algebraic multigrid methods. We first explain the multigrid principle by introducing the geometric multigrid methods. We then introduce the algebraic multigrid methods and describe, in particular, two of these methods : the first one, from the book of Briggs, Henson and McCormick [4], is called classical method, and the second one, proposed by Vanek, Mandel and Brezina in [20], is called aggregation method. Finally, we compare the numerical performance of the two methods, using a program implemented in this work (based on the theory of [4]), and a program implemented by Michal Kocvara (which uses the aggregation method of [20]).

J'adresse mes plus sincères remerciements à ma promotrice Annick Sartenaer pour m'avoir encadrée dans la réalisation de ce mémoire. Je souhaite également remercier M. Toint d'avoir suggéré ce sujet, que j'ai beaucoup apprécié.

Je remercie ma soeur Nathalie Detournay et Marie Holvoet qui ont relu et corrigé mon mémoire. Je voudrais aussi remercier Dimitri Tomanos pour m'avoir donné une introduction aux éléments finis.

Finalement, un grand merci à toutes les personnes, famille et amis, qui ont contribué au bon déroulement de mes études.

Table des matières

Introduction

Dans ce mémoire, nous présentons et étudions diverses méthodes de multigrilles pour la résolution de systèmes linéaires issus de la discrétisation de problèmes continus sur une grille. Ces discrétisations, généralement obtenues par des méthodes d'éléments finis ou de différences finies, seront brièvement expliquées.

Bien que notre attention se porte avant tout, dans ce travail, sur les méthodes de *multigrilles algébriques*, nous commencerons, dans le premier chapitre, par introduire les méthodes des *multigrilles géométriques*. Celles-ci s'appliquent à des problèmes discrétisés sur des grilles admettant une certaine structure et des propriétés géométriques importantes, et sont plus adéquates pour comprendre la notion de mutligrilles.

Dans le deuxième chapitre, nous introduirons les méthodes de multigrilles algébriques qui, contrairement au cas géométrique, ne nécessitent pas de grille physique de référence. Ces méthodes s'appliquent directement à la matrice du système, et donc à des problèmes ne possédant aucune grille de référence ou à des problèmes discrétisés sur des grilles irrégulières. Nous présenterons ensuite une méthode particulière de multigrilles algébriques, dite *de type classique* et dévelopée dans [4], que nous avons implémentée dans le cadre de ce mémoire. La conception des algorithmes pour cette méthode sera détaillée et illustrée en fin de chapitre.

Une seconde méthode particulière de multigrilles algébriques, dite *de type agrégation*, fera l'objet du troisième chapitre. Nous y détaillerons plus particulièrement la méthode proposée dans [20] et nommée "Smooth Aggregation", que nous illustrerons à l'aide d'exemples.

Dans le dernier chapitre, nous comparerons deux programmes de multigrilles algébriques : l'un de type classique (implémenté dans le cadre de ce mémoire et basé sur la théorie de [4]) et l'autre de type agrégation (implémenté par Michal Kocvara et élaboré à partir de la méthode de [20]). Nous commenceront par présenter différents problèmes particuliers, auxquels nous appliquerons les deux algorithmes, et analyserons ensuite les résultats numériques obtenus par les deux programmes.

Chapitre 1

Multigrilles Géométriques

Au cours de ce chapitre, nous allons étudier des méthodes servant à résoudre des systèmes linéaires d'un type particulier. En effet, ces systèmes linéaires sont issus de la discrétisation de problèmes continus via les méthodes des éléments finis ou des différences finies. Celles-ci consistent à d'abord discrétiser le domaine continu du problème en un ensemble fini de points, qui forment ce que nous appelons une grille (et que nous appellerons grille fine par la suite), et ensuite à établir un système linéaire de manière à trouver la solution du problème en chaque point de cette discrétisation. Après résolution, une approximation de la solution du problème continu peut être déterminée par interpolation de la solution du système linéaire, celle-ci étant d'autant meilleure que le nombre de points de la grille est élevé.

Cependant, au delà d'une certaine dimension, la résolution de systèmes linéaires par des méthodes directes devient inappropriée. Nous avons alors recours aux procédures itératives (méthodes de Jacobi, Gauss-Seidel, etc). Néanmoins, celles-ci présentent des inconvénients que nous allons découvrir. Nous verrons ensuite comment y remédier grâce aux *méthodes multigrilles*. La technique des multigrilles géométriques est de travailler sur des grilles contenant de moins en moins de points qu'une grille de départ (la grille fine). Cette grille fine, qui comporte le plus de points, correspond à la grille obtenue par la discrétisation du problème continu, qui a servi à former le sytème linéaire à résoudre.

La particularité des multigrilles géométriques est que la grille de discrétisation qui sert à engendrer le système linéaire admet une certaine structure et des propriétés géométriques importantes que nous allons décrire dans ce chapitre.

Le but de ce chapitre est d'introduire la notion des multigrilles sur base des sources [11] et [4]. Bien que la structure de ce chapitre est similaire à celles des ouvrages de référence, les détails et justifications théoriques qui y sont présentés (traitant de la convergence, de l'efficacité, de l'erreur, etc), ne seront pas repris ici.

1.1 Méthodes de relaxation

Dans cette section, nous allons d'abord déterminer notre cadre de travail. Ensuite, nous introduirons les méthodes de relaxation de Jacobi, Jacobi pondéré et Gauss-Seidel, qui font partie de la classe des *méthodes itératives linéaires stationnaires*. Finalement, nous étudierons les faiblesses de ces méthodes à partir des modes de Fourier.

1.1.1 Introduction

Considérons un système de n équations linéaires sous la forme matricielle suivante :

$$Au = f\,, \tag{1.1}$$

où A est une matrice non-singulière de $\mathbb{R}^{n\times n}$, $u \in \mathbb{R}^n$ est la solution unique du système et f est un vecteur de $\mathbb{R}^n$. La solution exacte du système sera notée u tandis que v sera utilisé pour en désigner une approximation.

Pour estimer la qualité d'une solution approchée, v, du système, nous allons introduire deux mesures définies comme suit :

- l'*erreur* (algébrique) :

$$e = u - v \tag{1.2}$$

- et le *résidu* :

$$r = f - Av. \tag{1.3}$$

La norme de l'erreur représente la distance entre la solution approchée et la solution exacte, alors que celle du résidu mesure la différence entre le membre de droite du système et le résultat du produit matriciel entre A et l'approximation de la solution.

L'erreur e de la solution est quasi indéterminable et un petit résidu n'implique pas nécessairement une petite erreur. Cependant, il existe une relation entre ces deux mesures qui permet de trouver une approximation de l'erreur, elle est appelée *équation résiduelle* et est définie de la manière suivante :

$$Ae = r\,. \tag{1.4}$$

Remarquons aussi que $r = 0$ si et seulement si $e = 0$ (par la définition du résidu et de l'erreur et par unicité de la solution).

1.1.2 Méthodes itératives linéaires stationnaires

Une méthode itérative consiste à générer une suite d'approximations $\{v^{(m)}\}$, à partir d'un vecteur initial, qui tente de converger vers la solution du problème (1.1).

Nous allons nous intéresser tout particulièrement à la classe des *méthodes itératives linéaires stationnaires*. Ces procédures ont comme particularité que la formule

de calcul de l'itéré suivant peut s'écrire sous la forme matricielle :

$$v^{(m+1)} = R\,v^{(m)} + Bf\,,$$

où B est une approximation de A^{-1} et R est la matrice d'itération propre à la méthode itérative choisie.

Cette classe de méthodes admet la propriété de convergence qui suit : une méthode associée à une matrice R converge, pour n'importe quel point de départ, lorsque la plus grande valeur propre λ_i de R est strictement inférieure à 1, c'est-à-dire lorsque

$$\rho(R) < 1\,, \tag{1.5}$$

où $\rho(R) = \max_i |\lambda_i|$ est le rayon spectral de la matrice R. Notons que plus le rayon spectral de R est proche de 0, plus le taux de convergence est élevé.

Nous allons présenter quelques méthodes de relaxation importantes : les méthodes de Jacobi, Jacobi pondéré, Gauss-Seidel et les variantes de cette dernière. Etablissons d'abord quelques notations matricielles. Soit la décomposition de la matrice A suivante : $A = D - L - U$, nous noterons D la diagonale de A, $-L$ sa partie triangulaire strictement inférieure et $-U$ sa partie triangulaire strictement supérieure.

Méthode de Jacobi

Cette méthode consiste à prendre comme approximation suivante :

$$v^{(m+1)} = R_J\,v^{(m)} + D^{-1}f\,,$$

où $R_J = D^{-1}(L+U)$ est la matrice d'itération de Jacobi. Cette procédure consiste à calculer la nouvelle approximation $v^{(m+1)}$ à partir de la précédente via un schéma particulier identique à chaque étape.

Méthode de Jacobi pondéré

Cette méthode est une amélioration de la procédure de Jacobi. En effet, elle utilise, à chaque itération $m+1$, une moyenne pondérée de l'approximation obtenue à l'étape précédente, $v^{(m)}$, et de celle obtenue par la méthode de Jacobi classique. La formule itérative s'écrit de la façon suivante :

$$v^{(m+1)} = R_w\,v^{(m)} + w\,D^{-1}f\,, \tag{1.6}$$

où $R_w = (1-w)I + wR_J$ est la matrice d'itération de Jacobi pondéré.

Méthode de Gauss-Seidel

Les deux méthodes précédentes ont l'inconvénient de ne pas utiliser les composantes du vecteur d'approximation de l'itération courante $v^{(m+1)}$ déjà mises à jour. La méthode de Gauss-Seidel remédie à ce problème en considérant, à chaque étape, l'information la plus récente sur chaque élément du vecteur d'approximation. La formule servant à déterminer l'itéré suivant est :

$$v^{(m+1)} = R_G\, v^{(m)} + (D - L)^{-1} f\,, \tag{1.7}$$

où $R_G = (D - L)^{-1}U$ est la matrice d'itération de Gauss-Seidel.

Variantes de la méthode de Gauss-Seidel

Comme décrit plus haut, lors de la mise à jour d'une composante du vecteur d'approximation, la méthode de Gauss-Seidel se sert de l'information la plus récente sur les autres éléments. L'ordre de passage des composantes a donc de l'importance. Cependant, au lieu de parcourir les éléments par ordre ascendant, nous pouvons les considérer par ordre descendant ou même en alternance (ascendant/descendant). Cette dernière façon de procéder porte le nom de Gauss-Seidel symétrique.

Une dernière approche intéressante est la méthode de Gauss-Seidel rouge et noir. Celle-ci consiste à, dans un premier temps, mettre à jour toutes les composantes paires (points rouges), et ensuite s'occuper de tous les éléments impairs (points noirs). L'avantage est que les composantes paires n'ont besoin que des éléments impairs pour être mises à jour et inversement. Cette méthode est intéressante pour l'utilisation de la programmation en parallèle. Dans le cadre des multigrilles, il est à noter que les points rouges correspondront aux points de la grille grossière.

1.1.3 Modes de Fourier

Afin d'étudier les méthodes de relaxation, il est nécessaire d'introduire le concept des modes de Fourier.

Définition des modes de Fourier

Les *modes de Fourier* sont des vecteurs particuliers qui ont comme caractéristique que chacune de leurs composantes prend la forme suivante :

$$v_j = \sin\left(\frac{j\,k\,\pi}{n}\right), \qquad 0 \le j \le n, \qquad 1 \le k \le n-1\,,$$

où j représente l'indice de l'élément du vecteur v, n est la longueur de l'intervalle sur lequel le mode de Fourier est défini, et k est la fréquence.

La figure 1.1 illustre le comportement des modes de Fourier pour les fréquences $k = 1, 3, 6$ ($n = 12$). Sur cet exemple, les vecteurs appartiennent à $\mathbb{R}^{n+1}$ car leurs

composantes commencent à v_0, ainsi nous pouvons remarquer que la fréquence correspond au nombre de demi-sinus formé par le mode de Fourier qui lui est associé. De plus, les vecteurs sont représentés par des fonctions continues car n est généralement grand et, dans ce cas, ils ont un comportement identique.

Nous pouvons distinguer deux types de modes de Fourier qui sont déterminés à partir de leurs fréquences :

- les *modes lisses* (basses fréquences), c'est-à-dire lorsque $1 \leq k \leq \frac{n}{2}$,
- et les *modes oscillants* (hautes fréquences), c'est-à-dire lorsque $\frac{n}{2} \leq k \leq n-1$.

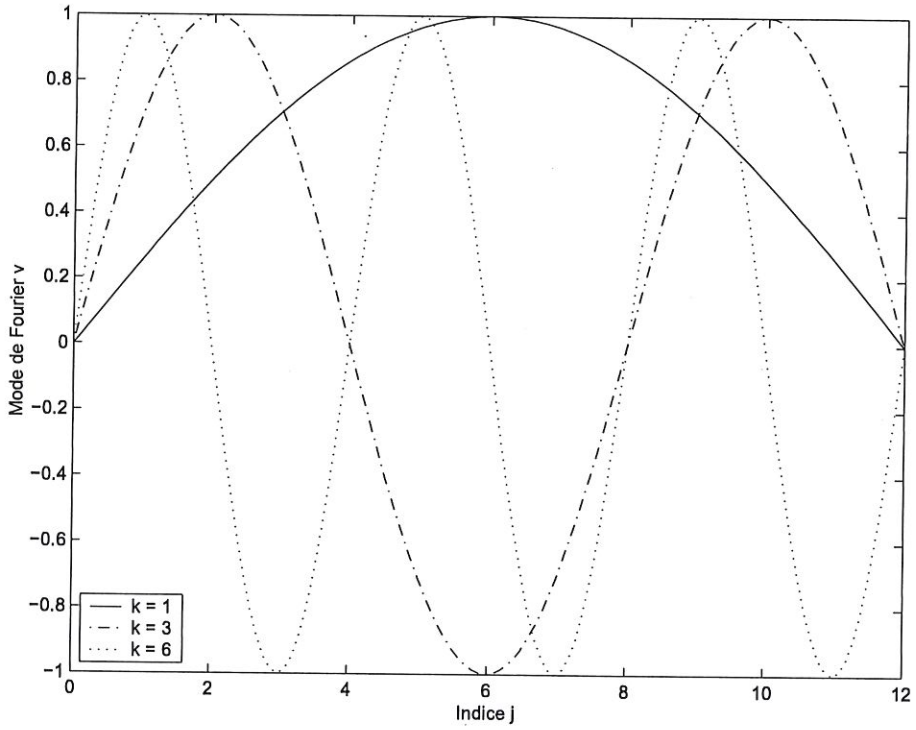

FIG. 1.1 – Représentation des modes de Fourier $v_j = \sin\left(\frac{jk\pi}{n}\right)$, $0 \leq j \leq n$, avec $n = 12$ pour les fréquences $k = 1, 3, 6$.

Propriété de lissage

Dans ce paragraphe, nous allons voir ce qui se produit lorsque nous appliquons un schéma de relaxation à un problème donné en prenant comme vecteur initial un mode de Fourier.

Dans [11], les méthodes de Jacobi pondéré et de Gauss-Seidel sont testées sur un problème issu de la discrétisation d'un Laplacien à une dimension dont le membre de droite est nul. Il est décrit comme tel :

$$\begin{aligned} -u_{j-1} + 2u_j - u_{j+1} &= 0 \\ u_0 = u_n &= 0. \end{aligned}$$

Les procédures de relaxation sont appliquées à cet exemple avec des modes de Fourier comme vecteurs initiaux d'approximation. Sur les graphiques 1.2 et 1.3, nous présentons l'évolution de la norme infinie de l'erreur au cours des itérations, pour les méthodes de Jacobi pondéré ($w = \frac{2}{3}$) et de Gauss-Seidel. Le nombre d'itérations maximal est fixé à 100 et les modes de départ sont $v_j = \sin\left(\frac{jk\pi}{n}\right)$, avec $0 \leq j \leq n$ et $n = 64$ pour les fréquences $k = 1, 5, 35$.

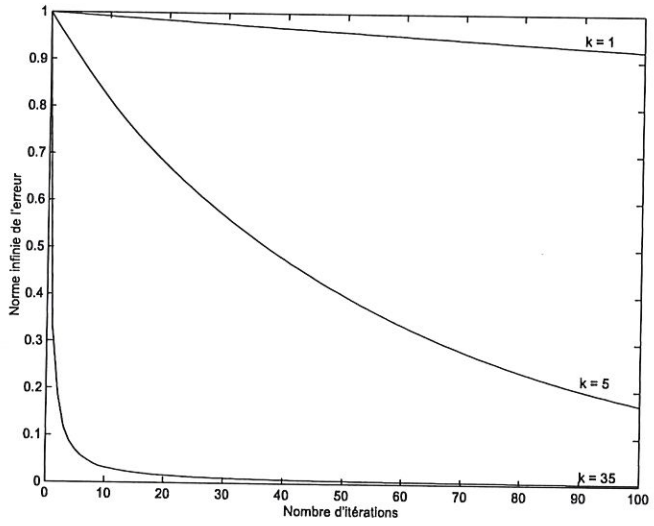

FIG. 1.2 – Evolution de la norme infinie de l'erreur au cours des itérations pour la méthode de Jacobi pondéré ($w = \frac{2}{3}$) avec des modes de départ de fréquences $k = 1, 5, 35$.

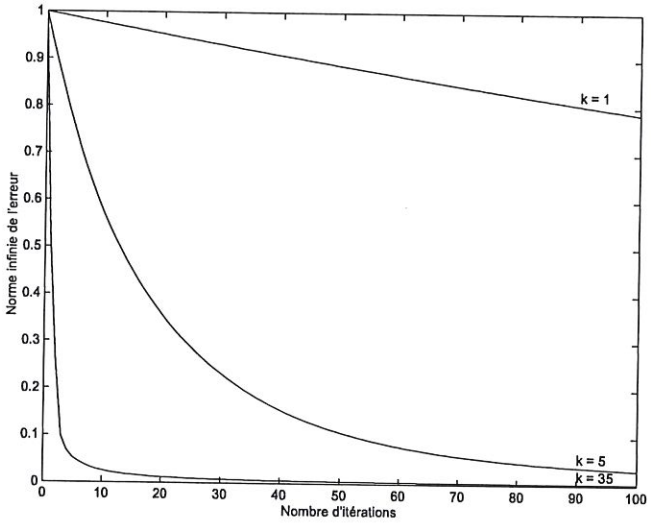

FIG. 1.3 – Evolution de la norme infinie de l'erreur au cours des itérations pour la méthode de Gauss-Seidel avec des modes de départ de fréquences $k = 1, 5, 35$.

Nous constatons, sur ces figures, que lorsque les méthodes de relaxation sont employées avec des modes de Fourier comme approximations de départ, les modes oscillants sont rapidement éliminés tandis que les modes les plus lisses ont tendance à persister. Ce phénomène est appelé *propriété de lissage.*

Modes de Fourier et méthodes de relaxation

Nous allons maintenant voir ce qui se produit lors de l'utilisation des méthodes de relaxation avec un vecteur d'approximation initial quelconque à l'aide des modes de Fourier.

Précédemment, nous avons constaté que la convergence des méthodes itératives stationnaires dépendait des valeurs propres de la matrice d'itération R. A partir de l'analyse de celles-ci, il est possible de montrer [11], que les vecteurs propres des matrices d'itération des méthodes de Jacobi pondéré et de Gauss-Seidel sont des modes de Fourier. Ces procédures vont, dès lors, stagner au bout d'un certain nombre d'itérations.

En effet, considérons la méthode de Jacobi pondéré de matrice d'itération R_w dont les vecteurs propres sont des modes de Fourier notés w_k. Comme un vecteur quelconque peut s'exprimer en fonction des vecteurs propres, nous pouvons écrire l'erreur initiale sous la forme :

$$e^{(0)} = \sum_{k=1}^{n} c_k \, w_k \,, \tag{1.8}$$

où $c_k \in \mathbb{R}$ sont les coefficients de poids dans la décomposition de l'erreur en fonction des vecteurs propres.

Remarquons par définition (1.2) que l'erreur à l'itération $m+1$ prend la forme suivante :

$$e^{(m+1)} = u - v^{(m+1)}.$$

En remplaçant $v^{(m+1)}$ par son expression (1.6), nous obtenons :

$$e^{(m+1)} = u - R_w \, v^{(m)} - w \, D^{-1} \, f \,, \tag{1.9}$$

où $R_w = (1-w)I + wR_J$ est la matrice d'itération de Jacobi pondéré et $R_J = D^{-1}(L+U)$. De plus, nous avons la relation suivante :

$$u = R_w \, u + w \, D^{-1} \, f \,, \tag{1.10}$$

en effet :

$$\begin{aligned} R_w \, u + w \, D^{-1} \, A \, u &= (((1-w)I + wR_J) + w \, D^{-1} \, A) \, u \\ &= (((1-w)I + wD^{-1}(L+U)) + w \, D^{-1} \, (D - L - U)) \, u \\ &= (((1-w)I + w(I + D^{-1}(L+U)) - D^{-1} \, (L+U))) \, u \\ &= u. \end{aligned}$$

En combinant (1.9) et (1.10), nous trouvons que :

$$e^{(m+1)} = R_w\, u - R_w\, v^{(m)} = R_w\, e^{(m)},$$

ainsi, nous pouvons exprimer l'erreur à l'itération m en fonction de l'erreur initiale $e^{(0)}$ de la manière suivante :

$$e^{(m)} = R_w^m\, e^{(0)}. \tag{1.11}$$

Par (1.8) et (1.11), après m itérations, l'erreur prend donc la forme suivante [11] :

$$e^{(m)} = R_w^m\, e^{(0)} = \sum_{k=1}^{n} c_k\, \lambda_k^m(R_w)\, w_k,$$

où les $\lambda_k^m(R_w)$ représentent les valeurs propres de la matrice d'itération R_w. Remarquons qu'après m itérations, le $k^{\text{ième}}$ mode de l'erreur de départ est diminué d'un facteur $\lambda_k^m(R_w)$, ce qui entraîne que si la valeur propre $\lambda_k(R_w)$ correspondant au $k^{\text{ième}}$ mode est proche de 1, celui-ci ne diminuera pratiquement pas. Or, il se trouve que les valeurs propres $\lambda_k(R_w)$ proches de 1 correspondent aux modes à basse fréquence [11]. Par conséquent, les modes lisses de l'erreur diminueront faiblement lors de l'application de la méthode de Jacobi pondéré.

Par un raisonnement similaire, nous pouvons trouver les mêmes conclusions pour la méthode de Gauss-Seidel.

En conclusion, nous pouvons dire que les méthodes de relaxation éliminent rapidement les composantes oscillantes de l'erreur en laissant les composantes lisses persister. Cette propriété de lissage est commune à beaucoup de schémas de relaxation. Les méthodes multigrilles sont des manières de remédier à cette faiblesse.

Dans la section suivante, nous allons développer ces méthodes de multigrilles géométriques. Celles-ci se servent de grilles contenant de moins en moins de points que la grille de départ. Cette dernière, appelée grille fine, est la grille de discrétisation sur laquelle a été engendré le système (1.1). Les points de discrétisation sont appelés noeuds. Nous allons tout d'abord introduire deux approches différentes de schémas multigrilles, ensuite nous préciserons les différents outils utilisés par ces méthodes. Nous finirons par développer des schémas multigrilles récursifs basés sur les deux méthodes introduites.

1.2 Multigrille

Rappelons-nous que notre système linéaire découle de la discrétisation d'un problème continu sur une grille. Dans cette section, nous allons montrer comment remédier aux lacunes des méthodes de relaxation en travaillant *avec plusieurs grilles* contenant de moins en moins de points que celle de départ. Une première approche consiste à chercher une bonne approximation de départ pour la relaxation en résolvant le système sur la grille la plus grossière (c'est-à-dire celle qui comporte le minimum de noeuds). La seconde méthode se sert du fait que les modes lisses qui posent problème aux procédures de relaxation deviennent plus oscillants lorsqu'ils sont transférés sur une grille grossière (voir section 1.2.2).

1.2.1 Itération "nested"

Une première manière de résoudre le système est de partir avec un bon point initial lors de la relaxation sur la grille de départ. Cette méthode consiste à appliquer un schéma de relaxation (vu en début de ce chapitre) au problème initial redéfini sur la grille la plus grossière car cette résolution est peu coûteuse vu que le nombre de noeuds y est moins élevé. La solution obtenue est ensuite progressivement amenée sur la grille fine pour y servir d'approximation de départ pour la relaxation. Cette méthode porte le nom d'*itération "nested"*.

Notons Ω^h la grille la plus fine contenant n points, où $h = \frac{1}{n}$ est le pas de discrétisation. Désignons par Ω^{2h} la grille plus grossière comportant $\frac{n}{2}$ noeuds qui sont les points pairs de Ω^h. De manière similaire, définissons Ω^{4h} la grille contenant la moitié des points de Ω^{2h} et ainsi de suite. Cette stratégie se décrit comme tel :

Schéma de l'itération "nested"

- Appliquer le schéma de relaxation sur la grille la plus grossière pour obtenir une bonne approximation de départ pour la grille fine suivante.

$\vdots$

- Appliquer le schéma de relaxation sur la grille Ω^{4h} pour obtenir un bon point de départ pour Ω^{2h}.
- Appliquer le schéma de relaxation sur Ω^{2h} pour obtenir un bon point de départ pour Ω^h.
- Appliquer le schéma de relaxation sur Ω^h pour obtenir une approximation finale de la solution.

Il reste encore à déterminer la façon de passer d'une grille à une autre et de préciser ce que signifie "appliquer le schéma de relaxation (ou relaxer) sur une grille grossière". Notons aussi que ce procédé ne règle en rien le problème des modes lisses rencontré lors de l'analyse des schémas de relaxation. Ceci sera traité dans la section

suivante.

1.2.2 Schéma de correction

Nous avons remarqué que les méthodes de relaxation rencontrent des difficultés lors de l'élimination des composantes lisses de l'erreur. Regardons ce que deviennent ces modes à basse fréquence si nous les apportons sur une grille grossière.

Considérons deux grilles, Ω^h contenant n points et Ω^{2h} comportant $\frac{n}{2}$ noeuds qui sont les points pairs de la grille fine. La figure 1.4 montre ce que devient un mode à basse fréquence ($k = 4$) lorsqu'il est transféré de la grille de départ Ω^h, contenant $n = 12$ noeuds, à la grille grossière Ω^{2h}. Sur cette grille grossière à $n = 6$ points, le mode de Fourier garde la même fréquence, $k = 4$.

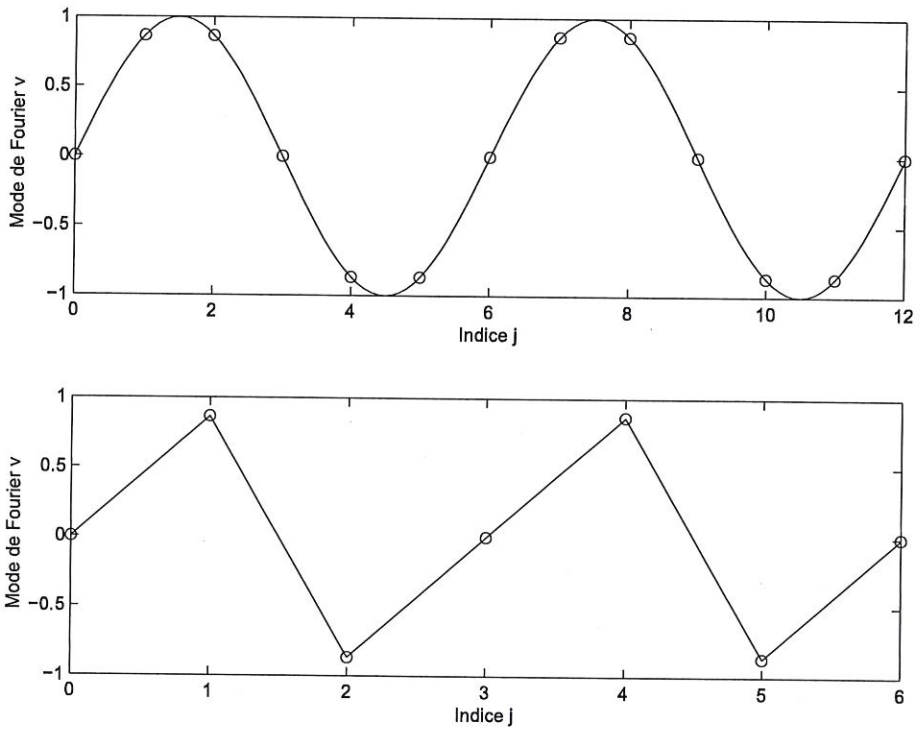

FIG. 1.4 – Représentation du mode de Fourier $v_j = \sin\left(\frac{j4\pi}{n}\right)$, $0 \leq j \leq n$, sur la grille fine Ω^h avec $n = 12$ (au dessus) et sa restriction à Ω^{2h} (en dessous).

Nous constatons sur ces graphiques que les modes lisses apparaissent plus oscillants lorsqu'ils sont restreints à Ω^{2h}. Et inversement, nous pouvons dire [11] que les modes à haute fréquence sont rendus plus lisses sur les grilles contenant moins de points.

Cela implique que si un schéma de relaxation est appliqué au problème sur la grille fine jusqu'à ce que toutes les composantes oscillantes de l'erreur soient élimi-

nées, il est ensuite possible de passer à une grille grossière. Sur cette grille, l'erreur apparaît plus oscillante et un schéma de relaxation peut y être appliqué efficacement.

Pour utiliser ces arguments, nous allons d'abord rappeler que nous possédons un système linéaire propre à l'erreur, l'équation résiduelle, et citer quelques propriétés qui s'y rapportent. Rappelons que l'équation résiduelle (1.4) est définie par :

$$Ae = r\,,$$

où e est l'erreur décrite en (1.2) et r le résidu défini en (1.3). Ce qui signifie, que nous pouvons appliquer un schéma de relaxation *directement sur l'erreur* en nous servant de l'équation résiduelle. Notons aussi qu'il est possible de montrer que résoudre l'équation résiduelle avec un point de départ nul revient à résoudre le problème de départ [11]. De plus, lorsque nous résolvons l'équation résiduelle via un schéma de relaxation avec un point de départ nul, nous obtenons une approximation de l'opposé de l'erreur $-e$ du problème (1.1) (voir [11]).

Après ces quelques constatations, nous pouvons préciser davantage la deuxième approche. Celle-ci consiste à appliquer un schéma de relaxation sur la grille fine au problème de départ jusqu'à ce qu'il ne reste plus que les composantes lisses de l'erreur, c'est-à-dire jusqu'à ce que la relaxation commence à stagner. Ensuite, l'équation résiduelle est transférée vers une grille grossière où un schéma de relaxation lui est appliqué, nous obtenons alors une approximation de l'opposé de l'erreur (et donc de l'erreur). Finalement, cette approximation est apportée sur la grille de départ pour y corriger l'approximation de la solution du problème obtenue précédemment. Cette deuxième méthode porte le nom de *schéma de correction.*

Cette stratégie se décrit comme tel :

Schéma de correction

- Appliquer le schéma de relaxation à $Au = f$ sur Ω^h pour obtenir une approximation v^h.
- Calculer le résidu $r = f - Av^h$.
  - Appliquer le schéma de relaxation à l'équation résiduelle $Ae = r$ sur Ω^{2h} pour obtenir une approximation de l'erreur e^{2h}.
- Corriger l'approximation obtenue sur Ω^h avec l'estimation de l'erreur obtenue sur Ω^{2h} : $v^h \leftarrow v^h + e^{2h}$.

Nous devons encore expliquer comment résoudre l'équation résiduelle sur Ω^{2h} et définir la manière de passer d'une grille à une autre. C'est ce que nous allons faire dans la section suivante en commençant par introduire les outils de transfert intergrilles.

1.2.3 Opérateurs de transfert

Pour pouvoir utiliser les idées présentées au paragraphe précédent, il nous faut encore déterminer la manière de passer d'une grille à une autre. Dans ce but, nous

allons introduire des outils de transfert intergrilles.

Interpolation

Pour passer d'une grille grossière Ω^{2h} à une grille fine Ω^h, nous allons définir un *opérateur d'interpolation* (ou de prolongation) *linéaire*, que nous noterons I_{2h}^h.

Cet opérateur transforme un vecteur v^{2h}, défini sur la grille grossière, en un vecteur v^h se trouvant sur la grille fine, de la façon suivante :

$I_{2h}^h\, v^{2h} = v^h$ où

$$\begin{array}{lcll} v_{2j}^h & = & v_j^{2h} & 1 \leq j \leq \frac{n}{2} - 1 \quad \text{si } n \text{ est pair,} \\ & & & 1 \leq j \leq \frac{n+1}{2} - 1 \quad \text{si } n \text{ est impair,} \\ v_{2j+1}^h & = & \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h}) & 1 \leq j \leq \frac{n}{2} - 1 \quad \text{si } n \text{ est pair,} \\ & & & 1 \leq j \leq \frac{n+1}{2} - 1 \quad \text{si } n \text{ est impair.} \end{array}$$

Nous pouvons remarquer dans cette formule que les composantes paires du vecteur v^h sont reprises telles quelles puisque les points pairs appartiennent à la grille grossière. Un élément impair de v^h est donné par la moyenne des deux composantes voisines de celui-ci.

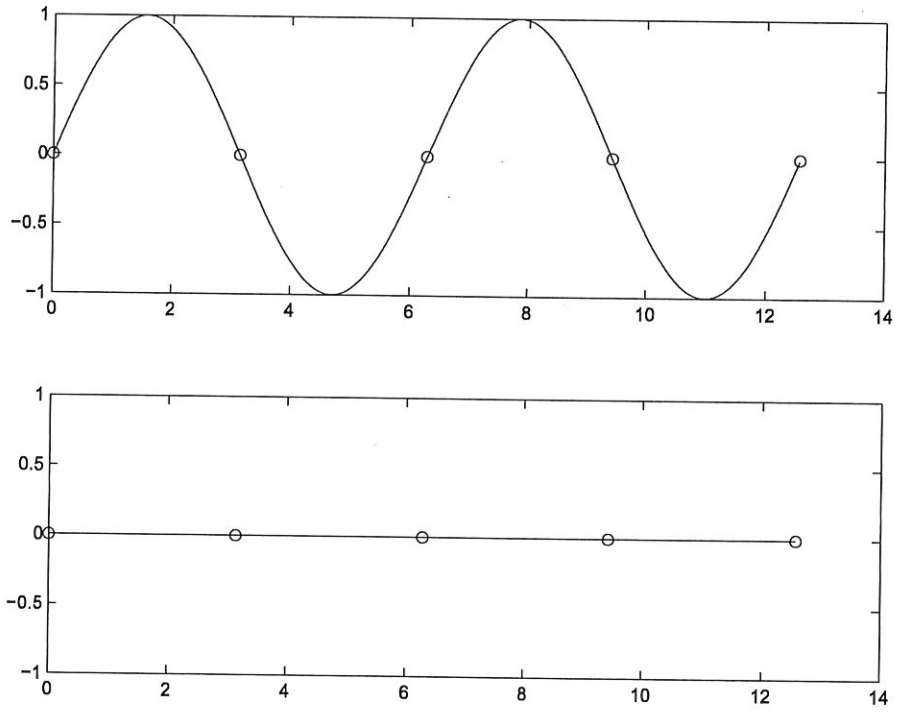

FIG. 1.5 – Interpolation linéaire de la fonction $sin(x)$ définie sur $[0, 4\pi]$ sur base des points $k\pi$ pour $k = 0, 1, 2, 3, 4$.

Il existe d'autres façons d'interpoler, mais ce procédé semble être assez efficace. De plus, il est d'autant plus précis que le vecteur à interpoler à un comportement lisse. Si le vecteur est oscillant, son interpolation linéaire pourra être assez médiocre. En effet, si nous représentons un vecteur oscillant par une fonction continue, comme nous l'avons fait précédemment pour montrer son comportement, nous constatons sur la figure 1.5 que si ce vecteur est défini par un nombre réduit de ces points (ici $k\pi$ pour $k = 0, 1, 2, 3, 4$) le caractère oscillant n'apparaît plus.

Opérateur de restriction

Nous allons maintenant déterminer la manière de passer un vecteur de la grille fine à la grille grossière en utilisant un *opérateur de restriction* noté I_h^{2h}.

Pour réaliser cela, nous pouvons utiliser l'opérateur de restriction, appelé *opérateur d'injection*, qui se limite à prendre comme composantes du vecteur de la grille grossière, les éléments correspondant aux points pairs de la grille fine.

Cependant, nous préférons employer un autre opérateur, nommé *opérateur de restriction avec pondération complète*, car celui-ci possède des propriétés intéressantes, notamment vis-à-vis de notre outil d'interpolation linéaire. Cet opérateur s'applique à un vecteur v^h de la grille fine de la manière suivante :

$I_h^{2h} v^h = v^{2h}$ où

$$v_j^{2h} = \tfrac{1}{4}(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h) \qquad \begin{array}{ll} 1 \leq j \leq \frac{n}{2} - 1 & \text{si } n \text{ est pair,} \\ 1 \leq j \leq \frac{n+1}{2} - 1 & \text{si } n \text{ est impair.} \end{array}$$

Notons qu'une composante du vecteur de la grille grossière v^{2h} est déterminée à partir d'une moyenne pondérée de l'élément pair correspondant et de ses deux voisins impairs du vecteur de la grille fine. Une autre propriété importante de l'opérateur de restriction avec pondération complète est que celui-ci est, à une constante multiplicative près, le transposé de l'opérateur d'interpolation [11]. C'est-à-dire :

$$I_{2h}^h = c(I_h^{2h})^T \qquad c \in \mathbb{R}. \tag{1.12}$$

Notons que nous n'avons pas encore défini la manière de restreindre la matrice A à la grille grossière. Pour cela, nous allons nous servir de la *condition de Galerkin* :

$$A^{2h} = I_h^{2h} A^h I_{2h}^h. \tag{1.13}$$

Ces deux équations (1.12) et (1.13) portent le nom de *propriétés variationnelles*. La justification de ces formules est développée dans [11].

Schéma de correction

Maintenant que nous avons plus d'outils à notre disposition, nous pouvons énoncer plus précisément le schéma de correction défini sur deux grilles. Introduisons

d'abord quelques notations pour une meilleure lecture : nous désignerons les objets définis sur la grille fine Ω^h à l'aide d'un exposant h (A^h, u^h, f^h, r^h, etc) et ceux de la grille grossière par un exposant $2h$ (A^{2h}, u^{2h}, f^{2h}, r^{2h}, etc).

Schéma de correction à deux grilles

$$v^h \leftarrow MG(v^h, f^h)$$

- Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ sur Ω^h avec pour point de départ v^h.
- Calculer le résidu sur la grille fine, $r^h = f^h - A^h v^h$, et le restreindre à la grille grossière Ω^{2h} par $r^{2h} = I_h^{2h} r^h$.
- Résoudre $A^{2h} e^{2h} = r^{2h}$ sur Ω^{2h} avec comme point de départ $e^{2h} = 0$.
- Interpoler l'erreur e^{2h} sur la grille fine par la formule $e^h = I_{2h}^h e^{2h}$ et corriger l'approximation sur la grille fine par $v^h \leftarrow v^h + e^h$.
- Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ sur Ω^h avec pour point de départ v^h.

Les paramètres ν_1 et ν_2 sont déterminés avant le lancement de la procédure. Ils indiquent le nombre maximum d'itérations à appliquer lors de la première et dernière relaxation du système (avant et après la correction sur la grille grossière). Avant de se servir de cette méthode, il est essentiel de définir le schéma de relaxation qui sera utilisé tout au long de la procédure. En général, on choisit le schéma de relaxation de Gauss-Seidel. Il en est de même pour le schéma de l'itération "nested".

Dans le paragraphe suivant, nous allons développer des schémas récursifs basés sur les deux méthodes que nous avons introduites : le schéma de correction à deux grilles et le schéma de l'itération "nested".

1.2.4 Schéma V-cycle

Lors de l'emploi du schéma de correction à deux grilles, nous ne nous sommes pas préoccupés de la manière de résoudre l'équation résiduelle sur la grille grossière Ω^{2h}. Ce problème étant fort semblable à celui de départ (1.1), une idée consiste alors à lui appliquer également un schéma de correction à deux grilles (Ω^{2h} et Ω^{4h}). Ce procédé peut, dès lors, se répéter récursivement, jusqu'à atteindre le niveau le plus grossier où l'équation résiduelle peut être résolue le plus précisément possible. Cette méthode porte le nom de *schéma V-cycle.*

Avant de détailler davantage cette approche, nous allons établir quelques notations. Pour une meilleure visibilité, nous noterons le membre de droite de l'équation résiduelle par f^{2h}, au lieu de r^{2h}, car c'est aussi le membre de droite d'un système. Aussi la solution de l'équation résiduelle sera notée u^{2h} au lieu de e^{2h} et son approximation sera stockée dans v^{2h}. Supposons également qu'il existe un nombre fini de grilles $l > 1$, la grille la plus grossière sera identifiée par Ω^{Lh} où $L = 2^{l-1}$.
La stratégie est, dès lors, décrite comme tel :

Schéma V-cycle

$$v^h \leftarrow V^h(v^h, f^h)$$

- Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.
- Calculer $f^{2h} = I_h^{2h} r^h$.
  - Appliquer ν_1 fois le schéma de relaxation à $A^{2h} u^{2h} = f^{2h}$ avec pour point de départ $v^{2h} = 0$.
  - Calculer $f^{4h} = I_{2h}^{4h} r^{2h}$.

    ⋮

    - Résoudre $A^{Lh} u^{Lh} = f^{Lh}$ par une méthode directe ou via un schéma de relaxation.

    ⋮

  - Corriger l'approximation $v^{2h} \leftarrow v^{2h} + I_{4h}^{2h} v^{4h}$.
  - Appliquer ν_2 fois le schéma de relaxation à $A^{2h} u^{2h} = f^{2h}$ avec pour point de départ v^{2h}.
- Corriger l'approximation $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
- Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.

Cet algorithme peut s'écrire de manière récursive de la façon suivante :

Schéma V-cycle (récursif)

$$v^h \leftarrow V^h(v^h, f^h)$$

1. Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.
2. Si Ω^h est la grille la plus grossière, alors aller à l'étape 4.
   Sinon
   $f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$,
   $v^{2h} \leftarrow 0$,
   $v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h})$.
3. Corriger l'approximation $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
4. Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.

Cette technique porte le nom de *schéma V-cycle*, dû à la forme particulière de sa représentation. La figure 1.6 illustre la stratégie de passage entre les niveaux décrits dans le schéma V-cycle.

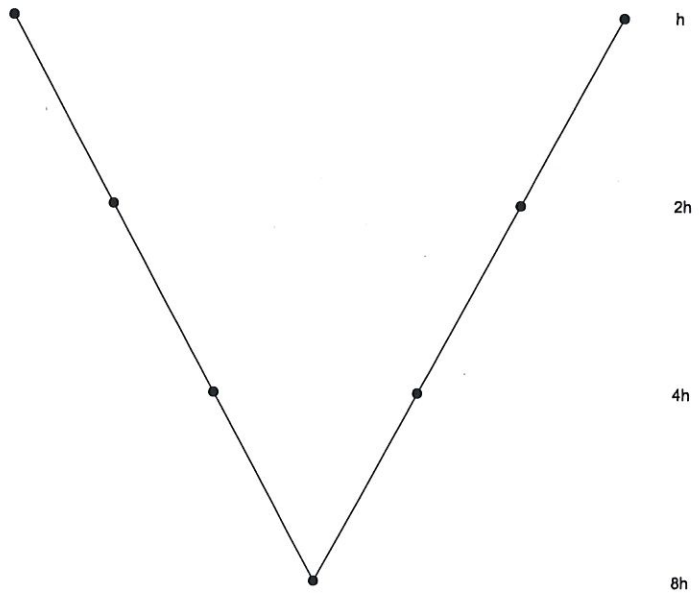

FIG. 1.6 – Représentation d'un V-cycle.

1.2.5 Schéma μ-cycle

Il existe une classe de procédures récursives, se basant sur le schéma de correction à deux grilles, qui porte le nom de *μ-cycle*. Un schéma de la sorte consiste à appeler un *certain nombre de fois (μ)* la procédure elle-même lors de l'appel récursif. La technique V-cycle correspond à un schéma μ-cycle de paramètre $\mu = 1$. Voici l'algorithme d'une méthode μ-cycle :

Schéma μ-cycle

$$v^h \leftarrow M\mu^h(v^h, f^h)$$

- Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.
- Si Ω^h est la grille la plus grossière, alors aller à l'étape 4.
  Sinon
  $f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$,
  $v^{2h} \leftarrow 0$,
  $v^{2h} \leftarrow M\mu^{2h}(v^{2h}, f^{2h})$ μ fois.
- Corriger l'approximation $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
- Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.

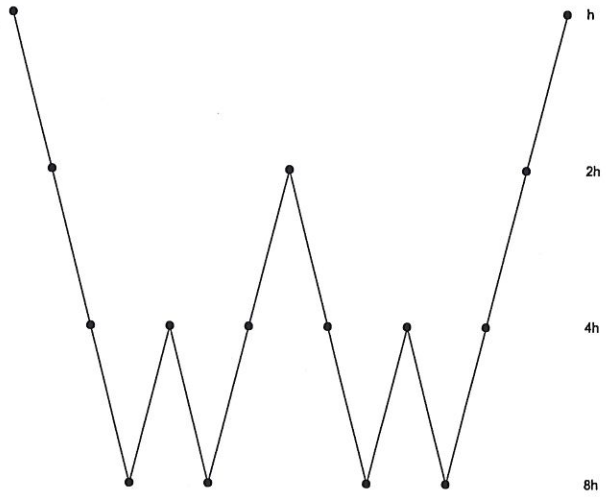

FIG. 1.7 – Schéma μ-cycle de paramètre $\mu = 2$, nommé W-cycle.

Pour mieux comprendre le fonctionnement de ces procédures, illustrons le procédé utilisé par le schéma μ-cycle de paramètre $\mu = 2$, nommé *schéma W-cycle* et représenté par la figure 1.7. Voici le détail de la procédure du schéma W-cycle avec quatre niveaux :

◇ Appliquer ν_1 fois le schéma de relaxation, sur Ω^h, à $A^h u^h = f^h$ avec pour point de départ v^h.

◇ Poser :

$f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$,

$v^{2h} \leftarrow 0$.

Appliquer le premier $\mu - cycle$ sur Ω^{2h} :

- Relaxer ν_1 fois sur Ω^{2h} avec pour point de départ v^{2h}.
- Poser :

  $f^{4h} \leftarrow I_{2h}^{4h}(f^{2h} - A^{2h} v^{2h})$,

  $v^{4h} \leftarrow 0$.

  Appliquer le premier $\mu - cycle$ sur Ω^{4h} :

  ▷ Relaxer ν_1 fois sur Ω^{4h} avec pour point de départ v^{4h}.

  ▷ Poser :

  $f^h \leftarrow I_{4h}^{8h}(f^{4h} - A^{4h} v^{4h})$,

  $v^{8h} \leftarrow 0$.

  Appliquer le premier $\mu - cycle$ sur Ω^{8h} :

  ⋆ Relaxer ν_1 fois sur Ω^{8h} avec pour point de départ v^{8h}.

  ⋆ Relaxer ν_2 fois sur Ω^{8h} avec pour point de départ v^{8h}.

  Appliquer le deuxième $\mu - cycle$ sur Ω^{8h} :

  ⋆ Relaxer ν_1 fois sur Ω^{8h} avec pour point de départ v^{8h}.

  ⋆ Relaxer ν_2 fois sur Ω^{8h} avec pour point de départ v^{8h}.

  ▷ Corriger l'approximation $v^{4h} \leftarrow v^{4h} + I_{8h}^{4h} v^{8h}$.

  ▷ Relaxer ν_2 fois sur Ω^{4h} avec pour point de départ v^{4h}.

  Appliquer le deuxième $\mu - cycle$ sur Ω^{4h} :

  ▷ ⋮

  ⋆ ⋮

  ⋆ ⋮

  ▷ ⋮

- Corriger l'approximation $v^{2h} \leftarrow v^{2h} + I_{4h}^{2h} v^{4h}$.
- Relaxer ν_2 fois sur Ω^{2h} avec pour point de départ v^{2h}.

Appliquer le deuxième $\mu - cycle$ sur Ω^{2h} :

- ⋮

  ▷ ⋮

  ▷ ⋮

- ⋮

◇ Corriger l'approximation $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.

◇ Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.

1.2.6 Schéma V-cycle complet avec multigrilles

Maintenant que nous avons développé les schémas V-cycles, nous pouvons y incorporer la stratégie "nested" vue au début de cette section. La nouvelle méthode, appelée *schéma V-cycle complet avec multigrilles*, consiste à d'abord descendre au niveau le plus grossier, ensuite d'y résoudre le problème restreint et finalement de ramener la solution trouvée par interpolation de grille en grille, tout en appliquant un certain nombre de fois le schéma V-cycle à chaque niveau. Voici la stratégie de ce procédé :

Schéma V-cycle complet avec multigrilles

$$v^h \leftarrow FMG^h(f^h)$$

Initialiser $f^{2h} \leftarrow I_h^{2h} f^h$, $f^{4h} \leftarrow I_{2h}^{4h} f^{2h}$, ...

- Résoudre le système ou appliquer un schéma de relaxation sur la grille la plus grossière.

⋮

- $v^{2h} \leftarrow I_{2h}^{4h} v^{4h}$.
- $v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h})$, $\underline{\nu_0 \text{ fois}}$.

- $v^h \leftarrow I_h^{2h} v^{2h}$.
- $v^h \leftarrow V^h(v^h, f^h)$, $\underline{\nu_0 \text{ fois}}$.

L'algorithme récursif du schéma V-cycle complet avec multigrilles est présenté ci-dessous.

Schéma V-cycle complet avec multigrilles (récursif)

$$v^h \leftarrow FMG^h(f^h)$$

1. Si Ω^h est la grille la plus grossière, poser $v^h \leftarrow 0$ et aller à l'étape 3.
   Sinon
   $f^{2h} \leftarrow I_h^{2h}(f^h)$,
   $v^{2h} \leftarrow FMG^{2h}(f^{2h})$.
2. Corriger l'approximation $v^h \leftarrow I_{2h}^h v^{2h}$.
3. $v^h \leftarrow V^h(v^h, f^h)$ $\underline{\nu_0 \text{ fois}}$.

En pratique, le paramètre ν_0 qui indique le nombre de V-cycle à appliquer à chaque niveau vaut 1. La figure 1.8 illustre un schéma V-cycle complet avec multigrilles où $\nu_0 = 1$ et où le nombre de niveaux est égal à quatre. Celui-ci consiste à d'abord descendre sur la grille la plus grossière Ω^{8h} afin d'y résoudre le problème initial. Ensuite, l'approximation trouvée est interpolée sur le niveau précédent Ω^{4h} pour y servir de point de départ d'un schéma V-cycle. Une nouvelle approximation de la solution est alors donnée par le V-cycle sur ce niveau et est ensuite amenée sur la grille fine précédente Ω^{2h} pour y démarrer un nouveau V-cycle, et ainsi de suite,

jusqu'à atteindre la grille de départ où l'approximation interpolée sert de point de départ au dernier V-cycle.

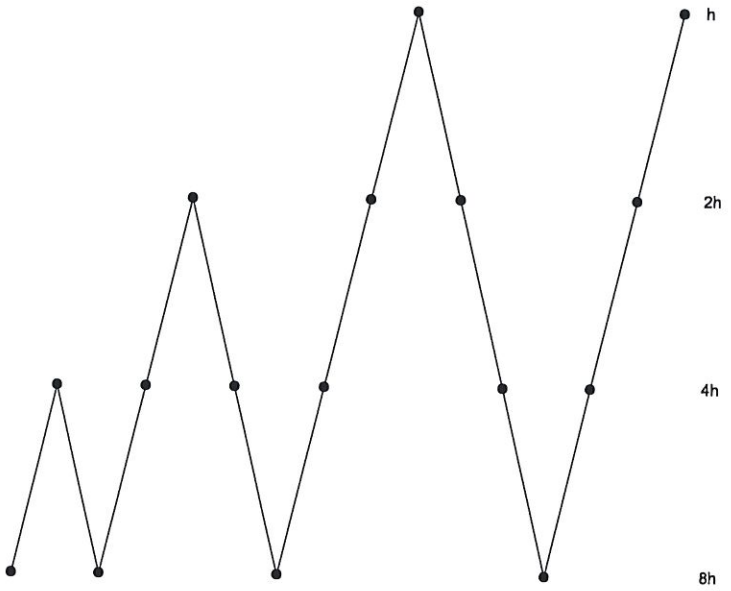

FIG. 1.8 – Schéma V-cycle complet avec multigrilles où $\nu_0 = 1$.

1.2.7 Conclusion

Nous avons, au cours de ce chapitre, présenté le fonctionnement des méthodes multigrilles géométriques sans argument théorique pour justifier nos choix. De telles justifications (traitant de la convergence, de l'efficacité, de l'erreur, etc) ont été développées dans [11].

Chapitre 2

Multigrilles Algébriques Classiques

Au cours de ce chapitre, nous allons nous intéresser aux *méthodes des multigrilles algébriques* introduites dans les ouvrages [4] et [14]. Ces méthodes ont la particularité de se baser uniquement sur les informations contenues dans la matrice du système à résoudre. C'est-à-dire que le problème de départ *ne nécessite plus de grille physique* de référence, comme c'était le cas pour les multigrilles géométriques. Dès lors, nous pourrons appliquer ces méthodes à des problèmes ne possédant pas de grille de référence ou définis sur des grilles de structures irrégulières.

L'idée principale des méthodes multigrilles algébriques est d'étendre les concepts développés dans le cadre des multigrilles géométriques (principe de lissage, correction sur les grilles grossières, etc) à une classe de problèmes ne possédant pas de grille structurée de référence. Nous serons amenés à adapter ou redéfinir les objets présentés dans le premier chapitre des multigrilles géométriques, à savoir, les grilles, les opérateurs de transfert intergrilles, le schéma de relaxation, etc.

Afin d'établir tous ces concepts, nous allons tout d'abord préciser ce que nous entendrons par "grille". Ensuite, nous définirons la notion d'*algébriquement lisse* et d'*influence - dépendance* qui serviront par la suite à déterminer l'opérateur d'interpolation et la façon de sélectionner les points de la grille grossière. Ensuite, nous construirons les différents opérateurs à l'aide des propriétés variationnelles ((1.12) et (1.13)) et nous établirons le schéma V-cycle de manière analogue au cas des multigrilles géométriques. Nous finirons par décrire les algorithmes utilisés lors de la conception du programme développé dans le cadre de ce mémoire et nous illustrerons différents cas à l'aide d'exemples.

2.1 Notion de grille dans le cadre des multigrilles algébriques

Afin de pouvoir utiliser tous les concepts présentés dans le premier chapitre sur les multigrilles géométriques, il est essentiel de commencer par déterminer ce que nous appellerons "grille" par la suite.

Rappelons que nous travaillons sur un système défini de la manière suivante :

$$Au = f \qquad \text{où } u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \in \mathbb{R}^n, \tag{2.1}$$

A est une matrice non-singulière de $\mathbb{R}^{n\times n}$ et f est un vecteur de $\mathbb{R}^n$. Dans ce cas-ci, la matrice A ne possède pas de grille structurée de référence. Comme précédemment, nous noterons v toute approximation de la solution exacte u.

Nous avons vu, dans le cas des multigrilles géométriques, que chaque inconnue, u_i, du système (2.1) est définie en un point physique de la grille fine. C'est en sélectionnant un ensemble de ces noeuds que nous pouvions représenter la solution sur la grille grossière. De manière similaire, nous allons maintenant chercher un ensemble d'inconnues pour représenter la solution sur la grille grossière pour le cas des multigrilles algébriques. Dès lors, une manière de construire notre grille fine est d'identifier les noeuds de celle-ci aux indices des inconnues du système (2.1), c'est-à-dire $\{1, 2, \ldots, n\}$. Les autres grilles sont définies de façon identique.

Par analogie au chapitre précédent, nous continuerons à noter la grille fine Ω^h et les grilles grossières : Ω^{2h}, Ω^{4h}, etc. Nous indicerons aussi les différents opérateur de manière identique aux cas des multigrilles géométriques : A^h, I^h_{2h}, etc. Notons que ces notation n'ont plus vraiment de sens ici puisque le problème (2.1) ne nécessite plus de pas de discrétisation.

Ensuite, définissons les connexions ou liens entre les points de la grille à l'aide du *graphe d'adjacence non orienté de la matrice A*. Notons un élément de la matrice par a_{ij}, où i est l'indice de ligne et j l'indice de colonne. Deux sommets du graphe de A, i et j, sont adjacents[1] si $a_{ij} \neq 0$ ou $a_{ji} \neq 0$. La figure 2.1 est la représentation du graphe d'adjacence non-orienté de la matrice décrite ci-dessous :

$$A = \begin{pmatrix} X & X & & X & X & \\ X & X & X & X & & \\ & X & X & X & X & X \\ X & X & X & X & & \\ X & & X & & X & X \\ & & X & & X & X \end{pmatrix}. \tag{2.2}$$

Dans les multigrilles algébriques, les points d'une grille Ω^h sont associés aux sommets du graphe d'adjacence non orienté de la matrice A^h et les connexions entre les noeuds correspondent aux arrêtes.

[1] Deux sommets d'un graphe non orienté sont dit adjacents s'il existe une arrête les joignant.

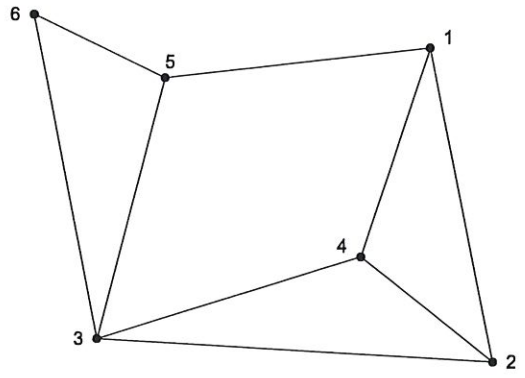

FIG. 2.1 – Représentation du graphe d'adjacence de la matrice A définie par (2.2).

Lors de l'étude des multigrilles géométriques, nous avions montré que l'application des schémas de relaxation avait pour effet de supprimer les composantes oscillantes de l'erreur. Ensuite, nous avions constaté que le passage d'une grille fine à une grille grossière rendaient les vecteurs lisses oscillants. Nous avions aussi remarqué qu'une erreur lisse pouvait être approximée de manière précise sur une grille grossière. De plus, les opérateurs de transfert intergrilles choisis permettaient de représenter avec précision les fonctions lisses à travers toutes les grilles. A partir de toutes ces constatations, nous avions pu développer les méthodes des multigrilles géométriques.

Pour les multigrilles algébriques, nous allons devoir déterminer la manière de sélectionner les points de la grille grossière et les opérateurs de transfert intergrilles tels que les fonctions lisses soient représentables précisément sur toutes les grilles. Pour réaliser cela, nous allons d'abord devoir préciser ce que nous entendons par "lisse" pour le cas algébrique. C'est dans cette optique que nous allons définir, dans le paragraphe suivant, le *caractère algébriquement lisse.*

2.2 Caractère algébriquement lisse

Dans le cadre des multigrilles géométriques, les fonctions lisses étaient *géométriquement lisses*, c'est-à-dire qu'elles étaient à basse fréquence. Pour les multigrilles algébriques, nous n'avons plus de référence physique. Nous allons donc devoir représenter ce caractère lisse autrement.

Dans le cas géométrique, la principale caractéristique des erreurs lisses est qu'elles n'étaient pas totalement éliminées par l'application d'un schémas de relaxation (propriété de lissage). C'est à partir de cette constatation que nous allons définir le *caractère algébriquement lisse.*

Définition 1. *Une erreur est dite algébriquement lisse si elle n'est pas réduite de manière significative par l'application d'un schéma de relaxation.*

Avant de détailler davantage cette définition, nous allons émettre une hypothèse assez forte sur la matrice A du système à résoudre (2.1) : celle-ci doit être une *M-matrice* symétrique. Une M-matrice est une matrice définie positive ($\forall u \neq 0 \;\; u^T A u > 0$), dont les éléments diagonaux sont positifs et les composantes hors-diagonales sont négatives. Une bonne partie des systèmes issus de la discrétisation de problèmes elliptiques vérifient cette propriété. Cette hypothèse n'est pas nécessaire au fonctionnement des méthodes multigrilles algébriques mais leur efficacité est fortement diminuée lorsque la matrice du système est loin d'être une M-matrice.

Nous allons préciser la notion d'algébriquement lisse en choisissant le schéma de relaxation de Jacobi pondéré défini au chapitre précédent. Rappelons que nous pouvons écrire sa formule itérative de la façon suivante, cfr (1.6) :

$$v^{(m+1)} = R_w \, v^{(m)} + w \, D^{-1} \, f \, ,$$

où $R_w = (1 - w)I + wR_J$ est la matrice d'itération de Jacobi pondéré et $R_J = D^{-1}(L + U)$ est la matrice d'itération de Jacobi[2]. De manière équivalente, nous pouvons écrire :

$$\begin{aligned}
v^{(m+1)} &= R_w \, v^{(m)} + w \, D^{-1} \, f \\
&= ((1 - w) \, I + w \, R_J) \, v^{(m)} + w \, D^{-1} \, f \\
&= v^{(m)} - w \, v^{(m)} + w \, R_J \, v^{(m)} + w \, D^{-1} \, f \\
&= v^{(m)} + w \, (-v^{(m)} + R_J \, v^{(m)} + D^{-1} \, f) \\
&= v^{(m)} + w \, ((R_J - I) \, v^{(m)} + D^{-1} \, f) \\
&= v^{(m)} + w \, ((D^{-1}(L + U) - I) \, v^{(m)} + D^{-1} \, f) \\
&= v^{(m)} + w \, D^{-1} \, ((L + U - D) \, v^{(m)} + f).
\end{aligned}$$

En remplaçant $(L + U - D)$ par $-A$, on obtient :

$$v^{(m+1)} = v^{(m)} + w \, D^{-1} \, (f - A \, v^{(m)}).$$

Remarquons aussi que

$$u = u + w \, D^{-1} \, (f - A \, u),$$

où u est la solution exacte du système (2.1) (cette propriété porte le nom de *condition de consistance*). Dès lors, nous pouvons écrire l'erreur (1.2) à l'itération $(m + 1)$ de la façon suivante :

$$\begin{aligned}
e^{(m+1)} &= u - v^{(m+1)} \\
&= (u + w \, D^{-1} \, (f - A \, u)) - (v^{(m)} + w \, D^{-1} \, (f - A \, v^{(m)})) \\
&= e^{(m)} - w \, D^{-1} \, A \, u + w \, D^{-1} \, A \, v^{(m)} \\
&= e^{(m)} - w \, D^{-1} \, A \, e^{(m)} \\
&= (I - w \, D^{-1} \, A) \, e^{(m)}.
\end{aligned} \tag{2.3}$$

[2] $A = D - L - U$ où D est la diagonale de A, $-L$ sa partie triangulaire strictement inférieure et $-U$ sa partie triangulaire strictement supérieure.

Maintenant que nous avons exprimé l'erreur à l'itération $(m+1)$ en fonction de l'erreur à l'itération précédente, nous allons pouvoir détailler la définition d'algébriquement lisse. Celle-ci stipule que les erreurs algébriquement lisses ne diminuent pas de manière significative lors de l'application d'une méthode de relaxation, ici celle de Jacobi pondéré. Cela signifie que si $e^{(m)}$ est une erreur algébriquement lisse, la valeur de $e^{(m+1)}$ ne sera pas tellement moins élevée que celle de $e^{(m)}$. Mais nous devons encore préciser ce que nous entendons par valeur de $e^{(m+1)}$.

Une manière de mesurer la valeur de l'erreur est d'utiliser sa A-norme. Introduisons d'abord le A-produit scalaire, de la manière suivante :

$$\langle e\,,\, e\rangle_A \;=\; \langle Ae\,,\, e\rangle\,.$$

La A-norme est la norme associée au A-produit scalaire et est définie comme telle :

$$\|e\|_A \;=\; \langle e\,,\, e\rangle_A^{1/2}\,.$$

En appliquant cette définition et (2.3), nous pouvons caractériser une erreur algébriquement lisse, e, par

$$\left\|(I \,-\, w\,D^{-1}\,A)\,e\right\|_A \;\approx\; \|e\|_A\,.$$

De cette équation, il est possible de montrer [4], en posant $w = \alpha\left\|D^{-1/2}AD^{-1/2}\right\|^{-1}$ avec $\alpha \in (0,2)$, que :

$$\left\langle D^{-1}Ae\,,\, Ae\right\rangle \;\ll\; \langle e\,,\, Ae\rangle\,,$$

c'est-à-dire, en utilisant l'équation résiduelle (1.4),

$$\left\langle D^{-1}r\,,\, r\right\rangle \;\ll\; \langle e\,,\, r\rangle\,,$$

ou encore

$$\sum_{i=1}^{n}\frac{r_i^2}{a_{ii}} \;\ll\; \sum_{i=1}^{n} r_i\,e_i.$$

Dès lors, nous pouvons dire qu'en moyenne une erreur algébriquement lisse, e, satisfait la relation suivante :

$$|r_i| \;\ll\; a_{ii}\;|e_i|\,,$$

ce qui signifie qu'une erreur algébriquement lisse correspond à un petit résidu. Cela implique que

$$A\,e \;\approx\; 0. \tag{2.4}$$

De plus, comme $r_i \approx 0$, nous obtenons de $Ae = r$ que

$$a_{ii}\,e_i \;\approx\; -\sum_{j\neq i} a_{ij}\,e_j, \tag{2.5}$$

c'est-à-dire que chacune des composantes, e_i, d'une erreur algébriquement lisse peut être exprimée comme une combinaison linéaire de ses autres composantes e_j, $j \neq i$. Cette dernière propriété nous sera utile lors de la construction de l'opérateur d'interpolation.

Notons qu'il est, en général, plus courant de travailler avec le schéma de relaxation de Gauss-Seidel (1.7). Une analyse du même type sur cette méthode de relaxation mène aux mêmes conclusions que pour la méthode de Jacobi pondéré [4].

Dans la première section de ce chapitre, nous avions défini les connexions entre deux noeuds. Dans la section suivante, nous allons préciser davantage la relation entre ces noeuds en introduisant les notions d'influence et dépendance. Ces concepts serviront par la suite pour la sélection des points de la grille grossière.

2.3 Relation d'influence et dépendance

Rappelons que nous travaillons avec une M-matrice symétrique, c'est-à-dire une matrice définie positive, dont les éléments diagonaux sont strictement positifs et les composantes hors-diagonales sont négatives. Ceci implique [4] que les éléments diagonaux de la matrice A du système (2.1) sont dominants (dans le sens où nous avons[3] que la somme des valeurs absolues des composantes hors-diagonales est inférieure à la somme des éléments diagonaux). Dès lors, nous pouvons dire qu'en moyenne l'élément diagonal a_{ii} est grand par rapport aux composantes hors-diagonales a_{ij}, $j \neq i$, pour la ligne i, $i \in \{1, \ldots, n\}$. Nous pouvons donc associer la $i^{ème}$ inconnue, u_i, à la $i^{ème}$ équation du système (2.1).

Considérons la $i^{ème}$ équation. A l'aide de celle-ci, nous allons chercher quelles variables, u_j, $j \neq i$, influencent l'inconnue u_i pour pouvoir par la suite définir une interpolation. Un moyen de trouver ces variables est de regarder leur coefficient a_{ij} : si celui-ci est élevé par rapport aux coefficients des autres éléments de la $i^{ème}$ ligne, alors nous pouvons conclure que l'inconnue u_j est importante dans la détermination de u_i. De cette constatation, nous pouvons dire que la variable u_j pourra servir à interpoler l'inconnue u_i et le point j sera un bon candidat pour faire partie de la grille grossière. Cette relation est décrite de manière plus précise dans la définition suivante (en nous rappelant que les points d'une grille sont associés aux indices des inconnues) :

Définition 2. *L'inconnue u_i (ou le point i) dépend fortement de la variable u_j (ou du point j) si*

$$-a_{ij} \geq \theta \max_{k \neq i} \{-a_{ik}\} \qquad \text{où } 0 < \theta \leq 1.$$

Cette définition signifie que le point i dépend fortement du point $j \neq i$ si a_{ij} est comparable au plus grand coefficient de la $i^{ème}$ ligne. En inversant, nous obtenons la définition suivante :

[3] En effet, nous avons que A est une matrice définie positive, c'est-à-dire $\forall\, x \neq 0, \in \mathbb{R}^n\ x^T A x > 0$. D'où, si nous prenons $x = (1\,1\,\ldots\,1)^T$, nous avons que $\sum_{i,j=1}^{n} a_{ij} > 0$. Nous pouvons donc conclure que $\sum_{i=1}^{n} \sum_{j \neq i} |a_{ij}| < \sum_{i=1}^{n} a_{ii}$ car A est une M-matrice.

Définition 3. *L'inconnue u_j (ou le point j) influence fortement la variable u_i (ou le point i) si l'inconnue u_i (ou le point i) dépend fortement de la variable u_j (ou du point j).*

A partir de ces notions d'influence et de dépendance, nous pouvons déjà définir deux types d'ensembles associés à chaque point :
- S_i, l'ensemble des points qui influencent fortement i et
- S_i^T, l'ensemble des points qui dépendent fortement de i.

Ces ensembles nous serviront par la suite pour la sélection des points de la grille grossière.

Les concepts vus dans cette section et la précédente (erreur algébriquement lisse, dépendance et influence) sont très importants pour la suite des évènements, c'est-à-dire pour la méthode de sélection des points de la grille grossière et la détermination de l'opérateur d'interpolation.

Parallèlement aux cas des multigrilles géométriques, nous avons encore à définir la grille grossière telle que les composantes lisses puissent y être représentées avec précision et telle qu'elle contienne significativement moins de points que la grille fine ; il nous reste ensuite à définir un opérateur d'interpolation tel que les fonctions lisses puissent être ramenées d'une grille grossière à une grille fine avec précision ; finalement, il nous faudra définir les opérateurs restants (opérateur de restriction et matrice A^{2h} du niveau grossier) en appliquant les propriétés variationnelles. Nous allons commencer, dans la section suivante, par déterminer l'opérateur d'interpolation.

2.4 Opérateur d'interpolation

Dans cette section, nous allons décrire comment construire un opérateur d'interpolation. Nous étudierons deux méthodes d'interpolation, l'une présentée dans [14] et une autre décrite dans [4] qui est celle qui a été implémentée dans le cadre de ce mémoire. Nous allons tout d'abord définir notre cadre de travail et établir une propriété importante concernant les erreurs lisses.

Partageons l'ensemble des points de la grille de départ, Ω^h, en deux ensembles distincts :
- F, l'ensemble des points appartenant à la grille fine uniquement, appelés F-points, et
- C, l'ensemble des points qui formeront la grille grossière, appelés C-points.

Dès lors, nous avons que $\Omega^h = \{1, \ldots, n\} = F \cup C$ et $F \cap C = \emptyset$. Supposons que dans cette section ce partage a déjà été effectué.

Pour définir correctement un opérateur d'interpolation, il nous faut un ensemble C tel que chaque vecteur lisse appartenant à la grille fine, Ω^h, puisse être interpolé de manière précise, c'est-à-dire qu'un vecteur lisse, $e \in \mathbb{R}^n$, doit pouvoir être interpolé

précisément à partir d'un ensemble de ses composantes, e_i, $i \in C$, définis sur la grille grossière.

Pour construire un opérateur d'interpolation, nous allons nous servir des notions d'influence, de dépendance et d'algébriquement lisse. En effet, si un point de la grille grossière, $j \in C$, influence fortement un F-point, $i \in F$, alors le point j doit servir à interpoler le point i. De plus, nous nous servirons de la propriété des erreurs lisses suivante : soit e une erreur lisse, nous avons [14] que :

$$\sum_{j \neq i} \left(\frac{|a_{ij}|}{a_{ii}} \right) \left(\frac{e_i - e_j}{e_i} \right)^2 \ll 1, \qquad 1 \leq i \leq n. \tag{2.6}$$

Nous remarquons que, dans (2.6), comme nous avons que la somme des termes positifs est fortement plus petite que 1, les termes sont eux-mêmes fortement plus petits que 1. De plus, si e_i dépend fortement de e_j, nous avons par définition de dépendance (Définition 2) que $|a_{ij}|$ est comparable au plus grand élément de la ligne i, ce qui implique que $\left(\frac{|a_{ij}|}{a_{ii}} \right)$ est "grand", dans le sens où il est proche de 1. Dès lors, par (2.6), nous pouvons dire que $e_i - e_j \approx 0$, c'est-à-dire que $e_i \approx e_j$. Cela signifie qu'*une erreur lisse varie peu en direction d'une connexion forte.* Ce qui justifie l'idée d'interpoler une quantité de la grille fine u_i par une composante de la grille grossière, u_j, telle que $j \in S_i$ (c'est-à-dire que j influence fortement i).

Dans la section suivante, nous allons décrire la méthode d'interpolation due à [4] qui a été implémentée dans le cadre de ce mémoire. Ensuite, nous consacrerons une autre rubrique à l'une des méthodes d'interpolation de [14], celle spécifique aux problèmes traitant des M-matrices.

2.4.1 Interpolation selon [4]

Définissons maintenant une série d'ensembles qui nous serviront dans la définition de l'opérateur d'interpolation. Pour chaque point i appartenant à la grille fine, Ω^h, nous définissons le voisinage du point i, noté N_i, comme étant l'ensemble des points $j \neq i$ tel que $a_{ij} \neq 0$, c'est-à-dire $N_i = \{j \in \Omega^h \mid j \neq i \text{ et } a_{ij} \neq 0\}$. Cet ensemble se partage en trois sous-ensembles distincts :

- C_i, l'ensemble des points voisins de la grille grossière, C, qui influencent fortement i, c'est-à-dire :

$$C_i = N_i \cap S_i \cap C = S_i \cap C,$$

  car $S_i \subset N_i$.
- D_i^s, l'ensemble des points voisins de la grille fine uniquement, F, qui influencent fortement i, c'est-à-dire :

$$D_i^s = N_i \cap S_i \cap F = S_i \cap F.$$

- D_i^w, l'ensemble des points voisins de la grille fine, Ω^h, qui n'influencent pas fortement i, c'est-à-dire :

$$D_i^w = N_i \cap (S_i)^c,$$

où $(S_i)^c = \Omega^h \backslash S_i$ est le complémentaire de S_i.

Nous avons que $N_i = C_i \cup D_i^s \cup D_i^w$ et que $C_i \cap D_i^s \cap D_i^w = \emptyset$.

Nous pouvons maintenant définir l'opérateur d'interpolation, noté I_{2h}^h comme dans le chapitre précédent, de la manière suivante : la $i^{ème}$ composante de $I_{2h}^h e$ est déterminée par :

$$(I_{2h}^h e)_i = \begin{cases} e_i & \text{si } i \in C \\ \sum_{j \in C_i} w_{ij}\, e_j & \text{si } i \in F, \end{cases} \tag{2.7}$$

où w_{ij} sont les constantes de poids qu'il nous reste à déterminer.

Notons que pour s'assurer d'une bonne interpolation, il est d'usage de vérifier qu'un vecteur constant, e, est interpolé exactement par l'opérateur I_{2h}^h, c'est-à-dire que $I_{2h}^h e = e$. Pour cela, il faut que $\sum_{j \in C_i} w_{ij} = 1$ pour tout point $i \in F$ dans (2.7). Pour vérifier cette condition, on se limite en général ([4] et [14]) à un type particulier de matrices : celles dont la somme des composantes de chaque ligne est nulle. Nous reviendrons sur cette condition par la suite.

Revenons à présent à notre opérateur d'interpolation (2.7) dont nous allons déterminer les constantes de poids. Nous avons vu en début de chapitre qu'une erreur lisse, e, correspond à un petit résidu (cfr (2.4)). Dès lors, par (2.5) et la définition de voisinage, nous pouvons écrire :

$$a_{ii}\, e_i \approx - \sum_{j \in N_i} a_{ij}\, e_j,$$

et en utilisant la décomposition $N_i = C_i \cup D_i^s \cup D_i^w$, nous obtenons :

$$a_{ii}\, e_i \approx - \sum_{j \in C_i} a_{ij}\, e_j - \sum_{m \in D_i^s} a_{im}\, e_m - \sum_{l \in D_i^w} a_{il}\, e_l. \tag{2.8}$$

Comme nous souhaitons définir la $i^{ème}$ composante du vecteur lisse en fonction des éléments de la grille grossière (e_j, $j \in C_i$), nous devons exprimer autrement les deux dernières sommes du membre de droite de l'équation (2.8).

Traitons d'abord la troisième somme de l'équation (2.8). En remplaçant e_l ($l \in D_i^w$) par e_i dans (2.8), nous obtenons :

$$\left(a_{ii} + \sum_{l \in D_i^w} a_{il} \right) e_i \approx - \sum_{j \in C_i} a_{ij}\, e_j - \sum_{m \in D_i^s} a_{im}\, e_m. \tag{2.9}$$

Cette approximation est justifiée. En effet, comme les points appartenant à l'ensemble D_i^w n'influencent pas fortement le point i, les éléments a_{il} ($l \in D_i^w$) sont très

petits. Dès lors, l'erreur engendrée par cette approximation ne peut être que très faible. De plus, si nous avions sous-estimé la dépendance entre le point i et certains points de D_i^w (c'est-à-dire qu'il existe au moins un point l dans D_i^w qui influence fortement i), nous aurions pour ces points-là que $e_l \approx e_i$ puisque les erreurs lisses varient peu en direction d'une connexion forte.

La deuxième somme du membre de droite de l'équation (2.8) concerne les points qui influencent fortement le point i et qui appartiennent à F. Une possibilité serait de remplacer les e_m par e_i dans cette somme. Cette solution est efficace pour beaucoup de problèmes. Cependant, les expériences montrent [4] qu'il est préférable d'exprimer ces composantes en fonction des points de C_i, dans le sens où :

$$\forall m \in D_i^s \qquad e_m \approx \frac{\sum_{k \in C_i} a_{mk}\, e_k}{\sum_{k \in C_i} a_{mk}}. \tag{2.10}$$

En effet, le numérateur résulte du fait que les composantes e_m dépendent des éléments e_k dans les proportions des coefficients a_{mk}. Le dénominateur assure que, dans le cas particulier des matrices dont la somme des composantes de chaque ligne est nulle, un vecteur constant, e, puisse être interpolé exactement, nous vérifierons cette condition par la suite. Notons aussi que l'approximation (2.10) nécessite que pour tous les points $m \in D_i^s$, il existe au moins un point $k \in C_i$ tel que a_{mk} soit non nul, ceci est vérifié lorsque m dépend fortement de k, c'est-à-dire que :

$$\forall m \in D_i^s \quad \exists k \in C_i \quad m \in S_k^T. \tag{2.11}$$

En substituant l'équation (2.10) dans (2.9), nous obtenons :

$$\left(a_{ii} + \sum_{l \in D_i^w} a_{il} \right) e_i \approx - \sum_{j \in C_i} a_{ij}\, e_j - \sum_{m \in D_i^s} a_{im} \frac{\sum_{j \in C_i} a_{mj}\, e_j}{\sum_{k \in C_i} a_{mk}},$$

c'est-à-dire, en réarrangeant les termes :

$$\left(a_{ii} + \sum_{l \in D_i^w} a_{il} \right) e_i \approx - \sum_{j \in C_i} a_{ij}\, e_j - \sum_{j \in C_i} \sum_{m \in D_i^s} \left(\frac{a_{im}\, a_{mj}}{\sum_{k \in C_i} a_{mk}} \right) e_j,$$

ou encore :

$$\left(a_{ii} + \sum_{l \in D_i^w} a_{il} \right) e_i \approx - \sum_{j \in C_i} \left[a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im}\, a_{mj}}{\sum_{k \in C_i} a_{mk}} \right) \right] e_j. \tag{2.12}$$

A l'aide de l'équation (2.12), nous pouvons définir les coefficients de poids w_{ij} dans (2.7) :

$$w_{ij} = -\frac{a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im}\, a_{mj}}{\sum_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum_{l \in D_i^w} a_{il}}. \tag{2.13}$$

Nous remarquons que le dénominateur des coefficients de poids est toujours non nul puisque la matrice A est une M-matrice. Cependant, pour que le numérateur existe, il faut que la condition (2.11) soit vérifiée pour tous les points de F.

Vérifions à présent que $\sum_{j \in C_i} w_{ij} = 1$, où les w_{ij} sont donnés par (2.13), dans le cas où $\sum_j a_{ij} = 0$ pour tout point $i \in F$. Soit i un point quelconque de F :

$$\sum_{j \in C_i} w_{ij} = 1$$

est équivalent à, par (2.13) :

$$-\sum_{j \in C_i} \left(a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im}\, a_{mj}}{\sum_{k \in C_i} a_{mk}} \right) \right) = a_{ii} + \sum_{l \in D_i^w} a_{il},$$

c'est-à-dire :

$$-\sum_{j \in C_i} a_{ij} - \sum_{m \in D_i^s} \left(\frac{a_{im} \sum_{j \in C_i} a_{mj}}{\sum_{k \in C_i} a_{mk}} \right) = a_{ii} + \sum_{l \in D_i^w} a_{il},$$

d'où nous avons que :

$$-\sum_{j \in C_i} a_{ij} - \sum_{m \in D_i^s} a_{im} = a_{ii} + \sum_{l \in D_i^w} a_{il},$$

ce qui est bien vérifié puisque nous avons considéré le cas particulier des matrices dont la somme des composantes de chaque ligne est nulle.

2.4.2 Interpolation selon [14]

Nous allons maintenant présenter la méthode d'interpolation définie par [14]. Rappelons que dans cette section, nous supposons que nous avons déjà une partition en C et F de la grille fine Ω^h. Nous définissons, pour tous les points i de F, C_i

comme étant *l'ensemble d'interpolation du point i* (cet ensemble correspond à l'ensemble C_i de la section précédente, c'est-à-dire l'ensemble des points qui influencent fortement i et qui appartiennent à la grille grossière). Nous supposerons que C_i est non vide pour tous les points $i \in F$.

Comme dans la rubrique précédente, nous définissons l'opérateur d'interpolation de la manière suivante : la $i^{ème}$ composante de $I_{2h}^h e$ est déterminée par :

$$(I_{2h}^h e)_i = \begin{cases} e_i & \text{si } i \in C \\ \sum_{k \in C_i} w_{ik}\, e_k & \text{si } i \in F, \end{cases} \tag{2.14}$$

où les w_{ik} sont les coefficients de poids à déterminer et les C_i sont les ensembles d'interpolation correspondant aux points de F.

Pour définir ces coefficients de poids, nous allons nous servir de la propriété des erreurs lisses (2.5) qui, combinée avec la définition de voisinage, donne pour une erreur lisse e :

$$a_{ii}\, e_i \approx - \sum_{j \in N_i} a_{ij}\, e_j. \tag{2.15}$$

En négligeant les termes qui n'appartiennent pas à l'ensemble d'interpolation (puisque ceux-ci sont considérés comme faibles), nous pouvons tout d'abord écrire, pour une erreur lisse e, l'approximation suivante :

$$\frac{\sum_{j \in N_i} a_{ij} e_j}{\sum_{j \in N_i} a_{ij}} \approx \frac{\sum_{k \in C_i} a_{ik} e_k}{\sum_{k \in C_i} a_{ik}}, \tag{2.16}$$

où les dénominateurs sont des constantes de normalisation. En introduisant (2.16) dans (2.15), nous obtenons :

$$a_{ii}\, e_i \approx - \alpha_i \sum_{k \in C_i} a_{ik}\, e_k,$$

avec

$$\alpha_i = \frac{\sum_{j \in N_i} a_{ij}}{\sum_{k \in C_i} a_{ik}}. \tag{2.17}$$

D'où, nous pouvons définir les coefficients de poids de l'opérateur d'interpolation (2.14) par :

$$w_{ik} = - \alpha_i \frac{a_{ik}}{a_{ii}}. \tag{2.18}$$

Remarquons que nous avons, par (2.18) :

$$\begin{aligned}\sum_{j\in\Omega^h} a_{ij} &= a_{ii} + \sum_{j\in N_i} a_{ij} \\ &= a_{ii}\left(1 + \frac{\sum_{j\in N_i} a_{ij}}{\sum_{k\in C_i} a_{ik}} \frac{\sum_{k\in C_i} a_{ik}}{a_{ii}}\right) \\ &= a_{ii}\left(1 - \sum_{k\in C_i} w_{ik}\right).\end{aligned}$$

Dès lors, nous avons que $\sum_{k\in C_i} w_{ik} = 1$ lorsque $\sum_{j\in\Omega^h} a_{ij}$ est nulle. Cela signifie que (comme pour l'opérateur d'interpolation de [4]) dans le cas de matrices dont la somme des composantes de chaque ligne est nulle, les constantes sont interpolées exactement.

La formule d'interpolation de [14] ne nécessite pas de condition d'existence contrairement à la méthode de [4]. En effet, le dénominateur de (2.17) qui intervient dans la définition (2.18) de w_{ik} est toujours non nul, puisqu'on suppose qu'il y a toujours au moins un élément dans C_i.

Pour construire l'opérateur d'interpolation, nous avions supposé que le partage des points de la grille de départ en C-points et F-points avait déjà été effectué. Dans la section suivante, nous allons présenter la méthode pour créer cette partition C-F de la grille la plus fine. Nous utiliserons tous les objets déjà introduits au cours de ce chapitre et nous veillerons à ce que la grille grossière, comme dans le cas géométrique, vérifie les hypothèses suivantes :

- Les erreurs lisses pourront être approximées avec précision sur la grille grossière.
- Les fonctions lisses pourront être interpolées avec précision de la grille grossière à la grille fine.
- La grille grossière comportera significativement moins de points que la grille fine telle que la résolution du problème sur cette grille soit peu coûteuse.

2.5 Méthodes de sélection des points de la grille grossière

Comme pour l'opérateur d'interpolation, nous allons traiter dans cette section deux méthodes de sélection des points de la grille grossière : l'une due à [4] et l'autre à [14]. Chacune de ces sélections est associée au type d'interpolation vu dans la section précédente.

2.5.1 Sélection des points selon [4]

Nous allons présenter dans cette rubrique la méthode de sélection des points de la grille grossière définie dans [4]. Cette méthode se déroule en deux étapes : la première étape consiste à faire une première partition C-F en se basant essentiellement sur les concepts d'influence-dépendance, et la seconde étape à pour but de garantir l'existence de l'opérateur d'interpolation en tous les points de la grille fine, Ω^h. Notre algorithme de sélection des points de la grille grossière tentera aussi de respecter deux règles de construction. La première règle de construction est la suivante :

H1 : L'ensemble C sera l'ensemble maximal de points tel qu'aucun point de C ne dépende fortement d'un autre point de C.

C'est-à-dire :

$$C = \max \{E \subset \Omega^h \mid \forall i \in E \ \nexists j \in E \quad j \in S_i\}. \tag{2.19}$$

Ce critère vise essentiellement à garder une taille acceptable pour la grille grossière, tout en assurant une bonne approximation des erreurs lisses sur la grille grossière. Le deuxième critère de sélection est décrit comme suit :

H2 : Pour chaque F-point i, chaque point $j \in S_i$ influençant fortement i sera soit un point de l'ensemble d'interpolation grossier C_i, soit dépendra fortement d'au moins un point de C_i.

C'est-à-dire :

$$\forall i \in F \qquad \forall j \in S_i \quad \begin{cases} \text{soit } j \in C_i, \\ \text{soit } \exists k \in C_i \quad j \in S_k^T. \end{cases} \tag{2.20}$$

Cette deuxième règle assure l'existence de l'opérateur d'interpolation en tous les points de la grille fine, Ω^h. En effet, nous avons vu que la condition d'existence de l'opérateur d'interpolation (2.7) était (cfr (2.11)) :

$$\forall i \in F \quad \forall j \in D_i^s \quad \exists k \in C_i \quad j \in S_k^T,$$

ce qui est équivalent à la règle **H2**, puisque $D_i^s = S_i \cap F$ par définition.

Il est à noter que les deux critères **H1** et **H2** ne sont pas toujours réalisables en même temps [4]. Comme la construction de l'opérateur d'interpolation demande que la règle **H2** soit satisfaite, nous préférerons garantir **H2** et nous servir de **H1** comme un guide.

Les règles de construction assure une bonne approximation des erreurs lisses de la grille grossière à la grille fine, tout en gardant une taille raisonnable pour la grille grossière.

Dans [4], l'algorithme de sélection des points de la grille grossière procède en deux étapes, nommées schémas de coloriage. Le premier schéma consiste à définir une première partition des points, en F-points et C-points. Ce premier tri à pour but de créer une répartition de départ telle que C soit un bon ensemble d'interpolation. Ce premier schéma tend à satisfaire **H1**. Le deuxième schéma à pour but de passer en revue tous les F-points de la répartition initiale et de changer ceux-ci en C-point si nécessaire pour assurer le critère **H2**.

Premier schéma de coloriage

La stratégie du premier schéma de coloriage consiste à d'abord assigner à chaque point i de la grille fine, Ω^h, un potentiel à appartenir à la grille grossière, C. Ce potentiel, noté λ_i, est calculé à partir des ensembles S_i^T. Plus précisément, λ_i est égal au cardinal de S_i^T, c'est-à-dire au nombre de points qui dépendent fortement du point i.

Lorsque chaque point a reçu son potentiel λ_i, nous choisissons le point i tel qu'il a le λ_i maximal. Ce point i est alors placé dans l'ensemble C et tous les points j qui dépendent fortement de i deviennent des F-points. Ceci est justifié puisque ces F-points, $j \in S_i^T$, peuvent déjà être interpolés par au moins un point de C, i.

Après, nous regardons les autres points qui pourraient aider à approximer ces nouveaux F-points, j. Ces points qui peuvent servir à interpoler les F-points, j, sont les points $k \in S_j$ qui influencent fortement les j. Dès lors, nous ajoutons une unité à chaque potentiel λ_k.

Ensuite, nous choisissons le suivant point i non classé avec un λ_i maximal et nous le plaçons dans l'ensemble C. Après, nous mettons tous les points $j \in S_i^T$ non classés dans F. Puis, pour tous les nouveaux F-points, j, nous augmentons les potentiels des points $k \in S_j$ non classés.

Après nous procédons de manière identique, jusqu'à ce que tous les points de la grille de départ soient classés soit dans C soit dans F.

Remarquons que ce tri dépend de l'ordre de passage des points. En effet, lors du choix du point ayant un potentiel λ_i maximal, il arrive souvent qu'il y ait plusieurs candidats. Ainsi, comme le reste de la procédure dépend de ces points i, il existe beaucoup de possibilités de tri.

Deuxième schéma de coloriage

Le deuxième schéma de coloriage consiste à repasser tous les F-points de la répartition obtenue par le premier tri pour vérifier si le critère **H2** est satisfait et ainsi assurer l'existence de l'opérateur d'interpolation.

Ce tri se déroule de la manière suivante : pour tous les points $i \in F$, on regarde si tous les points appartenant à D_i^s dépendent fortement d'au moins un point de C_i. Lorsque l'on trouve un point $l \in D_i^s$ qui ne vérifie pas cette règle, alors ce point l devient un candidat pour la grille grossière et est placé temporairement dans C. Ensuite, la vérification des points de D_i^s continue, et si tous ces points vérifient la condition, le candidat l devient définitivement un C-point. Par contre, si l'on rencontre un autre point j de D_i^s qui ne dépend d'aucun C-point, alors la vérification des points de D_i^s s'arrête, le point l est replacé dans l'ensemble F et c'est le point i qui devient un point de C (en effet, cela signifie qu'il existe au moins deux F-points l et j qui influencent fortement i et qui ne dépendent d'aucun point de C). On passe alors au point suivant de F.

Nous avons présenté ici une version théorique des deux schémas de coloriage de [4], nous décrirons davantage, dans la dernière section de ce chapitre, ces méthodes sous forme algorithmique et nous les illustrerons à l'aide d'exemples. Dans la section suivante, nous allons introduire une autre technique de sélection de points de la grille grossière présentée dans [14].

2.5.2 Sélection des points selon [14]

Deux méthodes de sélection sont présentées dans [14]. La première méthode de sélection, dite standard, ressemble fortement au premier schéma de coloriage de [4]. La deuxième méthode de sélection, dite agressive, pousse plus loin la notion de forte dépendance entre les points, en définissant des "chemins de connexion". Nous allons commencer par la sélection standard.

Sélection standard

Cette méthode se base sur la définition d'influence-dépendance et les ensembles qui en résultent, S_i et S_i^T. Dans cette méthode, nous commençons aussi par assigner à chaque variable de la grille de départ, Ω^h, un potentiel à appartenir à la grille grossière, C. Ce potentiel, noté λ_i, est défini à chaque étape par :

$$\lambda_i = \sharp(S_i^T \cap U) + 2\,\sharp(S_i^T \cap F) \qquad (\forall\, i \in U)$$

où U est l'ensemble des points qui ne sont pas encore classés (il vaut Ω^h à la première étape) et $\sharp(S_i^T \cap U)$ désigne le cardinal de l'ensemble $S_i^T \cap U$.

L'algorithme standard procède de la façon suivante :

1. Poser $F := \emptyset$, $C := \emptyset$ et $U := \Omega^h$.
2. Poser
$$\lambda_i = \sharp(S_i^T \cap U) + 2\,\sharp(S_i^T \cap F) \quad (\forall\, i \in U).$$
3. Répéter 3.1 à 3.3 jusqu'à ce que $\lambda_i = 0$ $(\forall\, i \in U)$.

3.1 Prendre $i \in U$ avec un λ_i maximal et poser

$$C := C \cup \{i\},\ U := U \backslash \{i\}.$$

3.2 $\forall j \in S_i^T \cap U$ faire :

$$F := F \cup \{j\},\ U := U \backslash \{j\}.$$

3.3 Poser

$$\lambda_i = \sharp(S_i^T \cap U) + 2\sharp(S_i^T \cap F) \quad (i \in U, \text{ localement}).$$

L'étape 3.3 est appliquée uniquement localement, ce qui signifie que seuls les points pour lesquels $S_i^T \cap U$ et $S_i^T \cap F$ ont changé seront mis à jour. Ces points correspondent aux points qui dépendent fortement d'au moins un nouveau point de F ou C. Cet algorithme est quasi identique au premier schéma de coloriage vu dans la section précédente, puisque la seule différence se trouve au niveau de la mise à jour du λ_i à l'étape 3.3.

Sélection agressive

La seconde méthode proposée par [14] se base sur une définition plus étendue de forte connexion pour y inclure des points qui ne sont pas *directement connectés*. Introduisons, d'abord, la notion de *fortement n-connecté le long d'un chemin* :

Définition 4.
Le point i est fortement n-connecté à un point j le long d'un chemin de longueur l s'il existe un ensemble de points $\{i_0, i_1, \cdots, i_l\}$ où $i = i_0$ et $j = i_l$ tel que

$$i_{k+1} \in S_{i_k} \quad \forall k = 0, \cdots, l-1.$$

Une deuxième définition qui découle de la précédente est :

Définition 5.
Soient $p \geq 1$ et $l \geq 1$ des valeurs fixées, le point i est fortement connecté à une variable j de paramètres (p, l) s'il existe au moins p chemins de longueur inférieure ou égale à l tels que i est fortement n-connecté à un point j le long de ces chemins, au sens de la définition précédente.

La méthode de sélection agressive consiste à appliquer le même schéma que l'algorithme standard, en remplaçant les ensembles S_i par les ensembles :

$$S_i^{p,l} = \{j \in \Omega^h \mid i \text{ est fortement connecté à } j \text{ de paramètres } (p, l)\}. \tag{2.21}$$

En pratique, cette méthode n'est pas utilisée telle quelle. En effet, seuls les cas où $p = 2$, $l = 2$ et $p = 1$, $l = 2$ sont efficaces, nous les noterons respectivement $A2$-sélection et $A1$-sélection. De plus, la méthode agressive est intéressante uniquement

lorsque nous travaillons avec plus d'un niveau grossier.

L'inconvénient de ces méthodes est qu'elles demandent beaucoup de mémoire pour le stockage des ensembles $S_i^{p,l}$ et de leurs transposés, pour tous les points i de la grille. Une façon de remédier à ce problème est d'appliquer le schéma standard en deux étapes. La première consiste à appliquer l'algorithme expliqué dans la section précédente. La seconde étape consiste à appliquer l'algorithme standard uniquement aux points de C et en remplaçant les ensembles S_i par un équivalent de (2.21) :

$$\widehat{S}_i^{p,l} = \{j \in C \mid i \text{ est fortement connecté à } j \text{ de paramètres } (p,l)\}.$$

Les nouveaux C-points obtenus après le second passage forment la grille grossière.

Maintenant que nous avons nos opérateurs d'interpolation, nous allons pouvoir définir l'opérateur de restriction et l'opérateur A^{2h} sur la grille grossière à l'aide des propriétés variationnelles. Ceci est l'objet de la section suivante.

2.6 Opérateurs restants

Comme dans le cas des multigrilles géométriques, nous allons définir l'opérateur de restriction et l'opérateur A^{2h} sur le niveau grossier à l'aide des propriétés variationnelles. Par (1.12), nous pouvons définir l'opérateur de restriction I_h^{2h} par :

$$I_h^{2h} = (I_{2h}^h)^T,$$

où I_{2h}^h est l'opérateur d'interpolation (2.7). Par la condition de Galerkin (1.13), nous obtenons l'opérateur A^{2h} :

$$A^{2h} = I_h^{2h} A^h I_{2h}^h.$$

Maintenant que nous disposons de tous les outils nécessaires aux méthodes multigrilles, nous allons dans la section suivante redéfinir les algorithmes récursifs pour le cas des multigrilles algébriques.

2.7 Algorithmes récursifs

Avant de définir les algorithmes récursifs, adaptons le schéma de correction à deux grilles vues au chapitre précédent (section 1.2.3) au cas des multigrilles algébriques. Notons que la seule différence entre les multigrilles géométriques et les multigrilles algébriques se trouve dans la définition des différents opérateurs et dans la manière de sélectionner les points.

Rappelons que nous utilisons les mêmes notations que dans le cas des multigrilles géométriques pour les différents objets, c'est-à-dire en désignant par un exposant h, les objets de la grille fine et par un exposant $2h$ les objets de la grille grossière. Le

schéma de correction à deux grilles pour le cas algébrique est décrit comme suit :

Schéma de correction à deux grilles

$$v^h \leftarrow AMG(v^h, f^h)$$

- Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ sur Ω^h avec pour point de départ v^h.
- Calculer le résidu sur la grille fine, $r^h = f^h - A^h v^h$, et le restreindre à la grille grossière Ω^{2h} par $r^{2h} = I_h^{2h} r^h$.
- Résoudre $A^{2h} e^{2h} = r^{2h}$ sur Ω^{2h} avec comme point de départ $e^{2h} = 0$.
- Interpoler l'erreur e^{2h} sur la grille fine par la formule $e^h = I_{2h}^h e^{2h}$ et corriger l'approximation sur la grille fine par $v^h \leftarrow v^h + e^h$.
- Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ sur Ω^h avec pour point de départ v^h.

Les paramètres ν_1 et ν_2 indiquent le nombre maximum d'itérations à appliquer lors de la première et dernière relaxation du système (avant et après la correction sur la grille grossière). Le schéma de relaxation utilisé lors de la procédure est en général celui de Gauss-Seidel. Le schéma "nested" peut être défini de la même manière à partir de celui présenté dans le chapitre précédent (section 1.2.1).

Comme dans le cas des multigrilles géométriques, nous pouvons définir des schémas récursifs à partir du schéma de correction à deux grilles et du schéma "nested". Les algorithmes récursifs sont identiques à ceux présentés dans le chapitre précédent (sections 1.2.4 à 1.2.6). Nous reprenons ici le schéma V-cycle car celui-ci est le plus répandu et celui que nous utiliserons lors de l'implémentation de la méthode :

Schéma V-cycle (récursif)

$$v^h \leftarrow V^h(v^h, f^h)$$

1. Appliquer ν_1 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.
2. Si Ω^h est la grille la plus grossière, alors *résoudre le problème* $A^h u^h = f^h$ *exactement.*
   Sinon
   $f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$,
   $v^{2h} \leftarrow 0$,
   $v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h})$.
3. Corriger l'approximation $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
4. Appliquer ν_2 fois le schéma de relaxation à $A^h u^h = f^h$ avec pour point de départ v^h.

Remarquons, dans cet algorithme, que lorsque le problème est amené sur la grille la plus grossière, il est résolu exactement. Or dans le cas des multigrilles géométriques, lorsque le problème arrivait sur la grille la plus grossière, il subissait une relaxation (étape 4). Néanmoins, cette relaxation pouvait déjà être remplacée par une résolution exacte, comme nous l'avions défini dans la stratégie V-cycle (section 1.2.4).

Les algorithmes de multigrilles algébriques, nommés AMG, procèdent en deux phases. La première phase, portant le nom de *phase de préparation*, consiste à définir récursivement les opérateurs d'interpolations et les opérateurs A^{kh} (où $k = 2, 4$, etc) des niveaux grossiers. La deuxième phase, nommée *phase de solution*, consiste à appliquer récursivement l'un des schémas du multigrille (sections 1.2.4 à 1.2.6), dont le plus communément utilisé est le schéma V-cycle défini ci-dessus, jusqu'au niveau de tolérance désiré. Dans la section suivante, nous allons voir quelques indices de coût.

2.8 Coût

Il est difficile de déterminer à l'avance le coût d'un algorithme de multigrille algébrique, que ce soit du point de vue du stockage ou du point de vue de la complexité. En effet, il n'est pas possible, contrairement au cas géométrique, de connaître à l'avance le taux de points qui va être choisi pour les grilles grossières. Néanmoins, nous pouvons définir deux indices de coût, l'un lié au stockage : la *Complexité de grille*, et l'autre lié au stockage et à la complexité de l'algorithme : la *Complexité d'opérateur*. En voici, les définitions :

Définition 6. *La complexité de grille est le nombre total de noeuds présents sur toutes les grilles Ω^{kh} (où $k = 1, 2, 4$, etc), divisé par le nombre total de points que comporte la grille la plus fine Ω^h.*

Définition 7. *La complexité d'opérateur est le nombre total d'éléments non nuls contenus dans toutes les matrices A^{kh} (où $k = 1, 2, 4$, etc), divisé par le nombre d'éléments non nuls contenus dans la matrice de départ A^h.*

La complexité de grille donne une idée de la place mémoire requise pour le stockage des membres de droites des systèmes définis sur chaque grille et des vecteurs d'approximations engendrés par l'algorithme V-cycle, vu dans la section précédente.

La complexité d'opérateur, quant à elle, nous donne deux indications, l'une concernant le stockage des opérateurs A^{kh} et une autre concernant le nombre d'opérations à effectuer. En effet, comme dans le cas géométrique, le coût dominant dans la phase de résolution [4] est celui de la relaxation et du calcul des résidus. Hors les coûts de ceux-ci sont directement proportionnels au nombre d'éléments non nuls

contenus dans les opérateurs. Dès lors, le coût d'un V-cycle est à peu près proportionnel à la complexité d'opérateur. La complexité d'opérateur est considérée comme un bon indice de coût pour l'algorithme V-cycle des multigrilles algébriques.

Dans la section suivante, nous allons davantage étudier l'algorithme AMG de [4] qui a été implémenté lors de ce mémoire.

2.9 Conception du programme à l'aide des algorithmes développés

Dans cette section, nous allons présenter les différents algorithmes nécessaires à l'implémentation de la méthode des multigrilles algébriques de [4]. Rappelons que l'algorithme global AMG procède en deux étapes : la *phase de préparation* et la *phase de résolution*. Dans le paragraphe suivant, nous allons commencer par détailler le fonctionnement de la phase de préparation.

2.9.1 Phase de préparation

La phase de préparation consiste à calculer les opérateurs d'interpolation et les opérateurs grossiers A^{kh} (où $k = 2$, 4, etc) pour chaque niveau grossier. Ces opérateurs sont ensuite stockés pour la phase suivante, dite de résolution.

La phase de préparation s'exécute en boucle et par niveau. Soit $l > 1$, le nombre de niveaux donnés, les grilles grossières seront identifiées par Ω^{kh} où $k = 2, \ldots, L$ avec $L = 2^{l-1}$. La stratégie de la phase de préparation est décrite comme tel :

Pour chaque niveau $m \in \{1, 2, \ldots, l-1\}$, faire :

1. Calculer les ensembles S_i et S_i^T pour tous les points i de la grille fine Ω^{kh} où $k = 2^{m-1}$.
2. Exécuter le premier schéma de coloriage pour obtenir une première répartition des points de la grille fine en les ensembles C et F.
3. Supprimer les ensembles S_i^T pour chaque point i de la grille fine Ω^{kh}.
4. Procéder au deuxième schéma de coloriage pour assurer l'existence de l'opérateur d'interpolation.
5. Construire l'opérateur d'interpolation I_{2kh}^{kh} et le stocker.
6. Déterminer l'opérateur grossier A^{2kh} et le stocker.

Dans les sections suivantes, nous allons détailler les différents points, 1 à 6, de la phase de préparation. Vu que l'algorithme de la phase de préparation procède par niveau, nous expliquerons les différentes étapes à l'aide de deux grilles : une fine Ω^h et

une grossière Ω^{2h}. Nous allons commencer par la construction des ensembles S_i et S_i^T.

2.9.2 Les ensembles S_i et S_i^T

La sélection des points de la grille grossière se base essentiellement sur la notion d'influence et de dépendance (vue à la section 2.3). Elle nécessite donc la construction des ensembles S_i et S_i^T pour chaque point i de la grille fine.

Rappelons que S_i est l'ensemble des points qui influencent fortement i et S_i^T l'ensemble des points qui dépendent fortement de i. La stratégie de construction de ces deux ensembles est décrite de la manière suivante :

Pour tous points $i \in \Omega^h$, faire :

1. Prendre l'élément maximal de la $i^{ème}$ ligne de A, c'est-à-dire,
$$max_i = \max_{k \neq i} \{|a_{ik}|\}.$$
2. Pour chaque point $j \neq i$ et tel que $a_{ij} \neq 0$, faire :
   2.1 Si $|a_{ij}| \geq \theta\, max_i$, poser :
   $$S_i = S_i \cup \{j\},$$
   $$S_j^T = S_j^T \cup \{i\}.$$

Commentaires

- Le code de cette sous-routine se trouve dans l'annexe à la page 115.
- Le paramètre θ est pris à 0.25 comme suggéré dans [14] ([4] ne propose aucune valeur).

Exemples

Nous allons illustrer cette construction à l'aide de deux exemples. Soit l'opérateur Laplacien suivant :

$$-\nabla^2 u(x,y) = -u_{xx} - u_{yy}. \tag{2.22}$$

Considérons d'abord la matrice de discrétisation de cet opérateur, obtenue par la méthode des différences finies suivante : soit l'approximation de Taylor du second ordre progressive par rapport à x :

$$u(x+h,y) \approx u(x,y) + u_x(x,y)\,h + \frac{u_{xx}(x,y)}{2}\,h^2, \tag{2.23}$$

et l'approximation régressive :

$$u(x-h,y) \approx u(x,y) - u_x(x,y)\,h + \frac{u_{xx}(x,y)}{2}\,h^2, \tag{2.24}$$

en additionnant (2.23) et (2.24), nous obtenons :

$$u(x-h,y) + u(x+h,y) \approx 2\,u(x,y) + u_{xx}(x,y)\,h^2,$$

c'est-à-dire :

$$-u_{xx}(x,y) \approx \frac{(-u(x+h,y) + 2\,u(x,y) - u(x-h,y))}{h^2}. \tag{2.25}$$

En procédant de la même manière avec la variable y, nous trouvons l'approximation de l'opérateur Laplacien (2.22) suivante :

$$\begin{aligned} -u_{xx}(x,y) - u_{yy}(x,y) \approx \frac{1}{h^2}(&- u(x+h,y) - u(x,y+h) + 4\,u(x,y) \\ &- u(x-h,y) - u(x,y-h)). \end{aligned} \tag{2.26}$$

En désignant les variables $u(x,y)$ par $u_{i,j}$, $u(x+h,y)$ par $u_{i+1,j}$, $u(x-h,y)$ par $u_{i-1,j}$, etc, l'approximation (2.26), pour une grille uniforme à neuf noeuds, prend la forme matricielle suivante :

$$A\,u = \frac{1}{h^2}\begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{pmatrix}. \tag{2.27}$$

et correspond au "stencil" à cinq points suivant :

$$\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}_h, \tag{2.28}$$

(une introduction à la notation "stencil" est disponible en annexe à la page 98).
En numérotant les variables $u_{i,j}$ de un à neuf, les ensembles S_i sont déterminés : $S_1 = \{2,4\}$, $S_2 = \{1,3,5\}$, $S_3 = \{2,6\}$, $S_4 = \{1,5,7\}$, $S_5 = \{2,4,6,8\}$, $S_6 = \{3,5,9\}$, $S_7 = \{4,8\}$, $S_8 = \{5,7,9\}$ et $S_9 = \{6,8\}$. Remarquons qu'ici tous les paramètres $\theta \in]0,1]$ donnent le même résultat. Notons que les S_i sont égaux aux S_i^T car la matrice est symétrique et les éléments hors-diagonaux non nuls sont identiques. Ce qui implique que si un point i influence fortement un autre point j, alors i dépend fortement du point j. Nous pouvons tracer un lien entre deux points du graphe de la matrice (voir figure 2.2) lorsque l'un des deux points influence l'autre ou si l'un des deux noeuds dépend fortement de l'autre : nous constatons que chaque point dépend fortement de ses points voisins et donc il les influence fortement.

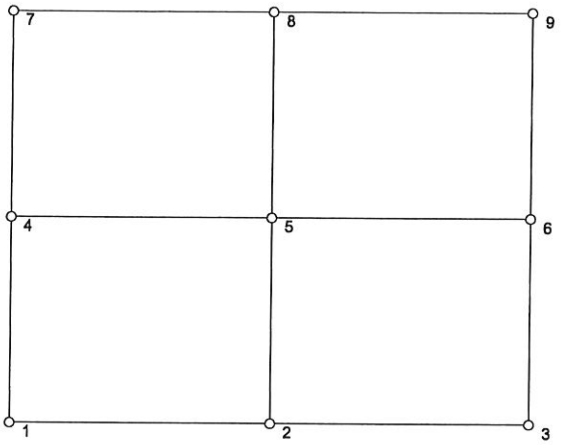

FIG. 2.2 – Représentation du graphe d'adjacence de la matrice A de (2.27) pour une grille à 9 noeuds.

Ensuite, considérons la matrice de discrétisation de l'opérateur Laplacien (2.22) associée au "stencil" à neuf points [4] suivant :

$$\frac{1}{h^2}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}_h . \tag{2.29}$$

Dessinons d'abord le graphe correspondant à la matrice de ce stencil (figure 2.3) pour une grille à 49 points. Ensuite, notons chaque noeud par $u_{i,j}$ où $1 \leq i \leq 7$ et $1 \leq j \leq 7$, et numérotons-les de bas en haut et de gauche à droite. Dès lors, le point inférieur gauche sera désigné par $u_{1,1}$ et le point supérieur droit par $u_{7,7}$. Sur la figure 2.4, nous pouvons voir de plus près un noeud intérieur, $u_{i,j}$ $(1 < i < 7,$ $1 < j < 7)$, de la grille, ainsi que ses huit voisins.

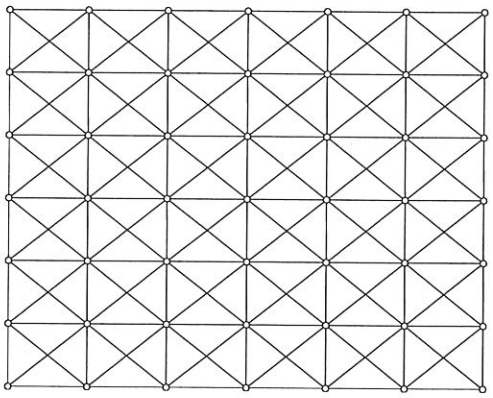

FIG. 2.3 – Représentation du graphe d'adjacence de la matrice associée au stencil (2.29) pour une grille à 49 noeuds.

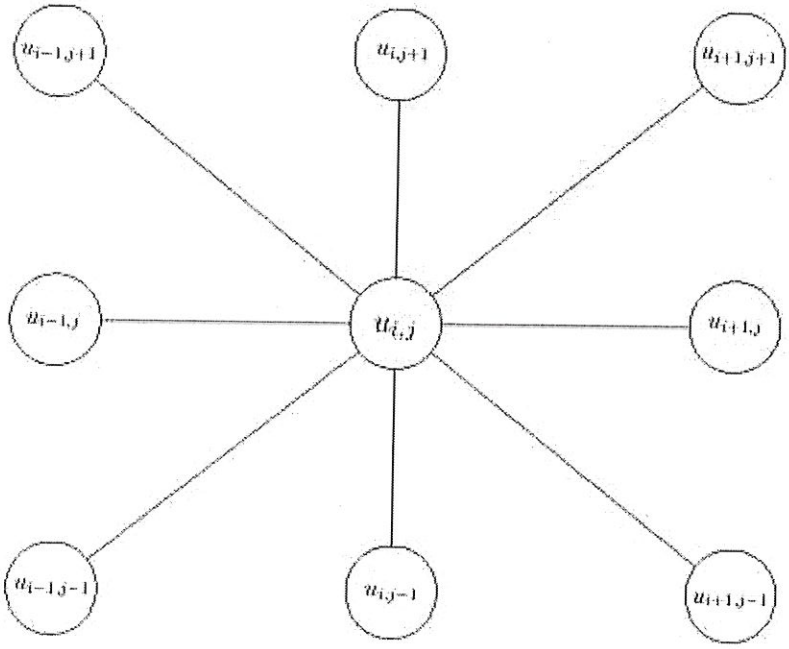

FIG. 2.4 – Zoom sur un noeud $u_{i,j}$ de la grille et ses huit voisins dont il dépend fortement et qu'il influence fortement.

Les différents choix du paramètre θ donnent toujours le même résultat et nous avons que tous les points dépendent fortement de leurs voisins. Dès lors, les ensembles $S_{u_{i,j}}$ correspondants aux points $u_{i,j}$ $(1 \leq i \leq 7, 1 \leq j \leq 7)$ prennent les formes suivantes :

- pour les points intérieurs, $u_{i,j}$ $(1 < i < 7, 1 < j < 7)$:

$$S_{u_{i,j}} = \{u_{i-1,j-1}, u_{i,j-1}, u_{i+1,j-1}, u_{i-1,j}, u_{i+1,j}, u_{i-1,j+1}, u_{i,j+1}, u_{i+1,j+1}\},$$

- pour les noeuds du dessous, u_{i1} $(1 < i < 7)$:

$$S_{u_{i,1}} = \{u_{i-1,1}, u_{i+1,1}, u_{i-1,2}, u_{i,2}, u_{i+1,2}\},$$

- pour les noeuds du dessus, $u_{i,7}$ $(1 < i < 7)$:

$$S_{u_{i,7}} = \{u_{i-1,6}, u_{i,6}, u_{i+1,6}, u_{i-1,7}, u_{i+1,7}\},$$

- pour les noeuds du côté gauche, $u_{1,j}$ $(1 < j < 7)$:

$$S_{u_{1,j}} = \{u_{1,j-1}, u_{2,j-1}, u_{2,j}, u_{1,j+1}, u_{2,j+1}\},$$

- pour les noeuds du côté droit, $u_{7,j}$ $(1 < j < 7)$:

$$S_{u_{7,j}} = \{u_{6,j-1}, u_{7,j-1}, u_{6,j}, u_{6,j+1}, u_{7,j+1}\},$$

- pour le point situé dans le coin inférieur gauche, $u_{1,1}$:

$$S_{u_{1,1}} = \{u_{2,1}, u_{1,2}, u_{2,2}\},$$

- pour le point situé dans le coin inférieur droit, $u_{7,1}$:

$$S_{u_{7,1}} = \{u_{6,1}, u_{6,2}, u_{7,2}\},$$

- pour le point situé dans le coin supérieur gauche, $u_{1,7}$:

$$S_{u_{1,7}} = \{u_{1,6}, u_{2,6}, u_{2,7}\},$$

- pour le point situé dans le coin supérieur droit, $u_{7,7}$:

$$S_{u_{7,7}} = \{u_{6,6}, u_{7,6}, u_{6,7}\}.$$

Pour ce problème, nous avons que les ensemble $S_{u_{i,j}}$ sont identiques aux $S^T_{u_{i,j}}$ car la matrice est symétrique et les coefficients hors-diagonaux non nuls sont tous égaux.

Maintenant que nous disposons de nos ensembles S_i et S_i^T, nous allons pouvoir commencer la sélection des points de la grille grossière à l'aide du premier schéma de coloriage. Ceci est l'objet de la section suivante.

2.9.3 Premier schéma de coloriage

Avant d'exposer l'algorithme du premier schéma de coloriage, nous allons détailler sa stratégie vue à la section 2.5.1. Ce schéma consistait à créer une première partition de la grille fine Ω^h en deux ensembles distincts F et C.

La première répartition des points de la grille fine en C-points et en F-points se déroule de la façon suivante :

1. Assigner à chaque point appartenant à la grille fine, $i \in \Omega^h$, un potentiel à appartenir à la grille grossière C. Ce potentiel est noté λ_i et correspond au nombre de points qui dépendent fortement de i :

$$\forall i \in \Omega^h \qquad \lambda_i = \sharp(S_i^T).$$

2. Répéter les instructions suivantes jusqu'à ce que tous les points de la grille fine soient répartis dans les ensembles F et C :
   2.1. Sélectionner le point i correspondant au λ_i le plus élevé, c'est-à-dire prendre :

$$i \quad \text{tel que} \quad \lambda_i = \max_{k \in \Omega^h \setminus \{F \cup C\}} \lambda_k.$$

   2.2. Placer i dans C, c'est-à-dire :

$$C = C \cup \{i\}.$$

   2.3. Mettre tous les points qui dépendent fortement de i, non classés, dans l'ensemble F, c'est-à-dire :

$$\forall j \begin{cases} \in S_i^T \\ \notin F \cup C \end{cases} \qquad F = F \cup \{j\}.$$

2.4. Pour tous les nouveaux F-points $j \in F$, incrémenter d'une unité les λ_k tel que $k \in S_j$ et n'est pas déjà classé, c'est-à-dire :

$$\forall j \in F \text{ nouveau}, \qquad \forall k \begin{cases} \in S_j \\ \notin F \cup C \end{cases} : \qquad \lambda_k = \lambda_k + 1.$$

Nous allons maintenant définir l'algorithme relatif au premier schéma de coloriage. Dans celui-ci, lorsqu'un point i de Ω^h sera classé, on posera $\lambda_i = -1$.

Premier schéma de coloriage

$$[C, F] \leftarrow SC1(\Omega^h)$$

1. $\forall i \in \Omega^h$ poser :
$$\lambda_i = \sharp(S_i^T).$$
2. Répéter les instructions 2.1 et 2.2 jusqu'à ce que $\sharp(C \cup F) = \sharp(\Omega^h)$:
   2.1. Prendre i tel que
   $$\lambda_i = \max_{k \in \Omega^h} \lambda_k,$$
   et faire :
   $$C = C \cup \{i\}$$
   $$\lambda_i = -1.$$
   2.2. $\forall j \in S_i^T$ tel que $\lambda_j \neq -1$ faire :
   $$F = F \cup \{j\}$$
   $$\lambda_j = -1$$
   $$\forall k \in S_j \text{ tel que } \lambda_k \neq -1, \text{ poser } \lambda_k = \lambda_k + 1.$$

Commentaire

- Le code de ce premier schéma de coloriage se trouve dans l'annexe à la page 116.

Exemples

Afin de mieux comprendre ce premier schéma de coloriage, nous allons reprendre l'exemple de la section précédente associé au "stencil" à neuf points (2.29). La figure 2.5 représente la numérotation choisie pour les noeuds du graphe de la matrice associée à (2.29), les points ij correspondant aux points $u_{i,j}$. La figure 2.6 illustre l'application du premier schéma de coloriage à notre exemple : à chaque étape, un nouveau point est ajouté à l'ensemble C et est identifié par une étoile noire.

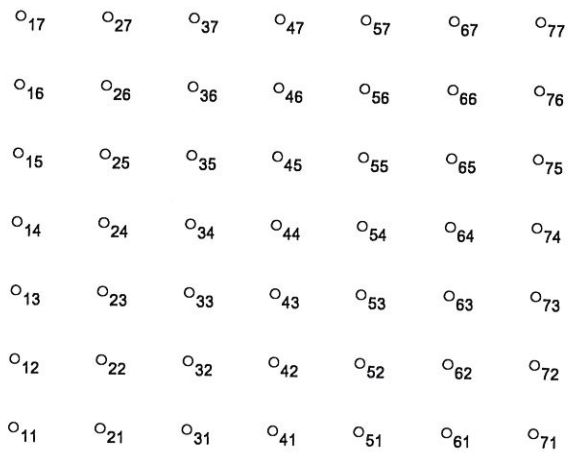

FIG. 2.5 – Représentation des noeuds numérotés du graphe de la matrice associée au stencil (2.29) pour une grille à 49 noeuds.

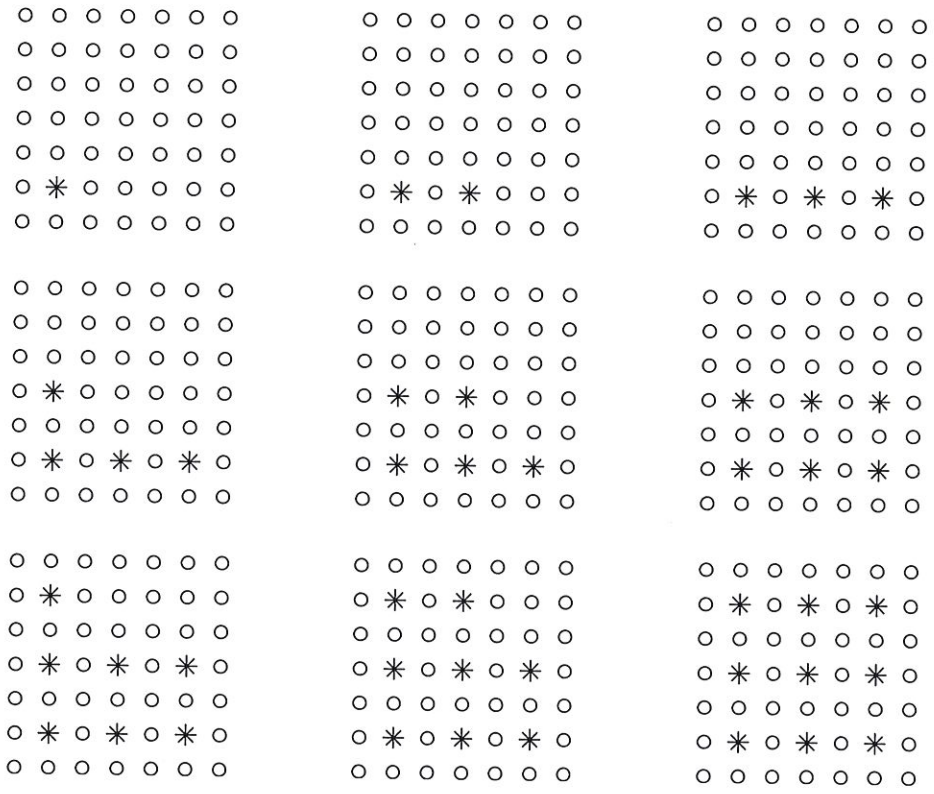

FIG. 2.6 – Représentation de l'application du premier schéma de coloriage à l'exemple associé au stencil (2.29) pour une grille de 49 noeuds. Les points de C sont désignés par des étoiles noires.

Notons que les ensembles $S_{u_{i,j}}$ calculés à la section précédente impliquent qu'à la première étape les $\lambda_{u_{i,j}}$ sont égaux à huit pour les noeuds intérieurs, $\lambda_{u_{i,j}} = 3$ pour

les points situés dans les quatre coins du graphe de la matrice et $\lambda_{u_{i,j}} = 5$ pour les autres points (sur les bords). Les points sont visités par l'algorithme de gauche à droite et de bas en haut. Dès lors, le premier point à entrer dans l'ensemble C est $u_{2,2}$. Les points appartenant à $S_{u_{2,2}}$ ($u_{1,1}$, $u_{2,1}$, $u_{3,1}$, $u_{1,2}$, $u_{3,2}$, $u_{1,3}$, $u_{2,3}$, $u_{3,3}$) sont placés dans F. Les $\lambda_{u_{i,j}}$ des noeuds qui influencent fortement les nouveaux F-points ($u_{4,1}$, $u_{4,2}$, $u_{4,3}$, $u_{4,4}$, $u_{1,4}$, $u_{2,4}$, $u_{3,4}$) sont incrémentés suivant le nombre de nouveaux F-points qu'ils influencent : $\lambda_{u_{4,1}} = 7$, $\lambda_{u_{4,2}} = 11$, $\lambda_{u_{4,3}} = 10$, $\lambda_{u_{1,4}} = 7$, $\lambda_{u_{2,4}} = 11$, $\lambda_{u_{3,4}} = 10$, $\lambda_{u_{4,4}} = 9$.

Ensuite, c'est le point $u_{4,2}$ qui devient le C-point suivant et les points appartenant à $S_{u_{4,2}}$ non classés ($u_{4,1}$, $u_{5,1}$, $u_{5,2}$, $u_{5,3}$, $u_{4,3}$), sont placés dans F. Puis les $\lambda_{u_{i,j}}$ des points qui influencent fortement ces points et qui ne sont pas encore classés ($u_{6,1}$, $u_{6,2}$, $u_{6,3}$, $u_{3,4}$, $u_{4,4}$, $u_{5,4}$, $u_{6,4}$), sont augmentés d'autant d'unités qu'ils influencent de nouveaux F-points : $\lambda_{u_{6,1}} = 7$, $\lambda_{u_{6,2}} = 11$, $\lambda_{u_{6,3}} = 10$, $\lambda_{u_{3,4}} = 11$, $\lambda_{u_{4,4}} = 11$, $\lambda_{u_{5,4}} = 10$, $\lambda_{u_{6,4}} = 9$.

En suivant l'ordre de passage, c'est le point $u_{6,2}$ qui devient le troisième point de C. Les points appartenant à $S_{u_{6,2}}$ non classés ($u_{6,1}$, $u_{7,1}$, $u_{7,2}$, $u_{6,3}$, $u_{7,3}$), deviennent les F-points suivants. Ensuite, les $\lambda_{u_{i,j}}$ des points qui influencent fortement ces points et qui ne sont pas encore classés sont augmentés d'autant d'unités qu'ils influencent de nouveaux F-points : $\lambda_{u_{5,4}} = 11$, $\lambda_{u_{6,4}} = 11$, $\lambda_{u_{7,4}} = 7$. A cette étape, plusieurs points ont un $\lambda_{u_{i,j}}$ égal à onze, le suivant dans l'ordre de passage est le point $u_{2,4}$. On procède ainsi de suite pour obtenir la dernière répartition représentée à la figure 2.6.

Remarquons que ce schéma de coloriage dépend fortement de l'ordre de passage des points. En effet, à la première étape, nous avons 25 points candidats (noeuds intérieurs) pour devenir le premier C-point. Notons aussi qu'à la fin du premier schéma de coloriage, tous les points de F vérifient la règle **H2** de la section 2.5.1.

Un second exemple se base sur le graphe d'une matrice de discrétisation de dimension 92×92, obtenue par une méthode d'éléments finis et fournie par Michal Kocvara. Le graphe d'adjacence de la matrice est représenté à la figure 2.7.

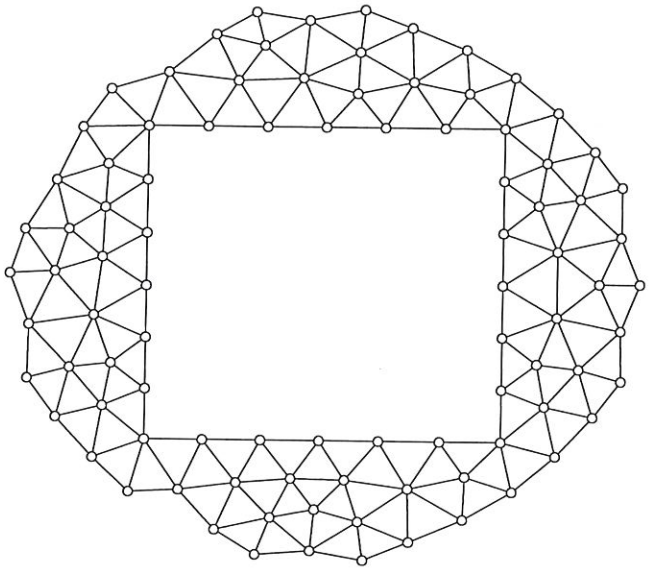

FIG. 2.7 – Graphe d'adjacence de la matrice à 92 noeuds.

La figure 2.8 représente la sélection du premier point de C obtenu à la première étape du premier schéma de coloriage.

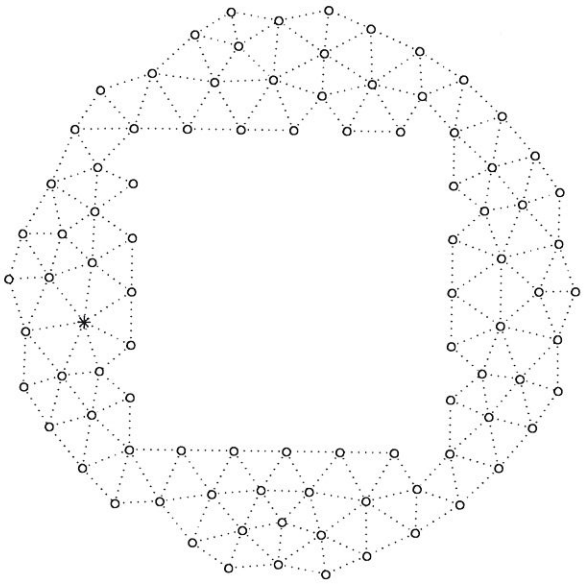

FIG. 2.8 – Première étape du premier schéma de coloriage, le premier C-point est représenté par l'étoile noire.

Ensuite, la figure 2.9 suivante représente les quatre premiers points de C obtenus par le schéma (les C-points sont représentés par des étoiles).

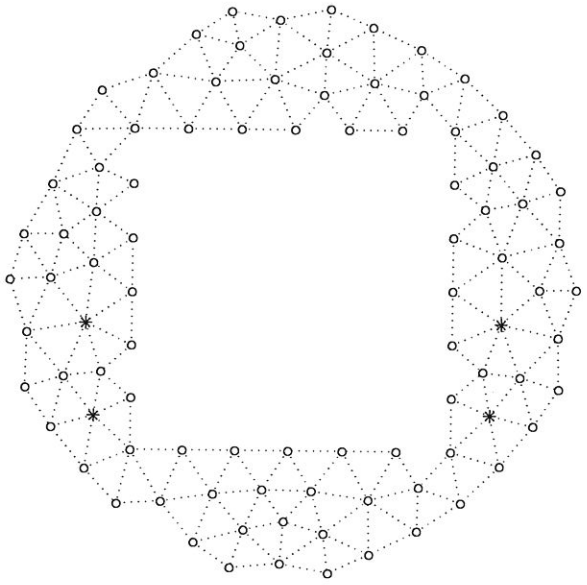

FIG. 2.9 – Répartition des C-points obtenus par quatre étapes du premier schéma de coloriage. Les quatre C-points sont désignés par des étoiles noires.

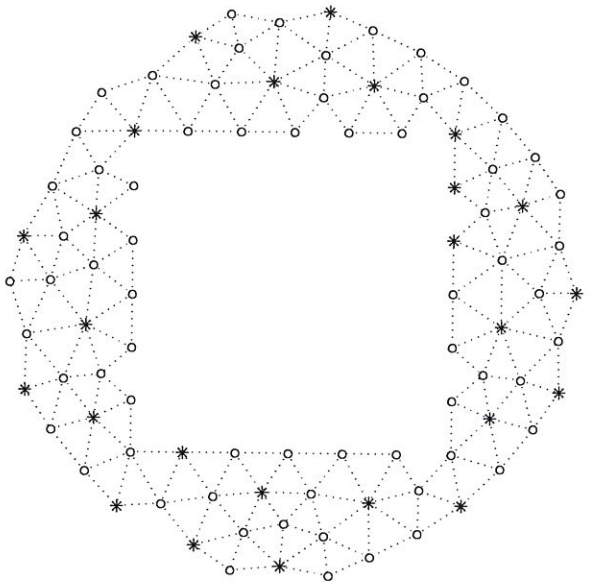

FIG. 2.10 – Représentation des deux ensembles F et C obtenus à la fin du premier schéma de coloriage, appliqué à la matrice associée au graphe de la figure 2.7. Les points de C sont identifiés par des étoiles et les points de F par des ronds blancs.

La dernière figure 2.10 représente la première partition de la grille fine Ω^h en les deux ensembles F et C obtenus par l'application du premier schéma de coloriage à la matrice associée au graphe de la figure 2.7. Nous verrons dans la section suivante que certains F-points de la dernière figure ne vérifient pas la règle **H2** de la section 2.5.1.

Dans la section suivante, nous allons décrire l'algorithme du deuxième schéma de coloriage pour que la condition **H2** de la section 2.5.1 puisse être vérifiée en chaque point de F.

2.9.4 Deuxième schéma de coloriage

Afin de construire l'algorithme du deuxième schéma de coloriage, nous allons d'abord détailler la stratégie de celui-ci. Rappelons que ce schéma a pour but de garantir l'existence de l'opérateur d'interpolation en tout point de la grille fine Ω^h. Cette condition d'existence est reprise dans la règle **H2**, cfr (2.20).

Observons tout d'abord que nous pouvons remplacer, dans la deuxième ligne de la condition (2.20), $j \in S_k^T$ par $k \in S_j$. En effet, dire que le point j dépend fortement du point k est équivalent à dire que k influence fortement j. Ainsi, nous n'aurons plus besoin des ensembles S_i^T que nous pouvons supprimer pour libérer de l'espace mémoire (étape 3 de la phase de préparation). La règle s'écrit alors :

H2 :

$$\forall i \in F \qquad \forall j \in S_i \qquad \begin{cases} \text{soit } j \in C_i, \\ \text{soit } \exists k \in C_i \;\; k \in S_j. \end{cases}$$

Maintenant, nous pouvons décrire la stratégie du deuxième schéma de coloriage présentée à la section 2.5.1 :

1. Pour tous les points i dans F, faire :
   - 1.1 Calculer C_i, l'ensemble des C-points qui influencent fortement i.
   - 1.2 Pour tous les points j qui influencent fortement i, c'est-à-dire $\forall j \in S_i$:
     - Si $j \in C_i$, passer au point suivant de S_i.
     - S'il existe $k \in C_i$ tel que $k \in S_j$, passer au point suivant de S_i.
     - Sinon le point j devient un candidat pour la grille grossière :
       - si j est le premier candidat trouvé dans S_i, alors placer j dans C et continuer la vérification des points,
       - sinon il existe déjà un point candidat $l \in D_i^s$ qui se trouve dans C, alors replacer l dans F et placer i dans C. Sortir de la boucle sur les $j \in S_i$ et passer au point suivant de F.

Nous pouvons à présent présenter l'algorithme relatif au deuxième schéma de coloriage. Dans celui-ci, nous avons associé une variable au candidat. Si cette variable est nulle cela signifie qu'il n'y pas encore de candidat, sinon elle contient le candidat lui-même.

Deuxième schéma de coloriage

$$[C, F] \leftarrow SC2(C, F)$$

1. Initialisation :
   - $candidat = 0$.

2. $\forall i \in F$ faire :
   - 2.1 Calculer C_i.
   - 2.2 $\forall j \in S_i$ faire :
     Si $j \notin C_i$ faire :
     Si $\nexists k \in C_i$ tel que $k \in S_j$ faire :
     Si $candidat = 0$ faire :
     $C = C \cup \{j\}$
     $candidat = j$
     $C_i = C_i \cup \{j\}$.
     Sinon faire :
     $C = C \backslash \{candidat\}$
     $C = C \cup \{i\}$
     $F = F \backslash \{i\}$
     $candidat = 0$
     break.
   - 2.3 Si $candidat \neq 0$
     $F = F \backslash \{candidat\}$,
     $candidat = 0$.

Commentaire

- Le code du deuxième schéma de coloriage se trouve dans l'annexe à la page 118.

Exemples

Reprenons l'exemple lié au graphique 2.7 de la section précédente, une représentation de l'application du deuxième schéma de coloriage à cet exemple est dessinée aux figures 2.11 et 2.12.

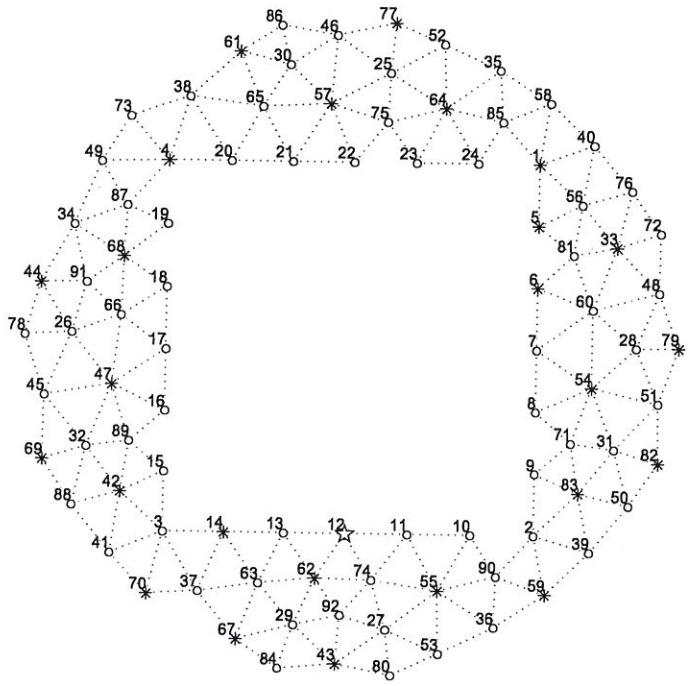

FIG. 2.11 – Représentation du premier nouveau point de C (numero 12) obtenu par l'application du deuxième schéma de coloriage. Le nouveau point de C est désigné par une étoile blanche tandis que les anciens C-points sont identifiés par des étoiles noires.

Nous remarquons sur la figure 2.11 que le premier C-point ajouté par l'algorithme est le point numéro 12. Celui-ci a été trouvé en examinant le premier point qui ne vérifie pas la règle **H2** : le point 11. En effet, considérons les données suivantes fournies par le premier schéma de coloriage et la matrice associée au graphique 2.7 :

- $S_{11} = \{10, 12, 55, 74\}$,
- $C_{11} = \{55\}$,
- $S_{10} = \{11, 55, 90\}$,
- $S_{12} = \{11, 13, 62, 74\}$,
- $S_{74} = \{11, 12, 27, 55, 62, 92\}$.

Le premier point trouvé de S_{11} qui n'est pas un point de C_{11} et qui ne dépend fortement d'aucun point de C_{11} est le point 12. Celui-ci devient alors un candidat pour la grille grossière et est placé dans C et donc aussi dans C_{11}. Comme le reste des points de C_{11} sont fortement influencés par au moins un point de S_{11}, le noeud 12 devient définitivement un C-point. La répartition finale obtenue à la fin de l'application du deuxième schéma est représentée à la figure 2.12. Il est à noter que, dans cet exemple, l'algorithme n'est jamais passé par le cas où il y a plus d'un candidat par S_i, $i \in F$. Pour ce cas, nous allons considérer un exemple similaire mais contenant 1532 noeuds fourni par Michal Kocvara, cet exemple est représenté à la figure 2.13.

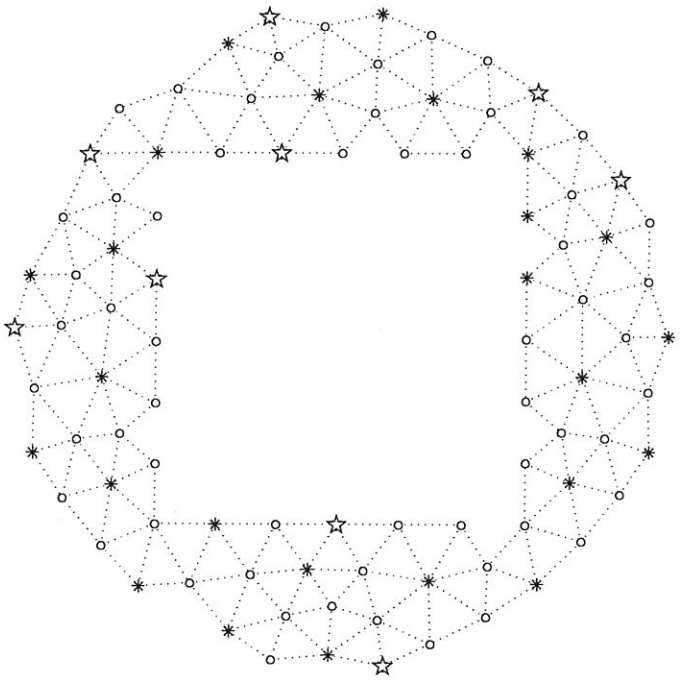

FIG. 2.12 – Répartition obtenue après l'application du deuxième schéma de coloriage. Les nouveaux points de C sont représentés par des étoiles blanches tandis que les anciens sont désignés par des étoiles noires.

Prenons le F-point 321 de la figure 2.13, un zoom sur ce point est représenté à la figure 2.14, et considérons les informations suivantes fournies par l'algorithme :

- $S_{321} = \{1242, 1243, 1244, 1245, 1246\}$,
- $C_{321} = \{1246\}$,
- $S_{1242} = \{60, 321, 772, 784, 1244\}$,
- $S_{1243} = \{81, 321, 1228, 1246, 1247\}$,
- $S_{1244} = \{217, 321, 772, 1230, 1242, 1246\}$,
- $S_{1245} = \{220, 321, 784, 1247\}$,
- $S_{1246} = \{318, 321, 1228, 1230, 1243, 1244\}$.

Le premier point de S_{321} qui ne dépend d'aucun point de C_{321} est le point 1242, ce point devient alors candidat pour la grille grossière et est placé dans C et dans C_{321}, la vérification des points continue. Un autre point qui ne dépend d'aucun point de C_{321} est rencontré : le point 1245, la vérification des points de S_{321} se termine, le point 1242 est retiré de l'ensemble C et c'est le point 321 qui devient un C-point. On passe alors au point suivant de F. On procède de manière analogue jusqu'à obtenir la distribution représentée à la figure 2.13.

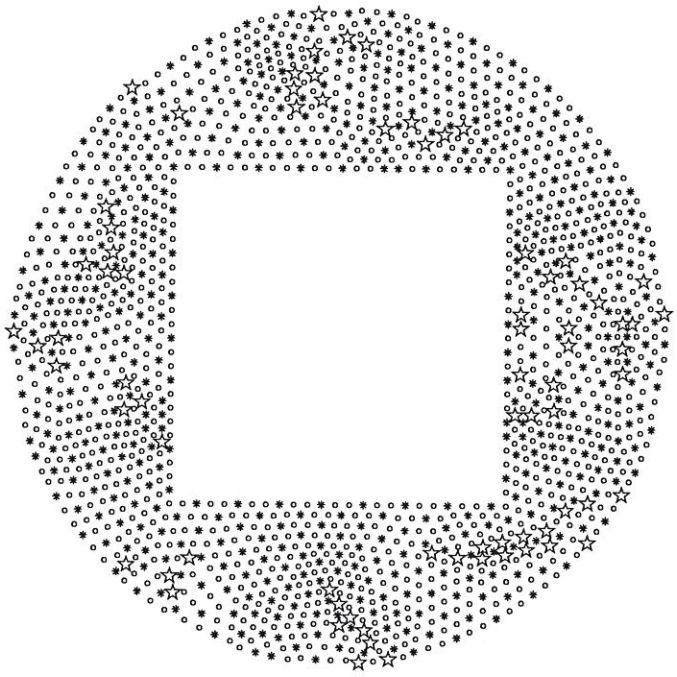

FIG. 2.13 – Répartition obtenue après l'application du deuxième schéma de coloriage au problème à 1532 noeuds.

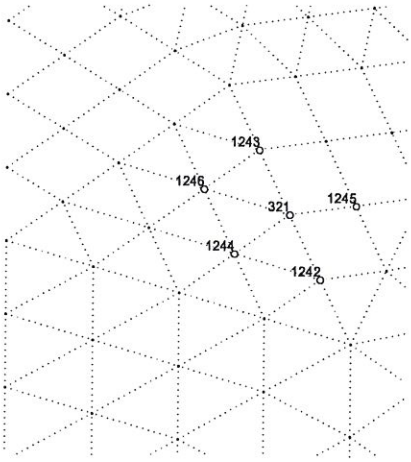

FIG. 2.14 – Zoom de la figure 2.13 sur le point 321 et ses voisins.

Nous possédons maintenant tous les outils nécessaires à la construction de notre opérateur d'interpolation. Nous allons donc établir l'algorithme de celui-ci dans la section suivante.

2.9.5 Opérateur d'interpolation et opérateur grossier

La construction de l'opérateur d'interpolation (2.7) se déroule de la manière suivante :

Pour tous points $i \in \Omega^h$, faire :

1. Initialiser (I_{2h}^h) à la matrice nulle.
2. Si $i \in F$, faire :
   - 1.1. Calculer les voisinages, C_i, D_i^s, D_i^w, de i.
   - 1.2. Pour tout $j \in C_i$, faire :
     - Construire w_{ij} à l'aide de sa définition (2.13).
     - Poser $(I_{2h}^h)_{ij} = w_{ij}$.
3. Sinon poser $(I_{2h}^h)_{ii} = 1$.

Le code de cet algorithme se trouve à la page 120. L'opérateur grossier A^{2h}, quant à lui, s'obtient simplement en appliquant la condition de Galerkin, cfr (1.13). Ces deux opérateurs, grossier et d'interpolation, sont stockés pour la deuxième phase, expliquée dans la section suivante.

2.9.6 Phase de résolution

La phase de résolution se déroule en appliquant l'un des schémas du multigrille en boucle, jusqu'à obtenir une erreur acceptable. Dans notre cas, nous utiliserons le schéma V-cycle décrit à la section 2.7. La phase de résolution, décrite de la manière suivante, dépend d'une approximation initiale donnée, v_0, et d'un niveau de tolérance souhaité, *tol*.

Poser $v^h = v_0$.

Répéter les instructions 1. et 2. tant que *erreur* $>$ *tol*

1. $v^h \leftarrow V^h(v^h, f^h)$.
2. Calculer l'*erreur*.

Nous entendons ici par erreur, l'erreur relative : $\frac{\|A^h v^h - f^h\|}{\|f^h\|}$.

Le code reprenant les deux phases et celui concernant les différents opérateurs se trouve en annexe à la page 111. Dans le chapitre suivant de ce mémoire, nous allons décrire un autre type de méthode des multigrilles algébriques, lié à la notion d'*agrégation*. Ensuite, nous présenterons plusieurs résultats numériques de la méthode de [4] implémentée en comparaison avec une méthode de *type agrégation* décrite dans le chapitre suivant.

Chapitre 3

Multigrilles Algébriques de type Agrégation

Dans ce nouveau chapitre, nous allons étudier les méthodes de multigrilles algébriques dites de *type agrégation*. Ces méthodes ont comme particularité que chaque point appartenant à la grille fine uniquement, c'est-à-dire de l'ensemble F, est interpolé à l'aide d'un seul point de la grille grossière, C. Pour réaliser cela, la grille de départ, Ω^h, est partitionnée en sous-ensembles disjoints, nommés *agrégats* et notés G_k^h. Chacun de ces ensembles contient des points de la grille fine, F, et un point de la grille grossière, C, servant à interpoler les autres points de son agrégat.

Nous allons tout d'abord présenter l'approche générale [14] de ces méthodes dans la section suivante, ensuite nous étudierons plus particulièrement la technique développée dans [20].

3.1 Agrégation : approche générale

Dans les méthodes d'agrégation, chaque point appartenant à l'ensemble F est interpolé à l'aide d'*un seul point* de la grille grossière C. Les ensembles d'interpolation C_i pour chaque F-point, i, sont donc réduits à un seul élément.

Pour obtenir cette condition, la grille fine Ω^h est divisée en *agrégats*, notés G_k^h où $k \in C$ est un noeud de la grille grossière et h est le niveau correspondant à la grille fine. Les autres points de G_k^h sont des F-points et seront interpolés par k. Une illustration d'une grille comportant 14 noeuds, divisée en trois agrégats, est représentée à la figure 3.1. Les points de F y sont identifiés par des points noirs, tandis que les trois C-points sont désignés par des ronds blancs contenant leurs indices respectifs.

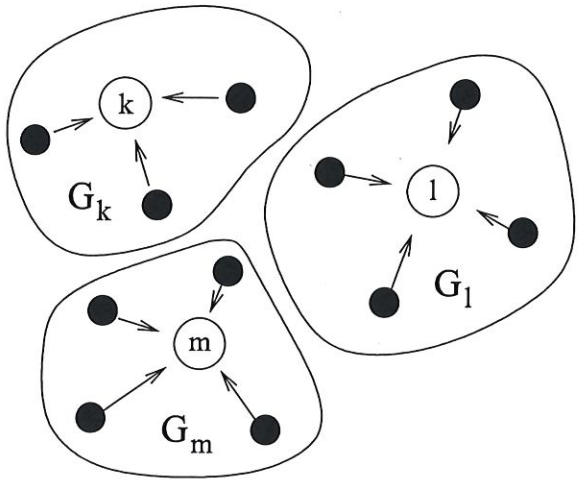

FIG. 3.1 – Représentation d'une grille à 14 noeuds, divisée en trois agrégats. Les points noirs sont des F-points et les ronds blancs sont des C-points.

Comme pour le chapitre précédent, nous devons définir un opérateur d'interpolation, un opérateur de restriction et un opérateur grossier. Chaque point de F est interpolé à l'aide du seul C-point appartenant au même agrégat. L'opérateur d'interpolation, pour passer de la grille grossière Ω^{2h} à la grille fine Ω^h, s'écrit de la manière suivante :

$$(I_{2h}^h)_{ij} = \begin{cases} 1 & \text{si } i \in G_j^h \\ 0 & \text{sinon.} \end{cases} \tag{3.1}$$

Cela signifie que lors de l'interpolation d'un vecteur e, les composantes e_i où $i \in F \cap G_j^h$ prennent la même valeur que l'élément e_j où j est un C-point. L'opérateur (3.1) représente une *interpolation constante par morceau.* D'où, pour obtenir une bonne interpolation, il est nécessaire que les points appartenant à un même agrégat dépendent fortement l'un de l'autre. Remarquons que, par construction, l'opérateur (3.1) interpole les constantes exactement. De plus, si nous reprenons la forme générale d'interpolation (2.7) vue dans le chapitre précédent, nous avons, dans ce cas-ci, qu'il n'existe qu'un seul w_{ij} pour chaque point $i \in F$ (car $\sharp(C_i) = 1$), et celui-ci vaut 1. Néanmoins, l'opérateur d'interpolation (3.1) constant par morceau est peu précis et converge lentement [14].

L'opérateur de restriction est construit de manière identique au chapitre précédent, en prenant le transposé de (3.1), c'est-à-dire :

$$(I_h^{2h})_{ij} = \begin{cases} 1 & \text{si } j \in G_i^h \\ 0 & \text{sinon.} \end{cases} \tag{3.2}$$

Il nous reste à définir l'opérateur grossier A^{2h} par la condition de Galerkin (1.13) :

$$A^{2h} = I_h^{2h} A^h I_{2h}^h. \tag{3.3}$$

En combinant avec les définitions (3.1) et (3.2), nous obtenons :

$$\begin{aligned}
(I_h^{2h}\, A^h\, I_{2h}^h)_{kl} &= \sum_{j=1}^{n} (I_h^{2h}\, A^h)_{kj}\, (I_{2h}^h)_{jl} \\
&= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} \underbrace{(I_h^{2h})_{ki}}_{\substack{=0 \text{ si } i \notin G_k^h \\ =1 \text{ sinon}}} (A^h)_{ij} \right) (I_{2h}^h)_{jl} \\
&= \sum_{j=1}^{n} \left(\sum_{i \in G_k^h} (A^h)_{ij} \right) \underbrace{(I_{2h}^h)_{jl}}_{\substack{=0 \text{ si } j \notin G_l^h \\ =1 \text{ sinon}}} \\
&= \sum_{j \in G_l^h} \sum_{i \in G_k^h} (A^h)_{ij}.
\end{aligned}$$

Nous remarquons que l'opérateur grossier A^{2h} dépend fortement de la définition des agrégats.

Remarquons qu'ici le fait de savoir si c'est un point qui influence un autre ou si c'est l'autre qui influence le premier n'a pas d'importance. Dès lors, nous ne définirons pas de notion d'influence et de dépendance, mais uniquement de connexion forte. Cela nous mène à une définition plus symétrique, c'est-à-dire : un point i est fortement connecté à un autre point j si $\frac{a_{ij}^2}{a_{ii}a_{jj}}$ ne dépasse pas une certaine grandeur.

En pratique, la méthode d'agrégation expliquée ci-dessus n'est pas utilisée telle quelle. En effet, elle est peu efficace dû notamment au mauvais opérateur d'interpolation. C'est pourquoi d'autres méthodes ont été élaborées pour corriger cette interpolation. Nous allons nous intéresser tout particulièrement à celle présentée dans [20]. Mais avant cela, nous allons introduire brièvement les méthodes des éléments finis.

3.2 Éléments finis

La méthode des multigrilles algébriques de [20] s'applique de préférence à des systèmes linéaires issus de la discrétisation de problèmes continus via l'une des méthodes des éléments finis. De plus, les niveaux grossiers engendrés par l'algorithme développé dans [20] vérifient un nombre d'hypothèses propres aux structures des méthodes des éléments finis. C'est pourquoi nous allons introduire les concepts de base de ces méthodes. Nous nous limiterons à un type de problème particulier et à une des méthodes des éléments finis, présentée dans [1] et [16]. Nous allons d'abord poser notre cadre de travail ensuite nous détaillerons la méthode choisie.

3.2.1 Le problème

Nous allons donc nous restreindre aux *problèmes avec conditions frontières BVP*. Soit Ω un domaine de $\mathbb{R}^d$, une application $k : \Omega \rightarrow \mathbb{R}^{d\times d}$ et une fonction $f : \Omega \rightarrow \mathbb{R}$, considérons le problème BVP qui consiste à trouver la fonction $u : \Omega \rightarrow \mathbb{R}$ telle que :

$$-\nabla(k(x)\,\nabla u) = f(x) \qquad \text{où } x \in \Omega, \tag{3.4}$$

admettant des conditions de type Neumann[1] sur Γ_N ou de type Dirichlet[2] sur Γ_D. Nous noterons $\partial\Omega = \Gamma_N \cup \Gamma_D$ la frontière du domaine Ω qui comprend l'union des conditions de Neumann et de Dirichlet[3].

La formulation (3.4) du problème est appelée la *forme forte*. Par la suite, nous allons aussi présenter la *forme faible* du problème. Pour cela nous devons définir un espace V de fonction test :

$$V = \{v \in H_1(\Omega) \mid v \text{ vérifie les conditions frontières}\},$$

où $H_1(\Omega)$ est l'*espace de Sobolev* qui permet au problème d'être posé correctement [7]. A partir de cet espace, nous pouvons formuler la *forme faible* du problème (3.4) : trouver $u \in V$ tel que

$$\int_\Omega -\nabla(k(x)\,\nabla u)\,v = \int_\Omega f\,v \qquad \forall v \in V,$$

qui est équivalente [7] à trouver $u \in V$ tel que

$$\int_\Omega k(x)\,\nabla u\,\nabla v = \int_\Omega f\,v \qquad \forall v \in V. \tag{3.5}$$

Il est possible de montrer que la forme faible est équivalente à la forme forte du BVP [7]. De manière équivalente, la forme faible (3.5) peut s'écrire sous la forme suivante : trouver $u \in V$ tel que

$$a(u,v) = \langle f\,,\,v\rangle \qquad \forall v \in V, \tag{3.6}$$

où par définition du produit scalaire sur $L^2(\Omega)$:

$$\langle f\,,\,g\rangle = \int_\Omega f\,g,$$

et $a(u,v)$ est la forme bilinéaire symétrique définie par :

$$a(u,v) = \int_\Omega k(x)\,\nabla u\,\nabla v. \tag{3.7}$$

Dans la section suivante, nous allons présenter la *méthode de Galerkin*. Il en existe plusieurs mais nous nous limiterons à la méthode générale.

---

[1] Condition de Neumann : seule la dérivée normale, $\frac{\partial u}{\partial n}$ (n est la normale à la frontière), est connue sur la frontière du domaine Ω. La partie de la frontière correspondant aux conditions de Neumann est notée Γ_N.

[2] Condition de Dirichlet : seule la valeur de la fonction, $u(x)$, est connue sur la frontière du domaine Ω. La partie de la frontière correspondant aux conditions de Dirichlet est notée Γ_D.

[3] Notons que lorque la frontière admet à la fois des conditions de Neumann et de Dirichlet, ces conditions sont en général appelées conditions de Cauchy.

3.2.2 La méthode de Galerkin

L'idée général des méthodes des éléments finis est de remplacer la solution exacte $u \in V$ du problème continu (3.6) par l'approximation $u^h \in V_n$, où V_n est un *sous-espace d'approximation de dimension finie* n ($h = \frac{1}{n}$ est le pas de discrétisation) de l'espace V.

Si l'on remplace V par V_n dans (3.5), nous pouvons approximer la forme faible par : trouver $u^h \in V_n$ tel que

$$a(u^h, v^h) = \langle f\,,\, v^h \rangle \qquad \forall v^h \in V_n. \tag{3.8}$$

Comme le sous-espace V_n est de dimension finie, il admet une base de la forme suivante : $\{\varphi_1, \ldots, \varphi_n\}$. Le problème (3.8) est alors équivalent à : trouver $u^h \in V_n$ tel que

$$a(u^h, \varphi_i) = \langle f\,,\, \varphi_i \rangle \qquad \forall i = 1, \ldots, n.$$

De plus, si on décompose u^h dans la base des $\{\varphi_i\}_{i=1}^n$, nous obtenons la forme équivalente : trouver $u^h \in V_n$ tel que

$$\sum_{j=1}^{n} a(\varphi_j, \varphi_i)\, u_j^h = \langle f\,,\, \varphi_i \rangle \qquad \forall i = 1, \ldots, n, \tag{3.9}$$

où $u^h = \sum_{j=1}^n u_j^h\, \varphi_j$.

Notre problème (3.9) peut donc s'écrire sous la forme du système linéaire suivant :

$$A^h\, u^h = f^h, \tag{3.10}$$

où les coefficient de A^h sont $A_{ij}^h = a(\varphi_j, \varphi_i)$ et les composantes de f^h correspondent à $f_i^h = \langle f\,,\, \varphi_i \rangle$.

3.2.3 Discrétisation du domaine et éléments

Dans une méthode d'éléments finis, le domaine Ω du problème est divisé en petites régions fermées qui portent le nom d'*éléments* et qui sont notées $\{e_l\}_{l=1}^L$, où L est le nombre total d'éléments. En général, ces éléments ont une *forme polygonale.* Par exemple la figure 2.7 représente une triangulation d'un domaine représentant un disque coupé en son milieu par un carré.

Le domaine Ω est aussi composé d'un ensemble de noeuds, $\{n_i\}_{i=1}^n$, chaque noeud correspond à un, deux ou plusieurs éléments (soit c'est un noeud intérieur alors il n'appartient qu'à un seul élément, soit il se trouve sur une arête alors il appartient à deux éléments, soit il est sur un sommet alors il appartient à deux éléments ou plus).

Il existe plusieurs façons de choisir les noeuds et les éléments. En général pour une bonne approximation de la solution, ce choix devra être [1] tel que :

- les grands ou petits angles seront à éviter dans la définition des éléments,
- il y aura plus d'éléments dans les endroits où l'on s'attend à ce que la solution varie fortement, et
- pour une plus grande précision, il y aura davantage de noeuds.

Pour la suite, nous considérerons des éléments de forme polygonale dans un domaine à deux dimensions et nous supposerons que tous les éléments d'un maillage sont de forme identique. Dans la section suivante, nous allons déterminer la forme des fonctions φ_i de la base du sous-espace de V_n introduites dans la section précédente.

3.2.4 Fonctions de forme des éléments finis

Soit un maillage d'éléments finis construit comme dans la section précédente, on associe à chaque noeud n_i, $i = 1, \ldots, n$, une fonction φ_i qui correspond aux φ_i de la base du sous-espace de V_n, elles portent le nom de *fonctions de forme*. Ces fonctions de forme sont construites en respectant les propriétés suivantes :

- elles satisfont :
$$\varphi_i(n_j) = \delta_{ij} \qquad \forall i, j = 1, \ldots, n, \tag{3.11}$$
où δ_{ij} est le symbole Kronecker :
$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon;} \end{cases}$$
- elles prennent la forme polynomiale suivante sur un élément e_l :
$$\varphi_i(x, y) = \sum_{s=1}^{T} c_{il_s} x^{p_s} y^{q_s} \qquad \forall (x, y) \in e_l, \tag{3.12}$$
où c_{il_s} sont des constantes (l désigne l'élément sur lequel la fonction est définie, i est le noeud auquel φ_i est associée et s est l'indice de sommation), p_s et q_s sont les puissances du pôlynome du $s^{ième}$ terme de la somme, et T est le nombre de noeuds contenu dans e_l. La forme du polynôme de (3.12) dépend du nombre de noeuds que contient les éléments. Par exemple, si nous avons des éléments triangulaires à trois noeuds comme à la figure 2.7, nous aurons trois degrés de liberté et donc des polynômes de la forme suivante :
$$\varphi_i(x, y) = c_{il_1} + c_{il_2} x + c_{il_3} y \qquad \forall (x, y) \in e_l.$$
Le nombre de degrés de liberté est déterminé par le nombre de noeuds des éléments ;
- les constantes c_{il_s} de l'équation (3.12) sont déterminées, à l'aide de (3.11), par le système suivant :
$$\varphi_i(x_{i_{r_l}}, y_{i_{r_l}}) = \sum_{s=1}^{T} c_{il_s} x_{i_{r_l}}^{p_s} y_{i_{r_l}}^{q_s} = \begin{cases} 1 & \text{si } i_{r_l} = i \\ 0 & \text{sinon} \end{cases} \qquad r = 1, \ldots, n,$$

où les indices $\{i_{r_l}\}_{r=1}^{T}$ désignent les r noeuds contenus dans l'élément e_l et dont i fait partie, les $(x_{i_{r_l}}, y_{i_{r_l}})$ sont les coordonnées du noeud $n_{i_{r_l}}$;
- les fonctions φ_i sont non nulles uniquement sur les éléments contenant le noeud n_i.

Ensuite, nous définissons le *support* d'une fonctions de forme, φ_i, comme étant l'union des éléments auxquels appartient le noeud n_i. Ce qui veut dire que les fonctions φ_i sont nulles partout, sauf sur leurs supports.

Une fonction de forme, φ_i, peut être vue [1] géométriquement comme une "tente" centrée au noeud n_i et dont le "sol" est délimité par son support.

Maintenant que nous disposons de nos fonctions de formes φ_i, nous allons pouvoir préciser la forme des coefficients de la matrice A^h et les composantes du vecteur f^h du système (3.10).

En considérant que le domaine Ω est égal à l'union des éléments, les coefficients de A^h sont donnés par, cfr (3.7) :

$$\begin{aligned} A^h_{ij} &= a(\varphi_j, \varphi_i) \\ &= \int_\Omega k(x)\, \nabla\varphi_i\, \nabla\varphi_j \\ &= \sum_{l=1}^{L} \int_{e_l} k(x) \nabla\varphi_i\, \nabla\varphi_j. \end{aligned}$$

De manière identique, nous pouvons déterminer les éléments du vecteur f^h par $f^h_i = \sum_{l=1}^{L} \int_{e_l} f^h \varphi_i$. Il est possible de montrer [16] que si l'intersection des supports de φ_i et φ_j est vide, égale à un sommet ou une arête, alors la composante A^h_{ij} est nulle.

Maintenant que nous en savons plus sur les éléments finis, nous allons aborder la méthode des multigrilles algébriques de [20] dans la section suivante.

3.3 "Smooth Aggregation"

L'algorithme des multigrilles algébriques développé dans [20] porte le nom de *"Smooth Aggregation"*. C'est une méthode de type agrégation qui utilise une itération de relaxation de Jacobi pour corriger l'interpolation.

La méthode de la "Smooth Aggregation" consiste à d'abord répartir les noeuds de la grille fine en des ensembles disjoints ; ensuite elle définit une tentative d'interpolation sur ces agrégats qui n'est autre que l'opérateur d'interpolation constant par morceau (3.1) ; ensuite, une itération de relaxation de type Jacobi est appliquée à l'opérateur provisoire pour obtenir l'opérateur d'interpolation définitif ; finalement, les opérateurs restants (opérateur de restriction et opérateur grossier) sont construits

à partir des propriétés variationnelles ((1.12) et (1.13)).

Nous pouvons appliquer cette méthode à n'importe quel système linéaire dont la matrice est symétrique définie positive. Néanmoins, la "Smooth Aggregation" est mieux adaptée [20] aux systèmes issus de la discrétisation de problèmes elliptiques du second ordre via une méthode des éléments finis. En effet, cette technique a pour avantage de garder une certaine structure liée aux éléments finis dans les niveaux grossiers. C'est ce que nous allons présenter dans la section suivante.

3.3.1 Lien avec les éléments finis

Considérons le problème elliptique du second ordre sous la forme variationnelle suivante : trouver $u \in V$ tel que :

$$a(u,v) = f(v) \qquad \forall v \in V, \tag{3.13}$$

où $V = H(\Omega)$ est un espace de Sobolev [18] dont les fonctions sont nulles sur $\Gamma_D \subset \partial\Omega$, $\mu(\Gamma_D) < c\mu(\partial\Omega)$ où μ désigne la mesure et c est une constante positive. La forme bilinéaire (3.13) est supposée symétrique, V-elliptique et bornée :

$$c_1 \|u\|^2_{H(\Omega)} \leq a(u,u) \leq c_2 \|u\|^2_{H(\Omega)} \qquad \forall u \in V,$$

où c_1 et c_2 sont des constantes positives.

La construction des niveaux grossiers de la méthodes de [20] est basée sur les structures des éléments finis. Dès lors, nous allons associer un ensemble de fonctions de base $\{\varphi_i^k\}_{i=1}^{n_k}$ à chaque niveau k (où n_k est le nombre de noeuds au niveau k). Le sous-espace d'approximation, noté $V_{n_k}^h$, pour chaque niveau k est donné par $V_{n_k}^h = \text{span}\{\varphi_i^k\}_{i=1}^{n_k}$. Les fonctions de base, $\{\varphi_i^1\}_{i=1}^{n_1}$, du niveau le plus fin correspondent aux fonctions de forme des éléments finis. Dès lors, les fonctions de base des niveaux grossiers sont obtenus par la formule :

$$\begin{bmatrix} \varphi_1^{k+1} \\ \varphi_2^{k+1} \\ \vdots \\ \varphi_{n_{k+1}}^{k+1} \end{bmatrix} = I_k^T \begin{bmatrix} \varphi_1^k \\ \varphi_2^k \\ \vdots \\ \varphi_{n_k}^k \end{bmatrix} \qquad k = 1, \ldots, l-1,$$

où I_k est l'opérateur d'interpolation de la méthode de [20] qui permet de passer de la grille du niveau $k+1$ à celle du niveau k, et l est le nombre de grilles données. On suppose aussi que les fonctions de base de $V_{n_1}^h$ satisfont la propriété suivante :

$$\sum_{i=1}^{n_1} \varphi_i^1 = c,$$

sur tout les éléments hors de la frontière, où c est une constante.

Exigences souhaitées pour la construction des opérateurs d'interpolation

Les niveaux grossiers sont construits avec l'intention d'atteindre certains objectifs [20]. Ceux-ci sont exprimés en termes des fonctions de base. Ces conditions ont pour but de garantir une bonne convergence théorique (démontrée dans [19]) et de borner la complexité de l'algorithme. Ces exigences sont :

($AMG1$) Le support des fonction de base, pour un niveau k, sera composé de l'union d'un petit nombre de supports de fonctions de base appartenant au niveau $k-1$.

($AMG2$) Deux noeuds contenus dans le support d'une fonction de base d'un niveau grossier doivent être connectés via un chemin de connexion forte. Rappelons que pour les méthodes d'agrégation, deux noeuds i et j sont fortement connectés si $\frac{a_{ij}^2}{a_{ii}a_{jj}}$ ne dépasse pas une certaine grandeur.
Cette condition et ($AMG1$) assurent le "semi-coarsing" pour les problèmes anisotropiques (c'est-à-dire qui dépendent des directions). L'anisotropie se reflète dans la matrice : les points sont fortement connectés dans les directions d'anisotropies. Dès lors, les supports des fonctions de base des espaces grossiers s'allongent dans ces directions. Nous détaillerons davantage ce type de problème dans le chapitre suivant.

($AMG3$) Le support de chaque fonction de base d'un niveau grossier s'intersecte avec un nombre fini de supports d'autres fonctions de base de ce niveau. Le nombre maximal d'intersections est indépendant du niveau. Cette propriété a pour effet d'obtenir des matrices creuses pour les niveaux grossiers.

($AMG4$) Les fonctions constantes pourront être représentées exactement (à l'exception des composantes correspondant aux conditions frontières) sur tous les espaces grossiers $V_{n_k}^h$. Cette condition a pour but d'assurer une borne sur l'erreur des approximations : $I^k v^{k+1}$, c'est-à-dire d'obtenir une bonne interpolation. Cette propriété revient à [20] :

$$\sum_{j=1}^{n_{k+1}} I_{ij}^k = 1 \qquad k = 1, \ldots, l-1$$

où i varie dans $\{1, \ldots, n_k\}$, en retirant les indices correspondant aux conditions frontières.

($AMG5$) L'énergie des fonctions de base des niveaux grossiers doit être minimale dans le sens :

$$\frac{a(\varphi_i^k, \varphi_i^k)}{\left\|\varphi_i^k\right\|_{L^2(\Omega)}^2} \leq c \inf_{u \in H(supp(\varphi_i^k))} \frac{a(u,u)}{\|u\|_{L^2(\Omega)}^2}$$

où $supp(\varphi_i^k)$ est le support de la fonction φ_i^k et c est une constante positive. Cette condition ajoutée à la propriété ($AMG3$) assure l'inégalité standard inverse suivante, pour le cas des problèmes V-elliptiques :

$$\left\|\varphi_i^k\right\|_{H(\Omega)}^2 \leq c\, \mu(supp(\varphi_i^k))^{-1} \left\|\varphi_i^k\right\|_{L^2(\Omega)}^2,$$

où c est une constante positive.

(AMG6) La norme discrète de l^2 est équivalente à la norme continue de L^2 , pour tous les niveaux grossiers k, dans le sens suivant :

$$\|v\|^2_{l^2(\Omega)} \approx \sum_{i=1}^{n_k} \alpha_i^2 \, \|\varphi_i^k\|^2_{L^2(\Omega)} \qquad \forall\, v = \sum_{i=1}^{n_k} \alpha_i \, \varphi_i^k.$$

Ces dernières conditions signifient que l'algorithme crée des espaces grossiers semblables à une hiérarchie d'espaces d'éléments finis. Dans la section suivante, nous allons présenter l'algorithme de la "Smooth Aggregation" décrit dans [20]. Malgré que celui-ci a été conçu pour des problèmes issus des méthodes des éléments finis, cet algorithme est appliquable à n'importe quel système $Au = f$, dont la matrice A est symétrique, définie positive.

3.3.2 Algorithme de la "Smooth Aggregation"

Comme pour les méthodes AMG classiques, les multigrilles algébriques de type agrégation procèdent en deux phases : la phase de préparation et la phase de résolution. Rappelons que pour la phase de préparation, nous devons construire les opérateurs d'interpolation et les opérateurs grossiers de chaque niveau. Pour cela, nous allons d'abord présenter l'algorithme de la méthode AMG : "Smooth Aggregation", développée dans [20].

Avant de nous lancer dans l'algorithme "Smooth Aggregation", nous allons commencer par établir la définition de voisinage d'un noeud quelconque i qui sera utilisée dans la méthode de [20].

Définition 8. *Soit $\epsilon \in [0, 1[$ fixé, le voisinage fortement connecté du point i est défini, au niveau m, par :*

$$N_i^m(\epsilon) = \{j \in \Omega^{kh} \;\mid\; |a_{ij}| \geq \epsilon \sqrt{a_{ii} a_{jj}}\},$$

où $k = 2^{m-1}$.

Comme la phase de préparation s'exécute en boucle et par niveau, dans la suite de cette section, nous n'utiliserons que deux grilles : une fine Ω^h et une grossière Ω^{2h}, et donc deux niveaux : h et $2h$.

La stratégie de la méthode de [20] au niveau h est la suivante : on commence par construire des voisinages pour chaque noeud, i, de la grille fine sur base de la Définition 8 ; ensuite on forme les agrégats en trois étapes :

1. on séléctionne des voisinages disjoints comme premiers agrégats,
2. puis on ajoute les points non classés aux agrégats auxquels ils sont fortement connectés,

3. finalement, s'il reste encore des points, on forme des agrégats supplémentaires avec ces points (normalement cette étape n'a pas lieu d'être puisqu'à la fin du point précédent il ne devrait plus y avoir de noeud).

Quand la répartition des points est terminée, on procède à la construction de l'opérateur d'interpolation provisoire, défini par (3.2). Finalement, l'opérateur d'interpolation définitif est obtenu par une itération de Jacobi pondéré que nous détaillerons par la suite.

Nous allons commencer par générer un recouvrement disjoint : $\{G_i^h\}_{i\in\Omega^{2h}}$ de Ω^h, ensuite nous allons construire l'opérateur d'interpolation à partir de ces agrégats. Nous allons détailler ces deux étapes dans les deux sections qui suivent.

Agrégation

Soit $\epsilon \in [0,1[$ fixé, la génération des agrégats, pour le niveau h, se déroule de la manière suivante :

1. Construction des voisinages de $N_i^h(\epsilon)$ pour chaque point i de la grille fine :

$$\forall i \in \Omega^h \qquad N_i^h(\epsilon) = \{j \in \Omega^h \;\mid\; |a_{ij}| \geq \epsilon\sqrt{a_{ii}a_{jj}}\}. \tag{3.14}$$

2. Poser $R = \{i \in \Omega^h \mid N_i^h(0) \neq \{i\}\}$ (les points isolés ne sont pas agrégés).
3. Initialiser $j = 0$.
4. Formation des agrégats G_k^h :

   4.1. **Partition de départ :**
   Sélection de voisinages disjoints comme premier essai de recouvrement :
   Pour tout les noeuds $i \in R$ faire :
   si $N_i^h(\epsilon) \subset R$, alors poser :
   - $j \leftarrow j+1$,
   - $G_j^h \leftarrow N_i^h(\epsilon)$,
   - $R \leftarrow R \backslash G_j^h$.

   Il est à noter que ces ensembles ne recouvrent pas nécessairement entièrement R.

   4.2. **Ajustement des agrégats initiaux :**
   Ajout des points non classés, $i \in R$, aux agrégats auxquels ils sont fortement connectés. Pour cela on procède de la manière suivante :
   - Copier tous les agrégats : $\tilde{G}_k^h \leftarrow G_k^h \qquad k = 1,\ldots,j.$
   - $\forall i \in R$ faire :
     si $\exists\, k$ tel que $N_i^h(\epsilon) \cap \tilde{G}_k^h \neq \emptyset$, alors poser :
     - $G_k^h \leftarrow \tilde{G}_k^h \cup \{i\}$
     - $R \leftarrow R \backslash \{i\}$

S'il existe plusieurs agrégats auxquels le point i est fortement connecté, alors on prendra celui auquel il est le plus fortement connecté, c'est-à-dire pour lequel $\sharp(N_i^h(\epsilon) \cap \tilde{G}_k^h)$ est le plus grand.

4.3. **Rassemblement des points restants :**
Formation d'agrégats supplémentaires avec les points restants s'il en reste :
$\forall i \in R$ poser :

- $j \leftarrow j+1$,
- $G_j^h \leftarrow R \cap N_i^h(\epsilon)$,
- $R \leftarrow R \backslash G_j^h$.

Remarquons qu'après l'étape 4.2, l'ensemble R est normalement vide et le point 4.3 n'est donc jamais exécuté. Néanmoins, pour des questions de performance et de rapidité, par expérience (cfr Michal Kocvara), on préfère sauter l'étape 4.2 en faveur du point 4.3.

A présent que nous disposons de nos agrégats, nous allons pouvoir construire l'opérateur d'interpolation dans la section suivante.

Interpolation

A partir des agrégats, G_k^h, obtenus dans la section précédente, une première tentative d'interpolation est définie sur base de l'équation (3.1). Cet opérateur, constant par morceau, est noté $\tilde{I}_{2h}^h$.

Ensuite, une itération de type Jacobi pondéré est appliquée à l'opérateur d'interpolation provisoire : $\tilde{I}_{2h}^h$. Nous obtenons alors l'opérateur d'interpolation final de la manière suivante :

$$I_{2h}^h = (I - w\, D^{-1}\, A_F^h)\, \tilde{I}_{2h}^h, \tag{3.15}$$

où A_F^h est la *matrice filtrée* donnée par :

$$\text{si } i \neq j : \qquad (A_F^h)_{ij} = \begin{cases} (A^h)_{ij} & \text{si } j \in N_i^h(\epsilon) \\ 0 & \text{sinon} \end{cases} \tag{3.16}$$

et

$$(A_F^h)_{ii} = (A^h)_{ii} - \sum_{\substack{j \in \Omega^h \\ j \neq i}} ((A^h)_{ij} - (A_F^h)_{ij}), \tag{3.17}$$

et D est la diagonale de A^h.

Le remplacement de la matrice A^h par la matrice filtrée A_F^h dans l'équation (3.15) a pour but de diminuer le nombre d'éléments non nuls de l'opérateur d'interpolation I_{2h}^h. Cela a de l'effet essentiellement dans le cas des problèmes anisotropiques (c'est-à-dire dépendant des directions).

Basés sur des expériences numériques [20], les paramètres ϵ et w sont donnés par :

$$\epsilon = 0.08 \left(\frac{1}{2}\right)^{l-1}, \qquad w = \frac{2}{3}, \tag{3.18}$$

où $l > 1$ est le nombre de niveaux donnés.

Après avoir obtenu l'opérateur d'interpolation, l'opérateur grossier A^{2h} est calculé à l'aide de la formule de Galerkin (1.13). Les opérateurs de chaque niveau sont ainsi construits puis stockés pour la seconde étape de résolution.

3.3.3 Algorithme du Multigrille

Après avoir stocké les différents opérateurs obtenus dans la section précédente, on peut appliquer la phase de résolution. Cette phase consiste à appliquer récursivement le schéma V-cycle décrit à la section 2.7, jusqu'au niveau de tolérance désiré. Dans ce cas-ci, les paramètres de l'algorithme du schéma V-cycle sont $\nu_1 = \nu_2 = 1$ et le schéma de relaxation utilisé est le schéma SOR (Succesive OverRelaxation method), appelé schéma de surrelaxation.

Méthode SOR

Reprenons la décomposition de la matrice A suivante : $A = D - L - U$, où D est la diagonale de A, $-L$ sa partie triangulaire strictement inférieure et $-U$ sa partie triangulaire strictement supérieure. La méthode de relaxation SOR consiste à prendre comme approximation suivante :

$$v^{(m+1)} = R_w\, v^{(m)} + w\,(D - wL)^{-1} f\,, \tag{3.19}$$

où $R_w = (D - wL)^{-1}[(1-w)D + wU]$ est la matrice d'itération de SOR. Remarquons que si $w = 1$, nous obtenons le schéma de relaxation de Gauss-Seidel. Si la matrice A est symétrique définie positive, la méthode de surrelaxation SOR converge vers la solution du système (2.1) pour n'importe quel w pris dans $(0, 2)$ et pour n'importe quel point de départ.

Pour mieux comprendre la méthode de la "Smooth Aggregation", nous allons illustrer celle-ci à l'aide d'exemples dans la section suivante.

3.4 Exemples

Dans cette section, nous allons commencer par illustrer la construction de l'opérateur d'interpolation en détaillant un simple exemple. Ensuite, nous utiliserons un second exemple pour montrer la méthode de sélection des points de la grille grossière

de la "Smooth aggregation".

Commençons par considérer l'opérateur de discrétisation de $-u_{xx}$ sur une grille à une seule dimension, munie d'un pas de discrétisation h. Par un procédé identique à celui utilisé pour obtenir (2.25), nous obtenons l'approximation de $-u_{xx}$ suivante :

$$-u_{xx}(x) \approx \frac{(-u(x+h) + 2\,u(x) - u(x-h))}{h^2},$$

ou, sous une forme équivalente, la discrétisation de $-u_{xx}$ est représentée par le "stencil" :

$$\frac{1}{h^2}\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}. \tag{3.20}$$

Considérons maintenant une grille à neuf noeuds avec un pas de discrétisation $h = 1$, la matrice associée au "stencil" (3.20) prend la forme suivante :

$$A^h = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}. \tag{3.21}$$

Commençons par construire les agrégats, pour cela, calculons les voisinage de chaque point (cfr (3.14)) :

$$\begin{array}{lll} N_1^h(0.04) = \{1,2\} & N_4^h(0.04) = \{3,4,5\} & N_7^h(0.04) = \{6,7,8\} \\ N_2^h(0.04) = \{1,2,3\} & N_5^h(0.04) = \{4,5,6\} & N_8^h(0.04) = \{7,8,9\} \\ N_3^h(0.04) = \{2,3,4\} & N_6^h(0.04) = \{5,6,7\} & N_9^h(0.04) = \{8,9\}. \end{array}$$

Choisissons les agrégats suivants :

$$\begin{aligned} G_2^h &= \{1,2,3\}, \\ G_5^h &= \{4,5,6\}, \\ G_8^h &= \{7,8,9\}, \end{aligned}$$

où $\{2,5,8\}$ forme l'ensemble des noeuds de la grille grossière. Remarquons que dans l'algorithme de [20], la formation d'agrégats dépend de l'ordre de passage des noeuds, dans ce cas-ci nous choisissons volontairement les agrégats G_2^h, G_5^h et G_8^h pour une meilleure illustration de l'opérateur d'interpolation (nous pouvons supposer que le point 2 est le premier à passer dans l'algorithme). A partir de ces agrégats, nous pouvons déterminer la première tentative d'interpolation, en utilisant l'équation (3.1).

L'opérateur d'interpolation constant par morceau obtenu, est le suivant :

$$\tilde{I}_{2h}^{h} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.22}$$

Définissons par $\tilde{e}^h$ le vecteur obtenu par $\tilde{e}^h = \tilde{I}_{2h}^{h} e^{2h}$ dont les composantes sont :

$$\begin{array}{lll} \tilde{e}_1^h = e_1^{2h} & \tilde{e}_4^h = e_2^{2h} & \tilde{e}_7^h = e_3^{2h} \\ \tilde{e}_2^h = e_1^{2h} & \tilde{e}_5^h = e_2^{2h} & \tilde{e}_8^h = e_3^{2h} \\ \tilde{e}_3^h = e_1^{2h} & \tilde{e}_6^h = e_2^{2h} & \tilde{e}_9^h = e_3^{2h}. \end{array} \tag{3.23}$$

Remarquons ensuite que la matrice filtrée est identique à la matrice d'origine, c'est-à-dire que $A^h = A_F^h$. En effet, ceci est dû au fait que tous les noeuds correspondants aux éléments non nuls d'une ligne i appartiennent au voisinage de i, $N_i^h(0.04)$. D'où, par (3.16), nous avons que tous les éléments hors-diagonaux sont égaux pour les deux matrices A_F^h et A^h, ce qui implique que les composantes diagonales de ces matrices sont également identiques par (3.17). Construisons maintenant l'opérateur d'interpolation donné par (3.15), celui-ci prend la forme :

$$\begin{aligned} I_{2h}^{h} &= (I - w\, D^{-1} A^h)\, \tilde{I}_{2h}^{h} \\ &= \tilde{I}_{2h}^{h} - w \begin{pmatrix} 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \tilde{I}_{2h}^{h} - w \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 0 & 0 \\ 1/2 & -1/2 & 0 \\ -1/2 & 1/2 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & -1/2 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 0 \\ 0 & 0 & 1/2 \end{pmatrix}. \end{aligned}$$

Appliquons maintenant notre opérateur d'interpolation I_{2h}^{h} à un vecteur e^{2h} de la

grille grossière, c'est-à-dire $e^h = I_{2h}^h e^{2h}$ dont les composantes sont :

$$\begin{aligned}
e_1^h &= \tilde{e}_1^h - w(\frac{1}{2}e_1^{2h}) \\
e_2^h &= \tilde{e}_2^h - 0 \\
e_3^h &= \tilde{e}_3^h - w(-\frac{1}{2}e_2^{2h} + \frac{1}{2}e_1^{2h}) \\
e_4^h &= \tilde{e}_4^h - w(-\frac{1}{2}e_1^{2h} + \frac{1}{2}e_2^{2h}) \\
e_5^h &= \tilde{e}_5^h \\
e_6^h &= \tilde{e}_6^h - w(-\frac{1}{2}e_3^{2h} + \frac{1}{2}e_2^{2h}) \\
e_7^h &= \tilde{e}_7^h - w(-\frac{1}{2}e_2^{2h} + \frac{1}{2}e_3^{2h}) \\
e_8^h &= \tilde{e}_8^h \\
e_9^h &= \tilde{e}_9^h - w(\frac{1}{2}e_3^{2h}),
\end{aligned}$$

par (3.23), nous obtenons

$$\begin{array}{lll}
e_1^h = (1-\frac{w}{2})\,e_1^{2h} & e_4^h = \frac{w}{2}\,e_1^{2h} + (1-\frac{w}{2})\,e_2^{2h} & e_7^h = \frac{w}{2}\,e_2^{2h} + (1-\frac{w}{2})\,e_3^{2h} \\
e_2^h = e_2^{2h} & e_5^h = e_2^{2h} & e_8^h = e_3^{2h} \\
e_3^h = \frac{w}{2}\,e_2^{2h} + (1-\frac{w}{2})\,e_1^{2h} & e_6^h = \frac{w}{2}\,e_3^{2h} + (1-\frac{w}{2})\,e_2^{2h} & e_2^h = (1-\frac{w}{2})\,e_3^{2h}
\end{array}$$

et en prenant $w = 2/3$

$$\begin{array}{lll}
e_1^h = \frac{2}{3}\,e_1^{2h} & e_4^h = \frac{1}{3}\,e_1^{2h} + \frac{2}{3}\,e_2^{2h} & e_7^h = \frac{1}{3}\,e_2^{2h} + \frac{2}{3}\,e_3^{2h} \\
e_2^h = e_2^{2h} & e_5^h = e_2^{2h} & e_8^h = e_3^{2h} \\
e_3^h = \frac{1}{3}\,e_2^{2h} + \frac{2}{3}\,e_1^{2h} & e_6^h = \frac{1}{3}\,e_3^{2h} + \frac{2}{3}\,e_2^{2h} & e_2^h = \frac{2}{3}\,e_3^{2h}.
\end{array}$$

Nous obtenons donc une interpolation linéaire.

Considérons maintenant une grille de longueur $[0, 8]$ comportant neuf noeuds avec un pas de discrétisation $h = 1$. A la figure 3.2, nous avons représenté les deux interpolations appliquées à un vecteur $e^{2h} = (\sin(\pi/16)\ \sin(\pi/4)\ \sin(7\pi/16))^T$ qui est la restriction de la fonction $\sin(t\pi/16)$ définie sur $[0, 8]$ et dessinée en pointillés sur le graphe. Nous pouvons constater que l'interpolation obtenue par (3.1), représentée par des tirets, est constante par morceau tandis que l'interpolation donnée par (3.15), tracée par un trait plus épais, est linéaire.

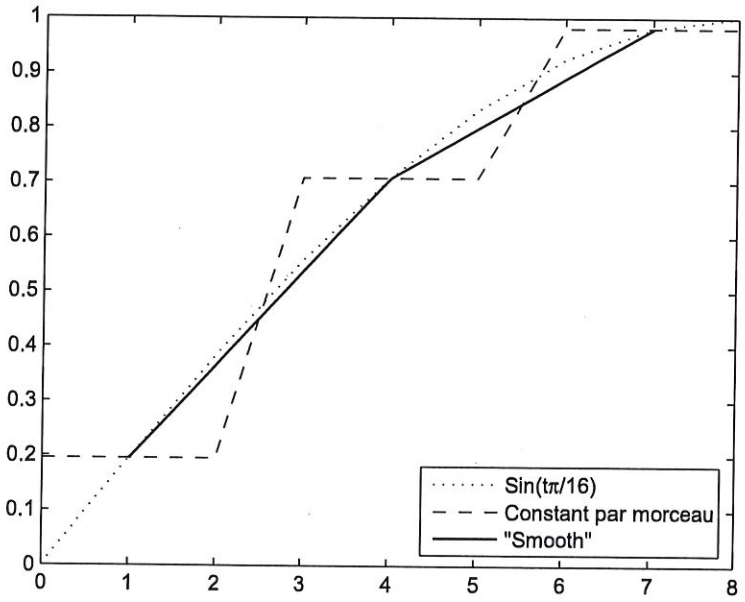

FIG. 3.2 – Représentation d'une interpolation constante par morceau et de l'interpolation obtenue par une itération de relaxation.

Illustrons à présent la méthode de sélection des points de la grille grossière expliquée à la section 3.3.2 et implémentée par Michal Kocvara (cfr annexe page 103). Reprenons l'exemple associé au graphe d'adjacence 2.7, notons que les arrêtes du graphe correspondent aux connexions fortes entre les noeuds. La figure 3.3 représente la sélection des agrégats obtenus par la méthode de la "Smooth Aggregation". Les premiers agrégats, G_k^h, disjoints obtenus à la fin de l'étape 4.1 de l'algorithme y sont représentés par des lignes rouges et les C-points k de chaque agrégat sont désignés par des étoiles rouges. Les agrégats obtenus à la fin de l'étape 4.3 de l'algorithme (le point 4.2 n'est pas considéré) sont représentés par des lignes bleues et leurs C-points sont identifiés par des étoiles bleues.

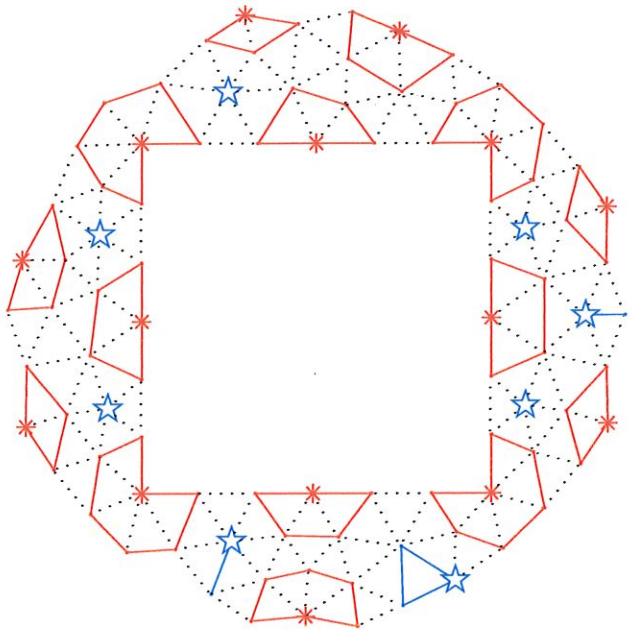

FIG. 3.3 – Représentation des agrégats obtenus par l'algorithme de [20].

Pour une simulation de la sélection des points, nous allons nous servir de la figure 2.11 illutstrant la numérotations des noeuds. Nous remarquons que le voisinage du noeud 1 (coin supérieur droit du carré blanc) est $N_1^h(0,04) = \{5, 56, 40, 58, 85, 24\}$, il est choisi comme premier agrégat G_1^h et tous les points qui en font partie sont retirés de l'ensemble R. Ensuite, le noeud 2 (coin inférieur droit du carré blanc) est examiné. Comme son voisinage, N_2^h, est disjoint avec le premier agrégat, N_2^h devient le second agrégat G_2^h. Le cas des noeuds 3 et 4 est similaire, et on obtient donc les agrégats : G_1^h, G_2^h, G_3^h et G_4^h. Ensuite, rappelons que le point 5 n'appartient plus à l'ensemble R puisqu'il fait déjà partie de G_1^h. En suivant l'ordre de passage, c'est le point 6 qui passe dans l'algorithme. Remarquons que l'intersection de son voisinage, $N_6^h = \{5, 6, 7, 60, 81\}$, avec le premier agrégat est non-vide. L'algorithme passe alors au point suivant qui est le noeud 7. Le voisinage de 7 est disjoint des agrégats G_1^h, G_2^h, G_3^h et G_4^h, il devient alors l'agrégat suivant. L'algorithme procède ainsi de suite jusqu'à obtenir les agrégats disjoints représentés à la figure 3.3 par des lignes rouges. Ensuite, l'algorithme passe à l'étape 4.3 et forme des agrégats avec les points restants représentes par des lignes bleues sur la figure 3.3 .

Dans le chapitre suivant, nous allons comparer les deux méthodes de multigrilles algébriques, étudiées dans ce chapitre et le précédent.

Chapitre 4

Résultats numériques

Dans ce chapitre, nous allons présenter les résultats numériques obtenus par le programme de Michal Kocvara, basé sur l'algorithme de [20], et celui implémenté dans le cardre de ce mémoire, basé sur la méthode décrite dans [4]. Nous commencerons par présenter différents problèmes particuliers auxquels nous appliquerons les deux méthodes de selection de points de la grille grossière. Ensuite, nous étudireons les résultats numériques obtenus en appliquant les programmes à une série de problèmes dérivants de ceux introduits dans la première partie de ce chapitre.

4.1 Problèmes particuliers

Dans cette setion, nous nous intéresserons plus particulièrement aux différentes manières de sélectionner les points de la grille grossière des deux méthodes. Pour cela, nous allons considérer plusieurs types de problèmes. Nous allons commencer par les problèmes dits "anisotropiques".

4.1.1 Problèmes anisotropiques

Les problèmes anisotropiques sont des problèmes dont la solution dépend fortement des directions, c'est-à-dire qu'elle évolue dans une direction plutôt qu'une autre.

Pour mieux situer ce type de problème, considérons l'équation suivante :

$$-u_{xx} - \epsilon u_{yy} = f(x,y). \tag{4.1}$$

Prenons la discrétisation de l'équation (4.1) sur une grille carrée avec un pas de discrétisation h identique dans les deux directions, x et y, et supposons que $0 < \epsilon << 1$. Cette discrétisation est représentée par le "stencil" à cinq points [4] suivant :

$$A^h = \frac{1}{h^2} \begin{bmatrix} & -\epsilon & \\ -1 & 2+2\epsilon & -1 \\ & -\epsilon & \end{bmatrix}_h . \tag{4.2}$$

Notons qu'il est possible de montrer [4] que la discrétisation de (4.1) avec $\epsilon = 1$ sur une grille rectangulaire de pas de discrétisation $h_x = h$ dans la direction des x et $h_y = \frac{h}{\sqrt{\epsilon}}$ dans la direction des y, est donnée par le même "stencil" que (4.2). L'anisotropie peut donc être présente dans l'équation du problème de départ ou dans la définition de la grille. Néanmoins, l'anisotropie se reflète toujours dans la matrice de discrétisation qui est celle qui nous intérresse dans le cadre des multigrilles algébriques.

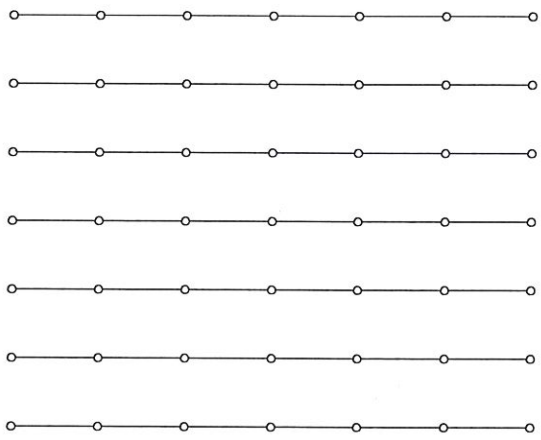

FIG. 4.1 – Représentation du graphe de la matrice associée au "stencil" (4.2).

Considérons maintenant une grille à 49 noeuds et dessinons le graphe d'adjacence de la matrice (4.2) sur la figure 4.1, où nous ne représentons que les connexions fortes et prenons $\epsilon = 1e^{-9}$. La figure 4.2 illustre l'application des deux algorithmes au problème anisotropique considéré.

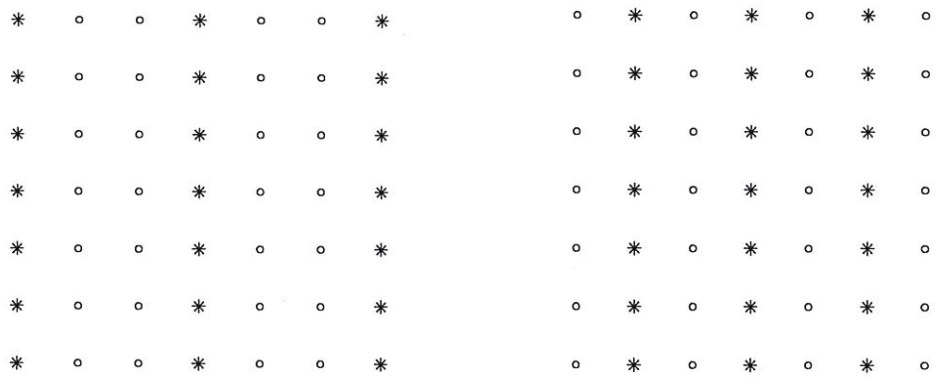

FIG. 4.2 – Représentation de la sélection des points de la grille grossière obtenue par les méthodes de multigrilles algébriques (à gauche celle de type agrégation et à droite celle de type classique).

Sur les deux dessins de la figure 4.2, nous pouvons remarquer que la sélection des points de la grille grossière ne se fait que dans la direction des x (direction des connexions fortes). Dans le sens vertical tous les points restent dans la grille, cette particularité porte le nom de "semi-coarsing". Les deux méthodes donnent ce "semi-coarsing" pour les problèmes anisotropiques. Pour la méthode classique de [4], cela s'explique par le fait que les erreurs lisses varient peu en direction des connexions fortes (2.6). Dès lors, l'algorithme favorise la sélection dans ces directions. Dans la méthode de type agrégation, nous avions vu que l'une des conditions de l'agorithme (cfr section 3.3.1) était de garantir le "semi-coarsing" pour les problèmes de type anisotropique.

4.1.2 Problèmes isotropiques

A l'opposé des problèmes anisotropiques, nous avons les problèmes isotropiques, c'est-à-dire indépendants des directions. Un exemple de ce type de problème est celui que nous avions traité dans le deuxième chapitre, associé au "stencil" (2.29). La méthode de type classique appliquée à cet exemple donnait la répartition représentée à la figure 2.6. L'application de l'algorithme de la méthode de type agrégation à ce problème est représentée à la figure 4.3. Nous remarquons sur ces graphiques que la sélection se fait dans toutes les directions, cette particularité porte le nom de "full-coarsing".

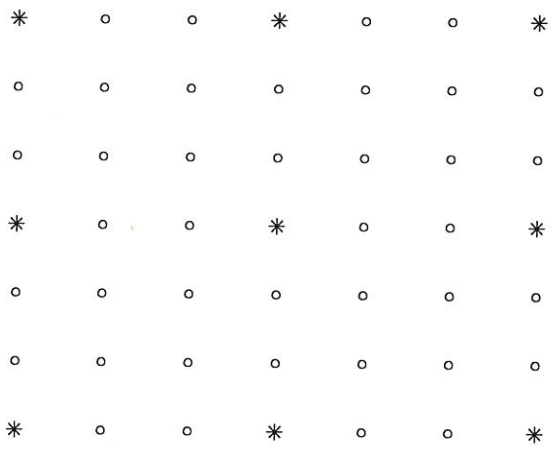

FIG. 4.3 – Application de l'algorithme de type agrégation au problème associé au "stencil" (2.29).

4.1.3 Problème mixte

Nous allons maintenant présenter un problème dû à [14] mais dont les détails sont repris de [4]. Cet exemple découle de l'équation suivante :

$$-a\,u_{xx} - b\,u_{yy} + c\,u_{xy} = f(x,y), \tag{4.3}$$

où a, b, c sont des constantes positives et f est une fonction à valeur dans $\mathbb{R}$. Nous considérons la discrétisation de cette équation sur une grille carrée unitaire, déterminée par la somme des "stencils" suivants :

$$-D_{xx}^h = \frac{1}{h^2}\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}_h,$$

$$-D_{yy}^h = \frac{1}{h^2}\begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}_h,$$

et du "stencil" concernant la dérivée mixte :

$$D_{xy}^h = \frac{1}{2h^2}\begin{bmatrix} -1 & 1 & \\ 1 & -2 & 1 \\ & 1 & -1 \end{bmatrix}_h.$$

Divisons ensuite la grille en quatre sous-grilles de taille identique, nous accordons à chaque quadrant des valeurs différentes pour les constantes a, b et c de l'équation (4.3), ces valeurs étant données par le tableau suivant :

$a = 1$ $b = 1000$ $c = 0$	$a = 1$ $b = 1$ $c = 2$
$a = 1$ $b = 1$ $c = 0$	$a = 1000$ $b = 1$ $c = 0$

(4.4)

Sur la figure 4.4 nous avons représenté le graphe de la matrice associée au problème considéré, nous y avons représenté les connexions fortes. Il est à noter que la matrice obtenue pour ce problème n'est pas une M-matrice.

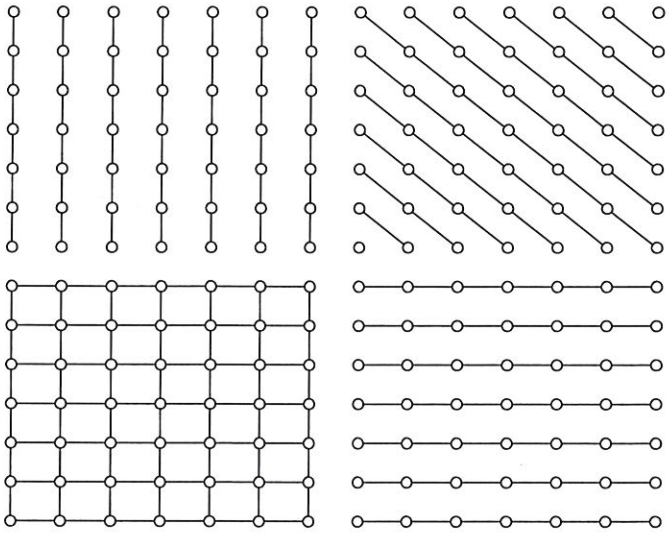

FIG. 4.4 – Représentation du graphe d'adjacence associé à la matrice de discrétisation de (4.3) de paramètres (4.4).

Les figures 4.6 et 4.5 illustrent l'application des deux méthodes de multigrilles algébriques étudiées au problème considéré. Nous pouvons constater que, dans les quadrants situés en haut à gauche et en bas à droite, nous retrouvons le "semi-coarsing" des problèmes anisotropiques introduits en début de ce chapitre. Ce phénomène est dû à la définition du problème (4.4). En effet, dans ces régions de la grille, nous pouvons remarquer que le graphe de la matrice (cfr figure 4.4) est semblable à celui du problème anisotropique (cfr figure 4.1). Dans le coin inférieur droit des figures, nous trouvons une distribution identique à celle d'un problème isotropique. En effet, les paramètres du problème (4.4) impliquent que le "stencil" associé à cette partie du graphe correspond à celui du problème Laplacien (2.28). Dans le dernier coin supérieur droit des graphiques, les points sont sélectionnés dans la direction des diagonales du graphe 4.4 de cette région. Nous pouvons noter que les sélections de points pour les deux méthodes sont semblables dans la structure, néanmoins l'algorithme de type agrégation a tendance à garder moins de points que celui de type classique.

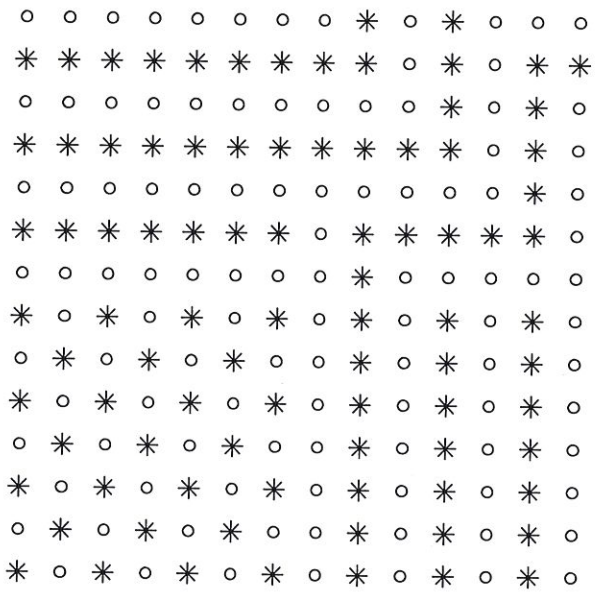

FIG. 4.5 – Représentation de la répartition des points obtenue après l'application de la méthode de type classique au problème (4.3).

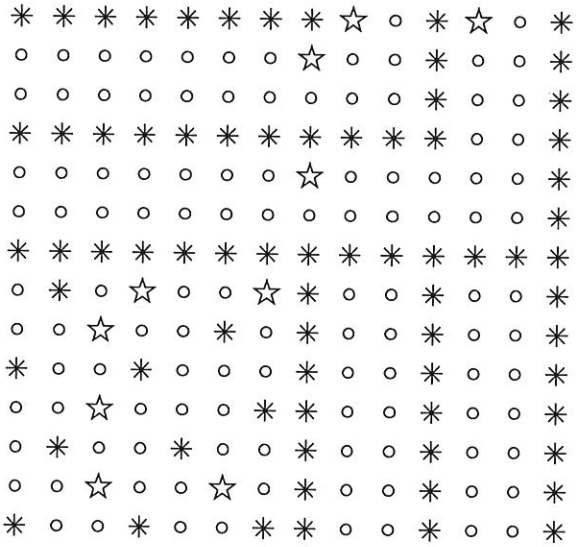

FIG. 4.6 – Représentation de la sélection des points obtenue par l'algorithme implémenté par Michal Kocvara pour le problème associé à l'équation (4.3).

4.1.4 Discrétisation par les éléments finis

Nous allons à présent considérer des problèmes issus de la discrétisation de problèmes continus via des méthodes d'éléments finis. Nous allons reprendre l'exemple fourni par Michal Kocvara que nous avions étudié lors des deux chapitres précédents et qui est associé au graphe 2.7. Nous avions la répartition obtenue à la fin de l'application de la méthode classique à la figure 2.12. Le programme de type agrégation donnait le résultat présenté à la figure 3.3. La répartition des points de la grille grossière semble plus régulière sur la représentation 3.3. De plus, nous pouvons à nouveau constater que la méthode classique garde davantage de points pour la grille grossière que l'algorithme de [20].

Maintenant que nous avons étudié la sélection des points de la grille grossière pour les deux types d'algorithmes (type classique et type agrégation), nous allons montrer son implication dans la phase de résolution.

4.2 Résultats numériques

Dans cette section, nous allons présenter des résultats numériques obtenus lors de l'application des deux programmes à divers problèmes, semblables à ceux illustrés dans la section précédente.

Pour présenter les résultats, nous utiliserons des tableaux contenant les données suivantes :

- *algo* est le type d'algorithme utilisé, il vaut *Clss* pour la méthode de type classique et *Agr* pour la méthode de la "Smooth Aggregation",
- L est le nombre de grilles souhaité,
- n est le nombre de noeuds que comporte la grille la plus fine,
- n_L est le nombre de points que contient la grille la plus grossière,
- t_{pr} est le temps CPU écoulé lors de la phase de préparation,
- $t_{rés}$ est le temps CPU utilisé pour la phase de résolution,
- *Iter* est le nombre d'itérations V-cylces effectuées lors de la phase de résolution,
- v_c est le R-taux de convergence dont la définition est la suivante [13] :

  Définition 9. *Soit $\{x^k\} \subset R^n$ une suite d'itérés qui converge vers x^*, le R-taux de convergence est donné par :*

$$R\{x^k\} = \limsup_{k\to\infty} \left\|x^k - x^*\right\|^{1/k}.$$

- C_{gr} est la complexité de grille (cfr Définition 6),
- et C_{op} est la complexité d'opérateurs (cfr Définition 7).

Dans les analyses des sections qui suivent, nous nous intéressont plus particulièrement aux deux colonnes, t_{pr} et $t_{rés}$, concernant le temps CPU de la phase de préparation et celui de la phase de résolution.

Pour ces analyses, nous avons fixé un nombre maximal d'itérations à 1000 et nous avons pris comme condition d'arrêt que l'erreur relative soit plus petite que 10^{-9} (ces paramètres sont valables pour tous les problèmes étudiés sauf contre-indication). Le paramètre θ de la méthode classique (cfr algorithme de la section 2.9.2) est fixé à 0.25, et les paramètres ϵ et w de l'algorithme de la "Smooth Aggregation" sont ceux définis dans (3.18). Pour chaque classe de problèmes, nous introduirons les problèmes utilisés puis nous présenterons les résultats obtenus.

4.2.1 Problèmes des éléments finis

Nous allons commencer par les problèmes fournis par Michal Kocvara. Nous avions déjà appliqué la méthode de [4] à l'un de ces exemples à la figure 2.13. Nous allons maintenant considérer trois problèmes de ce type : le problème 1668, le problème 6440 et le problème 25296. Ces systèmes ont les données suivantes :

	problème 1668	problème 6440	problème 25296
nombre d'équations	1532	6168	24752
nombre d'éléments non nuls de A	10258	42246	171406

problème 1668									
Algo	L	n	n_L	t_{pr}	$t_{rés}$	Iter	v_c	C_{gr}	C_{op}
Clss	2	1532	565	1.031	0.188	8	$5.4812e-2$	1.3688	0.7897
Agr	2	1532	394	0.219	0.234	14	$2.165949e-1$	1.257180	0.685904
Clss	3	1532	215	1.156	0.156	8	$6.2675e-2$	1.5091	1.1152
Agr	3	1532	63	0.328	0.281	20	$3.420289e-1$	1.298303	0.775492
problème 6440									
Clss	2	6168	2214	6.531	0.829	8	$6.0024e-2$	1.3589	0.7911
Agr	2	6168	1572	1.500	1.422	17	$2.831142e-1$	1.254864	0.709464
Clss	3	6168	823	8.062	0.688	8	$6.7616e-2$	1.4924	1.1345
Agr	3	6168	246	1.969	1.688	22	$3.749149e-1$	1.294747	0.831653
problème 25296									
Clss	2	24752	8860	80.344	4.360	8	$5.7696e-2$	1.3580	0.8025
Agr	2	24752	6324	18.828	8.781	20	$3.367848e-1$	1.255495	0.726054
Clss	3	24752	3179	92.141	3.422	8	$6.4453e-2$	1.4864	1.1505
Agr	3	24752	964	29.610	9.234	26	$4.452010e-1$	1.294441	0.865967

FIG. 4.7 – Tableau reprenant les résultats obtenus lors de l'application des programmes aux problèmes fournis par Michal Kocvara.

La figure 4.7 contient le tableau reprenant les résultats obtenus en appliquant les deux programmes aux problèmes. Remarquons que la construction des grilles est beaucoup plus rapide pour la méthode de type agrégation. Cependant, la méthode classique converge plus vite mais cela ne compense pas le temps perdu lors de la phase de préparation. Nous pouvons constater que le nombre de points sélection-

nés pour les grilles grossières est plus grand pour la méthode classique que pour la méthode d'agrégation, ce qui pourrait expliquer la convergence plus rapide de cette méthode.

4.2.2 Problèmes anisotropiques

Nous allons à présent considérer trois systèmes linéaires dont les matrices sont engendrées par le "stencil" (4.2) avec un pas de discrétisation $h = 1$ et dont les membres de droite sont des vecteurs dont toutes les composantes sont égales à 1. Ces problèmes ont les dimensions suivantes :

	problème 6400	problème 22500	problème 122500
nombre d'équations	6400	22500	122500
nombre d'éléments non nuls de A	31680	111900	611100

problème 6400									
Algo	L	n	n_L	t_{pr}	$t_{rés}$	Iter	v_c	C_{gr}	C_{op}
Clss	2	6400	3200	3.328	0.812	8	$6.3155e-2$	1.5	1.0885
Agr	2	6400	2160	1.672	0.750	13	$1.882561e-1$	1.3375	0.795518
Clss	3	6400	1600	4.422	0.562	8	$6.4655e-2$	1.75	1.5242
Agr	3	6400	800	2.422	0.625	14	$2.075179e-1$	1.4625	1.005871
problème 22500									
Clss	2	22500	11250	29.735	4.312	8	$6.0727e-2$	1.5	1.0939
Agr	2	22500	7650	42.719	3.313	12	$1.580715e-1$	1.34	0.805612
Clss	3	22500	5550	38.782	2.515	8	$6.4264e-2$	1.7467	1.5303
Agr	3	22500	2700	59.047	2.922	14	$2.169875e-1$	1.46	1.013798
problème 122500									
Clss	2	122500	61250	852.093	56.891	8	$5.7614e-2$	1.5	1.0974
Agr	2	122500	40950	1373.891	2265.437	1000	1.925578	1.334286	0.798972
Clss	3	122500	30450	1054.437	18.016	8	$6.3255e-2$	1.7486	1.5415
Agr	3	122500	14000	2058.359	18.031	14	$2.202786e-1$	1.448571	1.001335

FIG. 4.8 – Tableau reprenant les résultats obtenus lors de l'application des programmes aux problèmes anisotropiques, engendrés par le "stencil" (4.2).

Sur le tableau de la figure 4.8, nous pouvons constater que la méthode classique est plus rapide et plus efficace que l'algorithme de type agrégation pour les problèmes à 22500 noeuds et à 122500 noeuds. En effet, pour ces deux problèmes, nous avons que la somme des temps t_{pr} et $t_{rés}$ pour la méthode classique est moins élevée que la somme de ces temps pour l'algorithme de la "Smooth Aggregation". Dans certains de ces cas, nous avons même que les deux temps, t_{pr} et $t_{rés}$, sont meilleurs pour la méthode classique. Ceci est dû à la forme particulière du problème. Nous avons notamment que les ensembles S_i pour chaque noeud i de la grille fine sont composés

d'au plus deux éléments (cfr définition du "stencil" (4.2)), ce qui a pour effet d'accélérer le calcul de l'opérateur d'interpolation et du deuxième schéma de coloriage. Dans la section suivante, nous allons considérer d'autres problèmes anisotropiques.

Autres problèmes anisotropiques

Nous considérons ici des problèmes anisotropiques issus de la discrétisation de l'équation suivante :

$$-u_{xx} - 100\,u_{yy} = 1, \tag{4.5}$$

la matrice de discrétisation de ce problème étant donnée [12] par le "stencil" suivant :

$$\begin{bmatrix} -101 & -398 & -101 \\ 196 & 808 & 196 \\ -101 & -398 & -101 \end{bmatrix}_h . \tag{4.6}$$

Notons que les matrices engendrées par ce "stencil" ne sont pas des M-matrices. Nous allons considérer quatre problèmes de ce type dont les données sont les suivantes :

	problème 6400	problème 22500	problème 122500	problème 202500
nombre d'équations	6400	22500	122500	202500
nombre d'éléments non nuls de A	56644	200704	1098304	1817104

Nous constatons, en analysant les résultats du tableau de la figure 4.9, que la méthode classique est plus intéressante pour les problèmes à 6400 et 22500 noeuds, car elle converge plus vite. Nous avons que la somme des temps t_{pr} et $t_{rés}$, pour ces problèmes, est moins élevée pour la méthode classique que pour la méthode de type agrégation. Néanmoins, au delà d'une certaine dimension (cfr problème 122500 et problème 202500), le temps gagné lors de la phase de résolution par la méthode classique ne compense plus celui perdu lors de la phase de préparation. L'algorithme de [20] est alors plus intéressant.

Remarquons également que la méthode d'agrégation n'a pas d'avantages à prendre plus de grilles pour un problème. En effet, l'algorithme met plus de temps pour résoudre les problèmes à l'aide de trois niveaux grossiers plutôt qu'à l'aide de deux. Ce phénomène se présente aussi dans la plupart des autres exemples.

problème 6400									
Algo	L	n	n_L	t_{pr}	$t_{rés}$	Iter	v_c	C_{gr}	C_{op}
Clss	2	6400	1600	8.171	17.860	251	$9.2071e-1$	1.25	0.3588
Agr	2	6400	729	0.922	29.625	555	$9.633056e-1$	1.113906	0.223166
Clss	3	6400	420	9.281	13.313	238	$9.1648e-1$	1.3156	0.4206
Agr	3	6400	108	1.016	32.766	574	$9.645158e-1$	1.130781	0.239443
problème 22500									
Clss	2	22500	5625	70.219	85.766	274	$9.2699e-1$	1.25	0.3599
Agr	2	22500	2649	8.109	140.860	653	$9.687386e-1$	1.117733	0.229816
Clss	3	22500	1443	77.015	60.547	270	$9.2602e-1$	1.3141	0.4234
Agr	3	22500	339	8.359	176.625	824	$9.751390e-1$	1.1328	0.245008
problème 122500									
Clss	2	122500	30625	1884.735	769.625	294	$9.3185e-1$	1.2500	0.3606
Agr	2	122500	13689	239.844	947.656	710	$9.712213e-1$	1.111747	0.222435
Clss	3	122500	7743	1943.687	446.391	301	$9.3335e-1$	1.3132	0.4235
Agr	3	122500	1638	242.734	1201.511	1000	$9.816009e-1$	1.125118	0.235703
problème 202500									
Clss	2	202500	50625	4948.938	1268.781	299	$9.3289e-1$	1.2500	0.3607
Agr	2	202500	22949	893.781	1737.531	720	$9.716203e-1$	1.113328	0.224767
Clss	3	202500	12768	5278.890	816.844	307	$9.3470e-1$	1.3131	0.4236
Agr	3	202500	2623	903.844	2057.469	1000	$9.823456e-1$	1.126281	0.237754

FIG. 4.9 – Tableau reprenant les résultats obtenus lors de l'application des programmes aux problèmes anisotropiques, engendrés par le "stencil" (4.6).

Nous pouvons aussi noter qu'une itération de la méthode classique prend plus de temps qu'une itération de l'algorithme de type agrégation. En effet, si nous regardons, par exemple, le problème à 122500 noeuds avec deux grilles, nous pouvons voir que le temps de la phase de résolution, $t_{rés}$, est de 769.625 secondes CPU pour la méthode classique et est de 947.656 secondes CPU pour l'algorithme de type agrégation. Les nombres d'itérations effectuées par ces deux méthodes sont respectivement 294 et 710. A partir de ces données, nous pouvons en déduire qu'il faut à peu près 2.6178 secondes CPU pour une itération de l'algorithme classique et 1.3347 secondes CPU pour une itération de la méthode de type agrégation. Ce phénomène est en partie lié à la complexité d'opérateurs car, comme nous l'avions vu dans le deuxième chapitre, les coûts dominants de la phase de résolution sont ceux de la relaxation et du calcul des résidus. Ces coûts sont proportionnels au nombre d'éléments non nuls des opérateurs grossiers (cfr section 2.8). Ce phénomème peut être également observé pour la plupart des autres exemples.

4.2.3 Problèmes isotropiques

Considérons à présent des problèmes associés au "stencil" (2.29) donné par [4]. Comme pour les systèmes précédents, nous prendrons comme membre de droite un vecteur dont toutes les composantes valent 1 et nous choisirons un pas de discrétisation $h = 1$. Nous allons considérer trois problèmes de ce type dont les données sont

les suivantes :

	problème 2500	problème 6400	problème 22500
nombre d'équations	2500	6400	22500
nombre d'éléments non nuls de A	21904	56644	200704

Nous pouvons remarquer que les résultats obtenus (cfr tableau de la figure 4.10) pour ces problèmes sont similaires à ceux obtenus pour les problèmes fournis par Michal Kocvara. Ici, la différence de rapidité pour la construction des grilles est même plus fappante.

problème 2500									
Algo	L	n	n_L	t_{pr}	$t_{rés}$	Iter	v_c	C_{gr}	C_{op}
Clss	2	2500	625	2.235	0.187	8	$6.3787e-2$	1.2500	0.3574
Agr	2	2500	289	0.297	0.265	16	$2.616252e-1$	1.115600	0.223749
Clss	3	2500	144	2.422	0.187	9	$7.4750e-2$	1.3076	0.4102
Agr	3	2500	37	0.329	0.328	19	$3.303299e-1$	1.1304	0.235847
problème 6400									
Clss	2	6400	1600	8.172	0.547	8	$6.6542e-2$	1.25	0.3588
Agr	2	6400	729	0.922	0.765	16	$2.667769e-1$	1.113906	0.223166
Clss	3	6400	400	9.187	0.469	9	$9.5089e-2$	1.3125	0.4182
Agr	3	6400	109	1.015	0.891	18	$3.131581e-1$	1.130937	0.239602
problème 22500									
Clss	2	22500	5625	69.375	2.563	8	$6.9944e-2$	1.25	0.3599
Agr	2	22500	2649	7.813	3.437	16	$2.538825e-1$	1.117733	0.229816
Clss	3	22500	1369	75.922	2.000	9	$8.1189e-2$	1.3108	0.4191
Agr	3	22500	365	8.360	4.265	20	$3.388229e-1$	1.133956	0.246173

FIG. 4.10 – Tableau reprenant les résultats obtenus lors de l'application des programmes aux problèmes engendrés par le "stencil" (2.29).

4.2.4 Problèmes mixtes

Reprenons maintenant les problèmes mixtes (4.3) expliqués dans la première partie de ce chapitre. Comme précédemment, nous prendrons un vecteur dont toutes les composantes valent 1 pour membre de droite et un pas de discrétisation $h = 1$. Les données de ces problèmes sont reprises dans le tableau suivant :

	problème 2500	problème 6400	problème 25600
nombre d'équations	2500	6400	25600
nombre d'éléments non nuls de A	10852	28162	113922

problème 2500									
Algo	L	n	n_L	t_{pr}	$t_{rés}$	Iter	v_c	C_{gr}	C_{op}
Clss	2	2500	1213	0.969	8.672	343	$9.4126e-1$	1.4852	1.1512
Agr	2	2500	893	0.297	0.203	12	$1.459343e-1$	1.3572	0.839937
problème 6400									
Clss	2	6400	3180	3.703	70.328	838	$9.7555e-1$	1.4969	1.1903
Agr	2	6400	2228	1.156	0.625	12	$1.695093e-1$	1.348125	0.844258
problème 25600									
Clss	2	25600	12760	43.344	485.125	1000	$9.9060e-1$	1.4984	1.2063
Agr	2	25600	8680	27.313	3.484	13	$1.840674e-1$	1.339063	0.844824

FIG. 4.11 – Tableau reprenant les résultats obtenus lors de l'application des programmes aux problèmes mixtes (4.3).

Remarquons dans le tableau de la figure (4.11) que le programme de type agrégation converge plus vite que celui de type classique. Pour le problème 25600, l'algorithme classique a atteind le maximum d'itérations. De plus, à partir du troisième niveau, la méthode classique ne parvient plus à former les grilles grossières, ceci est dû au fait que la matrice n'est pas une M-matrice. En effet, la construction de l'opérateur d'interpolation échoue pour ces problèmes car les termes de la somme du dénominateur de l'équation (2.10) s'annulent exactement, ce qui ne peut pas arriver lorsqu'on traite une M-matrice.

4.3 Profil de performance

Dans cette section, nous allons comparer les deux algorithmes sur base de profils de performance. Ce type de graphe a pour but de comparer l'efficacité et la robustesse de différents algorithmes à l'aide d'un ensemble de problèmes tests.

Ce type de profil se construit de la manière suivante [15] : définissons d'abord un ensemble d'algorithmes $\mathcal{A}$ et considérons un ensemble de problèmes tests, noté $\mathcal{T}$. Dans notre cas, l'ensemble $\mathcal{A}$ comprendra les deux programmes (1 : la "Smooth Aggregation" et 2 : la méthode classique), et l'ensemble $\mathcal{T}$ contiendra 33 problèmes tests. Ensuite, définissons une performance $s_{ij} \geq 0$ pour chaque problème test, $j \in \mathcal{T}$, et pour chaque algorithme, $i \in \mathcal{A}$. La valeur de la variable s_{ij} diminue lorsque le problème est plus performant et vaut $+\infty$ lorsque le programme i ne résout pas le problème j. Dans notre cas, nous prendrons comme indice de performance, s_{ij}, la somme du temps CPU pour la phase de préparation et le temps CPU pour la phase de résolution. Si le nombre maximal d'itérations est atteind lors de la résolution du problème j par l'algorithme i, alors $s_{ij} = \infty$.

Le profil de performance est représenté à l'aide de la fonction $p(\sigma)$ qui est donné par :

$$p_i(\sigma) = \frac{\sum_{j \in \mathcal{T}} k(s_{ij}, s_j^*, \sigma)}{\sharp(\mathcal{T})} \qquad \forall \sigma \geq 1,$$

où s_j^* est la meilleure performance associée au problème j et définie par :

$$s^* = \min_{i \in \mathcal{A}} s_{ij},$$

et $k(s_{ij}, s_j^*, \sigma)$ est un indicateur de succès σ-relatif, donné par :

$$k(s_{ij}, s_j^*, \sigma) = \begin{cases} 1 & \text{si } s_{ij} \leq \sigma s_j^* \\ 0 & \text{sinon.} \end{cases}$$

Nous avons que lorsque σ vaut 1, la valeur $p_i(1)$ représente la part des problèmes pour lesquels l'algorithme i est le plus efficace. Ce qui signifie que la partie gauche du graphique montre l'efficacité des différents algorithmes étudiés. Lorsque la variable σ tend vers l'infini, $p_i(\sigma)$ représente la part des problèmes que le programme i parvient à résoudre. La partie de droite du graphe correspond à la robustesse des algorithmes.

Le tableau de la figure 4.12 contient la description des 33 problèmes tests, les résultats obtenus pour la variable performance s_{ij} ($i = 1$ pour la "Smooth Aggregation" et $i = 2$ pour la méthode classique) et les nombres d'itérations (k_1 pour la "Smooth Aggregation" et k_2 pour la méthode classique).

Ces problèmes tests sont soit des systèmes dont les matrices proviennent de sources internet ([5] et [9]), soit des problèmes étudiés dans la section précédente. Il est à noter que les matrices de ces systèmes sont définies positives et symétriques. Elles ne sont pas, pour la pluspart, des M-matrices.

j	n	s_{1j}	k_1	s_{2j}	k_2
1	1806	242.033000	2681	1702.391000	2107
2	3948	Inf	3000	Inf	3000
3	4884	466.359000	199	483.000000	621
4	1224	22.062000	383	50.312000	333
5	1138	20.250000	1242	Inf	3000
6	685	1.094000	99	4.359000	372
7	1806	245.094000	2681	1699.703000	2107
8	3948	Inf	3000	Inf	3000
9	1824	63.906000	1584	Inf	3000
10	40000	34.281000	8	92.422000	3
11	7102	Inf	3000	1044.391000	775
12	7102	110.453000	5	12.281000	3
13	14822	Inf	3000	531.047000	275
14	6867	Inf	3000	1411.641000	475
15	6168	2.203000	8	6.797000	4
16	1532	0.484000	6	0.891000	3
17	24752	23.375000	9	81.843000	4
18	2500	0.656000	6	0.922000	4
19	6400	2.110000	7	3.578000	4
20	10000	5.578000	6	7.500000	4
21	2500	0.515000	8	2.110000	4
22	6400	1.500000	8	8.500000	4
23	22500	9.625000	7	70.125000	4
24	2500	2.625000	137	4.031000	87
25	6400	10.062000	178	15.188000	102
26	22500	58.187000	235	104.922000	117
27	2500	0.500000	12	4.031000	343
28	6400	1.781000	12	15.188000	838
29	25600	528.469000	13	Inf	3000
30	40000	28.625000	3	172.813000	7
31	40806	61.594000	4	127.172000	3
32	40000	30.109000	3	172.203000	7
33	48962	Inf	3000	287.7500	18

FIG. 4.12 – Tableau reprenant les performances et les nombres d'itération obtenus pour les 33 problèmes tests (1 pour la "Smooth Aggregation" et 2 pour la méthode classique).

A présent, nous pouvons représenter sur la figure 4.13 le profil de performance obtenu à partir des 33 problèmes tests. Nous y avons dessiné la courbe $p_1(\sigma)$ en ligne continue pour l'algorithme de la "Smooth Aggregation" et la courbe $p_2(\sigma)$ en ligne discontinue pour la méthode classique. Nous avons pris pour condition d'arrêt que l'erreur soit plus petite que 10^{-4} et un maximum d'itérations fixé à 3000 itérations.

Nous pouvons remarquer sur le graphe de la figure 4.13 que la méthode de type agrégation est plus performante que celle de type classique. En effet, si nous regardons dans la partie gauche du graphique, nous constatons que la courbe $p_1(\sigma)$

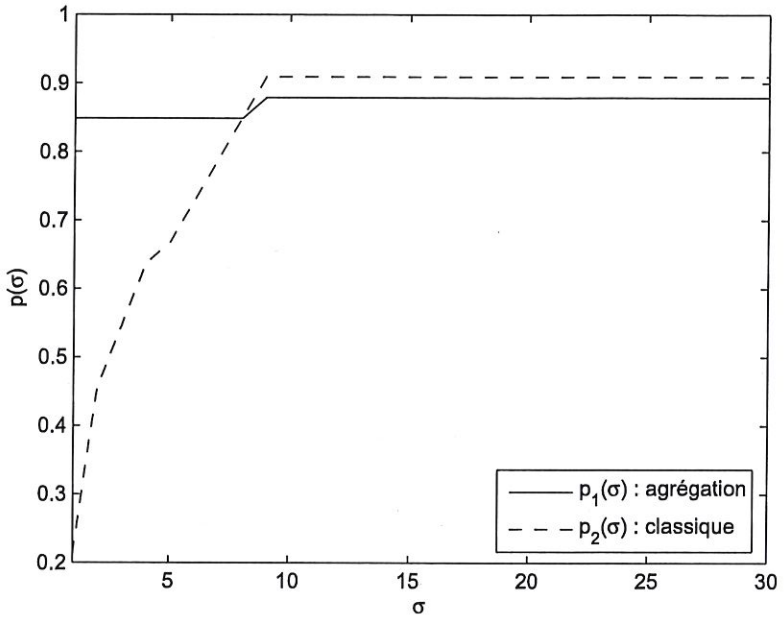

FIG. 4.13 – Profil de performance : Agrégation vs Classique pour une erreur relative inférieure à 10^{-4} et un maximum de 3000 itérations.

passe largement au-dessus de $p_2(\sigma)$ quand σ est proche de 1. La méthode classique est légèrement plus robuste puisqu'elle parvient à résoudre plus de problèmes (cfr partie droite du graphe). Ceci est lié au fait que la méthode d'agrégation converge moins vite et atteind la limite d'itérations autorisée (fixée à 3000 itérations) pour quelques problèmes.

Le code pour générer le profil de performance se trouve en annexe à la page 124.

En conclusion, nous avons constaté dans ce chapitre que la méthode de multigrilles algébriques de type classique, développée par [4], demande plus de points pour la construction de son opérateur d'interpolation que la méthode de la "Smooth aggregation", ce qui explique pourquoi elle converge plus vite. Cependant, la méthode classique prend beaucoup plus de temps pour la phase de prépation que l'algorithme de type agrégation, excepté pour certains problèmes anisotropiques.

Nous avons également remarqué que la méthode d'agrégation n'a pas d'avantages à prendre plus de deux grilles. En effet, pour beaucoup de problèmes, l'algorithme met plus de temps avec trois niveaux grossiers plutôt qu'avec deux. Ceci est dû au fait que les problèmes étudiés ne sont pas de très grandes dimensions et donc deux grilles suffisent pour cet algorithme.

Finalement, nous avons aussi noté qu'une itération de la méthode classique prend en général plus de temps qu'une itération de l'algorithme de type agrégation.

Conclusion

Dans ce mémoire, nous nous sommes intéressés aux méthodes de multigrilles algébriques pour la résolution de systèmes linéaires issus de la discrétisation de problèmes continus sur une grille. Ces discrétisations étant généralement obtenues par des méthodes d'éléments finis ou de différences finies.

Nous avons d'abord introduit, dans un premier chapitre, les méthodes de *multigrilles géométriques*. Celles-ci s'appliquaient à des problèmes discrétisés sur des grilles admettant une certaine structure et des propriétés géométriques importantes.

Dans le deuxième chapitre, nous avons commencé à analyser les méthodes de multigrilles algébriques qui, contrairement au cas géométrique, ne nécessitaient pas de grille physique de référence. En effet, ces méthodes s'appliquaient directement à la matrice du système, et donc à des problèmes ne possédant aucune grille de référence ou à des problèmes discrétisés sur des grilles irrégulières. Nous nous sommes ensuite attardés sur une méthode particulière de multigrilles algébriques, dite *de type classique* et dévelopée dans [4], que nous avons implémentée dans le cadre de ce mémoire.

Dans le chapitre suivant, nous avons présenté une méthode particulière de multigrilles algébriques, dite *de type agrégation*, proposée dans [20] et portant le nom de "Smooth Aggregation".

Dans le dernier chapitre, nous avons comparé numériquement les deux méthodes particulières étudiées : la méthode classique de [4] (implémentée dans le cadre de ce mémoire) et la méthode de la "Smooth Aggregation" de [20] (implémentée par Michal Kocvara).

A l'issu de ce mémoire, nous pouvons dire que la méthode de la "Smooth Aggregation" est plus performante, en général, que la méthode classique de [4]. Nous avons en effet remarqué que l'algorithme de type agrégation prenait moins de temps que la méthode classique, malgré le fait qu'elle converge moins vite. Néanmoins, nous avons constaté qu'il existe une classe de problèmes, à savoir les problèmes anisotropiques, pour laquelle la méthode classique donne de bons résultats et parfois même meilleurs que ceux obtenus par la méthode d'agrégation.

Annexe 1

Notation "Stencil"

Dans cette section, nous allons introduire la notation "stencil", qui est une formulation condensée d'une matrice. Elle s'applique aux opérateurs de discrétisation A^h qui sont définis sur une grille cartésienne carrée ou rectangulaire de pas de discrétisation $h = (h_x, h_y)$.

Nous allons considérer ici uniquement le cas des grilles en deux dimensions. Soit, définissons la fonction grille suivante :

$$\begin{array}{rccl} w_h : & G_h & \to & \mathbb{R} \\ & (x,y) & \rightsquigarrow & w_h(x,y) \end{array}$$

où $G_h = \{(x,y) \mid x = x_i = i\,h_x,\ y = y_i = i\,h_y,\ i,j \in \mathbb{Z}\}$ est la grille.

Notons l'opérateur "stencil" par $[s_{\kappa_1,\kappa_2}]_k$, où

$$[s_{\kappa_1,\kappa_2}]_k = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & s_{-1,1} & s_{0,1} & s_{1,1} & \cdots \\ \cdots & s_{-1,0} & s_{0,0} & s_{1,0} & \cdots \\ \cdots & s_{-1,-1} & s_{0,-1} & s_{1,-1} & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}_h ,$$

avec $s_{\kappa_1,\kappa_2} \in \mathbb{R}$ et définissons-le de la manière suivante :

$$[s_{\kappa_1,\kappa_2}]_k\, w_h(x,y) = \sum_{(\kappa_1,\kappa_2)} s_{\kappa_1,\kappa_2}\, w_h(x + \kappa_1\, h_x, y + \kappa_2\, h_y),$$

où on suppose qu'un nombre fini de coefficients s_{κ_1,κ_2} sont différents de zéros. Distinguons deux types de "stencil" couramment utilisés : le "stencil" à cinq points qui prend la forme suivante :

$$\begin{bmatrix} & s_{0,1} & \\ s_{-1,0} & s_{0,0} & s_{1,0} \\ & s_{0,-1} & \end{bmatrix}_h ,$$

et celui à neuf points qui est décrit comme tel :

$$\begin{bmatrix} s_{-1,1} & s_{0,1} & s_{1,1} \\ s_{-1,0} & s_{0,0} & s_{1,0} \\ s_{-1,-1} & s_{0,-1} & s_{1,-1} \end{bmatrix}_h .$$

Pour mieux visualiser ce concept, illustrons-le à l'aide d'un exemple.

Exemple

Sans frontière

Reprennons le Laplacien (2.22) introduit au chapitre 2 et sa discrétisation (2.26), nous avons alors :

$$\begin{aligned}[A^h u](x,y) &= \frac{1}{h^2}\big[-u(x+h_x,y) - u(x,y+h_y) + 4u(x,y) \\ &\qquad - u(x-h_x,y) - u(x,y-h_y)\big] \\ &= \frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}_h u(x,y),\end{aligned} \tag{4.7}$$

où $x = x_i = i\,h_x$, $y = y_i = i\,h_y$, $i,j \in \mathbb{Z}$. La figure (4.14) représente le point $u(x,y)$ (désigné par le point noir) et les points $u(x+h_x,y)$, $u(x,y+h_y)$, $u(x-h_x,y)$, $u(x,y-h_y)$ (identifiés par des points blancs).

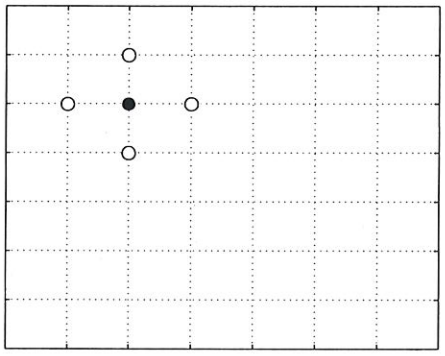

FIG. 4.14 – Représentation du concept de "stencil". Le point noir représente le noeud $u(x,y)$ et les points blancs désignent les noeuds qui interviennent dans la définition de $u(x,y)$ et qui sont concernés par le "stencil".

Avec frontière

Si l'on considère une grille de dimension $(n-1)*h \times (n-1)*h$ où n est le nombre de noeuds que comporte la grille, pour un point (x,y) appartenant à la frontière côté

gauche, nous avons que :

$$\begin{aligned}[A^h u](x,y) &= \frac{1}{h^2}\left[-u(x+h_x,y) - u(x,y+h_y) + 4u(x,y)\right. \\ &\qquad \left. - u(x,y-h_y)\right] \\ &= \frac{1}{h^2}\begin{bmatrix} & -1 & \\ 0 & 4 & -1 \\ & -1 & \end{bmatrix}_h u(x,y),\end{aligned}$$

où $x = x_i = i\,h_x$, $y = y_i = i\,h_y$, $i = 1$ et $1 < j < n$. La figure (4.15) représente le point $u(x,y)$ (désigné par le point noir) et les points $u(x+h_x,y)$, $u(x,y+h_y)$, $u(x,y-h_y)$ (identifiés par des points blancs).

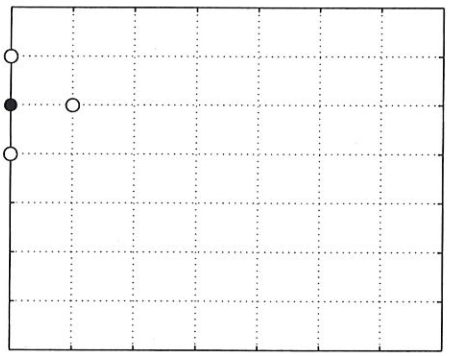

FIG. 4.15 – Représentation du concept de "stencil". Cas d'un point qui se trouve sur la frontière de la grille.

Remarque : on se contente en général de noter uniquement le "stencil" pour les points intérieurs au domaine comme (4.7), l'application aux conditions frontières est laissée au lecteur.

La fonction `UnStencil.m` est un code pour créer un problème à partir d'un "stencil du type" (4.7), pour une grille à deux dimensions. Elle se lance avec la commande suivante :

```
UnStencil(n,s)
```

dont les arguments sont :
- n^2 : le nombre de points de la grille ;
- $s \in \mathbb{R}^{3\times 3}$: "stencil" du problème.

Cette fonction rend la matrice de discrétisation et les coordonnées des noeuds.

Fonction UnStencil.m

```
function [A,nodes] = UnStencil(n,s)
%=========================================================================
% UnStencil : fonction pour la creation de matrices a partir de stencil
%          a max 9 points pour 1 grille à 2 dim
%
% INPUT  : - n^2 : dimension de la matrice A
%          - s : stencil de la forme :
%
%                ( s_(-1,1)    s_(0,1)    s_(1,1)   )
%                ( s_(-1,0)    s_(0,0)    s_(1,0)   )
%                ( s_(-1,-1)   s_(0,-1)   s_(1,-1)  )
%
% OUTPUT : - A : matrice associée au stencil s
%          - nodes : coordonnées des noeuds de la matrice
%
% auteur : Sylvie Detournay
%
% dernière modif : 1005
%=========================================================================

%creation de la matrice
dim = n*n;
A = s(2,2).*speye(dim);


for i = 1 : dim

     if (mod(i,n) ~= 1)
          A(i,i-1) = s(2,1);
     end
     if (mod(i,n) ~= 0)
          A(i,i+1) = s(2,3);
     end

     if (i-n-1 > 0) & (mod(i,n) ~= 1)
          A(i,i-n-1) = s(1,3);
     end
     if (i-(n) > 0)
          A(i,i-n) = s(1,2);
     end
     if (i-n+1 > 0) & (mod(i,n) ~= 0)
          A(i,i-n+1) = s(1,1);
     end

     if (i+n < dim+1 ) & (mod(i,n) ~= 1)
          A(i,i+n-1) = s(3,3);
     end
```

```
      if (i+n < dim+1 )
           A(i,i+n) = s(3,2);
      end
      if (i+n+1 < dim+1 ) & (mod(i,n) ~= 0)
           A(i,i+n+1) = s(3,1);
      end
end

%creation des noeuds pour les dessins
nodes = zeros(2,n*n);

for k = 1 : n
    nodes(1, ((k-1)*n + 1) :k*n) = 1 : n;
    nodes(2, ((k-1)*n + 1) :k*n) = k.* ones(1,n);
end
```

Annexe 2

Codes

Dans cette annexe se trouvent les codes des différents algorithmes expliqués dans le deuxième et troisième chapitres de ce mémoire. Ces programmes nécessitent le programme `matlab` version 7 ou supérieur.

Algorithme de [20] implémenté par Michal Kocvara

Le programme pour résoudre les systèmes linéaire, $Au = f$, issus de la discrétisation de problèmes continus, via la méthode de [20] et implenté par Michal Kocvara, se lance avec la commande suivante :

```
amg(levels,A,f,pred,picture,filter,fichier)
```

où les arguments sont :

- A : matrice du système,
- f : membre de droite du système,
- $levels > 1$: nombre de niveaux souhaités,
- $pred$: matrice contenant les coordonnées des noeuds de la grille la plus fine (nécessaire uniquement si $picture = 1$),
- $picture$: représentation graphique des noeuds du niveau grossier (1 -> OUI, 0 -> NON),
- $filter$: filtre la matrice du système lors de la relaxation de l'opérateur d'interpolation (1 -> OUI, 0 -> NON).

Fonction amg.m

```
function [x]=amg(levels,A,F,pred,picture,filter)
%
% amg...function for the solution of the linear system Ax=F with a
% symmetric positive definite matrix A by the algebraic multigrid method
% levels ...required number of levels
% pred ...matrix of coordinates of the fine-level nodes
%        (needed only for picture=1)
```

```
% picture ...graphical representation of coarse-level nodes
%            (1 -> YES, 0 -> NO)
% filter ...filter the system matrix when smoothing the prolongation
%            (1 -> YES, 0 -> NO)
%
% kocvara@utia.cas.cz, last change July 31, 2005

tic

%...Amat contains the coarser and coarser system matrices,
%Amat{1} is the finest level

str=sprintf('PREPARATION PHASE');disp(str)
Amat{1} = A;
n=length(A);
str=sprintf('Dimension of level 1: %d', n);
disp(str)

%...Pmat contains the interpolation matrices, Pmat{i} works between
%levels i and i+1

Pmat{1}=[];

%...here we compute the interpolation matrices Pmat{i} for the given number
%of levels. Using Pmat{i}, we also compute the coarse-level matrices
%Amat{i}

for level=1:levels-1
    eps = 0.08*(.5)^(level);
    [P]=aggr2(Amat{level},eps,pred,picture,filter);
    Amat{level+1} = P'*Amat{level}*P;
    Pmat{level} = P;
    [n1,m1]=size(Amat{level+1});
    str=sprintf('Dimension of level %d: %d', level+1,n1);
    disp(str)
end
toc

%...having the system and interpolation matrices, we solve the finest-level
%problem by the standard MG method (function mgm)
tic
str=sprintf('SOLUTION PHASE');disp(str)
x=zeros(length(F),1);
nono=1.0; iter=0; nof = norm(F);
while nono>1e-9
    iter = iter+1;
    x = mgm(1,levels,x,F,Amat,Pmat);
    nono=norm(A*x-F)/nof;
end
```

```
str=sprintf('Number of iterations: %d', iter);
disp(str)
avercon=nono^(1/iter);
str=sprintf('Average speed of convergence: %d', avercon);
disp(str)
toc

%=============================================================
function xn=mgm(level,levels,x,f,Amat,Pmat);
% multigrid method for solving A.x = f using recurrent formula
% level...actual level
% levels...total number of levels
% x...solution on the actual level
% f...right-hand side on the actual level
% Amat...system matrices
% Pmat...interpolation matrices

A = Amat{level};
if level==levels
    xn  =  A\f;
else
    P   =  Pmat{level};
    x   =  sor(A, x, f, 1.0, 1, 1e-5);
    res =  A*x - f;
    res = -P'*res;
    cor =  mgm(level+1,levels,zeros(length(res),1),res,Amat,Pmat);
    xn  =  x + P*cor;
    xn  =  sor(A, xn, f, 1.0, 1, 1e-5);
end
```

Il est à noter que, pour la réalisation des dessins de ce mémoire, le code a été légèrement modifié. Ici, nous présentons la version originale du code. La fonction `amg.m` fait appel aux sous-routines suivantes :

Fonction aggr2.m

```
function [P]=aggr2(A,eps,p,picture,filter)
%
% computing interpolation matrices for algebraic multigrid by the smoothed
% aggregation method of Vanek-Mandel-Brezina.
% A ...the system matrix which we want to coarsen
% eps ...epsilon parameter (recomended as eps = 0.08*(.5)^(level))
% p ...matrix of nodal coordinates (needed only for picture=1)
% picture ...graphical representation of coarse-level nodes
%            (1 -> YES, 0 -> NO)
% filter ...filter the system matrix when smoothing the prolongation
```

```
%            (1 -> YES, 0 -> NO)
%
% kocvara@utia.cas.cz, last change July 31, 2005

[n,m]=size(A);
N=sparse(n,m);

%...first precompute the strongly coupled neighborhoods and (when
% requested) the filtered matrix Afil

if filter
    Afil=A;
end
eps2=eps^2;
for i=1:n
    Nip = [];
    aiieps = eps2*A(i,i);
    jj= find(A(:,i));
    ico=1;
    ssum = 0.0;
    jjss=[];
    for j=1:length(jj)
        jjj = jj(j);
        if (A(jjj,i))^2 >= aiieps*A(jjj,jjj)
            N(ico,i)=jjj;
            ico=ico+1;
        else
            if filter
                jjss = [jjss jjj];
  %              Afil(jjj,i)=0.0;
                ssum = ssum + A(jjj,i);
            end
        end
    end
    if filter
        Afil(i,jjss)=0;
        Afil(i,i) = A(i,i) - ssum;
    end
end

% Step 1 of the aggregation algorithm

j=1; R = [1:n];
Kind = ones(n,1);
[koi,koj,Ni] = find(N(:,1));
C{1}=Ni';
Cpic(1)=1;
RC = C{1};
```

```
kkhelp=1;
while 1==1
    Kind(C{j}) = 0;
    [koi,koj,Ni] = find(N(:,C{j}));
    Kind(Ni)=0;
    kkhelp=find(Kind,1,'first');
  if numel(kkhelp)==0, break, end
    [koi,koj,Ni] = find(N(:,kkhelp));
    j=j+1;
    C{j}=Ni';RC=[RC C{j}];
    Cpic(j)=kkhelp;
end
R = setdiff(R,RC);
lele=length(C);

RR=R;

% Step 2 of the aggregation algorithm (recommended to skip at the moment)

%Ctilde = C;
%for i=1:length(R)
%    ir = RR(i);
%    [koi,koj,Ni] = find(N(:,ir));aa=sparse(n,1);aa(Ni)=1;
%    for k=1:length(C)
%        kr=k;
%        bb=sparse(n,1);bb(Ctilde{kr})=1;
%        if max(aa+bb) > 1
%            C{kr} = [C{kr} ir];
%            R = setdiff(R,ir);
%            break
%        end
%    end
%end

% Step 3 of the aggregation algorithm

if ~isempty(R)
    while ~isempty(R)
        ir=R(1);
        [koi,koj,Ni] = find(N(:,ir));
        j = j+1;
        C{j} = intersect(R,Ni);
        Cpic(j)=ir;
        R = setdiff(R,C{j});
    end
end

% now we can create the tentative prolongation operator Ptilde
Ptil = sparse(n,length(C));
```

```
for i=1:length(C)
    Ptil(C{i},i) = 1;
end

% the definite prolongation is obtained by smoothing
omega = 0.6666;%omega=0.5;
if filter
    P = (speye(n,n) - omega.*inv(diag(diag(A)))*Afil)*Ptil;
else
    P = (speye(n,n) - omega.*inv(diag(diag(A)))*A)*Ptil;
end

% the rest is to get a picture of the coarse-level nodes (so far this only
% works well for two levels)

if(picture)

hold on;
axis equal;
for node=1:n
    plot(p(1,node),p(2,node),'.','MarkerSize',4,'color','black')
end
%for i=lele+1:length(RR)
%    node=RR(i);
%    plot(p(1,node),p(2,node),'.','Linewidth',2,'color','cyan')
%end
for i=1:lele
    node=Cpic(i);
    plot(p(1,node),p(2,node),'.','MarkerSize',12,'color','red')
end
for i=lele+1:length(C)
    node=Cpic(i);
    plot(p(1,node),p(2,node),'.','MarkerSize',12,'color','blue')
end

end
```

Fonction sor.m

```
function [x, error, iter, flag]  = sor(A, x, b, w, max_it, tol)

%  -- Iterative template routine --
%     Univ. of Tennessee and Oak Ridge National Laboratory
%     October 1, 1993
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
%     Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
```

```
%     1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%
% [x, error, iter, flag]  = sor(A, x, b, w, max_it, tol)
%
% sor.m solves the linear system Ax=b using the
% Successive Over-Relaxation Method (Gauss-Seidel method when omega = 1 ).
%
% input   A        REAL matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         w        REAL relaxation scalar
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
%
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it

  flag = 0;                                    % initialization
  iter = 0;

 % bnrm2 = norm( b );
 % if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end

 % r = b - A*x;
 % error = norm( r ) / bnrm2;
 % if ( error < tol ) return, end

  [ M, N, b ] = split( A, b, w, 2 );           % matrix splitting

  for iter = 1:max_it                          % begin iteration

     x_1 = x;
     x   = M \ ( N*x + b );                    % update approximation

  %   error = norm( x - x_1 ) / norm( x );     % compute error
  %   if ( error <= tol ), break, end          % check convergence

  end
  b = b / w;                                   % restore rhs

  %if ( error > tol ) flag = 1; end;           % no convergence

% END sor.m
```

Fonction split.m

```
function [ M, N, b ] = split( A, b, w, flag )
%
% function [ M, N, b ] = split( A, b, w, flag )
%
% split.m sets up the matrix splitting for the stationary
% iterative methods: jacobi and sor (gauss-seidel when w = 1.0 )
%
% input   A        DOUBLE PRECISION matrix
%         b        DOUBLE PRECISION right hand side vector (for SOR)
%         w        DOUBLE PRECISION relaxation scalar
%         flag     INTEGER flag for method: 1 = jacobi
%                                           2 = sor
%
% output  M        DOUBLE PRECISION matrix
%         N        DOUBLE PRECISION matrix such that A = M - N
%         b        DOUBLE PRECISION rhs vector ( altered for SOR )

  [m,n] = size( A );

  if ( flag == 1 ),                    % jacobi splitting

     M = diag(diag(A));
     N = diag(diag(A)) - A;

  elseif ( flag == 2 ),                % sor/gauss-seidel splitting

%     b = w * b;
%     M =  w * tril( A, -1 ) + diag(diag( A ));
%     N = -w * triu( A,  1 ) + ( 1.0 - w ) * diag(diag( A ));
     M =  tril( A, -1 ) + diag(diag( A ));
     N = -triu( A,  1 );

  end;

% END split.m
```

Algorithme de [4] implémenté lors de ce mémoire

Le programme pour résoudre les systèmes linèaires, $Au = f$, issus de la discrétisation de problèmes continus, par la méthode [4], se lance avec la commande suivante :

```
multigrilles(Ah,fh,Lmax,graphe,coord)
```

dont les arguments sont :

- Ah : matrice A du niveau le plus fin,
- fh : membre de droite de l'équation,

- $Lmax > 1$: nombre de niveaux,
- *graphe* : 0 pas de graphe, 1 pour faire un graphe,
- *coord* : contient les coordonnées des points pour le dessin.

Fonction multigrilles.m

```
function vh = multigrilles(Ah,fh,Lmax,graphe,coord)
%=========================================================================
% multigrilles : fonction qui resoud un systeme lineaire, Au = f,
%        issu de la discretisation d un probleme continu via
%        une methode d elements finis ou de differences finies.
%        La resolution se fait par une methode des multigrilles algebriques
%        classique decrite dans :
%        "A Multigrid Tutorial, Second Edition" de
%        Briggs, W. L. and Henson, V. E. and McCormick,S. F.
%
% hypothese : A est une matrice symetrique, definie positive et
%        si possible une M-matrice (a_ii > 0 et a_ij < 0 pour j >< i)
%
% INPUT  : - Ah : matrice A du niveau le plus fin
%          - fh : membre de droite de l equation
%          - Lmax : nombre de niveau maximum
%          - graphe : vaut 1 si on desire un dessin
%                          0 sinon
%          - coord : contient les coordonnees des points pour le dessin
%
% OUTPUT : - vh : solution approchee de u
%
% AUTEUR : Sylvie Detournay
%
% modifie le 1703
% modif 1305
%=========================================================================

%initialisation
sortie = 1; %1 pour sortie sur ecran
schema = 1; %pour afficher les 2 etapes de coloriage sur les graphiques
tol = 1e-4; %niveau de tolerance pour l erreur
kMax = 1000; %nombre max d iterations

%vecteur de depart
vh = zeros(size(Ah,1),1);

%Phase de PREPARATION
tic
fprintf(sortie, 'Preparation : \n');
EnsA{1}= Ah;
```

```
fprintf(sortie, 'Dimension au niveau %d : %d \n',1,size(EnsA{1},1));

for i = 2 : Lmax
    if ~graphe
       EnsI{i-1} = samg(EnsA{i-1});
    else
       [EnsI{i-1}, EnsC{i-1}, EnsC1{i-1}] = samg(EnsA{i-1});
       %creation des dessins
       dessin(coord, EnsC{i-1}, EnsC1{i-1}, schema, i-1);
       coord = coord(:,EnsC{i-1});
    end
    EnsA{i} = EnsI{i-1}'*EnsA{i-1}*EnsI{i-1}; %'
    fprintf(sortie, 'Dimension au niveau %d : %d \n',i,size(EnsA{i},1));
end
t1 = toc

%Phase de RESOLUTION
tic
fprintf(sortie, 'Resolution : \n');
k=0;
norel = 1.0;
normfh = norm(fh);
while norel > tol & k < kMax
    k = k+1;
    vh = vcycleh(vh,fh,Lmax-1,EnsI,EnsA,Lmax);
    norel = norm(Ah*vh-fh)/normfh;
end
fprintf(sortie, 'Nombre d iterations : %d \n',k);
Vconv = norel^(1/k);
fprintf(sortie, 'Vitesse de convergence : %d ',Vconv);
t2 = toc


%-------------------------------------------------------------------
%            Fonction samg
%-------------------------------------------------------------------



function [I, C, C1] = samg(A);
%===================================================================
% samg : fonction qui sert a calculer un operateur d interpolation
%        d un niveau pour le schema multigrille
%        cette fonction rend aussi les ensembles C1 et C pour
%        la construction d un eventuel graphique
%
% INPUT  : - A : matrice A du systeme a resoudre
%
% OUTPUT : - I  : matrice d interpolation
```

```
%          - C1 : ensemble de points de la grille grossiere
%                 obtenu via le 1er schéma de coloriage
%          - C  : ensemble de points de la grille grossiere
%                 obtenu via le 2eme schéma de coloriage
%
% AUTEUR : Sylvie Detournay
%
% MODIF : 1801
% MODIF : 0105 (commentaires)
%=========================================================================

%n = taille de la matrice
n = size(A,1);

%nmax est le nombre maximal d elements sur une ligne - elt diag
nmax = 0;
for i = 1 : n
     if nnz(A(:,i))> nmax
          nmax = nnz(A(:,i));
     end
end
nmax = nmax - 1;

%calcul des S_i (Si) et des S_i transposes (Stri)
[Si, Stri] = S_i(A,n,nmax);

%1er schema qui rend les premiers ensembles C (C1) et F (F1)
[C1, F1] = schema_de_coloriage11(n,Si,Stri,nmax);
clear Stri;

%2eme schema qui rend les ensembles C et F definitifs
[C, F] = schema_de_coloriage20(C1,F1,A,n,Si);

%interpolation
I = interpolation(A,C,F,n,Si);



%-------------------------------------------------------------------
%            Fonction vcycleh
%-------------------------------------------------------------------

function vh = vcycleh(vh,fh,niv,EnsI,EnsA,Lmax);
%=========================================================================
% vcycleh : fonction recurssive du schema V-cycle des multigrilles
%         qui calcule la solution du systeme Au = f
%
% INPUT  : - vh : approximation de la solution au niveau h
%          - Ah : matrice A au niveau h
```

```
%          - fh : membre de droite au niv h
%          - niv : niveau auquel on se trouve
%          - EnsI : ensemble contenant les operateurs d interpolation
%                   pour chaque niveau
%          - EnsA : ensemble contenant les operateurs grossiers
%                   pour chaque niveau
%          - Lmax : nombre de niveau restant

%
% OUTPUT : - vh : approximation de depart pour le niveau 2h
%
% AUTEUR : Sylvie Detournay
%
% modifie le 1102 (parametres en argument)
% MODIF 0105 (commentaires)
%=====================================================================

%parametre pour le Vcycle
nu1 = 1; % nombre de relaxations initiales (pre-smoothing)
nu2 = 2; % nombre de relaxations apres le transfere grille (post-smoothing)

h = Lmax -niv;

%decomposition de la matrice A = D - U - L
U = -triu(EnsA{h},1);
L = -tril(EnsA{h},-1);
D = diag(diag(EnsA{h}));

%step 1 : relaxer nu1 fois
for i = 1 : nu1
    vh = (D-L) \ (U*vh + fh);   %Seidel
end

%step 2
if niv ~= 0

     f2h = EnsI{h}'*(fh-EnsA{h}*vh); %'
     v2h = zeros(size(f2h,1),1);
     v2h = vcycleh(v2h,f2h,niv-1,EnsI,EnsA,Lmax);

     %step 3
     vh = vh + EnsI{h}*v2h;

     %step 4 : relaxer nu1 fois
     for i = 1 : nu2
          vh = (D-L) \ (U*vh + fh);   %Seidel
     end

else
```

```
    %step 4 : relaxer nu1 fois
    %for i = 1 : nu2
    %   vh = (D-L) \ (U*vh + fh);   %Seidel
    %end
    %resoudre exactement

    vh = EnsA{h} \ fh;

end
```

Cette fonction fait appel aux sous-routines suivantes :

Fonction S_i.m

```
function [S, St] = S_i(A,n,nmax);
%=================================================================
% S_i : fonction qui construit les ensembles des points qui influencent
%       fortement i et des points qui dependent fortement de i
%       pour chaque point i de la grille fine
%
% INPUT  : - A : matrice A
%          - n : dimension de A
%          - nmax : le nombre maximal d elements non nuls par ligne
%
% OUTPUT : - S
%            où S(i,:) est l ensemble des points qui influencent
%                  fortement i
%          - St
%            où St(k,:) est l ensemble des points qui dependent
%                  fortement k
%
% AUTEUR : Sylvie Detournay
%
% MODIF : 1603
% MODIF : 0105 (ajout commentaires)
%=================================================================

%parametre
theta = 0.25;

%initialisation
S = zeros(n,nmax);
St = zeros(n,nmax);

for i = 1 : n
    %prendre l element maximal de la ligne de A (maxi)
    indices = find(A(:,i)); %indices des elmts non null de A(i,:)
    maxi = 0;
```

```
    for m = indices'
        if abs(A(i,m)) > maxi & m~=i
           maxi = abs(A(i,m));
        end
    end
    k = 1;
    %pour tout les points j >< i, mettre j dans S_i
    %s il influence fortement i
    for j = indices'
        if j ~= i & abs(A(i,j)) >= theta*maxi
           S(i,k) = j;
           %si j influence fortement i alors i depend fortement de j
           St(j,nnz(St(j,:))+1) = i;
           k = k+1;
        end
    end
end
```

Fonction schema de coloriage11.m

```
function [C, F] = schema_de_coloriage11(n,Si,Stri,nmax);
%=====================================================================
% schema_de_coloriage11 : fonction qui a pour but de creer une
%        premiere partition de la grille fine en deux ensembles
%        C et F
%
% INPUT  : - n : dimension de A
%          - Si : matrice telle que les S_i(i,:) est l ensemble des
%                 points qui influencent fortement i
%          - Stri : matrice telle que les S_i(i,:) est l ensemble des
%                 points qui dependent fortement i
%          - nmax : le nombre maximal d elements non nuls par ligne
%
% OUTPUT : - C ensemble des point de la grille grossière
%          - F ensemble des point de la grille fine
%
% AUTEUR : Sylvie Detournay
%
% modifier : 1403 ptit modif gain 0.1 de sec
% MODIF : 0305 (ajout commentaires)
%=====================================================================

%initialisation
C = [];
F = [];
lambda = zeros(n,1);

%lambda(i) = #(S_itransposé)
for i = 1 : n
```

```
     lambda(i) = nnz(Stri(i,:));
end

%initialisation
c = 1;
f = 1;

%tant que tous les points ne sont pas soit ds C soit ds F
while length(C) + length(F) ~= n %modif 0302

     %on recherche le point lbdm avec un lambda maximal
     [lambdamax,lbdm] = max(lambda);

     %si le point n est pas déjà classé on le met ds C
     if lambda(lbdm) > 0
          C(c) = lbdm;
          lambda(lbdm) = -1;
          c = c + 1;
     else
          lambda(lbdm) = -1;
          break;
     end

     %Tous les points appartenant à S_lbdm transposé sont mis ds F
     for j = 1 : nmax
          %Stri(:,j) termine par une serie de 0
          if Stri(lbdm,j) == 0
               break;
          end

          %si le point n est pas deja classe on le met ds F
          if lambda(Stri(lbdm,j)) > 0
               %le point Stri(lbdm,j) fortement influence par lbdm est mis ds F
               F(f) = Stri(lbdm,j);
               %pour ne pas le reprendre
               lambda(F(f)) = -1;

               for k = 1 : nmax
                    if Si(F(f),k) == 0
                         break;
                    end
                    if Si(F(f),k) ~= lbdm
                         %les lambdas des points non classes qui influencent
                         %fortement j sont incrementés de 1
                         if lambda(Si(F(f),k)) > 0
                              lambda(Si(F(f),k)) = lambda(Si(F(f),k)) + 1;
                         end
                    end
               end
```

```
            f = f+1;
            end
        end
end
```

Fonction schema de coloriage20.m

```
function [C, F] = schema_de_coloriage20(C,F,A,n,Si);
%=====================================================================
% schema_de_coloriage20 : fonction qui a pour but de creer les
%        ensembles C et F et a garantir l existence de
%        l operateur d interpolation (verification de H2)
%
% INPUT  : - C : ensemble des points de la grille grossiere,
%                obtenu apres le premier schema de coloriage
%          - F : ensemble des points de la grille fine,
%                obtenu apres le premier schema de coloriage
%          - n : dimension de A
%          - Si : matrice telle que les S_i(i,:) est l ensemble des
%                 points qui influencent fortement i
%
% OUTPUT : - C ensemble des points de la grille grossiere
%          - F ensemble des points de la grille fine
%
% AUTEUR : Sylvie Detournay
%
% MODIF 1801 gd modif
% MODIF 3004
% MODIF : 0305 (ajout commentaires)
%=====================================================================

%initialisation
% - trouve : vaut 1 si H2 est verifiee
%            pour le point en cours d analyse
% - candidat : vaut 0 s il n y a pas encore de candidat
%              vaut 1 sinon
% - icand : indice du candidat dans F
trouve = 0;
candidat = 0;
icand =0;

%tri des elements de F
F = sort(F);

%initialisation
i = 1;
%pour ts les points de F
while i <= length(F)
```

```
%calcul du voisinage Ci :
%allocation de memoire
Ci = [];
%initialisation des compteurs
c = 1;
%Parcours des elements de la grille gross C
for s = 1 : nnz(Si(F(i),:))
     %si le point Si(F(i),s)) est dans C
     if ~isempty((find(C == Si(F(i),s))))
          Ci(c) = Si(F(i),s);
          c = c + 1;
     end
end
c = 0;


%pour ts les points qui influencent fortement le point F(i),
%on regarde si H1 est verifie
for j = 1 : nnz(Si(F(i),:))

     %si le point Si(F(i),j)) n appartienr pas a Ci
     if isempty(find(Ci == Si(F(i),j)))

          %On regarde s il existe k appartenant Ci_F(i) tq
          %k appartient S_Si(F(i),j)
          for k = 1 : length(Ci)
               %Si Ci(k) appartient a S_Si(F(i),j)
               if ~isempty(find(Si(Si(F(i),j),:) == Ci(k)))
                    trouve = 1;
                    break;
               end
          end

          if trouve ~= 1
               %il existe un Si(Fi,j) ds Si(Fi,:) qui n appartient pas a Ci et
               %qui n est pas fortement influence
               %par un point de Ci
               %H2 n est pas verifie

               %si c est le premier candidat de Si(Fi,:)
               if candidat == 0
                    C(length(C)+1) = Si(F(i),j);
                    candidat = Si(F(i),j);
                    Ci(length(Ci)+1) = Si(F(i),j);

                    %sinon on rajoute F(i) car le point F(i) est influence
                    %par au moins 2points de F
               else
```

```
                        %on ecrase le candidat
                        C(length(C)) = F(i);
                        F(i) = [];
                        candidat = 0;
                        i = i -1; %car on a retire un point de F
                        break;
                    end
                end

            end %fin le point n est pas ds Ci

            trouve = 0;
        end %fin de ts les points qui influencent fortement le point F(i)

        %on n a pas trouve d autre candidat on doit rajouter celui-ci
        if candidat ~= 0
            icand = find(F == candidat);
            F(icand) = [];
        end

        %on passe au suivant point de F si on a pas retirer un point de F
        %qui se trouvait avant i
        if candidat == 0 | icand > i
            i = i + 1;
        end

        candidat = 0;
    end %fin de tous les point de F

    C = sort(C);
    F = sort(F);
```

Fonction interpolation.m

```
function W = interpolation(A,C,F,n,Si);
%=================================================================
% interpolation : fonction qui a pour but de construire
%        l operateur d interpolation
%
% INPUT  : - A : matrice du systeme
%          - C : ensemble des points de la grille grossière trié
%          - F : ensemble des points de la grille fine trié
%          - n : dimension de A
%          - Si : vecteur des points qui influencent fortement i
%
% OUTPUT : - W : matrice d interpolation en sparse
%
% AUTEUR : Sylvie Detournay
%
```

```
% MODIF 1603
% MODIF 3004
% MODIF : 0305 (ajout commentaires)
%=================================================================

%initialisation de la matrice d interpolation matrice creuse contenant des zeros
W = sparse(n,length(C));

% initialisation des compteurs
f = 1;
c = 1;

for i = 1 : n
     % si est un point de la grille fine
    if i == F(f)

          %Calcul du voisinage de i :
          %          - Ci : ensemble des voisins de la grille grossiere qui
          %                 influencent fortement i
          %          - Ds : ensemble des voisins de la grille fine qui
          %                 influencent fortement i
          %          - Dw : ensemble des voisins de la grille fine et
          %                 grossiere qui n influencent pas fortement i

          %les points appartenant au voisinage de i
          indices = find(A(:,i));

          [Ci, Ds, Dw, indiceCi] = voisinage(C,F,indices',n,Si(i,:),i);

          %calcul du denominateur de w (independant de j)
          den = A(i,i);
          for k = 1 : length(Dw)
               den = den + A(i,Dw(k));
          end

          %calcul du wij
          for j = 1 : length(Ci)

               %calcul du numerateur
               num = A(i,Ci(j));
               for m = 1 : length(Ds)
                    dnum = 0;
                    for l = 1 : length(Ci)
                         dnum = dnum + A(Ds(m),Ci(l));
                    end
                    if dnum == 0
                         fprintf(1,'Erreur division par 0!');
                    else
                         num = num + (A(i,Ds(m))*A(Ds(m),Ci(j)))/dnum;
```

```
                    end
                end

                % calcul de wij
                w = - num/den;

                % place les wij à la bonne place
                W(i,indiceCi(j)) = w;

            end

            if f ~= length(F)
                f = f + 1;
            end

        % si i est un point de la grille grossière
        else
            W(i,c) = 1;
            if c ~= length(C)
                c = c + 1;
            end
        end

end



%-------------------------------------------------------------


function [Ci, Ds, Dw, indiceCi] = voisinage(C,F,indices,n,Si,i);
%=================================================================
% voisinage : fonction qui calcule les voisinages du point i
%
% INPUT  : - n : dimension de A
%          - Si : vecteur ligne des points qui influencent fortement i
%          - C : ensemble des points de la grille grossière trié (colonne)
%          - F : ensemble des points de la grille fine trié (colonne)
%          - i : indice du point dont on veut calculer le voisinage
%          - indices : les points appartenant au voisinage de i
%
% OUTPUT : - Ci : ensemble des voisins de la grille grossiere qui
%                 influencent fortement i
%          - Ds : ensemble des voisins de la grille fine qui
%                 influencent fortement i
%          - Dw : ensemble des voisins de la grille fine et
%                 grossiere qui n influencent pas fortement i
%          - indiceCi : indices des points Ci dans C
%
```

```
% MODIF : 1801 gd modif find gain de temps ;)
% MODIF : 1403 ajout indiceCi pt gain de temps
% MODIF : 0105 (ajout commentaires)
%=====================================================================

% allocation de memoire
Ci = [];
indiceCi = [];
Ds = [];
Dw = [];

%initialisation des compteurs
c = 1;
f = 1;
w = 1;


for j = indices
     if j ~= i
          %si j appartient pas a Si
          if isempty(find(Si == j))
               Dw(w) = j;
               w = w + 1;
          else
               indicec = find(C == j); %indice du point Ci dans C
               %si j appartient C et appartient Si
               if ~isempty(indicec)
                    Ci(c) = j;
                    indiceCi(c) = indicec;
                    c = c + 1;
                    %si j appartient F et appartient Si
               else
                    Ds(f) = j;
                    f = f + 1;
               end
          end
     end
end
```

Fonction dessin.m

```
function y = dessin(noeud, C, C1, schema, k)
%---------------------------------------------------------------------
% fonction qui dessine les differentes selections de points
%
% INPUT : - noeud : coordonees des noeuds de la grille de depart
%         - C : ensemble des points de la grille grossiere
%               obtenus a la fin des deux schemas de
%               coloriage
```

```
%         - C1 : ensemble des points obtenus apres le 1er schema
%         - schema : vaut * 1 si on desire dessiner les selections des
%                            2 schema, alors C1 est non vide
%                          * 2 sinon, alors C1 peut etre vide
%         - k : numero de la fenetre
%
% OUTPUT : - y : 0 si tous s est bien passe
%
% AUTEUR : Sylvie Detournay
%
% modif 0105
%-------------------------------------------------------------------

figure(k);

indices = 1 : size(noeud,2);
indices(C) = [];
%noeud de F
plot(noeud(1,indices),noeud(2,indices),'.','MarkerSize',4,'color','black');

if schema
     hold on;
     axis equal;
     %noeud de C obtenu apres 1er schema
     plot(noeud(1,C1),noeud(2,C1),'.','MarkerSize',10,'color','red');
     hold on;
     axis equal;
     ind = setdiff(C,C1);
     %point de C obtenu apres 2eme schema
     plot(noeud(1,ind),noeud(2,ind),'.','MarkerSize',10,'color','blue');
else
     hold on;
     axis equal;
     %noeud de C
     plot(noeud(1,C),noeud(2,C),'.','MarkerSize',10,'color','blue');
end

y = 0;
```

Autres programmes

Nous présentons ici quelques routines élaborées et utilisées dans le cadre de ce mémoire.

Fonction profil.m

Voici le code pour créer un profil de performance :

```
function profildeperf(S)
```

```
%=========================================================================
% profildeperf : fonction qui trace un profil de performance a partir
%        de la matrice S, ou S(i,j) est la performance de l algorithme i
%        pour resoudre le probleme j
%
% AUTEUR : Sylvie Detournay
%
% modifie le 2405
%=========================================================================

nb = size(S,2);

st = min(S(1,:),S(2,:));

sigma = 1 : 30;
nsigma = length(sigma);

for i = 1 : nsigma
     p1(i) = sum(S(1,:)<=(sigma(i).*st))/nb;
     p2(i) = sum(S(2,:)<=(sigma(i).*st))/nb;
end

plot(sigma,p1,'r');
hold on;
plot(sigma,p2);
```

Formation de problèmes liés à des stencils

Fonction stencilAni.m

```
function stencilAni(n)
%=========================================================================
% stencilAni : fonction pour la creation de problemes a partir de stencil
%          s = [0 -epsi 0; -1 2+2*epsi -1; 0 -epsi 0]
%          pour 1 grille a 2 dim avec n^2 noeuds
%          (le stencil est pris du livre de Briggs & co)
%
% auteur : Sylvie Detournay
%
% dernière modif : 2305
%=========================================================================

epsi = 1e-9;
s = [0 -epsi 0; -1 2+2*epsi -1; 0 -epsi 0]; %matrice stencil

[A,nodes] = UnStencil(n,s);

%membre de droite de l equation Au=f
f = ones(size(A,1),1);
```

```
save(strcat('problemStencAni',int2str(n^2),'.mat'),'A','f','nodes');
```

Fonction stencilBriggs.m

```
function stencilBriggs(n)
%=========================================================================
% stencilBriggs : fonction pour la creation de problemes a partir de stencil
%          s = [- 1 -1 -1; -1 8 -1; -1 -1 -1]
%          pour 1 grille à 2 dim avec n^2 noeuds
%          (le stencil est pris du livre de Briggs & co)
%
% auteur : Sylvie Detournay
%
% dernière modif : 2305
%=========================================================================

s = [- 1 -1 -1; -1 8 -1; -1 -1 -1]; %matrice stencil

[A,nodes] = UnStencil(n,s);

%membre de droite de l equation Au=f
f = ones(size(A,1),1);

save(strcat('problemStencBriggs',int2str(n^2),'.mat'),'A','f','nodes');
```

Fonction stencilNotay.m

```
function stencilNotay(n)
%=========================================================================
% stencilNotay : fonction pour la creation de problemes a partir de stencil
%          s = [- 101 -398 -101; 196 808 196; -101 -398 -101]
%          pour 1 grille à 2 dim avec n^2 noeuds
%          (le stencil est pris de la presentation de Yvan Notay)
%
% auteur : Sylvie Detournay
%
% dernière modif : 2305
%=========================================================================

s = [- 101 -398 -101; 196 808 196; -101 -398 -101]; %matrice stencil

[A,nodes] = UnStencil(n,s);

%membre de droite de l equation Au=f
f = ones(size(A,1),1);

save(strcat('problemStencNotay',int2str(n^2),'.mat'),'A','f','nodes');
```

Fonction mixte.m

```
function mixte(n)
%=========================================================================
% mixte : fonction pour la creation de problemes a partir des stencils :
%                Dxx = [0 0 0; -1 2 -1; 0 0 0]
%                Dyy = [0 -1 0; 0 2 0; 0 -1 0]
%                Dxy = 1.5*[-1 1 0; 1 -2 1; 0 1 -1]
%          pour 1 grille à 2 dim avec n^2*4 noeuds
%          (le stencil est donné par Stuben)
%
% auteur : Sylvie Detournay
%
% dernière modif : 2305
%=========================================================================

%probleme spec

Dxx = UnStencil(n,[0 0 0; -1 2 -1; 0 0 0]);

Dyy = UnStencil(n,[0 -1 0; 0 2 0; 0 -1 0]);

Dxy = UnStencil(n,[-1 1 0; 1 -2 1; 0 1 -1]);
Dxy = 0.5.*Dxy;

% forme du probleme :
%
%   G3 G4    -> A = A1              nodes = G1
%   G1 G2              A2                   G2
%                         A3                G3
%                            A4             G4

n1 =n;
n = n^2;

%creation de la matrice
A = zeros(n*4,n*4);

A1 = Dxx + Dyy;
A2 = 1000.*Dxx + Dyy;
A3 = Dxx + 1000.*Dyy;
A4 = Dxx + Dyy + 2.*Dxy;

A(1:n,1:n) = A1;
A(n+1:2*n,n+1:2*n) = A2;
A((2*n)+1:3*n,(2*n)+1:3*n) = A3;
A((3*n)+1:4*n,(3*n)+1:4*n) = A4;

nodes = zeros(2,4*n);
```

```
for k = 1 : n1
     nodes(1, ((k-1)*n1 + 1) :k*n1) = 1 : n1;
     nodes(2, ((k-1)*n1 + 1) :k*n1) = k.* ones(1,n1);
end

for k = 1 : n1
     nodes(1, ((k-1)*n1 + 1)+n :k*n1+n) = [1 : n1]+7;
     nodes(2, ((k-1)*n1 + 1)+n :k*n1+n) = k.* ones(1,n1);
end

for k = 1 : n1
     nodes(1, ((k-1)*n1 + 1)+2*n :k*n1+2*n) = 1 : n1;
     nodes(2, ((k-1)*n1 + 1)+2*n :k*n1+2*n) = k.* ones(1,n1)+7;
end

for k = 1 : n1
     nodes(1, ((k-1)*n1 + 1)+3*n :k*n1+3*n) = [1 : n1]+7;
     nodes(2, ((k-1)*n1 + 1)+3*n :k*n1+3*n) = k.* ones(1,n1)+7;
end

%membre de droite de l equation Au=f
f = ones(size(A,1),1);

save(strcat('problemStencMixte',int2str(4*n),'.mat'),'A','f','nodes');
```

Readme

Nous reprenons ici le fichier Readme du package associé à ce mémoire.

```
Date : 23 mai 2006
Auteur : Sylvie Detournay
Description du package associé au mémoire :

"Comparaison de méthodes de multigrilles algébriques
      (type classique vs type agrégation)
 pour la résolution de systèmes linéaires issus
 de la discrétisation de problèmes continus"


Les programmes disponibles dans ce package nécessite
MATLAB version7 ou supérieur.

Contenu du package :
====================


Ce package contient les dossiers suivant :
```

- PackageAMG
  - Problemes
  - Test
  - ProfilPerf
    - ProblemesPerf

Le dossier principal PackageAMG contient, en plus des sous-dossiers, tous les fichiers servants aux deux programmes étudiés dans le mémoire dont les routines principales sont : amg.m et multigrilles.m.

Le sous dossier Problemes contient les pricipaux exemples étudiés dans le mémoire (problèmes générés par des stencils ou ceux fournis par Michal Kocvara) et certains codes pour générer certains de ces problèmes.

Le dossier Test sert à contenir les résultats obtenus lors de l'application des programmes.

Finalement le dossier ProfilPerf contient tous les fichiers nécessaires à la formation du profil de performance contenu dans le mémoire. Il contient aussi les fichiers principaux du dossier PackageAMG avec quelques modifications et les fichiers liés à la création du profil de performance.

Commandes diponibles :
==========

A partir du dossier principal :

Pour charger un problème :

```
> load Problemes/nom_du_probleme.mat
```

ou pour un des problèmes du dossier ProfilPerf :

```
> load ProfilPerf/ProblemesPerf/nom_du_probleme.mat
```

Ligne de commande pour lancer l'un des programmes :

pour l'algorithme de type agrégation :

```
> amg(levels,A,F,pred,picture,filter)
```

auteur : Michal Kocvara

```
  amg...function for the solution of the linear system Ax=F with a
  symmetric positive definite matrix A by the algebraic multigrid method

  levels > 1 : nombre de niveaux souhaité
  A : matrice du système
  f : membre de droite du système
  pred : matrice contenant les coordonnées des noeuds de la grille
        la plus fine (nécessaire uniquement si picture = 1)
  picture : représentation graphique des noeuds du niveau
            grossier (1 -> OUI, 0 -> NON)
  filter : filtre la matrice du système lors de la relaxation de
        l'opérateur d'interpolation (1 -> OUI, 0 -> NON).


pour l'algorithme de type classique

  > multigrilles(Ah,fh,Lmax,graphe,coord);


  multigrilles : fonction qui résoud un systeme lineaire, Au = f,
        issu de la discrétisation d'un problème continu via
        une méthode d'éléments finis ou de différences finies.
        La résolution se fait par une méthode des multigrilles algébriques
        classique décrite dans :
        "A Multigrid Tutorial, Second Edition" de
        Briggs, W. L. and Henson, V. E. and McCormick,S. F.

  Ah : matrice A du niveau le plus fin
  fh : membre de droite de l equation
  Lmax : nombre de niveau maximum > 1
  graphe : 0 pas de graphe, 1 pour faire un graphe
  coord : contient les coordonnees des points pour le dessin


A partir du dossier Problemes :
-------------------------------

  pour créer des problèmes tests à partir des stencils étudiés

  > stencilAni(n)

  stencil Ani : fonction pour la création de problèmes à partir du stencil
       s = [0 -epsi 0; -1 2+2*epsi -1; 0 -epsi 0]
       pour 1 grille à 2 dim avec n^2 noeuds
       (le stencil est pris du livre de Briggs & co)


  > stencilBriggs(n)
```

```
stencilBriggs : fonction pour la création de problèmes à partir de stencil
     s = [- 1 -1 -1; -1 8 -1; -1 -1 -1]
     pour 1 grille à 2 dim avec n^2 noeuds
     (le stencil est pris du livre de Briggs & co)


> stencilNotay(n)

stencilNotay : fonction pour la création de problèmes à partir du stencil
     s = [- 101 -398 -101; 196 808 196; -101 -398 -101]
     pour 1 grille à 2 dim avec n^2 noeuds
     (le stencil est pris de la présentation de Yvan Notay)


> mixte(n)

mixte : fonction pour la création de problèmes à partir du stencils :
     Dxx = [0 0 0; -1 2 -1; 0 0 0]
     Dyy = [0 -1 0; 0 2 0; 0 -1 0]
     Dxy = 1.5*[-1 1 0; 1 -2 1; 0 1 -1]
     pour 1 grille à 2 dim avec n^2*4 noeuds
     (le stencil est donné par Stuben)


> UnStencil(n,s)

UnStencil : fonction pour la création de matrices à partir de stencil
      à max 9 points pour 1 grille à 2 dimensions contenant n^2 noeuds


A partir du dossier ProfilPerf :
--------------------------------

les commandes load pour le dossier ProblemesPerf sont encore disponibles,
ainsi que les programmes principaux : amg.m et multigrilles.m

pour créer le profil de performance

> profil(nb)

profil : fonction qui crée des profils de performance pour les
     nb premiers problèmes contenus dans le dossier ProblemesInternet

nb : nombre de problèmes à considérer

à chaque profil de performance un fichier récapulatif des données des
problèmes et des variables de performance est généré.
```

```
  > profildeperf(nb)

  profildeperf : fonction qui trace un profil de performance à partir
       de la matrice S, ou S(i,j) est la performance de l algorithme i
       pour resoudre le probleme j


Liste des problèmes disponibles :
---------------------------------

   dans le dossier Problemes :

   problemMichal126.mat
   problemMichal1668.mat
   problemMichal25296.mat
   problemMichal6440.mat
   problemStencAni10000.mat
   problemStencAni122500.mat
   problemStencAni22500.mat
   problemStencAni2500.mat
   problemStencAni40000.mat
   problemStencAni6400.mat
   problemStencAni90000.mat
   problemStencBriggs10000.mat
   problemStencBriggs22500.mat
   problemStencBriggs2500.mat
   problemStencBriggs40000.mat
   problemStencBriggs6400.mat
   problemStencMixte10000.mat
   problemStencMixte2500.mat
   problemStencMixte25600.mat
   problemStencMixte40000.mat
   problemStencMixte6400.mat
   problemStencNotay10000.mat
   problemStencNotay122500.mat
   problemStencNotay202500.mat
   problemStencNotay22500.mat
   problemStencNotay2500.mat
   problemStencNotay40000.mat
   problemStencNotay490000.mat
   problemStencNotay62500.mat
   problemStencNotay6400.mat
   problemStencNotay90000.mat
   problemStencNotay900.mat

   dans le dossier ProfilPerf/ProblemesPerf :

   problemRandj.mat     pour 1 <= j <= 33
```

Bibliographie

[1] O. Axelsson and V. A. Barker. Finite Element Solution of Boundary Value Problems. Academic Press, Orlando, 1984.

[2] Mathieu Bastin. Note du cours d'Anne Lemaître : Méthodes numériques des équations aux dérivées partielles. Namur, FUNDP, 2005.

[3] M. Brezina. Robust Iterative Methods on Unstructured Meshes. PhD thesis, University of Colorado at Denver, 1997.

[4] William L. Briggs, Van Emden Henson, and Steve F. McCormick. A Multigrid Tutorial, Second Edition. SIAM, Philadelphia, 2000.

[5] T. Davis. University of Florida Sparse Matrix Collection. site internet : http ://www.cise.ufl.edu/research/sparse/matrices, 1996-1997-1998.

[6] Sylvie Detournay. Note du cours d'Annick Sartenaer : Algèbre linéaire numérique. Namur, FUNDP, 2005.

[7] Alexandre Ern and Jean-Luc Guermond. Theory and Practice of Finite Elements. Springer, New York, 2004.

[8] R. Horn and C. Johnson. Topics in Matrix Analysis. Cambridge University Press, 1991.

[9] Site internet. MatrixMarket. http ://math.nist.gov/MatrixMarket/, 2006.

[10] Ph. Lambin. Physique mathématique : première partie. Syllabus de première licence, Namur, FUNDP, 2000.

[11] Mélodie Mouffe. Etude des méthodes multigrilles dans le cadre de la résolution de systèmes linéaires. Mémoire de fin d'étude, Namur, FUNDP, 2005.

[12] Yvan Notay. Algebraic multilevel preconditioning with coarsening by aggregation. Présentation, 2006.

[13] J.M. Ortega and W.C. Rheiboldt. Iterative solution of nonlinear equation in several variables. Academic Press, New York, 1970.

[14] Klaus Stüben. Algebraic Multigrid (AMG) : An Introduction with Applications. GMD Report, 1999.

[15] Philippe Toint. Transparents du cours de qualité du logiciel scientifique. Namur, FUNDP, 2003-2004.

[16] Dimitri Tomanos. Introduction aux éléments finis. Transparents de la présentation, FUNDP, 2006.

[17] U. Trottenberg, C. W. Oosterlee, and A. Schuller. Multigrid. Elsevier, 2000.

[18] Petr Vanek, Marian Brezina, and Jan Mandel. Algebraic multigrid on unstructured meshes. Technical report, University of Colorado at Denver, 1994.

[19] Petr Vanek, Marian Brezina, and Jan Mandel. Convergence of Algebraic Multigrid based on Smoothed Aggregation. Technical Report, University of Colorado at Denver, 2000.

[20] Petr Vanek, Jan Mandel, and Marian Brezina. Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. Technical Report, University of Colorado at Denver, 1995.