AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

ISGMP

INSTITUT SUPERIEUR DE GENIE MECANIQUE ET PRODUCTIQUE

BIBLIOTHEQUE UNIVERSITAIRE - METZ
N° inv. 20020935
Cote S/Mz 02/29
Loc

THESE

Présentée à

UNIVERSITE DE METZ

Par

Rachid EL MOKHTARI

Pour l'obtention du grade de :

DOCTEUR de L'UNIVERSITE de METZ

SPECIALITE : *Sciences de l'ingénieur*
OPTION : *Mécanique*

Solveurs multigrilles et méthode de perturbation

Sera soutenue le 25 novembre 2002 à 14h 30 à la salle des assemblées (Bat A)
devant le jury composé de :

B. Cochelin	Professeur, ESM de Marseille	*Rapporteur*
C. Rey	Professeur, ENS de Cachan	*Rapporteur*
J.M Cadou	Maître de conférences, Université de Metz	*Codirecteur de thèse*
CH. Hirsch	Professeur, Université de Bruxelles	*Examinateur*
M. Potier-Ferry	Professeur, Université de Metz	*Directeur de thèse*
M. Vidrascu	Directeur de recherches INRIA, Paris	*Examinateur*

Laboratoire ... .N.R.S. N°7554

REMERCIEMENTS

Que Monsieur le professeur Michel Potier-Ferry, qui m'a confié le sujet de cette étude, trouve ici l'expression de ma profonde gratitude pour les conseils avisés qu'il m'a prodigué et l'aide patiente qu'il m'a apportée. L'enthousiasme qu'il m'a fait partager, sa conception originale du travail de recherche et l'éthique qu'il a suscitée dans son équipe m'ont permis de travailler dans des conditions idéales et dans un climat de réelle amitié.

Je voudrais également témoigner toute ma gratitude à Monsieur Jean Marc Cadou, maître de conférence à l'université de Metz. Je lui suis reconnaissant de l'intérêt qu'il a manifesté pour mes travaux.

Je suis très honoré que Madame le directeur Marina Vidrascu, Directeur de recherches INRIA de Paris ait accepté de présider le jury de cette thèse. Je lui en suis profondément reconnaissant.

Je remercie très sincèrement Monsieur le Professeur Bruno Cochelin de l'Ecole Supérieur de Mécanique de Marseille et Monsieur le Professeur Christian Rey de l'Ecole Normal Supérieur de Cachan qui ont bien volu accepeter de rapporter mes travaux.

Je remercie très vivement de l'honneur qu'il m'a fait en participant au jury : Monsieur le Professeur Charles Hirsch de l'université de Bruxelles.

Enfin, je remercie mes camarades du laboratoire LPMM, les enseignants qui ont facilité ma tâche et m'ont permis de travailler dans une ambiance cordiale et sympathique.

Sommaire

1 Analyse de quelques algorithmes itératifs classiques **12**

1.1 Introduction 12

1.2 Méthodes directes 14

1.3 Méthodes itératives 18

1.3.1 Principes des méthodes itératives 18

1.3.2 Une technique particulière : le gradient conjugué 21

1.3.3 Notions de préconditionneurs 24

1.3.4 Application à des systèmes avec second membres répétés . . . 30

1.4 Présentation de la méthode multigrille 36

1.4.1 Introduction 36

1.4.2 Maillages emboîtés, opérateurs de transfert 38

1.4.3 Algorithme à deux grilles classique 39

1.5 Méthodes multigrilles complètes 42

1.5.1 Stratégie de passage intergrille 46

1.5.2 Approximation initiale sur la grille fine 47

1.5.3 Coût de la méthode et implémentation 48

1.6 Conclusion 49

2 Méthode itérative d'ordre élevé à 2-grilles **51**

2.1 Introduction 51

2.2 Algorithme à 2-grilles proposé 52

2.2.1 Opérateurs de prolongement et de restriction 52

2.2.2 Homotopie et technique de perturbation 55
2.3 Mise en oeuvre numérique 60
2.3.1 Remarques techniques concernant la génération de grilles et des opérateurs 60
2.3.2 Techniques de stockage des matrices et opérateurs. 60
2.4 Premières applications, premiers résultats numériques 62
2.4.1 Présentation des tests 62
2.4.2 Détail du temps de calcul de l'algorithme 66
2.5 Comparaison avec d'autres méthodes 68
2.6 Comment optimiser la différence entre les deux grilles ? 71
2.6.1 Introduction 71
2.6.2 Un premier test sans itération 71
2.6.3 Technique itérative d'ordre élevé 72
2.6.4 Premiers tests avec la technique itérative 74
2.6.5 Influence du nombre de divisions N 75
2.6.6 Coût de calcul des approximants de Padé 78
2.6.7 Comparaison avec la méthode à deux grilles classique 79
2.7 Autres évaluations de la robustesse de l'algorithme 81
2.7.1 Test sur l'hétérogénéité de la structure 81
2.7.2 Test sur les seconds membres 82
2.8 Conclusion 83

3 Méthode multigrille d'ordre élevé avec divers préconditionneurs 85
3.1 Introduction 85
3.2 Une nouvelle classe de méthode multigrille d'ordre élevé 86
3.2.1 Une technique pour doubler une équation linéaire 86
3.2.2 Introduction d'un système augmenté doublant l'équation sur la grille grossière 88

3.2.3 Homotopie et technique de perturbation 90
3.3 Applications et résultats numériques 93
3.3.1 Introduction 93
3.3.2 Applications de la nouvelle technique à deux grilles avec préconditionnement incomplet de Cholesky 93
3.3.3 Etude détaillée du temps CPU de l'algorithme 95
3.3.4 Comparaison avec d'autres méthodes 97
3.3.5 Tests sur l'influence de la matrice C sur la vitesse de convergence de l'algorithme 99
3.3.6 Tests sur l'influence de la grille grossière 103
3.4 Maillage non-structuré : autres variantes 106
3.4.1 Influence de la distorsion du domaine 106
3.4.2 Solutions proposées 110
3.4.3 Applications numériques et discussions 113
3.5 Application du solveur multigrille pour une suite de systèmes linéaires à matrice invariante 115
3.5.1 Applications numériques et temps de calcul 116
3.5.2 Quelle précision pour le solveur multigrille ? 118
3.6 Conclusions 120

A Méthode d'ordre élevé à trois grilles **125**
A.1 Objectif 125
A.2 Technique de la méthode 125
A.3 Méthode proposée 126

B Construction de l'opérateur de prolongement P **130**

Références **132**

Introduction générale

Dans le contexe actuel, l'industrie utilise de plus en plus systématiquement la simulation numérique pour maîtriser des procédés de fabrication mécanique comme l'emboutissage ou le laminage, les codes de calcul étant basés sur la méthode des éléments finis. Ces codes doivent être à la fois fiables et rapides pour permettre à l'ingénieur d'obtenir une réponse en un temps très bref. Ceci est loin d'être réalisé dans le cas de ces procédés à cause du caractère fortement non-linéaire des modèles utilisés et du grand nombre de degrés de liberté (d.d.l) nécessaires pour discrétiser des cas industriels pour lesquels soit la forme des outils et des pièces à obtenir est complexe (cas de l'emboutissage), soit le modèle est nécessairement 3D (laminage), soit le phénomène couple des phénomènes à petite échelle et à grande échelle (couplage entre le plissement des bandes et le désalignement des rouleux dans les lignes de galvanisation et de recuit continu). Dans tous ces cas, le modèle numérique a plusieurs dizaines de milliers de d.d.l. C'est dans ce cadre de calcul numérique que s'inscrit ce travail.

A l'heure actuelle, au laboratoire de physique et mécanique des matériaux (LPMM), on possède des algorithmes très performants pour traîter des fortes non-linéarités dues soit au modèle physique (par exemple le contact unilatéral dans un contact outil-matière) [1, 12], soit à des instabilités (plissement, flambage de tôles ou de boites alimentaires). Avec ces algorithmes appelés "Méthode Asymptotique-Numérique" (MAN) [4, 17, 18, 19, 20, 21], la branche de solutions d'un problème non linéaire $R(u, \lambda) = 0$, $(u \in R^n, \lambda \in R)$ est recherchée sous forme de séries entières tronquées :

$$u(a) = \sum_{i=0}^{I} a^i u_i \tag{0.1}$$

$$\lambda(a) = \sum_{i=0}^{I} a^i \lambda_i \tag{0.2}$$

La technique de perturbation permet de montrer que les vecteurs u_i $(1 \leq i \leq I)$ sont solutions de problèmes linéaires de la forme :

$$k_t.u_i = f_i \tag{0.3}$$

où la matrice tangente k_t est la même pour les I problèmes (0.3) et où les vecteurs f_i ne sont fonctions que des vecteurs aux ordres inférieurs u_j tel que $j \leq i - 1$. La Méthode Asymptotique-Numérique (MAN) conduit donc à une famille de problèmes linéaires avec même matrice et divers seconds membres. Dans la version actuelle de la MAN, on accélère systématiquement la convergence des séries par des approximants de Padé [5, 11, 19, 35, 48], qui transforment les séries (0.1) et (0.2) sous la forme :

$$u(a) = u_0 + \sum_{i=1}^{I} f_i(a) u_i \tag{0.4}$$

$$\lambda(a) = \lambda_0 + \sum_{i=1}^{I} f_i(a) \lambda_i \tag{0.5}$$

où les $f_i(a)$ sont des fractions rationnelles. Les coefficients de ces fractions sont calculés à partir d'une orthogonalisation de l'ensemble des vecteurs u_i, $1 \leq i \leq I$.

Dans la plupart des problèmes étudiés avant cette thèse, le nombre de degrés de liberté (d.d.l) étant assez faible (en dessous de 20000 d.d.l) [15, 34, 56, 57]. Dans ce cadre, il a été montré que l'ordre de troncature optimal I se situe entre 10 et 30. Comme il y a une seule matrice à inverser pour les calculs des vecteurs u_i, on a utilisé des méthodes de résolution directe en sorte que la moitié au moins du temps de calcul est consacré à la triangulation de la matrice. En gros, on obtient toute une branche de solution pour au plus le prix de deux calculs linéaires.

A plus grand nombre de degrés de liberté, on souhaite éviter cette étape de triangulation de matrices, qui coûte cher en temps CPU parce que le nombre d'opérations nécessaires augmente comme le cube du nombre de degrés de liberté. Surtout, la mémoire nécessaire devient de plus en plus importante. Une idée est d'appliquer le technique des sous domaines [24, 29], ce qui a été fait par des équipes de recherches qui travaillent en collaboration avec notre équipe (Laboratoire de Mécanique et Acoustique de Marseille et l'université de Casablanca). Dans le cadre de cette thèse, on a travaillé en priorité sur deux types de solveurs itératifs : le gradient conjugué et surtout la méthode multigrille.

L'idée, commune à beaucoup de méthodes itératives, est de remplacer la matrice à inverser k ou k_t, par une autre matrice proche appelée "préconditionneur" notée dans ce travail k^*. Il a été montré récemmnent [14] que l'algorithme MAN + Padé peut s'appliquer, avec succès, à des problèmes linéaires :

$$R(u) = k.u - f = 0 \tag{0.6}$$

en introduisant une transformation d'homotopie de la manière suivante :

$$(1-\varepsilon)k^*.u + \varepsilon k.u - f = 0 \tag{0.7}$$

et en appliquant la technique de perturbation par rapport au paramètre ε. On déduit une succession de problèmes linéaires, où la seule matrice à inverser est k^* :

$$\left\{ \begin{array}{ll} k^*.u_0 & = f \\ k^*.u_i & = k^*.u_{i-1} - k.u_{i-1} \end{array} \right. \tag{0.8}$$

Ceci a été testé avec des préconditionneurs élémentaires (diagonal, décomposition de Cholesky incomplète de divers niveaux), l'algorithme converge, mais pas aussi vite que souhaité parce que ces préconditionneurs, utilisés seuls, ne sont pas assez efficaces.

Une technique voisine a été proposée pour la résolution itérative de problèmes non-linéaires [22, 46], puis appliquée dans un cadre de prédiction-correction d'ordre élevé [13, 36, 40, 41].

Supposons un problème de la forme :

$$R(u) = k_t.u + R^{nl}(u) + R_0 = 0 \tag{0.9}$$

où R_0 est le résidu initial et $R^{nl}(u)$ un opérateur non-linéraire. Une transformation d'homotopie assez générale est la suivante :

$$(1-\varepsilon)k^*.u + \varepsilon k_t.u + R^{nl}(u) + \varepsilon R_0 = 0 \tag{0.10}$$

Cette idée a été appliquée pour réduire le nombre de matrices à inverser, en prenant pour k^* une matrice tangente triangulée à une étape précédente du calcul. La question, qui sera discutée tout au long de cette thèse, est de trouver d'une part, des préconditionneurs beaucoup moins chers que des matrices tangentes complètes, mais tout de même très efficaces, d'autre part des stratégies de résolution bien adaptées. C'est dans cet esprit que nous nous sommes tournés vers les méthodes multigrilles.

Ces méthodes utilisent différentes discrétisations afin de résoudre itérativement un problème sur un maillage initialement fin. La grille fine correspond à la discrétisation du domaine sur lequel on veut obtenir une solution. Les grilles grossières, quant à elles, correspondent à des discrétisations moins fines du domaine initial. Le passage d'informations entre ces différentes grilles se fait à l'aide de deux opérateurs : le prolongement et la restriction. Les résolutions multigrilles reposent sur des schémas itératifs, chaque itération correspondant à une résolution approchée sur une grille. Au niveau le plus grossier, on utilise généralement une résolution exacte. Sur les autres grilles, on doit définir un préconditionneur appelé lisseur, ce qui sert à "amortir" les hautes fréquences de l'erreur. Ces méthodes sont connues pour être remarquablement efficaces : nombre d'opérations nécessaires en $\mathcal{O}(n)$ où n est le nombre de points discrétisés, nombre d'itérations petit et indépendant de la taille de la maille. Il reste la difficulté de définir plusieurs maillages emboîtés, ce qui complique la tache de l'utilisateur.

Dans le premier chapitre, on rappelle les différents solveurs utilisés pour résoudre un problème d'élasticité linéaire. Ensuite, on fait le choix de résoudre les systèmes

linéaires issus de la MAN par la méthode du gradient conjugué sans ou avec préconditionnement. De plus, on a utilisé un nouvel algorithme [25, 26, 50, 51] pour des systèmes linéaires avec seconds membres répétés. On présentera ensuite dans ce chapitre les méthodes multigrilles classiques. Une étude détaillée de l'algorithme à deux grilles classique et de la méthode multigrille complète sera présentée.

Les chapitres suivants de ce mémoire, sont consacrés à la présentation de nouvelles méthodes multigrilles d'ordre élevé. Ces méthodes sont basées sur les techniques d'homotopie, de perturbation et sur les approximants de Padé, ce qui introduit une transformation radicale dans la conception de ces algorithmes. Par rapport aux méthodes multigrilles classiques, ces nouvelles méthodes pourront facilement être utilisées pour le traîtement de problèmes non-linéaires, puisque nous savons résoudre des problèmes non-linéaires avec des algorithmes d'ordre élevé et qu'il est facile de faire les deux types d'itérations en même temps. Avec ces nouveaux algorithmes, nous pourrons nous passer de la décomposition de la matrice tangente complète (grille fine) et décomposer seulement une matrice construite sur une grille dite grossière.

L'idée de base des algorithmes proposés dans ce travail est de passer de manière continue d'une grille grossière à une grille fine à l'aide d'une transformation par homotopie [46]. Nous allons modifier ainsi le problème initial à résoudre en introduisant un paramètre ε de telle sorte que pour $\varepsilon = 0$, on obtient un problème posé sur la grille grossière et que pour $\varepsilon = 1$, on retrouve le problème posé sur la grille fine. Ensuite divers préconditionneurs peuvent être utilisés sur la grille fine. Dans cette thèse nous nous sommes limités à deux types de préconditionneurs : le préconditionnement par la diagonale et la factorisation incomplète de Cholesky de niveau 0 (IC(0)). Pour le préconditionneur diagonal, on commencera tout d'abord par séparer le problème initial à résoudre en variables globales (grille grossière) et variables locales (grille fine). Le problème réduit ainsi obtenu qui dépend uniquement des variables globales, sera résolu par une méthode directe, l'autre correspondant aux variables locales sera résolue par la technique d'homotopie. Pour la factorisation incomplète de Cholesky,

la séparation des variables est incompatible avec ce préconditionneur qui couple les noeuds locaux et globaux. L'idée est donc d'écrire deux fois le problème grossier en introduisant un multiplicateur de Lagrange, puis de définir une nouvelle homotopie à partir du problème ainsi modifié.

Le cas de maillages non-structurés sera discuté en détail. Nous terminerons par une première tentative pour utiliser ces solveurs multigrilles pour les problèmes linéaires avec seconds membres multiples, issus de la technique de perturbation. Ce test sera réalisé dans le cas de non-linéarités géométriques.

CHAPITRE 1

Analyse de quelques algorithmes itératifs classiques

1.1 Introduction

Le but de ce chapitre est d'analyser les méthodes utilisées pour résoudre les problèmes linéaires rencontrés dans le cadre de la discrétisation par éléments finis de problèmes de mécanique des solides. Ces problèmes seront écrits avec l'une des notations classiques : $A.x = b$ ou $k.q = f$.

Nous nous limiterons à des éléments finis avec formulations en déplacements. Les matrices A obtenues sont presque toujours symétriques. En élasticité linéaire, elles sont aussi définies positives sauf s'il n'y a pas de déplacement fixé au bord. S'il s'agit d'une matrice tangente en élasticité non-linéaire, A n'est pas forcement positive à cause des phénomènes de bifurcation et d'instabilité [21], mais il n'y a en général qu'un petit nombre de valeurs propres négatives.

Nous considérons des applications en élasticité bidimensionnelle, ainsi que des modèles de coques minces. Dans le premier cas, les matrices sont assez mal conditionnées, comme pour tous les problèmes d'équations aux dérivées partielles discrétisés. Les modèles de coques minces conduisent à des matrices très mal conditionnées, parce que la rigidité transversale d'une plaque est faible par rapport à la rigidité longitudinale.

En résumé, nous nous intéressons à des matrices symétriques, définies positives

ou presque définies positives et mal conditionnées.

Actuellement, un problème à un millier de degrés de liberté demande moins d'une seconde de temps calcul sur n'importe quel ordinateur de bureau. L'optimisation d'un tel calcul n'a donc plus aucun intérêt. Les problèmes visés sont donc les problèmes à très grand nombre d'inconnues : le million d'inconnues tend à devenir banal, aussi bien en laboratoire que par exemple pour des calculs de crash de véhicules. C'est cette évolution qui explique un regain d'intérêt pour les méthodes itératives, y compris dans le milieu des ingénieurs qui est le plus réticent vis à vis de ces méthodes.

Pour analyser l'efficacité d'une technique de résolution, quatre critères sont classiquement retenus :

1. Le coût en temps calcul (ou temps CPU).

2. Le coût de la mémoire.

3. La robustesse de l'algorithme.

4. La facilité d'utilisation.

Trop souvent, les recherches mettent l'accent sur le temps de calcul, parfois sur la mémoire. Au contraire, les utilisateurs mettent l'accent sur la robustesse et sur le "temps utilisateur" : il n'y a rien de plus pénalisant qu'un calcul qui ne converge pas ou qui demande beaucoup de temps de mise en place.

La facilité d'utilisation est un critère important qui conduit à rejeter en pratique des algorithmes théoriquement efficaces. Cela implique aussi que les opérations de préparation du calcul doivent être le plus légères possibles : aujourd'hui, on aimerait supprimer les maillages, alors que les méthodes les plus efficaces (sous-domaines, multigrilles) demandent plusieurs niveaux de maillages.

Un autre critère est l'efficacité pour les problèmes non-linéaires. Dans notre équipe, nous nous intéressons à la Méthode Asymptotique-Numérique [18], qui, comme l'algorithme de Newton modifié, conduit à la résolution de problèmes linéaires avec seconds membres répétés : $A.x_i = b_i$. Cela incite à l'utilisation de méthodes directes, ou, si ce n'est pas possible à cause d'un trop grand nombre de d.d.l, à la recherche de très

bons préconditionneurs, en sorte que le coût de calcul de chaque problème soit très réduit.

La revue d'algorithmes présentée ici inclurera :

– Les méthodes directes les plus classiques (Cholesky, Crout)

– Les méthodes itératives élémentaires (Jacobi, Gauss-Seidel)

– Le gradient conjugué préconditionné, qui sera testé y compris avec seconds membres répétés.

– La méthode multigrille, qui bien que trop peu utilisée en mécanique des solides, est souvent considérée comme la meilleure méthode possible.

1.2 Méthodes directes

On appelle méthode directe de résolution d'un système linéaire une méthode qui aboutit à la solution exacte du problème au bout d'un nombre fini d'opérations arithmétiques. Les plus utilisées sont :

– La méthode d'élimination de Gauss et ses dérivées basées sur la notion d'échange de lignes et de colonnes.

– La méthode de Cholesky, basée sur la décomposition triangulaire et qui s'applique à des systèmes dont la matrice est symétrique définie positive.

– La méthode de Crout s'utilise elle pour les matrices symétriques pas nécessairement définies positives.

Pour simplifier la présentation des méthodes directes, on se restreint aux matrices symétriques. Les méthodes directes considérées dans ce chapitre transforment la résolution du système linéaire (de taille $n \times n$) :

$$A.x = b \tag{1.1}$$

en deux systèmes linéaires à matrices triangulaires plus faciles à résoudre. Ces techniques, qui utilisent la factorisation de la matrice A en un produit de matrices triangulaires, varient suivant les propriétés de A. Nous renvoyons le lecteur à [16, 42, 38, 39]

pour plus de détails. On distingue généralement deux types de décomposition :

• La décomposition de Cholesky :

Si A est symétrique définie positive, il existe une matrice triangulaire inférieure unique L, telle que $A = L.^tL$.

Résoudre (1.1) revient alors à résoudre le système suivant :

$$\begin{cases} L.y &= b \\ {}^tL.x &= y \end{cases} \tag{1.2}$$

• La décomposition de Crout :

Si A est symétrique et régulière, il existe une matrice triangulaire inférieure L à diagonale unité, et une matrice diagonale D unique, telles que $A = L.D.L^t$. La résolution complète du système (1.1) s'effectue alors suivant l'ordre :

$$\begin{cases} L.z &= b \\ D.y &= z \\ {}^tL.x &= y \end{cases} \tag{1.3}$$

Ces deux méthodes sont très utilisées en mécanique des structures car elles sont robustes et que l'on peut prédire le temps CPU nécessaire à la résolution du système. On a vu qu'elles se décomposent en deux étapes : une étape de décomposition (triangulation ou factorisation) de la matrice et une étape de montée-descente pour la résolution. Si on considère la décomposition de Cholesky par exemple, on s'aperçoit tout de suite que cette méthode est très bien adaptée à la résolution d'une série de systèmes linéaires avec la même matrice. En effet, une fois la matrice factorisée lors de la résolution du premier système, la résolution des autres systèmes n'utilise plus que l'étape de montée-descente. D'ailleurs, c'est pour cette raison que, jusqu'à présent, la résolution en séquentiel des systèmes linéaires issus de la MAN est réalisée à l'aide d'un solveur direct de type Crout.

Maintenant et pour illuster en terme de temps CPU, le comportement des méthodes directes, nous proposons d'appliquer la méthode de Crout à plusieurs problèmes d'élasticité linéaire : un premier exemple de petite taille (l'exemple du toit :726 d.d.l), un second de taille moyenne (le cylindre :5190 d.d.l) et un troisième de grande taille (le

cylindre 40000 d.d.l). Les exemples numériques étudiés dans ce chapitre sont présentés dans les figures (1.1) et (1.2).

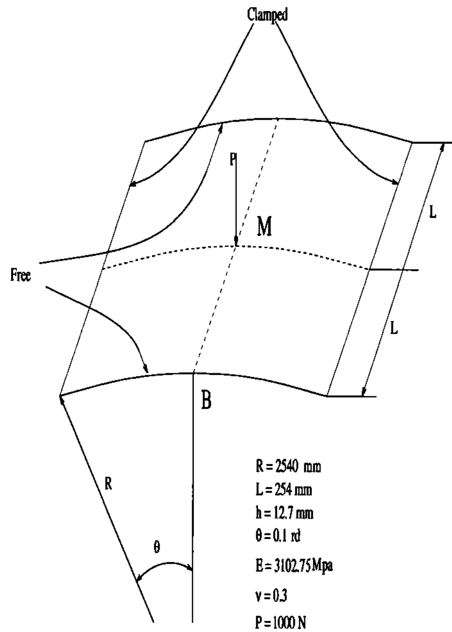


FIG. 1.1: Toit : description géométrique

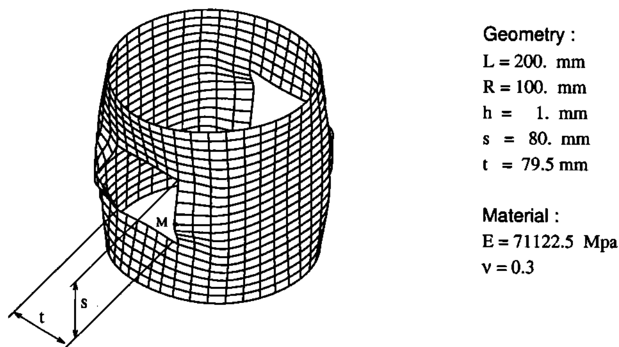


FIG. 1.2: cylindre : description géométrique

Le temps CPU, pour une décomposition de la matrice A par la méthode de Crout est donné dans le tableau (1.1) pour les trois valeurs de n.

On remarque que, conformément à la théorie, le temps de décomposition augmente de façon importante lorsque le nombre d'inconnues du problème augmente. En effet, le nombre d'opérations [42], requis pour effectuer une décomposition est souvent élevé (environ $\frac{2}{3}n^3$ en valeur asymptotique pour un système dont la matrice d'ordre n est

quelconque).

	Nombre de d.d.l.	Temps CPU (s)
Exemple 1	726	0.12
Exemple 2	5190	5
Exemple 3	39756	2000

TAB. 1.1: Temps CPU pour une décomposition de la matrice de rigidité par la méthode de Crout. Eléments finis de coques à 6 d.d.l par noeud (triangles DKT 18)

De plus, l'utilisation d'une méthode directe nécessite un stockage profil (ligne de ciel) de la matrice ce qui entraîne un surcoût de stockage lorsque le nombre de d.d.l du problème étudié devient important.

Sur le tableau (1.2), les mêmes trois maillages (726 d.d.l, 5190 d.d.l, 39756 d.d.l), on donne le pourcentage de coefficients nuls de la matrice de rigidité A et le nombre de réels stockés. Ce tableau montre que par exemple pour l'exemple 3 (39756 d.d.l), le nombre de réels stockés inutilement (termes nuls de la matrice A) dépasse les 95% de la totalité du stockage. Il faut cependant préciser que ces résultats numériques ont été réalisé pour un élément DKT à trois noeuds avec $6d.d.l$ par noeud.

	Nombre de d.d.l.	Réels stockés	Coefficients nuls de A
		profil	Pourcentage
Exemple 1	726	100122	72%
Exemple 2	5190	1 812066	89%
Exemple 3	39756	65 647380	97%

TAB. 1.2: Nombre de réels stockés de la matrice A, suivant le stockage profil. Pourcentage de coefficients nuls stockés sous forme profil. Les modèles numériques sont des modèles de coques à 6 d.d.l par noeuds.

A préciser que les méthode directes citées précédemment s'adaptent parfaitement bien avec un stockage profil, car la décomposition conserve la structure de la matrice et on n'a pas besoin de stocker de matrices suplémentaires. L'inconvénient de ce type de stockage est qu'il augmente lorsque le nombre de d.d.l augmente et conduit à un "sur-stockage"(nombre important de termes nuls stockés inutilement).

Cette étude du temps CPU et du nombre de réels stockés nous montre que la décomposition triangulaire et le stockage des matrices sont des opérations coûteuses pour les matrices de grande taille.

Ceci nous amène naturellement à considérer les méthodes itératives pour la résolution de grands systèmes linéaires. Etant donné la grande variété des méthodes itératives existantes, nous allons nous limiter aux méthodes de relaxation et à la méthode du gradient conjugué (les algorithmes les plus performants dans le cadre de la méthode des éléments finis).

1.3 Méthodes itératives

1.3.1 Principes des méthodes itératives

Dans cette section, nous donnons le principe des méthodes itératives, nous présentons en particulier les méthodes de Jacobi et de Gauss-Seidel.

Soit à résoudre l'équation :

$$A.x = b \tag{1.4}$$

Pour résoudre cette équation par une méthode itérative, on écrit (1.4) sous la forme suivante :

$$x = M.x + c \tag{1.5}$$

Partant d'une estimation initial x^0, que l'on porte au second membre de (1.5), on obtient une nouvelle estimation x^1, et ainsi de suite. Si à l'itération numéro k, on a une estimation x^k, l'estimation à l'itération numéro $k+1$ est donnée par :

$$x^{(k+1)} = M.x^{(k)} + c \tag{1.6}$$

Dans les méthodes itératives, trois problèmes se posent :

1. Comment obtenir la matrice M pour que les équations (1.4) et (1.5) soient équivalentes ?

2. Le processus converge t-il ?, si oui à quelle vitesse ?

3. Peut-on accélérer la convergence ?

La façon la plus simple d'obtenir M est de décomposer la matrice A en deux matrices : A_1 et A_2 sous la forme :

$$A = A_1 + A_2 \tag{1.7}$$

Ainsi le problème (1.4) s'écrit :

$$(A_1 + A_2)x = b \tag{1.8}$$

et donc

$$A_1.x = -A_2.x + b \tag{1.9}$$

Si A_1 est régulière alors le problème (1.4) devient :

$$x = -(A_1)^{-1}.A_2.x + (A_1)^{-1}.b \tag{1.10}$$

qui s'écrit bien sous la forme (1.5), en posant $M = -(A_1)^{-1}.A_2$ et $c = (A_1)^{-1}.b$.

Remplacer le système (1.4) par le système (1.9) n'est intéressant que si ce dernier système est bien plus facile à résoudre que le système de départ, c'est-à-dire si la matrice A_1 est simple, ou ce qui revient au même facilement inversible.

Le choix le plus simple consiste à décomposer A sous la forme :

$$A = L + D + U \tag{1.11}$$

où D est une matrice diagonale ayant pour éléments les éléments diagonaux de A, L est la partie triangulaire inférieur de A, U est la partie triangulaire supérieure de A comportant des zéros sur la diagonale. L'équation (1.4) s'écrit alors :

$$(L + D + U).x = b \tag{1.12}$$

• Méthode de Jacobi :

Dans la méthode de Jacobi, on choisit $A_1 = D$ et $A_2 = L + U$ et on écrit (1.12) sous la forme :

$$D.x = -(L + U).x + b \tag{1.13}$$

Ainsi la matrice M et le vecteur c dans le processus itératif (1.5) sont choisis par :

$$M_{Jacobi} = -D^{-1}.(L + U) \qquad \text{et} \qquad c_{Jacobi} = D^{-1}b \tag{1.14}$$

• Méthode de Gauss-Seidel

Dans cette méthode on écrit (1.12) sous la forme $A_1 = L + D$ et $A_2 = U$:

$$(L + D).x = -U.x + b \tag{1.15}$$

On a donc :

$$M_{GS} = -(L + D)^{-1}.U \qquad \text{et} \qquad c_{GS} = (L + D)^{-1}b \tag{1.16}$$

La matrice du premier membre de (1.15) est triangulaire inférieure avec diagonale non nulle, donc (1.15) est aussi un système très facile à résoudre.

Avant d'introduire les méthodes de relaxation, donnons quelques caractéristiques de ces deux méthodes itératives (Jacobi et Gauss-Seidel) :

1. Le nombre d'opérations est pratiquement le même que celui du calcul de $A.x$ pour x donné, ce qui est avantageux si la matrice est creuse.

2. On stocke en mémoire A, b et un vecteur (deux pour la méthode de Jacobi).

3. Dans le cas où les méthodes convergent, il faudra que cette convergence soit suffisamment rapide pour que le nombre d'itérations ne soit pas trop grand.

4. Ces méthodes sont très simples à programmer.

On peut généraliser les deux méthodes précédentes, Jacobi et Gauss-Seidel, en introduisant un paramètre réel ω. Soit $x_i^{(k)}$ calculé et $\hat{x}_i^k$ obtenu à partir de $x^{(k)}$ par l'une des deux méthodes précédentes. On définit alors la combinaison linéaire :

$$x_i^{(k+1)} = \omega \hat{x}_i^{(k+1)} + (1 - \omega) x_i^{(k)} \tag{1.17}$$

Si la méthode de base est celle de Jacobi, on obtient pour $i = 1$ à n :

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}}.(b_i - \sum_{j=1}^{n} a_{ij}.x_j^k) + (1-\omega).x_i^k \tag{1.18}$$

ou, en notation matricielle :

$$x^{(k+1)} = (1 - \omega D^{-1}.A)x^{(k)} + \omega D^{-1}.b \tag{1.19}$$

De même, en explicitant (1.17) dans le cas de la méthode de Gauss-Seidel, on construit la méthode appelée méthode de relaxation (S.O.R. en langue anglaise pour Successive Over Relaxation). Pour $i = 1$ à n :

$$x^{(k+1)} = (\frac{D}{\omega} - L)^{-1}b + (\frac{D}{\omega} - L)^{-1}[(\frac{1}{\omega} - 1)D + U]x^k \tag{1.20}$$

La généralisation, par relaxation, de la méthode de Jacobi est très peu utilisée parce qu'en général, elle n'apporte aucun gain. Par contre, la méthode de relaxation S.O.R. est d'un emploi courant car elle améliore souvent la rapidité de la convergence. L'introduction du paramètre ω peut s'interpréter en remarquant que si la "correction" apportée à la i^{eme} composante "va dans le bon sens", on a intérêt à l'augmenter en la multipliant par un facteur plus grand que 1. Au contraire, si on risque d'osciller ou de diverger, il faut alors "amortir" cette correction en la multipliant par un facteur plus petit que 1. Le paramètre ω est indépendant de k et de i. Pour certaines méthodes, il dépendra de k (par exemple dans la méthode de TCHEBYCHEFF [42]).

1.3.2 Une technique particulière : le gradient conjugué

La méthode du gradient conjugué [43] est adaptée à la résolution de systèmes linéaires dont la matrice est symétrique, définie positive et creuse. Elle utilise, à l'itération $(k+1)$, une direction de descente p^{k+1} qui ne coincide pas avec le gradient au point x^{k+1}, mais qui est la conjuguée de la direction précédente p^k. De façon plus précise, p^{k+1} et p^k sont telles que :

$$^tp^{k+1}.A.p^k = 0 \tag{1.21}$$

Dans les méthodes de descentes [43], il faut se donner la valeur initiale p^0 pour la direction de descente, et une relation de récurrence entre p^{k+1} et p^k que l'on peut écrire sous la forme :

$$x^{k+1} = x^k + \alpha^k p^k \qquad et \qquad p^{k+1} = r^{k+1} + \beta^k p^k \tag{1.22}$$

Plusieurs couples de valeurs α^k et β^k sont possibles, on peut choisir par exemple :

$$\alpha^k = \frac{{}^t p^k}{{}^t p^k.A.p^k} \qquad et \qquad \beta^k = -\frac{{}^t r^{k+1}.A.p^k}{{}^t r^k.A.p^k} \tag{1.23}$$

Dans la suite de ce travail, nous allons choisir :

$$\alpha^k = \frac{{}^t r^k.r^k}{{}^t p^k.A.p^k} \qquad et \qquad \beta^k = \frac{{}^t r^{k+1}.r^{k+1}}{{}^t r^k.r^k} \tag{1.24}$$

Pour démarrer le processus, on prend comme directions de descente $p^0 = r^0$. L'algorithme de la méthode du gradient conjugué sans préconditionnement (le produit scalaire est noté par (.,.)) est alors :

Initialisation

$\forall\ x^0$

$p^0 = r^0 = b - A.x^0$

Début d'itérations

Pour $k = 0, 1, ...$

$\alpha^k = \frac{||r^k||^2}{(A.p^k, p^k)}$

$x^{k+1} = x^k + \alpha^k p^k$

$r^{k+1} = r^k - \alpha^k A.p^k$

$\beta^{k+1} = \frac{||r^{k+1}||^2}{||r^k||^2}$

$p^{k+1} = r^{k+1} + \beta^{k+1} p^k$

Fin d'itérations si :

$\| r^{k+1} \| / \| b \| <$ précision souhaitée

Algorithme du gradient conjugué sans préconditionnement

En théorie, l'algorithme du gradient conjugué converge en au plus n itérations vers la solution du problème (1.1). Du point de vue pratique, il est parfois nécessaire d'effectuer plus de n itérations pour obtenir la convergence, car à cause des erreurs d'arrondi ou du mauvais conditionnement de la matrice A, les directions générées ne sont pas exactement orthogonales.

A chaque itération de l'algorithme du gradient conjugué, il faut effectuer :

- 1 produit matrice vecteur : $A.p^k$.
- 1 produit scalaire : $(\| r^{k+1} \|^2)$.
- 3 combinaisons linéaires de vecteurs pour calculer :x^{k+1}, r^{k+1}, p^{k+1}.

Soit un total de $2n_z + 4n$ opérations (additions et multiplications) par itérations, n_z étant le nombre de coefficients non nuls de la matrice A. Par ailleurs il faut stocker 4 vecteurs pour réaliser les calculs : x^k, r^k, p^k, $A.p^k$, soit un total de $n_z + 4n$ mots mémoire nécessaires pour implémenter la méthode.

Le tableau (1.3), donne pour deux types d'exemples, le nombre d'itérations et le temps CPU pour résoudre le système linéaire $A.x = b$.

On remarque que le nombre d'itérations et le temps CPU nécessaires à la convergence de l'algorithme du gradient conjugué augmentent rapidement lorsque la taille du problème devient importante. Pour accélérer la convergence de l'algorithme du gradient conjugué, il est indispensable d'utiliser les techniques de préconditionnement. Une des plus efficaces, dans le cas d'une matrice symétrique définie positive, est la technique de factorisation incomplète de Cholesky.

Nombre de d.d.l.	Nombre d'itérations	Temps CPU (s)
5190	4324	104
39756	9949	2388

TAB. 1.3: Nombre d'itérations et temps C.P.U. pour la méthode du gradient conjugué. Test d'arrêt : $\| r^{k+1} \| / \| b \| \leq \varepsilon = 10^{-10}$

1.3.3 Notions de préconditionneurs

La rapidité de convergence des méthodes itératives dépend du nombre de conditionnement de la matrice A : $cond(A)$ (où $cond(A)$ est le rapport entre la plus grande et la plus petite valeur propre de la matrice A). En général, on admet que plus $cond(A)$ est proche de 1, plus l'algorithme converge rapidement.

En effet dans le cas de la méthode des éléments finis, les valeurs propres de la matrice A ne sont pas réparties de manière satisfaisante et le nombre de conditionnement est très élevé. Pour pallier à cette difficulté, il est possible d'améliorer la vitesse de convergence de la méthode, en diminuant $cond(A)$ de la façon suivante :

$$C^{-1}.A.x = C^{-1}.b \tag{1.25}$$

On a ainsi remplacé la matrice A par la matrice $C^{-1}A$ qui, si C^{-1} est bien choisie, est mieux conditionnée que la matrice A. La matrice C doit être choisie de telle sorte que le conditionnement $cond(C^{-1}A)$ soit beaucoup plus petit que $cond(A)$. En théorie, le meilleur choix est $C^{-1} = A^{-1}$ car alors dans ce cas $cond(C^{-1}A) = 1$. En pratique, il faudra trouver C^{-1} le plus proche de A^{-1}, sans que les calculs pour C^{-1} ne soient trop coûteux. L'algorithme du gradient conjugué préconditionné appliqué à la matrice A avec un préconditionnement C^{-1} est :

$\forall\ x^0$
$r^0 = b - A.x^0$
$z^0 = C^{-1}.r^0$
$p^0 = z^0$

Pour $k = 0, 1, ...$

$\alpha^k = \frac{(z^k, z^k)}{(A.p^k, p^k)}$

$x^{k+1} = x^k + \alpha^k p^k$

$r^{k+1} = r^k - \alpha^k A.p^k$

Calcul de la nouvelle direction

$z^{k+1} = C^{-1}.r^{k+1}$

$\beta^{k+1} = \frac{(r^{k+1}, r^{k+1})}{(z^k, r^k)}$

$p^{k+1} = z^{k+1} + \beta^{k+1} p^k$

Stop si $\| r^{k+1} \| / \| b \| \leq$ précision souhaitée

Algorithme du gradient conjugué préconditionné

Le préconditionneur le plus simple à utiliser est le préconditionneur diagonal. C est alors une matrice diagonale dont les coefficients sont les termes de la diagonale de la matrice A.

Le tableau (1.4) donne le nombre d'itérations et le temps CPU nécessaires pour la résolution des mêmes problèmes linéaires par la méthode du gradient conjugué avec préconditionnement diagonal.

Ces résultats montrent que le préconditionneur diagonal améliore la convergence de l'algorithme, en diminuant le nombre d'itérations et le temps CPU par rapport à l'algorithme du gradient conjugué simple (1904 itérations au lieu de 4324 dans le cas de l'exemple à 5190 d.d.l. et 5426 itérations au lieu de 9949 dans le cas de l'exemple à 39756 d.d.l.).

Dans la littérature, il existe de nombreuses variantes de techniques de préconditionnement

Nombre de d.d.l.	Nombre d'itérations	Temps CPU (s)
5190	1904	50
39756	5426	1450

TAB. 1.4: Nombre d'itérations et temps CPU pour la méthode du gradient conjugué avec préconditionnement diagonal

autres que le préconditionnement diagonal. Ces techniques sont utilisées soit :

- pour améliorer le conditionnement du système résultant.

- pour favoriser l'aspect stabilisation numérique de la factorisation incomplète.

- pour son implémentation sur des ordinateurs vectoriels.

- pour son utilisation des calculateurs parallèles.

- ou encore pour l'influence de la numérotation des inconnues sur le préconditionnement.

Nous ne présentons dans ce travail qu'une forme de préconditionnement efficace dans le cas général, utilisable sur tous les ordinateurs et implémentée dans le code MODULEF : le préconditionnement par niveau.

Nous avons vu précédemment, que toute matrice symétrique définie positive pouvait, par décomposition de Cholesky, se mettre sous la forme :

$$A = L.L^t \tag{1.26}$$

Si on effectue cette décomposition et qu'on choisit $C = L$, on transforme la matrice A, éventuellement mal conditionnée, en une matrice unité de conditionnement 1. Evidemment, on n'applique pas cette méthode associée au gradient conjugué, car si on avait réussi à effectuer la décomposition (1.26), on aurait pratiquement résolu le système.

En fait, le gradient conjugué est utilisé dans le cas des matrices de grandes dimensions. Or les matrices triangulaires L et L^t de la décomposition de Cholesky sont pleines sous le stockage profil, même si A est creuse. Afin de profiter des zéros de la matrice, on effectue une décomposition de Cholesky incomplète par "niveau", c'est-

à-dire qu'on l'effectue pour tous les éléments non nuls et seulement eux, on laisse à zéro les éléments de A qui sont nuls. On conserve ainsi une matrice creuse.

La factorisation incomplète par niveau, a été développée par Behie et Forsyth [7, 54]. Cette factorisation incomplète comporte deux étapes :

– La première étape est une factorisation logique de la matrice du système, qui détermine à l'aide de la notion de niveau, le squelette de la matrice L, c'est-à-dire l'ensemble des indices (i, j) des coefficients de la matrice L que l'on va calculer.

– Dans la seconde étape, on calcule les coefficients correspondants de la matrice triangulaire L.

Pour résumer la factorisation logique (première étape), on introduit par récurrence la notion de niveau de remplissage suivant :

- Le niveau est initialisé à zéro pour tous les coefficients non nuls de A.

- Les termes de remplissage crées par l'élimination de coefficients de niveau k sont de niveau $k + 1$.

Il est évident qu'après quelques itérations de ce procédé, on réalise la factorisation complète de la matrice A, correspondant au niveau le plus élevé, noté $l(A)$. Pour obtenir une factorisation incomplète, il suffit de choisir un niveau quelconque l, tel que $0 \leq l < l(A)$ en vue d'améliorer la convergence de l'algorithme du gradient conjugué. Cette méthode, qui est valable pour toute matrice non singulière, est une généralisation de la méthode de factorisation incomplète décrite dans [47].

En résumé l'algorithme de décomposition de Cholesky incomplète se déduit directement de l'algorithme de décomposition de Cholesky, à deux modifications près : la première consiste à stocker C non pas à la place de la partie inférieure de A , mais à part, car on a besoin de A dans l'algorithme du gradient conjugué pour calculer $w = A.p$. La seconde consiste à ne traiter que les éléments non nuls de A (ou de C).

Les figures (1.3) et (1.4), présentent l'évolution du nombre d'itérations du gradient conjugué en fonction du niveau de la factorisation incomplète pour deux types d'exemples. Pour ces deux exemples, on observe que le nombre d'itérations décroit

quand le niveau augmente. Il est certain que pour avoir un minimum d'itérations, il faut augmenter le niveau. Mais dans ce cas, le nombre de termes calculés (pour évaluer C^{-1}) augmente sensiblement.

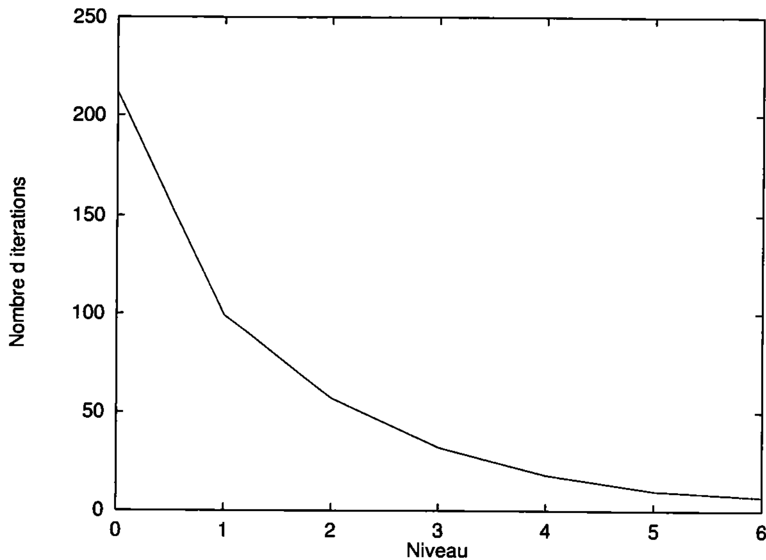


FIG. 1.3: Gradient conjugué préconditionné. Evolution du nombre d'itérations en fonction du niveau de factorisation incomplète de Cholesky (**exemple à 5190 d.d.l.**)

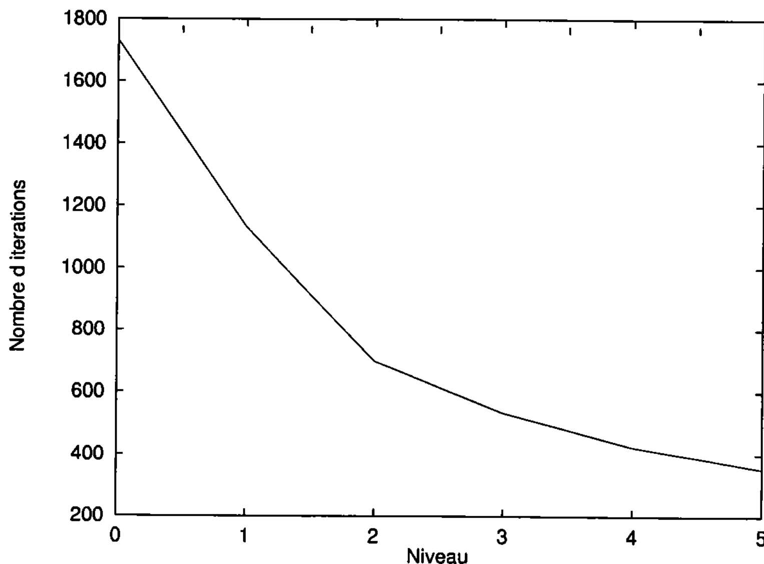


FIG. 1.4: Evolution du nombre d'itérations du gradient conjugué en fonction du niveau de factorisation incomplète de Cholesky (**exemple à 40000 d.d.l.**)

La figure (1.5) montre clairement que le gain, en nombre d'itérations, réalisé par l'augmentation du niveau est progressivement compensé par une augmentation du

temps nécessaire à la résolution du problème, si bien que le coût total est plus important pour les grandes valeurs du niveau que pour les petites. A noter, que même si la technique de préconditionnement par factorisation incomplète de Cholesky apporte une amélioration sensible du comportement de l'algorithme, elle nécessite plus d'espace mémoire qu'un gradient conjugué simple comme le montre la figure (1.6).

En pratique, il faut trouver un compromis (c'est-à-dire un niveau optimum de la factorisation incomplète) entre :

1 Le nombre de réels stockés pour la matrice C.

2 Le temps CPU de calcul de la matrice de préconditionnement C.

3 La convergence de la méthode itérative (peu d'itérations du gradient conjugué).

Au vu de tous ces critères, il semble que le préconditionneur le plus efficace et peut être le plus performant est le niveau 0, car la matrice C a la même structure que la matrice k (C et k ont le même nombre de termes) pour une diminution conséquente du nombre d'itérations (212 itérations pour la factorisation incomplète de Cholesky niveau 0, 1904 pour le préconditionnement diagonal, 4324 sans préconditionnement pour l'exemple à 5190 d.d.l).

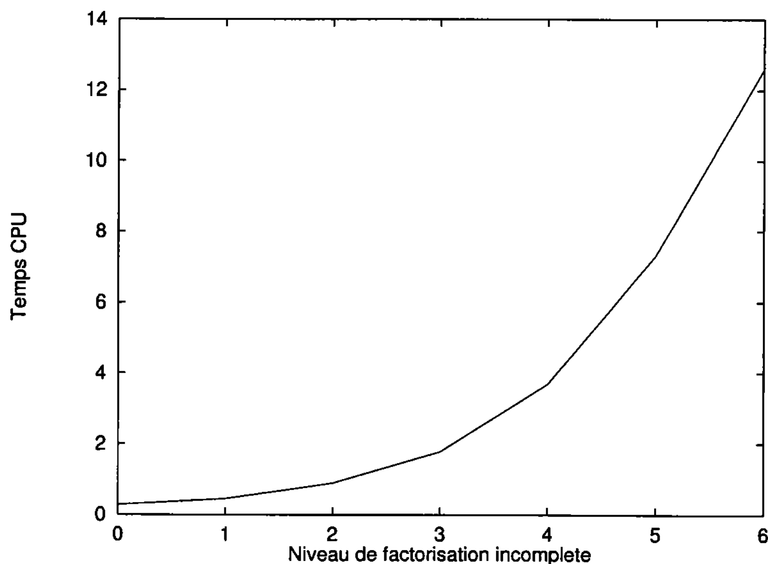


FIG. 1.5: Evolution du temps CPU de la factorisation incomplète en fonction du niveau de factorisation

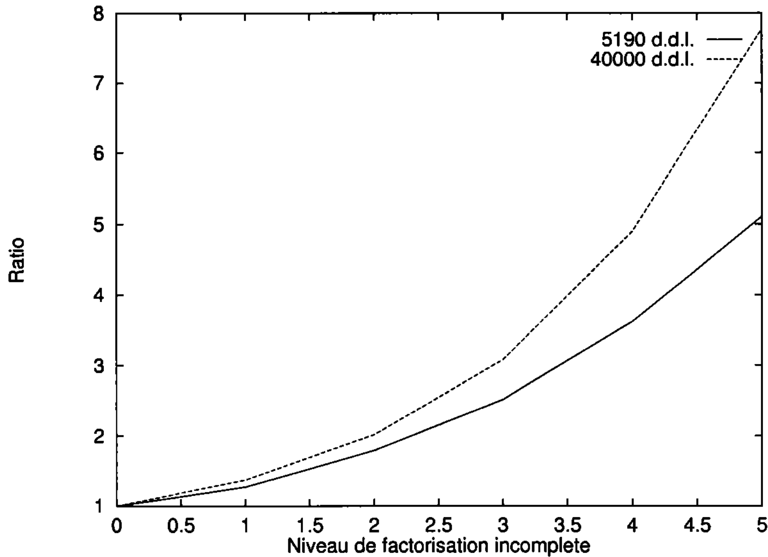


FIG. 1.6: Evolution du ratio : nombre de réels de C sur nombre de réels de A (ou k) en fonction du niveau

Il faut préciser de plus que la factorisation incomplète de niveau au moins égale à 1 est un processus instable difficile à maîtriser. Pour stabiliser cette décomposition, on introduit un paramètre réel σ compris entre 0 et 2 pour renforcer la dominance diagonale [38, 39]. La valeur de ce paramètre est à déterminer pour chaque problème et pour chaque niveau de la factorisation. D'après nos simulations, il ne semble pas y avoir de régle pour choisir la valeur de σ.

1.3.4 Application à des systèmes avec second membres répétés

Lorsque l'on utilise la M.A.N., pour résoudre des problèmes non-linéaires, on obtient une succession de problèmes linéaires avec une spécificité : un calcul de série demande la résolution de problèmes linéaires, qui ont tous la même matrice k, et ne diffèrent que par les seconds membres f_i :

$$k.u_i = f_i \qquad i = 1, ..., 20 \; ou \; 30 \tag{1.27}$$

Pour les problèmes de tailles moyennes (inférieures à 20000 d.d.l), on utilise classiquement des solveurs directs (décomposition de Cholesky ou de Crout) et le gain est naturel puisque la décomposition est réalisée une seule fois et le temps de calcul

d'une montée-descente est presque négligeable. Pour des problèmes de grande taille, le coût de la décomposition devient cher, puisqu'il augmente comme le cube du nombre d'inconnues. Si on utilise une méthode itérative pour résoudre les i problèmes linéaires, le temps CPU total de résolution sera multiplié par le nombre de problèmes linéaires à résoudre (soit i).

Si on utilise le gradient conjugué pour résoudre chaque problème, le challenge est de trouver des moyens efficaces pour résoudre un certain nombre de problèmes en tenant compte du fait que la matrice est la même.

Dans cette section on utilisera la méthode de Farhat, Roux [25, 26] et Rey [50, 51] pour résoudre chaque problème linéaire. Cette technique tient compte des calculs faits au cours des itérations précédentes. En effet si seuls les seconds membres du système linéaire à résoudre changent, la matrice étant la même, on peut utiliser les directions de descentes de l'ordre 1 pour calculer les solutions du système à l'ordre i, $i \geq 1$. L'algorithme du gradient conjugué, avec conservation des directions de descente p^k, d'après Farhat et Roux [25, 26] s'écrit de la manière suivante :

Début de calcul
On résoud à l'ordre 1 : $k.u_1 = f_1$
Résolution par la méthode du gradient conjugué préconditionné :

On sauvegarde les directions de descente p_1^k
Solution en n^1 itérations
Fin de calcul : ordre 1

Réorthogonalisation des n^1 directions de descente

Début de calcul : ordre 2, on résoud : $k.u_2 = f_2$

On pose : $u_2 = u_2^0 + v_2$

Calcul de u_2^0

$\hat{f}(j) =< p_1^j, f_2 >$, pour $j = 1..n^1$

$y(j) = \frac{\hat{f}(j)}{d_j}$, pour $j = 1, ..., n_1$ avec $d_j = \alpha_j$ (ordre 1)

$u_2^0(i) = y_2^0(j).p_1^j(i)$ pour $j = 1, ..., n_1$ et $i = 1, ..., n$
Orthogonalisation de u_2^0

On détermine v_2 **par G.C.P.** (avec u_2^0 comme solution initiale)

Algorithme du Gradient Conjugué Préconditionné

Pour chaque itration : orthogonalisation de la direction de descente p_2^k

Algorithme de Farhat et Roux [25, 26]

L'algorithme de Farhat et al [25, 26] nécessite une réorthogonalisation des directions de descente à chaque ordre : en effet lors de l'application de l'algorithme du gradient conjugué, on a constaté que les directions de descente p_k ne sont pas vraiment orthogonales entre elles. Il y a une perte d'orthogonalité au cours du processus itératif. Cette propriété d'orthogonalité est une condition restrictive pour la convergence de la méthode du gradient conjugué. A l'exception des deux modifications suivantes : calcul de u_2^0 et réorthogonalisation des p_2^k, l'algorithme du gradient conjugué préconditionné reste inchangé.

La figure (1.7) montre l'évolution du nombre d'itérations en fonction de l'ordre de

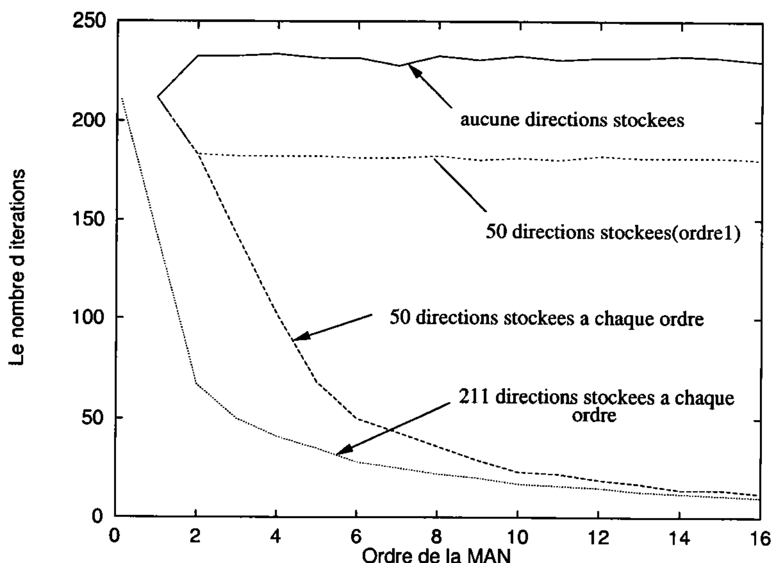


FIG. 1.7: Evolution du nombre d'itérations en fonction de l'ordre de troncature de la M.A.N. (exemple à 5190 d.d.l.)

troncature i de la M.A.N (ce qui veut dire qu'on résoud le i^{eme} problème linéaire en tenant compte des $i-1$ premiers) dans le cas où aucune direction de descente n'est stockée, le cas où seulement 50 directions de descente sont stockées (à l'ordre 1), le cas où on a stocké 50 directions de descente à chaque ordre i de la M.A.N et le cas où on stocke toutes les directions de descente pour chaque ordre i (c'est-à-dire 212 directions de descente stockées et réorthogonalisées à l'ordre 1, n_2 à l'ordre 2,...).

On remarque que dans le cas où aucune direction de descente n'est stockée, le nombre d'itérations reste pratiquement constant, le temps CPU nécessaire pour résoudre chaque problème reste constant et par conséquent le temps CPU total est important. Dans le cas où l'on stocke 50 directions de descente à l'ordre 1, les résolutions suivantes demandent alors environ 50 itérations de moins que l'ordre 1.

En stockant à chaque ordre le même nombre de directions de descente, la convergence devient alors monotone et le nombre d'itérations décroît très rapidement. La même remarque est observée quant on stocke ou on réorthogonalise toutes les directions de descente à chaque ordre. Ces résultats sont en accords avec ceux de la littérature [50].

Les résultats des trois stratégies en temps CPU et en nombre d'itérations sont présentés dans le tableau 1.5.

Ordre	Stratégie 1 CPU (Iter)	Stratégie 2 CPU (Iter)	 p_k stockés	Stratégie 3 CPU (Iter)	 p_k stockés
1	13.64(211)	13.41(211)	50	35.24(211)	211
2	17.2(233)	28.44(183)	100	19.24(67)	278
3	17.2(233)	27.6(142)	150	17.65(50)	328
4	17.27(234)	23.96(102)	200	16.5(41)	369
5	17.08(232)	19.36(68)	250	15.58(35)	404
6	17.08(232)	16.72(50)	300	13.81(28)	432
7	16.8(228)	16.47(43)	393	13.21(25)	457
8	17.09(233)	15.41(36)	429	12.5(22)	479
9	17.02(231)	13.75(29)	458	12(20)	499
10	17.09(233)	12.1(23)	471	11(17)	516
11	16.97(231)	12.15(22)	493	10.67(16)	532
12	17.05(232)	11.18(19)	512	10.37(15)	547
13	17.08(232)	10.6(17)	529	9.59(13)	560
14	17.09(233)	9.61(14)	543	9.27(12)	572
15	17.07(232)	9.58(14)	557	8.84(11)	583
16	16.87(230)	8.83(12)	569	8.5(10)	593
Total	**255.79(3478)**	**235.75(569)**	**569**	**188.78(593)**	**593**

TAB. 1.5: Temps CPU en seconde et nombre d'itérations pour chaque ordre de la MAN. Résolution par la méthode du gradient conjugué préconditionné : Cholesky incomplet de niveau 0, avec et sans application de l'algorithme de Farhat pour l'exemple du cylindre à 5190*d.d.l.* Trois stratégies sont étudiées, première stratégie : sans utiliser la méthode de Farhat, deuxième stratégie : 50 directions stockés à chaque ordre (résolution) de la MAN, troisième stratégie : toutes les directions de descente sont stockées à chaque ordre (résolution) de la MAN.

Le tableau 1.5 indique le temps de calcul en seconde et le nombre d'itérations de la MAN pour la résolution d'un problème non linéaire. On précise que l'objectif de ces tests est de diminuer le temps de calcul de la stratégie 1. Cette stratégie est une application simple de la méthode du gradient conjugué préconditionné (sans réorthogonalisation des directions de descente).

On constate sur ce tableau que pour la stratégie 1, le temps CPU et le nombre d'itérations reste pratiquement constant, le temps total est environ 16 fois le temps nécessaire pour résoudre le problème à l'ordre 1. Par conséquent, cette stratégie n'est

pas très intéressante pour résoudre un problème non-linéaire.

Pour la troisième stratégie où on réorthogonalise toutes les directions de descente à chaque ordre, la résolution du problème à l'ordre 1 est celle qui prend le plus de temps CPU (environ 19% du temps CPU total). Pour les ordres suivants, ce temps diminue progressivement en fonction des ordres. Nous constatons clairement sur cet exemple que le temps CPU devient trop important si le nombre de vecteurs à réorthogonaliser est trop élevé (voir la colonne concernant la deuxième stratégie).

Pour toutes ces stratégies, le temps CPU reste assez élevé par rapport à l'utilisation d'une méthode directe. En effet, la résolution du même problème par la méthode de Cholesky coûte beaucoup moins cher par rapport aux trois stratégies précédentes. Le coût d'une décomposition de la matrice tangente est de 5 secondes et le temps d'une résolution par montée /descente est presque négligeable (0.24 secondes). Le coût total de la résolution du problème non linéaire par une méthode directe est de 9 secondes.

Il faut cependant noter que Hadji a montré dans sa thèse [32] que chaque problème linéaire issu de la MAN pouvait être résolu avec un critére d'arrêt variable : une précision de résolution qui évolue avec l'ordre de troncature de la série. Récemment, Galliet [29] a montré qu'on peut obtenir une branche de la solution MAN avec un paramètre de précision égal à 10^{-4} par exemple, mais à condition de résoudre chaque problème linéaire issu de la MAN avec la même précision. Ces techniques n'influencent pas la qualité finale de la solution de la MAN et s'accompagne d'une diminution conséquente du nombre d'itérations et du temps CPU pour résoudre chaque problème linéaire.

D'autres tests ont été effectués pour différentes valeurs du niveau de préconditionnement et avec une renumérotation adéquate et en utilisant un critére d'arrêt variable. Malheureusement, les résultats restent peu satisfaisants, les temps de calcul étant assez important par rapport au temps de calcul d'une méthode directe (à titre d'exemple, le temps CPU donné par une renumérotation des noeuds + gradient conjugué

préconditionné niveau 6 + utilisation d'un critère d'arrêt variable est 60 secondes).

Cependant, une conclusion générale ne peut être avancée, car d'autres facteurs peuvent intervenir quand on utilise les stratégies précédentes pour résoudre un grand problème avec la M.A.N. Parmis ces facteurs, la difficulté de réorthogonaliser les directions de descente et le problème de stockage de ces directions, qui rendent presque inutilisable cette technique dans le cas des problèmes à grand nombre de degrés de liberté. En effet si l'on étudie un exemple à $40000d.d.l$ avec un solveur gradient conjugué avec préconditionnement incomplet de cholesky de niveau 0, la figure (1.4) montre que ce solveur converge à environ 1700 itérations. Pour résoudre un problème non-linéaire avec la MAN il nous faudra, à l'ordre 1 environ 1700 itérations pour obtenir la précision demandée. Il est évident que sur cet exemple, l'on ne pourra réorthogonaliser et stocker ces 1700 vecteurs (1700 itérations) de $40000d.d.l$: stockage et temps CPU énormes. Cette simple étude démontre la limite d'utilisation de ce type de solveur.

A noter cependant que cette technique de conservation des directions de descente présentée dans ce travail n'a pas été développée à l'origine pour traiter des problèmes non-linéaires ou linéaires complets (de grande dimension), mais des problèmes non-linéaires ou linéaires traités par la méthode de sous-structuration [29], (problèmes de petites tailles, problèmes d'interface).

1.4 Présentation de la méthode multigrille

Des méthodes alternatives à ces solveurs itératifs ont été developpées depuis les années 70. Ces méthodes s'appuient sur la résolution de problèmes à partir de plusieurs niveaux de maillages : ce sont les méthodes multigrilles.

1.4.1 Introduction

L'idée des méthodes multigrilles a été introduite pour la première fois par Federenko [27, 28] dans les années soixante pour la résolution de l'équation de Poisson sur

un domaine carré discrétisé en différences finies. Ces méthodes permettent d'améliorer la rapidité de résolution des problèmes issus de la modélisation, tout particulièrement pour les systèmes de grande taille. Leur efficacité a été initialement montrée sur des problèmes elliptiques et pour une discrétisation par différences finies [27]. Les premiers résultats pour une discrétisation par élément finis sont apparus environ dix ans plus tard [2]. L'impact de ces méthodes n'apparaît que quelques années plus tard grâce à l'école allemande [31], auparavant Brand [8, 9, 10] avait prouvé leur efficacité.

Si les premières applications concernent la mécanique des fluides [33, 55], par la suite, tous les domaines nécissitant des algorithmes rapides et robustes ont été touchés. La mécanique des structures a été tardivement concernée à cause des non-linéarités spécifiques [3, 44, 30].

Le principe des méthodes multigrilles s'appuie sur la considération des deux idées suivantes :

1. Si la solution d'un problème sur un maillage fin est assez régulière ou "lisse" au sens où elle ne varie pas fortement entre deux points voisins du domaine, on ne commet pas une grosse erreur en la calculant sur un maillage plus grossier.

2. La plupart des méthodes itératives usuelles comme par exemple Jacobi ou Gauss-Seidel fournissent en peu de pas une erreur qui est lisse.

Les méthodes multigrilles procèdent par traitement alternatif, au cours d'un processus itératif, du problème sur la grille la plus fine et sur des grilles de plus en plus grossières. Quelques opérations de lissage d'un solveur itératif sont effectuées sur la grille fine à l'aide d'une méthode adaptée au problème, puis une correction sur la grille grossière est calculée grâce à une résolution complète d'un problème déduit du précédent. L'efficacité des méthodes multigrilles peut être expliquée en terme de fréquences spatiales de l'erreur entre la solution et les itérés. L'erreur est écrite dans la base des vecteurs propres associés à l'opérateur discrétisé. Les lisseurs (Jacobi, Gauss-Seidel par exemple) réduisent les composantes de l'erreur de haute fréquence mais un grand nombre d'itérations est nécessaire pour réduire les composantes de

basse fréquence. Après avoir rappelé le principe des méthodes à deux grilles, nous développons, l'algorithme multigrilles complet.

1.4.2 Maillages emboîtés, opérateurs de transfert

La méthode 2-grilles nécessite deux niveaux de maillages. La grille fine Ω_h s'obtient en divisant chaque élément (par exemple) de la grille grossière Ω_H en quatre rectangles. Le passage des informations entre les grilles s'effectue à l'aide des opérateurs de prolongement et de restriction. Pour des raisons évidentes de simplification de ces opérateurs, nous nous limiterons à des maillages emboîtés.

• Opérateur de prolongement P.

On appelle prolongement tout opérateur d'interpolation sur le maillage fin des fonctions connues par leurs discrétisations sur le maillage grossier (1.8).

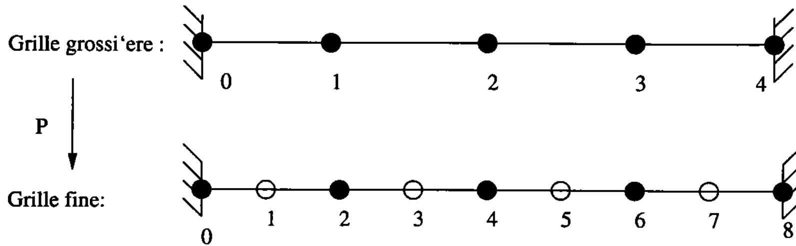


FIG. 1.8: Exemple d'opérateur de prolongement P

Par exemple, soit l'interpolation linéaire suivante :

$$q_{2h} = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \end{Bmatrix} \longrightarrow q_h = \begin{Bmatrix} q'_1 = (q_0 + q_1)/2 \\ q'_2 = q_1 \\ q'_3 = (q_1 + q_2)/2 \\ q'_4 = q_2 \\ q'_5 = (q_2 + q_3)/2 \\ q'_6 = q_3 \\ q'_7 = (q_3 + q_4)/2 \end{Bmatrix} \tag{1.28}$$

L'opérateur de prolongement P est alors :

$$P = \begin{bmatrix} 1/2 & 0 & 0 \\ 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 0 & 0 & 1/2 \end{bmatrix}$$

- Opérateur de restriction.

On appelle restriction tout opérateur d'interpolation sur le maillage grossier de fonctions connues par leurs discrétisations sur le maillage fin (1.9) .

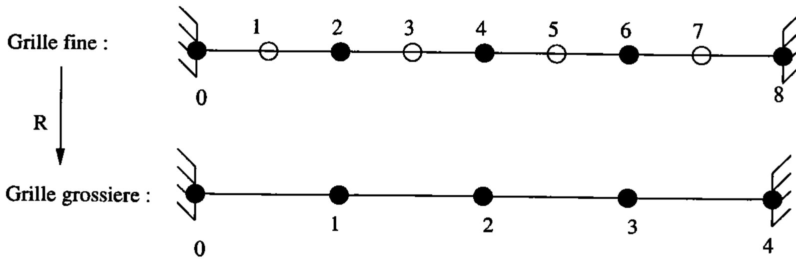


FIG. 1.9: Exemple d'opérateur de restriction

Sur l'exemple précédent, l'opérateur de restriction R peut être défini par :

$$R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (1.29)$$

1.4.3 Algorithme à deux grilles classique

Pour fixer les idées, on suppose que le maillage grossier est deux fois moins dense que le maillage fin : la grille fine est de pas h et la grille grossière est de pas $2h$. Un cycle de la méthode 2-grilles se compose de deux phases.

Au cours de la première phase de l'algorithme à deux grilles classique on effectue quelques itérations de la méthode de surrelaxation par exemple pour résoudre le problème (1.30) posé sur la grille fine Ω_h.

$$k_h.q_h = f_h \quad (1.30)$$

On appellera cela la méthode itérative de base ou lisseur, on notera S la méthode choisie.

Soit u_h l'approximation de q_h obtenue après quelques itérations (en pratique 2 ou 3). La correction $v_h = q_h - u_h$, qu'il faut ajouter à u_h pour obtenir q_h, doit vérifier :

$$k_h.v_h = f_h - k_h.u_h = r_h \tag{1.31}$$

v_h est donc la solution d'un problème de même type que celui qui définit q_h, où le second membre a été remplacé par le résidu r_h aux noeuds de Ω_h. Le calcul de v_h est, a priori, aussi coûteux que celui de q_h. Mais on peut chercher à en obtenir une approximation sur la grille grossière de pas H. Ceci nécessitera beaucoup moins de calculs puisque la grille de pas H possède beaucoup moins de degrés de liberté que celle de pas h.

La deuxième phase de chaque cycle de la méthode à 2-grilles consiste alors à :

• Définir la restriction du résidu r_h sur le maillage grossier à l'aide d'un opérateur de restriction R, on obtient ainsi r_H. L'opérateur R peut être, tout simplement une injection pour tous les noeuds du maillage grossier, mais il est possible également d'utiliser une restriction par moyenne pondérée.

• Résoudre le système d'équation :

$$k_H.v_H = r_H \tag{1.32}$$

• Prolonger v_H, obtenue sur Ω_H, en une fonction définie sur Ω_h, par exemple par interpolation linéraire.

Cette deuxième phase nécessite donc la résolution de (1.32) sur la grille grossière de pas H. Son coût sera bien inférieur à celui du problème analogue (1.31) sur la grille fine. La question principle qui se pose, est de savoir si effectivement le prolongement de v_H est une bonne approximation de v_h et dans ce cas quel est le gain que l'on peut attendre de cette méthode à deux grilles ? (pour plus de détail, voir [42, 43]). En résumé l'algorithme à deux grilles pour résoudre le système (1.30) s'écrit de la manière suivante :

1. Lissage

• ν_1 itérations de lissage (prè lissage) :

$u_h = S^{\nu_1}(k_h, q_h, f_h)$

2. Correction sur la grille grossière :

• Calcul du défaut (résidu) : $d_h = k_h.u_h - f_h$

• Restriction du défaut à la grille grossière : $d_H = R.d_h$

• Résolution de l'équation de correction : $k_H.v_H = d_H$

• Prolongement de la correction à la grille fine : $v_h = P.v_H$

• Correction de la solution : $\tilde{u}_h = u_h - v_h$

Algorithme d'un pas d'une méthode à deux grilles classique : AGC

Nous avons testé l'algorithme à 2-grilles classique dans le cas d'un problème d'élasticité linéaire 2D pour résoudre le problème posé sur la grille fine. Le lisseur utilisé est la méthode de Jacobi relaxée avec 3 itérations de lissage et $\omega = 0.8$.

Pour cela, on a fixé le nombre de degrés de liberté de la grille fine à $65522 d.d.l.$, la grille grossière dépend du nombre de division N. Le tableau (1.6) donne le nombre de degrés de liberté de la grille grossière, le nombre de division de la maille N, le nombre de cycles nécessaire à la résolution du problème fin, et le temps CPU total de l'algorithme. On rappelle ici que le nombre de cycles est le nombre de fois oû l'on effectue l'algorithme de la méthode à deux grilles classique.

On constate que, dans le cas d'un lissage par la méthode de Jacobi, le temps CPU total augmente quand le nombre de division N augmente, c'est-à-dire lorsque le rapport entre le nombre de degrés de liberté de la grille grossière et celui de la grille fine augmente. On remarque qu'à partir de $N = 2$, le fait d'augmenter le nombre de divisions N n'a plus beaucoup d'effet sur la convergence de l'algorithme. Ceci est essentiellement dû aux manques d'informations apportées par le lisseur de

Jacobi. Les matrices de préconditionnement de type Jacobi sont très intéressantes par leur simplicité de programmmation et l'occupation réduite de l'espace mémoire. Malheureusement, les résultats présentés, ci-dessous, ne sont pas satisfaisants. Il semble donc très fortement déconseillé d'utiliser des schémas seulement à 2-grilles en multigrilles classiques avec un lisseur de Jacobi. C'est la raison pour laquelle nous présenterons dans la prochaine section des schémas plus performants avec plusieurs niveaux de grilles.

Grille grossière(d.d.l.)	N	Itérations de lissage	nombre de cycles	CPU total (s)
16562	2	3	34	128
7442	3	3	71	153
4232	4	3	126	243
2738	5	3	201	382
1922	6	3	298	562

TAB. 1.6: Algorithme à 2-grilles classique, résolution itérative, lisseur de Jacobi relaxé($\omega = 0.8$), le test d'arrêt est 10^{-10}, la grille fine étant fixée à $65522 d.d.l.$

1.5 Méthodes multigrilles complètes

Dans la description de la méthode utilisant deux grilles, on a vu qu'il fallait réaliser les opérations suivantes, au cours d'un cycle :

1/ Effectuer quelques itérations de Jacobi ou de Gauss-Seidel sur la grille fine de pas h.

2/ Calculer le résidu sur la grille fine.

3/ Restreindre ce résidu sur la grille grossière de pas H, la restriction se faisant par injection ou par moyenne pondérée.

4/ Résoudre le système d'équations obtenu sur la grille grossière pour obtenir une approximation de la correction.

5/ Interpoler cette correction sur la grille fine pour corriger la solution approchée que l'on avait obtenue sur la grille fine.

Si à la fin de cette cinquième étape, le résidu sur la grille fine est trop grand, on

recommence un autre cycle, en reprenant les opérations au début de la première étape. A la quatrième étape, pour résoudre (avec une approximation suffisante) le système d'équations obtenu sur la grille grossière, on peut de nouveau utiliser la méthode à 2-grilles, en définissant un problème sur une grille encore plus grossière que la grille grossière de pas H. La méthode multigrille est donc l'application récursive de la méthode à 2-grilles. On utilise une méthode directe (Cholesky ou Crout) pour résoudre le système d'équations sur la grille la plus grossière, ne contenant que quelque degrés de liberté. Il reste à préciser, comment on déduit les équations sur la grille grossière de pas H, à partir des équations sur la grille fine de pas h. Ecrivons le système linéaire sur la grille fine de la manière suivante :

$$k.q = f \tag{1.33}$$

L'idée des méthodes multigrilles pour résoudre l'équation (1.33) repose sur le couplage de deux méthodes, une efficace pour réduire les composantes des grandes fréquences de l'erreur, et une autre pour les basses fréquences. Prenons par exemple la méthode de Jacobi relaxée :

$$q_{k+1} = q_k - \theta[k.q_k - f], \;\; \theta \in [0,1] \tag{1.34}$$

Pour toutes les valeurs de $[0,1]$, $\theta = 1$ est la meilleure solution du point de vue de la vitesse de convergence, ce qui revient à la méthode de Jacobi standard (non relaxée). Par contre, pour $\theta = 1/2$, la vitesse de convergence de la méthode de Jacobi relaxée, restreinte au sous-espace des hautes fréquences, est de $1/2$. La convergence lente est causée par les faibles fréquences uniquement. Cette méthode est donc une méthode de lissage. On peut alors combiner cette méthode avec une correction effectuée sur une grille plus grossière que le maillage initial. Représentons par l le niveau de maillage : $h_l = 2^{(l+1)}$. Soient $q_l^{(1)}$ une approximation donnée par $q_l = k_l^{-1}.f_l$ et q_l^2 le résultat obtenu après quelques itérations de lissage. L'erreur $v_l = q_l^2 - q_l$ vérifie :

$$k_l.v_l = d_l \tag{1.35}$$

avec $d_l = k_l.q_l^{(2)} - f_l$. On approche alors le système (1.35) par : $k_{l-1}.v_{l-1} = d_{l-1}$. Pour cela, on doit définir des opérateurs de restriction et de prolongation. Si P est l'opérateur de prolongement de la grille grossière sur la grille fine, et R l'opérateur de restriction de la grille fine à la grille grossière, alors on obtient une nouvelle approximation de q_l par :

$$q_l^{(3)} = q_l^{(2)} - P.k_{l-1}^{-1}.R.(k_l.q_l^{(2)} - f_l) \tag{1.36}$$

Remarquons que la correction par grille grossière est une méthode itérative non convergente. Elle ne peut donc être employée seule. C'est uniquement la combinaison des itérations de lissage et de la correction qui engendrera une méthode à convergence rapide. Nous avons ainsi défini une méthode à 2-grilles. Mais elle n'est pas souvent utilisable, car elle requiert la solution exacte de $k_{l-1}.v_{l-1} = d_{l-1}$. On va donc calculer une solution approchée de v_{l-1} par une autre méthode à 2-grilles. Ce procédé peut être répété jusqu'à ce que tous les niveaux $(l, l-1, \ldots\ldots\ldots\ldots 0)$ soient impliqués. Les équations au niveau 0 doivent être résolues exactement (ou presque). On obtient ainsi une méthode $(l+1)-grilles$, appelée méthode multigrilles. A noter qu'on peut choisir une autre méthode de lissage que celle de Jacobi relaxée, par exemple la méthode de Gauss-Seidel ou celle du Damier [42]. De plus, la méthode 2-grilles peut utiliser un près et un post-lissage, ainsi qu'un lissage intérmédiaire. L'algorithme MultiGrilles (MG) s'écrit de la façon suivante :

Procédure : MG(l,q,f)
Entrée : le niveau de la grille l, un vecteur q et le second membre f.

Sortie : une approximation de $k_l.q_l = f_l$, placée dans q (exacte si $l = 0$)
Début
Si $l = 0$ alors
• $q = k_0^{-1}.f$

Sinon

• $q = S^{\nu_1}(k_l, q, f)$

→ ν_1 itérations de lissage (près-lissage)

• $d = R(k_l.q - f)$ et $v = 0$

→ restriction du problème à la grille de niveau inférieur

• Pour $i = 1$ à β faire $MG(l-1, v, d)$

→ approximation de la solution de $k_{l-1}.v_{l-1} = d_{l-1}$

• $q = S^{\nu_2}(l, q, f)$

→ ν_2 itérations de lissage (lissage supplémentaire)

• $q = q - Pv$

→ correction par grille grossière

• $q = S^{\nu_3}(l, q, f)$

→ ν_3 itérations de lissage (post-lissage)

FIN

Algorithme MultiGrilles : MG

Les différentes étapes de lissage peuvent utiliser des itérations de méthodes différentes. De plus, on peut remplacer $q = l_0^{-1}.f$ pour $l = 0$ par ν itérations d'une méthode itérative afin d'obtenir une approximation de la solution. La procédure décrite ci-dessus est assimilable à un cycle de calcul et donc à une nouvelle forme d'itération. Pour résoudre entièrement le problème de départ il est utile de répéter plusieurs fois cette itération, ou ce cycle, pour avoir une bonne approximation de la solution. A noter

que la procédure de lissage doit être peu coûteuse puisqu'elle est employée un grand nombre de fois au cours d'un cycle. C'est pour cette raison que nous n'emploiyerons que des méthodes itératives simples telles que la méthode de Jacobi relaxée décrite précédement.

1.5.1 Stratégie de passage intergrille

Dans la méthode multigrille, les changements de grilles se font en deux circonstances :

1/ Quand on restreint le résidu d'une grille fine sur une grille plus grossière, pour définir le second membre du problème dont la solution est l'approximation de la correction à apporter.

2/ Quand on interpole la correction approchée d'une grille grossière à une grille plus fine.

Plusieurs stratégie sont possibles. Dans la première, le nombre d'itérations de Gauss-Seidel par exemple sur chaque niveau de grilles et l'ordre dans lequel elles sont faites, sont fixés à l'avance. Par exemple, on impose d'effectuer un nombre p fixe d'itérations de Gauss-Seidel avant de calculer le résidu à transférer sur la grille grossière. Par ailleurs, lors de la "remontée" d'une grille grossière à la grille immédiatement plus fine, on peut envisager, après avoir apporté la correction, de refaire n itérations de Gauss-Seidel pour le problème défini sur cette grille, avant d'apporter la correction à la grille de niveau supérieur. Cela conduit à la figure (1.10) dans le cas de 4 niveaux de grilles, ce que l'on appelle la stratégie des V-cycles.

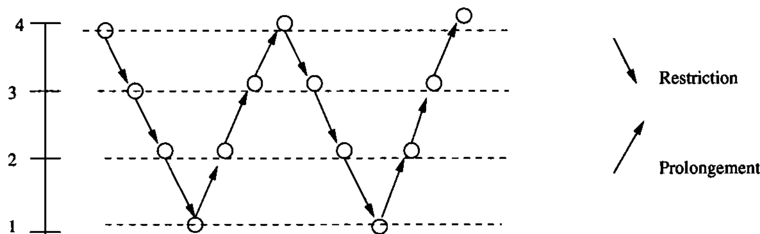


FIG. 1.10: Schéma d'un V-cycle

On peut également n'interpoler la correction sur une grille donnée qu'après avoir

effectuer deux cycles sur la grille immédiatement plus grossière, comme le montre la figure (1.11), ce que l'on appelle la stratégie des W-cycles.

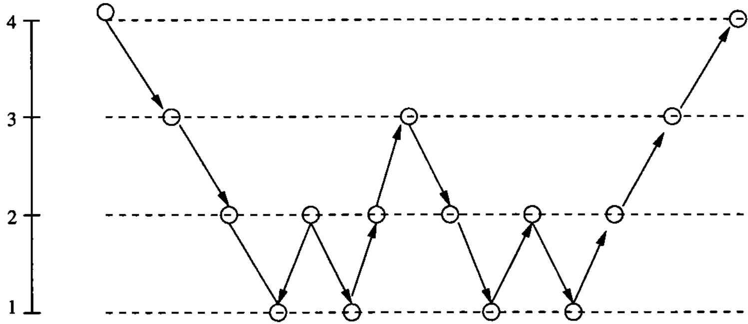


FIG. 1.11: Schéma d'un W-cycle

Une autre stratégie consiste à changer de niveau en fonction de la valeur du résidu et de la rapidité de décroissance de ce résidu. Tant que le résidu est à peu près divisé par deux d'une itération de Gauss-Seidel à l'autre, on continue ces itérations. Lorsque cette décroissance se ralentit, on décide de "descendre" à la grille la plus grossière. On poursuit ce processus sur chaque grille jusqu'à ce que le résidu soit suffisamment petit, auquel cas on admet avoir résolu, avec une précision suffisante, le problème sur ce niveau de grille et on peut donc "remonter" pour apporter la correction au niveau de grille immédiatement supérieur. Cette stratégie, bien qu'apparemment plus souple, présente l'inconvénient du choix des seuils sur le facteur de décroissance ou sur la norme du résidu pour changer de grille.

1.5.2 Approximation initiale sur la grille fine

Au lieu de choisir au hasard une approximation initiale sur la grille la plus fine, on peut envisager de la construire à partir de la solution des problèmes correspondant aux discrétisations sur chacune des grilles les plus grossières. Supposons que l'on ait obtenu une approximation suffisante de la solution du problème discrétisé sur le niveau j. On peut interpoler cette solution sur la grille la plus fine de niveau $(j+1)$ et s'en servir d'approximation initiale pour calculer la solution du problème sur cette grille de niveau $(j+1)$. En pratique, un ou deux cycles sur chaque grille sont suffisants.

L'avantage de cette méthode réside dans son coût relativement faible, compte tenu du fait que le rapport entre le nombre de points d'une grille et celui des points de la grille immédiatement plus fine est environ 1/4 pour un problème à deux dimensions d'espace.

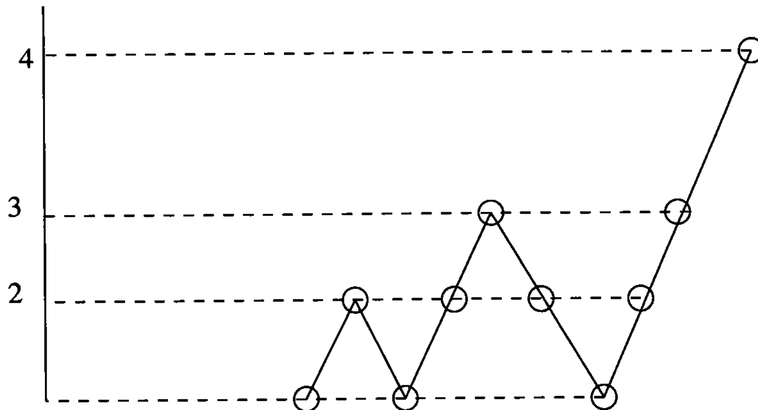


FIG. 1.12: V-cycle d'initialisation

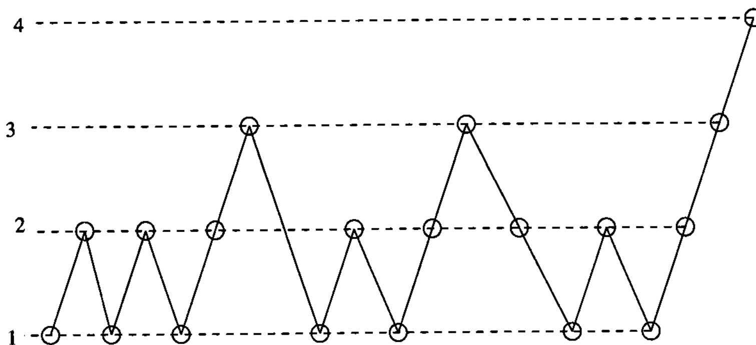


FIG. 1.13: W-cycle d'initialisation

1.5.3 Coût de la méthode et implémentation

Le principal avantage de la méthode multigrille est l'indépendance du nombre d'itérations nécessaires à la convergence par rapport à la taille du pas de discrétisation. En pratique, on peut espérer que le coût total en temps soit de l'ordre de celui nécessaire pour effectuer quelques dizaines d'itérations d'une méthode de relaxation

(ou ce qui revient au même de calculs de résidu). Néanmoins, il existe un coût initial de fabrication "d'assemblage" des matrices sur les grilles grossières qui ne pourrait être négligé lors de maillages quelconques.

La place nécessaire au stockage supplémentaire des matrices sur les grilles intermédiaires n'est pas considérable. Le coût des opérations de prolongement et restriction peut devenir très important si on doit définir ces opérations à l'aide des coefficients de la matrice du problème sur la grille fine.

Enfin la programmation de la méthode multigrille, plus complexe que celle des méthodes de relaxation ou du gradient conjugué reste relativement simple lorsque les différentes grilles se déduisent régulièrement les unes des autres. Elle se complique si l'on veut étendre la méthode multigrille à des matrices creuses de structures quelconques.

1.6 Conclusion

Dans ce chapitre, nous avons présenté sous forme d'algorithmes les principes généraux des méthodes itératives et les méthodes multigrilles. Les méthodes de Gradient Conjugué (GC) sont sans doute les méthodes itératives les plus employées pour la résolution de système linéaire. Plus simples que les méthodes multigrilles et particulièrement efficaces avec un préconditionneur adéquat, elles sont a priori adaptables à une architecture parallèle, car elles utilisent principalement des opérations de type produit matrice-vecteur facilement parallélisable.

Nous avons montré que dans le cas des problèmes non-linéaires, l'application de la méthode du gradient conjugué couplée avec l'algorithme de Farhat et Roux présente un inconvénient. En effet, la réorthogonalisation des directions de descente, les problèmes liés aux stockages, et l'augmentation du nombre d'itérations quand la taille du problème augmente, rendent presque inutilisables ces algorithmes. Dans le cas des méthodes multigrilles, le nombre de cycles pour résoudre de tels problèmes est indépendant de la taille de ces problèmes, ce qui leur permet d'être de très bons

préconditionneurs. C'est pour cette raison que nous cherchons à mettre au point des nouvelles méthodes multigrilles d'ordre élevé. Ces méthodes sont basées sur une transformation d'homotopie et une technique de perturbation.

CHAPITRE 2

Méthode itérative d'ordre élevé à 2-grilles

2.1 Introduction

L'objectif de ce chapitre est de proposer une nouvelle classe d'algorithmes à 2-grilles pour la résolution des problèmes à grand nombre de degrés de liberté comme ceux rencontrés dans les applications industrielles. L'idée de base de l'algorithme proposé est de passer de manière continue d'une grille grossière à une grille fine à l'aide d'une transformation par homotopie [46].

Nous allons modifier ainsi le problème initial à résoudre en introduisant un paramètre ε de telle sorte que pour $\varepsilon = 0$, on obtient un problème posé sur la grille grossière et que pour $\varepsilon = 1$, on retrouve le problème posé sur la grille fine. La solution du problème modifié, qui dépend de ε, sera alors cherchée sous la forme d'une représentation en série entière en ε dont on peut accélerer la convergence à l'aide de la technique des approximants de Padé [22].

Signalons que l'utilisation d'une homotopie pour passer de la grille grossière à la grille fine représente la différence principale avec les méthodes multigrilles classiques rappelées au chapitre précédent.

Notre problème consiste à définir des transformations d'homotopie, permettant un passage continu d'un problème posé sur une grille grossière à un problème posé sur une grille fine. Pour cela, on commencera par séparer le problème à résoudre en

deux problèmes, le premier est posé sur la grille grossière, le deuxième correspond aux autres inconnues.

Signalons que l'homotopie qui sera utilisée, introduit un opérateur linéaire L^*, qu'on désignera sous le nom de préconditionneur comme dans le langage des méthodes itératives. Plus précisement lorsque ce préconditionneur intervient sur la grille fine d'une méthode multigrille, il est appelé "lisseur".

Dans ce chapitre, nous nous limiterons à un lisseur diagonal (méthode de Jacobi), qui est la méthode la plus simple et la moins chère en espace mémoire, à défaut d'être la plus efficace.

En résumé, l'objectif de ce chapitre est d'introduire une méthode à 2-grilles s'appuyant sur une technique de résolution d'ordre élevé (homotopie + série + approximants de Padé).

Pour le reste de l'algorithme, on retiendra les outils classiques de la méthode multigrille : opérateurs de prolongement et de restriction et lisseur de Jacobi.

2.2 Algorithme à 2-grilles proposé

2.2.1 Opérateurs de prolongement et de restriction

Dans cette section, on propose de résoudre le système linéaire suivant :

$$k.q = f \tag{2.1}$$

Ce problème est par exemple obtenu par une discrétisation par la méthode des éléments finis d'une structure donnée, où k est la matrice de rigidité $n \times n$ de la structure qui est supposée symétrique et définie positive, q désigne le vecteur inconnu des déplacements nodaux de la structure et f est le chargement.

On appelle grille fine, la grille Ω_h correspondant à cette discrétisation. Dans la méthode proposée dans ce travail, on introduit une deuxième grille que l'on appelle grille grossière et que l'on note Ω_H. On fait l'hypothèse suivante : les deux grilles sont emboîtées, c'est-à-dire que les noeuds de la grille grossière Ω_H sont aussi des noeuds

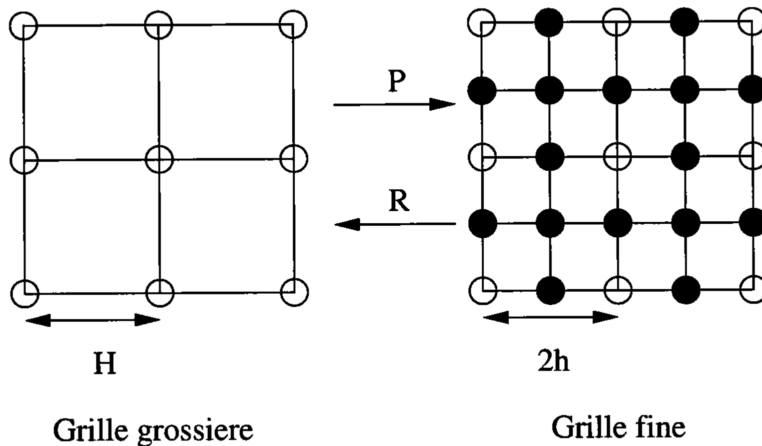


FIG. 2.1: Exemple de maillages : Ω_H est la grille grossière et Ω_h est la grille fine

de la grille fine comme le montre la figure (2.1).

Le problème (2.1) est supposé correspondre au maillage de la grille fine Ω_h. Dans cette grille, on distingue deux types de degrés de liberté (d.d.l) : les d.d.l associés à la grille grossière appellés d.d.l globaux, notés par q_g, et les autres degrés de liberté appelés d.d.l locaux et notés par q_l.

Le système (2.1) peut donc s'écrire sous la forme suivante :

$$\begin{array}{lll} k_{gg}.q_g + k_{gl}.q_l & = f_g & (a) \\ k_{lg}.q_g + k_{ll}.q_l & = f_l & (b) \end{array} \tag{2.2}$$

Comme dans les méthodes multigrilles classiques, l'opérateur de prolongement P permet de passer de la grille grossière à la grille fine. Les d.d.l globaux q_g restent inchangés et les d.d.l locaux q_l sont définis par interpolation linéaire à partir de l'opérateur *int* et les d.d.l q_g :

$$q_l = int(q_g) \tag{2.3}$$

l'opérateur de prolongement P peut être défini par l'équation suivante :

$$\left\{ \begin{array}{c} q_g \\ int(q_g) \end{array} \right\} = P.q_g \tag{2.4}$$

L'opérateur ${}^tP$ définit l'opérateur de restriction et permet le passage de la grille fine à la grille grossière, comme c'est souvent le cas dans la littérature.

En multipliant l'équation (2.1) par l'opérateur de restriction ${}^tP$, on obtient l'équation réduite suivante qui correspond à un problème posé sur la grille grossière :

$$ {}^tP.k.q = F \qquad \text{où} \qquad F =^t P.f \tag{2.5} $$

Le problème réduit ainsi obtenu (2.5), combiné avec l'équation (2.2-b), permet de définir un nouveau système équivalent au problème initial à résoudre (2.1) posé sur la grille fine :

$$ {}^tP.k.q = F \tag{2.6} $$

$$ k_{lg}.q_g + k_{ll}.q_l - f_l = 0 \tag{2.7} $$

On a ainsi séparé le problème à résoudre (2.1) en deux problèmes, l'un posé sur la grille grossière (2.6), l'autre correspondant aux autres inconnues (2.7).

L'équation (2.6) est posé sur la grille grossière mais contient les variables locales q_l. Elle peut être résolue par rapport à q_g et ceci sera fait à l'aide d'une méthode directe puisque la grille grossière contient relativement peu de degrés de liberté. Les équations locales (2.7) seront résolues à l'aide d'une transformation d'homotopie et d'une technique de perturbation.

Notons que si les d.d.l locaux sont interpolés à partir de la grille grossière par l'opérateur de prolongement (2.4), alors l'équation réduite devient un problème bien posé sur la grille grossière :

$$ K.q_g = F \tag{2.8} $$

où $K =^t P.k.P$ est une matrice qui représente un opérateur sur la grille grossière. Dans la cas où k est une matrice symétrique définie positive, la matrice réduite $K =^t P.k.P$ est symétrique définie positive. Si en plus le maillage est structuré (uniquement des rectangles par exemple) et si l'interpolation est linéaire, cette matrice est exactement égale à la matrice de rigidité de la grille grossière K_g. En effet dans la méthode des éléments finis, le nombre de points de Gauss n'est pas le même dans le calcul des

matrices ${}^tP.k.P$ et K_g, mais dans le cas des hypothèses évoquées précédemment, des résultats numériques ont montré que ces deux matrices coincident parfaitement.

Dans les applications numériques et pour éviter de calculer le double produit matrice-matrice ${}^tP.k.P$, qui est une opération assez coûteuse, on remplacera la matrice $K = {}^t P.k.P$ par la matrice de rigidité définie sur la grille grossière K_g.

2.2.2 Homotopie et technique de perturbation

Pour résoudre l'équation (2.7), on utilise une technique d'homotopie consistant à introduire un problème artificiellement généralisé dépendant d'un paramètre ε. Plus exactement on modifie le problème (2.7) de la manière suivante :

$$(1-\varepsilon)(q_l - int(q_g)) + \varepsilon C.\{k_{lg}.q_g + k_{ll}.q_l - f_l\} = 0 \tag{2.9}$$

L'opérateur C est appelé préconditionneur ou "lisseur" et peut être choisi arbitrairement, la seule restriction est la convergence de l'algorithme.

De cette manière $q_l(\varepsilon)$, solution de (2.9), passe progressivement de l'approximation grossière $q_l = int(q_g)$ pour $\varepsilon = 0$, à la solution du problème (2.7) pour $\varepsilon = 1$.

Ainsi on remplace la recherche de la solution du problème (2.6) et (2.7) par la solution du problème dépendant de ε suivant :

$${}^tP.k.q = F \tag{2.10}$$

$$(1-\varepsilon)(q_l - int(q_g)) + \varepsilon C.\{k_{lg}.q_g + k_{ll}.q_l - f_l\} = 0 \tag{2.11}$$

Pour $\varepsilon = 0$, ce nouveau système (2.10) et (2.11) se réduit à :

$$\left\{ \begin{array}{ll} q_l & = int(q_g) \\ K.q_g & = F \end{array} \right. \tag{2.12}$$

Pour résoudre le système (2.10) et (2.11), on utilise une technique de perturbation [46]. Cela consiste à chercher une représentation paramétrique de la solution $q(\varepsilon)$ sous la forme d'un développement en séries entières par rapport au paramètre ε tronqué à l'ordre I :

$$q(\varepsilon) = \sum_{i=0}^{I} \varepsilon^i q(i) \tag{2.13}$$

$$q_g(\varepsilon) = \sum_{i=0}^{I} \varepsilon^i q_g(i) \tag{2.14}$$

$$q_l(\varepsilon) = \sum_{i=0}^{I} \varepsilon^i q_l(i) \tag{2.15}$$

En introduisant (2.13), (2.14) et (2.15) dans (2.10) et (2.11) et en identifiant terme à terme suivant les puissances de ε, on obtient une succession de problèmes à résoudre :

A l'ordre 0 :

$$\begin{cases} {}^tP.k.q(0) & = F \\ q_l(0) - int(q_g(0)) & = 0 \end{cases} \tag{2.16}$$

A l'ordre 1 :

$$\begin{cases} {}^tP.k.q(1) & = 0 \\ q_l(1) & = int(q_g(1)) - R(1) \end{cases} \tag{2.17}$$

avec :

$$R(1) = C.\{k_{lg}.q_g(0) + k_{ll}.q_l(0) - f_l\}$$

A l'ordre $i \geq 2$:

$$\begin{cases} {}^tP.k.q(i) & = 0 \\ q_l(i) & = int(q_g(i)) - R(i) \end{cases} \tag{2.18}$$

avec :

$$R(i) = C.\{k_{lg}.q_g(i-1) + k_{ll}.q_l(i-1)\} - q_l(i-1) + int(q_g(i-1))$$

– Le problème à l'ordre 0

Dans le vecteur $q(0)$, on remplace les d.d.l locaux par leur expression en fonction des d.d.l. globaux donnés par :

$$q_l(0) = int(q_g(0)) \tag{2.19}$$

On peut ainsi écrire l'équation ${}^tP.k.q(0) = F$ sous la forme :

$${}^tP.k.\left\{ \begin{array}{c} q_g(0) \\ int(q_g(0)) \end{array} \right\} = F \tag{2.20}$$

En utilisant la définition de l'opérateur de prolongement P donnée par (2.4), l'équation (2.20) peut s'écrire sous la forme :

$$({}^tP.k.P).q_g(0) = F \tag{2.21}$$

Finallement le problème à l'ordre 0 s'écrit sous la forme suivante :

$$\left\{ \begin{array}{ll} ({}^tP.k.P).q_g(0) & = F \\ q_l(0) - int(q_g(0)) & = 0 \end{array} \right. \tag{2.22}$$

– Le problème à l'ordre 1 :

De la même manière qu'à l'ordre 0, dans le vecteur $q(1)$ de l'équation ${}^tP.k.q(1) = 0$, on remplace les déplacements correspondant aux d.d.l locaux par leur expression en fonction des déplacements des d.d.l globaux donnés par $q_l(1) = int(q_g(1)) - R(1)$.

On peut ainsi écrire (2.17) sous la forme :

$${}^tP.k.\left\{ \begin{array}{c} q_g(1) \\ int(q_g(1)) - R(1) \end{array} \right\} = 0 \tag{2.23}$$

avec :

$$R(1) = C.\{k_{lg}.q_g(0) + k_{ll}.q_l(0) - f_l\}$$

ce qui donne :

$${}^tP.k.\left\{ \begin{array}{c} q_g(1) \\ int(q_g(1)) \end{array} \right\} -{}^t P.k.\left\{ \begin{array}{c} 0 \\ R(1) \end{array} \right\} = 0 \tag{2.24}$$

qui peut aussi s'écrire :

$$({}^tP.k.P).q_g(1) -^t P.k.r(1) = 0 \tag{2.25}$$

Remarquons que la correction $r(1)$ est obtenue à partir du vecteur $R(1)$ par expansion : on met des zéros dans les composantes de $R(1)$ correspondant aux d.d.l globaux. Le vecteur $r(1)$ ainsi obtenu aura la même dimension que la grille fine.

Le problème à l'ordre 1 peut s'écrire alors sous la forme suivante :

$$\left\{\begin{array}{ll} ({}^tP.k.P).q_g(1) & =^t P.k.r(1) \\ q_l(1) & = int(q_g(1)) - R(1) \end{array}\right. \tag{2.26}$$

avec :

$$R(1) = C.\{k_{lg}.q_g(0) + k_{ll}.q_l(0) - f_l\}$$

En remplaçant la matrice réduite ${}^tP.k.P$ par la matrice K, et en tenant compte des remarques précédentes pour construire les problèmes aux ordres suivants, on obtient l'algorithme à 2-grilles suivant :

A l'ordre 0 :

$$\left\{\begin{array}{ll} K.q_g(0) & = F \\ q_l(0) - int(q_g(0)) & = 0 \end{array}\right. \tag{2.27}$$

A l'ordre 1 :

$$\left\{\begin{array}{ll} K.q_g(1) & = FQ(1) \\ q_l(1) & = int(q_g(1)) - R(1) \end{array}\right. \tag{2.28}$$

avec :

$$\left\{\begin{array}{ll} R(1) & = C.\{k_{lg}.q_g(0) + k_{ll}.q_l(0) - f_l\} \\ r(1) & = \left\{\begin{array}{c} 0 \\ R(1) \end{array}\right\} \\ FQ(1) & =^t P.k.r(1) \end{array}\right.$$

A l'ordre $i \geq 2$:

$$\left\{\begin{array}{ll} K.q_g(i) & = FQ(i) \\ q_l(i) & = int(q_g(i)) - R(i) \end{array}\right. \tag{2.29}$$

avec :

$$\begin{cases} R(i) & = C.\{k_{lg}.\ q_g(i-1) + k_{ll}.q_l(i-1)\} \\ & -q_l(i-1) + int(q_g(i-1) \\ r(i) & = \left\{ \begin{array}{c} 0 \\ R(i) \end{array} \right\} \\ FQ(i) & =^t P.k.r(i) \end{cases}$$

On remarque que le calcul des composantes du vecteurs $q(i)$ fait intervenir deux types d'opérateurs. Il y a d'une part la matrice K qui correspond à un opérateur défini sur la grille grossière. Cette matrice est la seule matrice à triangulariser dans cette technique de calcul, et puisqu'il s'agit d'une matrice sur la grille grossière, sa triangulation ne devrait pas nécessiter un temps de calcul conséquent.

D'autre part, la partie locale q_l se calcule à partir de trois opérateurs : int, k_{lg} et k_{ll} qui peuvent être définis localement. Cette dernière remarque nous amène naturellement à choisir l'inverse de la matrice diagonale locale comme "préconditionneur" : $C_{ll} = 1/k_{ll}$. On remarque de plus que tous ces problèmes ne diffèrent que par le second menbre $FQ(i)$. A chaque ordre i, ce second membre est à construire et dépend des vecteurs déplacements à l'ordre précédent $(i-1)$ comme le montre les équations (2.28) et (2.29).

Une fois le vecteur déplacement calculé pour chaque ordre i et pour accélérer la convergence de l'algorithme, on remplace l'approximation polynomiale (2.13) par des fractions rationnelles (approximants de Padé). Pour plus de détails sur les approximants de Padé, nous renvoyons le lecteur à la référence [22, 19].

Remarquons en plus que le vecteur q ainsi obtenu n'est qu'une estimation de la solution, car l'ordre de troncature I est nécessairement fini et que $\varepsilon = 1$ peut être à l'extérieur du domaine de convergence de la série (2.13).

2.3 Mise en oeuvre numérique

2.3.1 Remarques techniques concernant la génération de grilles et des opérateurs

Cette partie propose quelques explications concernant la génération automatique des grilles et des opérateurs de passage entre les grilles.

Premièrement, la grille fine est construite à partir de la grille grossière et non l'inverse comme cela se fait traditionnellement dans les méthodes multigrilles. Ce choix se justifie par la facilité de construire un maillage fin à partir d'un maillage grossier, la seule difficulté technique réside dans la gestion des "doublons" de noeuds. Ainsi un élément de la grille grossière est divisé en N^2 éléments qui formeront le maillage fin. Les coordonnées des noeuds fins sont calculées à l'aide des fonctions de forme de l'élément du maillage grossier (annexe B).

L'opérateur de prolongement est construit parallèlement à la génération des noeuds et des éléments de la grille fine (cf annexe B). Pour cette première étude sur les méthodes multigrilles, nous nous sommes limités à un élément fini isoparamétrique quadrangle à 4 noeuds. Ainsi les coefficients de l'opérateur P sont les mêmes que ceux utilisés lors de la création des coordonnées des noeuds du maillage fin. On peut cependant noter que deux tableaux, relatifs à l'opérateur P, sont générés : le premier contient des réels (les coefficients de l'opérateur) et le deuxième contient les relations pères(noeuds grossiers)-fils(noeuds fins). Ce deuxième tableau permet un gain de temps et de stockage considérable dans l'opération de multiplication par l'opérateur P : $P.q_g$.

2.3.2 Techniques de stockage des matrices et opérateurs.

Il existe plusieurs façons de stocker les matrices de rigidité issues de la méthode des éléments finis. Une des méthodes la plus courante est le stockage profil (ligne de ciel). Cette technique est bien adaptée à la résolution de systèmes linéaires par la méthode directe. Généralement, ces matrices ont une structure très creuse et par conséquent,

le nombre de termes nuls est considérable y compris à l'intérieur du stockage profil.

Nous proposons sur le tableau (2.1), pour des exemples avec un nombre de d.d.l. différents, le nombre de réels de la matrice k (matrice de rigidité de la grille fine) stockés avec les deux techniques : stockage profil et stockage compact (morse).

	Nombre de d.d.l.	Stockage profil	Stockage morse
Exemple 1	726	100122	28 122
Exemple 2	5190	1 812066	214 314
Exemple 3	39756	65 647380	1 681140

TAB. 2.1: Nombre de réels de la matrice k stockés suivant les deux techniques de stockage

Ce nombre de réels est à comparer au nombre de réels stockés avec la technique morse [38, 39].

Sur ce tableau, on voit que pour un exemple à 40000 d.d.l, la majorité des réels stockés avec la méthode de stockage en ligne de ciel (profil) sont des termes nuls. Ce type de stockage est d'une part coûteux en place mémoire et d'autre part inadapté aux opérations de multiplication matrice-vecteur (dû à la prédominance des zéros dans la matrice stockée). Pour éviter un coût de calcul conséquent lors de la multiplication matrice-vecteur, un autre type de stockage est utilisé : le stockage morse.

Cette technique permet de ne stocker que les termes non nuls de la matrice de rigidité. Elle est généralement utilisée lors de la résolution de systèmes linéaires par une méthode itérative, par contre elle est inadaptée aux solveurs directs.

Pour ces raisons, nous avons choisi d'utiliser un stockage profil pour la matrice de rigidité de la grille grossière, avec utilisation d'un solveur direct, et un stockage morse pour la rigidité de la grille fine (produits matrices-vecteurs).

Il faut cependant noter que même si la technique du stockage morse permet une diminution significative du nombre de réels stockés de la matrice de rigidité, elle nécessite un important temps CPU de construction de la rigidité de la grille fine comme le montre le tableau 2.2.

Cet inconvénient du stockage morse est connu et s'explique par la gestion de deux

	Nombre de d.d.l.	Stockage profil	Stockage morse
Exemple 1	8450	1.09	2.62
Exemple 2	33282	4.7	23.14
Exemple 3	132098	20.23	402

TAB. 2.2: Temps CPU en secondes de construction de la matrice de rigidité selon la technique de stockage

pointeurs (au lieu d'un seul pour la technique du stockage profil) pour caractériser un élément (i,j) de la matrice de rigidité.

Pour plus de détails, nous renvoyons le lecteur à [38, 39].

2.4 Premières applications, premiers résultats numériques

2.4.1 Présentation des tests

Dans cette section, on propose d'appliquer l'algorithme à 2-grilles proposé dans le paragraphe précédent dans le cas de l'élasticité linéaire $2D$. Pour ce faire, nous considérons l'exemple d'une plaque sollicitée en membrane, encastrée en A et B et soumise à une force ponctuelle f.

La plaque est discrétisée par des éléments quadrangles à 4 noeuds. Les caractéristiques géométriques et mécaniques de la plaque étudiée sont présentées dans la figure (2.2).

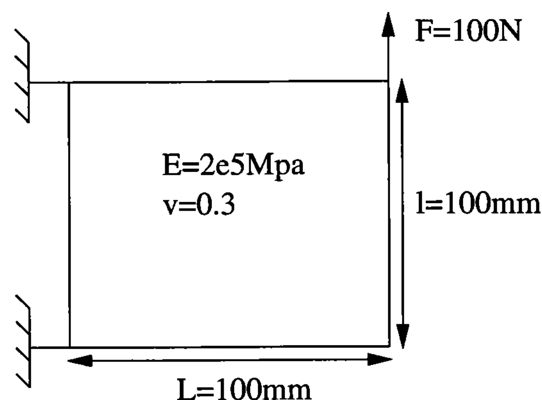


FIG. 2.2: Description géométrique et mécanique de la plaque étudiée

Ensuite et pour valider l'algorithme à 2-grilles proposé, nous considérons d'abord

deux types de maillages : la figure (2.3(a)) montre que le premier maillage donne un rapport de 4 entre le nombre d'éléments de la grille fine et celle de la grille grossière, le deuxième donne un rapport de 9 entre les grilles, voir figure (2.3(b)).

On obtient la grille fine en divisant chaque côté de la grille grossière par N, N étant le nombre de division de chaque côté de la maille.

Enfin et pour évaluer la performance de l'algorithme à 2-grilles, une étude plus systématique en fonction de N, sera présentée.

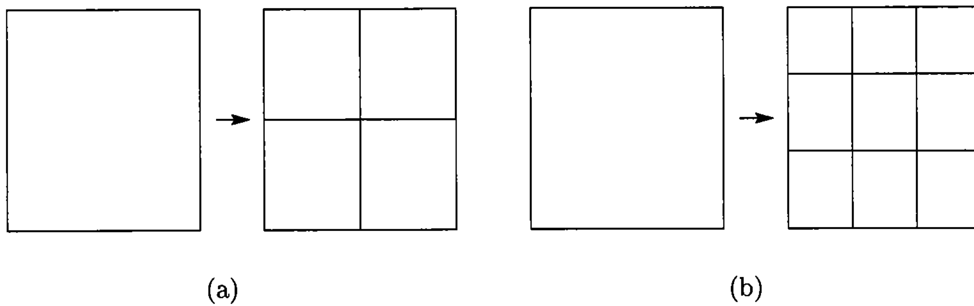


FIG. 2.3: Deux types de maillage (a) et (b)

Nous donnons les différents résultats obtenus par l'application de l'algorithme à 2-grilles proposé. Nous allons examiner pour chaque ordre de troncature le résidu obtenu à l'aide de deux représentations : la représentation en série et celle utilisant les approximants de Padé.

De nombreux exemples avec des nombres de degrés de liberté allant jusqu'à 132000*d.d.l* ont été traités. Pour l'algorithme proposé, nous discuterons tout d'abord la qualité de la solution. Celle-ci se pose d'abord en terme de choix du critère d'arrêt utilisé. Dans tous les cas, le critère d'arrêt de l'algorithme est $\frac{\|k.q-f\|}{\|f\|} \leq \eta$, où $\| \bullet \|$ désigne la norme euclidienne et η un paramètre de précision que l'on choisit égal à 10^{-10}. Ensuite une présentation détaillée du temps CPU pour chaque étape de l'algorithme est discutée.

Enfin, l'efficacité de l'algorithme proposé est évaluée en comparant les temps de calcul avec la méthode à 2-grilles classique et la méthode de résolution classique utilisant une seule grille (le gradient conjugué préconditionné).

Des commentaires relatifs aux tests de comportement et d'efficacité de l'algorithme, sont présentés. Cinq exemples seront étudiés dont les caractéristiques sont donnés dans le tableau 2.3 :

	Grille grossière	Grille fine	Rapport
test1	2178	8450	3.88
test2	8450	33282	3.93
test3	33282	132098	3.99
test4	1568	13448	8.57
test5	13448	119072	8.85

TAB. 2.3: *Nombre de d.d.l pour les cinq exemples étudiés*

- Les trois premiers exemples correspondent à un rapport 4 entre le nombre d'éléments de la grille fine et celle de la grille grossière, soit $N = 2$.
- Les deux derniers exemples correspondent à un rapport 9, soit $N = 3$.

Les tableaux (2.4), (2.5) et (2.6) montrent clairement l'accélération de la convergence obtenue, en utilisant la méthode basée sur les approximants de Padé. La représentation utilisant l'approximation polynomiale converge très lentement. Pour les trois premiers exemples, on obtient exactement le même ordre de convergence de l'algorithme, qui est de 30.

Ordre	3	10	15	20	25	30	35
Padé/F	4.47	2.9510^{-3}	6.0910^{-5}	1.2810^{-6}	2.7810^{-8}	5.4710^{-10}	1.0710^{-11}
Série/F	5.35	0.83	0.25	8.0910^{-2}	2.8410^{-2}	1.0410^{-2}	4.0810^{-3}

TAB. 2.4: *Résidus obtenus pour différents ordres de troncature :* **Test 1**

Ordre	3	10	15	20	25	30	35
Padé/F	5.54	7.310^{-3}	1.0910^{-4}	1.9310^{-6}	3.7810^{-8}	7.310^{-10}	1.410^{-11}
Série/F	5.35	0.83	0.25	8.0910^{-2}	2.8410^{-2}	1.0410^{-2}	4.0810^{-3}

TAB. 2.5: *Résidus obtenus pour différents ordres de troncature :* **Test 2**

Dans le cas où $N = 3$, les tableaux (2.7), (2.8), montrent une augmentation de l'ordre de convergence : l'ordre 59 pour le test 4 et 61 pour le test 5. La représentation

Ordre	3	10	15	20	25	30	35
Padé/F	5.28	6.310^{-3}	1.1110^{-4}	2.9310^{-6}	3.910^{-8}	7.810^{-10}	1.910^{-11}
Série/F	5.34	0.88	0.5	8.510^{-2}	2.210^{-2}	1.0410^{-2}	4.810^{-3}

TAB. 2.6: *Résidus obtenus pour différents ordres de troncature :* **Test 3**

en série est nettement moins bonne et converge très lentement, et ceci pour tous les exemples étudiés.

Une autre remarque très importante est que l'ordre de convergence de l'algorithme est de 30 pour $N = 2$, alors que pour $N = 3$ il vaut 61. On retrouve ainsi un résultat fondamental dans la théorie de la méthode multigrille classique [43] à savoir : la vitesse de convergence de l'algorithme est quasiment indépendante du pas de discrétisation du maillage.

Ordre	3	10	20	30	40	50	59
Padé/F	18.95	7.0510^{-2}	7.210^{-4}	9.5810^{-6}	1.2810^{-7}	2.6810^{-8}	8.7610^{-10}
Série/F	5.34	0.88	0.5	8.510^{-2}	2.210^{-2}	1.0410^{-2}	4.810^{-3}

TAB. 2.7: *Résidus obtenus pour différents ordres de troncature :* **Test 4**

Ordre	3	10	20	30	40	50	62
Padé/F	10.3	7.0410^{-2}	8.9310^{-4}	1.1910^{-5}	3.410^{-7}	1.8210^{-8}	9.0310^{-10}
Série/F	5.34	0.88	0.5	8.510^{-2}	2.210^{-2}	1.0410^{-2}	4.810^{-3}

TAB. 2.8: *Résidus obtenus pour différents ordres de troncature :* **Test 5**

Sur les figures (2.4, 2.5), on représente le logarithme décimal de la norme du résidu en fonction de l'ordre de troncature I. Ces courbes sont tracées pour la représentation en série et celle utilisant les approximants de Padé.

Dans le cas où $N = 2$ (figure 2.4), le résidu Padé décroît très rapidement en fonction de l'ordre de troncature. La convergence de l'algorithme utilisant les approximants de Padé est presque exponentielle (courbe linéaire sur la figure).

Dans le cas $N = 3$, figure (2.5), on constate l'apparition d'oscillations sur la courbe Padé. Ces oscillations apparaîssent parfois lorsque l'on calcule des approximants de Padé à des ordres élevés et sont probablement dûes au processus d'orthogonalisation

de Gram-Schmidt, qui est numériquement assez instable pour un nombre d'ordre de troncature élevé.

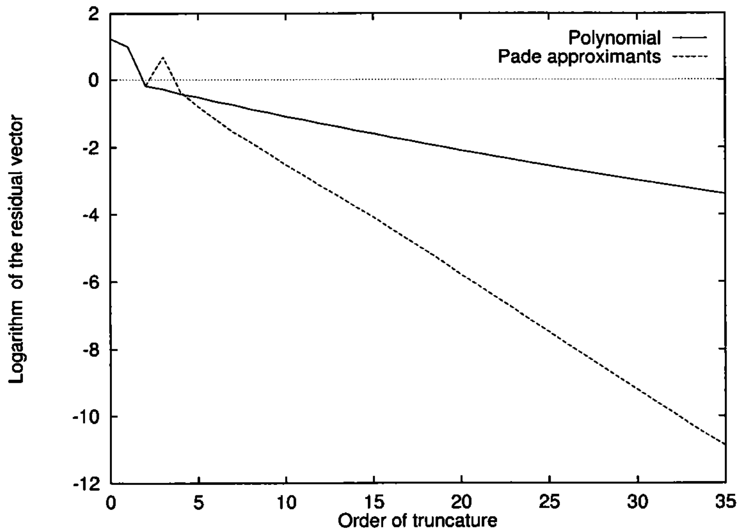


FIG. 2.4: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature, test 3 (N=2, Grille fine : 132098 d.d.l)

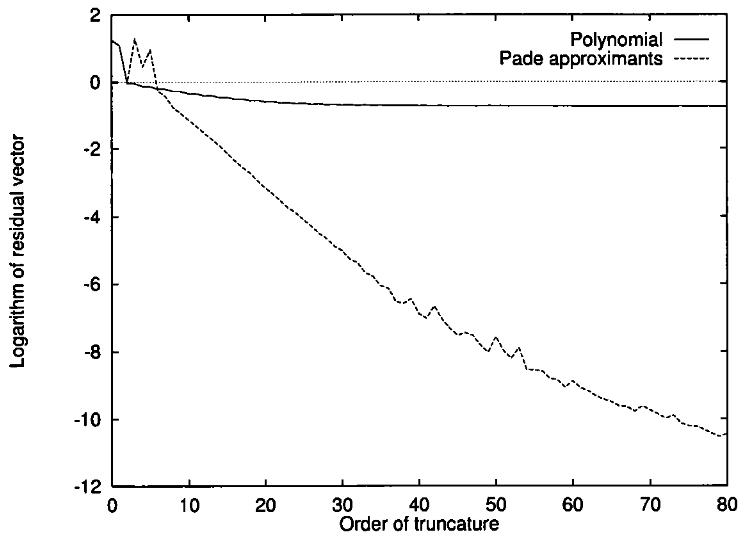


FIG. 2.5: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature, test 5 (N=3, Grille fine : 119072 d.d.l)

2.4.2 Détail du temps de calcul de l'algorithme

Dans ce paragraphe, on donne le détail du temps CPU nécessaire pour le calcul des différentes étapes utilisées dans l'algorithme pour tous les tests cités précédemment.

Dans le cas d'un préconditionnement par la diagonale, les opérations pour chaque étape de l'algorithme à 2-grilles sont :

A l'ordre 0

- Une construction de la matrice K (grille grossière).
- Une triangulation de la matrice K (méthode de Crout).
- Une résolution sur la grille grossière (montée-descente).
- Une multiplication par P : $P.int(q_g)$
- Un calcul du résidu.

A l'ordre i

- Une résolution sur la grille grossière (montée-descente).
- Des produits matrice-vecteur : $R(i) = k_{ll}.q_l(i-1) + k_{lg}.q_g(i-1)$.
- Une multiplication par la matrice C : $r(i) = C.(R(i)) - q_l(i-1) + int(q_g(i-1))$.
- Une multiplication par P : $P.int(q_g)$.
- Un produit matrice-vecteur : $k.r(i)$.
- Un produit par le transposé de P : ${}^tP.k.r(i)$.

Le tableau (2.9), donne le temps CPU en secondes de chaque étape de l'algorithme à 2-grilles. Nous remarquons que pour tous les tests présentés, les opérations qui demandent le plus de temps de calcul sont la construction et la décomposition de la matrice de rigidité de la grille grossière. En effet dans le cas des tests 3 et 5, où le nombre de degrés de liberté est le plus grand, environ 55.5% (test 3) et 16% (test5) du temps total, n'est rien d'autre que le temps CPU de la décomposition de la matrice. Par contre les produits matrices-vecteurs de l'opérateur de prolongement P et du préconditionneur C sont presque négligeables.

Si nous nous restreignons au temps CPU des produits matrices-vecteurs de la matrice de rigidité de la grille fine, nous remarquons que le stockage morse de cette matrice, a permis un gain énorme en temps CPU par rapport au stockage profil (ligne de ciel : LDC).

	test 1	test 2	test 3	test 4	test 5
Ordre 0					
Construction de K	0.25	1.17	4.87	0.21	1.81
Décomposition de K	0.49	11.77	153.8	0.41	23.07
Résolution sur K	0.03	0.34	2.7	0.02	0.91
Produit par P : $P.q_g(0)$	0.01	0.03	0.12	0.02	0.14
Calcul du résidu	0.02	0.1	0.37	0.04	0.34
Temps total à l'ordre 0	0.8	13.41	161.9	0.7	26.3
Ordre i					
Matrice-vecteur : $R(i) = k_{ll}.q_l(i-1) + k_{lg}.q_g(i-1)$	0.02	0.11	0.44	0.04	0.33
Produit par la matrice C : $r(i) = C(R(i)) - q_l(i-1) + int(q_g(i-1))$	0.	0.02	0.07	0.	0.06
Matrice-vecteur : $k.r(i)$	0.02	0.09	0.37	0.04	0.37
Produit par ${}^tP$: ${}^tP.k.r(i)$	0.01	0.03	0.11	0.02	0.11
Résolution sur K	0.04	0.35	2.71	0.03	0.91
Produit par P : $P.int(q_g)$	0.01	0.02	0.12	0.01	0.15
Temps total à l'ordre i	0.1	0.62	3.82	0.14	1.91
Total	3.3	32	276.5	8.96	144.7

TAB. 2.9: *Détail du temps CPU en seconde pour les 5 exemples.*

Le tableau 2.10, donne pour les deux types de stockages, le temps CPU des deux produits matrices-vecteurs utilisés dans l'algorithme à 2-grilles. Les résultats présentés ci-dessus montrent clairement le gain de temps CPU pour le stockage morse par rapport au stockage profil, essentiellement lorsque le nombre de degrés de liberté est grand. Mais même avec un stockage morse, le temps de calcul de ces deux produits reste important par rapport au temps total à l'ordre i.

	Test 2		Test 3		Test 4		Test 5	
	LDC	Morse	LDC	Morse	LDC	Morse	LDC	Morse
$R(i)$	2.09	0.11	20	0.44	0.65	0.04	24	0.33
$k.r(i)$	1.65	0.09	15	0.37	0.56	0.04	19	0.33

TAB. 2.10: Temps CPU en secondes des produits matrices-vecteurs pour les deux types de stockage (ligne de ciel et morse).

2.5 Comparaison avec d'autres méthodes

Le tableau (2.11) donne l'ordre de convergence de l'algorithme proposé (MBP), le nombre de cycles de l'algorithme à deux grilles classiques (MBC) et le nombre d'itérations de la méthode du gradient conjugué préconditionné, soit par la diagonale (GCD) soit par la factorisation incomplète de Cholesky niveau 0 (GCIC).

Pour l'algorithme à deux grilles classique, on a utilisé 3 itérations de lissage. Le lisseur utilisé est la méthode de Jacobi relaxée avec un paramètre de relaxation $\omega = 0.8$. Cette valeur est la valeur optimum conduisant à un minimum de cycles, qui a été déterminée après plusieurs tests numériques.

Le tableau (2.11), illustre la propriété caractéristique des méthodes multigrilles, l'indépendance du facteur de convergence par rapport au pas de discrétisation du maillage : pour les tests 1, 2 et 3, l'ordre de convergence de l'algorithme MBP est de 30, le nombre de cycles de la méthode MBC est de 38. Pour les autres tests, une augmentation de l'ordre de convergence et du nombre de cycles est observée : à peu près 60 pour la méthode MBP et 86 pour la méthode (MBC).

Dans le cas de la méthode du gradient conjugué et pour les deux types de préconditionneurs, le nombre d'itérations augmente avec la taille du problème. Ceci illustre la supériorité des préconditionneurs multigrilles.

	MBP	MBC	GCD	GCIC
	Ordre I	Nombre de cycles	IT	IT
test 1	30	38	574	136
test 2	30	38	1154	274
test 3	30	38	2454	554
test 4	59	86	729	169
test 5	62	86	2198	512

TAB. 2.11: Ordre de convergence et nombre d'itérations des différentes méthodes. MBP : Méthode Bi-grilles Proposé, résolution asymptotique, préconditionneur diagonal. MBC : Méthode Bi-grilles Classique, résolution itérative, préconditionneur de Jacobi relaxé ($\omega = 0.8$), GCD et GCIC représentent respectivement le gradient conjugué préconditionné soit avec la diagonale, soit avec la factorisation incomplète de Cholesky de niveau 0.

Le tableau (2.12), indique le temps CPU de l'algorithme proposé (MBP), celui de l'algorithme à deux grilles classique (MBC) et de la méthode du gradient conjugué préconditionné, soit par la diagonale (GCD) soit par la factorisation incomplète de Cholesky de niveau 0 (GCIC).

Il est logique de comparer ces méthodes, car chacune de ces méthodes utilise un préconditionneur diagonal et un stockage morse des matrices et opérateurs. A préciser

	MBP CPU	MBC CPU	GCD CPU	GCIC CPU
test 1	**3.3**	7.6	13.6	7.1
test 2	**28.4**	47.9	150	64.3
test 3	**276.5**	364	1387	541.1
test 4	**8.8**	24.9	35.5	16
test 5	**144.7**	300	1176	462.4

TAB. 2.12: Temps CPU en secondes des différentes méthodes utilisées.

que la méthode GCIC utilise la factorisation incomplète de Cholesky.

Pour les cinq exemples traités, l'algorithme proposé présente les meilleurs performances en termes de temps CPU total. La méthode à deux grilles classique est très compétitive tant que le nombre de degrés de liberté reste inférieur à environ $40000 d.d.l$ (ceci dépend de l'exemple traité). Au delà, notre algorithme parait le meilleur.

La méthode du gradient conjugué, est la moins efficace non seulement en terme de temps CPU, mais aussi à cause du paramétre σ qu'il faut ajuster pour chaque type de problème quand on augmente le niveau. Le même inconvénient, concerne le paramètre ω introduit dans l'algorithme à deux grilles classique. En effet, la valeur numérique de ω varie pour chaque type de problème et par conséquent, il est nécessaire d'effectuer des tests supplémentaires afin de la trouver. Ces résultats sont tout à fait cohérents avec les acquis de la littérature. On sait que le préconditionneur est la condition essentielle de l'efficacité d'un solveur itératif et que le préconditionnement par une grille grossière est exceptionnellement performant. La présente comparaison entre la méthode du gradient conjugué et les méthodes à deux grilles le confirme. La plus grande efficacité de la méthode proposée MBP, par rapport à la méthode classique MBC, est tout à fait naturelle : les deux préconditionnements sont les mêmes (avec toutefois 3 itérations de lissages pour MBC et une seule pour la nouvelle méthode MBP), mais nous avons introduit une technique de résolution (série + approximants de Padé) plus performante que celle de la Méthode Bi-grille Classique (MBC).

2.6 Comment optimiser la différence entre les deux grilles ?

2.6.1 Introduction

La discussion sur le nombre de divisions N revient à chercher la meilleure grille grossière. Avec la Méthode Bi-grille Classique, l'usage courant est d'avoir un écart minimum (soit $N = 2$) entre deux niveaux de grilles, quitte à utiliser plusieurs niveaux de grilles.

On a bien vu au chapitre précédent que l'algorithme MBC était le plus efficace avec un nombre de division $N = 2$. La question est de savoir si l'introduction d'un solveur efficace (série + Padé) peut modifier ce résultat.

Il est par ailleurs important de tester le nouvel algorithme sur des problèmes variés. On commencera par introduire des propriétés matérielles fortement hétérogènes, puis des efforts plus compliqués que précédemment pour vérifier si l'algorithme resiste à des matrices assez mal conditionnées.

Le cas de maillages irréguliers, jugé très important pour évaluer une méthode multigrille, sera abordé plus loin.

2.6.2 Un premier test sans itération

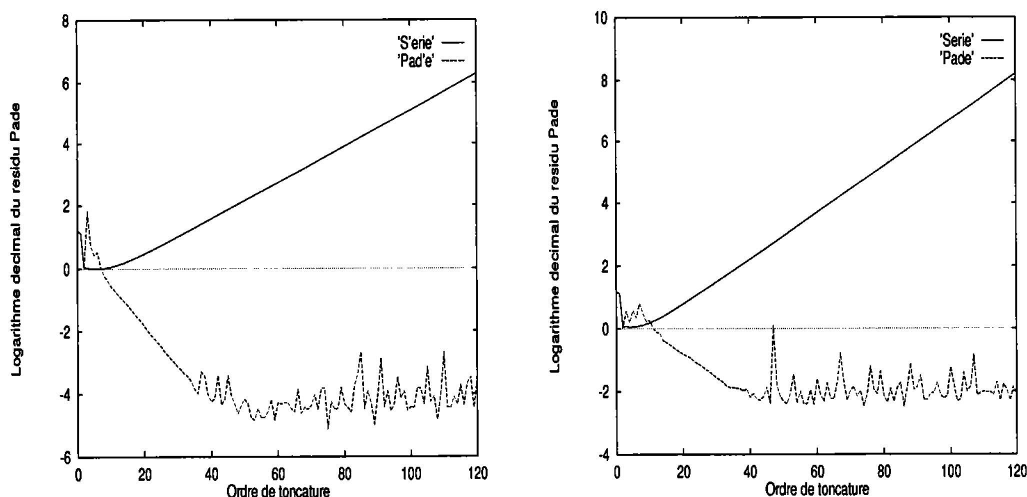
Afin de pousser encore plus loin cette étude sur le comportement de la convergence de l'algorithme à 2-grilles proposé, nous avons pris l'exemple d'une seule grille fine à 65522*d.d.l* avec les mêmes caractéristiques géométriques et mécaniques que dans le cas des exemples précédents.

Deux valeurs du nombre de divisions N seront étudiés : pour $N = 4$ la grille grossière a 4232*d.d.l* et pour $N = 6$, le nombre de degrés de liberté de la grille grossière n'est plus que de 1922*d.d.l*. Dans ces tests, le critère d'arrêt est fixé à 10^{-10}. Le préconditionnement utilisé est diagonal.

La figure (2.6) montre l'évolution de la convergence de l'algorithme à 2-grilles en fonction des ordres de troncature de l'algorithme pour $N = 4$ et $N = 6$. Ces courbes

sont tracées pour deux types de représentations : la représentation en séries et celle utilisant les approximants de Padé. Nous remarquons sur ces courbes que l'utilisation de la représentation polynomiale n'est pas intéressante en terme de convergence, l'algorithme ne converge pas pour les deux exemples étudiés.

Pour l'algorithme utilisant les approximants de Padé, on remarque que pour des ordres de troncature petits, le résidu décroit linéairement puis des oscillations apparaissent quand l'ordre augmente. On a augmenté l'ordre jusqu'à 200 mais sans atteindre la précision demandée. Pour ces deux exemples, le préconditionnement par la diagonal, n'apporte pas probablement l'information suffisante permettant à l'algorithme de converger, en tous cas pas avec un seul calcul de série.



(a) Préconditionneur diagonal : $N = 4$. (b) Préconditionneur diagonal : $N = 6$.

FIG. 2.6: Logarithme du résidu Padé en fonction des ordres de troncature, pour $N = 4$, la grille grossière : 4232 d.d.l, pour $N = 6$, la grille grossière : 1922 d.d.l.

2.6.3 Technique itérative d'ordre élevé

On a observé le même phénomène pour d'autres valeurs de N. Nous pouvons alors conclure que pour améliorer la convergence de l'algorithme proposé, il est préférable, soit de trouver d'autres préconditionneurs plus efficaces comme on le verifiera plus loin, soit d'utiliser l'algorithme à 2-grilles proposé dans une procédure itérative comme

le montre la figure (2.7).

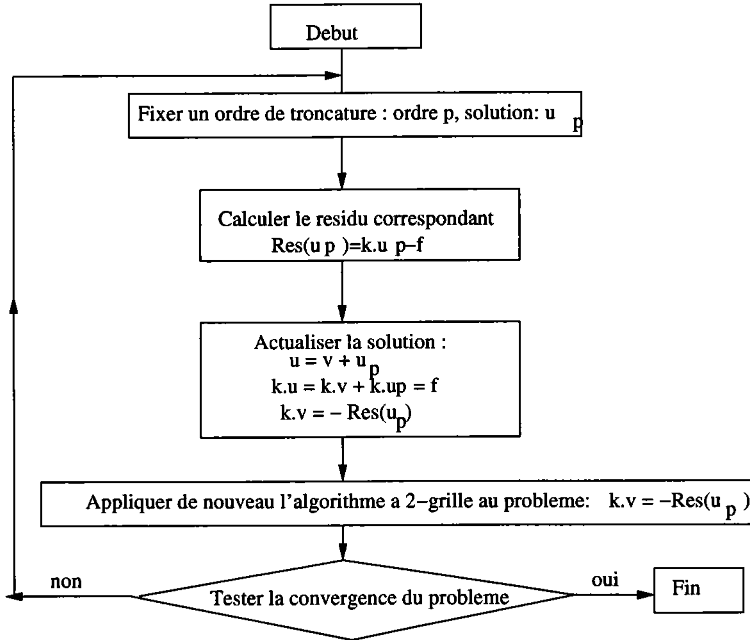


FIG. 2.7: Algorithme itératif proposé

Dans cette procédure itérative on résoud le système $k.u = f$ de la manière suivante : on fixe a priori l'ordre de troncature maximal, noté I. L'approximation polynomiale à l'ordre I fixé s'écrit sous la forme :

$$u = u_0 + u_1 + ... + u_I \tag{2.30}$$

l'approximation de Padé correspondante est :

$$u_p = \sum_{i=0}^{i=I} c_i.u_i \tag{2.31}$$

On note cette approximation u_p^1, le résidu correspondant noté Res^1 s'écrit sous la forme suivante :

$$Res^1 = k.u_p^1 - f \tag{2.32}$$

La solution du problème $k.u = f$ est alors cherchée sous la forme suivante :

$$u = u_p^1 + v \tag{2.33}$$

où v est la correction associée à l'approximation u_p^1. La correction v satisfait alors l'équation :

$$k.v = -Res^1 \tag{2.34}$$

Cette nouvelle équation est résolue par la méthode à 2-grilles avec l'ordre maximal I fixé. L'approximation obtenue est notée : v_p^2. La solution est alors :

$$u_p^2 = u_p^1 + v_p^2 \tag{2.35}$$

Cette solution est une nouvelle solution d'essai de l'équation $k.u = f$ avec un nouveau résidu noté : Res^2. On vient ainsi de définir un processus itératif, qui conduit après n itérations à la solution approchée suivante :

$$u = u_p^1 + u_p^2 + ... + u_p^n \tag{2.36}$$

où u_p^n désigne l'approximation obtenue avec l'algorithme bi-grilles + série + Padé utilisé pour résoudre le problème :

$$k.u_p^n = -Res^n \qquad \text{où} \qquad Res^n = k.(u_p^1 + v_p^2 + ... + v_p^{n-1}) - f \tag{2.37}$$

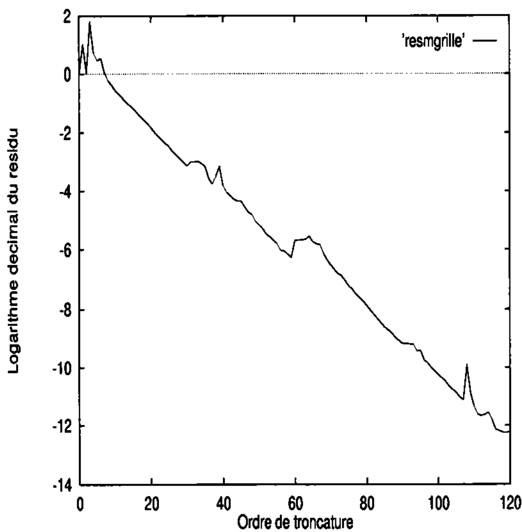
2.6.4 Premiers tests avec la technique itérative

Sur la figure (2.8), on représente le logarithme décimal de la norme du résidu (Padé) en fonction de l'ordre de troncature de l'algorithme itératif pour les mêmes valeurs de division N : $N = 4$ et $N = 6$, pour ces deux valeurs, l'ordre de troncature maximal I est fixé à 30. A noter que pour des facilités de représentations, nous avons additionné les ordres de troncature de chaque itération : ainsi l'ordre 60 de la figure (2.8) correspond en fait à l'ordre 30 de la deuxième itération, l'ordre 90 à l'ordre 30 de la troisième itération et ainsi de suite.

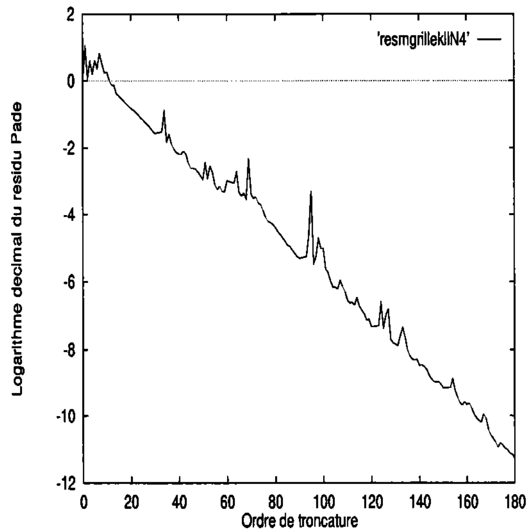
Ces courbes montrent que le résidu décroît d'une façon monotone en fonction de l'ordre de troncature I. Pour $N = 4$, l'algorithme bi-grilles utilisant les approximants de Padé permet d'obtenir la convergence en 3 itérations. Pour $N = 6$, la convergence

est obtenue à l'ordre 27 de la cinquième itération. On peut donc faire converger l'algorithme en fixant un ordre maximal et en appliquant l'algorithme de façon itérative, même avec $N = 4$ ou $N = 6$.

Nous remarquons aussi que le nombre de vecteurs (calculés à l'ordre fixé multiplié par le nombre d'itérations, auquel on ajoute l'ordre de convergence de la dernière itération) augmente, lorsque le nombre de divisions N augmente.



(a) Résultats obtenus par application de l'algorithme itératif avec un un ordre de troncature fixé à 30, préconditionneur diagonal, grille grossière : 4232 d.d.l, $N = 4$



(b) Résultats obtenus par application de l'algorithme itératif avec un un ordre de troncature fixé à 30, préconditionneur diagonal, grille grossière : 1922 d.d.l, $N = 6$

FIG. 2.8: Logarithme du résidu Padé en fonction des ordres de troncature, l'ordre de troncature est fixé à 30.

Pour ces deux exemples, on a fixé l'ordre à 30. Ce choix n'est pas unique et par conséquent on peut choisir d'autres valeurs autres la valeur 30.

La question qui se pose alors : quelle est la meilleure façon de fixer l'ordre, sans trop augmenter le nombre d'itérations et le temps CPU "total" de l'algorithme ?

2.6.5 Influence du nombre de divisions N

Pour cela, on propose dans un premier temps d'étudier l'influence du nombre de divisions N sur la convergence de l'algorithme à 2-grilles pour différentes valeurs de

l'ordre de troncature à fixer.

Sur la figure (2.9), on représente le nombre de vecteurs pour avoir un résidu inférieur à 10^{-10} en fonction de la valeur N pour un ordre fixé à 30.

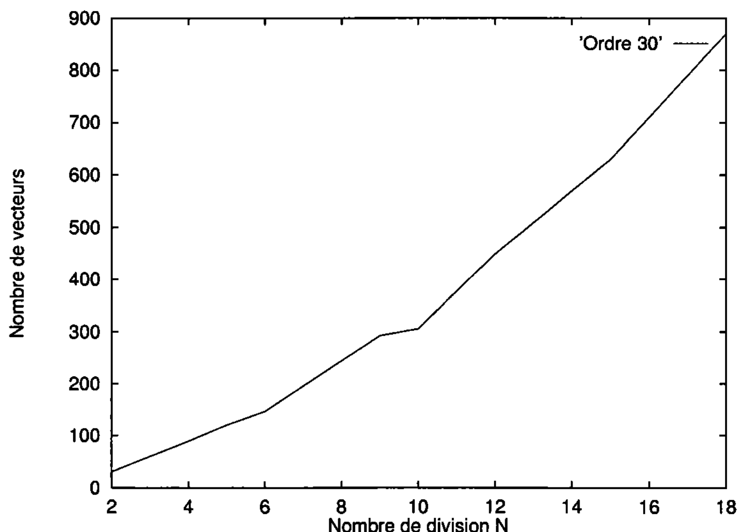


FIG. 2.9: Evolution du nombre de vecteurs en fonction du nombre de division N pour un ordre de troncature fixé à 30.

Ce résultat montre que le nombre de vecteurs à calculer augmente lorsqu'on augmente la valeur de N. On peut noter que dans le cas où on fixe $I = 20$, des tests numériques ont montré qu'à partir de la valeur $N = 6$, la résolution du problème linéaire par la méthode à 2-grilles coûte très cher en termes de nombre d'itérations et par conséquent à partir de cette valeur, le fait d'augmenter la valeur de N n'a plus de sens. Pour des ordres de troncature fixés à 30 et 40, l'algorithme converge quelle que soit la valeur du nombre de divisions N.

Dans un deuxième temps, nous avons calculé le temps CPU total pour plusieurs valeurs de N ($2 \leq N \leq 18$) par les deux méthodes suivantes : la méthode à 2-grilles proposée pour un ordre de troncature fixé à 30 et la méthode à 2-grilles classique (lisseur de Jacobi).

Le tableau (2.13) donne le nombre d'itérations nécessaire à la convergence de l'algorithme itératif (2.7), l'ordre de convergence et le temps CPU total pour chaque valeur de N.

Grille grossière	N	ITER	Ordre de convergence	CPU total (s)
16562	2	1	30	83
7442	3	2	30	63
4232	4	3	29	66
2738	5	4	30	81
2738	6	5	27	103
882	9	10	23	199
722	10	11	6	208.5
512	12	15	7	286.3
338	15	21	30	428
242	18	29	30	582

TAB. 2.13: Degrés de liberté de la grille grossière, grille fine fixée à 65522*d.d.l.* Nombre de division N, nombre d'itérations : ITER, ordre de convergence et le temps CPU total en secondes de la méthode à 2-grilles proposée. Résolution itérative avec $I = 30$, préconditionneur diagonal.

On observe sur le tableau 2.13 la présence d'un minimum de temps CPU dans le cas $N = 3$, c'est-à-dire une décroissance du temps CPU entre $N = 2$ et $N = 4$, puis une remontée entre $N = 4$ et $N = 18$. Pour vérifier cela, nous avons effectué un autre test, en fixant l'ordre de troncature à 20 et 40.

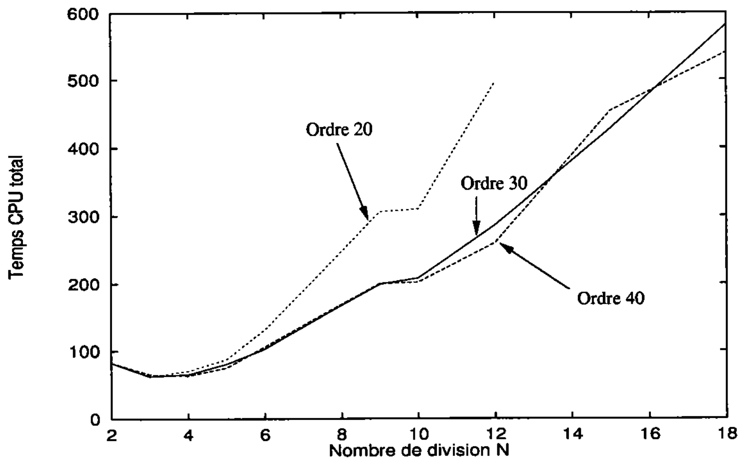


FIG. 2.10: Evolution du temps CPU (s) en fonction du nombre de division N pour des ordres de troncature fixés à 20, 30 et 40.

La figure (2.10) montre la présence d'un minimum de temps CPU total de l'algorithme itératif pour $I = 20$ et $I = 40$. On observe donc un comportement semblable pour un ordre de troncature fixé à 30 et 40, bien que la diminution du temps CPU dans ces deux cas, soit moins importante que lorsqu'on fixe l'ordre à $I = 20$.

2.6.6 Coût de calcul des approximants de Padé

L'augmentation de l'ordre de troncature entraine des temps CPU très importants lors du calcul des approximants de Padé. En effet jusqu'à présent, dans toutes les études relatives aux techniques de calcul des approximants de Padé, les temps de construction de ces derniers étaient négligeables, car l'ordre et surtout le nombre de degrés de liberté n'étaient pas aussi grands que dans cette étude. Lorsque le nombre de d.d.l de la grille fine est supérieur à 100000, les temps de construction (incluant l'orthonormalisation ainsi que le calcul des dénominateurs des fractions rationnelles) peuvent devenir grands. C'est pourquoi nous proposons dans les figures (2.11) et (2.12), une étude de ces temps de construction des approximants de Padé.

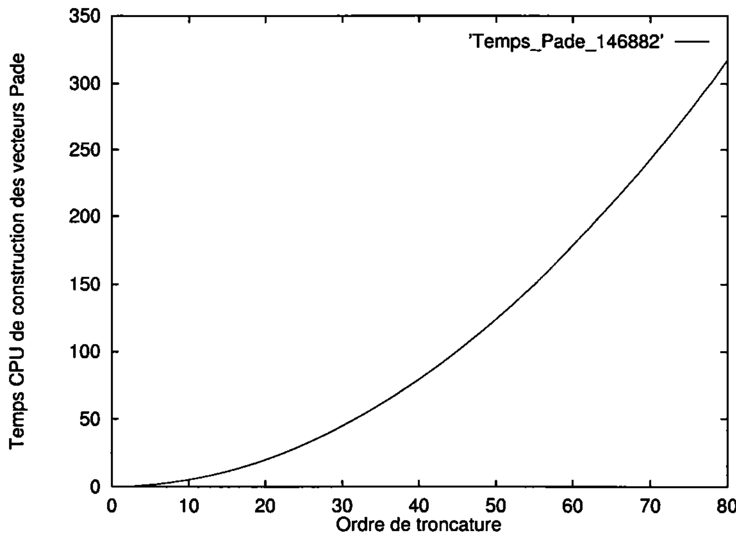


FIG. 2.11: Evolution du temps CPU en secondes de construction des vecteurs Padé en fonction de l'ordre de troncature de l'algorithme à 2-grilles (N=3, grille grossière : 16562 d.d.l, grille fine : 146882 d.d.l, convergence obtenue à l'ordre 63)

La courbe de la figure (2.11) montre que le temps CPU pour construire et calculer les approximants de Padé augmente fortement en fonction des ordres de troncature. D'autre part la figure (2.12) montre que ce temps semble ne plus être négligeable par rapport au temps CPU de l'algorithme pour des ordres relativement élevés. En effet cette figure montre que lorsque l'ordre de troncature est égal à 70, la construction des approximants de Padé demande 0.8 fois le temps de calcul de l'algorithme pour

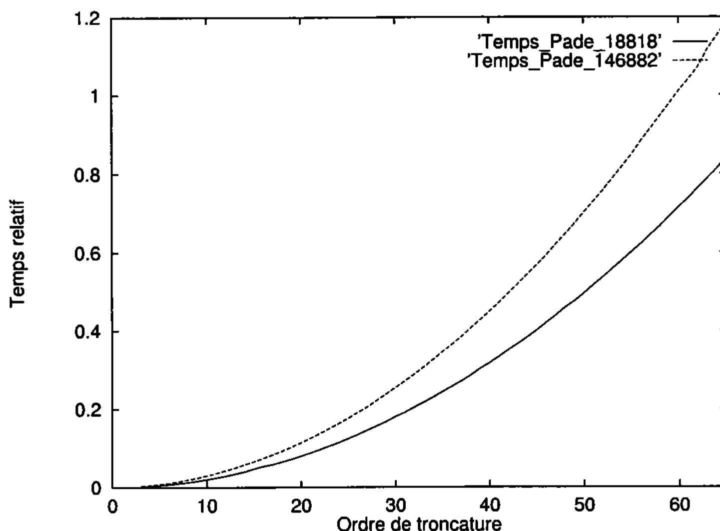


FIG. 2.12: Evolution du temps relatif en fonction de l'ordre de troncature (temps relatif = temps CPU de construction des Padé/temps CPU de l'algorithme à 2-grilles) pour deux exemples : premier exemple, N=3, grille fine : 18818 d.d.l et deuxième exemple, N=3, grille fine : 146882 d.d.l, convergence de l'algorithme obtenue à 63)

le problème à petit nombre de d.d.l (exemple à 18818 d.d.l) alors que pour l'exemple à 146882 d.d.l, ce temps est 1.2 fois le temps de calcul de l'algorithme. Cela veut dire, que pour l'exemple à grand nombre de degrés de liberté, le temps de calcul des approximants de Pade coûte beaucoup plus cher que le temps de calcul de l'algorithme lui même.

Cette remarque sur le temps CPU de construction des approximants de Padé montre tout l'intérêt de fixer des ordres de troncature petits (temps de construction des Padé faible) mais à condition que l'algorithme converge. Au vu de ces remarques, il semble plus intéressant de fixer un ordre de troncature entre 20 et 40 et d'utiliser un algorithme itératif pour des valeurs de $N > 3$.

2.6.7 Comparaison avec la méthode à deux grilles classique

Dans la partie précédente, nous avons montré que la méthode à 2-grilles proposée présente un minimum de temps CPU pour la valeur $N = 3$, et d'autre part que le nombre de vecteurs, et donc le temps CPU total, dépend de la valeur de N. Il est intéressant maintenant de comparer la méthode à 2-grilles proposée avec la méthode

à 2-grilles classique.

Les figures (2.13) et (2.14) montrent qu'en faisant augmenter la valeur de N, le nombre de cycles nécessaires à la convergence et le temps CPU total de l'algorithme à 2-grilles classique augmentent. Pour la valeur $N = 2$, nous obtenons un bon compromis entre le nombre de cycles et le temps CPU pour la résolution. La valeur optimum pour la méthode à 2-grilles classique utilisant la méthode de Jacobi comme lisseur est $N = 2$.

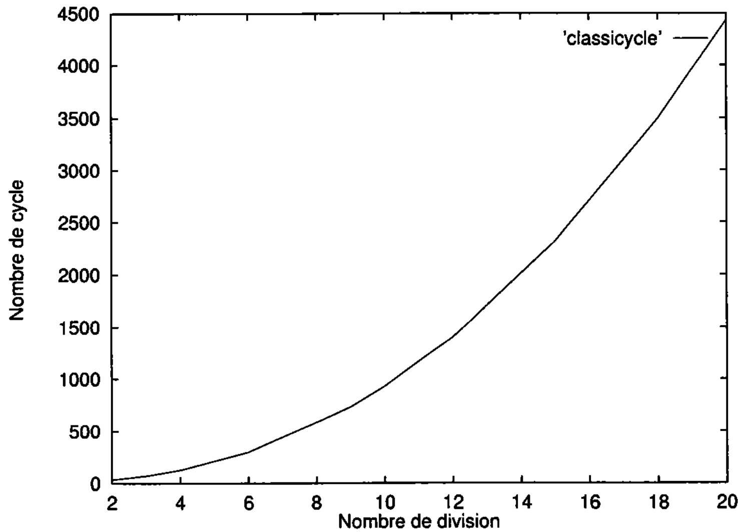


FIG. 2.13: Evolution du nombre de cycles de la méthode à 2-grilles classique en fonction du nombre de division N, lisseur de Jacobi, $\omega = 0.8$, résolution itérative

On peut remarquer d'après les résultats de la figure (2.14), que la méthode proposée est plus performante que la méthode à 2-grilles classique et ceci quelle que soit la valeur de N. Bien entendu, seules les petites valeurs de N seront intéressantes en pratique. Nos résultats confirment qu'avec la méthode classique, il faut conserver la plus grande proximité entre deux niveaux de grilles, soit $N = 2$. En revanche avec notre algorithme, on peut se permettre une plus grande différence, s'il est utilisé de façon itérative : on a constaté au tableau 2.13 qu'il y avait une efficacité maximale pour $N = 3$ ou $N = 4$ et des résultats presque identiques entre $N = 5$ et $N = 2$.

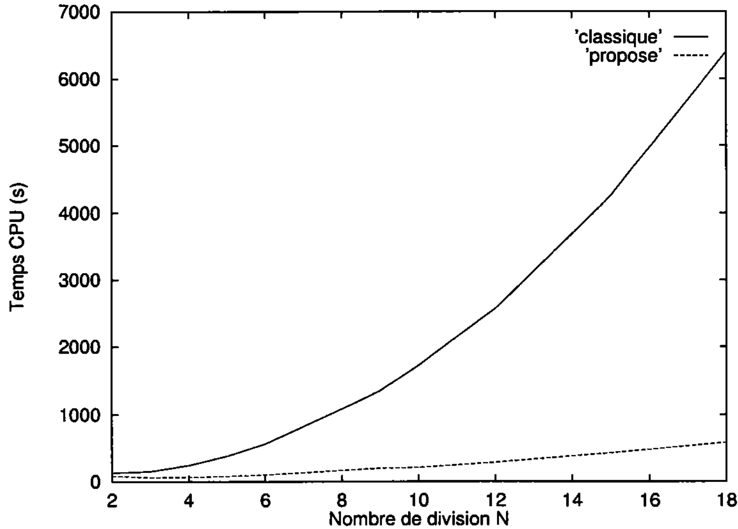


FIG. 2.14: Evolution du temps CPU total en secondes de la méthode à 2-grilles classique (Jacobi, $\omega = 0.8$) et de la méthode proposée (ordre de troncature fixé à 30) en fonction du nombre de division N, résolution itérative

2.7 Autres évaluations de la robustesse de l'algorithme

2.7.1 Test sur l'hétérogénéité de la structure

Pour tester l'influence du conditionnement sur la vitesse de convergence de l'algorithme, nous considérons l'exemple d'une grille composée de deux matériaux comme le montre la figure (2.15) de coefficients d'Young très différents, $E1 = 200000Mpa$ et $E2 = 200Mpa$. L'idée de ce test est que si la vitesse de convergence de l'algorithme change pour le matériau hétérogène, c'est que le "préconditionneur" C gouverne la vitesse de convergence de l'algorithme. Par contre si l'ordre de convergence reste le même ou change très peu, cela veut dire que l'algorithme proposé avec préconditionnement diagonal fonctionne aussi bien pour un matériau homogène que pour un matériau très hétérogène.

Sur le tableau (2.14), nous reportons l'ordre de troncature nécessaire à la convergence de l'algorithme, pour les deux types de matériaux.

Cette étude nous permet d'établir que dans le cas d'un préconditionneur diagonal, l'hétérogénéité de la structure n'a aucune influence sur la vitesse de convergence de

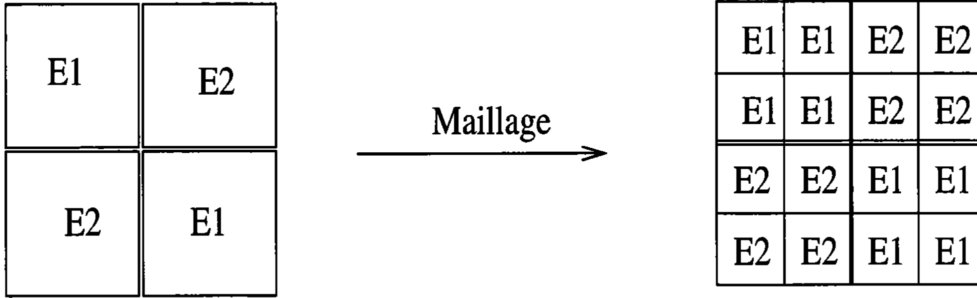


FIG. 2.15: Maillage de la plaque avec deux matériaux différents

$C_{ll} = \frac{1}{k_{ll}}$	Ordre de convergence de l'algorithme proposé	
	Matériau homogène	Matériau hétérogène
Grille grossière : 2178$d.d.l$ Grille fine : 8450 d.d.l	30	31
Grille grossière : 8450$d.d.l$ Grille fine : 33282 d.d.l	30	31
Grille grossière : 33282$d.d.l$ Grille fine : 132098 d.d.l	30	31

TAB. 2.14: Ordre de convergence obtenu pour les matériaux homogène et hétérogène

l'algorithme : ordre 30 pour le matériau homogène et l'ordre 31 pour le matériau hétérogène. En d'autres termes, l'algorithme à 2-grilles fonctionne aussi bien pour un matériau homogéne que pour un matériau hétérogène.

2.7.2 Test sur les seconds membres

Pour tester l'influence des seconds membres sur la convergence de l'algorithme proposé, nous considérons l'exemple d'une plaque dont les caractéristiques géométriques et mécaniques sont données sur la figure (2.16). La plaque est encastrée en deux points et chargée de l'autre par deux forces de même module.

Les tableaux (2.15) et (2.16) indiquent l'ordre de troncature nécessaire à la convergence de l'algorithme proposé, le résidu utilisant les approximants de Padé et le résidu utilisant l'approximation polynomiale (série).

Ces valeurs ont été obtenues sur la plaque de la figure (2.16) pour deux exemples : pour le premier, le nombre de degrés de liberté de la grille grossière est de 6402$d.d.l$, celui de la grille fine est 25090$d.d.l$. Dans le deuxième exemple la grille grossière est

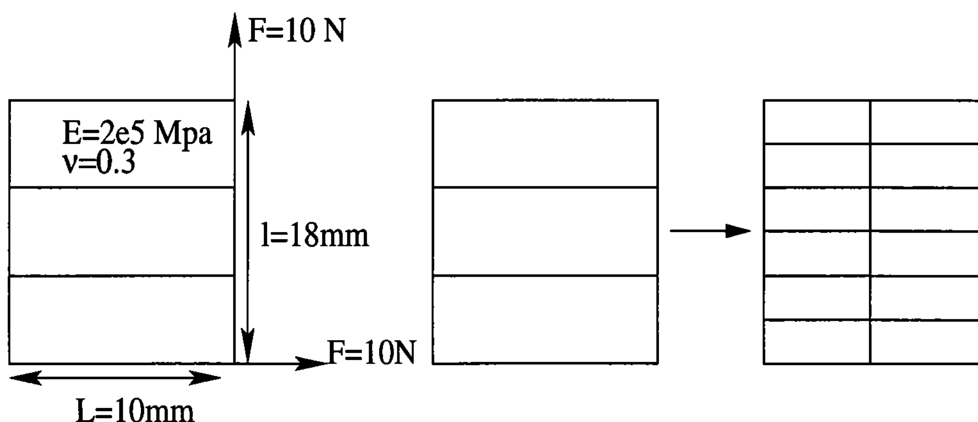


FIG. 2.16: Exemple de la plaque soumise au chargement F

de $25090 d.d.l$ et celle de la grille fine est de $99570 d.d.l$. La convergence de l'algorithme est obtenue lorsqu'une réduction de 10^{-10} de la norme du résidu Padé sur la grille fine est obtenue.

Remarquons que la vitesse de convergence de l'algorithme est indépendante du pas de discrétisation du maillage : l'ordre de convergence est égale à 51 pour les deux exemples étudiés. Ces résultats montrent que les seconds membres influencent relativement peu sur la vitesse de convergence de l'algorithme.

Ordre	3	10	20	30	40	50	51
Padé/F	1.57	2.8610^{-2}	1.1910^{-4}	5.8810^{-7}	7.910^{-9}	1.3610^{-9}	7.0410^{-10}
Série/F	9.56	10.05	27.18	85.66	282.7	955.7	1080.6

TAB. 2.15: *Résidus obtenus pour différents ordres de troncature :* **Exemple 1**

Ordre	3	10	20	30	40	50	51
Padé/F	1.57	2.8710^{-2}	1.1910^{-4}	5.8910^{-7}	9.7610^{-9}	1.0010^{-9}	8.84^{-10}
Série/F	9.55	9.98	26.81	83.99	275.47	925.75	1046.03

TAB. 2.16: *Résidus obtenus pour différents ordres de troncature :* **Exemple 2**

2.8 Conclusion

Dans ce chapitre, un nouveau type d'algorithme à 2-grilles a été proposé. Celui-ci est basé sur l'association d'une transformation d'homotopie, d'une technique de perturbation et des approximants de Padé.

Nous avons montré sur plusieurs exemples que cette nouvelle méthode converge rapidement même s'il y a une grande différence entre la grille grossière et la grille fine. Sur ces exemples nous avons pu voir que cette nouvelle classe d'algorithmes à 2-grilles est beaucoup plus efficace que les méthodes itératives classiques ou que les algorithmes à 2-grilles classiques, pour les problèmes avec un grand nombre de degrés de liberté.

Ainsi le couplage d'une approche multigrille et des techniques de perturbation, qui à notre connaissance n'a jamais été présenté dans la littérature, est prometteur au vu des résultats obtenus.

Par rapport aux approches bi-grilles classiques, notre contribution est l'introduction d'un solveur d'ordre élevé (série + Padé), qui améliore leur efficacité. En particulier, nous avons montré qu'avec cette nouvelle méthode, on peut se permettre une plus grande différence entre deux grilles successives, et que les résultats obtenus sont meilleurs avec un nombre de divisions $N = 3$ ou $N = 4$ qu'avec $N = 2$.

Nous avons également montré que ces nouvelles techniques étaient efficaces lorsque l'on utilisait les approximants de Padé. Les résultats numériques ont également montré que, comme pour les méthodes multigrilles classiques, la vitesse de convergence des algorithmes proposés était independante du pas de discréritisation de maillage.

Néanmoins, la méthode présentée dans ce chapitre reste relativement rustique. Pour être vraiment compétitive, elle devrait être améliorée sur plusieurs points : prise en compte de plusieurs niveaux de grilles, ou de maillages non-structurés, utilisation de préconditionneurs plus performants : dans le chapitre suivant, nous allons nous intéresser aux deux derniers points.

CHAPITRE 3

Méthode multigrille d'ordre élevé avec divers préconditionneurs

3.1 Introduction

On a montré au chapitre précédent l'intérêt d'une technique de résolution d'ordre élevé, associant homotopie, calcul de série et approximants de Padé.

Par rapport à la technique multigrille classique avec le même préconditionneur diagonal, les performances sont meilleures, avec un rapport en temps CPU de l'ordre de deux dans les cas à grand nombre de degrés de liberté. En particulier, le nouvel algorithme ne nécessite qu'une opération de lissage sur la grille fine (au lieu de 3 ou 4), il converge bien et il est même plus rapide avec une grille grossière plus petite qu'habituellement ($N = 3$ ou 4). De plus, il ne fait pas apparaître de paramètre à ajuster comme le paramètre de sur-relaxation ω pour les méthodes classiques. Mais dans les méthodes multigrilles classiques, on utilise souvent d'autres techniques de préconditionnement sur la grille fine, que la méthode de Jacobi.

Dans ce chapitre nous cherchons à coupler d'autres préconditionneurs avec une résolution par homotopie-calcul de série-approximants de Padé. En particulier nous nous intéresserons à la factorisation incomplète de Cholesky de niveau 0, notée $IC(0)$, qui a beaucoup d'avantages : la mémoire nécessaire reste faible, de même que le coût de calcul, puisque la matrice factorisée est aussi creuse que la matrice initiale. La difficulté pour introduire de tels préconditionneurs est que la décomposition en un

problème grossier (équation 2.6) est un problème sur "les noeuds locaux", (l'équation 2.7) n'est pas compatible avec le préconditionneur $IC(0)$, qui couple les noeuds locaux et les noeuds grossiers. L'idée est alors d'écrire deux fois le problème grossier (2.6), en introduisant un multiplicateur de Lagrange, puis de définir une nouvelle homotopie à partir du problème ainsi modifié.

On présentera également une technique simple pour résoudre les cas de maillages non structurés, ce qui est très intéressant pour les problèmes à géomètrie complexes.

3.2 Une nouvelle classe de méthode multigrille d'ordre élevé

Dans le chapitre précédent, nous avions décomposé le problème $k.q = f$ en un problème sur la grille grossière et un problème sur les noeuds locaux.

Ici, nous cherchons à réécrire le problème sous la forme d'une équation sur la grille grossière, d'une seconde sur toute la grille fine (et non sur les seuls noeuds locaux) et d'une équation de liaison entre les deux. Ainsi, nous doublerons l'équation sur la grille grossière, en introduisant un multiplicateur de Lagrange. Ensuite diverses transformations d'homotopies permettront de définir des techniques de résolution d'ordre élevé.

3.2.1 Une technique pour doubler une équation linéaire

Dans un but pédagogique, nous présentons d'abord la technique de doublement d'une équation par multiplicateur de Lagrange, dans le cas d'une équation linéaire dans R^n, où k est une matrice définie positive :

$$k.u = f \tag{3.1}$$

Au lieu de l'inconnue $u \in R^n$, on introduit deux inconnues $u, v \in R^n$ qui seront reliées par la condition de liaison $u = v$. Associant un multiplicateur $\lambda \in R^n$ à cette liaison

$u = v$, nous pouvons réécrire le système (3.1) sous la forme :

$$\delta u.(k.u - f) + \delta v.(k.v - f) - \delta[\lambda.(u - v)] = 0 \quad \forall \delta u, \delta v, \delta \lambda \in R^n \tag{3.2}$$

ce qui est équivalent au système suivant :

$$\begin{cases} k.u - \lambda & = f & (a) \\ k.v + \lambda & = f & (b) \\ u - v & = 0 & (c) \end{cases} \tag{3.3}$$

Notons que les équations (3.3-a) et (3.3-b) impliquent :

$$\begin{cases} k.(u + v) & = 2f & (a) \\ \\ k.(u - v) & = 2\lambda & (b) \end{cases} \tag{3.4}$$

Les équations (3.4) montrent que la solution (u, v, λ) de (3.3) vérifie les relations suivantes :

$$u = v = k^{-1}.f \quad \text{et} \quad \lambda = 0 \tag{3.5}$$

Le système doublé (3.3) est donc équivalent au système initial (3.1). Ensuite on modifie l'équation $u - v = 0$ du système (3.3) par une sorte de pénalisation de la manière suivante :

$$C.(u - v) = -\lambda \tag{3.6}$$

où C est une matrice. Le nouveau système (3.3) se réécrit alors sous la forme :

$$\begin{cases} k.u - \lambda & = f & (a) \\ k.v + \lambda & = f & (b) \\ C.(u - v) & = -\lambda & (c) \end{cases} \tag{3.7}$$

où on a remplacé l'équation $u - v = 0$ par l'équation de pseudo-pénalisation (3.6). Nous pouvons montrer que (3.7) reste en général équivalente à (3.1). En effet, si on additionne les équations (3.4-b) et (3.6), on obtient l'équation suivante :

$$k.(u - v) + 2C.(u - v) = 0 \tag{3.8}$$

ce qui implique :

$$(k + 2C).(u - v) = 0 \tag{3.9}$$

Si C est une matrice positive, alors $k + 2C$ l'est aussi et on a alors : $u = v = k^{-1}.f$ et $\lambda = 0$. Donc le changement de l'équation (3.3-c) en (3.7-c) n'a pas modifié la solution du problème, c'est pourquoi nous parlerons de pseudo-pénalisation. Si par exemple $C = \alpha k$, où α est un réel quelconque, les équations (3.3-a), (3.3-b) et (3.6) se réécrivent, après élimination du multiplicateur de Lagrange, sous la forme :

$$\left\{ \begin{array}{ll} (1-\alpha)k.u + \alpha k.v & = f \\ \alpha k.u + (1-\alpha)k.v & = f \end{array} \right. \tag{3.10}$$

Ce système admet une solution unique si et seulement si la matrice (3.11) est inversible, c'est-à-dire si $\alpha \neq 1/2$.

$$\begin{bmatrix} (1-\alpha)k & \alpha k \\ \alpha k & (1-\alpha k)k \end{bmatrix} \tag{3.11}$$

Pour la suite, on retiendra que la matrice de pseudo - pénalisation ne doit pas être trop grande. C'est une technique analogue que l'on va adopter pour résoudre un problème d'élasticité linéaire dans le cas d'une méthode multigrille.

3.2.2 Introduction d'un système augmenté doublant l'équation sur la grille grossière

Dans cette section on propose une nouvelle technique à deux grilles pour résoudre le système linéaire suivant :

$$k.q = f \tag{3.12}$$

où k représente la matrice de rigidité d'une structure discrétisée par élément finis et q est le vecteur déplacement des noeuds de la structure soumise au chargement extérieur f.

Pour cela, on considère deux grilles : une grille fine correspondant à la discrétisation du problème (3.12) notée Ω_h avec n degrés de liberté et une grille grossière notée Ω_H

avec N degrés de liberté ($N < n$). L'inconnue sur la grille grossière sera notée $Q \in R^N$ et elle sera supposée indépendante de q. Les informations entre les deux grilles sont transmises par deux opérateurs de passage :

1. L'opérateur de prolongement $P : R^N \to R^n$, comme aux chapitres précédents, qui en pratique sera défini par interpolation linéaire.

2. L'opérateur de restriction sur le maillage grossier, $S : R^n \to R^N$.

Si on décompose l'inconnue q en une inconnue sur les noeuds globaux q_g et une inconnue sur les noeuds locaux q_l :

$$q = \left\{ \begin{array}{c} q_g \\ q_l \end{array} \right\} \tag{3.13}$$

alors l'opérateur de restriction S se définit à partir de l'identité I_N dans R^N par :

$$S = [I_N, 0] \tag{3.14}$$

Cet opérateur permet de définir la liaison entre les deux inconnues Q et q :

$$S.q - Q = 0 \tag{3.15}$$

Comme au paragraphe précédent, on relaxe cette condition en introduisant un multiplicateur de Lagrange $\Lambda \in R^N$ et on écrit un principe variationnel à trois inconnues $(Q, q, \Lambda) \in (R^N \times R^n \times R^N)$:

$$\delta(P.Q).(k.q - f) + \delta q.(k.q - f) + \delta\{\Lambda(S.q - Q)\} = 0 \tag{3.16}$$

soit le système augmenté suivant dans $(R^N \times R^n \times R^N)$:

$$\left\{ \begin{array}{lll} {}^tP.(k.q - f) - \Lambda & = 0 & (a) \\ k.q +^t S.\Lambda & = f & (b) \\ S.q - Q & = 0 & (c) \end{array} \right. \tag{3.17}$$

On notera que le système augmenté (3.17) est équivalent à l'équation de départ (3.12) pourvu que l'opérateur de prolongement laisse invariant les d.d.l de la grille grossière, ce qui implique ${}^tP.{}^tS = I^N$. En effet, appliquant la réduction ${}^tP$ à (3.17-b), on en déduit l'équation suivante :

$${}^tP.(k.q - f) + \Lambda = 0 \tag{3.18}$$

qui après combinaison avec l'équation (3.17-a), implique $\Lambda = 0$ et (3.12).

Le système augmenté (3.17) est ensuite modifié de deux manières. Premièrement, on introduit dans l'équation de liaison (3.17-c) une pseudo-pénalisation associée à une matrice carrée C dans R^N. Ensuite, au lieu des variables q et Q, on utilisera de préférence $Q \in R^N$ et la correction définie par :

$$v = q - P.Q \tag{3.19}$$

Cela conduit au système suivant :

$$\begin{cases} K.Q +^t P.k.v - \Lambda & = F \\ k.(P.Q + v) +^t S.\Lambda & = f \\ C.(S.q - Q) & = \Lambda \end{cases} \tag{3.20}$$

où on a introduit la matrice réduite K et le vecteur réduit F :

$$\begin{cases} K =^t P.k.P \\ F =^t P.f \end{cases} \tag{3.21}$$

3.2.3 Homotopie et technique de perturbation

Comme dans l'algorithme précédent, cette nouvelle technique à deux grilles utilisera une méthode directe (méthode de Crout) pour résoudre le problème posé sur la grille grossière (contenant beaucoup moins de degrés de liberté que la grille fine), et la technique d'homotopie pour résoudre la deuxième et la troisième équation du système (3.20). Cela consiste à introduire un problème artificiel dépendant d'un paramètre $0 \leq \varepsilon \leq 1$ de la manière suivante :

$$\begin{cases} \varepsilon C.(S.q - Q) = \Lambda \\ (1-\varepsilon)k^*.v + \varepsilon[k.q - f] +^t S.\lambda = 0 \end{cases} \tag{3.22}$$

où k^* est un opérateur arbitraire, c'est l'interêt de cette technique de pouvoir mettre un opérateur quelconque, où donc la séparation des variables q_g et q_l n'est plus automatique.

Finalement, le problème initial à résoudre (3.12) est équivalent au problème suivant :

$$\begin{cases} \varepsilon C.(S.q - Q) = \Lambda \\ K.Q +^t P.k.v - \Lambda = F \\ (1-\varepsilon)k^*.v + \varepsilon[k.q - f] +^t S.\Lambda = 0 \\ q = v + PQ \end{cases} \tag{3.23}$$

Comme dans le chapitre précédent, on utilisera la technique de perturbation pour résoudre le sysème (3.23). On cherche une représentation paramétrique des vecteurs inconnus v, Q, q et Λ sous la forme d'un développement en séries entières tronquées à l'ordre I par rapport au paramètre ε :

$$q = \sum_{i=0}^{I} \varepsilon^i q(i) \tag{3.24}$$

$$Q = \sum_{i=0}^{I} \varepsilon^i Q(i) \tag{3.25}$$

$$\Lambda = \sum_{i=0}^{I} \varepsilon^i \Lambda(i) \tag{3.26}$$

$$v = \sum_{i=0}^{I} \varepsilon^i v(i) \tag{3.27}$$

En injectant ces expressions dans (3.23), et après identification terme à terme suivant les puissances de ε, on obtient une succession de problèmes à résoudre :

A l'ordre 0 :

$$\begin{cases} \Lambda(0) & = 0 \\ K.Q(0) & = F \\ v(0) & = 0 \\ q(0) & = P.Q(0) \end{cases} \tag{3.28}$$

A l'ordre 1 :

$$\begin{cases} \Lambda(1) & = C.[S.q(0) - Q(0)] = 0 \\ k^*.v(1) & = -[k.q(0) - f] \\ K.Q(1) & = -^tP.k.v(1) \\ q(1) & = v(1) + P.Q(1) \end{cases} \tag{3.29}$$

A l'ordre 2 :

$$\begin{cases} \Lambda(2) & = C.[S.q(1) - Q(1)] \\ k^*.v(2) & = k^*.v(1) - k.q(1) -^t P.\Lambda(2) \\ K.Q(2) & = -^tP.k.v(2) + \Lambda(2) \\ q(2) & = v(2) + P.Q(2) \end{cases} \tag{3.30}$$

A l'ordre $i \geq 2$:

$$\begin{cases} \Lambda(i) & = C.[S.q(i-1) - Q(i-1)] \\ v(i) & = (q(i-1) - P.q(i-1)) - (k^*)^{-1}[k.q(i-1) +^t S.\Lambda(i)] \\ K.Q(i) & = -^tP.k.v(i) + \Lambda(i) \\ q(i) & = v(i) + P.Q(i) \end{cases} \tag{3.31}$$

On remarque que tous ces problèmes ont la même matrice, donc une seule triangulation est nécessaire, et puisque cette matrice correspond à celle de la matrice de rigidité de la grille grossière, sa décomposition ne devrait pas nécessiter un temps de calcul conséquent.

A l'ordre 0, on retrouve le même problème que celui de l'algorithme 1, on démarre ainsi les calculs sur la grille grossière avec une bonne initialisation de la solution, puisqu'on effectue une résolution exacte sur cette grille.

Intuitivement, on anticipe un gain en efficacité par rapport au premier algorithme, la réduction du coût de l'itération étant due au début du processus, au faible nombre de degrés de liberté de la grille grossière (nombre de division N plus grand), et à chaque ordre, au fait qu'on dispose d'un très bon préconditionneur sur la grille fine (choix de la matrice k^*).

En ce qui concerne la programmation de cet algorithme, il existe une petite différence par rapport à l'algorithme 1 : la construction de l'opérateur de restriction S est très simple à programmer avec le rangement suivant :

$$q = \begin{Bmatrix} q_g \\ q_l \end{Bmatrix} \tag{3.32}$$

On introduit les composantes de Λ sur les équations globales, et on met 0 sur les lignes locales. En plus il n'est plus utile de stocker à la fois $q(i)$ et $v(i)$. Il suffit de stocker $q(i)$ et de calculer $v(i)$ par : $v(i) = q(i) - P.Q(i)$, puisque le calcul du prolongement $P.Q(i)$ est très rapide (stockage morse de P). Les seuls calculs potentiellement coûteux dans cet algorithme sont donc (voir algorithme) :

- Triangulation de k^*
- Triangulation de K et la montée descente, si Ω_H est assez fin
- Les deux produits matrices-vecteurs : $k.q(i)$ et $k.v(i)$

3.3 Applications et résultats numériques

3.3.1 Introduction

L'objectif de ce paragraphe est de présenter quelques exemples pour valider cette nouvelle technique à deux grilles. Les mêmes tests que dans le chapitre précédent seront étudiés. Nous considérons le même exemple : la plaque encastrée en deux coins et soumise à un chargement f de l'autre côté, les caractéristiques géométriques et mécaniques sont les mêmes(voir chapitre 3).

Dans tous les tests effectués, la matrice de rigidité de la grille grossière est stockée sous forme d'un stockage frofil, celle de la grille fine ainsi que l'opérateur de prolongement seront stockés sous forme morse.

Ensuite une étude détaillée sur les temps de calcul des différentes étapes de l'algorithme sera présentée. L'efficacité de l'algorithme à deux grilles proposé est évaluée en comparant les temps de calcul avec celle de la méthode à deux grilles classique et la méthode du gradient conjugué préconditionné.

Enfin, nous discuterons d'autres variantes de cette méthode pour des maillages non-structurés, nous verrons que les tests menés sur des exemples mécaniques à maillage non-structuré conduisent à des conclusions identiques.

3.3.2 Applications de la nouvelle technique à deux grilles avec préconditionnement incomplet de Cholesky

Dans cette partie, on propose de montrer l'efficacité de l'algorithme à deux grilles avec préconditionnement incomplet de Cholesky de niveau 0 sur les tests présentés précédement dans le tableau (3.1). Pour cela, on donne pour les cinq tests l'ordre de convergence obtenu et le résidu Padé correspondant.

On rappelle que pour les 3 premiers tests, le nombre de division N vaut 2. Les tests 3 et 4 correspondent à $N = 3$.

	Grille grossière	Grille fine	Rapport
Test1	2178	8450	3.88
Test2	8450	33282	3.93
Test3	33282	132098	3.99
Test4	1568	13448	8.57
Test5	13448	119072	8.85

TAB. 3.1: *Nombre de d.d.l pour les cinqs exemples étudiés*

Le tableau (3.2) indique l'ordre de convergence de l'algorithme 2, nécessaire pour avoir un résidu inférieur à 10^{-10}. La solution retenue est celle utilisant les approximants de Padé, et le préconditionneur k^* choisi est la factorisation incomplète de Cholesky de niveau 0 : $IC(0)$.

Comme dans le chapitre précédent, on observe une indépendance de la vitesse de convergence de l'algorithme 2 effectuée sur les trois premiers maillages dont le nombre de division est $N = 2$ (ordre de convergence est égale à 12 pour les tests 1, 2 et 3). Pour les autres maillages (tests 4 et 5) où $N = 3$, l'orde de convergence est égal à 14. Ce résultat permet d'en déduire l'indépendance de maillage pour l'algorithme à 2-grilles proposé.

Le tableau (3.2) montre aussi une nette diminution de l'ordre de convergence de l'algorithme à 2-grilles grâce à l'utilisation de la factorisation incomplète de Cholesky niveau 0 : l'ordre 12 au lieu de 30 pour les tests 1,2 et 3, et l'ordre 14 pour les tests 4 et 5 au lieu de 60.

Tests	Test1	test2	Test3	Test4	Test5
Ordre de convergence	12	12	12	14	14
Résidu Padé	1.410^{-10}	0.910^{-10}	2.710^{-10}	4.210^{-10}	2.310^{-10}

TAB. 3.2: Ordre de convergence du nouveau algorithme et le résidu Padé correspondant.

Le résultat le plus marquant de ces tests est le très faible ordre de troncature de convergence (ordres 12 et 14), même avec la plus grande différence entre les deux

grilles ($N = 3$).

Le changement de préconditionneur augmente l'efficacité de l'algorithme et devrait permettre des écarts importants entre deux niveaux de grilles.

Sur la figure (3.1), on représente le logarithme décimal de la norme du vecteur résidu utilisant les approximants de Padé en fonction de l'ordre de troncature I pour les tests 3 et 5. Les autres tests donnent des résultats similaires.

Ces courbes montrent que la convergence de la nouvelle technique avec préconditionnement incomplet de Cholesky de niveau 0 est presque linéaire en fonction des ordres de troncature.

Notons que nous nous sommes limités seulement au niveau 0. En effet, il est certain que pour avoir un minimum d'itérations, il faudrait augmenter le niveau de remplissage. Mais dans ce cas, le nombre de termes stockés augmente sensiblement surtout si nous traitons des problèmes à grand nombre de degrés de liberté. En pratique, le meilleur compromis se trouve pour le niveau 0.

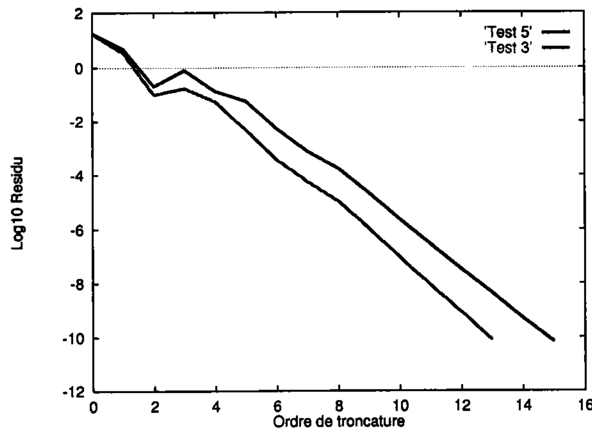


FIG. 3.1: Logarithme décimal du vecteur résidu (Padé) en fonction de l'ordre I pour les tests 3 et 5

3.3.3 Etude détaillée du temps CPU de l'algorithme

L'objectif de cette étude est de présenter en détail les temps CPU de chaque étape de l'algorithme. La différence par rapport au premier algorithme se situe au niveau du préconditionneur.

Avec cette nouvelle technique, on construit la matrice de préconditionnement k^*, cette matrice a une structure triangulaire inférieure (ou supérieure), puis une résolution incomplète par montée/descente est effectuée pour calculer la correction à la solution. Donc par rapport au premier algorithme, il y a en plus la construction de la matrice de préconditionnement incomplète de Cholesky et une résolution incomplète par montée/descente.

	test 1	test 2	test 3	test 4	test 5
Ordre 0					
Construction de K, grille grossière	0.29	1.15	5.27	0.17	2
Triangulation de K	0.36	8.22	155.23	0.19	23.72
Temps de résolution par K	0.03	0.29	2.31	0.02	0.59
Multiplication par P : $P.q_g(0)$	0.01	0.05	0.23	0.02	0.21
Calcul du résidu	0.03	0.09	0.37	0.15	0.32
Temps de la triangulation incomplète	0.09	0.37	1.5	0.15	1.35
Temps total à l'ordre 0	0.81	10.17	164.83	0.58	28.19
Ordre i					
Temps du produit matrice-vecteur : $k.q(i)$	0.02	0.09	0.38	0.04	0.32
Temps du produit matrice-vecteur : $k.v(i)$	0.03	0.09	0.37	0.03	0.31
Temps de multiplication par P : $P.k.v(i)$	0.01	0.05	0.21	0.02	0.19
Temps de résolution par K	0.03	0.3	2.32	0.02	0.59
Temps de multiplication par P : $P.Q(i)$	0.02	0.05	0.23	0.02	0.21
Temps de la montée-descente incomplète	0.03	0.12	0.5	0.05	0.46
Temps total à l'ordre i	0.14	0.7	4.01	0.18	2.08
Total	2.49	18.57	212.95	3.1	57.31

TAB. 3.3: **Détail du temps CPU en secondes pour les 5 exemples.**

Le tableau 3.3 donne le temps CPU en secondes pour chaque étape de l'algorithme pour les 5 tests. Tout d'abord on donne le temps CPU à l'ordre 0, ce temps comporte essentiellement la décomposition de la matrice de rigidité de la grille grossière.

Pour les tests 1 et 2, le temps CPU est presque négligeable par rapport au temps CPU total de l'algorithme : 14% pour le test 1 et seulement 1% pour le test 4. Par contre pour les tests 2, 3 et 5 où le nombre de degrés de liberté est assez grand, ce temps est très important comparé avec le temps CPU total de l'algorithme : pour le test 2, le temps CPU de la décomposition de la matrice de rigidité de la grille grossière représente 44% du temps CPU total de l'algorithme, 72% pour le test 3 et 41% pour

le test 5. Le temps CPU à l'ordre i, contient essentiellement les temps CPU des deux produits matrices-vecteurs et la résolution sur la grille gossière.

Comme nous pouvons le constater dans le tableau 3.3, le temps CPU des deux produits matrices-vecteurs utilisant la grille fine et l'opérateur de prolongement est négligeable en comparaison avec le temps CPU total de l'algorithme. Ceci est essentiellement dû au stockage morse de ces opérateurs.

A noter que le temps CPU pour la résolution et la triangulation incomplète de Cholesky qui est négligeable pour ces tests, peut devenir important si on traite des exemples à grand nombre de degrés de liberté.

Le seul temps qui reste très coûteux dans cet algorithme est la décomposition de la matrice de rigidité de la grille grossière calculé à l'ordre 0 de l'algorithme. Le seul moyen de diminuer ce temps est d'utiliser des grilles encore plus grossières. Ceci augmentera l'ordre de troncature nécessaire à la convergence sans pour autant augmenter de façon conséquente le temps CPU total de l'algorithme.

En effet le temps CPU correspondant au calcul d'un ordre I (avec $I > 0$) reste relativement faible comparé au temps CPU total.

3.3.4 Comparaison avec d'autres méthodes

Le tableau 3.4, montre la propriété caractéristique des méthodes multigrilles, c'est-à-dire l'indépendance du facteur de convergence par rapport au pas de discrétisation du maillage : pour les tests 1, 2 et 3, l'ordre de convergence de l'algorithme proposé (MBP) est de 12, le nombre de cycles de la méthode classique (MBC) est de 8. Pour les autres tests, une légère augmentation de l'ordre de convergence et du nombre de cycles est observée : à peu près 14 pour la méthode (MBP) et 10 pour la méthode (MBC).

A préciser que pour la méthode (MBC), nous avons utilisé comme lisseur la méthode du gradient conjugué préconditionné avec une décomposition incomplète de Cholesky de niveau 0. Trois itérations de lissage sont effectuées. Nous avons choisi

ce type de lisseur dans le but d'avoir une certaine homogénéité dans le choix des préconditionneurs pour les trois méthodes présentées et comparées dans ce chapitre.

On notera que la méthode classique MBC demande moins d'itérations que la méthode proposée (MBP), contrairement à ce qui a été observé au chapitre précédent. Du côté de la méthode proposée (MBP), l'amélioration est due à l'efficacité du solveur homotopie - série - Padé. Du côté de la méthode classique (MBC), la vitesse de convergence est accrue d'une part par les trois opérations de lissage, d'autre part par la résolution par la méthode du gradient conjugué, qui implique un line-search et une condition d'orthogonalité des directions de descente. Ces deux ingrédients (lissage, gradient conjugué) n'ont pas été introduits dans notre algorithme (MBP) : c'est ce qui explique que l'ordre de la méthode MBP soit supérieur au nombre de cycles dans la méthode classique.

Dans le cas de la méthode du gradient conjugué préconditionné par la factorisation incomplète de Cholesky, on remarque que le nombre d'itérations augmente avec la taille du problème. Cette dernière remarque montre encore une fois que les méthodes multigrilles présentent un avantage par rapport à la méthode du gradient conjugué.

	MBP	MBC	GCPIC
	Ordre I	Nombre de cycles	IT
test 1	12	8	136
test 2	12	8	274
test 3	12	8	554
test 4	14	10	169
test 5	14	10	512

TAB. 3.4: Ordre de convergence et nombre d'itérations des différents méthodes utilisées. MBP : algorithme proposé à deux grilles, résolution asymptotique, préconditionneur $IC(0)$. MBC : algorithme à deux grilles classique, résolution itérative, lissage par le gradient conjugué couplé avec $IC(0)$ avec 3 itérations de lissage, GCPIC : gradient conjugué prconditionné couplé avec $IC(0)$, résolution itérative.

Le tableau 3.5 présente le temps de calcul nécessaire pour obtenir la solution sur la grille fine soit par l'algorithme à deux grilles proposé (MBP), soit par la méthode à 2-grilles classique (MBC), soit par la méthode du gradient conjugué préconditionné

(GCPIC).

Nous observons sur ces résultats numériques que les méthodes à 2-grilles (algorithme proposé et la méthode classique) sont très compétitives en terme de temps CPU. En effet le temps CPU des opérateurs de prolongement et de restriction ainsi que le temps de résolution et décomposition des matrices est le même dans chacune des deux méthodes, la seule différence est due au temps de calcul des opérations matrices-vecteurs des deux méthodes.

Remarquons qu'avec trois itérations de lissage (plus un calcul du résidu) et pour $N = 3$, on obtient 40 produits matrices-vecteurs pour l'algorithme classique MBC. 42 produits matrices-vecteurs sont obtenus dans le cas de l'algorithme proposé (MBP) (2 produits matrices-vecteurs et un calcul du résidu). A ce niveau, les deux algorithmes MBP et MBC sont comparables, ce qu'on verifie en termes de temps CPU total, voir le tableau (3.5).

La méthode du gradient conjugué préconditionné est inefficace en terme de temps CPU par rapport aux autres méthodes de résolution. En effet le temps CPU de la méthode du gradient conjugué est environ 2 à 3 fois supérieur à celui des temps CPU des autres méthodes, pour $N = 2$ et 8 fois pour $N = 3$.

	MBP CPU	MBC CPU	GCPIC CPU
test 1	2.49	2.5	7.1
test 2	18.57	18.62	64.3
test 3	213	208.27	541.1
test 4	3	4.03	16
test 5	57.31	65	462.4

TAB. 3.5: Comparaison du temps CPU en secondes des différentes méthodes utilisées.

3.3.5 Tests sur l'influence de la matrice C sur la vitesse de convergence de l'algorithme

L'objectif de ce test est de voir l'influence de la matrice C introduite dans l'algorithme à deux grilles sur sa vitesse de convergence.

Nous avons vu précédement que $k+2C \neq 0$, ce qui implique un choix évident pour la matrice C. Dans ce test on a choisi $C = \alpha K$ où K représente la partie diagonale de la matrice de rigidité de la grille grossière (on peut prendre aussi pour K la totalité de la matrice de rigidité de la grille grossière ou tout simplement une constante pour la matrice C).

Le problème étudié est le test 1 présenté dans le tableau (2.3) : le nombre de degrés de liberté de la grille grossière est de $2178 d.d.l$, celle de la grille fine est 8450 d.d.l.

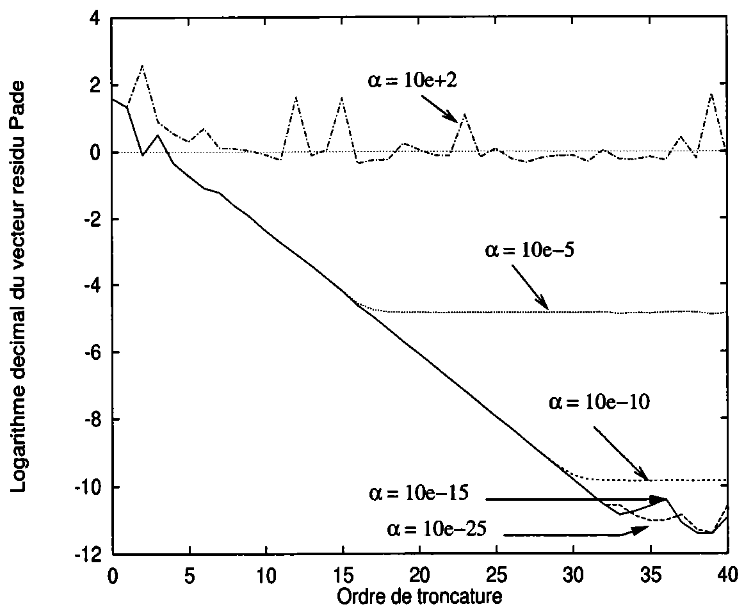


FIG. 3.2: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 **préconditionneur diagonal, N=2** , grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l

La figure (3.2) représente l'évolution du vecteur résidu (Padé) en fonction de l'ordre de troncature de l'algorithme pour différentes valeurs du paramètre α. Nous remarquons que pour des petites valeurs de α, la convergence de l'algorithme est rapide. Quand la valeur de α augmente, la solution se dégrade rapidement et l'algorithme ne converge pas. Toutes les valeurs de α comprises entre 10^{-10} et 10^{-20}, permettent une bonne convergence de l'algorithme. A noter cependant que pour des plus grandes valeurs de α, la figure (3.3) montre que l'algorithme converge également, lorsque l'on

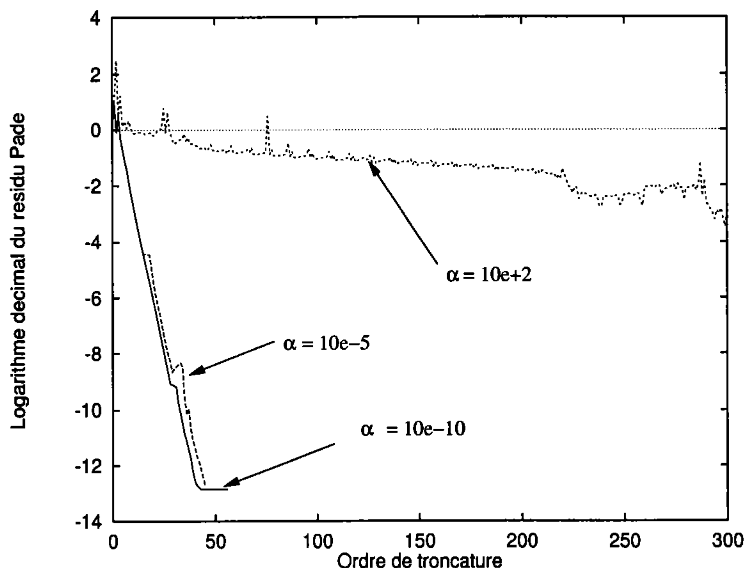


FIG. 3.3: Algorithme à deux grilles **itératif**, logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 **préconditionneur diagonal, N=2**, grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l

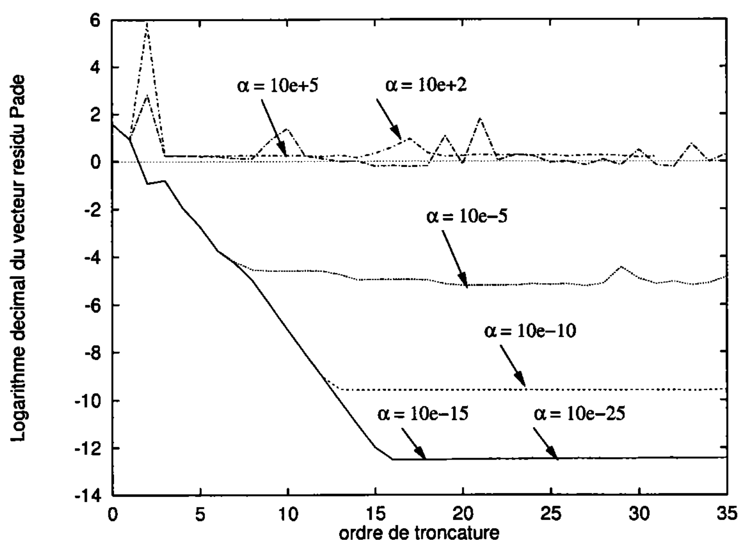


FIG. 3.4: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=2**, grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l)

utilise un processus itératif comme celui défini au chapitre précédent. Pour des valeurs de α très grandes ($\alpha = 10^2$), l'algorithme proposé ne converge plus, même en utilisant un algorithme itératif.

Les figures (3.4), (3.5) et (3.6) représentent l'évolution du vecteur résidu Padé en

UNIVERSITÉ DE METZ S.C.D.

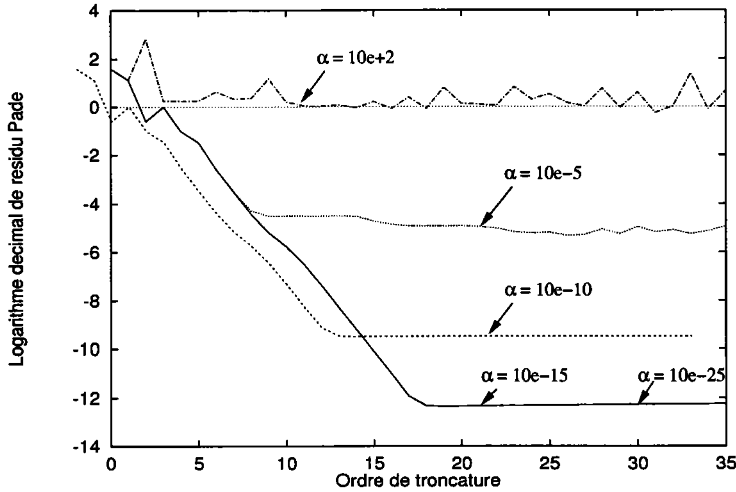


FIG. 3.5: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour diffŕentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=3**, grille grossière : 2178 d.d.l, grille fine : 18818 d.d.l)

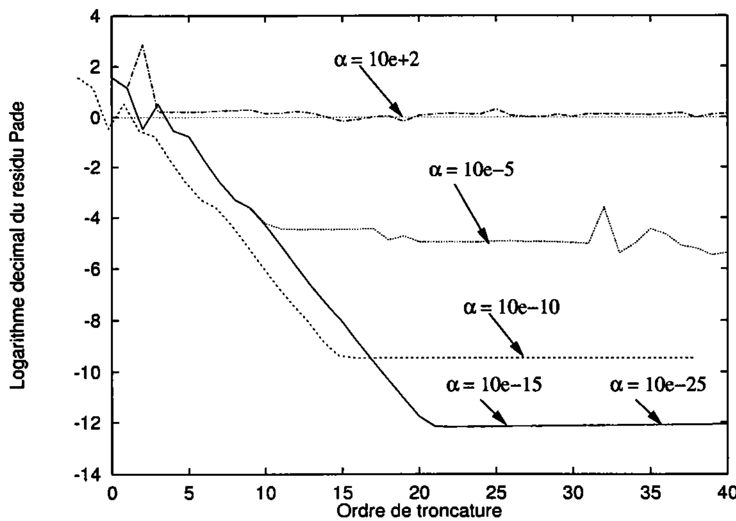


FIG. 3.6: Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=4**, grille grossière : 2178 d.d.l, grille fine : 33282 d.d.l)

fonction de l'ordre de troncature pour $N = 2$, $N = 3$ et $N = 4$, et pour différentes valeurs de α. Le préconditionnement utilisé est la factorisation incomplète de Cholesky de niveau 0.

On remarque que lorsque N augmente, l'ordre de convergence de l'algorithme augmente naturellement mais le même comportement de la convergence est observé.

Concernant la valeur du paramètre α, il semble d'après les tests présentés ici,

qu'une valeur de α comprise entre 10^{-10} et 10^{-20} conduit à une bonne convergence de l'algorithme et ceci quelle que soit la valeur de N et le type de préconditionneur choisi. Dans la suite, on prendra $\alpha = 10^{-15}$.

3.3.6 Tests sur l'influence de la grille grossière

Le but des tests suivants est de voir l'influence du choix de la grille grossière sur la vitesse de convergence de l'algorithme. Le préconditionnement étudié est la factorisation incomplète de Cholesky.

Pour ces tests, on a fixé le nombre des noeuds de la grille fine à $126 * 126$ noeuds, soit $65522 d.d.l$, de telle sorte qu 'on puisse utiliser plusieurs grilles grossières. Pour ces tests on a pris l'exemple du chapitre précédent : la plaque encastrée des deux côtés et soumise à un chargement ponctuel f de l'autre, avec les mêmes caractéristiques mécaniques.

Les tableaux 3.6 et 3.7 donnent le détail du temps CPU des différentes étapes de l'algorithme, l'évolution de l'ordre de convergence en fonction de N et le temps CPU total pour chaque valeur de N. Les grilles grossières dépendent du nombre de division N, on obtient des grilles de plus en plus grossières quand N augmente.

On observe sur ces tableaux que les temps CPU donnés par les étapes utilisant la grille fine ainsi que les opérations impliquant l'opérateur de prolongement P, ne changent pratiquement pas puisque le nombre de degrés de liberté de la grille fine est fixée à $65522 d.d.l.$

Les seules temps succeptibles de changer quand la valeur de N varie sont les temps de résolution impliquant la grille grossière, c'est-à-dire la décomposition, la construction et le temps de résolution sur cette grille.

Les tableaux (3.6) et (3.7) montrent que l'augmentation du paramètre N (c'est-à-dire la diminution du nombre de d.d.l de la grille grossière) conduit à une augmentation de l'ordre de convergence de l'algorithme (12 pour $N = 2$, 22 pour $N = 6$ et 156 pour $N = 18$).

Toute fois cette augmentation de l'ordre de convergence ne se traduit pas nécessairement par une augmentation du temps CPU de l'algorithme. En effet l'algorithme demande un temps equivalent pour $N = 2$ et pour $N = 12$, soit environ $60(s)$ (tableaux 3.6 et 3.7), alors que l'ordre de convergence pour ces deux valeurs de N est respectivement de 12 et 71. Cette égalité du temps CPU, s'explique par le temps nécessaire au traitement de la matrice de la grille grossière, qui est relativement élevé pour $N = 2$ (environ 60% de l'algorithme) et très faible pour $N = 12$.

	N=2	N=3	N=4	N=5	N=6
Ordre de convergence	12	14	17	20	22
Ordre 0					
Construction de K, grille grossière	2.37	1.01	0.59	0.39	0.23
Décomposition de K	35.71	6.62	1.22	0.56	0.3
Temps de résolution	0.82	0.24	0.11	0.05	0.02
Temps de multiplication par P : $P.q_g(0)$	0.1	0.1	0.08	0.11	0.1
Calcul du résidu	0.18	0.18	0.19	0.19	0.19
Temps de la triangulation incomplète	0.82	0.82	0.82	0.83	0.83
Temps total à l'ordre 0	40	8.97	3.01	2.13	1.67
Ordre i					
Temps du produit matrice-vecteur :$k.q(i)$	0.18	0.18	0.18	0.18	0.19
Temps du produit matrice-vecteur :$k.v(i)$	0.18	0.18	0.18	0.18	0.19
Temps de multiplication par P :$P.k.v(i)$	0.11	0.1	0.11	0.11	0.1
Temps de résolution	0.82	0.24	0.1	0.05	0.03
Temps de multiplication par P : $P.Q(i)$	0.11	0.1	0.11	0.11	0.1
Temps de la montée/Descente incomplète	0.24	0.25	0.25	0.25	0.22
Temps total à l'ordre i	0.64	1.05	0.93	0.88	0.83
Total	59.7	23.67	18.82	19.73	19.93

TAB. 3.6: **Détail du temps CPU en secondes pour différentes valeurs du nombre de division N**

Cette observation demeure valable pour des valeurs de N comprises entre 2 et 6. Pour $N = 9$ et 10, on observe une augmentation significatrice du temps CPU du traitement de la matrice de rigidité de la grille grossière, ce qui entraîne une augmentation du temps CPU total de l'algorithme. Pour comprendre et expliquer ce phénomène, nous avons utilisé un algorithme de renumérotation des noeuds de la grille grossière (Cuthill-Mac Kee pour plus de détails voir [42]) dans l'objectif de diminuer

	N=9	N=10	N=12	N=15	N=18
Ordre de convergence	42	50	71	111	156
Ordre 0					
Construction de K, grille grossière	0.12	0.1	0.07	0.04	0.03
Décomposition de K	0.45	0.33	0.03	0.01	0.06
Temps de résolution	0.38	0.29	0.	0.	0.06
Temps de multiplication par P :$P.q_g(0)$	0.11	0.09	0.09	0.01	0.1
Calcul du résidu	0.18	0.19	0.19	0.18	0.19
Temps de la triangulation incomplète	0.83	0.83	0.83	0.85	0.84
Temps total à l'ordre 0	2.07	1.83	1.21	1.09	1.28
Ordre i					
Temps du produit matrice-vecteur :$k.q(i)$	0.19	0.19	0.19	0.19	0.18
Temps du produit matrice-vecteur : $k.v(i)$	0.18	0.18	0.18	0.18	0.18
Temps de multiplication par P : $P.k.v(i)$	0.1	0.11	0.1	0.1	0.1
Temps de résolution	0.39	0.3	0.01	0.	0.06
Temps de multiplication par P : $P.Q(i)$	0.1	0.1	0.1	0.11	0.1
Temps de la montée/descente incomplète	0.23	0.23	0.25	0.25	0.25
Temps total à l'ordre i	1.19	1.11	0.83	0.83	0.87
Total	52.05	57.33	60	93.22	137

TAB. 3.7: Détail du temps CPU (s) pour différentes valeurs de N

le temps de traitement de la grille grossière. Les résultats de cette renumérotation sont donnés dans le tableau (3.8) où l'on voit clairement que grâce à une renumérotation efficace de la grille grossière, les temps de traitement des matrices de rigidité grossière passent respectivement de $0.38(s)$ à $0.01(s)$ pour $N = 9$ et de $0.29(s)$ à $0.01(s)$ pour $N = 10$. Cette diminution du temps CPU du traitement de la matrice grossière se

	Numérotation 1		**Numérotation 2**	
Nombre de division N	9	10	9	10
Ordre de convergence	42	50	42	50
Temps de résolution	0.38	0.29	0.01	0.01
Total	52.05	57.33	35.7	42.5

TAB. 3.8: Temps CPU en secondes de la résolution pour les deux types de numérotations : numérotation 2 est obtenue par l'algorithme de Cuthill-Mac Kee

traduit par une diminution du temps CPU total de l'algorithme ($35.7(s)$ pour $N = 9$, $42.5(s)$ pour $N = 10$).

Remarquons aussi que le temps CPU total dépend principalement de la valeur de

N. Quand N augmente, l'ordre de convergence augmente et le meilleur compromis en ce qui concerne le temps CPU total est obtenu pour $N = 4$. Pour $N = 5$ et$N = 6$, l'algorithme donne toujours de très bons résultats. Finalement, notons que quelle que soit la valeur de N (pour $2 \leq N \leq 18$), l'algorithme converge toujours.

3.4 Maillage non-structuré : autres variantes

3.4.1 Influence de la distorsion du domaine

D'autres schémas algorithmiques peuvent se déduire de l'algorithme à deux grilles précédent dans le cas où le maillage n'est pas structuré (curviligne). Nous avons vu dans le chapitre précédent, que lorsque le maillage n'est pas structuré, le nombre de points de Gauss n'est pas le même dans le calcul de ${}^{t}P.k.P$ et K. Ces deux matrices sont légèrement différentes et pour appliquer d'une manière exacte l'algorithme proposé, il faut introduire une autre équation qui tient compte de la différence entre ${}^{t}P.k.P$ et K, on corrige ainsi la mauvaise interpolation.

Les tests numériques suivants concernent l'influence de la distorsion du domaine sur la méthode à deux-grilles proposée. Dans un premier temps, on applique l'algorithme 2 avec multiplicateur de Lagrange dans le cas d'un maillage non-structuré sans correction de l'interpolation. Dans un deuxième temps, on donnera d'autres types de variantes avec correction de l'interpolation.

L'exemple étudié est une plaque encastrée suivant l'axe des abscisses et l'axe des ordonnés, est soumise à une pression sur sa face supérieure. Les caractéristiques mécaniques et géométriques de la plaque sont représentées dans la figure (3.7).

La grille grossière est composée de 11×11 noeuds soit $222d.d.l$, elle est maillée par des élèments quadrangles à quatre noeuds, la grille fine pour $N = 2$, est composée de 21×21 noeuds soit $882d.d.l.$

Le maillage fin est obtenu par transformation du maillage grossier sur un carré. Les coordonnées des noeuds intérieurs du maillage fin sont obtenues par interpolation des noeuds grossiers.

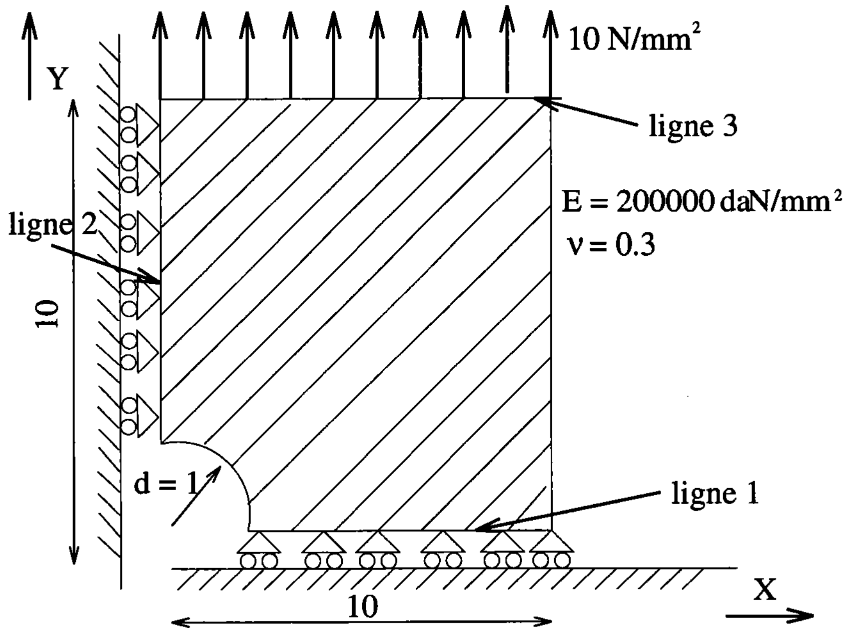


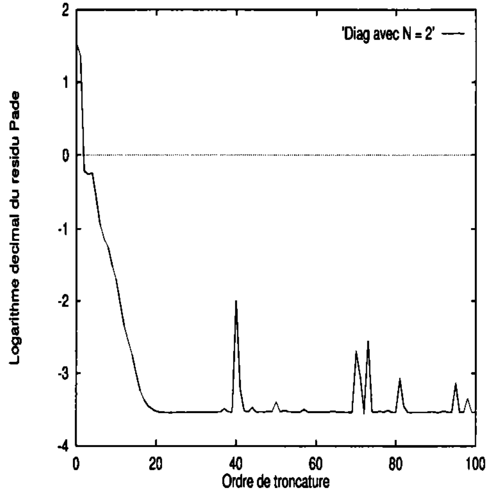
FIG. 3.7: Exemple de plaque soumise à un chargement de pression

Les résultats que nous présentons ci-après ont pour but de tester numériquement l'algorithme à deux grilles proposé précédemment dans le cas d'un maillage non-structuré sans correction de l'opérateur de prolongement.

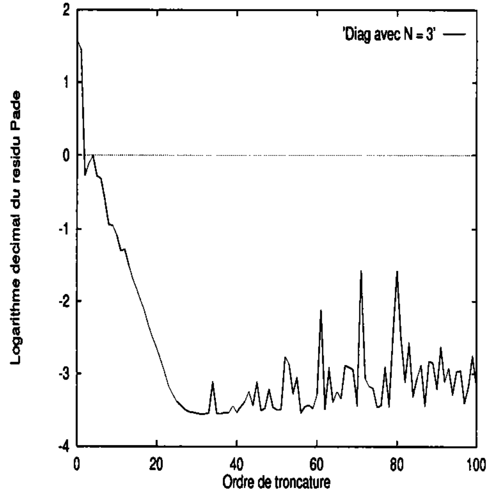
Les figures (3.8) et (3.9) montrent l'évolution du vecteur résidu Padé en fontion des ordres de troncature pour deux types de préconditionneurs : le préconditionneur diagonal et la factorisation incomplète de Cholesky de niveau 0.

Nous observons sur ces résultats numériques que sans correction de l'interpolation, l'algorithme ne peut converger quelle que soit le préconditionneur, puisque le résidu stagne toujours autour de la valeur 10^{-5}. Pour des petites valeurs de l'ordre de troncature, le résidu diminue rapidement. Des oscillations sont ensuite observées. Ce qui s'explique peut être par le fait que les informations apportées par l'opérateur de prolongement sont insuffisantes et ne peuvent permettre à l'algorithme de converger. Une version itérative est ensuite proposée pour améliorer la convergence.

Les figures (3.10) et (3.11) donnent l'évolution de la norme du résidu Padé en fonction des ordre de troncature et montrent l'influence de l'utilisation d'une procédure itérative sur la convergence de l'algorithme.

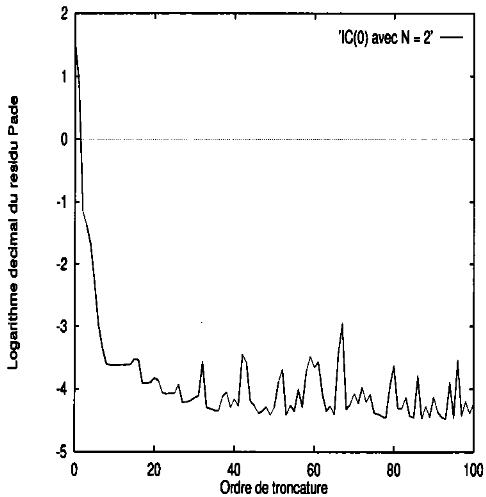


(a) Préconditionneur diagonal pour N = 2

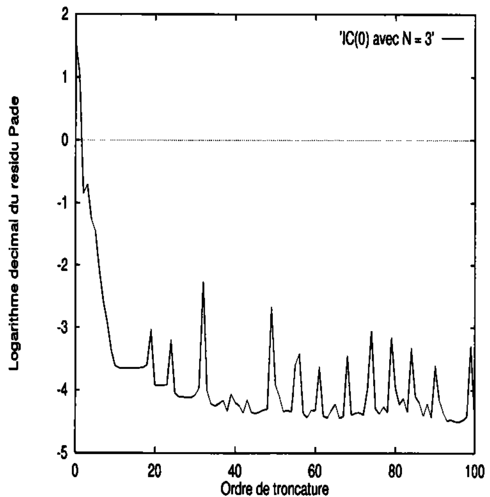


(b) Préconditionneur diagonal pour N = 3

FIG. 3.8: Logarithme du résidu Padé en fonction des ordres de troncature, paramètre de précision : $\varepsilon = 10^{-10}$



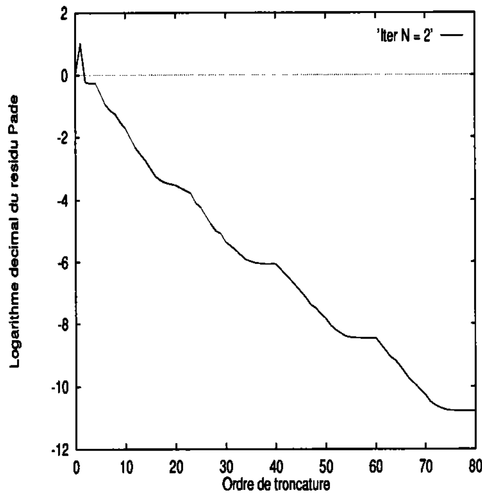
(a) Préconditionneur Incomplet de Cholesky pour N = 2



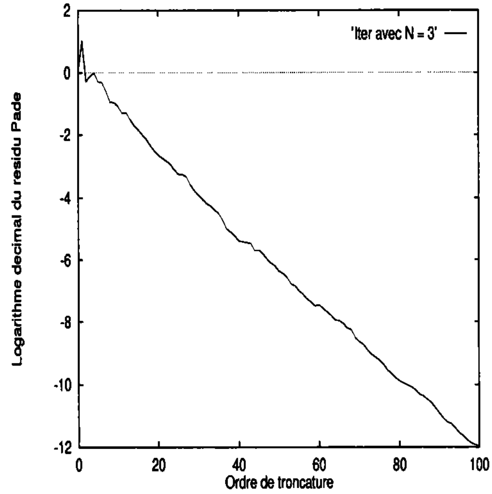
(b) Préconditionneur Incomplet de Cholesky pour N = 3

FIG. 3.9: Logarithme du résidu Padé en fonction des ordres de troncature, paramètre de précision : $\varepsilon = 10^{-10}$

Nous observons sur ces résultats que même sans correction de l'interpolation, l'algorithme itératif à 2-grilles converge. La convergence est obtenue dans le cas du préconditionneur diagonal et dans le cas de la factorisation incomplète de Cholesky.

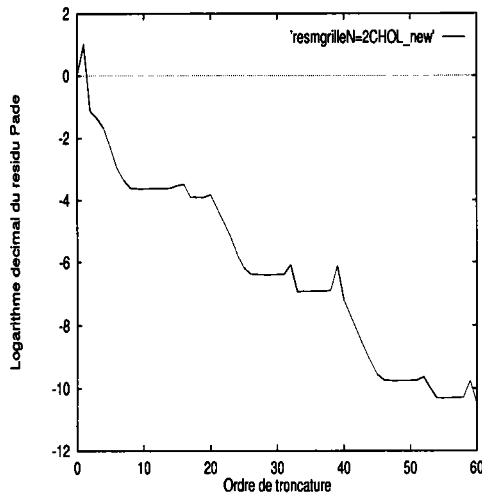


(a) Résultat de l'algorithme itératif : préconditionneur diagonal pour $N = 2$, convergence obtenue à l'ordre 3, itération 4, 20 ordres fixés, soit 63 vecteurs

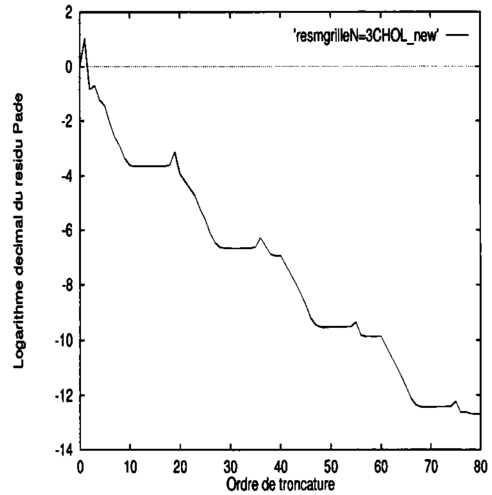


(b) Résultat de l'algorithme itératif : préconditionneur diagonal pour $N = 3$, convergence obtenue à l'ordre 13 de la quatrième itération, 20 ordres fixés, soit 73 vecteurs

FIG. 3.10: Evolution du résidu Padé en fonction des ordres de troncature



(a) Résultat de l'algorithme itératif : préconditionneur Incomplet de Cholesky pour $N = 2$, convergence obtenue à l'ordre 4 de la troisième itération, 20 ordres fixés, soit 44 vecteurs



(b) Résultat de l'algorithme itératif : préconditionneur Incomplet de Cholesky pour $N = 3$, convergence obtenue à l'ordre 6, itération 3, 20 ordres fixés, soit 46 vecteurs

FIG. 3.11: Evolution du résidu Padé en fonction des ordres de troncature

Pour ce dernier préconditionneur les performances sont meilleures par rapport au préconditionnement diagonal. Par exemple pour $N = 2$ et avec un ordre de troncature égale à 20, l'algorithme à deux grilles converge à l'ordre 4 de la troisième itération au lieu de la quatrième itération pour le préconditionnement diagonal. D'autres tests ont été effectués pour des ordres de troncature fixés à 30 et 40. On obtient les mêmes résultats avec une légére différence pour les ordres de convergence dans le cas où $I = 40$ et même dans le cas $I = 30$. Ceci montre tout l'intérêt d'utiliser une procédure itérative pour améliorer la convergence des algorithmes proposés (si ne on corrige pas l'interpolation).

Mais même si cette version itérative permet d'obtenir la convergence de l'algorithme, ce schéma n'est pas à priviligier car il n'est pas certain que celui-ci converge toujours. Dans le but d'éviter ce type de problème, nous proposons d'autres algorithmes qui prennent en compte les difficultés dues au maillage non-structuré.

3.4.2 Solutions proposées

La difficulté de résoudre des problèmes utilisant des maillages non-structurés provient de la résolution du problème posé sur la grille grossière. Par conséquent pour un maillage non-structuré, on ne trouve pas exactement la matrice de rigidité de la grille grossière notée $K^{\oplus}$ qui est déjà construite, mais on retrouve la matrice réduite : $K =^t P.k.P$. Si on veut travailler avec la matrice $K^{\oplus}$, il faut tenir compte de la différence entre ces deux matrices. Pour cela on réécrit tout d'abord l'équation à résoudre sur la grille grossière, soit :

$$^tP.(k.q - f) - \Lambda = 0 \tag{3.33}$$

Ensuite, on modifie cette équation par une transformation d'homotopie. Il y a plusieurs choix possibles pour l'utilisation de l'algorithme à deux grilles dans le cas d'un maillage non-structuré. Ce choix concerne principalement la variable à utiliser. Dans la suite de nos propos, on centrera notre étude sur deux variantes. La première sera notée "variante 1" , elle est très simple et consiste à utiliser la variable q au lieu de

$P.Q$. La transformation d'homotopie correspondante s'écrit de la manière suivante :

$$(1-\varepsilon)K^{\oplus}.Q + \varepsilon^t P.k.q - \Lambda = F \tag{3.34}$$

La deuxième sera notée "variante 2", elle est plus générale que la première et utilise la matrice réduite $K =^t P.k.P$ et la correction v calculée par une factorisation incomplète de Cholesky :

$$(1-\varepsilon)K^{\oplus}.Q + \varepsilon({}^tP.k.P).Q +^t P.q.v - \Lambda = F \tag{3.35}$$

On peut remarquer que si on prend $\varepsilon = 1$ dans les relations (3.34) et (3.35), on retrouve le problème posé sur la grille grossière (3.33)

Pour parler d'efficacité numérique, les deux variantes que l'on vient de décrire vont être appliquées dans la cas d'un maillage non-structuré et ensuite comparées avec le deuxième algorithme avec multiplicateur de Lagrange.

En remplaçant l'équation ${}^tP.(k.q - f) - \Lambda = 0$ du problème (3.23) de l'algorithme à 2-grilles par les équations modifiées (3.34) et (3.35), on obtient les deux systèmes suivants :

- **Pour la premiére variante on a :**

$$\begin{cases} \varepsilon C.(S.q - Q) = \Lambda \\ (1-\varepsilon)k^*.v + \varepsilon[k.q - f] +^t S.\Lambda = 0 \\ (1-\varepsilon)K^{\oplus}.Q + \varepsilon^t P.k.q - \Lambda = F \\ q = v + PQ \end{cases} \tag{3.36}$$

- **Pour la deuxième variante on a :**

$$\begin{cases} \varepsilon C.(S.q - Q) = \Lambda \\ (1-\varepsilon)k^*.v + \varepsilon[k.q - f] +^t S.\Lambda = 0 \\ (1-\varepsilon)K^{\oplus}.Q + \varepsilon({}^tP.k.P).Q +^t P.q.v - \Lambda = F \\ q = v + PQ \end{cases} \tag{3.37}$$

En injectant les expressions (3.24), (3.25), (3.26) et (3.27) dans (3.36) et (3.37), et après identification terme à terme suivant les puissances de ε, on obtient les deux nouvelles variantes à deux grilles pour le maillage non-structuré :

- **Premiére variante**

A l'ordre 0 :

$$\begin{cases} \Lambda(0) & = 0 \\ K^{\oplus}.Q(0) & = F \\ q(0) & = P.Q(0) \end{cases} \tag{3.38}$$

A l'ordre 1 :

$$\begin{cases} \Lambda(1) & = C.[S.q(0) - Q(0)] = 0 \\ k^{*}.v(1) & = -[k.q(0) - f] \\ K^{\oplus}.Q(1) & = K^{\oplus}.Q(0) -^{t} P.k.q(0) \\ q(1) & = v(1) + P.Q(1) \end{cases} \tag{3.39}$$

A l'ordre 2 :

$$\begin{cases} \Lambda(2) & = C.[S.q(1) - Q(1)] \\ k^{*}.v(2) & = k^{*}.v(1) - k.q(1) -^{t} P.\Lambda(2) \\ K^{\oplus}.Q(2) & = K^{\oplus}.Q(1) -^{t} P.k.q(1) + \Lambda(2) \\ q(2) & = v(2) + P.Q(2) \end{cases} \tag{3.40}$$

A l'ordre $i \geq 2$:

$$\begin{cases} \Lambda(i) & = C.[S.q(i-1) - Q(i-1)] \\ v(i) & = (q(i-1) - P.q(i-1)) - (k^{*})^{-1}.[k.q(i-1) +^{t} S.\Lambda(i)] \\ K^{\oplus}.Q(i) & = K^{\oplus}.Q(i-1) -^{t} P.k.q(i-1) + \Lambda(i) \\ q(i) & = v(i) + P.Q(i) \end{cases} \tag{3.41}$$

- **Deuxième variante**

A l'ordre 0 :

$$\begin{cases} \Lambda(0) & = 0 \\ K^{\oplus}.Q(0) & = F \\ q(0) & = P.Q(0) \end{cases} \tag{3.42}$$

A l'ordre 1 :

$$\begin{cases} \Lambda(1) & = C.[S.q(0) - Q(0)] = 0 \\ k^{*}.v(1) & = -[k.q(0) - f] \\ K^{\oplus}.Q(1) & = K^{\oplus}.Q(0) - ({}^{t}P.k.P).Q(0) -^{t} P.k.v(1) \\ q(1) & = v(1) + P.Q(1) \end{cases} \tag{3.43}$$

A l'ordre 2 :

$$\begin{cases} \Lambda(2) & = C.[S.q(1) - Q(1)] \\ k^{*}.v(2) & = k^{*}.v(1) - k.q(1) -^{t} P.\Lambda(2) \\ K^{\oplus}.Q(2) & = K^{\oplus}.Q(1) - ({}^{t}P.k.P).Q(1) -^{t} P.k.v(2) + \Lambda(2) \\ q(2) & = v(2) + P.Q(2) \end{cases} \tag{3.44}$$

A l'ordre $i \geq 2$:

$$\begin{cases} \Lambda(i) & = C.[S.q(i-1) - Q(i-1)] \\ v(i) & = (q(i-1) - P.q(i-1)) - (k^{*})^{-1}.[k.q(i-1) +^{t} S.\Lambda(i)] \\ K^{\oplus}.Q(i) & = K^{\oplus}.Q(i-1) - ({}^{t}P.k.P).Q(i-1) -^{t} P.k.v(i) + \Lambda(i) \\ q(i) & = v(i) + P.Q(i) \end{cases} \tag{3.45}$$

En remplaçant dans chaque problème de cet algorithme l'inconnue $q(i)$ par sa valeur calculée aux ordres précédents, on fait apparaitre l'expression suivante :

$$K^{\oplus} - ({}^{t}P.k.P) \tag{3.46}$$

Cette différence entre la matrice $K^{\oplus}$ et la matrice réduite ${}^{t}P.k.P$ est très importante si on veut utiliser des maillages non-structurés. On vérifiera par la suite que si le maillage est structuré, l'expression (3.46) est nulle et que l'on retrouve exactement les résultats du chapitre précédent.

3.4.3 Applications numériques et discussions

L'objectif de ces tests est d'étudier le comportement des deux nouvelles variantes lorsque le maillage n'est pas structuré. Pour cela, nous conservons l'exemple de la plaque étudiée précédemment. Les préconditionnements étudiés sont la factorisation incomplète de Cholesky de niveau 0 : $IC(0)$ et le préconditionnement par la diagonale pour quatre valeurs du nombre de division N. Ces résultats seront comparés avec ceux obtenus par la méthode à deux grilles classique pour deux types de lisseurs : la méthode de Jacobi et le gradient conjugué préconditionné par la factorisation incomplète de Cholesky (IC(0)). Pour ces derniers, trois itérations de lissage ont été utilisées et le paramètre de relaxation ω est égal à 0.8.

N	Variante 1	Variante 2	2-grilles classique
2	54 (54)	49 (51)	Div
3	83 (82)	73 (79)	Div
4	108 (113)	103 (129)	Div
5	143 (141)	131 (162)	Div

TAB. 3.9: *Préconditionneur diagonal pour les deux variantes (version itérative avec un ordre de troncature fixé à 20, les résultats entre parenthèse sont obtenus pour un ordre fixé à 30. Nombres de vecteurs nécessaires à la convergence et nombre de division N. Pour la méthode à deux grilles classique, nombres de cycles nécessaires à la convergence, méthode de Jacobi avec trois itérations de lissage*

On donne dans les tableaux (3.9) et (3.10), le nombre de vecteurs nécessaire pour la convergence des deux variantes et le nombre de cycles de la méthode à deux grilles

N	Variante 1	Variante 2	2-grilles classique
2	25 (23)	17 (17)	7
3	31 (29)	23 (21)	10
4	37 (33)	26 (24)	12
5	43 (40)	30 (28)	15

TAB. 3.10: *Préconditionneur incomplet de Cholesky IC(0) pour les deux variantes, résultats obtenus pour un ordre de troncature fixé à 30 (valeurs entre parenthèse), les autres sont obtenus pour un ordre de troncature fixé à 20. Nombres de vecteurs nécessaires à la convergence et nombre de division N, nombres de cycles nécessaires à la convergence de la méthode classique, la méthode du lissage est le gradient conjugué préconditionné par la factorisation incomplète de Cholesky avec trois itérations de lissage.*

classique, selon que l'on utilise soit un préconditionneur par la diagonale, soit la factorisation incomplète de Cholesky.

On remarque que pour toutes ces méthodes le nombre de vecteurs et le nombre de cycle nécessaires à la convergence dépendent du nombre de division N : plus on augmente la valeur de N, plus le nombre de vecteurs augmente.

Dans le cas d'un préconditionnement par la diagonale, les deux variantes convergent presque de la même manière et ceci quelle que soit la valeur de N ($2 \leq N \leq 5$). Des tests numériques ont montré que ces deux variantes convergent même pour des valeurs de N supérieurs à 5.

Par rapport aux résultats trouvés dans le cas d'un maillage structuré (30 vecteurs pour $N = 2$ et 42 vecteurs pour $N = 3$), le tableau (3.9) montre que le nombre de vecteurs a augmenté de 100% dans le cas $N = 2$ et environ 52% dans le cas $N = 3$.

Il faut noter qu'avec le préconditionneur diagonal, les deux variantes proposées convergent à condition d'utiliser une version itérative. La méthode à deux grilles classique diverge dans tous les cas. Par contre l'utilisation de la factorisation incomplète de Cholesky conduit à une diminution importante du nombre d'itérations. Comme on peut le constater dans le tableau (3.10), le nombre d'itérations obtenue par la variante 2 peut augmenter jusqu'à 42% pour $N = 2$ et 50% pour $N = 3$ par rapport au résultats obtenus dans le cas d'un maillage structuré (12 pour $N = 2$ et 14 pour

$N = 3$).

Par conséquent, on peut dire que pour un problème de maillage non-structuré, il vaut mieux utiliser un préconditionneur de Cholesky incomplet ou encore utiliser un préconditionneur diagonal mais avec plusieurs niveaux de grilles (ce qui n'a pas été fait ici).

Finalement, on constate d'après le tableau (3.10) que les résultats, en nombre de vecteurs, obtenus par la méthode à deux grilles classique sont presque identiques par rapport à ceux obtenus par la deuxième variante.

D'autre part, les resultats présentés dans le tableau (3.11) montrent clairement que l'application de la variante 2 à des maillages structurés (tests 1, 2, 3, 4 et 5) conduit aux mêmes résultats que ceux obtenus avec l'algorithme avec multiplicateur de Lagrange. Ce tableau montre également que la variante 1 n'est pas intéressante car elle conduit à une augmentation de l'ordre de convergence.

	Test 1	Test 2	Test 3	Test 4	Test 5
Variante 1	21	21	21	23	23
Variante 2	12	12	12	14	14

TAB. 3.11: *Ordre de convergence des deux variantes à des maillages structurés, préconditionneur incomplet de Cholesky de niveau 0*

3.5 Application du solveur multigrille pour une suite de systèmes linéaires à matrice invariante

Dans cette section, on cherche à tester l'utilisation du solveur multigrille sur une Méthode Asymptotique-Numérique (MAN).

Dans une première partie, on présente le temps de calcul de cette version de la MAN avec le nouveau solveur à deux grilles. L'évaluation des performances de ce couplage est montrée en comparant les résultats obtenus avec ceux obtenus par la MAN couplée avec la méthode directe (Crout).

Dans une deuxième partie, on discute l'utilisation du solveur multigrille sur la qualité de la solution obtenue par une MAN.

3.5.1 Applications numériques et temps de calcul

Les résultats donnés dans ce paragraphe concernent une poutre homogène encastrée en flexion. Cette poutre est maillée régulièrement avec des éléments $Q4$, quadrangles à 4 noeuds et 2 d.d.l par noeud. Les caractéristiques géométriques et mécaniques de cette poutre sont données dans la figure (3.12).

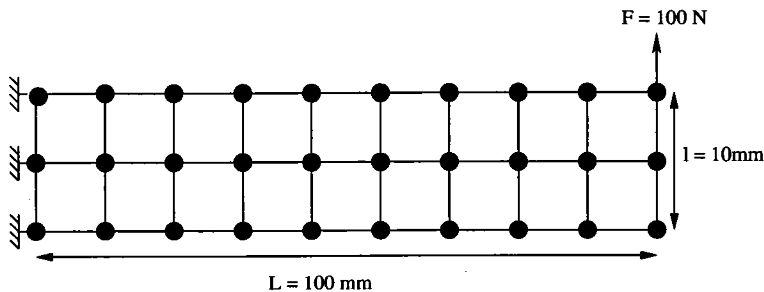


FIG. 3.12: Description géométrique de la poutre étudiée

On résoud ce problème avec 1 seul pas de la MAN à l'ordre 20. Ainsi, on doit résoudre avec le solveur multigrille une série de 20 problèmes linéaires à matrice invariante. La méthode multigrille utilisée est celle avec le multiplicateur de Lagrange avec un préconnditionneur incomplet de Cholesky de niveau 0.

Le paramètre de précision choisi pour le solveur mutigrille est de 10^{-10} pour résoudre chaque problème linéaire de la MAN.

L'efficacité du couplage MAN avec solveur multigrille par rapport au couplage MAN avec méthode directe est évaluée en comparant les temps de calcul pour les exemples présentés dans le tableau (3.12).

	Grille grossière	Grille fine	Nombre de division N
Exemple 1	4290	37442	3
Exemple 2	4290	66306	4
Exemple 3	4290	103362	5

TAB. 3.12: *Nombre de d.d.l pour les trois exemples étudiés*

Avant de donner le temps CPU total obtenu avec l'utilisation du solveur multigrille, on précise tout d'abord les différentes étapes réalisées au cours du calcul. L'étape de préparation des données correspond au temps nécessaire pour la construction et la décomposition de la matrice de rigidité de la grille grossière. Le temps dit du solveur du premier système correspond au temps total du solveur à deux grilles y compris le temps de la triangulation incomplète de Cholesky. Cette triangulation n'est réalisée qu'une seule fois, puisque la matrice est la même pour les 20 systèmes linéaires de la MAN. Enfin le temps du solveur multigrille pour résoudre les 19 problèmes linéaires suivants.

En ce qui concerne la méthode directe, on donne le temps de décomposition (méthode de Crout) de la matrice de rigidité de la grille fine (réalisée aussi une seule fois) et le temps de résolution des seconds membres, c'est-à-dire le temps pour les 20 montée/descente de la MAN (pour ces deux méthodes, on a pas pris en compte le temps CPU de construction de la grille fine).

On donne dans les tableaux (3.13) et (3.14), le temps CPU en seconde, nécessaire à la résolution des 20 problèmes linéaires par le solveur multigrille et le solveur direct. On peut voir sur ces tableaux que l'on peut gagner 20% sur le temps CPU pour l'exemple 1 et jusqu'à 45% du temps pour l'exemple 2. Ces résultats montrent que le solveur multigrille couplé avec la factorisation incomplète de Cholesky, donne de très bons résultats en terme de temps CPU total. De plus, les tests numériques présentés ici ont montré que l'utilisation d'un solveur multigrille pour résoudre le problème linéaire choisi conduisait à une diminution de la taille mémoire nécessaire comparée à une méthode directe (la résolution par la méthode directe de l'exemple 3 demande trop d'espace mémoire)

On vient de voir les bénifices apportés par l'utilisation du solveur multigrille sur la résolution d'un pas de calcul par une MAN. Cette technique risque de devenir encore plus intéressante en terme de temps CPU si on varie le nombre de vecteurs pour atteindre la convergence souhaitée avec le solveur multigrille. C'est ce que l'on

	Préparation	CPU (système 1)	CPU (système suivant)	CPU total (s)
Exemple 1	5.27	19.79	12.34	271
Exemple 2	6.22	44.08	24.82	546.4
Exemple 3	7.71	74.17	37.63	834.48

TAB. 3.13: *Temps CPU(s) pour le solveur multigrille, critère d'arrêt :* $\frac{|Residu|}{|F|} \leq 10^{-10}$, *préconditionneur Incomplet de Cholesky de niveau 0, MAN (ordre 20)*

	Décomposition	Résolutions (second membre)	CPU total(s)
Exemple 1	276.29	3.1	338.45
Exemple 2	861.8	7.5	1011.27

TAB. 3.14: *Temps CPU(s) pour la méthode directe (Crout), MAN (ordre 20)*

va voir dans la partie suivante.

3.5.2 Quelle précision pour le solveur multigrille ?

Les calculs précédents ont été faits en demandant une très grande précision pour chaque terme de la série ($\varepsilon = 10^{-10}$). On peut se demander si une telle précision est nécessaire surtout qu'on demande généralement une précision de 10^{-3} environ pour le problème non-linéaire. On rappelle à ce propos qu'il avait été montré, dans le cadre de la méthode de sous-domaine [29], qu'il suffisait de résoudre les problèmes linéaires avec la précision souhaitée pour le problème non-linéaire.

Le test numérique est celui décrit au paragraphe précédent. On considère deux nombres de division : $N = 3$ et $N = 4$. Nous calculons une série tronquée à 20 termes et nous nous intéressons au résidu du problème non-linéaire en fonction du paramètre de charge λ. Nous controlons la précision de la résolution des problèmes linéaires en modifiant le nombre de vecteurs utilisés dans le solveur multigrille (figures (3.13) et (3.14)).

On voit qu'à partir d'un certain nombre de vecteurs (ici 8 vecteurs), la précision de la solution du problème non-linéaire en fin de pas n'est pas affectée par la précision de résolution des problèmes linéaires. L'augmentation de la qualité de la résolution multigrille influe seulement sur la précision du calcul non-linéaire en début de pas.

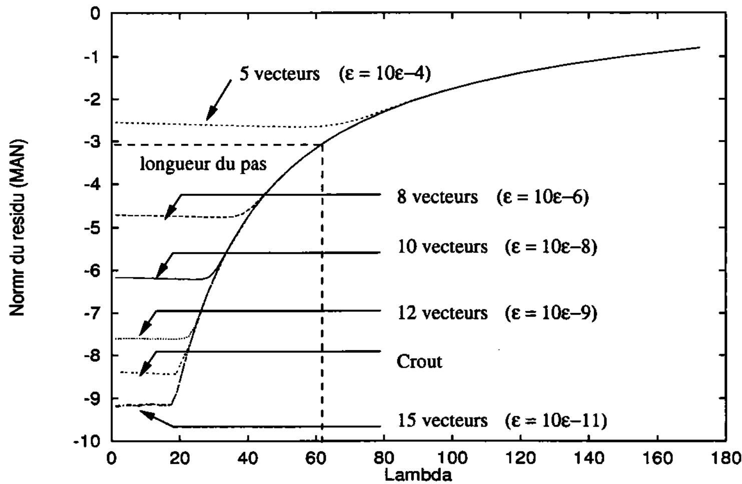


FIG. 3.13: Evolution du résidu MAN (échelle logarithmique, ordre 20) en fonction du paramètre de charge λ, (ordre 20), exemple 1 : (**préconditionneur de Cholesky Incomplet de niveau 0**, **N=3**, grille grossière : 4290 d.d.l, grille fine : 37442 d.d.l. Entre parenthèse, précision du solveur multigrille)

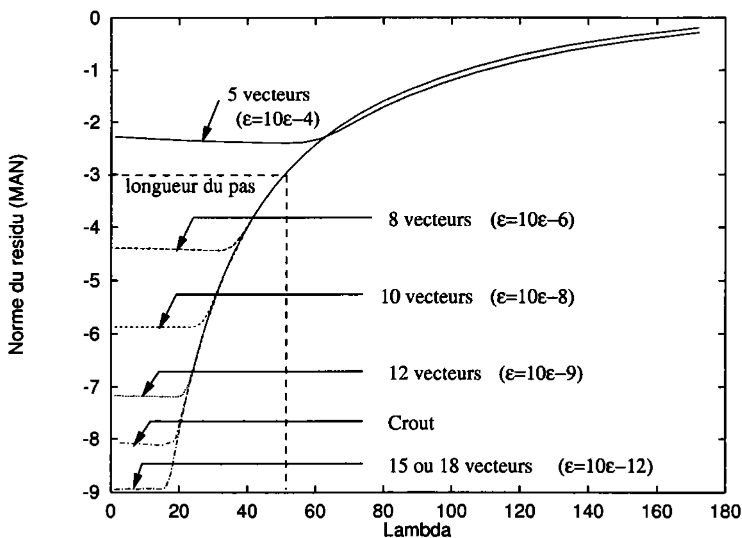


FIG. 3.14: Evolution du résidu MAN (échelle logarithmique, ordre 20) en fonction du paramètre de charge λ, (ordre 20), exemple 2 : (**préconditionneur de Cholesky Incomplet de niveau 0**, **N=4**, grille grossière : 4290 d.d.l, grille fine : 66306 d.d.l. Entre parenthèse, précision du solveur multigrille)

En pratique, il est donc tout à fait inutile de résoudre les problèmes linéaires avec une très grande précision : dans les cas étudiés dans les figures (3.13) et (3.14), 8 vecteurs pour une précision linéaire $\varepsilon = 10^{-6}$ permettent d'obtenir la courbe non-linéaire avec la précision souhaitée (par exemple $\varepsilon = 10^{-3}$), sans modification de la longueur de pas. D'après cette remarque, les temps de calculs présentés dans le tableau (3.13)

peuvent donc être réduits de 30% à 40%.

Une optimisation ou une suite à apporter à ce travail serait d'avoir un nombre de vecteurs évoluant en fonction du problème linéaire à résoudre issu de la MAN : par exemple 10 vecteurs pour les 5 premiers systèmes linéaires, 6 vecteurs pour les 5 suivants et finalement 4 vecteurs pour les derniers.

3.6 Conclusions

Dans ce chapitre, nous avons couplé la factorisation incomplète de Cholesky avec un nouvel algorithme à deux grilles. Cet algorithme est basé sur l'association d'une transformation d'homotopie, d'une technique de perturbation et des approximants de Padé. Nous avons vu que la vitesse de convergence de cette nouvelle technique est quasiment indépendante du pas de discrétisation des maillages. En plus on peut se permettre une grande différence entre deux grilles successives, et les meilleurs résultats sont obtenus avec un nombre de divisions $N = 3$ ou $N = 4$ ou même pour $N = 6$ plutôt qu'avec $N = 2$.

Les nouveaux algorithmes proposés dans ce chapitre associent un très bon solveur (homotopie-séries- Padé) avec un très bon préconditionneur (décomposition incomplète de Cholesky IC(0)) couplée avec une résolution sur grille grossière. En conséquence l'ordre de la série reste petit (inférieur à 30), même avec des maillages non structurés et un rapport 4 ou 5 entre les tailles de grille. L'algorithme classique que nous avons choisi comme référence associe le même préconditionneur et un autre très bon solveur (3 itérations de gradient conjugué à chaque cycle). C'est pourquoi les deux algorithmes ont un coût de calcul quasiment identique. Ce n'était pas le cas au chapitre précédent où la méthode classique employait un lisseur assez simple. L'introduction de ce préconditionneur IC(0) au niveau de la grille fine divise par environ un facteur deux le temps de calcul global : ceci est dû à la diminution de l'ordre de convergence, qui n'est compensé que par une très faible augmentation du coût de la décomposition incomplète.

Nous avons proposé sous forme d'algorithmes, des nouvelles variantes pour résoudre des problèmes issus de maillages non-structurés. Les résultats obtenus ont montré clairement que ces deux variantes ont convergé quel que soit le préconditionnement utilisé. De plus, la deuxième variante couplée avec la factorisation incomplète de Cholesky s'est avérée la plus performante quelle que soit la valeur de N.

En analyse non-linéaire, les résultats obtenus avec le solveur multigrille sont très intéressants soit en terme de temps CPU, soit en terme de qualité de la solution MAN. Ce solveur multigrille coûte beaucoup moins cher qu'une résolution directe, en temps CPU et surtout en termes d'espace mémoire, même pour de problèmes de taille moyenne (environ 100000 d.d.l).

Conclusion générale

Au cours de cette étude, nous avons proposé des nouveaux solveurs à deux grilles basés sur l'association des techniques d'homotopie, de perturbation et les approximants de Padé. Nous avons constaté sur différents exemples que l'utilisation de ces derniers entraîne une accélération de la convergence et que cette convergence est monotone. Par contre la représentation polynomiale est beaucoup plus irrégulière. Dans le cas où un grand nombre de termes des séries est nécessaire, les algorithmes proposés doivent être utilisés de façon itérative (au dela de 30 ou 40).

Deux nouvelles classes de méthodes à deux grilles ont été introduites. La première s'appuie sur une décomposition des variables en variables globales (maillage grossier) et variables locales. Cette méthode ne se couple facilement qu'avec un lisseur diagonal. Dans la seconde classe, on introduit un multiplicateur de Lagrange, ce qui permet d'utiliser toutes sortes de lisseurs. On s'est intéressé en particulier au lisseur issu d'une décomposition incomplète de Cholesky, qui conduit à des algorithmes rapides et fiables.

Ces nouvelles méthodes sont beaucoup plus rapides que les méthodes multigrilles classiques lorsque ces dernières utilisent un lisseur assez simple, comme le lisseur diagonal. En comparaison avec un très bon lisseur comme le gradient conjugué, notre méthode donne quasiment les mêmes résultats. Nous avons testé diverses méthodes pour les cas des maillages non structurés et établi une méthode "optimale"

Une autre application des solveurs bi-grilles a été réalisée sur des problèmes avec seconds membres répétés. Cette première application laisse entrevoir de réelles possibilités et un réel intérêt d'utiliser ces solveurs pour résoudre des problèmes linéaires

issus de la MAN.

Les études réalisées dans cette thèse ont montré que la chose la plus importante lorsque l'on utilise un solveur itératif (méthode multigrille ou gradient conjugué) est le choix du préconditionneur. Définir un bon préconditionneur conduit à un solveur efficace. Parmi les méthodes testées, c'est l'association multigrille - lisseur de Cholesky incomplet qui donnent les meilleurs résultats.

C'est pourquoi une suite logique de ce travail est l'étude de nouveaux préconditionneurs. Pour la résolution d'un problème non-linéaire, un bon préconditionneur serait par exemple la matrice de rigidité.

Une stratégie simple serait de résoudre le premier pas de la MAN avec une méthode directe et de résoudre les pas suivants avec une méthode multigrille, où le lisseur k^* serait la matrice triangulée au premier pas.

Une autres idée serait d'appliquer un préconditionnement non-linéaire et non plus un lisseur linéaire k^*. Par exemple on pourrait utiliser la méthode du gradient (avec line search) ou du gradient conjugué (avec line search et orthogonalisation des directions de descente), ce qui serait comparable à l'algorithme classique MBC testé au chapitre 4.

Pour continuer les recherches présentées ici, il semple inévitable de travailler avec plusieurs niveaux de grille. A titre d'exemple, un algorithme possible à trois grilles est présenté en annexe A.

Une autre perspective à ce travail serait également l'application des méthodes multigrilles définies dans cette thèse sur des problèmes de coques où la difficulté réside dans le mauvais conditionnement des matrices, dans la définition des opérateurs de transfert (opérateurs de prolongement et de restriction),....etc.

L'application des solveurs multigrilles avec des techniques de prédiction-correction d'ordre élevé était le but de notre travail à moyen terme. Dans cette thèse, nous avons montré qu'on pouvait gagner beaucoup de temps et surtout de mémoire en remplaçant à l'étape de la prédiction le solveur direct par un solveur multigrille.

Ces résultats restent à confirmer et à optimiser. En ce qui concerne la correction, on peut envisager d'associer des techniques de correction non-linéaire d'ordre élevé [46, 41] et des préconditionneurs multigrilles, pour obtenir des solveurs non-linéaires bon marché. Ce travail reste à faire, mais cette thèse a permis d'avancer nettement dans cette direction.

ANNEXE A

Méthode d'ordre élevé à trois grilles

A.1 Objectif

Dans cette thèse, le choix a été fait de se concentrer sur des algorithmes à deux grilles. Si on vise à mettre au point des algorithmes généraux et optimaux, il faut prévoir la possibilité d'algorithmes avec plusieurs grilles. Dans cette annexe, on montre une manière assez simple de combiner l'algorithme du chapitre 2 avec celui du chapitre 3, pour obtenir un algorithme à 3 grilles. Ceci est présenté à titre d'exemple, car il y a certainement d'autres techniques possibles.

On combine les deux algorithmes précédents à un procédé à trois grilles pour résoudre le système linéaire :

$$k.u = f \tag{A.1}$$

Dans ce système supposé de très grande taille, k est une matrice sysmétrique définie positive de dimension n, u est le vecteur inconnu et f le chargement.

On rappelle que le premier algorithme utilise un préconditionneur diagonal, le deuxième algorithme avec multiplicateur de Lagrange utilise le préconditionnement par la factorisation incomplète de Cholesky de niveau 0.

A.2 Technique de la méthode

Soient les trois grilles de la figure (A.1) :

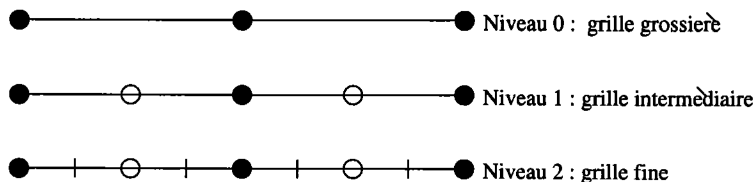


FIG. A.1: Trois grilles : grille grossière (niveau 0), grille intermédiaire (niveau 1) et la grille fine (niveau 2)

Les inconnues sur chaque niveau de grilles sont : pour la grille grossière (niveau 0) :

$$q_g = Q \tag{A.2}$$

pour la grille intermédiaire (niveau 1) :

$$q_I = \left\{ \begin{array}{c} q_g \\ q_{L1} \end{array} \right\} \tag{A.3}$$

et pour la grille fine (niveau 2) :

$$q = \left\{ \begin{array}{c} q_I \\ q_{L2} \end{array} \right\} = \left\{ \begin{array}{c} Q = q_g \\ q_{L1} \\ q_{L2} \end{array} \right\} \tag{A.4}$$

A.3 Méthode proposée

La nouvelle méthode à trois grilles permet de résoudre le problème (A.1) posé sur la grille fine (niveau 2) à partir de deux autres grilles moins fines.

Comme la grille fine contient beaucoup de degrés de liberté, l'idée est de calculer seulement la diagonale de la matrice de rigidité ce qui nous permettra d'éviter le stockage de la matrice. Donc pour passer de la grille fine à la grille intermédiaire, on utilise la première méthode (procédé 1) à deux grilles avec préconditionnement diagonal.

Ensuite pour passer de la grille 1 à la grille grossière on applique la deuxième méthode (procédé 2) avec multiplicateur de Lagrange.

Le procédé 1 implique avec les inconnues q_I et q_{L2}, la décomposition suivantes des

équations :

$$^tP_2.k.q = F_I =^t P_2.f \tag{A.5}$$

$$[k.q - f]_{L2} = 0 \tag{A.6}$$

Avec les notations suivantes :

P_2 : opérateur de prolongement de la grille 1 à la grille de niveau 2.

f_I : force réduite sur la grille intermédiaire de niveau 1. $[\quad]_{L2}$: indique une restriction aux noeuds locaux de la grille fine (indiqués par un tiret, figure A.1).

Maintenant pour passer du niveau 1 au niveau 0, on applique le procédé 2. L'équation (A.5) peut se réecrire sous la forme :

$$^tP_2.k.(P_2.q_I + q_{L2}) =^t P_2.f \tag{A.7}$$

ce qui implique :

$$k_I.q_I +^t P_2.k.q_{L2} = f_I \tag{A.8}$$

avec $k_I =^t P_2.k.q_I$ est la matrice de rigidité de la grille intermédiaire (niveau 1).

En résumé, résoudre le problème (A.1) revient à résoudre le problème suivant :

$$[k.q - f]_{L2} = 0 \tag{A.9}$$

$$k_I.q_I +^t P_2.k.q_{L2} = f_I \tag{A.10}$$

et (A.10) peut être remplacé par les équations du système augmenté du procédé 2 (passage de niveau 1 au nivau 0), ce qui donne le nouveau système augmenté pour le procédé à trois grilles :

$$\begin{cases} [k.q - f]_{L2} = 0 & \\ C(S.q_I - Q) & = \Lambda \\ k_I^*.q_I +^t S.\Lambda - f_I & = 0 \\ K.Q +^t P_I.k_I.v_I - \Lambda & = F \\ v_I = q -^t P_1.Q & \end{cases} \tag{A.11}$$

avec les notations suivantes :

S : opérateur de restriction du niveau 1 au niveau 0.

P_1 : opérateur de prolongement du niveau 0 au niveau 1.

F : force réduite sur la grille grossière : niveau 0.

$K =^t P_1.k_I.P_1$: matrice réduite sur la grille grossière de niveau 0.

Int_2 : opérateur d'interpolation sur la grille fine.

C_D : préconditionneur diagonal (inverse de la matrice diagonale locale).

C : préconditionneur pour le multiplicateur de Lagrange, dans notre exemple : $C = \alpha K$.

k_I^* : préconditionneur sur la grille intermédiaire, par exemple la factorisation incomplète de Cholesky.

Techniques d'homotopie et de perturbation

En appliquant la transformation d'homotopie au sysème augmenté (A.11), on obtient le système suivant :

$$\left\{ \begin{array}{c} (1-\varepsilon)[q_{L2} - Int_2(q_I)] + \varepsilon C_D.[k.q - f]_{L2} = 0 \\ \varepsilon C(S.q_I - Q) = \Lambda \\ k_I.q_I +^t S.\Lambda - f_I = 0 \\ (1-\varepsilon)k_I^*.v_I + \varepsilon[k_I.q_I - f_I] +^t S.\Lambda = 0 \\ K.Q +^t P_1.k_I.v_I - \Lambda = F \\ v_I = q_I - P_1.Q \end{array} \right. \tag{A.12}$$

En suite on développe les vecteurs inconnus du système (A.12) en puissance de ε, et après identification terme à terme suivant les puissances de ε on obtient une série de problèmes à résoudre :

A l'ordre 0 :

$$\left\{ \begin{array}{l} \Lambda(0) = 0 \\ v_I(0) = 0 \\ K.Q(0) = F \\ q_I(0) = P1.Q(0) \\ q_{L2}(0) = Int_2(q_I(0)) \end{array} \right. \tag{A.13}$$

A l'ordre 1 :

$$\begin{cases} \Lambda(1) = 0 \\ v_I(1) = v_I(0) - k_I^{*-1}[k_I.q_I(0) - f_I] \\ K.Q(1) = -{}^tP_1.k_I.q_I(1) \\ q_I(1) = v_I(1) + P_1.Q(1) \\ v_I(1) = Int_2(q_I(1)) - C_D.[k.q(0) - f]_{L2} \end{cases} \tag{A.14}$$

A l'ordre $i \geq 2$:

$$\begin{cases} \Lambda(i+1) = C.[S.q_I(i) - Q(i)] \\ v_I(i+1) = v_I(i) - k_I^{*-1}[k_I.q_I(i) +^t S.\Lambda(i+1)] \\ K.Q(i+1) = -{}^tP_1.k_I.v_I(i+1) +^t S\Lambda(i+1) \\ q_I(i+1) = v_I(i+1) + P_1.Q(i+1) \\ v_I(i+1) = Int_2(q_I(i+1)) + q_{L2}(i) - Int_2(q_I(i)) - C_D.[k.q(i)]_{L2} \end{cases} \tag{A.15}$$

On remarque que tous ces problèmes utilisent trois inversions de matrice : la matrice de rigidité de la grille grossière K, le préconditionneur sur la grille intermédiaire k_I^* et la matrice diagonale sur la grille fine C_I ou C.

ANNEXE B

Construction de l'opérateur de prolongement P

L'opérateur de prolongement P a été construit à l'aide des fonctions de forme de l'élément fini quadrangle à quatre noeuds et 2 d.d.l par noeud.

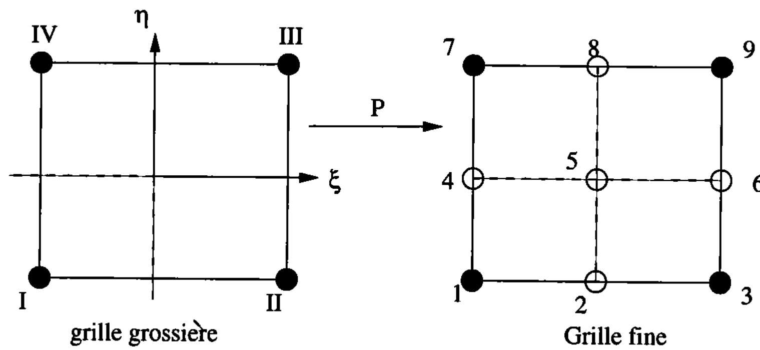


FIG. B.1: Construction de l'opérateur de prolongement P dans le cas ou $N = 2$

Nous rappelons ici les fonctions d'interpolation de cet élément dans le repère de référence (ξ, η) :

$N_I(\xi, \eta) = 1/4(1 - \xi)(1 - \eta)$

$N_{II}(\xi, \eta) = 1/4(1 - \xi)(1 + \eta)$

$N_{III}(\xi, \eta) = 1/4(1 + \xi)(1 + \eta)$

$N_{IV}(\xi, \eta) = 1/4(1 + \xi)(1 - \eta)$

Par conséquent les déplacements des noeuds de la grille fine s'écrivent :

$u_1(\xi = -1, \eta = -1) = N_I.U_I + N_{II}.U_{II} + N_{III}.U_{III} + N_{IV}.U_{IV}$

avec :

$N_I(\xi = -1, \eta = -1) = 1$

$N_{II}(\xi = -1, \eta = -1) = 0$

$N_{III}(\xi = -1, \eta = -1) = 0$

$N_{IV}(\xi = -1, \eta = -1) = 0$

ce qui donne :

$u_1(\xi = -1, \eta = -1) = 1.U_I$

De même pour le déplacement du noeud 2 de la grille fine :

$u_2(\xi = -1, \eta = 0) = N_I.U_I + N_{II}.U_{II} + N_{III}.U_{III} + N_{IV}.U_{IV}$

avec :

$N_I(\xi = -1, \eta = 0) = 1/2$

$N_{II}(\xi = -1, \eta = 0) = 1/2$

$N_{III}(\xi = -1, \eta = 0) = 0$

$N_{IV}(\xi = -1, \eta = 0) = 0$

et donc :

$u_2(\xi = -1, \eta = 0) = 1/2.U_I + 1/2.UII$

Pour le noeud 5 de la grille fine, le déplacement s'écrit sous la forme :

$u_5(\xi = 0, \eta = 0) = N_I.U_I + N_{II}.U_{II} + N_{III}.U_{III} + N_{IV}.U_{IV}$

et les fonctions de forme s'écrivent :

$N_I(\xi = 0, \eta = 0) = 1/4$

$N_{II}(\xi = 0, \eta = 0) = 1/4$

$N_{III}(\xi = 0, \eta = 0) = 1/4$

$N_{IV}(\xi = 0, \eta = 0) = 1/4$

finallement :

$u_5(\xi = 0, \eta = 0) = 1/4.U_I + 1/4.U_{II} + 1/4.U_{III} + 1/4.UIV$

Par généralisation du procédé précédent, on obtient pour tous les noeuds de la grille fine les formules suivantes :

$$
\begin{aligned}
u_1 &= 1.U_I \\
u_2 &= 1/2(U_I + U_{II}) \\
u_3 &= 1.U_I I \\
u_4 &= 1/2(U_I + U_{IV}) \\
u_5 &= 1/4(U_I + U_{II} + U_{III} + U_{IV}) \\
u_6 &= 1/2(U_{II} + U_{III}) \\
u_7 &= 1.U_I V \\
u_8 &= 1/2(U_{III} + U_{IV}) \\
u_9 &= 1.U_{III}
\end{aligned}
\tag{B.1}
$$

et donc l'opérateur de prolongement s'écrit :

$$\{\ u_f\} \ = [P].\{\ u_g\ \} \tag{B.2}$$

avec :

$$
P = \begin{bmatrix}
1 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1/2 & 0 & 0 & 1/2 \\
1/4 & 1/4 & 1/4 & 1/4 \\
0 & 1/2 & 1/2 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 1 & 0
\end{bmatrix}
$$

Bibliographie

[1] **H.Abichou, H.Zahrouni, M.Potier-ferry,** " *Asymptotic numerical method for problems coupling several non linearities* ", *A paraître dans Computer Methods in Applied Mechanics and Engineering.*

[2] **G.D.Astrachancev** , " *An iterative method of solving elliptic net problems* ", *USSR Comp. Math. Math. Phys, vol. 11, Pp. 171-182, 1971.*

[3] **C.Auburtin,** "*Analyse linéaire et non-linéaire des structures poutres planes par éléments finis et méthodes multigrilles sur micro-ordinateur*", *Thèse de doctorat, Institut National Polytechnique de Lorraine, Nancy, 1990.*

[4] **L.Azrar, B.Cochelin,N.Damil,M.Potier-Ferry** , "*An asymptotic numerical method to compute the postbuckling behaviour of elastic plates and shells* ", *International Journal for Numerical Methods in Engineering, vol. 36, Pp. 1251-1277, 1993.*

[5] **G.A.Baker, P.G.Morris** , "*Padé approximants, Encyclopedia of mathematics and its applications* ", *Cambridge university Press, Cambridge, 2nd edition, 1996.*

[6] **M.Barboteu** , "*Contact, Frottement et Techniques de Calcul Parallèle* ", *Thèse de doctorat, Université Montpellier II, 1999.*

[7] **A.Behit, P.A.Forsyth** , "*Practical Considerations for Incomplete Factorization Methods in Reservoir Simulation* ", *Society of Petroleum Engineers Journal, 1983.*

[8] **A.Brandt** , " *Multilevel adaptive technique (MLAT) for fast numérical solution to bondary value problems*", *Lecture notes in Phys. Springer-Verlag, vol. 18, Pp.*

82-89, 1973.

[9] **A.Brandt** , " *A multilevel adaptive solutions to bondary value problems*", *Math. Comp, vol. 31, Pp. 333-390, 1977.*

[10] **A.Brandt** , " *Guide to multigrid development, multigrid methods*", *Lecture notes in Math. Springer-Verlag, vol. 960, Pp. 220-312, 1982.*

[11] **C.Brezinski, V.Iseghem** , " *In Padé approximants, Handbook of numerical Analysis* ", *Ciarlet PG, Lions JL (eds), vol. 3, North-Holland : Amsterdam, 1994.*

[12] **J.Brunelot** , "*Simulation de la mise en forme à chaud par la Méthode Asymptotique-Numérique*", *Thèse de Doctorat, Université de Metz, 1999.*

[13] **J.M.Cadou, N.Damil, M.Potier-Ferry, B.Braikat** , " *Projection tehniques to improve high order iterative correctors* ", *Soumis à Finite Element in Analysis and Design*

[14] **J.M.Cadou, N.Moustaghfir, EH.Mallil, N.Damil, M.Potier-Ferry** , " *Linear iterative solvers based on perturbation techniques* ", *Comptes Rendus de l'Académie des Sciences Paris, Série II-b t.329 : Pp. 457-462, 2001.*

[15] **J.M.Cadou, M.Potier-Ferry, B.Cochelin, N.Damil** , "*ANM for stationary Navier-Stokes equations and with Petrov-Galerkin formulation*", *International Journal for Numerical Methods in Engineering, vol. 50, Pp. 825-845, 2001.*

[16] **P.Ciarlet** , " *Introduction à l'analyse numérique matricielle et à l'optimisation*", *Masson, 1982.*

[17] **B.Cochelin** , " *A path-following technique via an asymptotic-numerical method* ", *Computers and structures, vol. 53 n5, Pp. 1181-1192, 1994.*

[18] **B.Cochelin** , "*Méthodes Asymptotiques-Numériques pour le calcul non-linéaire géométrique des structures élastiques* ", *Habilitation à diriger des recherches, Université de Metz, 1994.*

[19] **B.Cochelin, N.Damil, M.Potier-Ferry** , *"Asymptotic numerical methods and Padé approximants for nonlinear elastic structures "*, *International Journal for Numerical Methods in Engineering, vol. 37, n5, Pp. 1187-1213, 1994.*

[20] **B.Cochelin, N.Damil, M.Potier-Ferry** , *"The asymptotic numerical method : an efficient perturbation technique for non linear structural mechanics "*, *Revue Européenne des Eléments Finis, vol. 3, n2, Pp. 281-297, 1994.*

[21] **N.Damil, M.Potier-Ferry** , *" A new method to compute pertured bifurcations : Application to the buckling of imperfect elastic structures "*, *International Journal in Engineering Sciences, vol. 28, n9, Pp. 943-957, 1990.*

[22] **N.Damil, M.Potier-Ferry, A.Najah, R.Chari, H.Lahmam** , *" An iterative method based upon Padé approximants "*, *Communications in Numerical Methods in Engineering, vol. 15, Pp. 701-708, 1999.*

[23] **Y.Escaig** , *"Décomposition de domaine multiniveaux et traitements distribués pour la résolution de problémes de grande taille "*, *Thèse de doctorat, Université de Technologie de Compiène, 1992.*

[24] **M.Essakhi, B.Braikat, H.Lahmam, J.M.Cadou, N.Damil, M.Potier-Ferry** , *"Sous-structuration et méthode asymptotique-numérique en élasticité non-linéaire "*, *Soumis à la Revue Européenne des Eléments Finis.*

[25] **C.Farhat, L.Crivelli, F.X.Roux** , *"Extending substructure based iterative solvers to multiple load and repeated analyses "*, *Computer Methods in Applied Mechanics and Engineering, vol. 117 Pp. 195-209, 1994.*

[26] **C.Farhat, P.S.Chen, F.Risler, F.X.Roux** , *"A unified framework for accelerating the convergence of iterative substructuring methods with Lagrange multipliers "*, *International Journal for numerical methods in Engineering, vol. 42 Pp. 257-288, 1998.*

[27] **R.P.Fedorenko** , *" A relaxation method for solving elliptic difference equations "*, *USSR Comp. Math. and Math. Phys, vol. 1, Pp. 1092-1096, 1962.*

[28] **R.P.Fedorenko** , " *The speed of convergence of one iterative process* ", *USSR Comp. Math. and Math. Phys, vol. 4, Pp. 227-235, 1964.*

[29] **I.Galliet**, "*Une version parallèle des méthodes asymptotiques-numériques. Application à des structures complexes à base d'élastomères*", *Thèse de doctorat, Université de Marseille II, 1996.*

[30] **L.Grego** "*Méthodes multiniveaux pour des problèmes de contact unilatéral avec frottement*", *Thèse de doctorat, Université de Provence, Marseille, 1995.*

[31] **W.Hackbusch** , " *Multigrid methods and applications*", *Springer, 1971.*

[32] **S.Hadji** , "*Méthode de résolution pour les fluides incompréssibles* ", *Thèse de doctorat, Université de Technologie de Compiegne, 1995.*

[33] **C.Hirsch** , " *Numerical computation of internal and external flows*", *Wiley, New York, vol. 1, 1988.*

[34] **A.E.Hussein, N.Damil, M.Potier-Ferry** , "*A asymptotic numerical algorithm for frictionless contact problems* ", *Revue Européenne des éléments finis, vol. 7, n1-2-3, Pp. 119-130, 1998.*

[35] **A.E.Hussein, N.Damil, M.Potier-Ferry** , "*A numerical continuation method based on Padé approximants* ", *International journal of solids and structures, vol. 37, Pp. 6981-7001, 2000.*

[36] **M.Jamal, B.Braikat, S.Boutmir, N.Damil, M.Potier-Ferry** , " *A high order implicit algorithm for solving nonlinear problems* ", *Computational Mechanics, vol. 28, Pp. 375-380, 2002.*

[37] **A.Jennings** , "*A compact storage scheme for the solution of symmetric linear simultaneous equations* ", *Computing Journal, vol. 9, 1966.*

[38] **P.Joly** , " *Mise en oeuvre de la méthode des éléments finis*", *INRIA-Rocquencourt, 1994.*

[39] **P.Joly, M.Vidrascu** , " *Quelques méthodes classiques de résolution de systèmes linéaires*", *(Eds),INRIA-Collection-Didactique, 1994.*

[40] **N.Kessab** , "*Un algorithme de prédiction-correction d'ordre élevé basé sur une linéarisation partielle pour les problèmes fortement non-linéaires* ", *Thèse de 3ème cycle, Université Hassan II Mohammeddia, Casablanca Maroc, 2001*

[41] **H.Lahmam, J.M.Cadou, H.Zahrouni, N.Damil, M.Potier-Ferry** , " *High order predictor algorithms* ", *International Journal of Numerical Methods in Engineering, vol. 190, Pp. 1845-1858, 2002.*

[42] **P.Lascaux, R.Thèodor** , " *Analyse numérique matricielle appliquée à l'art de l'ingénieur*", *Tome 1, 2eme Edition, Masson, Paris Milan Barcelone Boon, 1993.*

[43] **P.Lascaux, R.Thèodor** , " *Analyse numérique matricielle appliquée à l'art de l'ingénieur*", *Tome 2, 2eme Edition, Masson, Paris Milan Barcelone Boon, 1993.*

[44] **F.Lebon** "*Résolution numérique de problèmes de frottement de Coulomb. Accélération de convergence pour une méthode multigrille interne*", *Thèse de doctorat, Université de Provence, Marseille, 1989.*

[45] **P.Le Tallec et M.Vidrascu** , "*Méthodes de décomposition de domaines en calcul des structures* ", *Premier Colloque National en Calcul des Structures, Giens, 1993.*

[46] **E.Mallil, H.Lahmam, N.Damil, M.Potier-Ferry** , " *An iterative process based on homotopy and perturbation techniques* ", *Computer Methods in Applied Mechanics and Engineering, vol. 190, Pp. 1845-1858, 2000.*

[47] **J.A.Meijerink, H.A.Van Der Vorst** , "*Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems* ", *Journal of Computational Physics, vol. 44, 1981.*

[48] **A.Najah, B.Cochelin, N.Damil, M.Potier-Ferry** , "*A critical review of Asymptotic Numerical Methods* ", *Archives of Computational Methods in Engineering, vol. 5, n1, Pp. 31-50, 1998.*

[49] **J.S Prezemienieski** , "*Matrix structural analysis of substructures* ", *Am. Inst. Aero. Astro. J, vol. 1, Pp. 138-147, 1963.*

[50] **C.Rey**, "*Une technique d'accélération de la résolution de problèmes d'élasticité non linéaire par décomposition de domaines* ", *Comptes Rendus de l'Académie des sciences, Paris, Série II-b t.322, Pp. 601-606, 1996.*

[51] **F.Risler, C.Rey**, "*Iterative accelerating algorithms with Krylov sub-spaces for the solution to large-scale nonlinear problems* ", *Numerical algorithms*

[52] **F.X Roux** , "*Méthodes de résolution par sous-domaines en statique* ", *La Recherche Aérospatiale, vol. 1, Pp. 37-48, 1990.*

[53] **D.Soulat** , "*Méthodes de décomposition de domaines et parallélisme en calcul de structures hétérogènes et composites* ", *Thèse de doctorat, Université Pierre et Marie Curie, Paris, 1996.*

[54] **J.R.Wallis** , "*Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration* ", *Proceeding of the Seventh Symposium on Numerical of Reservoir Performance of the Society of Petroleum Engineers, 1983.*

[55] **P.Wesseling** , " *An introduction to multigrid methods*", *John Wiley et Sons, 1992.*

[56] **H.Zahrouni, B.Cochelin, M.Potier-Ferry** , "*Computing finite rotations of shells by an asymptotic-numerical method* ", *Computer methods in applied mechanics and engineering, vol. 175, Pp. 71-85, 1999.*

[57] **H.Zahrouni, M.Potier-Ferry, H.Elasmar, N.Damil** , "*Asymptotic numerical method for nonlinear constituve laws* ", *Revus Européenne des eléments Finis, vol. 7, Pp. 841-869, 1998.*

Liste des figures

1.1 Toit : description géométrique 16

1.2 cylindre : description géométrique 16

1.3 Gradient conjugué préconditionné. Evolution du nombre d'itérations en fonction du niveau de factorisation incomplète de Cholesky (**exemple à 5190 d.d.l.**) 28

1.4 Evolution du nombre d'itérations du gradient conjugué en fonction du niveau de factorisation incomplète de Cholesky (**exemple à 40000 d.d.l.**) 28

1.5 Evolution du temps CPU de la factorisation incomplète en fonction du niveau de factorisation 29

1.6 Evolution du ratio : nombre de réels de C sur nombre de réels de A (ou k) en fonction du niveau 30

1.7 Evolution du nombre d'itérations en fonction de l'ordre de troncature de la M.A.N. (exemple à 5190 d.d.l.) 33

1.8 Exemple d'opérateur de prolongement P 38

1.9 Exemple d'opérateur de restriction 39

1.10 Schéma d'un V-cycle 46

1.11 Schéma d'un W-cycle 47

1.12 V-cycle d'initialisation 48

1.13 W-cycle d'initialisation 48

2.1 Exemple de maillages : Ω_H est la grille grossière et Ω_h est la grille fine 53

2.2 Description géométrique et mécanique de la plaque étudiée 62
2.3 Deux types de maillage (a) et (b) 63
2.4 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature, test 3 (N=2, Grille fine : 132098 d.d.l) 66
2.5 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature, test 5 (N=3, Grille fine : 119072 d.d.l) 66
2.6 Logarithme du résidu Padé en fonction des ordres de troncature, pour $N = 4$, la grille grossière : 4232 d.d.l, pour $N = 6$, la grille grossière : 1922 d.d.l. 72
2.7 Algorithme itératif proposé 73
2.8 Logarithme du résidu Padé en fonction des ordres de troncature, l'ordre de troncature est fixé à 30. 75
2.9 Evolution du nombre de vecteurs en fonction du nombre de division N pour un ordre de troncature fixé à 30. 76
2.10 Evolution du temps CPU (s) en fonction du nombre de division N pour des ordres de troncature fixés à 20, 30 et 40. 77
2.11 Evolution du temps CPU en secondes de construction des vecteurs Padé en fonction de l'ordre de troncature de l'algorithme à 2-grilles (N=3, grille grossière : 16562 d.d.l, grille fine : 146882 d.d.l, convergence obtenue à l'ordre 63) 78
2.12 Evolution du temps relatif en fonction de l'ordre de troncature (temps relatif = temps CPU de construction des Padé/temps CPU de l'algorithme à 2-grilles) pour deux exemples : premier exemple, N=3, grille fine : 18818 d.d.l et deuxième exemple, N=3, grille fine : 146882 d.d.l, convergence de l'algorithme obtenue à 63) 79
2.13 Evolution du nombre de cycles de la méthode à 2-grilles classique en fonction du nombre de division N, lisseur de Jacobi, $\omega = 0.8$, résolution itérative 80

2.14 Evolution du temps CPU total en secondes de la méthode à 2-grilles classique (Jacobi, $\omega = 0.8$) et de la méthode proposée (ordre de troncature fixé à 30) en fonction du nombre de division N, résolution itérative 81

2.15 Maillage de la plaque avec deux matériaux différents 82

2.16 Exemple de la plaque soumise au chargement F 83

3.1 Logarithme décimal du vecteur résidu (Padé) en fonction de l'ordre I pour les tests 3 et 5 95

3.2 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 **préconditionneur diagonal, N=2** , grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l . 100

3.3 Algorithme à deux grilles **itératif**, logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 **préconditionneur diagonal, N=2**, grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l 101

3.4 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=2**, grille grossière : 2178 d.d.l, grille fine : 8450 d.d.l) 101

3.5 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour diffŕentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=3**, grille grossière : 2178 d.d.l, grille fine : 18818 d.d.l) 102

3.6 Logarithme décimal de la norme du vecteur résidu en fonction de l'ordre de troncature pour différentes valeurs de α, test 1 (**préconditionneur incomplet de Cholesky niveau 0, N=4**, grille grossière : 2178 d.d.l, grille fine : 33282 d.d.l) 102

3.7 Exemple de plaque soumise à un chargement de pression 107

3.8 Logarithme du résidu Padé en fonction des ordres de troncature, paramètre de précision : $\varepsilon = 10^{-10}$ 108

3.9 Logarithme du résidu Padé en fonction des ordres de troncature, paramètre de précision : $\varepsilon = 10^{-10}$ 108

3.10 Evolution du résidu Padé en fonction des ordres de troncature 109

3.11 Evolution du résidu Padé en fonction des ordres de troncature 109

3.12 Description géométrique de la poutre étudiée 116

3.13 Evolution du résidu MAN (échelle logarithmique, ordre 20) en fonction du paramètre de charge λ, (ordre 20), exemple 1 : (**préconditionneur de Cholesky Incomplet de niveau 0**, **N=3**, grille grossière : 4290 d.d.l, grille fine : 37442 d.d.l. Entre parenthèse, précision du solveur multigrille) 119

3.14 Evolution du résidu MAN (échelle logarithmique, ordre 20) en fonction du paramètre de charge λ, (ordre 20), exemple 2 : (**préconditionneur de Cholesky Incomplet de niveau 0**, **N=4**, grille grossière : 4290 d.d.l, grille fine : 66306 d.d.l. Entre parenthèse, précision du solveur multigrille) 119

A.1 Trois grilles : grille grossière (niveau 0), grille intermédiaire (niveau 1) et la grille fine (niveau 2) 126

B.1 Construction de l'opérateur de prolongement P dans le cas ou $N = 2$ 130

Liste des tableaux

1.1 Temps CPU pour une décomposition de la matrice de rigidité par la méthode de Crout. Eléments finis de coques à 6 d.d.l par noeud (triangles DKT 18) 17

1.2 Nombre de réels stockés de la matrice A, suivant le stockage profil. Pourcentage de coefficients nuls stockés sous forme profil. Les modèles numériques sont des modèles de coques à 6 d.d.l par noeuds. 17

1.3 Nombre d'itérations et temps C.P.U. pour la méthode du gradient conjugué. Test d'arrêt : $\| r^{k+1} \| / \| b \| \leq \varepsilon = 10^{-10}$ 23

1.4 Nombre d'itérations et temps CPU pour la méthode du gradient conjugué avec préconditionnement diagonal 26

1.5 Temps CPU en seconde et nombre d'itérations pour chaque ordre de la MAN. Résolution par la méthode du gradient conjugué préconditionné : Cholesky incomplet de niveau 0, avec et sans application de l'algorithme de Farhat pour l'exemple du cylindre à $5190 d.d.l.$ Trois stratégies sont étudiées, première stratégie : sans utiliser la méthode de Farhat, deuxième stratégie : 50 directions stockés à chaque ordre (résolution) de la MAN, troisième stratégie : toutes les directions de descente sont stockées à chaque ordre (résolution) de la MAN. 34

1.6 Algorithme à 2-grilles classique, résolution itérative, lisseur de Jacobi relaxé($\omega = 0.8$), le test d'arrêt est 10^{-10}, la grille fine étant fixée à $65522 d.d.l.$ 42

2.1 Nombre de réels de la matrice k stockés suivant les deux techniques de stockage . . . 61
2.2 Temps CPU en secondes de construction de la matrice de rigidité selon la technique de stockage . . . 62
2.3 *Nombre de d.d.l pour les cinq exemples étudiés* . . . 64
2.4 *Résidus obtenus pour différents ordres de troncature :* **Test 1** . . . 64
2.5 *Résidus obtenus pour différents ordres de troncature :* **Test 2** . . . 64
2.6 *Résidus obtenus pour différents ordres de troncature :* **Test 3** . . . 65
2.7 *Résidus obtenus pour différents ordres de troncature :* **Test 4** . . . 65
2.8 *Résidus obtenus pour différents ordres de troncature :* **Test 5** . . . 65
2.9 *Détail du temps CPU en seconde pour les 5 exemples.* . . . 68
2.10 Temps CPU en secondes des produits matrices-vecteurs pour les deux types de stockage (ligne de ciel et morse). . . . 68
2.11 Ordre de convergence et nombre d'itérations des différentes méthodes. MBP : Méthode Bi-grilles Proposé, résolution asymptotique, préconditionneur diagonal. MBC : Méthode Bi-grilles Classique, résolution itérative, préconditionneur de Jacobi relaxé ($\omega = 0.8$), GCD et GCIC représentent respectivement le gradient conjugué préconditionné soit avec la diagonale, soit avec la factorisation incomplète de Cholesky de niveau 0. . . 69
2.12 Temps CPU en secondes des différentes méthodes utilisées. . . . 70
2.13 Degrés de liberté de la grille grossière, grille fine fixée à $65522 d.d.l.$ Nombre de division N, nombre d'itérations : ITER, ordre de convergence et le temps CPU total en secondes de la méthode à 2-grilles proposée. Résolution itérative avec $I = 30$, préconditionneur diagonal. 77
2.14 Ordre de convergence obtenu pour les matériaux homogène et hétérogène 82
2.15 *Résidus obtenus pour différents ordres de troncature :* **Exemple 1** . . 83
2.16 *Résidus obtenus pour différents ordres de troncature :* **Exemple 2** . . 83

3.1 *Nombre de d.d.l pour les cinqs exemples étudiés* 94

3.2 Ordre de convergence du nouveau algorithme et le résidu Padé correspondant. 94

3.3 **Détail du temps CPU en secondes pour les 5 exemples.** . . . 96

3.4 Ordre de convergence et nombre d'itérations des différents méthodes utilisées. MBP : algorithme proposé à deux grilles, résolution asymptotique, préconditionneur $IC(0)$. MBC : algorithme à deux grilles classique, résolution itérative, lissage par le gradient conjugué couplé avec $IC(0)$ avec 3 itérations de lissage, GCPIC : gradient conjugué prćonditionné couplé avec $IC(0)$, résolution itérative. 98

3.5 Comparaison du temps CPU en secondes des différentes méthodes utilisées. 99

3.6 **Détail du temps CPU en secondes pour différentes valeurs du nombre de division** N 104

3.7 Détail du temps CPU (s) pour différentes valeurs de N 105

3.8 Temps CPU en secondes de la résolution pour les deux types de numérotations : numérotation 2 est obtenue par l'algorithme de Cuthill-Mac Kee . . . 105

3.9 *Préconditionneur diagonal pour les deux variantes (version itérative avec un ordre de troncature fixé à 20, les résultats entre parenthèse sont obtenus pour un ordre fixé à 30. Nombres de vecteurs nécessaires à la convergence et nombre de division* N*. Pour la méthode à deux grilles classique, nombres de cycles nécessaires à la convergence, méthode de Jacobi avec trois itérations de lissage* 113

3.10 *Préconditionneur incomplet de Cholesky IC(0) pour les deux variantes, résultats obtenus pour un ordre de troncature fixé à 30 (valeurs entre parenthèse), les autres sont obtenus pour un ordre de troncature fixé à 20. Nombres de vecteurs nécessaires à la convergence et nombre de division N, nombres de cycles nécessaires à la convergence de la méthode classique, la méthode du lissage est le gradient conjugué préconditionné par la factorisation incomplète de Cholesky avec trois itérations de lissage.* 114

3.11 *Ordre de convergence des deux variantes à des maillages structurés, préconditionneur incomplet de Cholesky de niveau 0* 115

3.12 *Nombre de d.d.l pour les trois exemples étudiés* 116

3.13 *Temps CPU(s) pour le solveur multigrille, critère d'arrêt :* $\frac{|Residu|}{|F|} \leq 10^{-10}$*, préconditionneur Incomplet de Cholesky de niveau 0, MAN (ordre 20)* 118

3.14 *Temps CPU(s) pour la méthode directe (Crout), MAN (ordre 20)* . . 118

Solveurs multigrilles et méthode de perturbation

Dans ce travail de thèse, nous avons proposé deux nouvelles classes d'algorithmes à deux grilles pour résoudre les problèmes d'élasticité de grande taille. Ces méthodes sont basées sur l'association des techniques d'homotopie, de perturbation et sur les approximants de Padé.
Un premier algorithme s'appuie sur une décomposition des variables en variables globales (maillage grossier) et variables locales (maillage fin).
Cette méthode ne se couple facilement qu'avec un lisseur diagonal.
Des résultats numériques ont montré que cet algorithme est beaucoup plus rapide que les méthodes multigrilles classiques et également plus rapide que la méthode du gradient conjugué.
Dans une seconde partie, on propose une nouvelle méthode à deux grilles. On introduit un multiplicateur de Lagrange, ce qui permet d'utiliser toutes sorte de lisseurs. On s'est intéressé en particulier au lisseur issu d'une décomposition incomplète de Cholesky, qui conduit à des algorithmes rapides et fiables.
Nous avons testé diverses méthodes pour les cas des maillages curvilignes et établi une méthode « optimale».
Une autre application de ces solveurs a été réalisée sur des problèmes avec seconds membres répétés. Cette première application laisse entrevoir de réelles possibilités et un bon intérêt d'utiliser ces solveurs pour résoudre des problèmes linéaires issus de la Méthode Asymptotique-Numérique (MAN).

Mots clé :
Homotopie – perturbation – approximants de Padé – multigrille – MAN - lissage

A multigrid solvers and perturbation method

In this work, we propose a new class of bi-grid algorithms to solve large scale linear algebraic equations. These methods are based on association on homotopy, perturbation technique and Padé approximants.
A first algorithm is based on a decomposition of the variables in global (coarse meshes) and local (fine meshes) one. This method is easily coupled with a diagonal smoother.
The numerical results showed that this algorithm is most faster than classical multigrid methods and also faster than a conjugate gradient method.
In the second section, we purpose a new bi-grid method. We introduced a Lagrange multipliers, that means used of several type of smoothers.
In particular, we are interested in the preconditioners resulting from incomplete decomposition of Cholesky, which leads to fast and efficient algorithms.
We have tested various methods for the case of the unstructured meshes and established an "optimal" method.
Another first application of these solvers was realized in the case of problems with repeated right hand side. This first application showed real possibilities and a real interest to use these solvers to solve linear problems resulting from the Asymptotic Numerical-method (ANM).

Keywords :
Homotopy, perturbation, Padé approximants, multigrid, ANM, smoother