République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique

Ecole Nationale Polytechnique
Département génie mécanique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Université de Technologie de Compiègne
Laboratoire Roberval

PROJET DE FIN D'ETUDES

Implémentation de la méthode GMRES pour la résolution des problèmes 3D par la méthode des équations intégrales

Proposé et Dirigé par :
P^r S.RECHAK
D^r Y.BELKACEMI
M^r H.KEBIR

Réalisé par :
Yasser DJERAFA

Promotion : 2005/2006

A tous ceux qui me connaisse.

A tous ceux qui m’aime.

A toute ma famille.

A ma très cher mère.

Yasser

Remerciement

Je remercie tous les enseignants du département de génie mécanique de l'Ecole Nationale Polytechnique d'Alger qui ont contribué à ma formation durant ces trois dernières années.

Je remercie monsieur KEBIR docteur chercheur du laboratoire Roberval de l'Université de Technologie de Compiègne qui est l'initiateur et le suiveur de ce travail et l'a rendu possible en m'accueillant au sein de son équipe de chercheurs et en me donnant la chance de travailler directement sur leur logiciel le KSP qui a donné plus de crédibilité à mon travail.

J'exprime ma sincère et profonde gratitude à monsieur BELKACEMI mètre de conférence à l'ENP et monsieur RECHAK professeur à l'ENP qui par leur compétence et leur appui moral ont su à la fois me conseiller et m'aider pour l'achèvement de ce travail ainsi que le rapport.

Je tiens à remercier Messieurs BOUHDJER et TAZI pour avoir accepté d'être président de jurys et examinateur de mon travail, je leur témoigne ma reconnaissance pour leur lecture attentive de mon rapport et leurs suggestions constructives.

Notation

λ, μ : coefficients de Lamé.

ν : coefficient de Poisson.

E : module d'Young.

$[\varepsilon]$: état de déformation.

ε_{ij} : la composante (i,j)de l' état de déformation.

$[\sigma]$: état de contrainte.

σ_{ij} : la composante (i,j)de l' état de contrainte.

$\vec{u}$: vecteur de déplacement.

$\vec{t}$: vecteur de tension.

$\vec{f_v}$: vecteur des forces de volume.

U_i : déplacement suivant la direction e_j.

$U_{i,j} = \frac{\partial U_i}{\partial e_j}$

U_j^i : la composante j du vecteur déplacement suivant la direction e_j.

$U_{j,k}^i = \frac{\partial U_j^i}{\partial e_k}$

T_j^i : la composante j du vecteur tension suivant la direction e_j.

σ_{jk}^i : le tenseur de contrainte lié à T_j^i et $U_{j,k}^i$

Ω : domaine à étudier.

$\partial\Omega$: frontière du domaine à étudier.

Γ : contour d'intégration.

ε : rayon d'isolation d'un point pour le contour.

ملخص

التمثيل الرقمي في الميكانيك يأخذ أهمية أكبر فأكبر في الوسط الصناعي والمسائل المطروحة تصبح أضخم فأضخم، هذا العمل يتمثل في تطوير طريقة حل تكرارية لتسريع كود للتمثيل الرقمي لسلوك الهياكل ذات العدد الكبير من درجات الحرية باستعمال طريقة العنصر الحدودي.

وبما أن جملة المعادلات المطروحة للحل والناتجة عن استعمال هذه الطريقة تمثل بمصفوفة مملوءة وغير متناظرة فقد احترنا الطريقة العامة للباقي الأقل(GMRES) والتي تعتبر الطريقة الأمثل لهدا النوع من المعادلات.

كلمات مفتاح:التمثيل الرقمي،سلوك الهياكل ذات العدد الكبير من درجات الحرية،طريقة العنصر الحدودي،طرق الحل التكرارية،GMRES .

Résumé

La simulation numérique en mécanique prend de plus en plus d'importance dans le milieu industriel et les problèmes à modéliser deviennent de plus en plus grand, ce travail consiste à développer une méthode de résolution itérative pour accélérer un code de simulation numérique du comportement des structures à grand nombre de degrés de liberté par la méthode des équations intégrales.

Puisque le système à résoudre résultant de cette méthode est représenté par une matrice pleine et non symétrique on a choisi la méthode GMRES qui est la plus adaptée pour ce type de système.

Mots clé : simulation numérique, comportement des structures à très grand nombre de ddl, le méthode d'élément frontière, méthode itérative de résolution, GMRES.

Absract

The numerical simulation in mechanics takes more and more importance in the industrial environment and the problems to be modelled become more and more big, this work consists in developing a method of iterative resolution to accelerate a code of numerical simulation of the conduct of the structures with big number of degrees of freedom by the method of boundary element method.

Because the system to be resolved resulting from this method is represented by a full and not symmetric matrix we chose the method GMRES which is the most adapted for this type of system.

Key words: numerical simulation, conduct of structures with big number of dof, boundary element method, iterative method of resolution, GMRES.

Table des matières

Introduction **1**

1 La méthode des équations intégrales **5**

1.1 Introduction 5

1.2 Formulation des équations intégrales 6

1.2.1 Hypothèses 6

1.2.2 Equations de l'élasticité linéaire 7

1.2.3 Solution élémentaire (Problème de Kelvin) 8

1.2.4 Théorème de réciprocité de Maxwell Betti 10

1.2.5 Equations intégrales en déplacements 11

1.2.6 Déplacements des points intérieurs 13

1.2.7 Contraintes des points intérieurs 14

1.2.8 Equations intégrales en tension 14

1.2.9 Contraintes des points du contour 15

1.3 Traitement numérique des équations intégrales 15

1.3.1 Discrétisation de la géométrie 16

1.3.2 Discrétisation des champs élastiques. 17

1.3.3 Discrétisation de l'équation intégrale. 18

1.3.4 Calcul des intégrales. 20

1.3.5 Constitution du système d'équations. 20

1.3.6 Résolution du système d'équations. 21

1.3.7 Calcul des déplacements des points intérieurs. 21

1.3.8 Calcul des contraintes des points intérieurs. 22

1.3.9 Calcul des contraintes des points du contour. 22

1.4 conclusion. 23

2 La POO et le code KSP **24**

2.1 Introduction. 24

2.2 La programmation orienté objet (POO) 24

2.2.1 Petite histoire de l'évolution des méthodes de programmation. 25

2.2.2 Le « modèle objet » 27

2.2.3 Modulariser. 29

2.2.4 L'encapsulation 29

2.2.5 L'intérêt de la POO par rapport au procédural et au fonctionnel 30

2.2.6 Les inconvénients. 30

2.3 Le Microsoft Visual C++. 32
2.4 Le KSP. 32
2.4.1 L'introduction des données. 32
2.4.2 Le traitement. 32
2.4.3 La sortie des résultats. 34

3 La méthode GMRES 35
3.1 Introduction. 35
3.2 Les méthodes de Krylov. 36
3.2.1 Définition de l'espace de Krylov. 36
3.2.2 La propriété de minimisation 37
3.3 Décomposition de Hessenberg en arithmétique exacte. 37
3.3.1 Définition d'une matrice de Hessenberg supérieure. 37
3.3.2 Présentation des décompositions de Hessenberg. 38
3.4 Processus d'Arnoldi. 39
3.4.1 Principe de l'algorithme d'Arnoldi : effectuer une factorisation QR . 39
3.4.2 Trois façons classiques de réaliser la factorisation QR 39
3.4.3 Méthode d'Arnoldi pour la décomposition incomplète de Hessenberg. . . 41
3.4.4 L'arrêt heureux 43
3.4.5 La perte d'orthogonalité 44
3.5 Les rotations de Givens. 44
3.6 GMRES idée de base. 45
3.7 GMRES algorithmes. 47
3.8 Organigramme. 51

4 L'implémentation de l'algorithme GMRES 52
4.1 Introduction. 52
4.2 Le problème de moindre carré. 52
4.3 Calcule de V_j 54
4.4 Calcule de résidu au redémarrage. 55
4.5 Réglage des paramètres 56
4.5.1 Le paramètre m. 56
4.5.2 Le paramètre m_{max}. 59
4.5.3 Le paramètre restart 61
4.6 Le critère d'arrêt. 62
4.7 L'influence de la précision sur le nombre d'itération 64

5. Le préconditionnement 66
5.1 Introduction. 66
5.2 Principe du préconditionnement 66
5.3 Différents types du préconditionnement 67
5.3.1 GMRES hybride 67
5.3.2 Préconditionneur polynomiale. 67
5.3.3 GMRES-DR. 68
5.3.4 décomposition LU Incomplète 68
5.3.5 approximation de l'inverse 68
5.4 Critère d'arrêt 69
5.5 Implémentation du préconditionneur 70
5.6 Algorithme. 72
5.7 Résultats. 72

Conclusion 78

Bibliographie 81

Introduction

La mécanique -théorique ou appliquée- a connu ces 50 dernières années des progrès considérables grâce à l'utilisation de l'ordinateur. Ces développements ont eu un fort impact dans bien des domaines de la société : transport, communication, médecine, environnement, défense, agro-alimentaire… Que ce soit dans le domaine de la mécanique des fluides ou des structures, l'utilisation de l'ordinateur a permis d'améliorer les prédictions, la caractérisation, et la simulation de phénomènes physiques et de systèmes industriels principalement gouvernés par les lois de la mécanique. L'utilisation d'outils informatiques a globalement permis d'améliorer la compréhension et la prédiction de systèmes complexes.

Le calcul numérique en mécanique peut aujourd'hui être considéré comme une discipline à part entière de la mécanique. De nombreuses revues et conférences scientifiques y sont consacrées (Revues : Computational Methods in Applied Mechanics and Engineering, Computers & Structures, Computers & Fluids, Advances in Engineering Software,… Conférences : European Congress on Computational Methods in Applied Sciences and Engineering,…). Le développement des technologies informatiques, autant logicielles que matérielles a conduit à l'émergence de cette nouvelle discipline à caractère naturellement multidisciplinaire : concepts et méthodes issus de la mécanique, des mathématiques, et de l'informatique. Aujourd'hui, ce caractère pluridisciplinaire est encore accentué par l'adjonction de nouvelles disciplines permettant de décrire des phénomènes de plus en plus complexes Il est raisonnable de penser que pour les années à venir le calcul numérique en mécanique prendra une importance de plus en plus grande, d'une part dans l'étude et la compréhension de phénomènes complexes, et d'autre part dans la conception de produits et de systèmes. C'est en particulier dans les interactions avec des disciplines connexes que l'on peut attendre le plus de progrès : biologie, médecine, problèmes liés à l'environnement, aéronautique, aérospatial,… Le chalenge des années à venir est donc véritablement l'intégration de phénomènes à physiques multiples, ces physiques intervenants à des échelles spatiales et temporelles très hétérogènes : du nanomètre à plusieurs milliers de km en espace, de la picoseconde à plusieurs centaines d'année en temps. La conséquence inévitable de cette évolution est qu'il sera nécessaire de développer des stratégies de calcul forcément plus complexes, pour des codes de calculs devant être exécutés sur des grands systèmes informatiques (supercalculateurs, grilles d'ordinateurs,…), et ça à cause de la très grande taille des systèmes d'équations résultants. Le développement d'outils de simulation, efficaces,

précis et fiables implique une maîtrise convenable de ces disciplines, et ce à des niveaux variables. Le développement d'un nouvel outil dont un des aspects est la simulation sous-entend :

- une bonne connaissance des phénomènes physiques étudiés
- de bonnes connaissances des modèles mathématiques utilisés
- une analyse mathématique des modèles afin d'obtenir un problème bien posé
- la vérification des limites des modèles mathématiques
- le développement de stratégies de calcul basées sur un modèle discret et sur des algorithmes numériques, pour approximer le modèle mathématique, et en valider les limites.

Aujourd'hui, ces développements sont rendus plus aisés, plus rapides et plus efficaces par l'utilisation ***'d'outils informatiques de haut niveau'***. D'un point de vue logiciel, ces outils peuvent être regroupés en trois grandes classes. La première correspond à la plus récente génération d'outils de programmation de haut niveau que l'on peut décomposer en deux groupes : les langages de programmation procéduraux classiques (FORTRAN 77, FORTRAN90, PASCAL,...) et les langages orientés objet (SMALLTALK, C++, Java,...). La deuxième classe regroupe tous les logiciels mathématiques de manipulations algébriques tels que Maple, Mathematica, Matlab,... une troisième voie associe les deux premières approches et conduit à une approche où outils de calcul symbolique et outils de calcul numérique sont utilisés conjointement pour la résolution de problème en mécanique.

D'autre part, les méthodes de discrétisation sont aujourd'hui fort nombreuses : éléments finis, volumes finis, différences finies, méthodes spectrales, méthodes 'meshless', collocation,… Leur utilisation dépend souvent des domaines d'application. Leur intégration dans des stratégies globales de résolution sur des systèmes informatiques variés (logiciel et matériel) s'avère être une tâche souvent délicate. Malgré ça, on assiste à plusieurs travaux de recherches depuis plus de 10 ans qui se placent essentiellement dans la conception et le développement d'outils informatiques de simulation numérique pour la mécanique basés surtout sur la méthode des éléments finis. Dans le cas des problèmes 3D, et c'est souvent le cas, cette méthode et plusieurs d'autres méthodes demandent de très grandes capacités d'ordinateurs en mémoire et en vitesse.

Une nouvelle méthode appelée méthode des équations intégrales permet de diminuer une dimension pour la résolution du problème par rapport aux autres méthodes. Ainsi elle nous permet de gagner en mémoire et d'élargir l'horizon des problèmes possible à traités. Le principale inconvénient de cette méthode est d'être nouvelle présentant ainsi un manque terrible de chercheurs qui travail dessus et qui a comme conséquence l'inexploitation totale de ces capacités réelles.

L'une des différences majeures entre la méthode des équations intégrales et la méthode des éléments finis est que la forme de la matrice du système à résoudre est totalement aléatoire et diffère d'un cas à un autre au lieu d'être symétrique et bande (avec une bonne numérotation). Ce dernier point l'a défavorise face à la méthode des éléments finis, mais on peut surpasser ce petit inconvénient en utilisant une méthode de résolution adapté à ce type de système. Cette méthode existe, elle s'appelle GMRES et notre but est de l'implémenter et l'adapter pour le calcule des problèmes avec la méthode des équations intégrale.

Ainsi, dans le premier chapitre nous présentons la formulation de la méthode des équations intégrales en élasticité linéaire et sont traitement numérique.

Dans le deuxième Chapitre nous exposons le support logiciel utilisé dans cette étude, nous présentons le code KSP qui utilise la méthode des équations intégrales et qui est écrit en Visual C++ que nous exposons également, et nous rappelons le concept de base de la programmation orientée objet.

Le troisième chapitre est consacré à une présentation générale et minutieuse de la méthode GMRES, nous entourons la méthode théoriquement et nous exposons ces différentes variantes.

Le quatrième chapitre est consacré pour le détail de l'implémentation au niveaux algorithmique et gestion de mémoire ainsi que le réglage des différents paramètres et l'adaptation de sont implantation direct pour la résolution des problèmes 3D en utilisant le code KSP comme plateforme.

Il vient enfin dans le dernier chapitre une présentation générale du principe du préconditionnement ainsi que le choix d'un type adéquat de préconditionneur et la présentation de son efficacité et des améliorations que peut apporter son utilisation.

Enfin dans la conclusion nous présentons une petite comparaison entre la méthode GMRES et la méthode de décomposition de gauss précédemment utilisé par le KSP juste pour voir le niveau d'importance de l'utilisation de tel méthode dans le cadre de simulation numérique des problèmes de grande taille.

Chapitre 1

méthode des équations intégrales

1.1 Introduction

En physique, on s'intéresse souvent à ce qui se passe dans un domaine D, limité par une surface S s'appuyant sur un contour C. Si le problème est bidimensionnel, le domaine est une surface S limité par un contour C. En appliquant au problème considéré les principes généraux de la physique, on établit des équations, au mieux différentielles. Leur intégration comporte toujours des conditions aux limites.

Si on prend l'exemple du cas simple de la poutre homogène à section constante, il sera nécessaire de préciser les déplacements aux extrémités et les rotations (dérivées premières) en certains points.

L'équation ainsi établie sera dite équation de base. Elle concerne essentiellement le domaine. Sa solution est dite solution fondamentale.il y a dans [1] une table correspondant aux différentes équations en une, deux ou trois dimensions.

Les solutions analytiques sont très délicates à obtenir, aussi les ingénieurs ont-ils cherché des méthodes purement numériques : différences finies puis éléments finis. L'inconvénient est de devoir traiter, dans de nombreux cas, des problèmes à trois dimensions, ce qui exige souvent de grandes capacités de mémoire des ordinateurs. La méthode des équations intégrales de contour appelée également, méthode des équations intégrales aux frontières permet de revenir à deux dimensions.

Dans son principe, la méthode des équations intégrales consiste à transformer les équations locales qui décrivent le comportement des fonctions à l'intérieur du domaine (déplacements u dans le cas de l'élastostatique) et sur ses limites (déplacements u ou traction t imposée) en une équation intégrale liant les fonctions inconnues et certaines de leurs dérivées aux fonctions connues en surface.

Autrement dit, pour tout point x de la surface limitant la structure considérée, l'équation intégrale relie les déplacements et les tensions en tous les points de cette surface ; certains de ces termes sont connus par les conditions aux limites, les autres sont les inconnues du problème. Après résolution, on obtient les déplacements, les tensions et les contraintes en tous les points de la surface, une relation donne les mêmes informations en tous les points du domaine.

C'est au département de Génie civil de l'université de Southampton qu'une variante intéressante de cette méthode a été développée [2]. Ses auteurs lui ont donné le nom de *boundary element method* que l'on peut traduire par méthode d'élément frontière. Elle repose sur les méthodes dites d'annulation de l'erreur. Sous sa forme la plus simple elle part d'une solution satisfaisant les équations de base du domaine mais comportant quelques coefficients inconnus que l'on détermine en imposant les conditions aux limites en un certain nombre de points. On parle de méthode indirecte.

La méthode des équations intégrales sur le contour qu'on présente dans ce chapitre fait partie de la classe des méthodes dans lesquelles on recherche une solution présentant certaines propriétés sur le contour du domaine. La plupart de ces solutions en mécanique du solide sont recherchées à partir d'équations intégrales faisant intervenir des quantités connues et inconnues sur le contour. La méthode des équations intégrales sur le contour est fréquemment utilisée en mécanique de la rupture. Outre l'avantage de sa grande précision dans l'évaluation des grandeurs locales au voisinage du front de fissure, elle présente l'avantage de réduire le problème d'une dimension puisqu'elle nécessite la modélisation des données qu'en surface. En effet, le maillage de la structure ne concerne que son contour. Ainsi par rapport aux autres méthodes, comme celles des éléments finis, où l'intérieur du domaine est modélisé, cette approche permet de diminuer la taille des problèmes en nombre d'équations à résoudre. La solution à l'intérieur du domaine peut être obtenue ensuite par une simple relation. De plus la propagation de fissures se fait simplement par addition d'éléments ce qui nous évite le problème du remaillage local où globale.

1.2 Formulation des équations intégrales

1.2.1 Hypothèses

La méthode des équations intégrales est utilisable pour tout phénomène physique régi par des équations aux dérivées partielles linéaires à coefficients constants. Donc, dans le cadre

de la mécanique des solides, le matériau doit être élastique, linéaire et homogène. Dans cette étude, on traite les problèmes avec les hypothèses supplémentaires suivantes :

- le matériau est isotrope ;
- les forces de volumes sont négligeables ;
- le contour de la structure est borné.Ainsi, on peut traiter le cas d'un trou dans un plan infini et non dans un demi-plan.
- la formulation des équations intégrales est faite pour des problèmes en déformations planes.

Les problèmes en contraintes planes seront traités en remplaçant le module de Young E et le coefficient de poisson ν respectivement par $(1-\nu^2)E$ et $\nu/(1+\nu)$.

1.2.2 Equations de l'élasticité linéaire

On fait un petit rappel des équations régissant l'équilibre statique d'un solide linéairement élastique isotrope et homogène, sous l'hypothèse des petites perturbations.

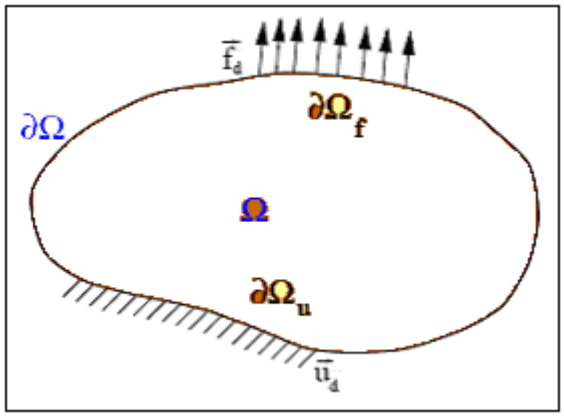


FIG1.1 Problème à résoudre

Pour un solide isotrope linéairement élastique et homogène occupant un domaine Ω et de frontière $\partial\Omega$. La normale unitaire est, par convention, systématiquement orientée vers l'extérieur du solide. Les champs de déplacement, de déformation et de contrainte en un point M de Ω, notés respectivement u(M), $\varepsilon(M)$, $\sigma(M)$, vérifient les relations locales suivantes :

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu\varepsilon_{ij}$$

$$\sigma_{ij,j} + b_i = 0$$

Où b désigne le vecteur des efforts volumiques, pouvant traduire des effets variés : pesanteur, inertie ou effets thermiques. La frontière $\partial\Omega$ supporte par ailleurs des conditions aux limites :

- en déplacements $u_j = \overline{u_j}$ sur $\partial\Omega_U$
- en tension : $t_i = o'_{ij} n_j = \overline{t_i}$ sur $\partial\Omega_F$

1.2.3 Solution élémentaire (Problème de Kelvin)

On appelle solution élémentaire où fondamentale de l'élastostatique un état associé à une force ponctuelle unitaire appliquée en un point fixé du domaine Ω et de direction e_l, le comportement du matériau étant considéré linéaire élastique. Il y a autant de solutions élémentaires que de choix de domaine Ω et de conditions aux limites sur $\partial\Omega$. Pour pouvoir écrire l'équation intégrale on a besoin d'une solution analytique d'un problème particulier sur le domaine Ω. Plusieurs solutions élémentaires existent,par exemple :

- demi-espace avec surface libre : solutions de Boussinesq, Cerruti, Mindlin(force ponctuelle appliquée en un point intérieur du demi-espace).
- demi-espaces collés : Solution de Mura (demi-espaces isotropes).
- plaque épaisse infinie élastique: Solution de Benitez, Rosakis.
- problème de GREEN ou de Neumann, etc…

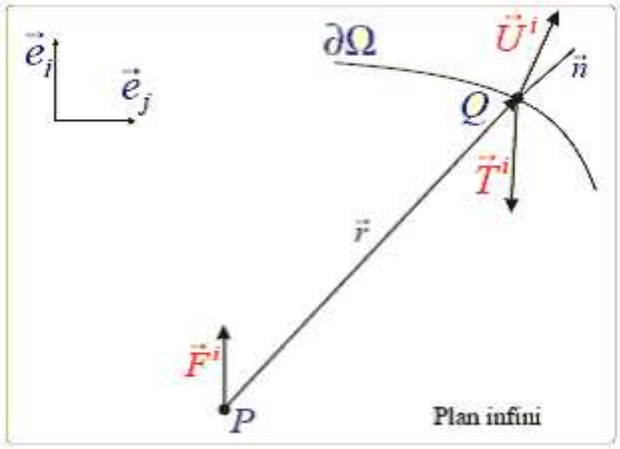


FIG.1.2 : Problème de KELVIN

La solution la plus simple et la plus générale correspond à l'espace infini élastique isotrope avec conditions de décroissance à l'infini (déplacement nuls à l'infini), dite « Solution de Kelvin » décrite ci-dessous.

Soit une force unitaire $\overrightarrow{F^i}$ appliquée en un point P d'un milieu infini bidimensionnel suivant la direction $\vec{e}_i$ .pour ce problème, l'équation de NAVIER s'écrit en un point Q du domaine :

$$(\lambda+\mu)u_{k,kj}+\mu u_{j,kk}=-\delta_{PQ}\delta ij \tag{1.1}$$

avec les déplacements nuls à l'infini comme conditions aux limites

$\lim_{\|PQ\|\to\infty} u_j = 0$

la résolution de cette équation différentielle conduit à :

- la composante du vecteur déplacement en un point Q suivant la direction $\overrightarrow{e_i}$ est :

$$U_j^i(P,Q)=\frac{1}{8\pi\mu(1-\nu)}\left[(3-4\nu)\ln\left(\frac{1}{r}\right)\delta_{ij}+r_{,i}r_{,j}\right] \tag{1.2}$$

où :

r :est la distance entre les point P et Q, $\mu=\dfrac{E}{2(1+\nu)}$

et

$$r_{,i}=\begin{cases}\dfrac{x_Q-x_P}{r} & si \quad i=1\\ \dfrac{y_Q-y_P}{r} & si \quad i=2\end{cases} \tag{1.3}$$

- le tenseur des contraintes est calculé à partir de la loi de comportement en élasticité linéaire (loi de Hooke) :

$$\sigma_{jk}^i=\lambda\delta_{jk}U_{l,l}^i+\mu(U_{j,k}^i+U_{k,j}^i) \tag{1.4}$$

- la composante du vecteur tension en un point Q sur une facette de normale $\overrightarrow{n(Q)}$ suivant la direction $\overrightarrow{e_j}$ est :

$$T_j^i(P,Q)=\sigma_{jk}^i(P,Q)n_k(Q) \tag{1.5}$$

soit sous une forme explicite :

$$T_j^i(P,Q)=\frac{1}{4\pi(1-\nu)r}\left[\frac{\partial r}{\partial n}\left[(1-2\nu)\delta_{ij}+2r_{,i}r_{,j}\right]+(1-2\nu)(n_jr_{,i}-n_ir_{,j})\right] \tag{1.6}$$

où :

$\dfrac{\partial r}{\partial n}=n_ir_i=n_1r_{,1}+n_2r_{,2}$

1.2.4 Théorème de réciprocité de Maxwell Betti

Le théorème de Maxwell Betti découle directement du théorème des travaux virtuels, il lie entre les solutions de deux problèmes d'élasticité linéaire sur un même domaine Ω

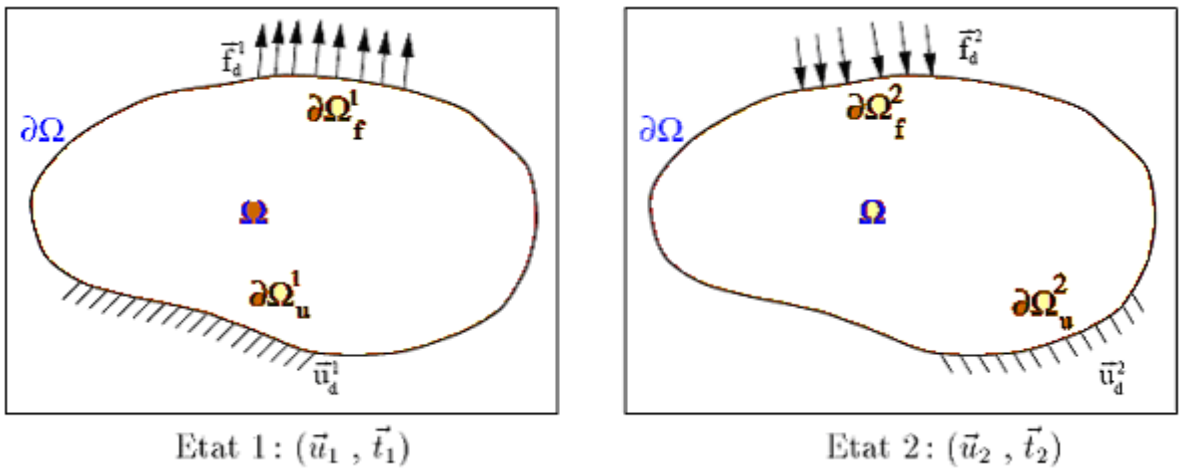


FIG.1.3 : *Deux problèmes différents sur un même domaine* Ω

Supposons que le premier problème ait comme solution le couple $(\vec{u^1},\vec{t^1})$; champ des déplacements et champ des tensions sur le contour $\partial\Omega$.

Soit un second problème défini sur le même domaine et qui a comme solution le couple $(\vec{u^2},\vec{t^2})$. D'après le théorème des travaux virtuels on peut écrire :

$$\int_\Omega [\sigma^1] : [\varepsilon^2] d\Omega - \int_{\partial\Omega} \vec{t^1}\vec{u^2} ds - \int_\Omega \vec{f_v^1}\vec{u^2} d\Omega = 0 \qquad (1.7)$$

et

$$\int_\Omega [\sigma^2] : [\varepsilon^1] d\Omega - \int_{\partial\Omega} \vec{t^2}\vec{u^1} ds - \int_\Omega \vec{f_v^2}\vec{u^1} d\Omega = 0 \qquad (1.8)$$

en faisant la différence des deux équations(1.7)et(1.8) on a :

$$\int_\Omega \left([\sigma^2] : [\varepsilon^1] - [\sigma^1] : [\varepsilon^2]\right) d\Omega - \int_{\partial\Omega} \left(\vec{t^2}\vec{u^1} - \vec{t^1}\vec{u^2}\right) ds - \int_\Omega \left(\vec{f_v^2}\vec{u^1} - \vec{f_v^1}\vec{u^2}\right) d\Omega = 0 \qquad (1.9)$$

or,d'après la symétrie du comportement élastique, on a :

$$[\sigma^2] : [\varepsilon^1] = [\sigma^2] : ([D][\sigma^1]) = ([D][\sigma^2]) : [\sigma^1] = [\sigma^1] : [\varepsilon^2] \qquad (1.10)$$

d'où le théorème de réciprocité :

$$\int_{\partial\Omega} \left(\vec{t^1}\vec{u^2} - \vec{t^2}\vec{u^1}\right) ds - \int_\Omega \left(\vec{f_v^1}\vec{u^2} - \vec{f_v^2}\vec{u^1}\right) d\Omega = 0 \qquad (1.11)$$

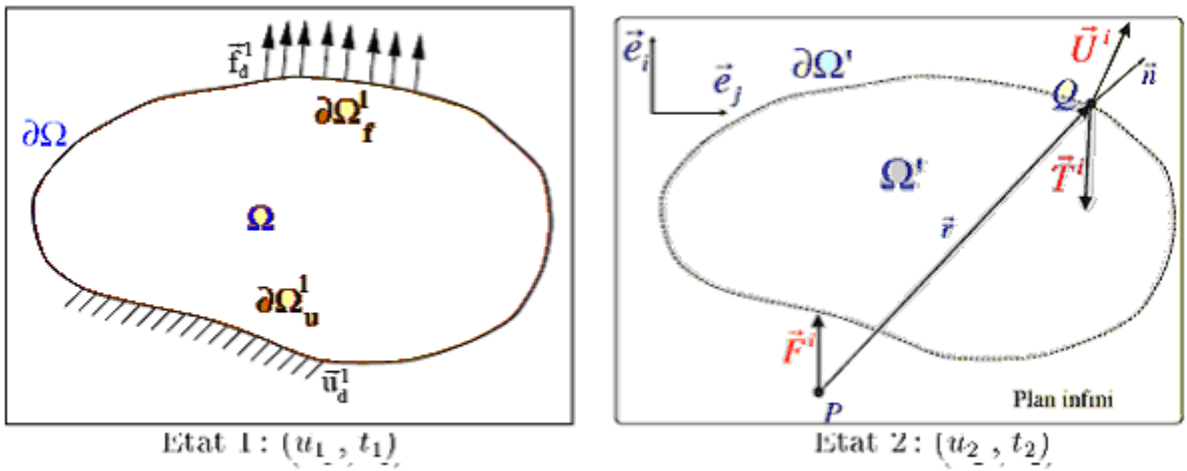
en l'absence des forces de volume, ce théorème s'écrit :

$$\int_{\partial\Omega} \vec{t^1}\vec{u^2} ds = \int_{\partial\Omega} \vec{t^2}\vec{u^1} ds \qquad (1.12)$$

ce théorème permet d'établir de relations intégrales entre les inconnues du problème à résoudre (1) ou (2),l'autre étant connu.

1.2.5 Equations intégrales en déplacements

L'équation intégrale en déplacements solution du problème déjà posé est obtenue à l'aide de la solution de Kelvin et du théorème de réciprocité de Maxwell-Betti.
Cette équation dépend de la position du point P, où on applique l'effort unitaire F_i.



a) Problème à résoudre : $(\vec{u},\vec{t})$ b)Problème de KELVIN : $(\vec{U^i},\vec{T^i})$

FIG. 1.4 : Domaine Ω et sa trace Ω' sur le plan infini

La figure (Fig.1.4b) représente le domaine Ω du problème à résoudre :

- $u_j(Q)$ est la composante du vecteur déplacement d'un point Q suivant la direction $\vec{e_j}$;
- $t_j(Q)$ est la composante du vecteur tension suivant la direction $\vec{e_j}$ en un point Q sur une facette de normale $\vec{n}(Q)$.

Pour un problème bien posé, soit $u_j(Q)$, soit $t_j(Q)$, Q variant sur le contour $\partial\Omega$ et (j=1,2), est donné comme condition aux limites, l'autre étant l'inconnue qu'on cherche. La figure (Fig.1.4b) représente un plan infini, Ω' étant la trace de Ω sur ce plan. $\vec{F_i}$ est une force unitaire appliquée en un point P suivant la direction $\vec{e_j}$, la composante du vecteur déplacement en un point Q de $\partial\Omega'$ suivant la direction $\vec{e_j}$ est donnée par la solution

analytique du problème de KELVIN $(U_j^i(P,Q))$, et la composante du vecteur tension en un point Q de $\partial\Omega'$ sur la droite tangente est $T_j^i(P,Q)$.

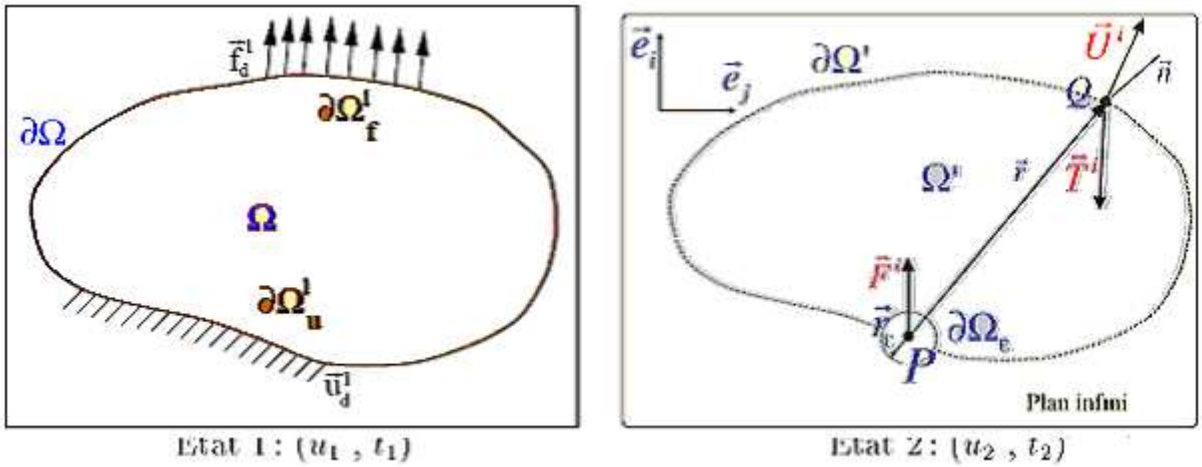
- Si le point P est à l'extérieur de Ω' :

Dans ce cas, Ω' est en équilibre sous l'action du champ des tensions $T_j^i(P,Q)$. Ainsi on a deux états où le domaine Ω est en équilibre élastostatique : le premier est le problème à résoudre $(u_j(Q), t_j(Q))$, le second est $(U_j^i(P,Q), T_j^i(P,Q))$. Alors d'après le théorème de Maxwell-Betti, on a :

$$\int_{\partial\Omega} T_j^i(P,Q)u_j(Q)ds(Q) = \int_{\partial\Omega} U_j^i(P,Q)t_j(Q)ds(Q) \qquad (1.13)$$

- Si le point P est sur Ω' :

Le domaine Ω' n'est plus en équilibre sous l'action du champ $T_j^i(P,Q)$. Par conséquent, l'équation (1.13) ne peut plus être écrite pour un point P appartenant à $\partial\Omega$.



Problème à résoudre : $(\vec{u},\vec{t})$ Problème de KELVIN : $(\overrightarrow{U^i},\overrightarrow{T^i})$

FIG.1.5 :Cas où le point d'application de la force est sur le contour $\partial\Omega$

Pour pouvoir écrire le théorème de MAXWELL-BETTI pour un point P du contour, on isole P par un cercle de rayon ε comme le montre la figure (Fig.1.5). Le nouveau contour d'intégration sera :

$$\Gamma = \partial\Omega - (\Omega_\varepsilon \cap \partial\Omega) + (\partial\Omega_\varepsilon \cap \Omega) \qquad (1.14)$$

où Ω_ε est la région du cercle, $\partial\Omega_\varepsilon$ son contour. On peut écrire alors :

$$\int_\Gamma T_j^i(P,Q)u_j(Q)ds(Q) = \int_\Gamma U_j^i(P,Q)t_j(Q)d(Q) \qquad (1.15)$$

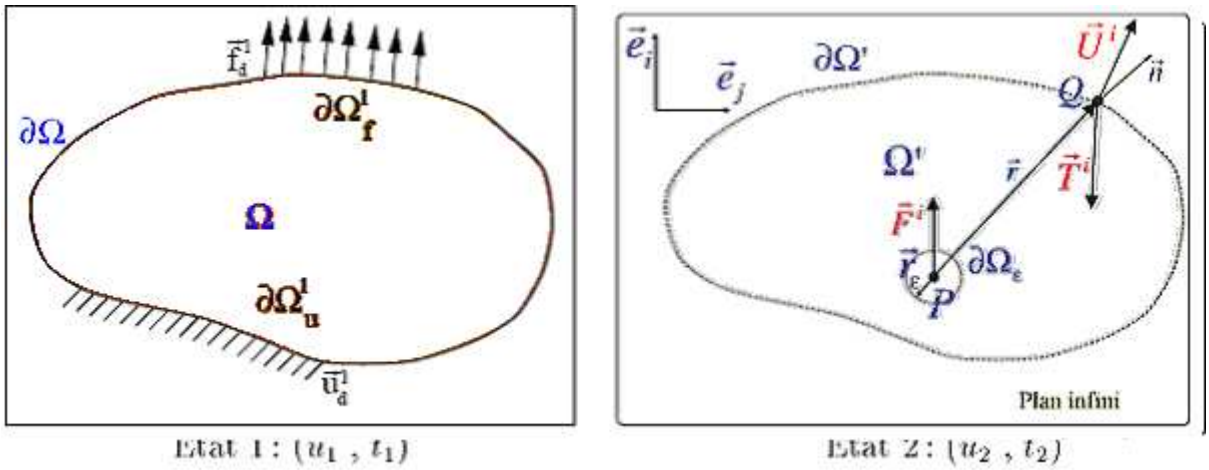
et en faisant tendre ε vers zéro, on obtient l'équation intégrale en déplacements :

$$C_{ij}u_i(P)+\int_{\partial\Omega}T_j^i(P,Q)u_j(Q)ds(Q)=\int_{\partial\Omega}U_j^i(P,Q)t_j(Q)ds(Q) \qquad (1.16)$$

où $C_{ij}=\frac{1}{2}\delta_{ij}$ si la normale en P est continue. Dans ce cas l'équation intégrale en déplacement s'écrit :

$$\frac{1}{2}u_i(P)+\int_{\partial\Omega}T_j^i(P,Q)u_j(Q)ds(Q)=\int_{\partial\Omega}U_j^i(P,Q)t_j(Q)ds(Q) \qquad (1.17)$$

1.2.6 Déplacements des points intérieurs



Problème à résoudre : $(\vec{u},\vec{t})$ Problème de KELVIN : $(\overrightarrow{U^i},\overrightarrow{T^i})$

FIG.1.6 : Cas où le point d'application de la force est un point intérieur

- Si le point P est à l'intérieur de Ω' :

Pour pouvoir écrire le théorème de Maxwell-Betti, on isole P par un cercle de rayon ε comme l'indique la figure (Fig.1.6). Le nouveau contour d'intégration devient :

$$\Gamma=\partial\Omega+\partial\Omega_\varepsilon$$

On a alors :

$$\int_\Gamma T_j^i(P,Q)u_j(Q)ds(Q)=\int_\Gamma U_j^i(P,Q)t_j(Q)ds(Q) \qquad (1.18)$$

en faisant tendre ε vers zéro, on obtient :

$$u_i(P)+\int_{\partial\Omega}T_j^i(P,Q)u_j(Q)ds(Q)=\int_{\partial\Omega}U_j^i(P,Q)t_j(Q)ds(Q) \qquad (1.19)$$

ou

$$u_i(P) = \int_{\partial\Omega} U^i_j(P,Q) t_j(Q) ds(Q) - \int_{\partial\Omega} T^i_j(P,Q) u_j(Q) ds(Q) \qquad (1.20)$$

Cette équation est connue sous le nom de « Identité de Somigliana » pour les déplacements. Elle permet de donner la valeur du déplacement en tous points intérieurs en termes de déplacement et d'efforts (u_i et t_i) et pour tous points P où l'effort unitaire est appliqué, une fois que u_i et t_i sont connus en tous points de la frontière.

1.2.7 Contraintes des points intérieurs

La composante σ_{ij} du tenseur des contraintes est calculée à partir de la loi de comportement de Hooke (1.4) :

$$\sigma_{ij}(P) = \int_{\partial\Omega} D^i_{jk}(P,Q) t_k(Q) ds(Q) - \int_{\partial\Omega} S^i_{jk}(P,Q) u_k(Q) ds(Q) \qquad (1.21)$$

où

$$D^i_{jk} = \frac{1}{4\pi(1-\nu)r}\left[(1-2\nu)(\delta_{ik} r_{,j} + \delta_{jk} r_{,i} - \delta_{ik} r_{,j}) + 2 r_{,i} r_{,j} r_{,k}\right] \qquad (1.22)$$

et

$$S^i_{jk} = \frac{2\mu}{4\pi(1-\nu)r^2}\left\{2\frac{\partial r}{\partial n}\left[(1-2\nu)\delta_{ij} r_{,k} + \nu(\delta_{ik} r_{,j} + \delta_{jk} r_{,i}) - 4 r_{,i} r_{,j} r_{,k}\right] + \nu(n_i r_{,j} r_{,k} + n_{,j} r_{,i} r_{,k}) + (1-2\nu)(2 n_k r_{,i} r_{,j} + n_j \delta_{ik} + n_i \delta_{jk}) - (1-4\nu) n_k \delta_{ij}\right\} \qquad (1.23)$$

1.2.8 Equations intégrales en tension

Le tenseur des contraintes σ_{ij} est obtenu en appliquant la loi de Hooke aux équations de déplacement. L'application de la loi de Hooke nous permet de relier la composante σ_{ij} du tenseur des contraintes à la solution de Kelvin et aux tensions et déplacements du contour.

L'équation (1.21) donne le tenseur des contraintes pour un point P à l'intérieur du domaine $(r \neq 0)$. En prenant P sur le contour et en suivant le même raisonnement que pour l'équation (1.17), on obtient l'équation intégrale en contraintes :

$$\frac{1}{2}\sigma_{ij}(P) + \int_{\partial\Omega} S^i_{jk}(P,Q) u_k(Q) ds(Q) = \int_{\partial\Omega} D^i_{jk}(P,Q) t_k(Q) ds(Q) \qquad (1.24)$$

avec les conditions : $u_k(P) \in C^1$ et $t_k(P) \in C^1$

En multipliant l'équation (1.24) par $n_j(P)$, on obtient l'équation intégrale en tensions :

$$\frac{1}{2}t_i(P)+n_j(P)\int_{\partial\Omega}S^i_{jk}(P,Q)u_k(Q)ds(Q)=n_j(P)\int_{\partial\Omega}D^i_{jk}(P,Q)t_k(Q)ds(Q) \quad (1.25)$$

1.2.9 Contraintes des points du contour

Pour calculer le tenseur des contraintes pour un point P situé sur le contour, trois méthodes sont envisageables :

- connaissant le champ des déplacements sur le contour, on peut déterminer celui des contraintes variant linéairement sur un élément. Cette méthode nous donne des résultats peu précis.
- En calculant les contraintes pour un point intérieur assez proche de P puis en faisant une extrapolation. Cela ne donne pas de bons résultats pour les zones de fortes variations de contraintes.
- La troisième méthode,la plus précise, est d'utiliser la formule (1.24) comme une formule des contraintes sur le contour. Ainsi on peut écrire :

$$\sigma_{ij}(P)=2\int_{\partial\Omega}D^i_{jk}(P,Q)t_k(Q)ds(Q)-2\int_{\partial\Omega}S^i_{jk}(P,Q)u_k(Q)ds(Q) \quad (1.26)$$

1.3 Traitement numérique des équations intégrales

Les solutions analytiques des équations intégrales relatives aux problèmes d'élasticité sont extrêmement rares et n'existent que pour quelques géométries et conditions aux limites simples.

Pour être en mesure de traiter des cas plus complexes qui correspondent à ceux rencontrés dans la pratique, on est amené à la résolution numérique de ces problèmes. On approche alors la solution du problème avec une erreur qui dépend généralement de :

- la représentation de la géométrie ;
- le nombre d'éléments utilisés ;
- la représentation des champs élastiques ;
- la méthode d'intégration.

Cette partie est consacrée à la présentation des principales étapes du traitement numérique de l'équation intégrale en déplacements.

1.3.1 Discrétisation de la géométrie

La frontière $\partial\Omega$ est discrétisée en Ne segments de droite Γ_m (m=1,Ne), comme en éléments finis chaque élément est défini par deux nœuds N_m^1 et N_m^2. Connaissant les coordonnées de ces nœuds, on détermine par interpolation les coordonnées d'un point quelconque Q de l'élément à l'aide des fonctions de forme linéaire :

$$x_Q(s) = x_{N_m^1} N_1(s) + x_{N_m^2} N_2(s) \tag{1.27}$$

$$y_Q(s) = y_{N_m^1} N_1(s) + y_{N_m^2} N_2(s) \tag{1.28}$$

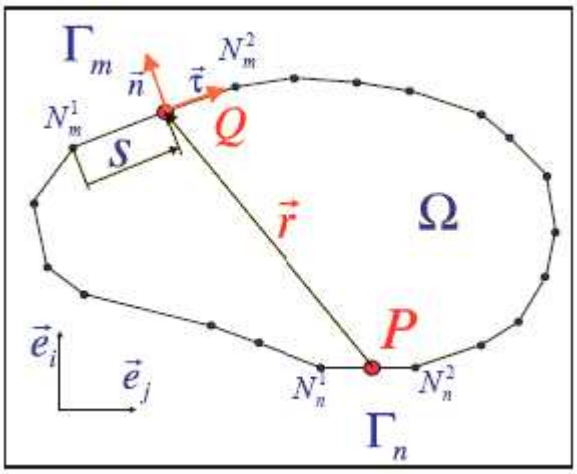


FIG.1.7 : Discrétisation de la géométrie

avec

$$N_1(s) = 1 - \frac{s}{L_m} \tag{1.29}$$

$$N_2(s) = \frac{s}{L_m} \tag{1.30}$$

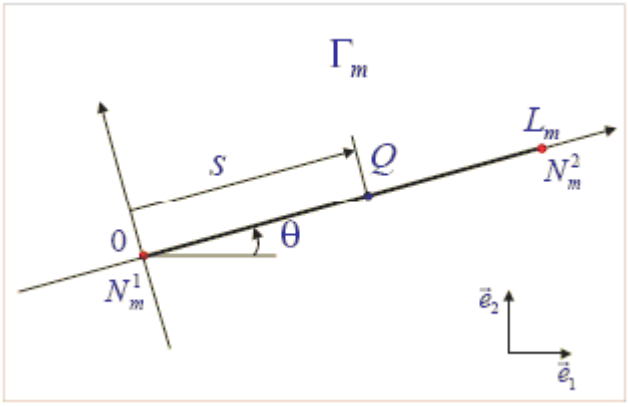


FIG.1.8 : Elément linéaire pour la représentation de la géométrie

où L_m est la longueur de l'élément Γ_m .

Avec ce type d'élément on peut évaluer analytiquement les intégrales qui apparaissent dans la formulation des équations intégrales. Un autre choix de type d'élément, quadratique par exemple, nous obligera à les évaluer numériquement par la méthode de quadrature de Gauss, ce qui très coûteux en temps de calcul.

1.3.2 Discrétisation des champs élastiques

La formulation des équations intégrales en déplacements n'impose aucune condition sur les champs des déplacements et des tensions, par conséquent, on peut prendre une variation constante sur chaque élément

$$u_j \in C^{-1} \qquad t_j \in C^{-1}$$

Pour d'autre cas on doit prendre une variation au moins quadratique des champs de déplacements et de tensions, soient :

$$u_j \in C^{1} \qquad t_j \in C^{1}$$

Les points de collocation ont été choisis à l'intérieur de l'élément de discrétisation (Fig.1.9) pour éviter les problèmes dus à la discontinuité de la normale (le terme libre $C_{ij}(P,Q)$ de l'équation intégrale change en fonction de la discontinuité de la normale), ainsi que pour pouvoir traiter les problèmes de singularité en fond de fissure.
Sur un élément Γ_n on a :

$$u_i(s) = \sum_{l=1}^{3} u_i(M_n^l) N_l(s) \tag{1.31}$$

et

$$t_i(s) = \sum_{l=1}^{3} t_i(M_n^l) N_l(s) \tag{1.32}$$

où M_n^l (l=1,3) sont les points de collocation de l'élément Γ_n, et avec :

$$N_1(s) = \frac{15}{8} - \frac{6}{L_n} s + \frac{9}{2L_n^2} s^2 \tag{1.33}$$

$$N_2(s) = -\frac{5}{4} + \frac{9}{L_n} s + \frac{9}{L_n^2} s^2 \tag{1.34}$$

$$N_3(s) = \frac{3}{8} - \frac{3}{L_n} s + \frac{9}{2L_n^2} s^2 \tag{1.35}$$

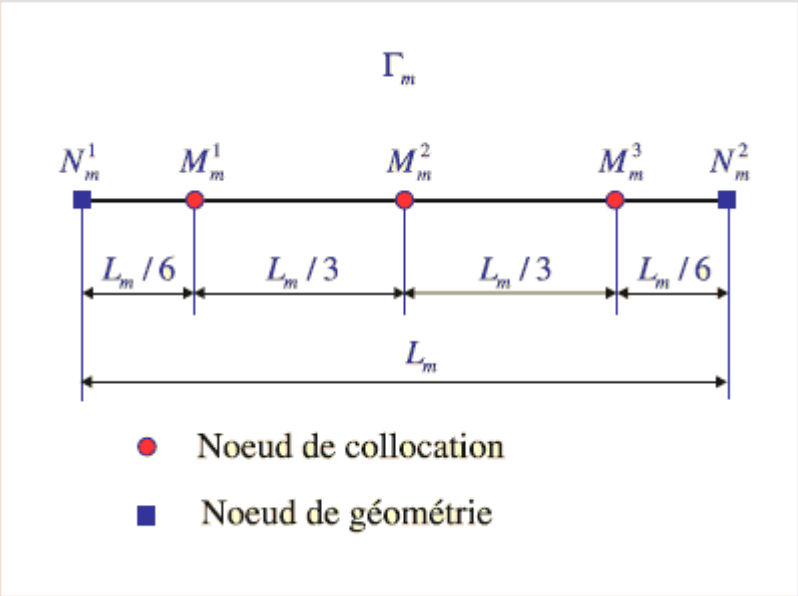


FIG.1.9 : Elément quadratique non conforme

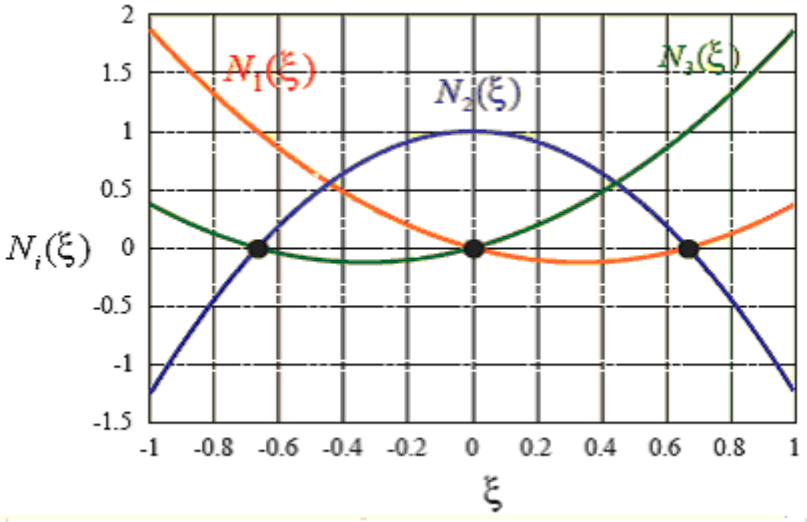


FIG.1.10 : Fonctions de Formes

1.3.3 Discrétisation de l'équation intégrale

L'équation intégrale en déplacements (1.17) appliquée à un point de collocation $(P = M_n^k)$ de l'élément Γ_n s'écrit :

$$\frac{1}{2}u_i(M_n^k)+\int_{\partial\Omega}T_j^i(M_n^k,Q)u_j(Q)ds(Q)=\int_{\partial\Omega}U_j^i(M_n^k,Q)t_j(Q)ds(Q) \quad (1.36)$$

En discrétisant le contour $\partial\Omega$ en Ne éléments $\Gamma_m(m=1,NEG)$, on aura :

$$\frac{1}{2}u_i(M_n^k)+\sum_{m=1}^{Ne}\int_{\Gamma_m}T_j^i(M_n^k,Q)u_j(Q)ds(Q)= \sum_{m=1}^{Ne}\int_{\Gamma_m}U_j^i(M_n^k,Q)t_j(Q)ds(Q) \quad (1.37)$$

et en remplaçant la valeur des variables par leurs représentations, on obtient :

$$\frac{1}{2}u_i(M_n^k)+\sum_{m=1}^{Ne}\int_{\Gamma_m}T_j^i(M_n^k,Q)\left(\sum_{l=1}^{3}N_l(s)u_j(M_m^l)\right)ds(Q)= \sum_{m=1}^{Ne}\int_{\Gamma_m}U_j^i(M_n^k,Q)\left(\sum_{l=1}^{3}N_l(s)t_j(M_m^l)\right)ds(Q) \quad (1.38)$$

soit :

$$\frac{1}{2}u_i(M_n^k)+\sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\left(\int_0^{Lm}T_j^i(M_n^k,s)N_l(s)ds\right)u_j(M_m^l)= \sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\left(\int_0^{Lm}U_j^i(M_n^k,s)N_l(s)ds\right)t_j(M_m^l) \quad (1.39)$$

Finalement, on a :

$$\sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\left(\frac{1}{2}\delta_{mn}\delta_{kl}\delta_{ij}+\int_0^{Lm}T_j^i(M_n^k,s)N_l(s)ds\right)u_j(M_m^l)= \sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\left(\int_0^{Lm}U_j^i(M_n^k,s)N_l(s)ds\right)t_j(M_m^l) \quad (1.40)$$

C'est une équation à $(12\times Ne)$ variables. En faisant varier le point de collocation M_n^k sur tous les éléments (n=1,Ne), sur chaque point de collocation k(k=1,3) et suivant chaque direction $\vec{e_i}(i=1,2)$, on obtient ainsi un système de $(6\times Ne)$ équations à $(12\times Ne)$ variables de la forme :

$$[A]\{u\}=[B]\{t\} \quad (1.41)$$

1.3.4 Calcul des intégrales

C'est l'étape la plus importante de la méthode des équations intégrales. En effet il faut déterminer avec précision les coefficients des matrices [A] et [B] pour obtenir une bonne approximation de la solution du problème. Le nombre de coefficients à déterminer est égal à $2\times(6\times Ne\times 6\times Ne)$, soit $72\times(Ne)^2$ coefficients, ce qui prend une grande partie du temps de calcul. La solution idéale est de les évaluer analytiquement. C'est ce que nous avons finalement choisi. Pour cela il est nécessaire de discrétiser la géométrie en segment de droite pour pouvoir effectuer une intégration analytique.

1.3.5 Constitution du système d'équations

La construction du système d'équations consiste à écrire l'équation intégrale en déplacement discrétisée (1.39) pour les points de collocation M_m^k(k=1,N_{pc}) de chaque élément du contour Γ_m(m=1,Ne) suivant les deux directions, N_{pc} étant le nombre de nœuds de collocation par élément. Ainsi, on obtient un système d'équations algébriques de $(6\times Ne)$ équations à $(12\times Ne)$ variables :

Après discrétisation de l'équation intégrale en déplacements on est arrivé à un système

$$[A]\{u\}=[B]\{t\}$$

Les coefficient des matrices [A] et [B] ne sont pas du même ordre de grandeur. En effet, A_{ij} est du même ordre de grandeur que r tandis que B_{ij} est du même ordre de grandeur que $\frac{r}{2G}$ (G étant le module de cisaillement). Pour obtenir un système bien conditionné, on doit multiplier les coefficients de la matrice [B] par 2G. On aura alors à résoudre :

$$[A]\{u\}=(2G)[B]\left\{t'=\frac{t}{2G}\right\} \qquad (1.42)$$

Les vecteurs {u} et {t} contiennent les valeurs du déplacement et des tentions avant l'application des conditions aux limites. $(6\times NEG)$ de ces variables sont données comme conditions aux limites. Ces conditions peuvent être introduites par un réarrangement dans les colonnes de [A] et [B] ce qui nous permet de transformer le système à résoudre en mettant les variables connues d'un coté et les inconnues de l'autre coté et obtenir un système d'équations de $(6\times Ne)$ équations à $(6\times Ne)$ inconnues de la forme :

$$[K]\{x\} = \{y\} \tag{1.43}$$

où {x} est le vecteur qui contient toutes les inconnues en déplacement et en tension.

Après la résolution par une méthode approprié, les valeurs des tentions trouvées seront multipliées par 2G pour avoir la solution du problème.

1.3.6 Résolution du système d'équations

La matrice [K] du système à résoudre est une matrice pleine, non symétrique et à diagonale dominante, ce qui représente une différence considérable par rapport aux méthodes des éléments finis, pour lesquelles il est bien connu que la matrice de rigidité est symétrique et bande. Pour les matrices de taille moyenne, les méthodes directes sont très efficaces. Par contre, pour les matrices de grande taille, et du fait que la matrice [K] est à diagonale dominante les méthodes itératives sont les plus efficaces. Cela représente le cœur de notre travail qui consiste à implémenter une des plus célèbres méthodes itératives adaptée à ce type de système d'équation. La représentation de cette méthode ainsi que son implémentation sont représenté dans les chapitres 3 et 4.

1.3.7 Calcul des déplacements des points intérieurs

Le champ des déplacements des points intérieurs P se calcule par une simple intégration, de l'équation (1.20), on a :

$$u_i(P) = \int_{\partial\Omega} U_j^i(P,Q)t_j(Q)ds(Q) - \int_{\partial\Omega} T_j^i(P,Q)u_j(Q)ds(Q) \tag{1.44}$$

En discrétisant cette équation, on obtient :

$$u_i(P)=\sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\int_0^{L_m}\left(U_j^i(P,s)N_l(s)ds\right)t_j(M_m^l)- \sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{j=1}^{2}\int_0^{L_m}\left(T_j^i(P,s)N_l(s)ds\right)u_j(M_m^l) \tag{1.45}$$

1.3.8 Calcul des contraintes des points intérieurs

Le tenseur des contraintes d'un point intérieur P se calcule par une simple sommation, de l'équation (1.21) on a :

$$\sigma_{ij}(P)=\int_{\partial\Omega}D_{jk}^i(P,Q)t_k(Q)ds(Q)-\int_{\partial\Omega}S_{jk}^i(P,Q)u_k(Q)ds(Q) \tag{1.46}$$

On peut écrire alors après discrétisation :

$$\sigma_{ij}(P)=\sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{k=1}^{2}\left(\int_0^{L_m}D_{jk}^i(P,s)N_l(s)ds\right)t_k(M_m^l)- \sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{k=1}^{2}\left(\int_0^{L_m}S_{jk}^i(P,s)N_l(s)ds\right)u_k(M_m^l) \tag{1.47}$$

1.3.9 Calcul des contraintes des points du contour

Le tenseur des contraintes d'un point P du contour se calcule par simple sommation à partir de l'équation (1.26) :

$$\sigma_{ij}(P)=2\left\{\int_{\partial\Omega}D_{jk}^i(P,Q)t_k(Q)ds(Q)-\int_{\partial\Omega}S_{jk}^i(P,Q)u_k(Q)ds(Q)\right\} \tag{1.48}$$

Soit :

$$\sigma_{ij}(P)=2\left(\sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{k=1}^{2}\left(\int_0^{L_m}D_{jk}^i(P,s)N_l(s)ds\right)t_k(M_m^l)- \sum_{m=1}^{Ne}\sum_{l=1}^{3}\sum_{k=1}^{2}\left(\int_0^{L_m}S_{jk}^i(P,s)N_l(s)ds\right)u_k(Q_m^l)\right) \tag{1.49}$$

1.4 Conclusion

Dans ce chapitre on a présenté le principe de la méthode des équations intégrales directes en élasticité linéaire. A partir du théorème de réciprocité de Maxwell_Betti et de la solution élémentaire du problème de Kelvin, on a obtenu les équations intégrales en déplacements. La discrétisation de la géométrie a été effectuée par des segments de droite alors que les champs élastiques (champ des déplacements et champ des tentions) ont été discrétisés par des éléments quadratiques non conformes.

Chapitre 2

La POO et le code KSP

2.1 Introduction

Pour résoudre des problèmes modernes de modélisation, de dimensionnement où bien de calcule en mécanique et dans n'importe quel autre domaine de la technologie il ne suffie pas de développé des méthodes numériques. Il faut en plus trouver des algorithmes adéquats et développer des codes de simulation numérique pour rendre possible le traitement de ces problèmes où le rendre le moins compliquer que possible.

Le KSP est un nouveau née dans la famille de ces codes qui à beaucoup de perspectives, il utilise la méthode des équations intégrales déjà présentée dans le premier chapitre et est écrit en Visual C++, le langage orienté objet par excellence qui offre des solutions graphiques très à la hauteur des exigences du marché. Cela représente notre environnement du travail exposé dans ce chapitre.

2.2 La programmation orienté objet (POO):

« Au cours des 35 dernières années, les concepteurs de matériel informatique sont passés de machines de la taille d'un hangar à des ordinateurs portables légers basés sur de minuscules microprocesseurs.

Au cours des mêmes années, les développeurs de logiciels sont passés de l'écriture de programmes en assembleur et en COBOL à l'écriture de programmes encore plus grands en C et C++. On pourra parler de progrès (bien que cela soit discutable), mais il est clair que le monde du logiciel ne progresse pas aussi vite que celui du matériel. Qu'ont donc les développeurs de matériel que les développeurs de logiciels n'ont pas ?

La réponse est donnée par les composants. Si les ingénieurs en matériel électronique devaient partir d'un tas de sable à chaque fois qu'ils conçoivent un nouveau dispositif, si leur première étape devait toujours consister à extraire le silicium pour fabriquer des circuits

intégrés, ils ne progresseraient pas bien vite. Or, un concepteur de matériel construit toujours un système à partir de composants préparés, chacun chargé d'une fonction particulière et fournissant un ensemble de services à travers des interfaces définies. La tâche des concepteurs de matériel est considérablement simplifiée par le travail de leurs prédécesseurs.

La réutilisation est aussi une voie vers la création de meilleurs logiciels. Aujourd'hui encore, les développeurs de logiciels en sont toujours à partir d'une certaine forme de sable et à suivre les mêmes étapes que les centaines de programmeurs qui les ont précédés. Le résultat est souvent excellent, mais il pourrait être amélioré. La création de nouvelles applications à partir de composants existants, déjà testés, a toutes chances de produire un code plus fiable. De plus, elle peut se révéler nettement plus rapide et plus économique, ce qui n'est pas moins important. »

Extrait de « Au coeur de ActiveX et OLE », de *David Chappel*

2.2.1 Petite histoire de l'évolution des méthodes de programmation

Au début, la programmation se résumait à du code machine entré en langage binaire (c'était l'époque des switches et des programmes sur cartes perforées). Puis on est passé à l'assembleur : c'est toujours du code machine, on remplace juste les 0 et les 1 par des jeux d'instructions plus intuitifs, et on compile le tout.

Puis vinrent les langages plus évolués, comme le COBOL, ou le FORTRAN, qui permettaient de coder des choses beaucoup plus complexes, mais qui avaient le défaut d'obliger les développeurs à continuer à réaliser des programmes avec des branchements conditionnels (GOTO). La création de programmes complexes était difficile, et la maintenance du code pouvait s'avérer très délicate

Puis vinrent les L4G (comme le C, ou le Pascal) qui permettaient une programmation structurée, avec des boucles, des bibliothèque de fonction et des fonctions utilisateurs, et qui ont permis pour la première fois de faire de gros programmes « maintenables »…

Sauf que la taille des projets informatiques ne cesse de croître, et que les langages structurés ont vite révélé quelques manques pour répondre à certaines logiques de programmation.

Chronologie de l'objet

1967 : le langage Simula introduit la notion d'objet

(1967) C'est un langage orienté vers la simulation

1973 : introduction de la notion de classe dans Simula en 1973.

1972 SmallTalk, premier langage tout objet – introduit la notion d'envoi de message

(communication entre les objets)

1976 : apparition de la notion d'héritage dans SmallTalk

1980 : (C avec Classes)

1982 : naissance du C++

1983 : apparition de l'encapsulation dans le langage ADA

1986 : le C++ arrive à maturation

1995 : JAVA

1997 : standardisation ANSI / ISO du langage c++

Le problème qui se pose quand on a besoin d'élaborer des programmes complexes, c'est que l'on est très vite confronté aux problèmes suivants :

- comment réutiliser les programmes que vous avez écrits il y'a six mois et dont vous avez oublié le fonctionnement
- comment comprendre et réutiliser les programmes faits par d'autres
- comment « cloner » rapidement des programmes déjà fait pour des applications légèrement différentes
- comment programmer simplement des actions simples sur des éléments variés et complexes
- comment ajouter des fonctions sans tout réécrire et tout retester

L'approche traditionnelle du développement, dite fonctionnelle, marche bien tant que le développeur sait où il va : le client a fourni un cahier des charges parfait et complet, ne changera pas d'avis, et une fois que le logiciel sera fini, on y touchera plus…

Dans ce cas, on peut décomposer les programmes en « fonctions » : un processus global, et des sous processus ou fonctions qui créent un arbre décomposant le problème en une série d'actions (c'est une décomposition « algorithmique »)

Sauf que ces programmes idéaux n'existent pas : le client change d'avis, les logiciels vivent, et le développeur doit vraiment savoir où il va, surtout en cas de travail en équipe, car il ne s'agit pas de « découvrir » des choses en cours de route

Or avec une approche fonctionnelle, tout changement de spécification à des conséquences catastrophiques… Il faut parfois tout refaire du sol au plafond

Pour résoudre ces problèmes, il faut développer trois stratégies :

- **modéliser** différemment
- **modulariser**
- **encapsuler**

2.2.2 Le "modèle objet

A quoi sert un modèle en programmation

L'intérêt du "modèle" en programmation, c'est de permettre de réaliser une "maquette" simplifiée du programme à réaliser. Un modèle est une "abstraction", une simplification de la réalité.

Un bon modèle doit réunir deux qualités :

- faciliter la **compréhension** du programme étudié, en réduisant la complexité
- permettre de **simuler** le comportement du programme

L'intérêt du "modèle objet

La force de la programmation objet, c'est qu'elle s'appuie sur un modèle calqué sur la réalité physique du monde. Les objets se comportent comme des entités indépendantes, autosuffisantes qui collaborent par échange de message.

On peut aussi raisonner du particulier au général (généraliser) grâce à la notion de classes d'objets, qui permet de partager entre les objets de la même classe des propriétés et des comportements…

Un objet, dans ce modèle, est défini comme un "truc" (une "entité") qui a des propriétés, et un comportement. Les caractéristiques fondamentales d'un objet sont :

- une identité (ne pas confondre "voiture" et la voiture RENAUT CLIO V6)
- un comportement (freinage)
- un état (elle est rouge, le réservoir est plein)

le triomphe de l'UML

La modélisation objet consiste à créer une représentation informatique des éléments du monde réel auxquels on s'intéresse, sans se préoccuper de l'implémentation, ce qui signifie indépendamment d'un langage de programmation. Il s'agit donc de déterminer les objets présents et d'isoler leurs données et les fonctions qui les utilisent. Pour cela des méthodes ont été mises au point. Entre 1970 et 1990, de nombreux analystes ont mis au point des approches orientées objets, si bien qu'en 1994 il existait plus de 50 méthodes objet. Toutefois seules 3 méthodes ont véritablement émergé:

- La méthode **OMT** de *Rumbaugh*
- La méthode **BOOCH'93** de *Booch*
- La méthode **OOSE** de *Jacobson*

A partir de 1994, Rumbaugh et Booch (rejoints en 1995 par Jacobson) ont uni leurs efforts pour mettre au point le langage de description UML (Unified Modeling Language), qui permet de définir un langage standard en incorporant les avantages des différentes méthodes précédentes (ainsi que celles d'autres analystes). Il permet notamment de "programmer" entièrement une application avec un langage qui modélise toutes les composantes du futur programme.

UML standardise :

- les modèles
- les notations
- les diagrammes. Par contre, UML ne standardise pas les méthodes de développement.

2.2.3 Modulariser

Il est possible de découper un programme en modules autonomes grâce à une programmation de type "procédurale". On découpe le problème en éléments simples, on confie le développement de chacun des morceaux à des développeurs différents, avec obligation de respecter des "sous spécifications" particulières.

Sauf que cela oblige les équipes de développement à créer leurs propres normes en matière de méthodes de développement, et l'expérience prouve qu'en l'absence d'une très grande rigueur, rien ne garantit que le produit fini fonctionnera de matière nominale.

La programmation objet apporte tous les outils pour avoir une approche "modulaire" de la programmation : l'objet est un "module" parfaitement opérationnel, et on peut construire des programmes complexes en jouant au Lego avec les objets :

- on peut réutiliser des bibliothèques de classes d'objets prêts à l'emploi
- on peut "personnaliser" des objets existants, sans toucher au code testé et revérifié du code de départ
- on peut faire agir des objets sur des objets, ou interagir des objets entre eux, sans compromettre
- et modifier un objet (bien fait) n'oblige pas à modifier les autres objets…

La POO apporte aussi des "outils" simplifiant le travail du développeur, comme une notation particulière, et des "règles" qui produisent de la modularité "par construction".

2.2.4 L'encapsulation

La complexité dans les programmes est génératrice de bogues... Et la probabilité de l'existence de bogues inattendues croit de manière exponentielle avec le nombre de fonctions et de procédures qui interagissent entre elles... Pour limiter ces phénomènes, il est donc impératif de pouvoir écrire des modules qui fonctionnent comme des boîtes noires hermétiques : ces boîtes attendent un certain type de données, de commandes, de message. Elles réagissent de manière prévisible, et renvoient un certain type de données, de messages.

Par contre : en dehors de ces "interactions" avec le monde extérieur, tout le reste est "encapsulé" dans la boîte, il est donc impossible que ce qui se passe dans la boîte influe sur une autre boîte, ou l'inverse…

L'encapsulation permet donc de "cloisonner" les données et les fonctions à l'intérieur de l'objet…

Bien sûr, là encore, on peut reproduire ce fonctionnement en "boîte noire" en procédural, mais cela oblige à réinventer un comportement qui existe en "natif" dans les objets…

2.2.5 L'intérêt de la POO par rapport au procédural et au fonctionnel

Le code est plus sûr

Les programmes sont plus clairs

La maintenance des applications est facilitée

Le code est facilement réutilisable

Il est facile de créer de nouveaux programmes légèrement différents par clonage d'un programme existant

Il est facile de faire évoluer des programmes

L'approche RAD est possible

La POO rend possible le développement de gros programmes

2.2.6 Les inconvénients

La POO oblige à réfléchir et à modéliser avant de programmer (est-ce réellement un inconvénient)

2.3 Le Microsoft Visual C++

Microsoft Visual C++ offre des outils et un environnement de développement très puissant et très flexible pour créer et développer différentes applications. Il est constitué des composants suivants :

- Les outils du compilateur de Visual C++. Le compilateur possède de nouvelles fonctionnalités destinées à assister les développeurs dans les différentes étapes de leurs travail, et surtout pour ceux qui ciblent des plates-formes de machine virtuelle, comme le CLR (Common Language Runtime).

- Les bibliothèques Visual C++. Il s'agit de la bibliothèque ATL (Active Template Library), de la bibliothèque MFC (Microsoft Fondation Class) et des bibliothèques standard, telles que la bibliothèque C++ standard et la bibliothèque Runtime C (CRT). Une nouvelle bibliothèque, la bibliothèque de prise en charge C++, a été conçue pour simplifier les programmes qui visent le CLR.

- L'environnement de développement Visual C++. Bien que les outils de compilateur et les bibliothèques C++ puissent être utilisés depuis la ligne de commande, l'environnement de développement aide puissamment à la gestion et la configuration des projets, ainsi que pour modifier ou parcourir le code source, et offre des outils de débogage. Cet environnement prend également en charge IntelliSense, qui effectue des suggestions contextuelles intelligentes au cours de la création du code.

Outre des applications classiques d'interface graphique utilisateur, Visual C++ permet aux développeurs de générer des applications Web, des applications Windows clientes intelligentes, ainsi que des solutions pour client basique et pour périphériques mobiles clients intelligents. C++. Visual C++ appartient à la suite de logiciel Visual Studio.

Mais avant tout le Visual C++ est du C++ qui est un langage de programmation dit "haut niveau", c’est le langage de niveau système le plus répandu au monde créé en 1983 par Bjarne Stroustrup des laboratoires Bell. Il se distingue de son prédécesseur (le langage C) en proposant une programmation orientée objet. Ce langage est particulièrement utilisé dans les applications demandant de hautes performances.

2.4 Le KSP

Le KSP est un code de calcul des problème de la mécanique des solides développé à l'UTC sur la plateforme Microsoft Visual C++ et basé sur la méthode des équations intégrales. Il offre la possibilité de modélisation des problèmes d'élasticités 2D et 3D, des problèmes d'élastoplasticité et de fissuration.

Comme tout logiciel de calcul en mécanique, le KSP est constitué essentiellement de trois parties : l'introduction des données, le traitement et la sortie des résultats.

2.4.1 L'introduction des données

L'introduction des données est la première étape dans le déroulement d'une manipulation sur KSP. Elle se décompose elle même à deux étapes : l'introduction des données géométriques et l'introduction des conditions aux limites.

L'introduction des données géométriques :

Elles sont introduites dans le cas de problèmes 2D par un sketch de dessin offrant la possibilité de faire les différentes opérations principales d'un dessin 2D. Le maillage est réalisé par un mailleur propre à KSP et qui donne un maillage bien adapté pour l'utilisation par la méthode des équations intégrales.

En ce qui concerne les problèmes 3D les données géométriques sont introduites à l'aide d'une fonction qui permet leur importation depuis d'autre logiciel telle que ABAQUS où IDEAS.

L'introduction des conditions aux limites

Le KSP offre un menu de fonction pour l'introduction des différentes conditions aux limites (déplacements imposés ou charges imposés) directement sur son interface d'utilisation pour les problèmes 2D et 3D.

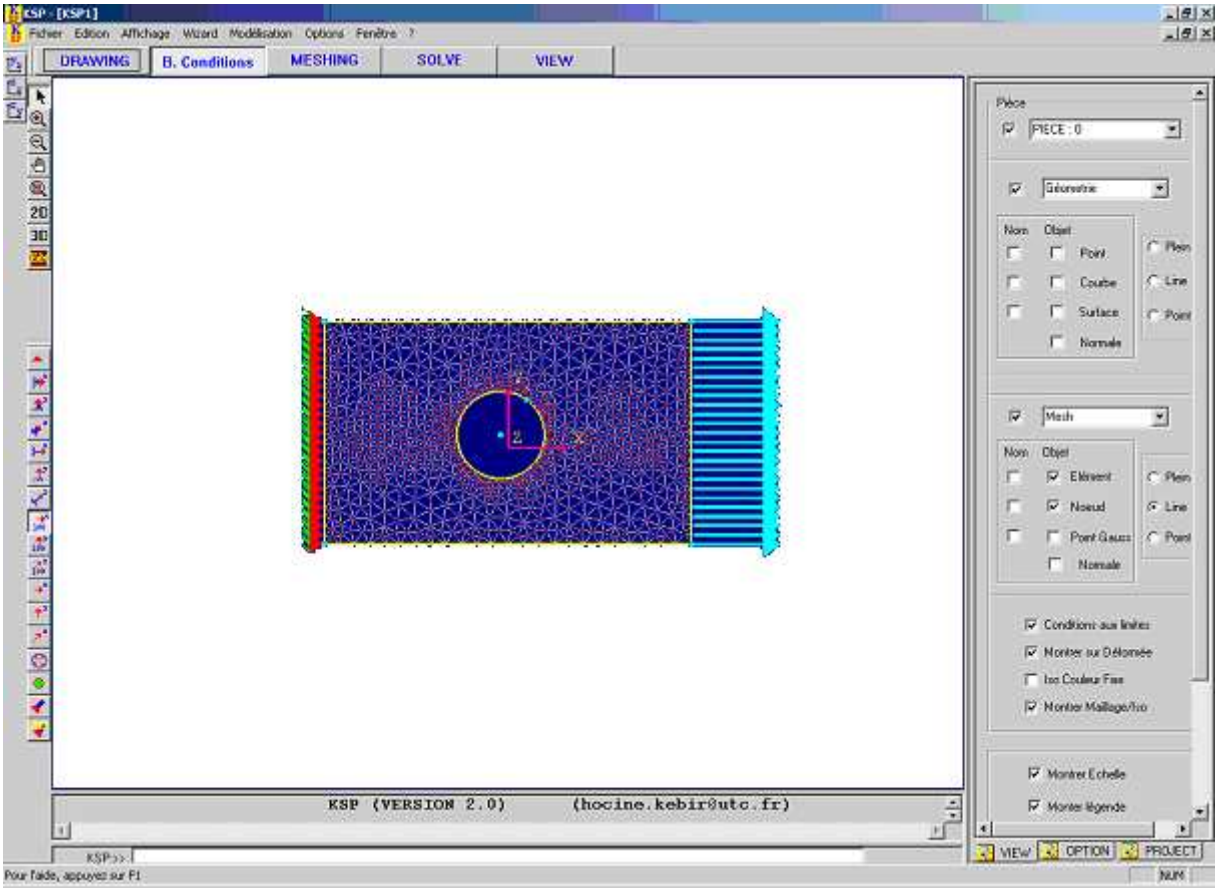


FIG.2.1. exemple d'introduction des donnés sur le KSP

2.4.2 Le traitement

C'est l'étape la plus intéressante, elle aussi est constitué de deux étape : la création du système et la résolution.

La création du système

Elle est effectuée par la méthode des équations intégrales expliquée dans le premier chapitre.

La résolution

C'est la partie où intervient notre travail. Le KSP utilisait la méthode de décomposition de gauss pour la résolution du système d'équations résultant de l'algorithme de la méthode des équations intégrales. Pour les raisons expliquées dans le paragraphe 1.3.6 du chapitre 1 il était préférable qu'on change cette méthode directe par une autre méthode itérative qui sera bien adapté pour les problèmes de grande taille.

On a choisi l'algorithme GMRES qu'on va implémenter dans le KSP pour qu'il devient plus rapide et offre des possibilités de traitement de problèmes beaucoup plus larges qu'avant.

2.4.3 La sortie des résultats

Elle est assuré dans le KSP par une interface graphique qui permet de voire la distribution des contraintes clairement à l'aide d'un jeux de couleur qui offre une très bonne visibilité aux résultats. Le KSP donne aussi la possibilité de voir les déformés ainsi que des animations de déformation à l'échelle normale et exagéré. Beaucoup d'autres résultats peuvent être tiré du KSP tel que les contraintes maximales et minimale ainsi que des graphiques pour une lecture et une interprétation plus fiable.

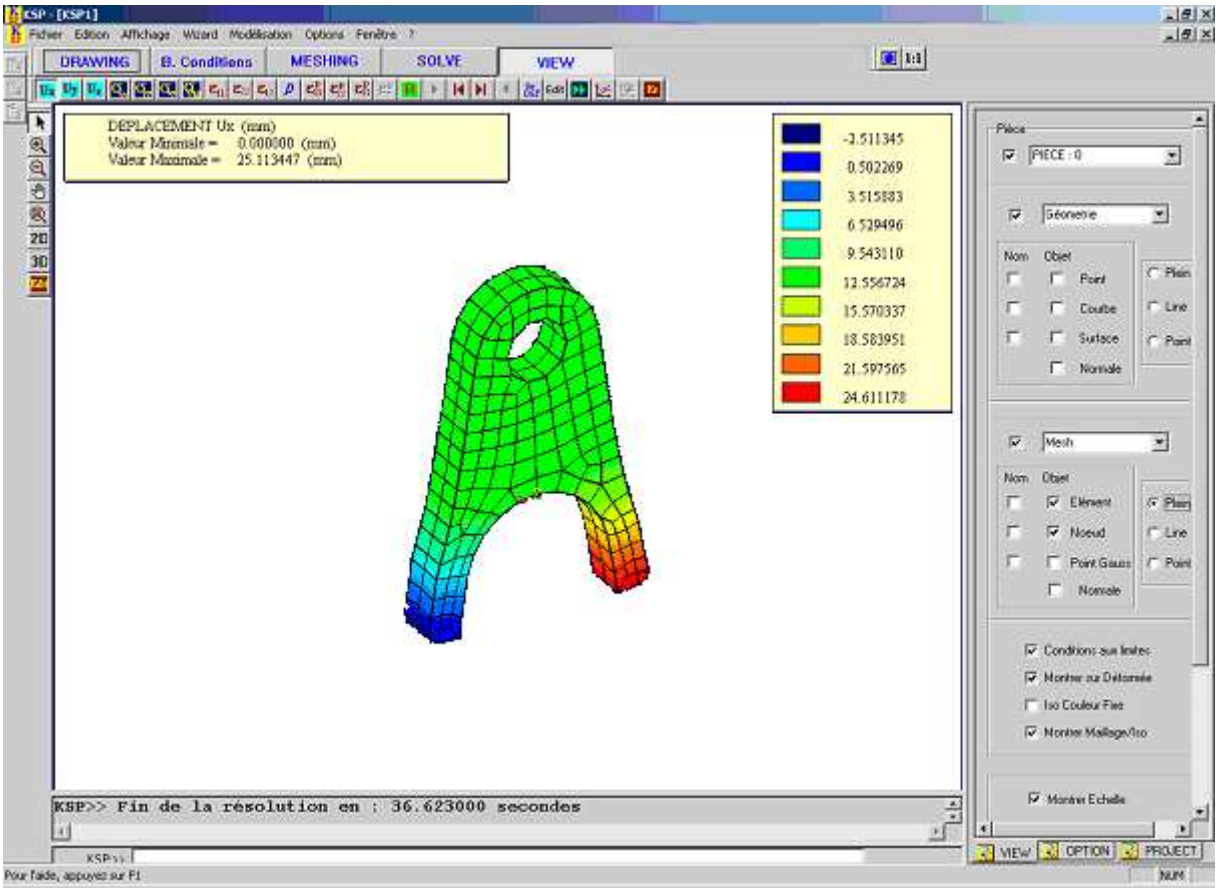


FIG.2.2. exemple de sortie de résultats sur le KSP

Chapitre 3

La méthode GMRES

3.1 Introduction

Pour la résolution d'un système $Ax = b$ où $A \in R^{n \times n}$ est une matrice carrée d'ordre $n \times n$ et x et b sont des vecteurs de longueur n, et quand la taille du système est très grande ($n > 10^3$), plusieurs auteurs ont proposé des généralisations différentes de la méthode du gradient conjugué et du gradient biconjugué ces 25 dernière années.

Paige et Saunders ont proposé la méthode MINRES caractérisé par la minimisation de la norme du résidu via un sous espace de Krylov. MINRES représente une généralisation de la méthode du gradient conjugué pour la résolution des systèmes linéaire symétriques indéfinis. MINRES comporte deux étapes distinctes. La première est de construire une base orthonormale avec le processus de Lanczos et la deuxième est de résoudre le problème au moindres carré en utilisant les rotations de Givens introduite par Gentleman.

Pour généraliser MINRES, en 1986, Saad et Schultz ont formulé la très populaire méthode GMRES (Generalised Minimized RESidual) pour la résolution des problèmes non symétrique. Ils présentent une implémentation pratique basée sur le processus d'Arnoldi et les rotations de Givens. Elle construit une base orthonormale du sous espace de Krylov $K_k(r_0) = span\{r_0, Ar_0, ..., A^{k-1}r_0\}$ où $r_0 = b - Ax_0$ et l'initie avec le vecteur $v_1 = r_0 / \| r_0 \|$.

Une autre implémentation très intéressante de GMRES consiste à utiliser les transformations de Housholder a été proposé en 1988. L'implémentation (appelée Simpler GMRES) proposée en 1994 démarre le processus d'Arnoldi par $v_1 = Ar_0 / \| Ar_0 \|$. Cette implémentation construit une base orthonormale du sous espace $K_k(Ar_0)$ au lieu de $K_k(r_0)$ et peut être considéré comme une implémentation simplifiante qui à un avantage de mémoire mais qui présente un problème d'arrêt d'algorithme.

Le coût de l'orthogonalisation complète du sous espace de Krylov $K_k(r_0)$ devient très pénalisant quand $K_k(r_0)$ atteint une certaine taille. Pour détourner ce problème on redémarre l'algorithme après un kmax d'itération qui représente la taille maximale du sous espace de Krylov à mémoriser. Une autre alternance est d'utiliser une orthogonalisation incomplète. Les propriétés de la convergence pour l'orthogonalisation incomplète ne sont pas bien maîtrisées. Pour la version redémarrée de GMRES, et au moment de redémarrage, on perd quelques informations. En 1995 Morgan accroît l'efficacité du GMRES redémarré en réduisant l'effet négatif du redémarrage par l'augmentation de la base du nouveau sous espace de Krylov par les vecteurs propres liés aux valeurs propres des plus petites magnitudes. Plusieurs autres possibilités de préconditionnement ont été proposées. Les plus importante sont FGMRES proposé par Saad en 1993 qui change de préconditionneur à chaque étape et sa plus grande rivale GMRESR qui été développé en1994.

D'autres tentatives moins importantes pour accélérer l'algorithme ou pour minimiser le coût de mémorisation n'ont pas cessé d'être proposées ces 10 dernières années. Ceci dit le développement de cette méthode est toujours d'actualité, et le meilleur exemple est la nouvelle variante très intéressante proposé par Ayachour tout récemment (en 2003) et qui consiste à utiliser un nouveau algorithme pour la résolution du problème au moindre carrée au lieu des rotation de Givens.

Ce chapitre est consacré à la présentation générale de la méthode de GMRES, de ses principes de bases et des différentes étapes de son algorithme. On commence par la description de l'environnement de la méthode et les différents processus qui la constituent. On trouve dans la section 3.2 la description des méthodes de Krylov et dans les sections qui suivent 3.3, 3.4 et 3.5 la représentation de la décomposition de Hessenberg le processus d'Arnoldi et les rotations de Givens. On continue, dans les sections 3.6 et 3.7, par la présentation des principes de la méthode et de ses algorithmes de base. Et on termine par son organigramme dans la section 3.8.

3.2 Les méthodes de Krylov:

3.2.1 Définition de l'espace de Krylov

L'espace de Krylov de dimension k lié à une matrice A et un vecteur v est le sous espace engendré par $\{ v, A v, \ldots, A^{k-1} v \}$, on le note K_k.

$$K_k(A,v) = \text{span}\{ v, Av, \ldots, A^{k-1}v \},$$

3.2.2 La propriété de minimisation :

Contrairement aux méthodes itératifs stationnaires les méthodes de Krylov n'utilisent pas une matrice d'itération, et puisque la matrice $A \in R^{n \times n}$ n'est pas hermitienne définie positive, des méthodes relativement simples ne peuvent pas être appliquée car on ne dispose pas de la norme $\|.\|_A$. L'idée va être de construire des méthodes itératives consistant à minimiser la norme $\|.\|_2$ du résidu. A la $k^{ièm}$ itération, une minimisation de l'erreur est effectuée envers l'espace voisin. La solution sera dans

$$x_0 + K_k ,$$

où x_0 est la solution initiale et K_k est le $k^{ième}$ sous espace de Krylov et le résidu est

$$r = b - Ax, \tag{3.1}$$

alors $\{r_k\}_{k \geq 0}$ représente le résidu séquentiel

$$r_k = b - Ax_k , \tag{3.2}$$

La $m^{ième}$ itération des algorithmes des méthodes de Krylov donne la solution du problème aux moindre carrée

$$\min_{x \in x_0 + K_m} \| b - Ax \| \tag{3.3}$$

3.3 Décomposition de Hessenberg en arithmétique exacte

3.3.1 Définition d'une matrice de Hessenberg supérieure

La matrice $H \in R^{n \times n}$ est sous forme Hessenberg supérieure si et seulement si :

$$\boldsymbol{h}_{i,j} = 0 \text{ pour } i > j + 1$$

Cette matrice est dite irréductible si et seulement si :

$$\boldsymbol{h}_{i+1,i} \neq 0 \text{ pour } i=1,\ldots,n-1$$

On appelle matrice non dérogatoire une matrice dont l'indice de toute valeur propre est égal à sa multiplicité algébrique. On peut affirmer en se référant à [8], qu'une matrice de Hessenberg irréductible est non dérogatoire.

Exemple

La matrice suivante appelé matrice compagne est une matrice de Hessenberg non dérogatoire

$$c=\begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{pmatrix}.$$

3.3.2 Présentation des décompositions de Hessenberg

Une décomposition de Hessenberg d'une matrice $A \in R^{n \times n}$ consiste à trouver deux matrices, la première Q unitaire et la deuxième H de forme Hessenberg telles que :

$$A=QHQ^H \tag{3.4}$$

Les matrices A et H sont unitairement équivalentes. Et il n'y a pas unicité de la décomposition de Hessenberg [11].

Il existe plusieurs familles de méthodes de décomposition de Hessenberg. Nous nous intéressons principalement à la méthode itérative d'Arnoldi pour obtenir une décomposition de Hessenberg. Elle effectue une projection orthogonale de A sur le sous-espace de Krylov engendré par $\{v_1, Av_1, ..., A^{i-1}v_1\}$. A l'étape i, on obtient la matrice Vi que l'on note $Vi= \{v_1, \ldots, vi\}$. La matrice $V_i^H AV_i=H_i$ est une matrice de Hessenberg d'ordre i. La matrice $H_{i+1}= V_{i+1}^H AV_{i+1}$ d'ordre i+1 s'obtient en rajoutant une colonne puis une ligne à H_i :

$$H_{i+1}=\left[\begin{array}{ccc|c} & & & \times \\ & H_i & & \cdot \\ & & & \cdot \\ & & & \times \\ \hline 0 & \ldots & 0 & \times \end{array}\right].$$

Il existe plusieurs manières de calculer la base Vi dans la méthode d'Arnoldi en utilisant essentiellement la factorisation QR à l'aide d'une des trois implantations : Gram Schmidt classique, Gram Schmidt modifiée ou Householder.

3.4 Processus d'Arnoldi

3.4.1 Principe de l'algorithme d'Arnoldi : effectuer une factorisation QR

L'algorithme d'Arnoldi effectue une factorisation QR récursive de la matrice $[v_1, Av_1, ..., Av_m] = [v_1, Av_m]$ pour $m = 1, ..., n-1$:

$$[\mathrm{v}_1, \mathrm{AV}_\mathrm{m}] = \mathrm{V}_{\mathrm{m+1}}[1, \overline{H_m}] = \mathrm{V}_{\mathrm{m+1}}\mathrm{R}_{\mathrm{m+1}} \ .$$

avec R_{m+1} la matrice triangulaire supérieure d'ordre m+1.

Ainsi, tant que $h_{m+1,m} \neq 0$, pour $m = 1, ..., n-1$.

$$\mathrm{AV}_\mathrm{m} = \mathrm{V}_{\mathrm{m+1}}\overline{H_m} \tag{3.5}$$

Pour m=n, se situe un arrêt mathématique qui correspondrait à $h_{n+1,n}=0$ et

$$AV_n = V_n H_n \ \Leftrightarrow \ A = V_n H_n V_n^H \tag{3.6}$$

3.4.2 Trois façons classiques de réaliser la factorisation QR

Il existe plusieurs manières de calculer la base Vi dans la méthode d'Arnoldi en utilisant essentiellement la factorisation QR à l'aide d'une des trois implantations classiques: Gram Schmidt classique, Gram Schmidt modifiée et Householder.

Pour une même matrice A et un même vecteur v_1, les algorithmes de Gram Schmidt classique, Gram Schmidt modifié et Householder permettent de calculer la base orthonormale {v1,…,vk} du sous espace de Krylov K_k et la matrice H_k à chaque itération k et donnent les mêmes résultats en arithmétique exacte avant l'arrêt. Nous pouvons donc choisir l'un ou l'autre de ces trois algorithmes pour effectuer une décomposition de Hessenberg par la méthode d'Arnoldi. Une distinction entre ces trois algorithmes existe cependant lorsqu'on les utilise en précision finie lors de calculs sur ordinateur.

Algorithme 3.1 : ***algorithme d'Arnoldi-Gram Schmidt classique*** **(CGS)**

- Choisir un vecteur v_1 de norme 1.
  - Pour k=1,…,n
  - Calculer $h_{i,k}=(Av_{k,}v_i)$ pour i=1,…,k
  - Calculer $w_k=Av_k-\sum_{i=1}^{k} h_{ik} v_i$
  - $h_{k+1,k}=\|w_k\|$. Si $h_{k+1,k}=0$, Arrêt.
  - $v_{k+1}=w_k/h_{k+1,k}$
- Fin Pour

Algorithme 3.2 : ***algorithme d'Arnoldi-Gram Schmidt modifié*** **(MGS)**

La variante MGS qui est présentée comme une modification de l'algorithme de Gram Schmidt classique a en fait été utilisée directement par le Marquis Pierre Simon de Laplace dès le début du 19ème siècle, lors de ses calculs astronomiques de moindres carrés.

- Choisir un vecteur v_1 de norme 1.
- Pour k=1,…,n
  - $w_k^{(1)}=Av_k$
  - Pour *i*=1,…,k
    - calculer $h_{ik}=(w_k^{(i)}, v_i)$
    - $w_k^{(i+1)}=w_k^{(i)}-h_{ik} v_i$
  - fin pour
  - $h_{k+1,k}=\|w_k\|$. Si $h_{k+1,k}=0$, Arrêt.
  - $v_{k+1}=w_k/h_{k+1,k}$
- Fin Pour

Algorithme 3.3 : ***algorithme d'Arnoldi-Householder***

Pour cet algorithme, le calcul de $V_m=[v_1,…,v_m]$ utilise des matrices P_i, i=1,…,n, hermitiennes unitaires (donc carrées n×n) construites à partir de matrices de Householder.

1. Choisir un vecteur v_1 de norme 1 et calculer P_1 tel que $P_1v_1=e_1$.
2.
- Pour m=2,…,n

- $z_m=P_{m-1}\ldots P_1 Av_{m-1}$
- Partitionner z_m en $z_m=[\, z_m^{(1)\,T} \;\; z_m^{(2)\,T}]^T$ avec $z_m^{(1)}$ de taille m-1.
- Si $z_m^{(2)} = 0$,
    - $P_m=I_n$
    - $h_{m-1}=z_m$
    - Arrêt des itérations ou continuation par $P_m=I_n$ et $v_m=P_1\ldots P_{m-1}e_m$.
- Sinon
    - Calculer Pm à partir de $z_m^{(2)}$.
    - $h_{m-1}= P_m z_m$
    - $v_m=P_1\ldots P_m e_m$

- fin poir

3. $h_n=P_n...P_1 Av_n$

Il existe également d'autres techniques d'orthogonalisation, telles que la méthode de Givens, mais elles se basent sur des techniques semblables à celles employées par ces trois algorithmes. D'autres techniques plus récentes ont été proposées ces dernières années mais on ne dispose pas d'informations suffisantes pour leurs implémentation et pas assez de recule pour voir leurs efficacité.

3.4.3 Méthode d'Arnoldi pour la décomposition incomplète de Hessenberg

Cette méthode associe à une matrice $A \in R^{n\times n}$

- une base orthonormale $V_k = \{v_1,\ldots, v_k\}$ du sous-espace de Krylov

$$K_k(A,v_1)= \text{span}\{\, v_1, Av_1, \ldots, A^{i-1}v_1\},$$

et

- une matrice $\overline{H_k}$ de taille (k+1,k) qui a pour bloc principal une matrice de Hessenberg supérieure H_k, et qui est augmentée d'une ligne supplémentaire dont le seul élément non nul est $h_{k+1,k}$.

La matrice $\overline{H_k}$ a donc la forme suivante :

$$\overline{H}_k = \begin{pmatrix} h_{11} & \cdots & \cdots & \cdots & \cdots & h_{1k} \\ h_{12} & \ddots & & & & \vdots \\ 0 & h_{23} & & & & \vdots \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & & h_{kk-1} & h_{kk} \\ 0 & \cdots & & \cdots & 0 & h_{k+1k} \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} & & \\ & H_k & \\ & & \end{matrix}} & & \\ & \ddots & \\ & & \ddots \\ & & & \ddots & \\ & & & & h_{k+1k} \end{pmatrix}$$

On peut montrer que, à l'itération k de l'algorithme d'Arnoldi, les matrices A, V_k et H_k vérifient l'égalité :

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T \tag{3.7}$$

où le vecteur e_k est le $k^{ième}$ vecteur colonne de la matrice identité d'ordre n.
Cette égalité peut également s'écrire :

$$A V_k = V_{k+1} \overline{H_m} \ . \tag{3.8}$$

En multipliant l'égalité (3.7) à gauche par V_k^H, nous pouvons écrire

$$V_k^H A V_k = H_k . \tag{3.9}$$

H_k est la matrice quotient de Rayleigh de A par V_k. C'est une autre manière de voir que H_k est la matrice de l'application A projetée orthogonalement sur K_k dans la base V_k.

Un grand intérêt de cette méthode pour les très grandes matrices ($n > 10^3$) est de pouvoir stopper le processus au pas $m << n$ et de travailler avec H (respectivement $\overline{H}$) de taille $m \times m$ (respectivement $(m+1) \times m$). Cela s'appelle une décomposition incomplète de Hessenberg, avec H la matrice qui représente la projection orthogonale de A sur le sous-espace de Krylov de dimension m dans la base V_m.

L'algorithme d'Arnoldi s'arrête lorsque $h_{k+1,k}=0$. La matrice V_k est alors une base d'un sous espace invariant associé à A. Cela se produit pour $k_0 \leq n$ et on a :

$$AV_{k0} = V_{k0}H_{k0} \tag{3.10}$$

Pour $k \geq k0$, on a $K_k(A,v_1)= K_{k0}(A,v_1)$. Plus précisément :

pour $k < k_0$, $rg(K_k)=k$,

pour $k \geq k_0$, $rg(K_k)=k_0$

où $rg(K_k)$ est le rang de K_k.

Nous qualifions le phénomène $h_{k+1,k}=0$ à l'itération $k=k_0$ par le terme d'arrêt heureux[12], qui est traduit du terme anglais : ‘’happy break-down’’. Une justification de cette appellation sera expliquée dans le paragraphe suivant.

3.4.4 L'arrêt heureux

Proposition (établie par Y.Saad) [13]:
La procédure d'Arnoldi s'arrête en fournissant la solution x_k du problème. L'algorithme GMRES s'arrête à l'étape k lorsque $h_{k+1,k}=0$ *, si et seulement si x_k est solution exacte du système* **Ax = b** .

L'algorithme d'Arnoldi comporte deux étapes essentielles :

1. Définir $r_0 = b – Ax_0$ et $v_1 = r_0/\|r_0\|$.
2. pour i=1,….,k-1

$$v_{i+1} = \frac{Av_i - \sum_{j=1}^{i}((Av_i)^T v_j)v_j}{\| Av_i - \sum_{j=1}^{i}((Av_i)^T v_j)v_j \|}$$

Tant qu'il n'y a pas de division par zéro dans la deuxième étape de l'algorithme d'Arnoldi, alors les colonnes de la matrice V_k constituent une base orthonormale de K_k.
Une division par zéro représente un arrêt de l'algorithme (break-down) et se manifeste seulement si la solution exacte $x_k \in x_0 + K_{k-1}$.

Ces deux faits formulés de deux manières différentes représentent le même phénomène et justifient clairement le terme d'arrêt heureux.

3.4.5 La perte d'orthogonalité

Nous savons que le terme $h_{k+1,k}$ est le critère d'arrêt de la décomposition de Hessenberg par la méthode d'Arnoldi. Lorsque ce terme est nul à l'itération $k=k_0$, algorithmiquement, le vecteur v_{k0+1} ne peut pas être construit et l'algorithme d'Arnoldi exact s'arrête s'il en est ainsi.

Supposons que $h_{k+1,k}= \varepsilon$ avec ε petit. Le vecteur Av_k peut être calculé algorithmiquement mais sera presque linéairement dépendant des vecteurs v_i, i=1,…,k, précédents. Les vecteurs v_i, i=1,…,k, forment une base unitaire mais les vecteurs calculés pour i=1,…,k+1, ne forment plus une base unitaire : il y a perte d'orthogonalité de la matrice V_{k+1}. Cette perte d'orthogonalité est le phénomène flagrant qui se produit lorsque l'on effectue une décomposition de Hessenberg par la méthode d'Arnoldi en précision finie, la précision finie occulte l'arrêt heureux : la valeur de h_{k_0+1,k_0} calculée est non nulle et égale à un ε petit. La perte d'orthogonalité de la matrice V_{k_0+1} se manifeste à l'itération qui succède l'arrêt heureux de l'algorithme d'Arnoldi en arithmétique exacte.

3.5 Les rotations de Givens

Quand k est très grand (k étant la dimension de sous espace de Krylov K_k), l'implémentation en utilisant les rotations de Givens est plus efficace et plus rapide qu'une approche directe.
Une rotation de 2×2 est une matrice de la forme :

$$G=\begin{pmatrix} c & -s \\ s & c \end{pmatrix}, \tag{3.11}$$

où $c=\cos(\theta), s=\sin(\theta)$ pour $\theta\in[-\pi,\pi]$.La matrice orthogonale G fait une rotation du vecteur(c,-s), qui fait un angle de $-\theta$ avec l'axe x, d'un angle de θ se qui le fait coïncidé avec l'axe x :

$$G\begin{pmatrix} c \\ -s \end{pmatrix}=\begin{pmatrix} c & -s \\ s & c \end{pmatrix}\begin{pmatrix} c \\ -s \end{pmatrix}=\begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Une rotation de Givens de $n\times n$ remplace un bloc de 2×2 d'une diagonale d'une matrice identité $n\times n$ par un bloc d'une rotation de Givens de 2×2.

$$G=\begin{pmatrix} 1 & 0 & & & \cdots & \cdots & \cdots & & & 0 \\ 0 & \ddots & \ddots & & & & & & & \\ & \ddots & \ddots & \ddots & & & & & & \\ & & \ddots & 1 & 0 & & & & & \vdots \\ \vdots & & & 0 & c & -s & & & & \vdots \\ \vdots & & & & s & c & 0 & & & \vdots \\ \vdots & & & & & 0 & 1 & \ddots & & \\ & & & & & & \ddots & \ddots & \ddots & \\ & & & & & & & \ddots & \ddots & 0 \\ 0 & & & \cdots & \cdots & \cdots & & & 0 & 1 \end{pmatrix}. \qquad (3.12)$$

on note une rotation de Givens de $n\times n$ $G_j(c,s)$ une rotation de type (3.12) avec une rotation de Givens de 2×2 pour la colonne et la ligne j et j+1.

Les rotations de Givens sont utilisées pour réduire la matrice de Hessenberg supérieur à une matrice triangulaire supérieure et par conséquent à résoudre le problème au moindre carrée résulter de l'algorithme de GMRES. Cette méthode peut être aussi utilisé pour la résolution des problèmes aux valeurs propres par la méthode de QR.

On réduit H à la forme triangulaire en premier par sa multiplication par une matrice de rotation de Givens qui annule $h_{2,1}$(et bien sur qui change la valeur de $h_{1,1}$ et les colonnes suivantes).

On défini $G_1 = G_1(c_1, s_1)$ par :

$$c_1 = h_{11} / \sqrt{h_{11}^2 + h_{21}^2} \quad \text{et} \quad s_1 = -h_{21} / \sqrt{h_{11}^2 + h_{21}^2}\ . \qquad (3.13)$$

Si on remplace H par G_1H, alors la première colonne de H ne contient maintenant qu'un seule élément non nule h_{11}. D'une manière similaire on peut appliquer $G_2(c_2, s_2)$ à H où :

$$c_2 = h_{22} / \sqrt{h_{22}^2 + h_{32}^2} \quad \text{et} \quad s_2 = -h_{32} / \sqrt{h_{22}^2 + h_{32}^2}\ . \qquad (3.14)$$

et qui annule h_{32}. Notons que l'application de G_2 n'affecte pas la première colonne, et similairement l'application de G_i n'affecte pas les i-1 colonnes précédentes.
En continuant ainsi et en mettant :

$$Q = G_n G_{n-1} .... G_2 G_1.$$

On peut voir que $QH = R$ est une triangulaire supérieure.

3.6 GMRES idée de base

Supposons avoir un projecteur orthogonale V_k envers K_k. alors, n'importe quelle $z \in K_k$ peut être écrit de la façon suivante :

$$z = \sum_{l=1}^{k} y_l v_l^k , \tag{3.15}$$

où v_l^k est la l^ième colonne de V_k.

On peut transformer (3.3) en un problème aux moindre carrée dans R^k pour les coefficients du vecteur y de $z = x - x_0$. Depuis que $x - x_0 = V_k y$, pour un certain $y \in R^k$ on doit avoir $x_k = x_0 + V_k y$ où y minimise :

$$\| b - A(x_0 + V_k y) \| = \| r_0 - AV_k y \| . \tag{3.16}$$

Ce qui réduit notre problème aux moindre carrée dans R^k à :

$$\min_{y \in R^k} \| r_0 - AV_k y \| . \tag{3.17}$$

Ceci est un problème aux moindre carré linéaire standard qui peut être résolu par une factorisation QR. Le problème avec une telle approche directe est tel que le produit matrice vecteur de A avec la matrice base V_k doit être performer à chaque itération. On utilise le processus d'Arnoldi pour représenter efficacement le problème (3.17), le problème au moindre carré résultant ne demande pas des multiplications supplémentaire de A par des vecteurs.

La structure d'une matrice de Hessenberg supérieure peut être exploité pour trouver la solution du problème aux moindre carrée d'une façon très efficace.

Le processus d'Arnoldi (sauf s'il termine prématurément avec une solution) produit la matrice V_k avec des colonnes orthonormales telle que :

$$AV_k = V_{k+1} H_k .$$

De la, pour certain $y^k \in R^k$, et on prenant

$$r_k = b - Ax_k = r_0 - A(x_k - x_0) = V_{k+1}(\beta e_1 - H_k y^k). \tag{3.18}$$

la relation $r_0 = V \beta e_1$ est vérifié grasse au choix fait pour la première colonne de V.

Alors, quand y^k minimise $\| \beta e_1 - H_k y \|$ envers R^k, on a :

$$x_k = x_0 + V_k y^k . \tag{3.19}$$

On note que lorsque y^k sera calculé, la norme de r_k peut être trouvé sans un calcul explicite de x_k et puis de $r_k = b - Ax_k$ qui peut être très pénalisant dans certain cas où *A* est très grande et pleine. En utilisant l'orthogonalité de V_{k+1}, on a :

$$\| r_k \| = \| V_{k+1}(\beta e_1 - H_k y^k) \| = \| \beta e_1 - H_k y^k \|_{R^{k+1}} . \tag{3.20}$$

et notre problème aux moindre carré (3.17) aura la forme suivante :

$$\min_{y \in R^k} \| \beta e_1 - H_k y \| . \tag{3.21}$$

Théorème :
Soit A une matrice non symétrique d'une grande taille n.
L'algorithme de GMRES trouvera la solution bien avant la $n^{ième}$ *itération.* [13]

3.7 GMRES algorithmes

Par l'application de la construction de Gram-Schmidt dans le processus d'Arnoldi on aura la matrice $k+1 \times k$ H_k de coefficients h_{ij} une matrice de Hessemberg supérieure.

En gardant le problème au moindre carrée donné par l'équation (3.21) telle qu'il est, et en se basant sur les informations et les explications fournis par les sections précédentes, et où le teste d'arrêt est que $\rho = \|r_k\|$ soit inférieur à une valeur ε fixé par l'utilisateur, on aura l'algorithme GMRES de base avec orthogonalisation de Gram-Schmidt classique qu'on note ***gmres_cgs*** qui à la forme suivante :

Algorithme 3.4 : algorithme *gmres_cgs* (x, b, A, ε , k*max*)

1. $r = b - Ax, v_1 = \frac{r}{\| r \|}, \rho = \| r \|, \beta = \rho, k = 0.$
2. tant que $\rho > \varepsilon$ et k≤ k*max*
   (a) $k = k + 1.$
   (b) pour j = 1,….,k $h_{jk} = (Av_k)^T v_j$
   (c) $v_{k+1} = Av_k - \sum_{j=1}^{k} h_{jk} v_j$

(d) $h_{k+1,k} = \| v_{k+1} \|$

(e) $v_{k+1} = \frac{v_{k+1}}{\| v_{k+1} \|}$

(f) minimise $\| \beta e_1 - H_k y^k \|_{R^{k+1}}$ envers R^k pour obtenir y^k

(g) $\rho = \| \beta e_1 - H_k y^k \|_{R^{k+1}}$

3. $x_k = x_0 + V_k y^k$.

Une importante propriété de GMRES est qu'il est indispensable de mémoriser la base de l'espace de Krylov au fur et à mesure que la progression des itérations. Pour réaliser k GMRES itérations on doit sauvegarder k vecteurs de tailles n. Pour les très grands systèmes cela peut devenir prohibitive. Une manière de procéder est d'itérer jusqu'à un k maximum qui représente le nombre maximum de vecteurs à sauvegarder. GMRES teste la norme du résidu $b - Ax_k$ et la compare à la valeur tolérée ε, si celle si est toujours supérieur GMRES redémarrera avec comme solution initiale le résultat de la dernière série d'itérations x_{kmax}. Cette version de GMRES avec redémarrage de l'algorithme est dénoté ***gmres(m)***. Il n'y a aucun théorème pour la convergence de cet algorithme et le redémarrage entraînera un retardement de la convergence qui est une résultante d'un autre problème plus gênant pour l'implémentation de l'algorithme gmres_cgs est le fait que les vecteurs v_i peuvent perdre leur orthogonalité à cause des calculs en précision finie (paragraphe 3.4.5) qui font accumuler les erreurs, s'il arrive que cette perte soit un peut plus importante l'évaluation du résidu peut ne pas être juste et la solution peut ne pas être atteinte. Ce qui veut dire que l'algorithme va stagné et ne fournira jamais de résultat.

Un remède partiel à ce problème est de modifier l'orthogonalisation classique de Gram-Schmidt : on remplace l'étape 2c de l'algorithme ***gmres_cgs*** par :

- $v_{k+1} = Av_k$
- pour $j = 1,....,k$

$$v_{k+1} = v_{k+1} - (v_{k+1}^T v_j) v_j .$$

Ce qui nous conduit à l'algorithme dit avec orthogonalisation de Gram-Schmidt modifié présenté si après et noté ***gmres_mgs*** :

Algorithme 3.5 : algorithme *gmres_mgs*(x, b, A, ε , k*max*)

1. $r = b - Ax, v_1 = \frac{r}{\| r \|}, \rho = \| r \|, \beta = \rho, k = 0.$

2. tant que $\rho > \varepsilon$ et k≤ k*max*

   (a) $k = k + 1.$

   (b) $v_{k+1} = Av_k$

   pour j = 1,….,k

      i. $h_{jk} = v_{k+1}^T v_j$

      ii. $v_{k+1} = v_{k+1} - h_{jk} v_j$

   (c) $h_{k+1,k} = \| v_{k+1} \|$

   (d) $v_{k+1} = \frac{v_{k+1}}{\| v_{k+1} \|}$

   (e) minimise $\| \beta e_1 - H_k y^k \|_{R^{k+1}}$ envers R^k pour obtenir y^k

   (f) $\rho = \| \beta e_1 - H_k y^k \|_{R^{k+1}}$

3. $x_k = x_0 + V_k y^k.$

Cette algorithme représente seulement l'introduction de l'orthogonalisation de Gram-Schmidt modifié dans le processus d'Arnoldi et pas la version redémarré de la méthode qui peut être appliquée pour n'importe quelle type d'orthogonalisation, et que nous présenterons un peut plus loin.

Une simple application dans l'algorithme **gmres_mgs** des idées exposée dans le paragraphe 3.6 a comme optique la résolution du problème au moindre carrée par l'application du produit de k rotations de Givens qui donne Q.

Mettons $g = \beta Q e_1$ et notons que la résolution du problème se réduit à :

$$\| \beta e_1 - H_k y^k \|_{R^{k+1}} = \| Q(\beta e_1 - H_k y^k) \|_{R^{k+1}} = \| g - R_k y^k \|_{R^{k+1}}, \qquad (3.22)$$

où R_k est une matrice triangulaire supérieure de $k+1 \times k$ résultante de la factorisation QR de H_k.

On peut performer la factorisation QR de H au fur et à mesure que la réalisation des itérations de l'algorithme ***gmres_mgs***. En fait, si $R_k = Q_k H_k$ et après orthogonalisation, on rajoute une colonne h_{k+2} à H_k, on peut faire une mis à jour de Q_k et R_k en premier lieu par une multiplication de h_{k+2} par Q_k (se qui va appliqué la première rotation de Givens à h_{k+2}), et puis réaliser la rotation de Givens G_{k+1} qui élimine le $(k+2)^{\text{ième}}$ élément de $Q_k h_{k+2}$, et finalement $Q_{k+1} = G_{k+1} Q_k$ et former R_{k+1} par l'augmentation de R_k par $G_{k+1} Q_k h_{k+2}$.

Ainsi après l'utilisation des rotations de Givens l'algorithme ***gmres_mgs*** sera noté ***gmres*** tout court. Pour nous c'est l'algorithme de GMRES de base, et il prendra la forme suivante :

Algorithme 3.6 : algorithme *gmres* (x, b, A, ε , k*max*)

1. $r = b - Ax, v_1 = \frac{r}{\| r \|}, \rho = \| r \|, \beta = \rho, k = 0, g = \rho(1, 0...., 0)^T \in R^{k\max+1}.$

2. tant que $\rho > \varepsilon$ et k$\leq$ k*max*
   (a) $k = k + 1$.
   (b) $v_{k+1} = Av_k$
      pour j = 1,….,k
         i. $h_{jk} = v_{k+1}^T v_j$
         ii. $v_{k+1} = v_{k+1} - h_{jk} v_j$
      fin pour
   (c) $h_{k+1,k} = \| v_{k+1} \|$
   (d) $v_{k+1} = \dfrac{v_{k+1}}{\| v_{k+1} \|}$
   (e)
      i. si k>1 appliquer Q_{k-1} à la k^{ième} colonne de H.
      ii. $\upsilon = \sqrt{h_{k,k}^2 + h_{k+1,k}^2}$.
      iii. $c_k = h_{k,k} / \upsilon, s_k = -h_{k+1,k} / \upsilon$
         $h_{k,k} = c_k h_{k,k} - s_k h_{k+1,k}, h_{k+1,k} = 0$.
      vi. $g = G_k(c_k, s_k) g$.
   (f) $\rho = | (g)_{k+1} |$.
3. $r_{i,j} = h_{i,j}$ pour $1 \leq i, j \leq k$.
   $(w)_i = (g)_i$ pour $1 \leq i \leq k$.
   résoudre le système triangulaire supérieure $Ry^k = w$.
4. $x_k = x_0 + V_k y^k$.

On close cette section par la présentation de la variante de l'algorithme dite avec redémarrage ou réinitialisation dénoté ***gmres(m)*** et proposé par Saad lui même pour détourné le problème de la taille du sous espace de Krylov quand le système initiale est de très grande taille. La convergence de cette variante est loin d'être assuré et même qu'elle est beaucoup moins rapide que les versions de base appelées ***full gmres*** et qui ont la structure de l'algorithme 3.6 avec des différences par fois intéressantes au niveau techniques de programmations, de minimisation des opérations à effectué, de gestion de la mémoire, de calcul de l'erreur et au niveau testes d'arrêt.
L'idée principale de cette variante et la structure générale est représentée si après :

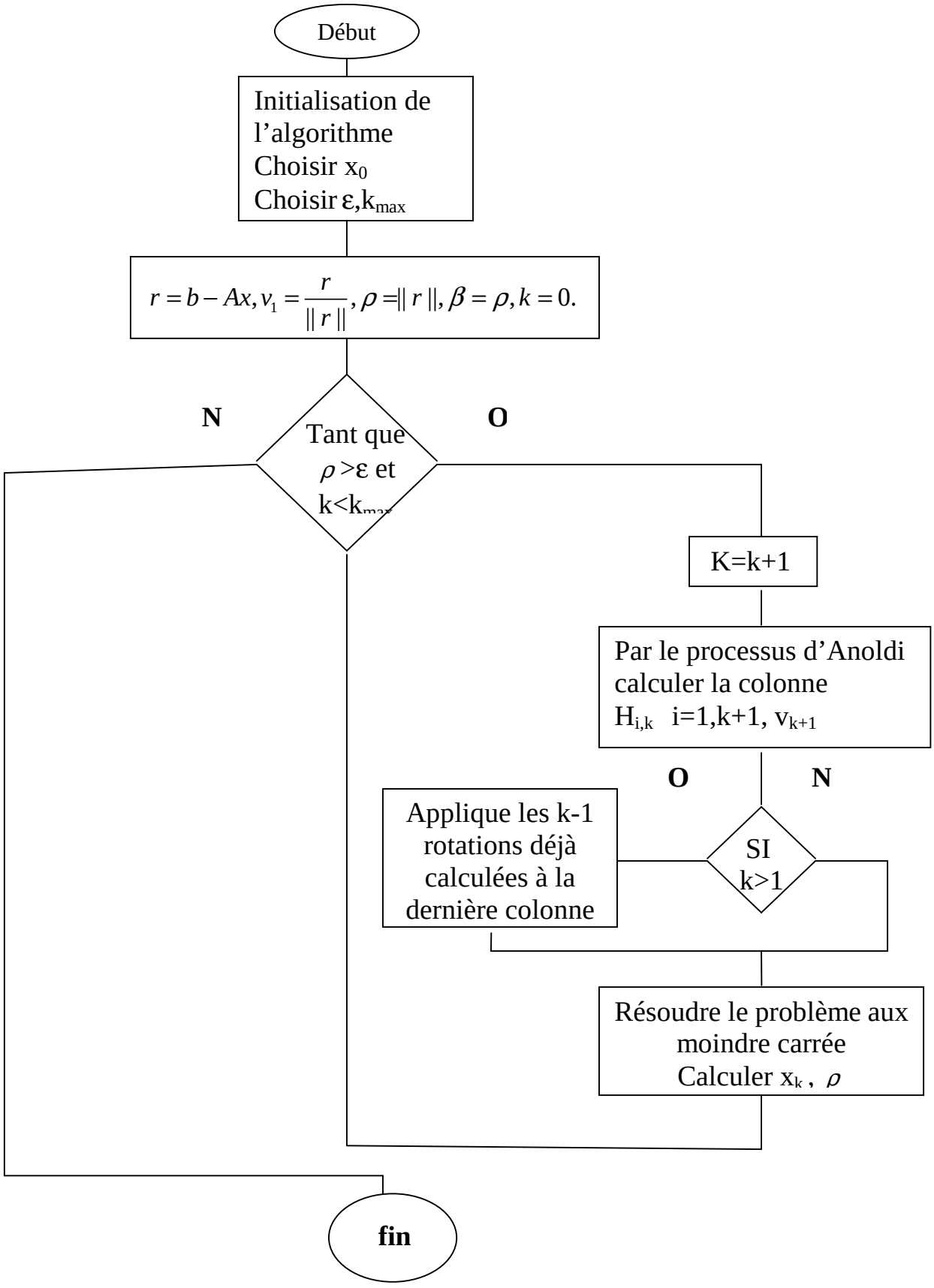
Algorithme 3.7 : algorithme *gmres(m)* (x, b, A, ε , k*max*, m)

1. ***gmres*** (x, b, A, ε , m)

2. $k = m$.

3. tant que $\rho > \varepsilon$ et k$\leq$ k*max*
   (a) ***gmres*** (x, b, A, ε , m)
   (b) $k = k + m$.

3.8 Organigramme



Chapitre 4

L'implémentation de l'algorithme GMRES

4.1 Introduction

La méthode GMRES est devenue très célèbre ces dernières années, plusieurs tentatives d'implémentations ont été réalisées par différents auteurs. La majorité des implémentations sont basées sur le même algorithme, à savoir l'algorithme GMRES avec redémarrage (restart), cet algorithme est représenté par l'algorithme 3.7 dans le paragraphe 3.7 du troisième chapitre. Chaque implémentation de cet algorithme comprend la touche personnelle de son auteur, et c'est ce qui fait la différence entre les différentes versions.

On utilise ce même algorithme avec une caractéristique fondamentale d'être utilisé comme la méthode de résolution pour le KSP qui est un logicielle de calcul des problèmes de mécanique utilisant les équations intégrales comme on a déjà vu dans le deuxième chapitre. Cela implique qu'on n'a pas d'interface de manipulation, ainsi on ne peut pas introduire des chois de certains paramètres. Tout doit être bien réglé et bien adapté pour notre cas. Aussi, notre but principale est de gagné un maximum sur le temps de calcul et sur la mémoire nécessaire pour le stockage des informations essentiels pour le bon déroulement de l'algorithme.

4.2 Le problème de moindre carré

A chaque étape j de l'algorithme, on doit résoudre le problème aux moindre carré de l'équation (3.21). la matrice $\overline{H_j}$ du problème est une matrice $(j+1)\times j$ de forme Hessemberg supérieur qu'on va utiliser la structure à notre faveur.

Premièrement, on fait une décomposition QR de la matrice $[\overline{H}_j, \beta e_1]$ ou Q est une matrice orthonormale et R est une matrice triangulaire supérieur de $(j+1)\times(j+1)$.

On développe le problème aux moindre carrée et on réécrire la solution du problème sous la forme :

$$y_j = R(1:j,1:j)^{-1} R(1:j, j+1)$$

ici, $R(1:j,1:j)$ représente la $j \times j$ première sous matrice de R, et $R(1:j, j+1)$ sa dernière colonne.

On effet,

$$[\overline{H}_j, \beta e_1] = QR$$

$$\Rightarrow$$

$$\overline{H_j} = QR(1:j,1:j) \quad et \quad \beta e_1 = QR(1:j, j+1)$$

le problème de moindre carrée étant

$$\min_{y \in R^j} \| \beta e_1 - \overline{H}_j y_j \|$$

aussi

$$\overline{H}_j y_j = \beta e_1$$

et

$$QR(1:j,1:j) y_j = QR(1:j, j+1)$$

enfin

$$y_j = R(1:j,1:j)^{-1} R(1:j, j+1)$$

On utilise les rotations de Givens pour faire la factorisation QR de $[\overline{H}_j, \beta e_1]$ pour les raisons présenté dans le chapitre précédent.

On sait que $\overline{H}_{j+1}$ est obtenue par simple addition d'une colonne à $\overline{H_j}$ de même en ce qui concerne $[\overline{H}_{j+1}, \beta e_1]$ par rapport à $[\overline{H}_j, \beta e_1]$ et les coefficients de R_{j+1} sont obtenue par une mise à jour qu'on va expliquer dans un exemple où j=3 :

$$R_j = \begin{pmatrix} + & + & + & + \\ 0 & + & + & + \\ 0 & 0 & + & + \\ 0 & 0 & 0 & + \end{pmatrix}$$

On calcule le vecteur $w = Q^T c$ où c est le vecteur ajouté à $[\overline{H}_j, \beta e_1]$, on l'introduit entre la dernière et l'avant dernière colonne de R_j et on aura une matrice transitoire $\widetilde{R}_j$ de $(j+2)\times(j+2)$:

$$\widetilde{R}_j = \begin{pmatrix} + & + & + & * & + \\ 0 & + & + & * & + \\ 0 & 0 & + & * & + \\ 0 & 0 & 0 & * & + \\ 0 & 0 & 0 & * & 0 \end{pmatrix}$$

où

$$w = \begin{pmatrix} * \\ * \\ * \\ * \\ * \end{pmatrix}$$

On applique les rotations de Givens à $\widetilde{R}_j$ où l'élément nul sera $\widetilde{R}_j(j+2, j+1)$ la matrice résultante est la matrice R_{j+1} :

$$R_{j+1} = \begin{pmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & + \end{pmatrix}$$

Ainsi, on a bien notre factorisation à moindre coût au niveau temps de calcul, et en plus cette procédure est très bien adaptée à notre algorithme, essentiellement parce qu'elle peut être effectuée à chaque itération sans refaire les calculs précédents.

4.3 Calcul de V_j

La qualité de l'orthogonalité de V_j joue un rôle important dans l'algorithme GMRES comme critère de convergence. Comme on a déjà vu, la perte de l'orthogonalité entraîne une grande perte de convergence. L'implémentation d'un algorithme à très bonne orthogonalité est très pénalisant point de vue temps de calcul.

Pour notre implémentation, on utilise le processus d'arnoldi avec l'algorithme de Gram Schmidt modifié où on gagne sur l'orthogonalité de la base tout en perdant un minimum sur le temps de calcul.

4.4 Calcul de résidu au redémarrage

La réalisation d'un produit matrice-vecteur peut par fois être compliqué et prend du temps pour les systèmes de grande taille, même si on verra par la suite qu'on essai de redémarrer l'algorithme le moins possible, il y'aura toujours des redémarrages dans des cas où la recherche de la solution sera très compliqué et prendra trop de temps.

Dans ce paragraphe on essai de gagner du temps dans le calcul du résidu, et ça en le réécrivant sous une autre forme qui rend le calcul du produit matrice-vecteur (le vecteur ici est le résidu) plus facile à calculer.

Pour cela, on doit calculer le résidu juste avant le redémarrage de l'algorithme. On aura alors le nouveau $x_0 = x_m$, et on peut écrire $r_0 = b - Ax_m$ avec $x_m = x_0 + V_m y_m$. On développe comme suit :

$$r_0 = b - A(x_0 + V_m y_m)$$
$$r_0 = V_{m+1}(\beta e_1 - \bar{H} y_m)$$
$$r_0 = V_{m+1} Q_m (Q_m^T \beta e_1 - \begin{bmatrix} R(1:m,1:m) \\ 0 \end{bmatrix} y_m)$$
$$r_0 = V_{m+1} Q_m \begin{bmatrix} 0 \\ r_{m+1,m+1} \end{bmatrix}$$

Le calcul du résidue revient à faire une combinaison faire une combinaison linéaire des vecteurs de la base issu du processus d'arnoldi, et les coefficients de la combinaison sont obtenues en appliquant les rotations de Givens dans le sens inverse à un vecteur qui a tous ses coefficients nuls sauf le dernier.

Cette procédure demande n(2m+1)+2m opérations arithmétiques au lieu de 2n(m+1) opérations.

Aussi, on peut voire que

$$\| r_j \|_2 = \| b - Ax_j \|_2 = \| \beta e_1 - \bar{H}_j y_j \|_2 = | r_{j+1,j+1} |$$

4.5 Réglage des paramètres

4.5.1 Le paramètre m

On introduit à l'algorithme qu'on va utilisé une modification principale qui joue un rôle principale dans son déroulement et qui réside dans l'unification des deux paramètres k et m on un seule paramètre m. on va étudier son influence sur la convergence de l'algorithme et l'optimisé envers le temps de calcul et le nombre d'itérations. On représente l'influence de ce paramètre sur le temps de calcul et sur le nombre d'itération. Pour plus de clarté on représente le temps de calcul normé par le temps minimale en fonction de m/n, et pareille pour le nombre d'itération. Ainsi on aura un graphe plus représentatif des caractéristiques de l'algorithme.

On observe d'abord ces deux exemples :

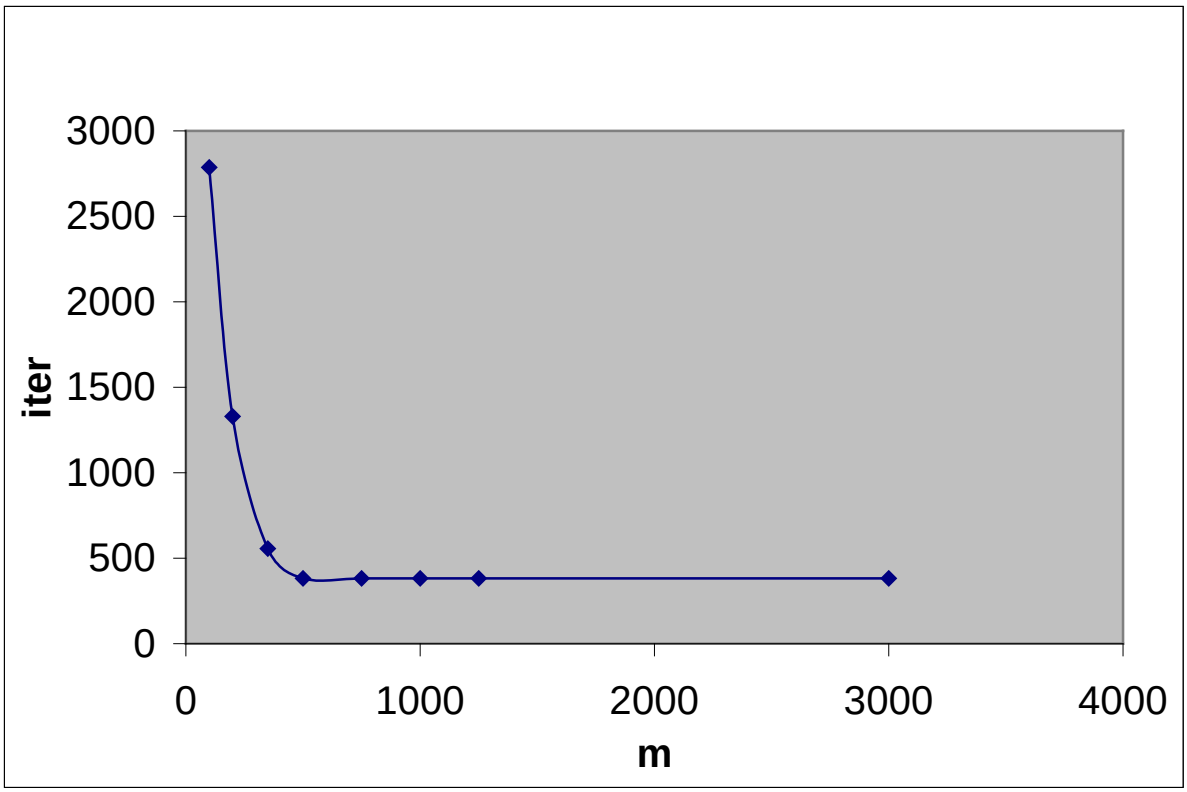


FIG.4.1 : nombre d'itération en fonction de m pour un problème à 3276ddl

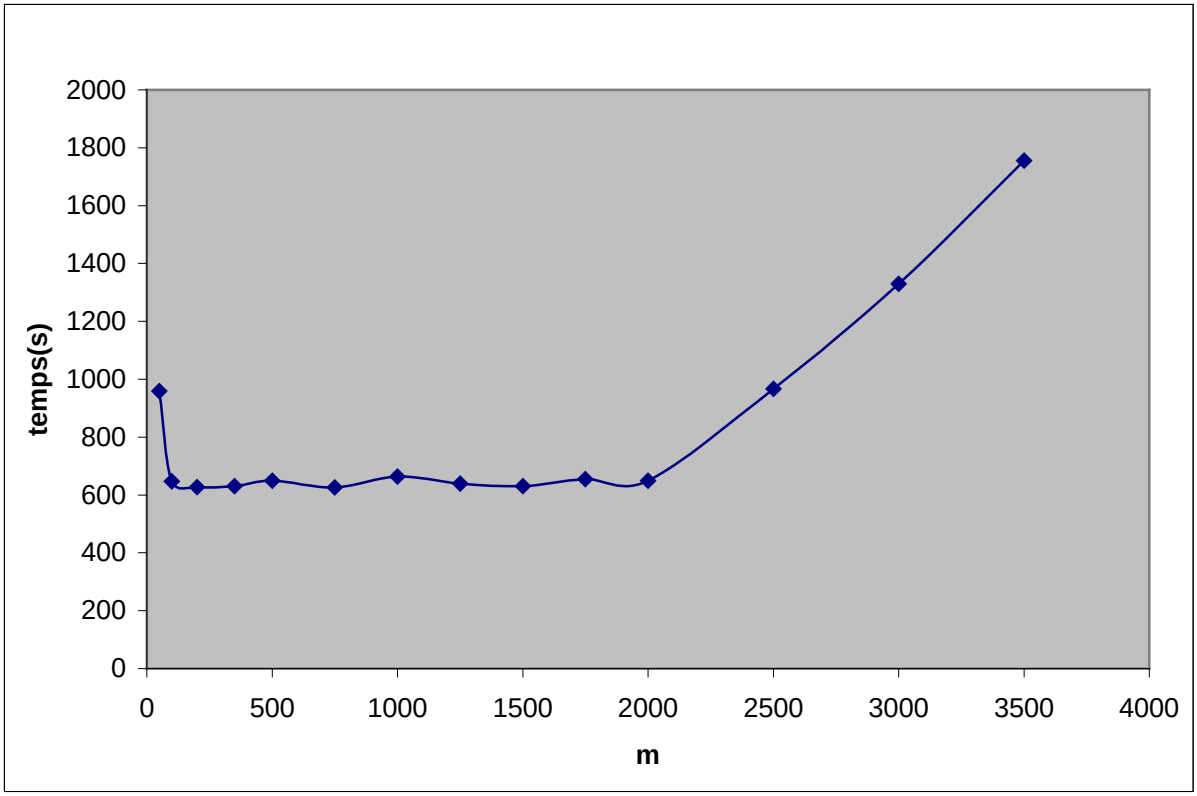


FIG.4.3 : temps de calcul en fonction de m pour un problème à 9216ddl

En observant les graphes on remarque qu'après une certaine limite l'algorithme ne fait plus d'itération, ce qui veut dire qu'il a trouvé la solution et il s'est arrêté. Ceci dit le temps de calcul recommence à augmenté quand m devient un peut plus grand à cause du stockage nécessaire de la base V et de la matrice H dont les tailles augmente avec m. alors, on ne peut pas se donner le lux de prendre m grand et éviter le cassement de tête due à la détermination du nombre d'itérations minimale.

Un autre point très important est que le nombre d'itération comme le tems de calcul descend très rapidement d'une valeurs très grande et même inacceptable à une valeur acceptable et puis optimale, ce qui nous incite à déterminer aussi la borne inférieur du paramètre m.

On lance les calculs sur plusieurs exemple et on représente les résultats dans les graphes qui suivent :

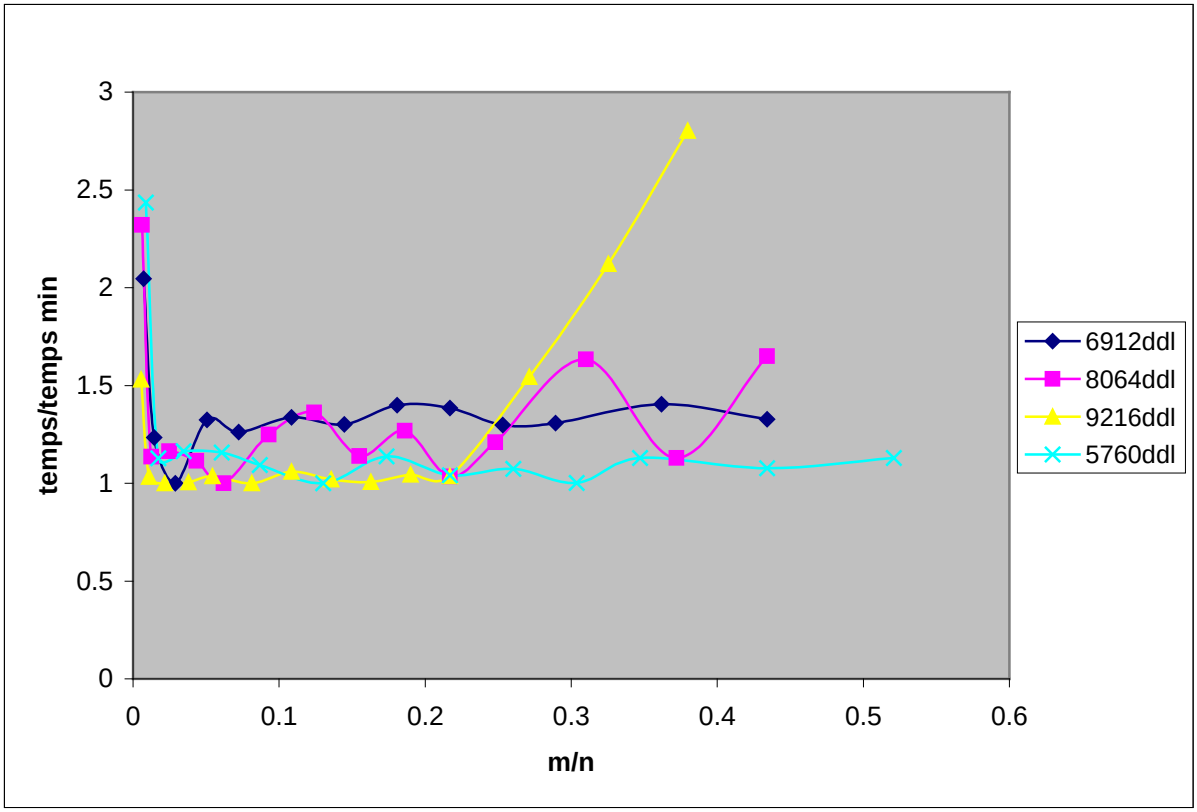


FIG.4.3 : détermination de m optimale via le temps de calcul

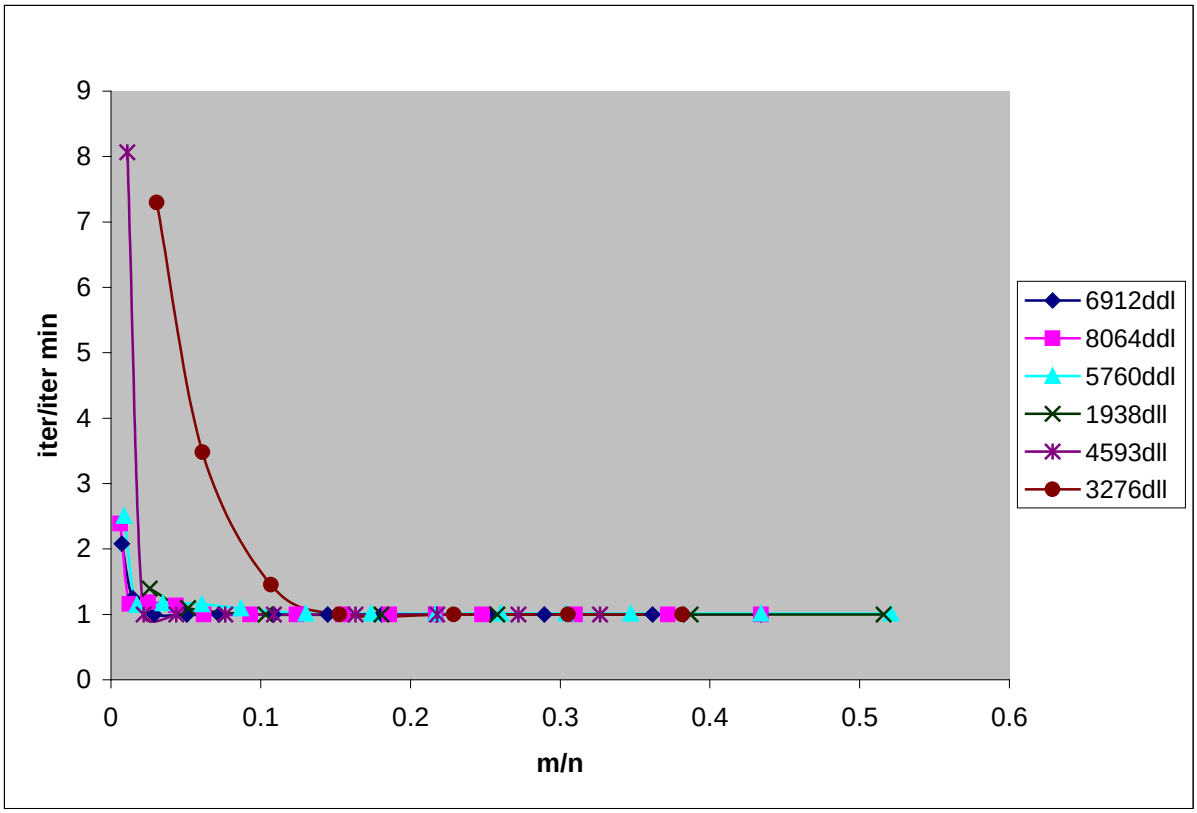


FIG.4.4 : détermination de m optimale via le nombre d'itérations

Apres observation des graphes on peut voire qu'il faut à tous prix éviter les valeurs de m inférieur à $0,05\times n$ puisque ça donne des temps de calculs et des nombres d'itérations qui tendent vers l'infinie. On peut voir aussi, surtout dans le graphe de la figure FIG 4.4 des nombres d'itérations, que ça sert à rien d'augmenter m au dessus de $0,15\times n$ puisque on aura atteint la valeur minimale et on aura plus rien à gagner. Au contraire, augmenter m entraînera l'utilisation de plus de mémoire et augmente aussi le temps de calcul.

De ça, on peut affirmer que m optimal est compris entre $0,05\times n$ et $0,15\times n$.

Pour une utilisation générale du logiciel on choisie la borne supérieur de l'intervalle comme valeurs par défaut, dans tous ce qui suit la valeur de m est fixée à $0,15\times n$.

4.5.2 Le paramètre m_{max}

On a vu dans les paragraphes 3.4 3.6 et 3.7 du chapitre précédent qu'on a besoin de stocké les deux matrice V et H, et que la taille de V est de $n\times m$ et la taille de H est de $(m+1)\times(m+1)$.

Par ailleurs, d'après le paragraphe précédent on a $m = 0,15\times n$ .Et on sais que le code qu'on est entrain de développé est destiné à résoudre des système à très grande taille :

$$n \text{ est très grand } \Rightarrow m \text{ aussi peut être très grand}$$

Ce qui veut dire que le stockage de V et H peut devenir très gênant voir même impossible dans des ordinateurs à performances modestes. Cela nous oblige à imposer une valeur maximale à m qui sera appelée m_{max} et qui dépendra des caractéristiques de la machine où les calculs seront effectués.

On prendra alors $m=\min(0,15\times n, m_{max})$.

Dans le graphe qui suit, on représente le temps de calcul en fonction de m pour un exemple à 1152 degré de liberté dans deux machines qui n'ont pas les même capacités de mémoire. La première machine est un ordinateur Pentium III avec une mémoire RAM de 96 Mo, et la deuxième est un Pentium IV avec 1Go de RAM.

Pour pouvoir comparer les résultats et les tracer sur le même graphe, en représente le temps par la grandeur adimensionnelle $(t-t_{min}) / t_{min}$.

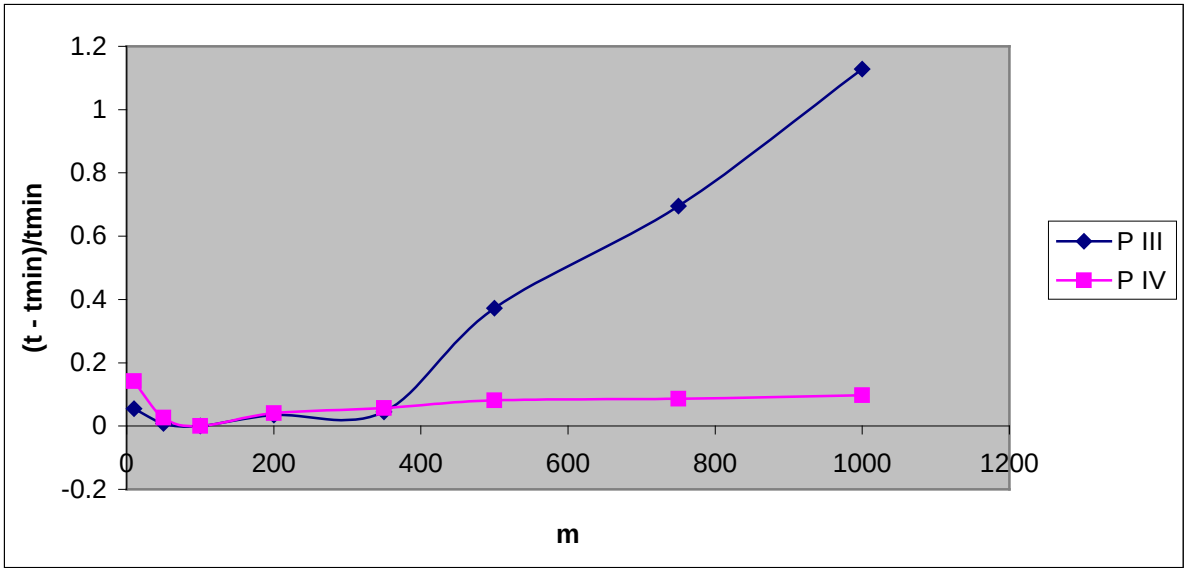


FIG.4.5 : effet des caractéristiques de la machine

On remarque clairement que les deux graphes suivent la même allure à gauche. Mais à partir d'une certaine valeur de m, le graphe qui correspond au premier ordinateur commence à augmenter et à s'écarter de celui de deuxième d'une façon très rapide. Ce qui veut dire que l'algorithme prend de plus en plus de temps pour trouver la solution, pourtant, on à vue dans les graphes des nombres d'itérations que ces derniers ne s'accroissent pas après leurs valeurs minimales.

On trace un deuxième graphe semblable au précédent mais on augmente la taille du système traité par le deuxième ordinateur, la nouvelle taille sera 9216 degrés de liberté.
On aura le graphe suivant :

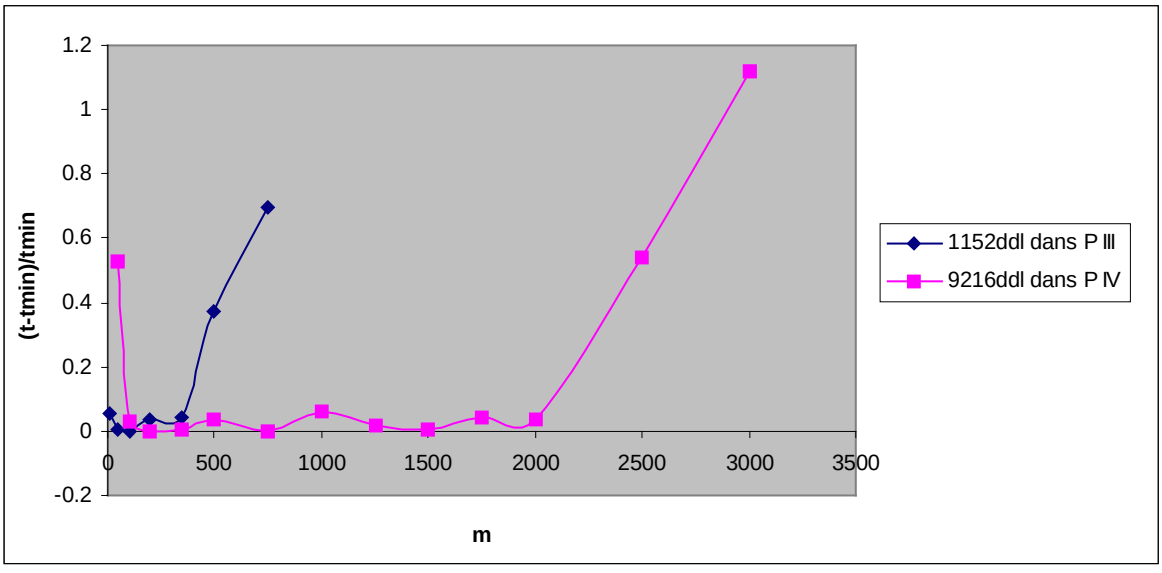


FIG.4.6 : la ressemblance du comportement des différentes machines

D'après ce deuxième graphe on voie l'apparition du phénomène qui était absent dans le cas précédent pour le Pentium IV. Le comportement de la deuxième machine est devenu semblable au comportement de la première dés qu'on a augmenté suffisamment la taille du système à résoudre.

On remarque que la valeur de m_{max} est de à peut prés de 400 pour le premier ordinateur et de 2000 pour le deuxième. De là, on voie bien que le paramètre m_{max} ne dépend que de la machine et pas de l'algorithme.

On peut dire aussi que l'algorithme ne fonctionnera pas d'une façon optimale si $m_{max} < (0,05 \times n)$.

4.5.3 Le paramètre restart

Puisqu'on ne peut pas prévoir la forme exacte de la matrice K et qu'il y a des cas où l'algorithme GMRES devient très instable on doit introduire un autre paramètre de sécurité qui limite le nombre maximal des itérations. Ce paramètre était présent dans l'algorithme du paragraphe 3.7 et représenté par k_{max} même Sauf qu'on l'a supprimé dans le paragraphe 4.5.1 lorsqu'on a fait la modification de l'algorithme et l'unification de m et k. on ne peut pas le restauré tel qu'il est car on n'a plus de k pour avoir un k_{max}, et en plus il ne remplie pas bien sa fonction dans notre cas.
On définie le paramètre restart comme le nombre maximum des cycle de redémarrage. Il doit être fixé antérieurement à l'utilisation du code dans le logiciel et il sera pour n'importe qu'elle taille du système. On effet, le travail effectué sur le paramètre m a pour optique de trouvé la solution dans le premier cycle. Si la première tentation échoue, on procède à une deuxième et puis une troisième et ainsi de suite. La seule différence entre les tentations successives est le vecteur initial. En limite le nombre de tentation indépendamment de la taille du système n. Ceci dit le nombre maximal des itérations sera directement et relativement lié à n et il sera égale à $restart \times m$, et puisqu'on a $m = 0,15 \times n$; on a alors le nombre maximal des itérations égale à $restart \times 0,15 \times n$

On n'a pas cette liaison entre le nombre maximal des itérations et la taille n si on utilise le paramètre k_{max} de l'algorithme 3.7. Et avec l'absence d'une interface de manipulation le nombre maximale des itérations sera égale à k_{max} et sera le même pour toutes les tailles confondues. Chose qu'on ne peut pas toléré si on veut réaliser un vrai code pour un vrai logiciel.

On le voie bien sur le graphe ci-dessous, où on a tracé la relation entre le nombre de redémarrage effectué en fonction du nombre d'itération normé par le nombre d'itération minimale, que restart n'atteint pas vingt pour un rapport du nombre d'itération sur le nombre d'itération optimale égale à cinq. Et déjà on ne peut pas permettre que le rapport atteint cette valeur pour permettre qu'il la dépasse.

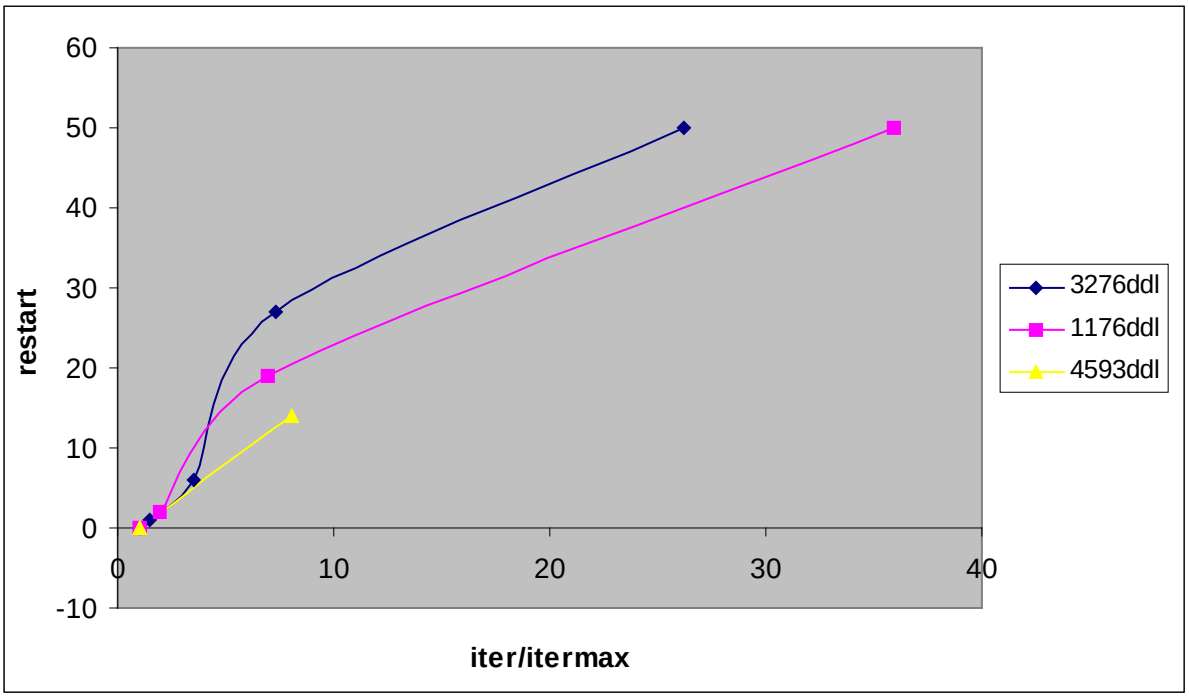


FIG.4.7 : la relation entre le nombre de redémarrage et le nombre des itérations

4.6 Le critère d'arrêt

Notre critère d'arrêt est basé sur le calcul de la distance entre la solution approchée et la solution exacte qui revient à calculer la norme de $||b-Ax_j||$ et la comparer à la précision demandée sur les éléments de la solution approchée. Ainsi, si en cherche une précision de ε en a qu'à imposer comme condition de sortie de l'algorithme la condition suivante : $||b-Ax_j|| \leq \varepsilon$.

Il faut dire que le calcul de $||b-Ax_j||$ à chaque itération peut devenir très coûteux quand la taille du système devient importante, merci à l'égalité

$$|| r_j ||_2 = || b - Ax_j ||_2 = || \beta e_1 - \bar{H}_j y_j ||_2 = | r_{j+1,j+1} |$$

où on peut voir que la norme du résidu est donné directement dans l'algorithme durant la résolution du problème des moindres carrées ce qui réduit considérablement le coût et le temps de résolution.

Ceci dit, il est connu que dans l'arithmétique en précision finie, le calcul du résidu issu du processus d'arnoldi peut être significativement différant du résidu réel, alors en va suivre la stratégie suivante :

On fait une première itération jusqu'à ce que $|r_{j+1,j+1}|$ vérifie le critère d'arrêt $(|r_{j+1,j+1}| \leq \varepsilon)$ puis une deuxième itération jusqu'à ce que $||b - Ax_j||$ le vérifie $(||b - Ax_j|| \leq \varepsilon)$.

En procédant de la sorte notre bute est de minimiser les opérations relatives aux calcul du résidu (le calcul du produit $A \times x$) en ayant en même temps un réel critère d'arrêt.

Ce premier critère d'arrêt nous permet d'avoir une solution approché de la solution exacte d'une manière simple et efficace mais lors que les différents coefficients du vecteur solution sont disproportionné elle présente un petit problème qui peut rapidement devenir très gênant surtout quand on a quelques coefficients qui approchent la valeur de ε. Pire encor, quand ces coefficients sont inférieur à ε et dans ce cas on peut avoir des résultats très erronée(l'incertitude des résultats sera supérieur aux résultats).

Pour avoir un algorithme plus fidèles, on peut normé chaque coefficient du résidu par le coefficient qui lui correspond du vecteur solution, calculer la norme du vecteur résultant et puis la comparer au pourcentage de la précision voulu. Dans ce cas, ε représente la tolérance sur ce rapport et pas la tolérance des résultats. Ainsi l'algorithme continua à itérer jusqu'à ce qu'il atteint la précision relative demandé sur chaque coefficient à part. cette manière de procéder augmente un peut le nombre d'itération et par conséquent temps de calcul ce qui est à l'opposé de notre objectif de départ mais qui nous donne une précision très fidèle de la solution quelle que soit la forme de celle si.

Un autre problème qu'on rencontre dans l'implémentation de ce critère est ce qu'on doit effectué une vérification de la va leur de tous les coefficients de la solution puisqu'ils représentent le dénominateur du rapport et en aucun cas un coefficient pourra être nul. Pour

détourné ce problème en rajoutant au dénominateur une très petite valeur juste pour éviter la division par zéro. Cela entraînera une toute petite perte de précision de la solution pour les déplacements nuls ou très petits.

Une autre manière de procéder est de relativisé la norme du résidu $||r_j||$ par la norme du premier résidu $||r_0||$ et comparer la résultante à une valeur qu'on fixe et qui sera stocké dans ε. Ce critère est bien adaptée pour les cas où la solution initiale est assai proche de la solution exacte, mais elle n'est pas efficace dans le cas où la solution initiale est arbitraire et peut par ce fait être très loin de la solution exacte. Par ailleurs, on perd les informations que nous avions sur la précision, en fait on a aucune information précise sur la valeur du résidu ou sur la précision de la solution. Tous ce qu'on a est ce que la solution finale comparée à la solution de départ est améliorée par un rapport égal à $1/\varepsilon$. On voie bien que ce critère ne nous convient pas et que cette approche ne peut être introduite dans un logiciel de calcul en mécanique, mais on peut introduire une modification qui consiste à relativisé la norme du résidu $||r_j||$ pas par le résidu initiale, mais par la norme du résidu précédent $||r_{j-1}||$ et observer la tendance de la résultante vers 1. Le calcul s'arrête des que l'algorithme stagne, ce qui est possible d'y parvenir.

Ce critère nous empêche de faire des itération sans profit (sans amélioré la solution), c'est très important pour la robustesse de notre algorithme. Malheureusement même si on a plus de chance d'avoir des résultats meilleurs, cet algorithme ne nous garantie pas la fidélité des résultat encor moins la rapidité de la convergence, mais on peut utilisé cette approche en combinaison avec un autre critère d'arrêt, ainsi on aura pas un deuxième critère d'arrêt mais une mesure de sécurité en cas où l'exemple traité sera si délicat que l'algorithme de GMRES n'atteint pas la précision demandé par l'utilisateur.

4.7 L'influence de la précision sur le nombre d'itération

On représente dans ci-dessous l'influence de la valeur de ε sur la convergence de l'algorithme, on trace le graphe représentant le nombre d'itération en fonction de la précision demandé, où la précision est calculée comme étant l'inverse de la tolérance :

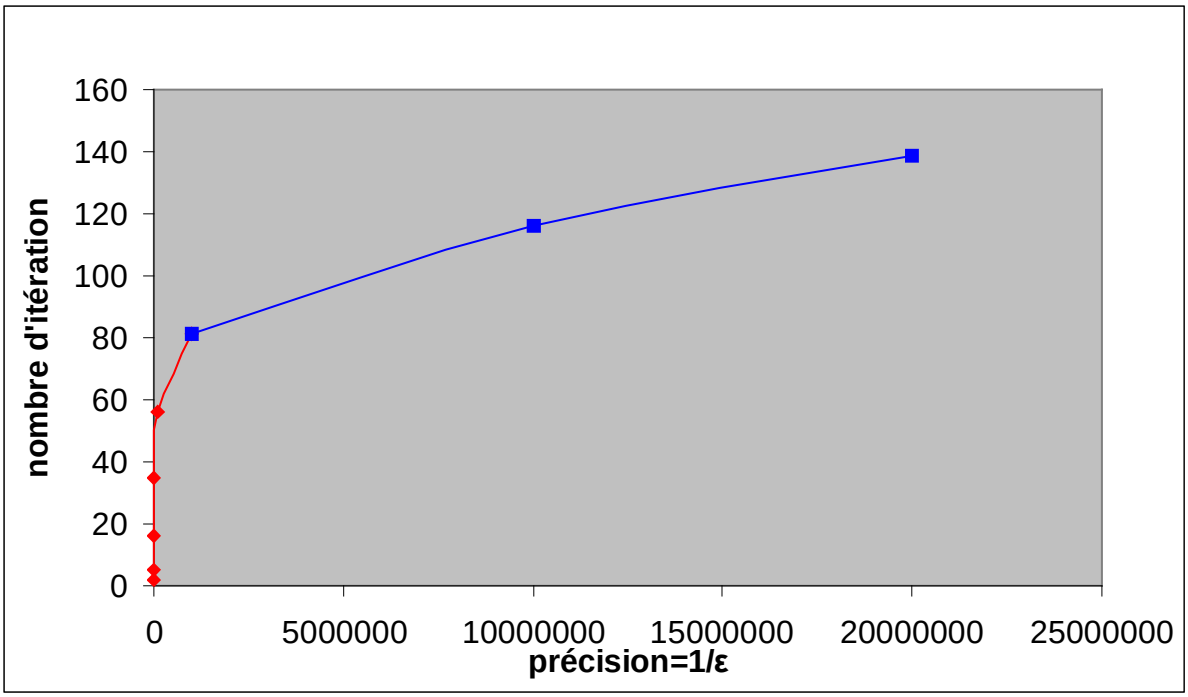


FIG.4.8 : influence de la précision sur le nombre d'itérations

On distingue deux zones :

Une première zone représenté par la couleur rouge où on remarque une très grande influence de la précision sur le nombre d'itération, une petite augmentation de la précision entraîne une très grande augmentation du nombre d'itération et par conséquent du temps de calcul (perdant pas de vue que notre objectif est de le minimiser).

Une deuxième zone représentée par la couleur bleue où on voit clairement qu'une augmentation même considérable de la précision n'engendre qu'une petite augmentation du nombre d'itération.

Ainsi, si on se retrouve dans la première zone (et c'est le plus fréquent) on doit faire très attention en fixant la précision, on doit pas être gourmand au niveau précision des résultats car cela peut ce révéler très pénalisant au niveau temps de calcul surtout pour de grands systèmes.

Pour la deuxième zone on peut se permettre d'augmenté un peut la précision pour avoir une certaine garantie des résultats sans se méfier des conséquences que ça aura sur le temps de calcule.

Chapitre 5

Le préconditionnement

5.1 Introduction

La démarche suivie dans le chapitre précédent nous offre des performances optimales pour une très large plage de la taille des problèmes traités, mais de toute façon on doit envisager le traitement des problème qui dépasse cette plage, au delà de 10000 degré de liberté. Le redémarrage de l'algorithme devient alors inévitable, ce qui provoque une perte d'orthogonalité et un ralentissement de la convergence comme on a déjà vu dans le troisième chapitre. Comme alternative, plusieurs auteurs ont proposé de combiner GMRES avec d'autre méthodes itératives plus simple

La combinaison de la méthode GMRES avec d'autres méthodes s'appel Hybrid GMRES. L'utilisation de la méthode GMRES elle même en première et en deuxième phase de résolution ou préparer le système initiale à résoudre avant même de commencer les itération et utiliser une seule phase de résolution s'appel Preconditioned GMRES (GMRES préconditionnée).

5.2. Principe du préconditionnement

Il est plus facile de résoudre le système $MAx = Mb$ ou $AMy = b \quad et \quad x = My$ avec MA (respectivement AM) est proche de l'identité que de résoudre le système original. On parle alors du préconditionnement à gauche (premier cas), du préconditionnement à droite (deuxième cas)ou du préconditionnement mixte pour la résolution du système $RALy = Rb \quad et \quad x = Ly$ avec RAL proche de l'identité.

L'objectif d'une tel démarche est de réduire le nombre d'itération au détriment de rajouter des opérations supplémentaires ou réaliser plus d'opération par itération.

5.3. Différents types du préconditionnement

5.3.1 GMRES hybride

L'idée est de performer un nombre d'itération de GMRES limité et puis calculer les valeurs de Ritz. Ainsi les information requise nous permet de lancer un autre procédé itérative comme la méthode de Richardson ou de Chebychev par exemple en seconde phase de la résolution.

5.3.2 Préconditionneur polynomiale

La résolution se fait sur deux phases comme pour GMRES hybride sauf qu'on utilise les valeurs de Ritz pour construire un polynôme caractéristique qui engendre les valeurs propres du système. On applique se polynôme à la matrice du système et on l'utilise comme préconditionneur à droite Le schéma d'une telle approche est le suivant :

Réaliser un nombre limité des itérations appliqué à $Ax = b$
(première étape)

↓

calculer les valeurs de Ritz → calculer les paramètres du préconditionneur polynomial $P_n(A)$

↓

appliquer GMRES à $AP_n(A)y = b, x = P_n(A)y$ avec approche initiale de la solution tiré de la première étape
(deuxième étape)

5.3.3. GMRES-DR (Deflating Restart)

il a été observé qu'enlever les plus petites valeurs propres améliore le taux de convergence de l'algorithme GMRES. Une manière de procéder pour éliminer les valeurs propres indésirables est de rajouter les vecteurs propres correspondant au sous espace de Krylov. C'est ça la philosophie qui gouverne le développement de cette variante de GMRES appelé GMRES-DR.

Cette méthode recycle quelques approximations de vecteurs propres d'un redémarrage à un autre par l'augmentation du prochain sous espace de Krylov avec ces approximations.

5.3.4. Décomposition LU Incomplète

L'idée est de chercher une décomposition LU incomplète de A, approchée A par M = LU avec L triangulaire inférieur et U triangulaire supérieur. Mais en imposant aux coefficients de L d'être nuls en dehors d'un ensemble fixé et des coefficients nuls sauf dans un ensemble fixé pourU.

Si la décomposition LU incomplète de A est telle que les coefficients non nuls de L et U sont situés à l'emplacement des coefficients non nuls de A, la décomposition incomplète est appelée ILU0.

5.3.5. Approximation de l'inverse

L'idée est d'approcher directement l'inverse de A : $M \approx A^{-1}$. Il s'agie de trouver la matrice M tel que $AM \approx I$ pour le préconditionnement à droite et $MA \approx I$ pour le préconditionnement à gauche.

Il y a plusieurs approches pour construire la matrice M. On a la décomposition de A suivante :

$$A = VHV^T$$

on peut approcher l'inverse de A par approcher l'inverse de la décomposition :

$$M \approx A^{-1} \approx VH^{-1}V^T$$

Une autre approche totalement différente est basée sur la minimisation de la norme suivante :

$$\min || I - MA ||^2$$

ce qui nous conduit à :

$$\min || e_i - Am_i ||^2$$

et enfin :

$$Am_i = e_i$$

où m_i i=1,n sont les colonnes de la matrice M.
On peut déterminer avec une certaine tolérance les colonnes désirées de M .

5.4. Critère d'arrêt

Le critère d'arrêt étant basé essentiellement sur la norme du résidu, son changement ou pas est liée directement au fait que le résidu change ou pas. Ainsi, on peut observer que le résidu ne change pas dans le cas d'un préconditionnement à droite. En fait, on a à résoudre le système suivant :

$$A\,M\,y \;=\; b$$

$$x = My$$

au lieu du système initiale :

$$Ax \;=\; b$$

de là, le résidu pour le système initiale s'écrit :

$$r \;=\; Ax - b$$

et le résidu pour le système préconditionné s'écrit :

$$\tilde{r} = AMy - b = Ax - b = r$$

On voie bien que les deux résidus sont équivalent. Alors, le critère d'arrêt demeure le même pour le cas d'un préconditionnement à droite, et ça c'est un des plus grands avantages de ce type de préconditionnement car l'algorithme reste le même.

Pour le préconditionnement à gauche, le système à résoudre est :

$$MAx = My$$

le résidu dans ce cas prendra la forme suivante :

$$\tilde{r} = MAx - My = M(Ax - y) = Mr$$

de cette relation, on voie bien que nous somme contraint de changer le critère d'arrêt pour qu'il soit fiable, et même avec ça on aura jamais la même maîtrise des résultats qu'au paravent.

Pour le préconditionneur mixte, le système à résoudre s'écrit :

$$M_1 A M_2 y = M_1 b$$

et

$$x = M_2 y$$

et le résidu aura les même propriétés que dans le cas du préconditionnement à gauche :

$$\tilde{r} = M_1 r$$

5.5. Implémentation du préconditionneur

Il s'agit d'abord de trouver un préconditionneur bien adapté à notre système d'équation. Les coefficients de la matrice du système à résoudre n'ont pas de caractéristiques qui faciliteront la prévision de la forme de la matrice, sauf le fait d'avoir une diagonale dominante. C'est ce dernier point qu'on va exploiter pour construire notre préconditionneur.

Pour commencer, on choisit un préconditionnement à droite pour la simplicité de son implantation et surtout pour la préservation de la maîtrise des résultats fournis par l'algorithme.

En dehors de la classification par type des préconditionneurs, on peut les classer aussi par leur philosophie ou leur objectifs. On distingue alors deux classe de préconditionneur. La première classe cherche à modifier la structure de la matrice du système pour qu'il soit plus facile

à résoudre, on cherche alors à atteindre la solution avec moins d'itération que pour le système initiale. On peut dire que la modification qui subit le système de départ est un changement de variable qui nous offre une première approximation de la solution ce qui nous fait gagner plusieurs itérations.

La deuxième classe cible l'origine de la dégradation de la convergence après redémarrage et tente de ramener le taux de convergence de GMRES redémarré à un taux proche de celui de GMRES complet. Cette classe est destinée uniquement à GMRES avec redémarrage alors que la première classe peut être introduite sur n'importe quelle variante de GMRES.

Pour un premier essai, et vu que le travail effectué dans le chapitre précédent nous permet de résoudre les problèmes sans redémarrage de l'algorithme ou avec peu de redémarrage au moins pour la plage de problèmes ciblés (réalisés sur un PC pas sur des calculateurs à très haute performance), il est plus intéressant de chercher un préconditionneur de la première classe.

La notion de première classe et deuxième classe n'existe pas dans la littérature, c'est juste une appellation qu'on a introduit pour faire une comparaison entre les préconditionneurs.

Parmi les préconditionneurs exposés dans le paragraphe 5.3 les préconditionneurs de la première classe sont la décomposition LU incomplète et l'approximation de l'inverse. Alors que GMRES Hybride, le préconditionneur polynomiale et GMRES-DR font partie de la deuxième classe.

Après les explications et les arguments déployés jusque là, on choisit d'utiliser un préconditionneur de la première classe et on l'utilise comme préconditionneur à droite. Reste de trouver un bon préconditionneur bien adapté pour notre système.

On a déjà vu que la matrice du système est à diagonale dominante. Il existe dans la bibliographie un préconditionneur mixte de la première classe appelé le préconditionneur diagonale, mais il est à la base destiné à des matrice hermitiennes définies positives:

Soit A une matrice hermitienne définie positive et D la matrice diagonale dont les coefficients diagonaux sont les racines carrées de ceux de A. la matrice préconditionnée a la forme suivante :

$$D^{-1} A D^{-1}$$

Pour notre cas, on ne peut pas utiliser ce préconditionneur tel qu'il est, d'abord, parce qu'on veut un préconditionneur à droite et pas mixte, et encor, parce qu'a priori les termes diagonaux de A peuvent être négative et on peut pas alors faire leurs racines carrées dans IR.

Ces problèmes disparaisse en utilisant une matrice diagonale D comme préconditionneur à droite et dont les coefficients diagonaux ne sont pas les racines carrées de ceux de A mais leurs inverses. Et on aura à résoudre le système :

$$A\,D\,y = b$$

et

$$x = D\,y$$

ce préconditionneur est très facile à implémenter et n'a vraiment pas de coût additionnel.

5.6. Algorithme

i. lire les données (n,A,b,…)
ii. calculer les coefficients de la matrice préconditionneur D.
iii. appliquer GMRES pour le système AD y = b.
iv. calculer x de x = Dy.

5.7. Résultats

Pour voir l'apport du préconditionneur et son impact sur les performances de l'algorithme, on procède à une comparaison entre GMRES sans préconditionnement et GMRES avec préconditionnement en ce qui concerne le nombre des itérations et le temps de calcul.

Pour cela on lance trois séries de calcul pour différents exemples et on représente dans le premier graphique une comparaison entre GMRES (sans préconditionneur) et PRECGMRES (avec précoonditionneur) en ce qui concerne leur nombre d'itération respective et puis dans le deuxième graphique une comparaison des temps de calcul entre la méthode de décomposition de gausse (précédemment utilisé par le KSP), GMRES et PRECGMRES.

1er exemple

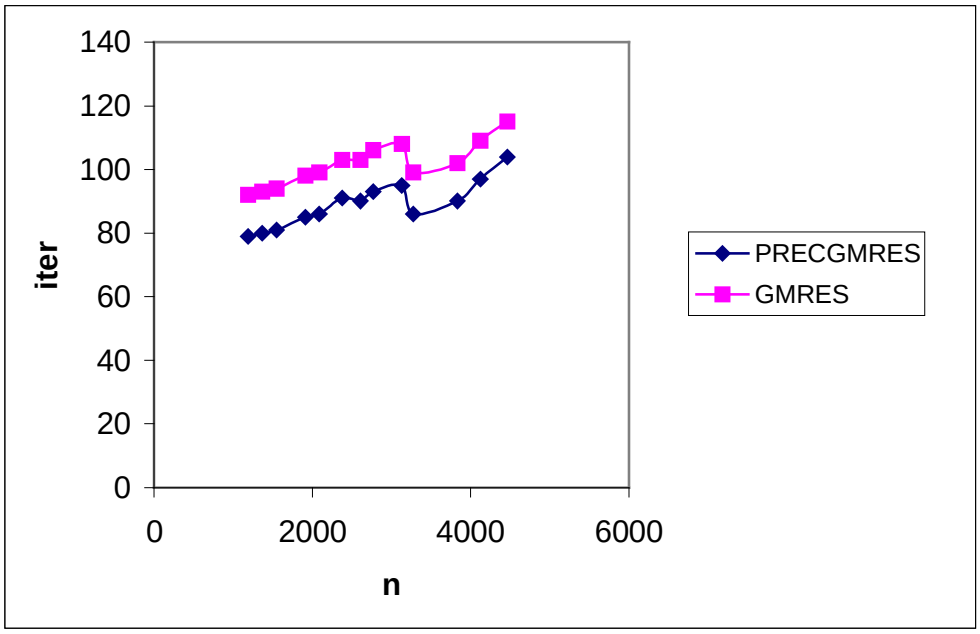


FIG.5.1. comparaison des nombres d'itérations 1er exemple

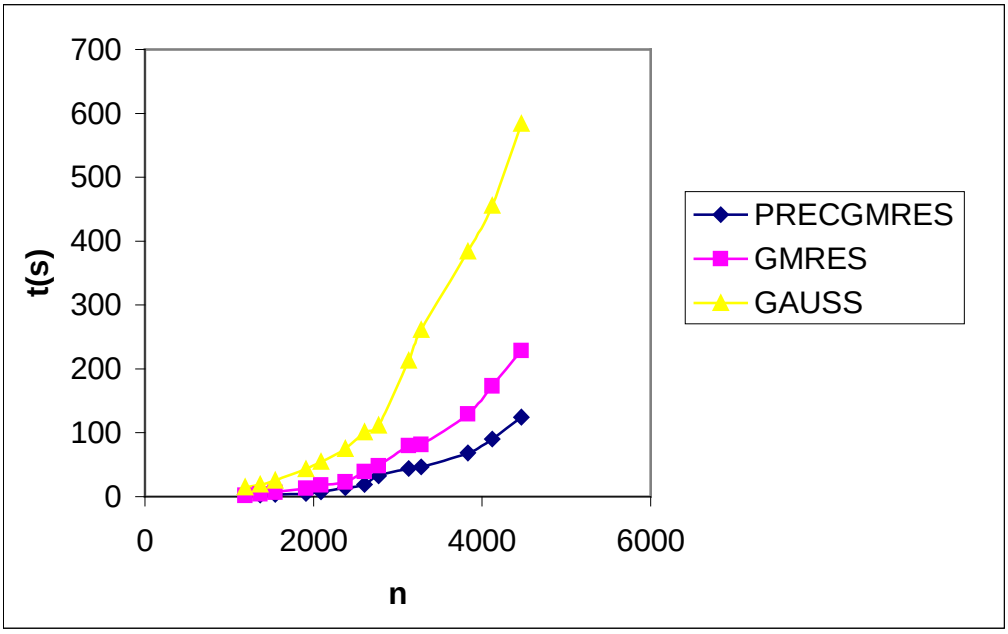
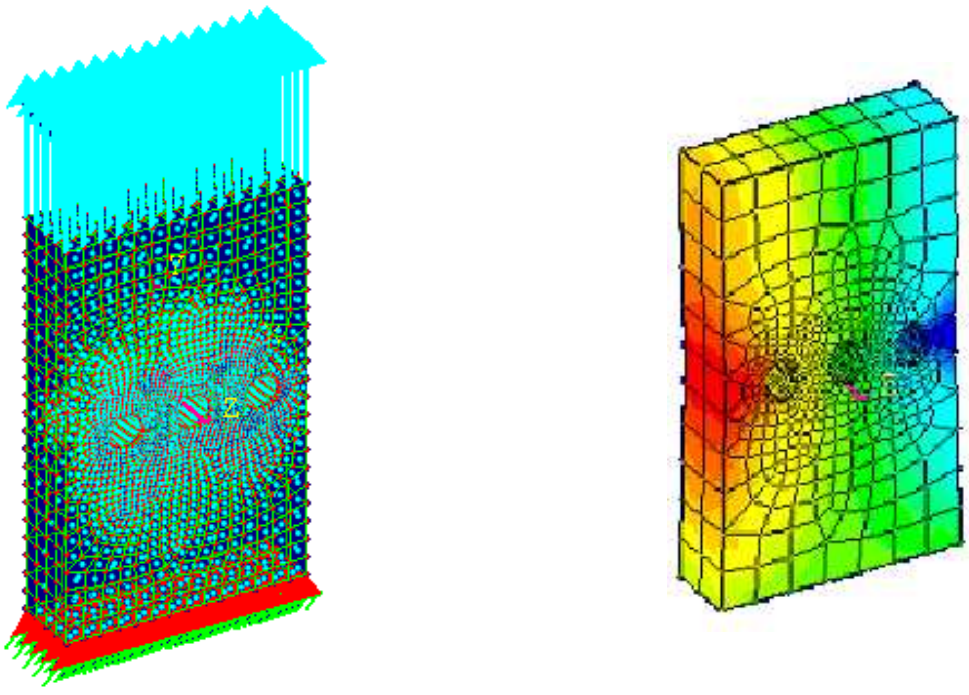


FIG.5.2.comparaison des temps de calculs 1er exemple

2^{em} exemple



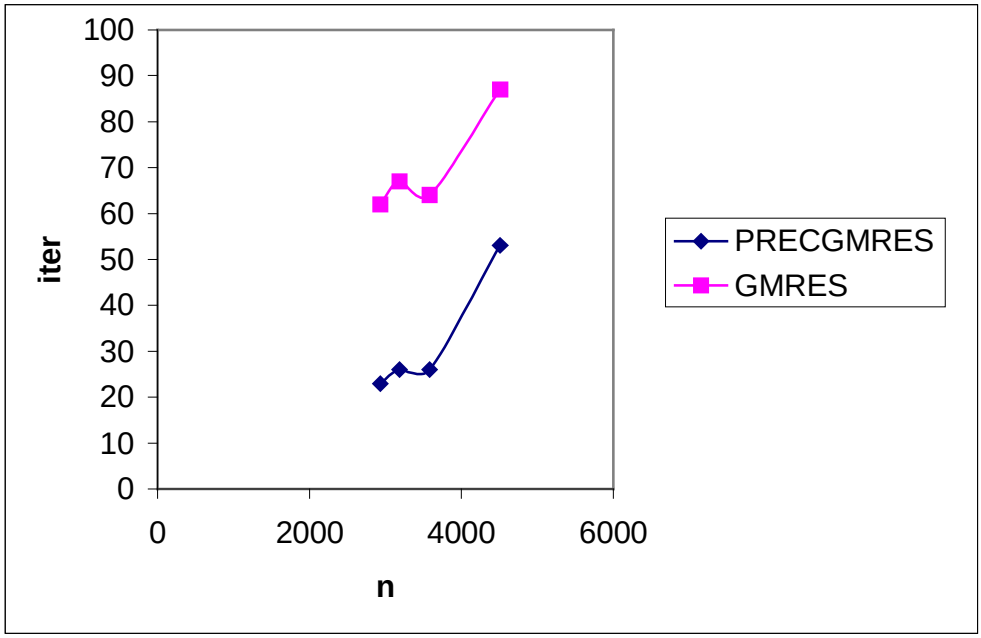


FIG.5.3. comparaison des nombres d'itérations 2^{em} exemple

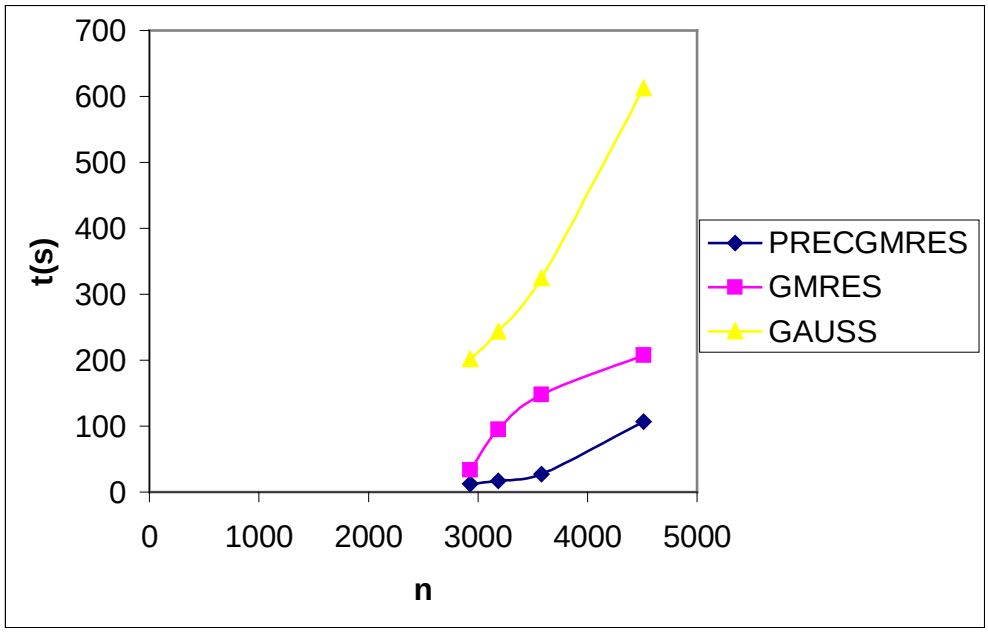
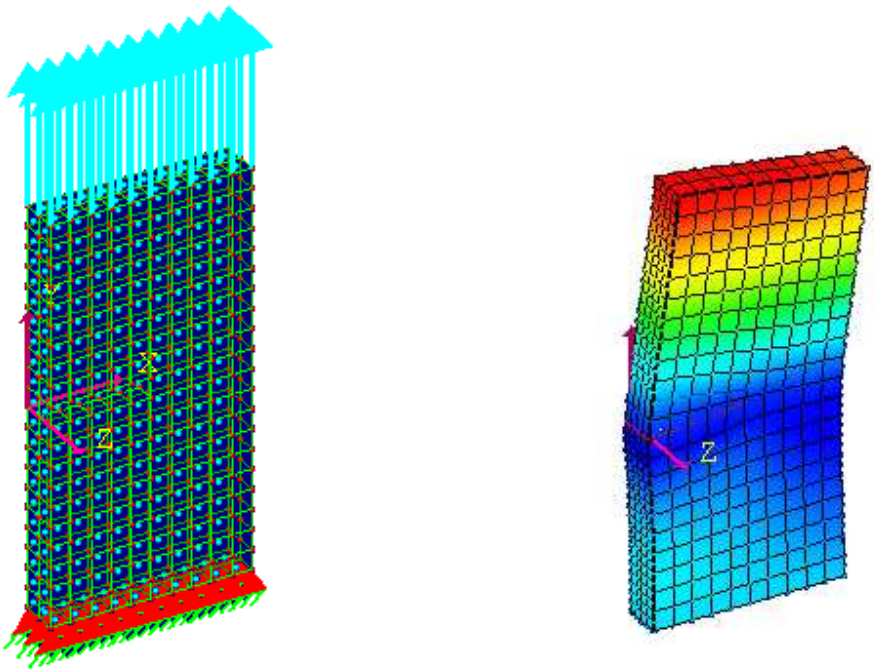


FIG.5.4. comparaison des temps de calculs 2^{em} exemple

3^{em} exemple



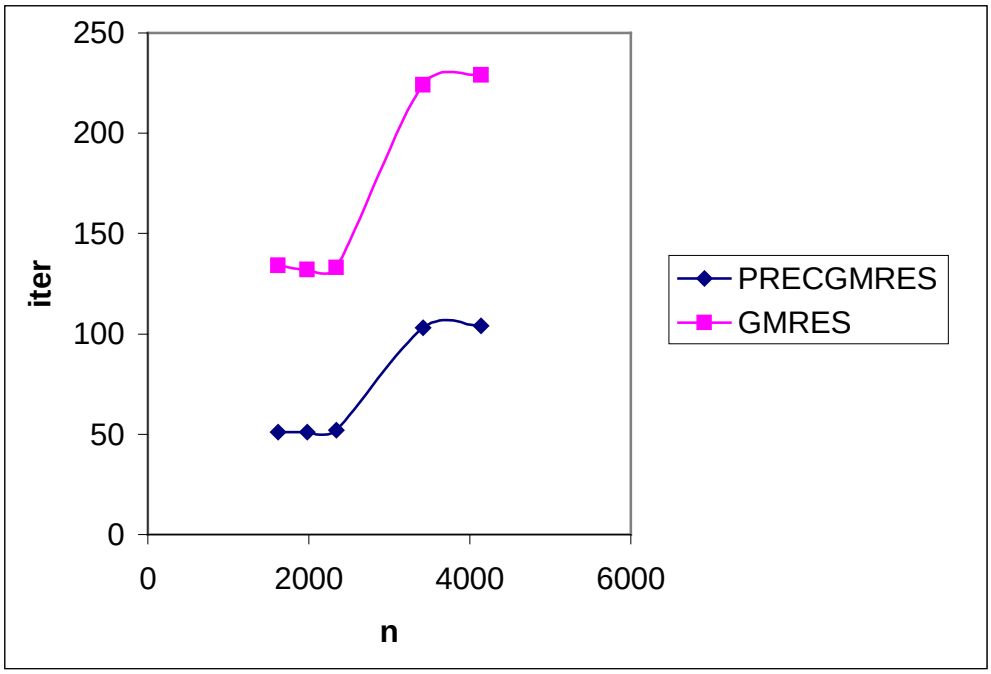


FIG.5.5. comparaison des nombres d'itérations 3^{em} exemple

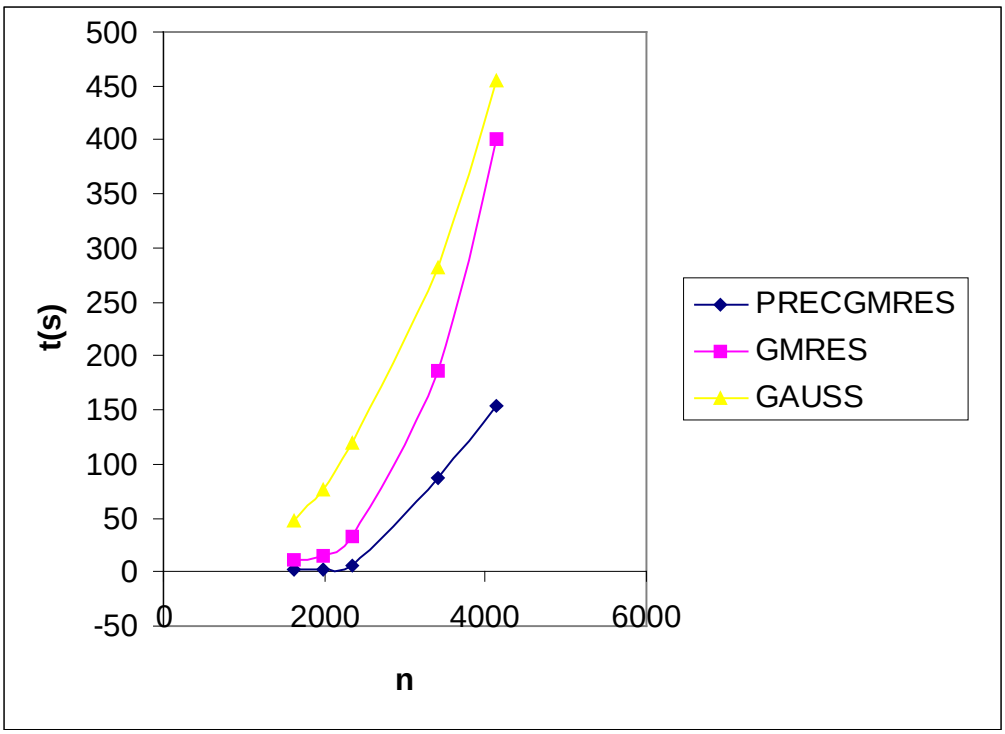


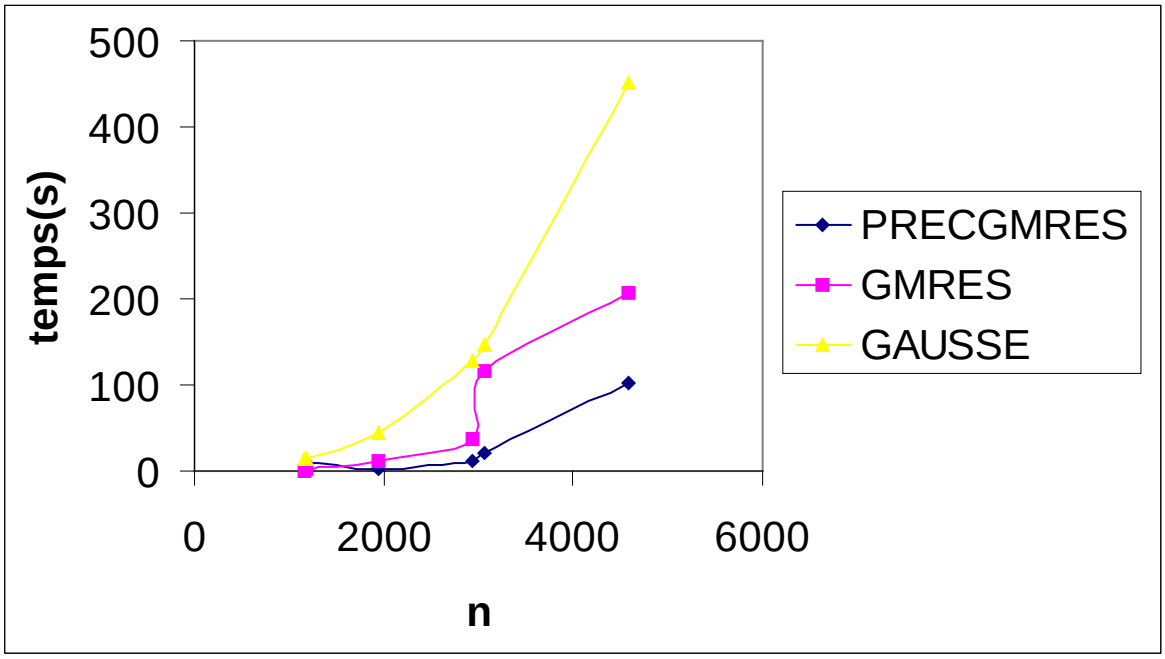
FIG.5.6. comparaison des temps de calculs 3^{em} exemple

On voie bien sur des trois exemples traités que le nombre d'itération est considérablement réduit par le préconditionnement améliorant ainsi la vitesse de GMRES déjà excellente, sauf pour le troisième exemple où GMRES présente des capacités moins satisfaisante, une pénurie bien rattraper par le préconditionnement qui offre de meilleurs performances.

Conclusion

L'objectif du départ de ce travail était d'améliorer les performances de rapidité et de capacité de traitement de problèmes de très grande taille d'un code utilisant la méthode des équations intégrales par l'implémentation et l'adaptation de la méthode Generalized Minimized RESidual (GMRES) pour le type de système d'équations résultant de la modélisation des structures avec une telle méthode.

Le graphe suivant est une comparaison effectuer entre les temps de calcul pour quelques exemples traités en utilisant la décomposition de gauss qui est l'une des méthode de résolution directe et précédemment utilisée par le KSP face au temps de calcul des même exemples en utilisant la GMRES et GMRES préconditionné (PRECGMRES).

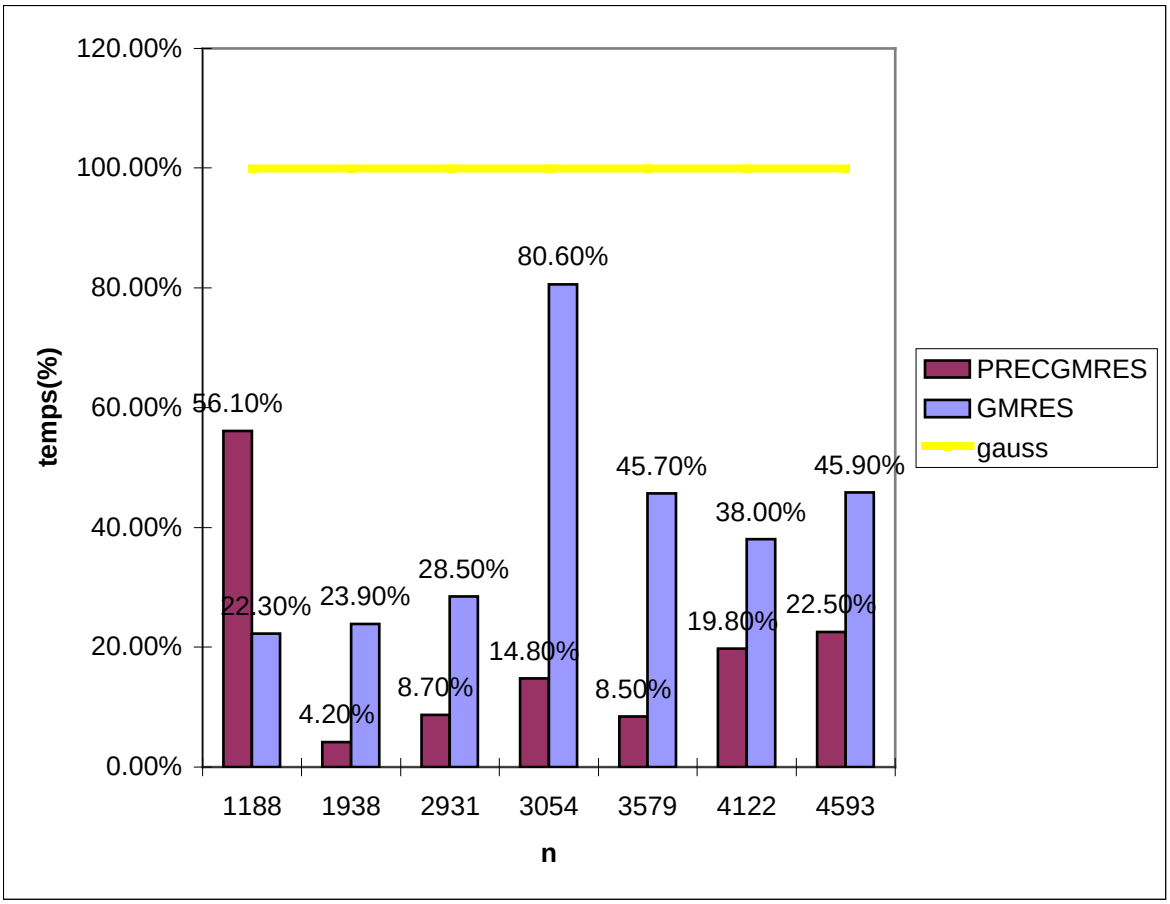


GRAPHE 1 : Comparaison entre gauss, GMRES et GMRES préconditionné

On voit bien que l'écart entre les deux méthodes GMRES et GMRES préconditionné d'une part et gausse d'une autre part augmente de plus en plus que la taille du système à traiter devient importante. L'utilisation d'autres méthodes itératives tel que richardson ou gauss-seidel par exemple n'améliore pas vraiment le temps de calcul comme c'est le cas pour GMRES, et en exigeant une bonne précision des résultats ces méthode deviennent encor plus

lentes que les méthodes directes. Le gain apporté par GMRES devient très important quand le nombre de degré de liberté du problème traité devient grand..

On représente dans le deuxième graphe pour plus de visualisation des résultats, le pourcentage des temps de calculs utilisant GMRES et GMRES préconditionné en prenant le temps d'utilisation de la décomposition de gauss comme référence.



GRAPHE 2 : Comparaison en pourcentage entre gauss,
GMRES et GMRES préconditionné

On voie que l'utilisation de GMRES accélère le calcul qui devient à peu prés entre quatre et deux fois plus rapide que la méthode de décomposition de gauss sauf pour quelques cas de figure (le quatrième cas par exemple) où on n'a pas jusqu'à ce taux d'amélioration. GMRES préconditionné est au minimum cinq fois plus rapide que la méthode de décomposition de gauss, et ça pour tous les cas étudiés, elle présente une stabilité meilleure et des performance beaucoup plus élevés que celles présentées par GMRES sans préconditionnement

Ceci dit, ni GMRES ni GMRES préconditionné est entièrement stable, ce qui est logique pour des méthodes itératives où la convergence ne suit pas une loi particulière et dépend de plusieurs paramètres tel que la forme du système ou la solution initiale.

Ce travail a conduit à la réalisation d'une DLL de résolution très rapide d'un système d'équation de très grande taille issu d'une modélisation par la méthode des équations intégrales où les seules entrées sont la taille du système à résoudre, la matrice du système à résoudre et le vecteur des coefficients à droite des équations, et une seul sortie qui est le vecteur de la solution recherchée.

Comme on a vu l'utilisation d'un préconditionneur améliore considérablement les performances de GMRES. Notre approche présente un comportement optimale pour une très grande plage de problèmes (jusqu'à 10000 ddl la limite de notre machine d'essai) et redémarre l'algorithme le moins possible, mais dans le cadre d'une utilisation pour des problèmes encore plus important le redémarrage de l'algorithme devient alors inévitable et prend beaucoup d'importance. L'introduction d'un deuxième préconditionneur de la deuxième classe sera alors le bien venue pour amortire les pertes causées par le redémarrage et libérer d'avantage de la mémoire

Bibliographie

[1] C.A. BREBBIA. ‘‘The boundary element method’’. Pentech press. 1978.

[2] C.A. BREBBIA et S.WALKER. ‘‘Boundary element techniques in Engineering’’. Newnes Butterworths.1980.

[3] J.C.LACHAT. ‘‘Méthode des équations intégrales sur le contour’’. Techniques de l’ingénieur. A658.

[4] H. KEBIR. ‘‘Approches déterministe et probabiliste de la prévision de la durée de vie de structures aéronautiques à l’aide de la méthode des équations intégrales duales’’. Université de Technologie de Compiègne. 1998.

[5] D. EYHERAMENDY. ‘‘Le paradigme objet : une approche moderne pour gérer la complexité des codes éléments finis en mécanique’’. Université Claude Bernard.2004.

[6] D. CHAPMAN. ‘‘Microsoft Visual C++ 6’’.CampusPress.2004.

[7] A. BOURAS et V. FRAYSEE. ‘‘Inexact matrix-vector products in Krylov methods for solving linear systems: a relaxation strategy’’. SIAM J. Matrix Anal. Appl., 26:660{678, 2005.

[8] V.FRAYSEE, L.GIRAUD, S.GRATTON et J.LANGOU. ‘‘A Set of GMRES Routines for Real and Complex Arithmetics on High Performance Computers’’. Technical Report TR/PA/03/3, CERFACS, Toulouse, France, 2003.

[9] A. BOURAS, V. FRAYSEE et L. GIRAUD. ‘‘A relaxation strategy for inner-outer linear solvers in domain decomposition methods’’. Technical Report TR/PA/00/17, CERFACS, Toulouse, France, 2000.

[10] M. EIERMANNA, O.G. ERNSTA et O. SCHNEIDERB. ‘‘Analysis of acceleration strategies for restarted minimal residual methods’’. Journal of Computational and Applied Mathematics 123 (2000) 261-292

[11] R.J.HANSON et D.R.KINCAID. ‘‘Notes on GMRES Algorithm Organization’’. Technical Report TR-05-05. Department of Computer Sciences, University of Texas at Austin, March 05, 2005.

[12] R.B. MORGAN. ‘‘Restarted block-GMRES with deflation of eigenvalues’’. *Department of Mathematics, Baylor University, Waco, TX 76798-7328, USA.2004.*

[13] Y. SAAD et M. H. SCHULTZ. ‘‘GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems’’. SIAM J. Scienti_c Stat. Comput., 7:856-869, 1986.

[14] J.Y. CHEN. ‘‘Iterative Solution of Large Sparse Nonsymmetric Linear Systems’’. Report CNA–285 ,Ph. D. thesis, Dept. of Mathematics, Center for Numerical Analysis, University of Texas at Austin, 1997.

[15] J-Y. CHEN, D. R. KINCAID et D. M. YOUNG. ‘‘Generalizations and modifications of the GMRES iterative method’’. Numerical Algorithms 21 (1999) 119–146.
.
[16] A. GREENBAUM. ‘‘Iterative Methods for Solving Linear Systems’’. SIAM, Philadelphia, 1997

[17] M. ARIOLI, I. S. DFF et D. RUIZ. ‘‘Stopping criteria for iterative solvers. SIAM J. Matrix’’.Anal. Appl., 13:138-144, January 1992.

[18] D. BINDEL, J. DEMMEL, W. KAHAN et O. MARQUES. ‘‘On computing givens rotations reliably and effciently’’. ACM Transactions on Mathematical Software (TOMS), 28(2):206-238, June 2002.

[19] A. BIJORCK. ‘‘Numerics of Gram-Schmidt orthogonalization’’. Linear Algebra Appl., 197-198:297-316, 1994.

[20] A. BIJORCK. ‘‘Numerical Methods for Least Squares Problems’’. SIAM, Philadelphia, 1996.

[21] J.L. BATOZ et G. DHATT. ‘‘Modélisation des structures par éléments finis-Volume1 :Solides élastiques’’. Edition Hermes, Paris.1990.

[22] Y. HUANG et H. V. D. VORST, ‘‘Some observations on the convergence behavior of GMRES’’. Technical Report 89-09, Faculty of technical mathematics and informatics, Delft, 1989.

[23] L.N. TREFETHEN, N.M. NACHTIGAL et L. REICHEL. ‘‘A hybrid GMRES algorithm for nonsymmetric linear systems’’.*SIAM J. Matrix Anal. Appl.* 13(3) (1992) 796-825.

[24] H.V.DVORST et C. VUIK. ‘‘The superlinear convergence behaviour of GMRES’’. *J. Comput. Appl. Math.* 48 (1993) 327-341.

[25] H. .V.DVORST et C. VUIK. ‘‘GMRES: a family of nested GMRES methods’’. *J. Num. Linear Algebra Appl.*1 (1994) 369-386.

[26] C.L. CALVEZ et B. MOLINA. ‘‘Implicitly restarted and deflated GMRES’’. Numer. Algorithms 21 (1999) 261–285.

[27] T.F. CHAN et W.WAN. ‘‘Analysis of projection methods for solving linear systems with multiple right-hand sides’’. SIAM J. Sci.Comput. 18 (1997) 1698–1721.

[28] A. CHAPMAN et Y. SAAD. ‘‘Deflated and augmented Krylov subspace techniques’’. Numer. Linear Algebra Appl. 4 (1997) 43–66.

[29] J. CULLUM et W.E. DONATH. ‘‘A block Lanczos algorithm for computing the *q* algebraically largest eigenvalues and a corresponding eigenspace of large sparse real symmetric matrices’’. in: Proc. of the 1974 IEEE Conf. on Decision and Control, Phoenix, Arizona, 1974, pp. 505–509.

[30] E.D. STURLER. ‘‘Truncation strategies for optimal Krylov subspace methods’’. SIAM J. Numer. Anal. 36 (1999) 864–889.

[31] T.A MANTEUEL et J.S. OTTO. ‘‘On the roots of the orthogonal polynomials and residual polynomials associated with a conjugate gradient method’’. Numer. Linear Algebra Appl. 1 (5) (1994) 449{475.

[32] L.N. TREFETHEN et N.M. NACHTIGAL. ‘‘How fast are non-symmetric matrix computations?’’. *SIAM J.Matrix Anal. Appl.* 13(3) (1992) 778 795.