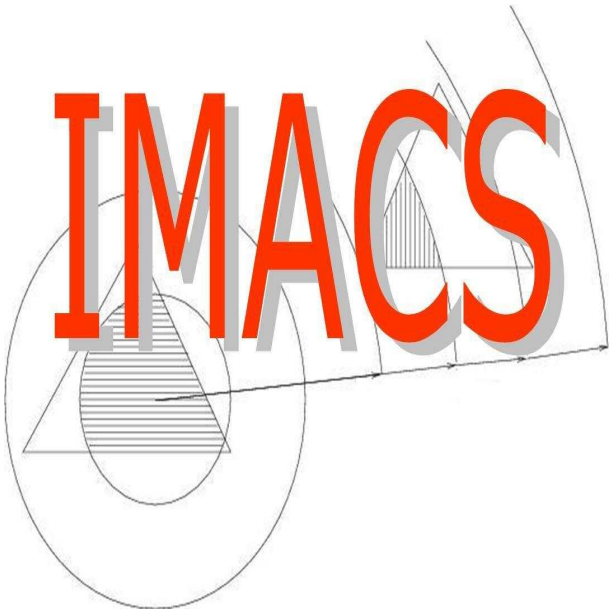

Solveurs Directs Rapides Pour Les Equations Intégrales

Kieran DELAMOTTE

IMACS - Ingénierie MAthématique et Calcul Scientifique :
M. Touffic ABBOUD & M. François BEREUX

Contexte du stage

Ce présent rapport est la synthèse de mon stage long en entreprise, nécessaire à la validation du diplôme d'ingénieur en mathématiques appliquées et calcul scientifique (MACS) de l'école d'ingénieurs,SuP Galilée, de l'université Paris XIII. Ce stage sanctionne également l'obtention du Master 2 option Maths/Info de l'université Paris XIII poursuivi en parallèle de ma troisième année.
J'ai réalisé mon stage au sein de la société IMACS (Ingénierie MAthématique et Calcul Scientifique) sur le site de l'Ecole Polytechnique à Palaiseau entre mars 2010 et septembre 2010, sous la direction de MM Abboud Toufic et Béreux François.

Table des matières

Première partie

Introduction

Sujet du stage

Les équations intégrales permettent de résoudre des équations de la physique comme les équations de Maxwell, approche de plus en plus utilisée dans l'industrie. Leur approximation par les éléments finis conduit à la résolution de systèmes linéaires pleins de taille N, pour un coût de résolution par méthode directe en $\mathcal{O}(N^3)$ qui devient prohibitif pour N assez grand. Une première révolution (fin des années 1980) a consisté à accélérer le produit matrice-vecteur en $\mathcal{O}(N \log N)$ au lieu de $\mathcal{O}(N^2)$, en exploitant le développement multipolaire du noyau de Green (Fast Multipole Method). Ceci suppose l'utilisation d'une méthode itérative pour la résolution du système linéaire qui pose problème dans le cas de systèmes très mal conditionnés ou avec un grand nombre de seconds membres. Une nouvelle révolution s'annonce dans ce domaine avec des méthodes de factorisation approchées rapides exploitant le faible rang des blocs de matrices "extra-diagonaux". Le stage a pour objet de faire le point sur la littérature, d'étendre la méthode aux équations de première espèce et de maquetter une première implémentation.

Résultats à propos des équations intégrales

On expose ici un exemple de problème intégral issu de [8],de sa formulation jusqu'à l'obtention d'une équation intégrale. On présente brièvement des schémas de discrétisation possible pour la résolution de l'équation intégrale.

Exemple d'obtention d'une équation intégrale

On considère le problème suivant,issu d'un problème d'électrostatique moléculaire, sous forme d'une équation aux dérivées partielles

$$-\nabla.(\epsilon(x)\nabla\Phi(x)) \;=\; \sum_{j=1}^{K} q_j \delta(x - x_j) \text{ dans } \mathbb{R}^3, \qquad (1)$$

où $\epsilon(x)$ vaut ϵ_{in} dans toute la molécule jusqu'à sa frontière Γ et ϵ_{out} à l'extérieur. On note par Ω_{in} et Ω_{out} les régions de l'espace composées par l'intérieur et l'extérieur de la molécule. On suppose que toutes les sources sont localisées à l'intérieur de la molécule. Afin de se ramener à une formulation intégrale, on choisit de représenter la fonction Φ comme étant la somme d'une fonction harmonique Φ^{pol} vérifiant l'équation de Poisson dans chaque domaine ainsi que d'une fonction Φ^{source} due aux sources,

$$\Phi = \Phi^{pol} + \Phi^{source},$$

où classiquement,

$$\Phi^{source} = \sum_{j=1}^{K} \frac{q_j}{4\pi\epsilon_{in}} \frac{1}{\|x - x_j\|}.$$

On note que Φ^{source} vérifie l'équation de Poisson et est continue sur la frontière Γ. On renvoie à [1] pour l'établissement de cette propriété. Si l'on impose que le potentiel Φ et le flux $\epsilon\frac{\partial\Phi}{\partial\nu}$ soient continues à la frontière Γ. Ainsi, on peut aisément montrer que Φ^{pol} satisfait à

$$\begin{aligned}
\nabla^2\Phi^{pol} &= 0 \text{ dans } \Omega_{in}, \Omega_{out}, & (2)\\
\left[\Phi^{pol}\right] &= 0 \text{ sur } \Gamma, & (3)\\
\left[\epsilon\frac{\partial\Phi^{pol}}{\partial\nu}\right] &= -\left[\epsilon\frac{\partial\Phi^{source}}{\partial\nu}\right] \text{ sur } \Gamma. & (4)
\end{aligned}$$

où $[f]$ désigne le saut de la fonction f à travers Γ.
On renvoie à [1] pour la démonstration théorique de l'expression de la solution Φ^{pol} sous forme d'un potentiel de simple couche dû à une distribution de charge σ définie sur la surface Γ

$$\Phi^{pol}(x) = \frac{1}{4\pi}\int_\Gamma \frac{\sigma(y)}{\|x - y\|}dy. \quad (5)$$

D'après les résultats sur le potentiel de simple couche (5), il est continu à travers Γ et satisfait les conditions de saut suivantes

$$\begin{aligned}
\frac{\partial\Phi^{pol}}{\partial\nu_-}(y_0) = \lim_{x\to y, x\in\Omega_{in}} \frac{\partial\Phi^{pol}}{\partial\nu_0}(x) &= \frac{1}{2}\sigma(y_0) + \int_\Gamma \frac{\partial G}{\partial\nu_0}(y_0, y)\sigma(y)dy, & (6)\\
\frac{\partial\Phi^{pol}}{\partial\nu_+}(y_0) = \lim_{x\to y, x\in\Omega_{out}} \frac{\partial\Phi^{pol}}{\partial\nu_0}(x) &= -\frac{1}{2}\sigma(y_0) + \int_\Gamma \frac{\partial G}{\partial\nu_0}(y_0, y)\sigma(y)dy, & (7)
\end{aligned}$$

où y_0 est un point de Γ, $\frac{\partial\Phi^{pol}}{\partial\nu_0}(x)$ désigne la dérivée normale extérieure en y_0 et $G(x, y)$ désigne le noyau de Green $\frac{1}{4\pi\|x-y\|}$. On note alors

$$\mathbf{K}\sigma(y_0) = \int_\Gamma \frac{\partial G}{\partial\nu_0}(y_0, y)\sigma(y)dy.$$

La représentation (5) induit que l'équation (2) est automatiquement vérifiée. La continuité du potentiel de simple couche induit que l'équation (3) est vérifiée également. On impose alors la condition (3) et en utilisant les conditions de saut, on obtient l'équation suivante en σ

$$\frac{1}{2}\sigma + \lambda \mathbf{K}\sigma = -\lambda \frac{\partial \Phi^{source}}{\partial \nu}, \quad (8)$$

où $\lambda = \frac{\epsilon_{in} - \epsilon_{out}}{\epsilon_{in} + \epsilon_{out}}$. L'équation (8) est appelée une équation intégrale et doit être discrétisée afin d'être résolue numériquement.

Schéma numérique pour la discrétisation d'une équation intégrale

Considérons le problème intégral de simple couche

$$\mathcal{S}\lambda = p_0. \quad (9)$$

L'approche numérique que l'on privilégiera est celle des éléments finis de frontière (désignés aussi par l'acronyme anglais *BEM* pour Boundary Element Method) développée dans [1]. Au lieu de discrétiser l'EDP, on discrétise l'équation intégrale (9). On part de la formulation variationnelle

$$\begin{gathered} \text{trouver } \lambda \in V \text{ tel que} \\ s(\lambda, \lambda^t) = \mathcal{P}_0(\lambda^t) \ \forall \lambda^t \in V, \end{gathered}$$

avec $V = H^{-\frac{1}{2}}(\Gamma)$,

$$s(\lambda, \lambda^t) = {}_{+\frac{1}{2}}(\mathcal{S}\lambda, \lambda^t)_{-\frac{1}{2}},$$

et

$$\mathcal{P}_0(\lambda^t) = {}_{+\frac{1}{2}}(p_0, \lambda^t)_{-\frac{1}{2}}.$$

L'approximation par éléments finis de frontière consiste d'abord à approcher la surface Γ par une surface Γ_h en triangle par exemple. Pour avoir des résultats raisonnables, il convient d'utiliser des éléments dont la taille h est plus petite que la longueur d'onde, typiquement h est plus petite qu'un cinquième de la longueur d'onde pour des champs lointain. On approche alors l'espace fonctionnel V par l'espace

$$V_h^0 = \{\lambda_h = \lambda_i \text{ sur le triangle } T_i, T_i \in \Gamma_h\}.$$

Cette approximation est dite P^0. Cette approximation est conforme : $V_h \subset V$ et lorsque l'on fait tendre h vers 0, on approche de mieux en mieux l'espace fonctionnel V. Notons N le nombre de triangles formant la discrétisation de

Γ_h, V_h^0 est alors de dimension N. et on dispose d'une base simple $\phi_i, i = 1, \ldots, N$, avec

$$\phi_i(x) \quad = \quad \frac{1}{\text{aire}(T_i)}.\mathbf{1}_{T_i}.$$

On peut alors décomposer λ_h dans cette base, en notant ses coordonnées par α_i^h,

$$\lambda_h(x) \quad = \quad \sum_{i=1}^{N} \alpha_i^h \phi_i(x).$$

Le problème discret est donc équivalent à

$$\text{trouver } \lambda_h \in V_h \text{ tel que}$$
$$s_h(\lambda_h, \lambda_h^t) = \mathcal{P}_{0,h}(\lambda_h^t) \ \forall \lambda_h^t \in V_h.$$

Ce problème discret revient *in fine* à résoudre le système linéaire plein symétrique,

$$S^h \alpha^h \quad = \quad P_0^h,$$

dont les coefficients sont des intégrales doubles, définis par

$$S_{i,j}^h \quad = \quad \frac{1}{\text{aire}(T_i)} \frac{1}{\text{aire}(T_j)} \int_{T_i} \int_{T_j} G(x,y) dT_j(y) dT_i(x).$$

Cette méthode est connue pour sa grande précision dans la pratique, et est massivement utilisée dans l'industrie. On note cependant que l'étape d'initialisation s'avère coûteuse avec le calcul des intégrales doubles. Une autre méthode, dite de collocation a aussi été populaire dans l'industrie, simple à programmer au détriment d'un besoin de sur-maillage.

$$S_{i,j}^h \quad = \quad \frac{1}{\text{aire}(T_i)} \frac{1}{\text{aire}(T_j)} \int_{T_i} \int_{T_j} G(x,y) dT_j(y) dT_i(x). \tag{10}$$

Résolution du système linéaire

Le système (10) est de grande taille et a le défaut, comparé à une méthode de FEM usuelle, d'être plein. La méthode FMM initialement développée par Greengard et Rokhlin dans [9], consiste à accélérer le produit matrice-vecteur à partir du développement multipolaire du noyau G ce qui permet de

regrouper des sources voisines et de les considérer comme une seule et même source. Ainsi, les coefficients correspondants n'ont pas besoin d'être stocké de manière explicite réduisant ainsi l'espace mémoire nécessaire. L'utilisation de manière hiérarchique ou multi-niveaux de ce procédé permet de réduire la complexité du produit matrice-vecteur de $\mathcal{O}(N^2)$ à $\mathcal{O}(N \log N)$ ce qui en fait un algorithme particulièrement performant. On voit que le fait de posséder une méthode rapide pour effectuer le produit matrice-vecteur nous oriente vers des solveurs itératifs lesquels font appel à de nombreux produits de ce type. Cependant, cet algorithme possède néanmoins des défauts et présente certaines difficultés lorsque l'on veut résoudre un système linéaire avec plusieurs second membres ou lorsque la matrice est mal conditionné. On note de plus un *breakdown* à basse fréquence qui le rend moins performant. Le développement ultérieur a pour but de présenter une approche différente amenant à une résolution directe d'un système linéaire "proche" de celui d'origine.

Quelques problèmes liés à la résolution de systèmes linéaires pleins de grandes tailles

Le théorème de représentation intégrale permet l'établissement d'une équation intégrale que l'on discrétise par un schéma numérique. Le système linéaire correspondant ne possède en général pas de structure particulière hormis la symétrie et a les principaux défaut d'être plein et de grande taille. En effet, une discrétisation typique en fonction de la longueur d'onde λ peut conduire à de grands systèmes suivant l'odre de grandeur de cette dernière. On peut dénombrer trois problèmes majeurs pour ces matrices pleines de grandes tailles :

1. la construction de la matrice $A = (a_{ij})_{ij}$ de discrétisation,

$$A \in \mathbb{C}^{m \times n}, n, m \in \mathbb{N}$$

   peut être coûteuse en ce sens où si la dimension de la matrice est grande (typiquement 10^4, 10^5 ou plus), alors la construction nécessite $\mathcal{O}(mn)$ opérations arithmétiques.

2. Le stockage de la matrice nécessite $\mathcal{O}(nm)$ unités de mémoire. En double précision, un réel est représenté par 8 bits. Pour une matrice de taille $10^5 \times 10^5$, on utilise alors 8.10^{10} bits,soit environ 9 Go. La matrice ne peut être contenue en mémoire sur un PC standard et on

doit la stocker sur disque.

3. Le troisième problème est lié à la résolution du système linéaire correspondant. La solution directe pour une matrice pleine est en $\mathcal{O}(N^3)$ opérations pour une matrice carrée de dimension N. Ceci ne peut être acceptable pour de grands problèmes.

Structure du rapport

Ce document est structuré de la façon suivante.

- La deuxième partie fait le point bibliographique sur différentes méthodes de résolution directes étudiées durant ce stage :

  le chapitre 1 est consacré à un survol rapide du solveur proposé par Greengard *et al.*,

  le chapitre 2 consiste à résumer la méthodologie employée par Shaeffer pour construire une factorisation de type LU à partir de l'utilisation de l'ACA. C'est cette méthode qui sera implémentée par la suite.
- La troisième partie de ce rapport pose le cadre théorique pour construire une factorisation de type LU mono-niveau (*cf* article de Shaeffer) en détaillant les points suivant :

  le chapitre 3 rappelle des résultats utiles d'algèbre linéaire et définit la notion de matrice de rang faible,

  le chapitre 4 constitue le coeur du rapport en ce sens où l'on y expose de manière théorique l'algorithme de compression ACA,

  le chapitre 5 contient les résultats théoriques pour la manipulation de matrices compressées,

  le chapitre 6 est dédié à l'algorithme de renumérotation des degrés de liberté,

  enfin, le chapitre 7 est consacré à une version compressée de l'algorithme de Crout.
- La quatrième partie présente les applications numériques de la méthode mono-niveau de Shaeffer.
- La cinquième et dernière partie est notre conclusion où l'on dresse un bilan de la méthode implémentée ainsi que des améliorations possibles du code.

Deuxième partie

*É*tude biliographique

Chapitre 1

Compression LSR & Solveur de Greengard

1.1 Introduction

On présente dans ce chapitre une courte description de la méthode développée par Greengard *et al.* dans l'article [8] en employant la méthode de décomposition des blocs matriciels de rangs faibles décrite dans [6]. On développera par la suite toute une théorie sur les matrices de rangs faibles lorsque l'on examinera la méthode employée par [13]. Pour l'instant, on considérera qu'une matrice est de rang faible si son rang est très inférieur à ses dimensions. Cette construction d'un solveur direct et rapide pour un système linéaire provenant d'équations intégrales est basée sur la notion de rang faible d'une matrice ainsi que sur la possibilité de construire numériquement une factorisation avantageuse d'une telle matrice, ici sous la forme

$$A = LSR. \tag{1.1}$$

On introduit dans un premier temps les théorèmes nécessaires à la construction d'un algorithme fournissant la décomposition (1.1) à la *section 2* d'après [6]. La *section 3* décrit l'utilisation de cette factorisation pour former un solveur direct et rapide selon la méthode décrite dans [8].

Notations

Dans la suite de ce chapitre, on adoptera les notations suivantes. Les lettres majuscules désignent exclusivement des matrices tandis que les minuscules désignent les vecteurs ou les scalaires. On se réserve les lettres Q et U pour

des matrices ayant des colonnes orthonormales, P (éventuellement indicée) pour les matrices de permutation. La base canonique de $\mathbb{C}^n$ est noté $(e_j)_j$. *É*tant donnée une matrice X, X^* désigne son adjoint(transposée du complexe conjugué), $\sigma_k(X)$ représente la k^{eme} valeur singulière, $\|X\|_2$ la norme l^2 et $\|X\|_F$ la norme de Frobenius.

1.2 Décomposition $A = LSR$

1.2.1 Résultats sur l'existence de la décomposition LSR

On obtient la décomposition (1.1) annoncée à l'aide des deux théorèmes suivants. Le premier assure l'existence et la possibilité de construire les matrices L, S et R. Le second montre qu'on peut le faire en un temps raisonnable.

Théorème 1.2.1 (Gu-Eisenstat)**.** *Soit A une matrice de taille $m \times n$, posons $l = min(m, n)$ et soit k un entier tel que $1 \leq k \leq l$. Alors il existe une factorisation*

$$AP = QR \tag{1.2}$$

où P est une matrice de permutation de taille $n \times n$, Q est une matrice ayant ses colonnes orthonormales de taille $m \times l$ et R est une matrice de taille $l \times n$ triangulaire supérieure. En décomposant Q et R de la sorte :

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \quad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

où Q_{11} et R_{11} sont de taille $k \times k$, Q_{21} de taille $(m-k) \times k$, Q_{12} de taille $k \times (l-k)$, Q_{22} de taille $(m-k) \times (l-k)$, R_{12} de taille $k \times (n-k)$ et R_{22} de taille $(l-k) \times (n-k)$, on obtient les inégalités suivantes

$$\sigma_k(R_{11}) \geq \frac{1}{\sqrt{1+k(n-k)}}\sigma_k(A), \tag{1.3}$$

$$\sigma_1(R_{22}) \leq \sqrt{1+k(n-k)}\sigma_{k+1}(A), \tag{1.4}$$

$$\|R_{11}^{-1}R_{12}\|_F \leq \sqrt{k(n-k)} \tag{1.5}$$

L'approximation de A par une matrice de rang k conduit naturellement à considérer que $\sigma_{k+1}(A) = \epsilon$ avec ϵ suffisamment petit.

Théorème 1.2.2 (Gu-Eisenstat)**.** *Étant donnée une matrice A de taille $m \times n$ décomposée sous la forme (1.2) qui au lieu des conditions (1.3),(1.4) et (1.5) vérifie les conditions suivantes*

$$\begin{aligned} \sigma_k(R_{11}) &\geq \frac{1}{\sqrt{1+nk(n-k)}}\sigma_k(A), \\ \sigma_1(R_{22}) &\leq \sqrt{1+nk(n-k)}\sigma_{k+1}(A), \\ \|R_{11}^{-1}\|_F &\leq \sqrt{nk(n-k)} \end{aligned}$$

peut être calculée en $\mathcal{O}(mn^2)$ opérations.

Remarque 1.2.1. *Il s'agit là d'une borne pessimiste. Typiquement on obtient un nombre d'opérations similaire au nombre d'opérations requis pour effectuer un processus de Gram-Schmidt pivoté, $\mathcal{O}(mnk)$.*

Théorème de décomposition

Le théorème suivant sert de base à la décomposition (1.1) utilisée pour le solveur de Greengard. Il fournit une décomposition d'une matrice A de rang faible k sous forme d'une matrice extraite (le "squelette" de A) constituée de k lignes et colonnes de cette dernière. Pour ce faire, on utilise successivement le théorème 1 sur les colonnes puis les lignes de A.

Théorème 1.2.3. *Soient A une matrice de taille $m \times n$ et k un entier tel que $1 \leq k \leq \min(m,n)$. Alors il existe une factorisation*

$$A = P_L \begin{bmatrix} I \\ S \end{bmatrix} A_S[I|T]P_R^* + X \tag{1.6}$$

où $I \in \mathbb{C}^{k\times k}$ est la matrice de l'identité, P_L et P_R sont des matrices de permutation et A_S est la sous-matrice principale de taille k de la matrice $P_L^*.A.P_R$. De plus les matrices S,T et X vérifient les inégalités suivantes

$$\|S\|_F \leq \sqrt{k(m-k)} \qquad \|T\|_F \leq \sqrt{k(n-k)}$$

et

$$\|X\|_2 \leq \sigma_{k+1}(A)\sqrt{1+k(\min(m,n)-k)}$$

Preuve 1. *On démontre l'existence de la décomposition (1.6) en appliquant le théorème 1 d'abord sur les colonnes de la matrice A afin de prouver l'existence de k colonnes formant une base bien conditionnée de l'espace des colonnes, regroupées dans la matrice A_{CS}("Column Skeleton"). On applique*

alors à nouveau ce théorème sur la transposée de la matrice extraite construite A_{CS} afin de trouver k lignes formant une base de l'espace des lignes. La matrice finalement construite est alors la matrice A_S ("Skeleton"). Les inégalités s'obtiennent quant à elles à l'aide des définitions des normes considérées.

1.2.2 Algorithme de compresssion

Méthode générale

L'algorithme proposé par Cheng *et al.* se base sur l'emploi de la procédure d'orthogonalisation de Gram-Schmidt avec pivotage ainsi que la résolution d'une équation matricielle. L'algorithme se base sur la décomposition (10) en négligeant le terme X qui est majoré par la $(k+1)$-ème valeur singulière de A qui est supposée être suffisamment petite. Ainsi, il ne reste que la décomposition mentionné dans l'article [8] aux sections 3 et 5.

Étant donnés une matrice A de $\mathbb{C}^{m\times n}$ et un réel positif ϵ, l'algorithme se déroule en quatre étapes, les deux premières opérant sur les colonnes de la matrice et les deux secondes sur les lignes.

$\underline{1^{ere}étape}$

La première étape consiste à appliquer le processus de Gram-Schmidt avec pivotage(GSP) avec la précision ϵ. À ce stade, on obtient la factorisation QR suivante :

$$AP_R = Q[R_{11}|R_{12}]$$

où $P_R \in \mathbb{C}^{n\times n}$ est une matrice de permutation, $Q \in \mathbb{C}^{m\times k}$ ayant des colonnes orthonormales, $R_{11} \in \mathbb{C}^{k\times k}$ est triangulaire supérieure et $R_{12} \in \mathbb{C}^{k\times(n-k)}$.

Remarque 1.2.2. *L'entier k correspondant au rang numérique de la matrice A initiale n'est pas un argument de l'algorithme mais retourné par l'algorithme de GSP.*

$\underline{2^{eme}étape}$

On résout le sytème linéaire avec plusieurs second membres suivante à la précision ϵ avec $T \in \mathbb{C}^{k\times(n-k)}$:

$$R_{11}T = R_{12}$$

Dans le cas où la matrice R_{11} est mal conditionnée, on est face à des solutions multiples. On choisira alors celle de norme minimale au sens de la norme de

Frobenius. En notant A_{CS} la matrice constituée des k premières colonnes de la matrice AP_R,on peut décomposer la matrice A sous la forme suivante

$$A = A_{CS}[I|T]P_R^*$$

Preuve 2.

$$\begin{aligned} A.P_R &= Q[R_{11}|R_{12}] \\ &= [QR_{11}|QR_{12}] \\ &= [A_{CS}|QR_{11}R_{11}^{-1}R_{12}] \\ &= [A_{CS}|A_{CS}T] \\ &= A_{CS}[I|T] \end{aligned}$$

Les deux dernieres étapes sont analogues à celles que l'on a détaillées ici à la différence que l'on opère sur les lignes de la matrice A_{CS}. L'application de l'algorithme de GSP aux lignes de la matrice A_{CS} fournit la décomposition suivante

$$P_L^* A_{CS} = \left[\frac{S_{11}}{S_{12}}\right] U$$

On résout alors une équation matricielle similaire à la deuxième étape,

$$SS_{11} = S_{12}$$

Cela conduit alors, avec les mêmes idées que dans la preuve précédente à la décomposition suivante, en notant A_S la matrice constituée des k premières lignes de $P_L^* A_{CS}$,

$$A_{CS} = P_L \left[\frac{I}{S}\right] A_S.$$

On obtient alors la décomposition finale donnée par la relation (? ?) en remplaçant A_{CS} par sa décomposition donnée par la formule (? ?) dans l'expression (? ?). On a alors :

$$A = P_L[\frac{I}{S}]A_S[I|T]P_R^*$$

Remarque 1.2.3. *La précision de la méthode est directement liée à la précision utilisée pour effectuer la méthode de Gram-Schmidt avec pivotage et c'est donc celle-ci qui influencera le plus l'erreur en bout de compression. Le temps de calcul est lui plus proche de celui d'une méthode de factorisation QR que d'une SVD.*

Gram-Schmidt avec pivotage

L'idée de cette version de l'algorithme de Gram-Schmidt est de trier les colonnes de la matrice à orthogonaliser par norme décroissante. Ainsi, on orthogonalise en priorité les colonnes de normes importantes lesquelles apporteront la plus grande contribution. L' algorithme est basé sur la méthode de Gram-Schmidt modifiée(MGS).

Résolution de l'équation matricielle

La deuxième étape consiste en la résolution d'un système linéaire avec plusieurs second membres, ces derniers étant les colonnes de la matrice du terme de droite dans les équations matricielles (14) et (17).

1.3 Solveur de Greengard

Le développement suivant montre comment intervient la décomposition (1.1) au sein de la construction d'un solveur direct. La méthode consiste à effectuer une décomposition de Schur (*cf* [7] pour de plus amples détails à propos de la factorisation de Schur) sur la matrice de discrétisation du problème.

1.3.1 Réécriture du problème.

On considère un système d'équations plein $Ax = b$ dont les blocs extra-diagonaux sont de rangs faibles. On représente ce système par blocs comme suit

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \ldots + A_{1N}x_N &= b_1, \\ A_{21}x_1 + A_{22}x_2 + \ldots + A_{2N}x_N &= b_2, \\ &\vdots \\ A_{N1}x_1 + A_{N2}x_2 + \ldots + A_{NN}x_N &= b_N, \end{aligned}$$

où x_i,$b_i \in \mathbb{R}^{n_i}$ et $A_{ij} \in \mathbb{R}^{n_i \times n_j}$. On note que la solution de ce système linéaire est en $\mathcal{O}(N_{tot}^3)$ avec $N_{tot} = \sum_{i=1}^{N} n_i$.
On se sert ici de la décomposition vue à la section précédente (*cf*[6]). On suppose que les blocs extra-diagonaux peuvent être décomposés (à l'aide de l'algorithme de Cheng) sous la forme

$$A_{ij} = L_i S_{ij} R_j \tag{1.7}$$

où $L_i \in \mathbb{R}^{n_i \times k_i}$, $S_{ij} \in \mathbb{R}^{k_i \times k_j}$, et $R_j \in \mathbb{R}^{k_j \times n_j}$ avec $k_i << n_i$ et $k_j << n_j$. On remarque que la factorisation (1.7) précédente est un peu "spéciale" en ce sens où tous les blocs d'une même ligne ont une matrice de permutation L commune et de même pour les colonnes avec R (L ne dépend que de " i" , R de "j"). Ceci est expliqué plus loin et aussi dans [8] et [6].

On introduit les variables auxiliaires y,

$$y_j = R_j x_j$$

et on réécrit le système en fonction de ces nouvelles inconnues afin de pouvoir utiliser le complément de Schur :

$$\begin{aligned} A_{11}x_1 + L_1S_{12}y_2 + \ldots + L_1S_{1N}y_N &= b_1, \\ L_2S_{21}y_1 + A_{22}x_2 + \ldots + L_2S_{2N}x_N &= b_2, \\ &\vdots \\ L_NS_{N1}x_1 + L_NS_{N2}x_2 + \ldots + A_{NN}x_N &= b_N, \end{aligned}$$

Sous forme de matrice blocs, on a de manière équivalente

$$\left[\begin{array}{cccc|cccc} A_{11} & 0 & \ldots & 0 & 0 & L_1S12 & \ldots & L_1S_{1N} \\ 0 & A_{22} & \ldots & 0 & L_2S_{21} & 0 & \ldots & L_2S_{2N} \\ 0 & 0 & \ldots & 0 & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & A_{NN} & L_NS_{N1} & L_NS_{N2} & \ldots & 0 \\ \hline R_1 & 0 & \ldots & 0 & -I_1 & 0 & \ldots & 0 \\ 0 & R_2 & \ldots & 0 & 0 & -I_2 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & R_N & 0 & 0 & \ldots & -I_N \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ \ldots \\ x_N \\ \hline y_1 \\ y_2 \\ \ldots \\ y_N \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ \ldots \\ b_N \\ \hline 0 \\ 0 \\ \ldots \\ 0 \end{array}\right]$$

On note bien que ce système est plus creux que le système précédent grâce à l'ajout des variables auxiliaires. De plus, la présence des nombreux zéros permet de former le complément de Schur de la matrice du système aisément. Avec une élimination par lignes, on obtient le système

$$\begin{bmatrix} E_{11} & S_{12} & \ldots & S_{1N} \\ S_{21} & E_{22} & \ldots & S_{2N} \\ \ldots & \ldots & \ldots & \ldots \\ S_{N1} & S_{N2} & \ldots & E_{NN} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{bmatrix} = \begin{bmatrix} C_1b_1 \\ C_2b_2 \\ \ldots \\ C_Nb_N \end{bmatrix}$$

où $E_{ii} = (R_i A_{ii}^{-1} L_i)^{-1}$,$C_i = E_i R_i A_{ii}^{-1}$. On peut alors à l'aide du système précédent, trouver l'inverse de la matrice A en fonction des matrices L_i,S_{ij},R_j et les A_{ii}. En effet, le système augmenté et le sytème précédent donnent les relations suivantes, en notant $\tilde{A} = A_{ii}$,

$$\tilde{A}x + LSy = b \tag{1.8}$$

$$(E + S)y = Cb \tag{1.9}$$

La relation (1.9) est équivalente à

$$Sy = Cb - Ey$$

En injectant dans (1.8) et en composant par $\tilde{A}^{-1}$

$$x = \tilde{A}^{-1}(I - LC)b + \tilde{A}^{-1}LEy$$

Or d'après ce qui précède on a aussi $y = (E + S)^{-1}b$ et en posant alors $D = \tilde{A}^{-1}(I - LC)$ et $B = \tilde{A}^{-1}LE$ on obtient en remplaçant y dans l'expression précédente

$$A^{-1} = D + B(E + S)^{-1}C \tag{1.10}$$

On note de plus que les matrices D et B sont diagonales par blocs.

Remarque 1.3.1. *Lors de l'implémentation de l'algorithme, on doit initialement calculer les inverses des blocs diagonaux ce qui représente $\mathcal{O}(Nm^3)$. L'inversion du complément de Schur $(E+S)$ requiert quant à lui $\mathcal{O}(N^3k^3)$ opérations. Le reste des calculs est dominé par le calcul des x_i une fois les y_j connus ce qui nécessite $\mathcal{O}(N^2k^2 + Nmk^2 + Nm^2)$ opérations.*

Remarque 1.3.2. *Le solveur de Martinsson et Rokhlin proposé dans [12] est basé sur la formule d'inversion de la matrice A.*

Ceci décrit un schéma direct rapide à un seul niveau. Cela suppose de plus que les blocs de rangs faibles aient été arrangés d'une certaine façon au préalable, typiquement à l'aide d'un *octree*, suivant la géométrie du domaine. On peut adapter cet algorithme aux blocs eux-mêmes afin de construire un solveur multi-niveaux. L'avantage d'un solveur multi-niveaux (comme le sont les $\mathcal{H}$-matrices et la méthode des multipôles rapides, FMM) est de réduire drastiquement la complexité du solveur et de pouvoir viser une complexité en $\mathcal{O}(N \log N)$, tout comme l'aspect *divide and conquer* de l'algorithme *Quicksort* permet de faire passer la complexité d'un algorithme de tri de $\mathcal{O}(N^2)$ à $\mathcal{O}(N \log N)$.

1.3.2 Utilisation de la compression LSR pour la construction du solveur

On a vu ci-dessus que chaque bloc A_{ij} de la matrice A doit être compressé afin de pouvoir appliquer la formule (1.10) et résoudre le problème linéaire. Cependant, pour que la nouvelle formulation matricielle puisse être utilisée, on doit obtenir des matrices de permutation L_i et R_j qui soient respectivement communes aux mêmes blocs de la ligne "i" et de la colonne "j". On peut obtenir une telle propriété en considérant les blocs suivants :

$$A_i = [A_{i1}A_{i2}A_{i(i-1)} \dots A_{i(i+1)} \dots A_{iN}A_{1i}^T A_{2i}^T A_{(i-1)i}^T \dots A_{(i+1)i}^T \dots A_{Ni}^T],$$

de taille $n_i \times 2(N_{tot} - n_i)$. On applique alors directement l'algorithme de Cheng *et al.* sur les lignes de chacune de ces concaténations pour obtenir à la fois L_i et R_j grâce à l'ajout des transposées qui permet de rendre symétrique la factorisation au sens où $R = L^*$ avec le défaut de détériorer le taux de compression(*cf* [6]).

Remarque sur l'obtention de A_i

La concaténation précédente ne tient pas compte de l'éloignement des blocs. Or les interactions lointaines ne devraient pas induire de rangs élevés pour les blocs matriciels correspondants. Le principe est le même que pour la méthode des multipôles où l'on distingue pour une cellule donnée, ses voisins immédiats de ses voisins lointains. Pour une cellule $\mathcal{C}_i$ fixée, on construit une boîte $\mathcal{B}_i$ englobant cette cellule et ses voisins directs. Les éléments contenus dans cette boîte sont traités de manière usuelle tandis que les éléments restants sont considérés comme éloignés ("bien séparés" suivant la terminologie de la FMM) et peuvent faire l'objet d'un traitement particulier.

L'article [8] ayant servi de support à l'exposé de cette méthode est suffisamment récent pour ne pas être beaucoup cité dans la littérature. Par conséquent, le traitement de la concaténation des blocs "éloignés" n'apparaît pas de manière suffisamment explicite pour que l'on ait investit du temps à développer cette méthode. Dans le chapitre suivant, on évoque une méthode différente, les $\mathcal{H}$-matrices, qui sous sa forme mono-niveau sera plus approfondie d'un point de vue numérique à partir de la seconde partie.

Chapitre 2

Description du solveur de Shaeffer

Ce cours chapitre est un résumé de la méthode employée par Shaeffer dans [13] et [14] afin de construire un solveur direct rapide pour une méthode d'équations intégrales.

2.1 Introduction

Les grands problèmes matriciels posent de nombreux problèmes, notamment pour leur résolution directe. La méthode exhibée ici est une version simple et mono-niveau de la méthode dite des $\mathcal{H}$-matrices initialement développée par Hackbush et Bebendorf. On renvoie aux sources [3],[2] et [5] pour une introduction à cette méthode. Bien que destinée initialement à la construction de préconditionneurs pour des solveurs itératifs, cette méthode permet de construire des solveurs directs. Cependant, nous n'avons pas trouvé de références bibliographiques décrivant une telle construction. On a alors choisi de s'intéresser puis d'implémenter la méthode proposée par Shaeffer (*cf*[13]) puisqu'elle est plus documentée et que des résultats encourageants ont déjà été publiés. On propose dans ce chapitre, un court résumé de la méthode étudiée en renvoyant aux sources pertinentes et aux chapitres ultérieurs.

2.2 Regroupement des inconnues par pavage

Comme dans la méthode proposée par Greengard, on doit traiter les interactions proches et lointaines entre les sources. Contrairement à l'usage d'un *octree* utilisé dans [8], la démarche adoptée par Shaeffer dans [13] et [14] est d'utiliser un algorithme produisant des groupes de degrés de liberté tels que le bloc matriciel correspondant à l'interaction de groupes éloignés aura un rang faible. Les

blocs diagonaux correspondront donc à "l'auto-interaction" des groupes d'inconnues. Les groupes sont construits suivant une direction privilégiée reliant les points extrêmes, du minimum au maxmimum. Chaque groupe est alors numéroté en fonction de l'itération à laquelle il est construit, le premier groupe aura le numéro 1, le second le 2 et ainsi de suite jusqu'à N_{blocs}. On note G_I le I^{eme} groupe formé par l'algorithme, on désigne par A_{IJ} l'interaction entre les groupes G_I et G_J. Puisque la numérotation dépend de la distance au point minimum on montre alors que

$$\forall I, J, K \in \{1, \ldots, N_{blocs}\} : I \leq J \leq K, \qquad rg(A_{IK}) \leq rg(A_{IJ}).$$

Dès lors, en dehors de la diagonale, le rang diminuera puisque les blocs correspondront à des groupes de plus en plus éloignés. En l'absence de regroupements, le rang des blocs matriciels n'a pas cette propriété de décroissance et ceci met à mal l'algorithme de compression utilisé par la suite. Cette découpe est considérée au chapitre 6 consacré au regroupement des degrés de liberté. On mentionne que cette découpe n'est pas celle employée par Bebendorf dans [3] pour séparer les degrés de liberté.

2.3 Compression par algorithme *ACA*

La répartition des degrés de liberté effectuée au préalable permet d'obtenir des blocs matriciels de rangs faibles. Algébriquement, cela correspond à une redondance des informations contenues dans le bloc et on peut supposer que, en notant r le rang du bloc, seulement r lignes ou colonnes seront nécessaires à la construction du bloc. L'algorithme ACA développé par Bebendorf dans [2] et [3] permet d'expliciter ces colonnes et ces lignes afin de construire une approximation du bloc à une tolérance ϵ donnée. L'algorithme est énoncé dans le chapitre 4, et montre que l'on peut obtenir la décomposition d'une matrice de taille $m \times n$ et de rang k sous la forme suivante,

$$A \simeq U.V^T, \tag{2.1}$$

où $U \in \mathbb{K}^{m \times k}$ et $V \in \mathbb{K}^{n \times k}$.
Ainsi la décomposition (2.1) permet de ne stocker que $k(m+n)$ coefficients au lieu de mn dans le cas usuel. Un faible rang k produira des résultats d'autant plus appréciables puisque le nombre de coefficients à stocker sera moindre. En effet, la matrice sera stockée sous la forme (2.1) et nous n'effectuerons pas le produit UV^T !

2.4 Opérations matricielles élémentaires compressées

La décomposition (2.1) énoncée ci-dessus permet d'amoindrir la capacité de stockage nécessaire pour la manipulation de matrices dont les blocs sont de bas

rangs. Cette décomposition permet également d'effectuer des opérations telles que la somme ou le produit de blocs matriciels en diminuant le nombre d'opérations. Ceci est le but du chapitre 5 qui aborde de manière complète les opérations avec de telles matrices. On signale de plus qu'une introduction complète et exhaustive du sujet se trouve dans [3].
Sans exposer ici tous les détails, on remarque que dans le simple cas d'un produit de matrice par un vecteur, on peut gagner sur le nombre d'opérations. En effet, en considérant la matrice A de la section précédente et un vecteur x de $\mathbb{K}^n$, le produit matrice-vecteur devient

$$\begin{aligned} A.x &= (U.V^T).x \\ &= U(V^T.x) \\ &= U.y, \end{aligned}$$

où $y = V^T.x \in \mathbb{K}^k$. Dans le cas usuel, on devrait effectuer mn opérations élémentaires tandis que la forme compressée ne nécessite que $k(m+n)$ opérations.
Les opérations impliquant ce type de matrice font intervenir des opérations comme la factorisation QR ou la décomposition SVD et sont très facilement implémentables sur machine.

2.5 Applications à une factorisation de Crout par blocs

Hormis l'étape de regroupement des données, les autres étapes mentionnées ci-dessus sont en réalité communes à la méthode des $\mathcal{H} - matrices$ décrite dans [3]. Shaeffer propose d'utiliser ces matrices pour construire un solveur direct contrairement à Bebendorf et Hackbush lesquels mentionnent explicitement la possibilité d'une telle manipulation mais ne l'utilisent cependant que pour la construction de préconditionneurs.
La méthode proposée par Shaeffer reprend l'usage d'une factorisation LU d'une matrice découpée en blocs (*cf* [4]) en remplaçant simplement chaque bloc de la matrice par son approximation de rang faible déterminée avecc l'algorithme ACA. Ceci constitue le développement du chapitre 7.
Notre but est de se comparer à cette méthode et d'en reproduire les résultats qu'il obtient, à savoir factoriser une matrice avec une complexité de l'ordre de $\mathcal{O}(N^{2.4})$.

Troisième partie

Matrices de rangs faibles : obtention & applications

Chapitre 3

Définitions et notions élémentaires

On introduit dans ce chapitre les notions et définitions utilisées tout au long de la méthode. La méthode utilisée dans toute la suite est basée sur l'emploi de matrices de rangs faibles. Par définition, ces matrices contiennent beaucoup moins d'informations essentielles à stocker car on peut exhiber de nomreuses combinaisons linéaires entre les colonnes et /ou les lignes. Dans un soucis de complétude, on rappelle brièvement des définitions et des résultats d'algèbre linéaire utiles par la suite (*cf* [7],[10]) puis on définit alors la notion de matrice de rang faible de la même manière qu'en [3], à partir d'une décomposition sous forme d'une somme de produits tensoriels. Nous verrons que de telles matrices sont très utiles en pratique afin de réduire la quantité d'espace disque utilisée ainsi que le nombre de calculs. Cependant, la détermination de la décomposition par l'algorithme proposé en [2] et [13] fera l'objet du chapitre suivant.

3.1 Définitions & résultats d'algèbre linéaire

3.1.1 Rang & rang numérique d'une matrice

Rang

L'algorithme ACA ainsi que la méthode développée dans les chapitres suivants se base sur la notion de rang d'une matrice. On peut définir le rang de plusieurs manières convenables la plus courante étant la suivante,

Définition 3.1.1 (rang d'une matrice)**.** *Soit $A \in \mathbb{C}^{m\times n}$,le rang de A ($rg(A)$) est défini par*

$$rg(A) \quad = \quad dim(Im(A)),$$

où $Im(A)$ est l'espace image de A défini par

$$Im(A) \quad = \quad \{y \in \mathbb{C}^m : \exists x \in \mathbb{C}^n, y = Ax\}.$$

Dans la pratique, les erreurs d'arrondis et l'incertitude liée aux données font de la détermination du rang un problème non trivial. On travaillera alors avec une tolérance ϵ donnée et on considérera la notion de rang numérique défini par

Définition 3.1.2 ($\epsilon - rang$)**.** *Soient un réel $\epsilon > 0$ et une matrice $A \in \mathbb{C}^{m\times n}$. On appelle $\epsilon - rang$, noté rg_ϵ, la quantité définie par*

$$rg_\epsilon(A) \quad = \quad \min_{\|A-B\|_2 \leq 2} rg(B).$$

Dans toute la suite, le rang d'une matrice fera allusion à un rang numérique même si aucune précision n'est donnée. Ainsi, si l'on peut déterminer les coefficients d'une matrice avec une précision ϵ, on s'intéressera tout naturellement à la quantité $rg_\epsilon(A)$. On considérera qu'une matrice telle que

$$rg_\epsilon(A) \quad < \quad \min(m,n),$$

avec $\epsilon = \mathbf{u}\|A\|_2$,$\mathbf{u}$ étant la précision machine, sera considérée comme déficiente en terme de rang. La relation entre le rang numérique et la déficience d'une matrice est aussi liée à la décomposition en valeurs singulières puisque les valeurs singulières fournissent des renseignements sur la distance à une matrice donnée d'une matrice de rang moins élevé.

Décomposition en valeurs singulères

On fournit ici le théorème d'existence d'une décomposition particulière, liée au rang d'une matrice et abondamment commentée dans [7].

Théorème 3.1.1. *Soit $A \in \mathbb{C}^{m\times n}$, alors il existe des matrices unitaires $\mathcal{U}$ et $\mathcal{V}$,*

$$\begin{aligned} \mathcal{U} &= [u_1, \ldots, u_m], \\ \mathcal{V} &= [v_1, \ldots, v_n], \end{aligned}$$

telles que

$$\mathcal{U}^H A \mathcal{V} \quad = \quad \Sigma = diag(\sigma_1, \ldots, \sigma_p),$$

avec $p = \min(m,n)$ et $\sigma_1 \geq \ldots \geq \sigma_p \geq 0$, ^{H} désignant la transposée du complexe conjugué.

On notera plus souvent la décomposition en valeurs singulières (SVD) sous la forme suivante

$$A \quad = \quad \mathcal{U}\Sigma\mathcal{V}^H.$$

Dans le cas où $m \geq n$, on dispose du théorème suivant, caractérisant la SVD réduite.

Théorème 3.1.2. *Soit $A = \mathcal{U}\Sigma\mathcal{V}^H$ la décomposition SVD de A et supposons de plus que $m \geq n$. Alors*

$$A = \mathcal{U}_1\Sigma_1\mathcal{V}^H,$$

où

$$\begin{aligned}\mathcal{U}_1 &= \mathcal{U}(:, 1:n) = [u_1, \ldots, u_n] \in \mathbb{C}^{m\times n},\\ \Sigma_1 &= \Sigma(1:n, 1:n) = diag(\sigma_1, \ldots, \sigma_n) \in \mathbb{C}^{n\times n}.\end{aligned}$$

Considérons une matrice $A \in {}^{m\times n}$ de rang exactement r, $r \leq \min(m, n)$. On s'intéresse au problème de l'approcher de la façon la plus précise possible par une matrice de rang k, $k < r$. On peut alors utiliser le théorème (3.1.1) afin de répondre à cette question.

Théorème 3.1.3. *Soit la décomposition SVD de A donnée par le théorème (3.1.1). On conserve les mêmes notations et pour tout entier k,$k \leq r = rg(A)$, on pose*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^H.$$

On a alors l'égalité suivante, liant A, son rang r et la matrice A_k,

$$\min_{rg(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

Ce théorème signifie que la plus petite valeur singulière de A est la distance en norme 2 de A à l'ensemble des matrices dont le rang est déficient. De plus, on a l'inégalité suivante

$$\begin{aligned}\sigma_1 \geq \ldots \geq \sigma_{r_\epsilon} &> \epsilon \geq \sigma_{r_\epsilon+1} \geq \ldots \geq \sigma_p,\\ p &= \min(m, n).\end{aligned}$$

Dans le cas où l'on dispose de cette décomposition SVD, et d'une tolérance ϵ, on peut grâce à cette inégalité, déterminer le rang r_ϵ de la matrice A et sa meilleure approximation de rang r_ϵ.

On peut trouver dans la littérature(*cf* [7]), plusieurs algorithmes pour former la décompostion SVD (par ex. ceux de Golub-Reinsch et le R-SVD) qui ont tous en commun le fait de nécessiter un grand nombre d'opérations de l'ordre de $\mathcal{O}(N^3)$ ($N = n, n \simeq m$), ce qui le rend presque inutilisable pour de grandes applications numériques.

L'idée du développement ultérieur est de trouver une méthode permettant une approximation efficace et basée sur des idées similaires. Ceci fera l'objet du chapitre suivant avec l'algorithme ACA.

3.1.2 Autres notions importantes

On énonce ici des résultats utiles pour la suite, notamment pour la manipulation de matrices de rangs faibles. Outre la décomposition SVD, il peut être utile d'obtenir la factorisation QR d'une matrice A.

Théorème 3.1.4 (Factorisation QR)**.** *On considère une matrice $A \in \mathbb{C}^{m\times n}$ et on suppose que $m \geq n$. La factorisation QR de A est donnée par*

$$A = QR,$$

avec Q est une matrice unitaire et R est une matrice triangulaire supérieure. On effectue aouvent la réduction suivante

$$\begin{aligned} A &= QR \\ &= [Q_1; Q_2].[R_1; 0]^T \\ &= Q_1.R_1, \end{aligned}$$

dite factorisation QR réduite de A.

Notons que plusieurs méthodes sont disponibles pour parvenir à cette décomposition telles que celles de Givens ou de Householder. Le coût de cette factorisation est typiquement de l'ordre de $\mathcal{O}(n^3)$.
Une autre décomposition matricielle utile est la décomposition $A = LU$ qui permet de décomposer la matrice en un produit de matrices triangulaires inférieures et supérieures. Cela se révèle utile dans la pratique afin d'inverser un système linéaire puisqu'un système linéaire triangulaire est aisé à inverser. C'est exactement cette décomposition qui sera implémentée par la suite, de manière combinée à l'algorithme ACA.

Théorème 3.1.5 (Factorisation LU)**.** *Une matrice $A \in \mathbb{C}^{n\times n}$ posssède une factorisation $A = LU$ si ses mineurs principaux sont non nuls. Dans le cas où A est non singulière alors cette décomposition est unique et on a*

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

Si de plus A est symétrique, alors la factorisation est dite factorisation de Crout et s'écrit

$$A = LDL^T,$$

avec D une matrice diagonale.

Des algorithmes pour déterminer ces factorisations peuvent être trouvés facilement dans [7] ou [10]. Ces derniers possèdent aussi une factorisation dont la complexité est de l'ordre de $\mathcal{O}(N^3)$. On signale que ces factorisations existent pour des matrices définies par blocs et que l'emploi de telles matrices favorisent les calculs lors d'applications numériques de grandes tailles. La factorisation de Crout par blocs sera étudiée dans la suite afin de construire notre solveur.

3.2 Matrices de rangs faibles

3.2.1 Représentation efficace

Puisque parmi les n colonnes d'une matrice $A \in \mathbb{C}_k^{m\times n}$ seulement k sont nécessaires pour représenter la totalité des coefficients de la matrice, on peut stocker la matrice A autrement que de manière totale, les coefficients superflus pouvant être omis. On note alors par $\mathbb{C}_k^{m\times n}$, l'ensemble des matrices ayant au plus k colonnes (ou lignes) linéairement indépendantes

$$\mathbb{C}_k^{m\times n} := \{A \in \mathbb{C}^{m\times n} : \mathrm{rg(A)} \leq k\}.$$

Remarquons que $\mathbb{C}_k^{m\times n}$ n'est pas un espace vectoriel. En effet, le rang d'une somme de deux matrices chacune de rang k est en général inférieur à $2k$. Une propriété néanmoins importante est que chaque sous-bloc d'une telle matrice est de rang inférieur ou égal à k.

Théorème 3.2.1. *Une matrice $A \in \mathbb{C}^{m\times n}$ appartient à $\mathbb{C}_k^{m\times n}$ si et seulement si il existe deux matrices $U_A \in \mathbb{C}^{m\times k}$ et $V_A \in \mathbb{C}^{n\times k}$ telles que*

$$A = U_A V_A^H \tag{3.1}$$

Cette représentation (3.2.1) est appelée représentation compressée ou somme de produits tensoriels. En effet, si l'on note u_i et v_i pour $i = 1, \ldots, k$ les colonnes respectives de U_A et V_A, alors la représentation (3.2.1) est équivalente à

$$A = \sum_{i=1}^{k} u_i v_i^H.$$

Ainsi, au lieu de stocker les mn coeffcients de la matrice A, on se contente des $k(m+n)$ coefficients des vecteurs u_i et v_i. On évalue le gain en mémoire par la quantité suivante.

Définition 3.2.1 (Taux de compression)**.** *Soit $A \in \mathbb{C}^{m\times n}$ une matrice de rang k donnée par sa représentation tensorielle $U_A V_A^H$. On appelle taux de compression la quantité ,notée $dBrf$, définie par*

$$dBrf \quad = \quad -10.log_{10}\left(\frac{k.(m+n)}{m.n}\right)$$

Outre le fait de réduire l'espace mémoire utilisé, cette représentation facilite aussi le produit matrice-vecteur

$$\begin{aligned} Ax &= (U_A V_A^H)x \\ &= U_A(V_A^H)x. \end{aligned}$$

La mise à jour $y := y + Ax$ n'est pas effectuée éléments par éléments, on multiplie A par x en deux étapes

1. $z := V_A^H x \in \mathbb{C}^n$,

2. $y := U_A z$,

ce qui ne nécessite que $2k(m+n) - k$ opérations au lieu des $2mn$ opérations usuelles. On note cependant que cette représentation peut ne pas être efficace ! Supposons que $m = n$ et que la matrice considérée est de rang maximal,alors la représentation tensorielle amène à manipuler $2n^3$ coefficients soit deux fois plus que sous sa forme usuelle. On caractérise alors les matrices de faible rang à l'aide de la définition suivante.

Définition 3.2.2 (Matrice de rang faible)**.** *Une matrice $A \in \mathbb{C}_k^{m\times n}$ est appelée matrice de rang faible si et seulement si*

$$k(m+n) << mn.$$

Par abus de langage, on parlera aussi de matrice compressée. On notera toujours une telle matrice sous sa forme tensorielle(ou compressée) $U_A V_A^H$.

Calcul de normes Typiquement calculées en tant que critère d'arrêt d'un algorithme, les normes matricielles sont d'autant plus facile d'accès avec des matrices compressées. Ainsi, la norme de Fobénius d'une matrice compressée de rang k peut être calculée en $2k^2(m+n)$ opérations en remarquant que

$$\|U_A V_A^H\|_F^2 \quad = \quad \sum_{i,j=1}^{k} (u_i^H u_j)(v_i^H v_j),$$

tandis que la norme spectrale, donnée par

$$\|A\|_2 = \sqrt{\rho(V_A U_A^H U_A V_A^H)} = \sqrt{\rho(U_A^H U_A V_A^H V_A)}.$$

où $\rho(A)$ désigne le rayon spectral de la matrice A, peut être calculée en $\mathcal{O}(k^2(m+n))$ opérations en calculant les matrices $U_A^H U_A$ et $V_A^H V_A$ de tailles $k \times k$ puis la plus grande valeur propre du produit. On note que cela est avantageux si l'on vérifie l'inégalité $k^2(m+n) << mn$ ce qui est une condition bien plus restrictive que la définition de faible rang précédente.

Décomposition orthonormale Enfin, il peut parfois être utile que les colonnes de U_A et V_A soient orthonormales. Dans ce cas, on doit introduire une matrice $X \in \mathbb{R}^{k \times k}$ de coefficients et remplacer la décomposition sous forme tensorielle par une décomposition de la forme

$$A = U_A X V_A^H$$

Dans ce cas, le calcul de la norme de Frobénius se simplifie de la façon suivante

$$\begin{aligned} \|U_A X V_A^H\|_F &= \|X\|_F \\ &= \sqrt{\sum_{i,j=1}^{k} |x_{ij}|^2}. \end{aligned}$$

Ceci ne requiert que $\mathcal{O}(k^2)$ opérations. La norme spectrale vérifie quant à elle la relationn suivante

$$\|U_A X V_A^H\|_2 = \|X\|_2,$$

menant au calcul de la plus grande valeur propre d'une matrice de taille $k \times k$.

Chapitre 4

Construction de la représentation efficace : algorithme *ACA*

4.1 Première approche de la décomposition tensorielle

On remarque que la forme compressée d'une matrice de rang faible pourrait être obtenue presque directement à l'aide d'une décomposition en valeurs singulières de la matrice. Cependant, cette décomposition nécessite $\mathcal{O}(N^3)$ opérations pour une matrice de taille N ce qui est particulièrement couteux (on renvoit le lecteur à la référence [7] pour de plus amples détails sur la décomposition SVD). Le but de cette partie est de montrer une approche plus pratique avec "les mains", aidant à la compréhension de la méthode que l'on va développer par la suite.
Considérons une matrice $A \in \mathbb{C}^{m\times n}$ de rang k. Le théorème 3.2.1 du chapitre précédent nous garantit l'existence de la décomposition

$$A = U_A V_A^H,$$

mais ne nous renseigne pas sur la façon de l'obtenir.

Considérons une matrice $A \in \mathbb{C}^{m\times n}$ et le scalaire γ_1 défini par

$$\gamma_1 = \max_{i\in\{1,..,m\},j\in\{1,..,n\}} |A_{ij}|,$$

et notons i_1, j_1 les indices réalisant le maximum. Posons également

$$\begin{aligned} u_1 &= A(:, j_1) \\ v_1 &= \frac{1}{\gamma_1} A(i_1, :) \end{aligned}$$

qui représentent respectivement la j_1^{eme} colonne et la i_1^{eme} ligne (au coefficient γ_1 près) de la matrice A. Le coefficient (k,l) de la matrice $u_1v_1 \in \mathbb{C}^{m\times n}$ s'écrit donc

$$(u_1.v_1)_{kl} = (u_1)_k.(v_1)_l.$$

En particulier, les éléments de la j_1^{eme} colonne et de la i_1^{eme} ligne sont reconstruits de manière exacte :

$$\begin{aligned}(u_1v_1)_{j_1l} &= (u_1)_{j_1}.(v_1)_l \\ &= \gamma_1.\frac{A_{j_1l}}{\gamma_1} \\ &= A_{j_1l},\end{aligned}$$

$$\begin{aligned}(u_1v_1)_{ki_1} &= (u_1)_k.(v_1)_{i_l} \\ &= A_{ki_1}.\frac{A_{i_1j_1}}{\gamma_1} \\ &= A_{ki_1}\end{aligned}$$

Les autres coefficients sont eux différents des coefficients d'origine. Posons à présent $A^{(0)} = A$ et $A^{(1)} = A^{(0)} - u_1v_1$, alors la ligne i_1 et la colonne j_1 de la matrice $A^{(1)}$ sont nulles. Le même procédé peut alors être appliqué à la matrice $A^{(1)}$, on construit γ_2,u_2 et v_2 de manière similaire.

On remarque que si γ_{k+1} est nul à la $(k+1)^{eme}$ itération c'est donc que $A^{(k)}$ est identiquement nulle ce qui se traduit par

$$A^{(k-1)} = u_kv_k,$$

ce qui de proche en proche est équivalent à

$$A = \sum_{i=1}^{k} u_iv_i.$$

qui est exactement la décomposition tensorielle attendue d'après le théorème 3.2.1 et la matrice A est alors de rang k. D'autre part, si une matrice $A \in \mathbb{C}^{m\times n}$ est de rang k, alors il existe des matrices inversibles P et Q telles que A soit équivalente à la matrice $J_r \in \mathbb{C}^{m\times n}$ définie par

$$J_r = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}.$$

L'application à cette matrice de k itérations de la méthode proposée conduit aisément à une matrice identiquement nulle d'où la conclusion.

Cette méthode présente néanmoins quelques défauts d'un point de vue numérique. Le coût du calcul du maximum qui requiert tout d'abord N^2 puis $(N-1)^2, (N-2)^2, \ldots$ opérations, soit une complexité cubique en N ! On remarque de plus que la totalité de la matrice est nécessaire afin d'en calculer le maximum ce qui peut s'avèrer coûteux si chaque coefficient doit être calculé de manière approchée au préalable. Pourtant, si elle est de rang faible, il doit exister un moyen astucieux de ne pas calculer d'informations redondantes. Cette méthode ne paraît donc pas plus avantageuse qu'une décomposition SVD dans le cas d'un rang élevé mais son principe est très similaire à l'algorithme *ACA*(*Adaptative Cross Approximation*), développé par M. Bebendorf et étudié dans [3],[2],[15],[11].

4.2 Version analytique de l'*ACA*

On présente l'algorithme *ACA* sous sa forme analytique tel qu'il est développé dans [2] et [11]. Le contexte d'utilisation ainsi que le théorème de convergence établis dans [2] sont retranscrits ici Nous ne discuterons cependant pas les démonstrations des théorèmes, le but n'étant pas de reproduire le travail effectué dans [2], mais de disposer d'un cadre théorique stable et puissant pour la compression de matrices de faibles rangs.

4.2.1 Contexte d'utilisation

On rappelle que l'équation intégrale s'écrit de la façon suivante

$$\mathcal{A}[u](y) = \int_\Gamma K(x,y)u(x)dF_x = f(y), y \in \Gamma,$$

où Γ est la frontière du domaine de calcul, $K : \Gamma \times \Gamma \to \mathbb{R}$ le noyau et $u, f : \Gamma \to \mathbb{R}$ sont respectivement l'inconnue et le second membre du problème. On s'intéresse dans notre cas à l'application d'un schéma de Galerkin afin de discrétiser l'équation intégrale ci-dessus. On discrétise la surface Γ par $\Gamma_h = \bigcup_{j=1}^n \Gamma_j$, on considère alors un ensemble de fonctions de base $\{\phi_j, j = 1, \ldots, n\}, Supp(\phi_j) \subset \Gamma_h$. Les coefficients a_{ij} de la matrice A du problème discrétisé sont donnés par

$$a_{ij} = \int_\Gamma \int_\Gamma K(x,y)\phi_j(x)\psi_i(y)dF_x dF_y.$$

pour un ensemble de fonctions test $\{\psi_i, i = 1, \ldots, n\}, Supp(\psi_i) \subset \Gamma_h$. On présentera cependant la méthode dans le cas de la méthode de Nyström dont les coefficients sont donnés par

$$a_{ij} = K(x_j, y_i),$$

afin de simplifier la présentation ainsi que les énoncés. Les résultats présentés dans ce cas demeurent valables pour les schémas de collocation et de Galerkin. C'est d'ailleurs dans cet ordre que l'algorithme a été mis au point (*cf* [11]).

Si $K(x,y)$ est lisse et défini partout dans l'espace, sa série de Taylor en la variable y en y^* peut s'écrire

$$K(x,y) \quad = \quad \sum_{i=1}^{r} v_i(x)u_i(y) + R_r(x,y), \tag{4.1}$$

où $u_k(y) = (y-y^*)^\alpha$,$v_k(x) = \frac{1}{\alpha!}\partial_y^\alpha K(x,y^*)$ et $\alpha = (\alpha_1,\alpha_2,\alpha_3)$ correspond à l'index de sommation dans 4.1.

D'un point de vue numérique, étant donnée une tolérance $\epsilon > 0$, la manipulation de la décomposition (4.1) ne peut être envisagée autrement qu'en la tronquant au rang (terme) $r = r(\epsilon)$(le rang étant en fait l'ϵ-rang), de sorte que

$$|R_r(x,y)| \quad \leq \quad \epsilon|K(x,y)|, \forall x, y.$$

Il découle le résultat suivant sur l'approximation $\tilde{A}$ de la matrice de discrétisation A

$$\|A - \tilde{A}\|_F \leq \epsilon \|A\|_F.$$

Malheureusement, la plupart des noyaux rencontrés ne possèdent pas autant de régularité et ne sont pas lisses dans tout l'espace. Cependant, la singularité du noyau est souvent localisée pour les points "diagonaux" $x \to y$ et ils sont lisses ailleurs. Cela nous amène à la définition suivante.

Définition 4.2.1 (Fonction asymptotiquement lisse)**.** *Soit $K(x,y) : D_X \times D_Y \to \mathbb{R}$ une fonction de deux variables. On dit que K est asymptotiquement lisse par rapport à y si $K(x,.) \in \mathcal{C}^\infty(\mathbb{R}^3 - \{x\})$ pour tout $x \in \mathbb{R}^3$ et qu'il existe une constante $g < 0$ telle que pour tout multi-indice $\alpha \in \mathbb{N}_0^3$*

$$|\partial_y^\alpha K(x,y)| \quad \leq \quad c_p|x-y|^{g-p}, p = |\alpha|. \tag{4.2}$$

où c_p dépend uniquement de p.

La proposition suivante, issue de [2] relie cette définition à un critère géométrique afin de prouver l'existence d'une décomposition similaire à (4.1).

Proposition 4.2.1. *Si une fonction asymptotiquement lisse vérifie le critère géométrique*

$$diam(D_Y) \quad \leq \quad \eta.dist(D_X, D_Y), \eta < 1,$$

alors elle possède une décomposition de la forme (4.1).

Cette propriété est capitale en ce sens où elle signifie que le noyau peut être considéré comme dégénéré, *ie* il s'exprime comme une somme finie de produits de fonctions d'une seule variable

$$K(x,y) \simeq \sum_{i=1}^{r} u_i(x)v_i(y),$$

sous réserve de pouvoir négliger le résidu $R_r(x,y)$. Si tel est le cas, on pourra alors produire un algorithme générant une approximation de $K(x,y)$ qui de plus pourra se discrétiser et s'appliquer à la matrice discrétisée A.

Remarque 4.2.1. *La découpe de la matrice de discrétisation A suivant l'application de ce critère aux inconnues du problème permet d'obtenir une matrice dont les blocs pourront chacun être approché par une somme finie semblable à la relation (4.3). Par exemple, le noyau de Green dans l'espace,* $K(x,y) = \frac{1}{4\pi}\frac{1}{|x-y|}$ *vérifie cette propriété.*

Remarque 4.2.2. *Notons que dans le cas des* $\mathcal{H}$*-matrices, il s'agit du critère utilisé pour renuméroter les inconnues afin qu'avec cette renumérotation, les blocs de la matrice A satisfassent aux hypothèses de la compression ACA. On utilisera dans nos applications numériques un autre critère géométrique afin d'assurer la séparation des groupes (cf chapitre sur le regroupement des degrés de libertés).*

On présente la version analytique de l'algorithme *ACA* visant à construire une décomposition similaire à (4.1) pour une fonction $K(x,y) : D_X \times D_Y \to \mathbb{R}$:

$$K(x,y) = K(x,[y]_k)^T G K([x]_k,y) + R_k(x,y)$$

où $G \in \mathbb{R}^{k\times k}$, $x_{i_l} \in D_X$, $y_{j_l} \in D_Y$ avec la notation suivante

$$K(x,[y]_k) = \begin{bmatrix} K(x,y_{j_1}) \\ \vdots \\ K(x,y_{j_k}) \end{bmatrix},$$

$$K([x]_k,y) = \begin{bmatrix} K(x_{i_1},y) \\ \vdots \\ K(x_{i_k},y) \end{bmatrix}.$$

4.2.2 Algorithme

Soient deux domaines non vides D_X,D_Y et une fonction $K : D_X \times D_Y \to \mathbb{R}$. On considère de plus deux ensembles non nécessairement finis de points x_i et y_j tels que

$$\{x_1, x_2, \ldots\} \subset D_X \quad , \quad \{y_1, y_2, \ldots\} \subset D_Y.$$

L'algorithme suivant construit une approximation de la fonction $K(x,y)$ pouvant être utilisée pour le schéma de Nyström.

On considère les séquences $\{r_k\}$ et $\{s_k\}$ définies dans [2] comme suit

$$r_0(x,y) = K(x,y), \quad s_0(x,s) = 0$$

et on définit pour $k = 0, 1, \ldots$

$$\begin{aligned} r_{k+1}(x,y) &= r_k(x,y) - \gamma_{k+1} r_k(x, y_{j_k+1}).r_k(x_{i_k+1}, y), & (4.3)\\ s_{k+1}(x,y) &= s_k(x,y) + \gamma_{k+1} r_k(x, y_{j_k+1}).r_k(x_{i_k+1}, y), & (4.4) \end{aligned}$$

avec $\gamma_{k+1} = (r_k(x_{i_k+1}, y_{j_k+1}))^{-1}$ et x_{i_k+1},y_{j_k+1} choisis à chaque étape de façon à ce que $r_k(x_{i_k+1}, y_{j_k+1}) \neq 0$.

On remarque que les fonctions r_k accumulent des zéros ce qui nous permet d'affirmer que les fonctions s_k interpolent petit à petit la fonction $K(x,y)$.

On donne ci-après le théorème de convergence de l'algorithme publié dans [2]. La preuve exhaustive (et technique !) de la convergence de l'algorithme est entièrement présente dans la référence à laquelle on renvoie le lecteur pour de plus amples détails.

Notons par $M_k^{(l)}(x)$ la matrice suivante

$$M_k^{(l)}(x) = \begin{bmatrix} K(x_{i_1}, y_{j_1}) & \ldots & K(x_{i_1}, y_{j_k}) \\ \vdots & & \vdots \\ K(x, y_{j_1}) & \ldots & K(x, y_{j_k}) \\ \vdots & & \vdots \\ K(x_{i_k}, y_{j_1}) & \ldots & K(x_{i_k}, y_{j_k}) \end{bmatrix},$$

où à la l^{eme} ligne, on remarque que l'on a $K(x, [y]_k)$. On pose de plus

$$M_k = M_k^{(l)}(x_{i_l}).$$

Le théorème suivant garantit la convergence de l'algorithme.

Théorème 4.2.2 (Bebendorf)**.** *Soient les points x_{i_k} choisis de telle sorte que*

$$|r_{k-1}(x_{i_k}, y_{j_k})| \geq |r_{k-1}(x, y_{j_k})|, \forall x \in D_X.$$

Alors pour une fonction asymptotiquement lisse K, les fonctions r_k et s_k respectivement définies en (4.3) et (4.4) satisfont la relation

$$K(x,y) = s_{n_p}(x,y) + r_{n_p}(x,y),$$

où pour $x \in D_X$ *et* $y \in D_Y$,

$$s_{n_p}(x,y) \quad = \quad K(x,[y]_{n_p})M_{n_p}^{-1}K([x]_{n_p},y)$$

et

$$|r_{n_p}(x,y)| \quad \leq \quad c_p dist^g(D_X,D_Y)\eta^p$$

où c_p *ne dépend pas de* η *mais seulement des points* $y_1,\ldots,y_{n_p}$.

4.3 Version matricielle de l'*ACA*

La section précédente a prouvé qu'il était possible de décomposer le noyau $K(x,y)$ sous la forme d'une somme de produits de fonctions d'une variable. On désire à présent en tirer un algorithme rapide et efficace pour l'appliquer à des matrices dont le rang est supposé faible puisque générée à partir de fonctions asymptotiquement lisses. En effet, la grande taille des blocs et du système nous pousse à vouloir les compresser un à un afin de gagner en stockage et lors de la multiplication avec un vecteur.

Soit A la matrice de discrétisation du problème. On cherche une approximation S de faible rang r de A à une précision ϵ donnée. On se réfère à [11] pour le descriptif de l'algorithme.

Sous réserve de disposer d'une routine calculant les coefficients de la matrice un à un, on peut construire l'algorithme suivant afin d'effectuer un assemblage à la voée de la matrice et la déterminer directement sous forme compressée.

4.3.1 Algorithme ACA

Définissons la première ligne de l'approximant comme suit

$$S_0 = 0 \qquad i_1 = 1,$$

Pour $k = 0, 1, 2...$, on calule la i_{k+1}^{eme} ligne de la matrice

$$a = A^T e_{i_{k+1}},$$

et celle du résidu ainsi que l'indice de colonne du pivot

$$\begin{aligned} (R_k)^T e_{i_{k+1}} &= a - \sum_{l=1}^{k} (u_k)_{i_{k+1}} v_k, \\ j_{k+1} &= ArgMax|(R_k)_{i_{k+1}j}|. \end{aligned}$$

La valeur du pivot est

$$\gamma_{k+1} = ((R_k)_{i_{k+1}j_{k+1}})^{-1}.$$

On calule alors la j_{k+1}^{eme} colonne de la matrice

$$a = Ae_{j_{k+1}},$$

ainsi que celle du résidu ainsi que le nouvel indice de ligne du pivot

$$\begin{aligned} (R_k)e_{j_{k+1}} &= a - \sum_{l=1}^{k} (v_k)_{j_{k+1}} u_k, \\ i_{k+2} &= ArgMax|(R_k)_{ij_{k+1}}|. \end{aligned}$$

Les nouveaux vecteurs de la décomposition sont donnés par

$$u_{k+1} = \gamma_{k+1} R_k e_{j_{k+1}} \quad , \quad v_{k+1} = R_k^T e_{i_{k+1}},$$

et on met à jour le nouvel approximant par

$$S_{k+1} = S_k + u_{k+1}.v_{k+1}^T.$$

Puisque la matrice originale A n'est pas générée entièrement, seulment la norme de l'approximant S_k peut être utilisée et calculée itérativement de la sorte

$$\|S_{k+1}\|_F^2 = \|S_k\|_F^2 + 2\sum_{j=1}^{k} u_{k+1}^T u_j v_j^T v_{k+1} + \|u_{k+1}\|_F^2 \|v_{k+1}\|_F^2.$$

Un critère d'arrêt adéquat est donc

$$\|u_r\|_F \|v_r\|_F \leq \epsilon. \|S_r\|_F.$$

Dans le cas où on lit la matrice à partir d'un disque, l'algorithme a un coût en $\mathcal{O}(mnr)$ tandis que le bon critère d'arrêt est donné par

$$\|R_r\|_F \leq \epsilon. \|A\|_F.$$

Coût de l'algorithme Pour obtenir S_r, on effectue les opérations suivantes

Evaluations de K	$r(m+n)$
Additions et multiplications	$r(r-1)(m+n)$
Divisions	mr
Total	$\simeq \mathcal{O}(r^2(m+n))$.

L'algorithme requiert donc $\mathcal{O}(r^2(m+n))$ opérations élémentaires pour fournir une approximation de rang r de la matrice A à la précision ϵ.

4.3.2 Remarques

Complétude de l'algorithme On prouve que l'algorithme est toujours fini et que pour une matrice A de rang r, alors seulement r étapes de l'algorithme sont nécessaires pour retrouver le rang de la matrice.

Proposition 4.3.1. *Soit $A \in \mathbb{C}^{m\times n}$ et de rang r. Alors $S_r = A$*

Preuve 3. *Si $r = 1$, il est manifeste que $R_1 = 0$ et $S_1 = A$. Pour un rang supérieur, on applique une étape de l'algorithme. Sans perdre de généralité, on supposera que $a_{11} \neq 0$ et ainsi*

$$A \begin{bmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{m1}}{a_{11}} & & & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & \dots & a_{1n} \\ 0 & & & \\ \vdots & & \hat{A} & \\ 0 & & & \end{bmatrix}.$$

Ceci implique que $rg(\hat{A}) = rg(A) - 1$ mais d'autre part

$$\tilde{A} = A - \frac{1}{a_{11}} A e_1 e_1^T A = \begin{bmatrix} 0 & \dots & \dots & a_{1n} \\ 0 & & & \\ \vdots & & \hat{A} & \\ 0 & & & \end{bmatrix}.$$

Donc $rg(\tilde{A}) = rg(\hat{A}) = rg(A) - 1$.

On note que l'algorithme réduit successivement le rang.

Parallèle avec la factorisation LU On peut considérer que sans pivotement, cet algorithme revient à une factorisation LU de la matrice révélant le rang. On peut trouver une illustration de ce propos dans [15] ainsi qu'en reprenant la première approche développée.

Autres techniques On peut trouver dans la littérature [3],[5] des améliorations de cet algorithme à propos du choix des pivots et des techniques de recompression.

Chapitre 5

Opérations élémentaires sur les matrices de rangs faibles

Le stockage d'une matrice de faible rang A sous sa forme efficace $U_A V_A^H$ permet de gagner en espace mémoire. Cependant, l'utilisation de cette forme permet aussi d'économiser des calculs lors d'opérations matricielles élémentaires comme le produit matrice-vecteur. On présente dans ce chapitre, les opérations courantes effectuées sur les matrices à savoir la multiplication et l'addition. On présente aussi l'obtention d'une décomposition SVD pour les matrices données sous leurs formes compressées laquelle permet d'affiner l'algorithme d'addition. On trouve le descriptif de ces opérations dans [3] ainsi que dans [13].

5.1 Multiplication de matrices compressées

On rappelle que lorsqu'une matrice $A \in \mathbb{C}^{m\times n}$ peut être représentée sous la forme d'une somme de produits tensoriels $U_A V_A^H, U_A \in \mathbb{C}^{m\times k}, V_A \in \mathbb{C}^{n\times k}$ avec $k << m, n$, on dit qu'elle est *compressée*.
On veut obtenir sous forme compressés le produit matriciel $C = AB$ où A est compressée. On distingue alors deux cas suivant la nature de la matrice B en comparant dans chaque cas le nombre d'opérations effectuées dans le cas usuel ainsi que compréssé.

B est non-compressée On effectue le produit en notant $C = U_C V_C^H$ et $AB = U_A V_A^H B$ ce qui matriciellement donne

$$\begin{aligned} U_C V_C^H &= (U_A V_A^H) B \\ U_C V_C^H &= U_A (V_A^H B) \end{aligned}$$

Dans ce cas, on pose simplement $U_C = U_A$ ce qui n'entraîne aucun calcul et le nombre d'opérations repose sur le calcul de $V_C^H = V_A^H.B$ soit (knp) opérations

par rapport à (mnp) dans le cas usuel. Le rapport entre ces deux quantités est

$$\frac{knp}{mnp} = \frac{k}{m} << 1 \quad \text{si} \quad k << m.$$

Le gain de calul est donc du premier ordre en $\frac{k}{m}$.

B est compressée On suppose à présent que les deux matrices A et B sont données sous forme compressée, $B = U_B V_B^H$ où $U_B \in \mathbb{C}^{n\times r}$ et $V_B \in \mathbb{C}^{p\times r}$ avec $r << n, p$. Avec les mêmes notations que précédemment pour les matrices A et C, on a

$$\begin{aligned} U_C V_C^H &= (U_A V_A^H)(U_B V_B^H) \\ &= U_A (V_A^H U_B) V_B^H \\ &= U_A (T) V_B^H. \end{aligned}$$

La matrice intermédiaire $T = V_A^H U_B$ est un produit scalaire avec un nombre réduit d'opérations (knr). On doit alors effectuer le produit de T avec U_A ou V_B^H ce qui respectivement mène à (mkr) ou (rkn) opérations. Finalement, le coût total du produit $C = AB$ est de $\{knr + mkr\}$ ou $\{knr + rkn\}$. Le rapport entre les coûts des méthodes compressée et usuelle est donné par

$$\frac{knr + mkr}{mnp} = \frac{kr(m+n)}{mnp} << 1 \quad \text{si} \quad k, r << m, n, p.$$

Dans le cas où les matrices sont carrées $(m = n = p)$ le rapport est du second ordre

$$\frac{knr + mkr}{mnp} = 2\frac{kr}{m^2} << 1 \quad \text{si} \quad k, r << m.$$

On peut alors énoncer la règle suivante, à utiliser lors de l'emploi de matrices compressées.

Proposition 5.1.1 (Règle de calcul)**.** *Lors de produits matriciels utilisant des matrices compressées, on choisisra l'ordre des multiplications favorisant au possible les produits scalaires par rapport aux produits tensoriels afin de minimiser le nombre d'opérations élémentaires.*

5.2 Décomposition SVD d'une matrice compressée

L'accès à une décomposition en valeurs singulières d'une matrice peut avoir un intérêt lorsque l'on souhaite en extraire l'information principale portée par ses plus grands éléments singuliers. On dispose dans le cas de matrice compréssée d'une méthode peu coûteuse afin de construire une décomposition SVD.

On considère une matrice compressée donnée par sa représentation efficace $A = U_A V_A^H$ avec $U_A \in \mathbb{C}^{m\times k}, V_A \in \mathbb{C}^{n\times k}$ et $k << \min(m,n)$. On souhaite obtenir la décomposition suivante

$$U_A V_A^H = \mathcal{U}\Sigma\mathcal{V}^H,$$

avec $\mathcal{U} \in \mathbb{C}^{m\times k}$,$\mathcal{V} \in \mathbb{C}^{n\times k}$ et $\Sigma \in \mathbb{C}^{k\times k}$ diagonale.

Calculons les factorisations QR de U_A et V_A

$$\begin{aligned} U_A &= Q_U R_U, \\ V_A &= Q_V R_V. \end{aligned}$$

On forme alors le produit $R = R_U.R_V^T$ en utilisant le fait que $k << \min(m,n)$. En effet, dans les décompositions QR ci-dessus, cette inégalité implique que les $m_A - k$ et les $n_A - l$ dernières colonnes respectives de R_U et R_V sont nulles. Ainsi le calcul de R ne nécessite que la connaissance des k premières colonnes de R_U et R_V.

On peut alors effectuer une décomposition en valeurs singulières "usuelle" de la matrice réduite R,

$$R = \hat{\mathcal{U}}\Sigma\hat{\mathcal{V}}^H,$$

où $\hat{\mathcal{U}} \in \mathbb{C}^{k\times k}$,$\hat{\mathcal{V}} \in \mathbb{C}^{k\times k}$ et $\Sigma \in \mathbb{C}^{k\times k}$.

Puisque les matrices Q_U et Q_V sont orthogonales, les matrices $\mathcal{U}$ et $\mathcal{V}$ définies par

$$\begin{aligned} \mathcal{U} &= Q_U\hat{\mathcal{U}}, \\ \mathcal{V} &= Q_V\hat{\mathcal{V}}, \end{aligned}$$

sont aussi orthogonales et on obtient bien la décomposition SVD voulue.

Coût de la décomposition Le détail des opérations effectuées à chaque étape de la construction précédente est donc

Décomposition QR de U_A et V_A	$4k^2(m+n) - \frac{8}{3}k^3$
Calcul de R	$\frac{2}{3}k^3 + \frac{11}{6}k^2 - \frac{1}{3}k$
Décomposition SVD de $R_U R_V^H$	$22k^3$
Calcul de $\mathcal{U}$ et $\mathcal{V}$	$k(2k-1)(m+n)$
Total	$\simeq 6k^2(m+n) + 20k^3$

Remarque 5.2.1. *Dans tout ce qui précède, on s'est placé dans le cas compressé où $k << m, n$. Cependant, on peut adapter les calculs à une représentation sous forme tensorielle dont les dimensions ne vérifient pas la précédente relation.*

5.3 Addition de matrices compressées

On note pour toute cette partie :

$$\begin{aligned} A &= U_A V_A^H \\ B &= U_B V_B^H \\ C &= U_C V_C^H \end{aligned}$$

avec $U_A \in \mathbb{C}^{m\times k_A}, V_A \in \mathbb{C}^{n\times k_A}, U_B \in \mathbb{C}^{m\times k_B}, V_B \in \mathbb{C}^{n\times k_B}, U_C \in \mathbb{C}^{m\times k}, V_C \in \mathbb{C}^{n\times k}$ et $k_A, k_B \leq \min(m,n), k \leq k_A + k_B$.

Première approche On peut effectuer l'addition $C = A + B$ en construisant les concaténations suivantes

$$\begin{aligned} U_C &= [U_A; U_B], \\ V_C &= [V_A; V_B]. \end{aligned}$$

le produits $U_C V_C^H$ fournissant bien la somme $A + B$ sans effectuer aucun calcul. Cependant, on ne possède pas de borne inférieure sur le rang d'une somme de matrices. Ainsi, il est possible (on verra dans les applications numériques que cela est bien le cas) que le rang k de la somme soit nettement inférieur à $k_A + k_B$.

On dispose fort heureusement d'un moyen économique pour parvenir à déterminer le rang(et éventuellement le réduire) de la somme à partir des concaténations. Cette diminution du rang permet donc de réduire l'espace mémoire nécessaire au stockage de la somme et par là même de réduire le nombre d'opérations élémentaires lors de calculs impliquant cette somme.

Addition approchée *É*tant données les matrices concaténées $U_C \in \mathbb{C}^{n\times(k_A+k_B)}$ et $V_C \in \mathbb{C}^{n\times(k_A+k_B)}$ du paragraphe précédent, on peut construire une approximation de rang plus faible de la somme $C = A + B$ à l'aide de la décomposition SVD vue auparavant en $\mathcal{O}((k_A + k_B)^3)$ opérations.

Théorème 5.3.1. *Soient $A \in \mathbb{C}_{k_A}^{m\times n}$, $B \in \mathbb{C}_{k_B}^{m\times n}$ et $k \in \mathbb{N}$ avec $k \leq k_A + k_B$. Alors il existe une matrice $S \in \mathbb{C}_{k}^{m\times n}$ vérifiant*

$$\|A + B - S\| \quad = \quad \min_{M\in\mathbb{C}_k^{m\times n}} \|A + B - M\|$$

par rapport à toute norme matricielle peut être calculée en $6(k_A + k_B)^2(m + n) + 20(k_A + k_B)^3$ opérations.

Il s'agit là d'une application directe de la décomposition SVD vue à la section précédente à la représentation efficace $U_C = [U_A; U_B], V_C = [V_A; V_B]$ en tronquant la SVD aux k premières valeurs singulières.

Remarque 5.3.1. *On peut être amené lors de certaines applications à calculer une somme $A = A_1+A_2+\ldots+A_N, A_i \in \mathbb{C}_{k_i}^{m\times n}, i = 1 \ldots N$ de matrices compressées pour lesquelles la concaténation amènerait un coefficient $(\sum_{i=1}^{N} k_i)^2$ devant $(m+n)$. Pour éviter cette situation, on effectuera les additions deux à deux ce qui réduit le nombre d'opérations à*

$$6\sum_{i=1}^{N-1}(k_i + k_{i+1})^2(m + n) + 20\sum_{i=1}^{N-1}(k_i + k_{i+1})^3,$$

au détriment de la meilleure approximation.

Autres opérations D'autres opérations sont possibles telle que l'agglomération de matrices de rangs faibles. Nous renvoyons le lecteur au livre de M. Bebendorf, [3], pour de plus amples détails.

Chapitre 6

Regroupement des degrés de libertés

La résolution d'un système linéaire revient par définition à traiter une matrice de rang maximal. Il est par contre courant que les inconnues du problème(noeuds de discrétisation, degrés de libertés,...) ne soient pas localisées de façon aléatoire et qu'elles soient réparties en plusieurs groupes, d'après un critère géométrique. De ce fait, on doit traiter plusieurs groupes ayant des interactions entre eux. Un exemple en électromagnétisme serait le champ créé par un ensemble de charges au voisinage d'un autre ensemble de charges. Avec une renumérotation adaptée, on peut alors faire apparaître cette structure particulière dans l'écriture de la matrice.

L'écriture par blocs des matrice convient particulièrement à ce cas. En effet, chaque groupe géométrique contenant des inconnues va correspondre à un bloc matriciel. En numérotant les groupes et en renumérotant les inconnues(typiquement un mailleur fournira une certaine numérotation qui n'est pas nécessairement celle voulue) appartenant à ces groupes, on obtient une certaine permutation P laquelle une fois appliquée à la matrice originale A donne une matrice $\tilde{A} = PA$ dont le rang n'a pas été modifié mais dont les blocs matriciels correspondent à l'interaction d'un groupe sur un autre. Dans le cas d'une interaction liée à la distance entre les groupes, on peut alors raisonnablement supposer qu'un bloc représentant l'interaction de groupes éloignés aura un rang d'autant plus faible que les groupes sont distants.
Parmi les méthodes existantes amenant à une telle découpe, on peut rappeler la construction d'*octree* qui sont utilisés dans la méthode FMM ou par exemple dans [8]. On présente ici une autre méthode proposée dans [13] et qui a l'avantage d'être simple à implémenter puisque mono-niveau par opposition aux *octree* qui sont eux multi-niveaux.

6.1 Algorithme de pavage, *Cobblestone sorting technique*

6.1.1 Algorithme

Considérons N points répartis dans l'espace $\mathbb{R}^3$. L'algorithme de pavage proposé en [13] regroupe ces points de la façon suivante :

Algorithme de pavage Cobblestone

– Initialisation

1. Trouver les points $V_{min} = \min(x, y, z)$ et $V_{max} = \max(x, y, z)$ par rapport à l'origine.

2. Définir la direction de référence pour le tri $V_{REF} = V_{max} - V_{min}$.

3. Projeter les inconnues sur cette direction de référence.

– Itérer sur les inconnues restantes :

1. Trouver le point dont la projection est la plus petite sur V_{REF} parmi les inconnues non triées. Ce point (V_{start}) est le premier point du groupe courant.

2. Calculer les distances de V_{start} à tous les autres points non triés.

3. Trier ces distances par ordre croissant.
4. Intégrer au groupe courant les points vérifiant le critère $\mathcal{C}$.

Le critère $\mathcal{C}$ peut être de deux natures différentes. La première, ordinale, consiste à sélectionner les n plus proches points $x_1, \ldots, x_n$ de V_{start} ; la seconde consiste à créer des régions sphériques définies comme étant pour chaque groupe l'ensemble des points x vérifiant $\|V_{start} - x\| \leq \eta, \eta > 0$.

6.1.2 Coût de la méthode de pavage

On évalue ici, le coût de la méthode de pavage lorsque l'on fixe le nombre de points contenus dans chaque groupe. Dans le cas où le critère porte sur la distance entre les points, on peut néanmoins se donner une borne maximale sur le nombre de points dans un groupe et effectuer la même démarche.

Notons :
– N, le nombre total de points à regrouper,
– n, le nombre de points contenus dans un groupe,

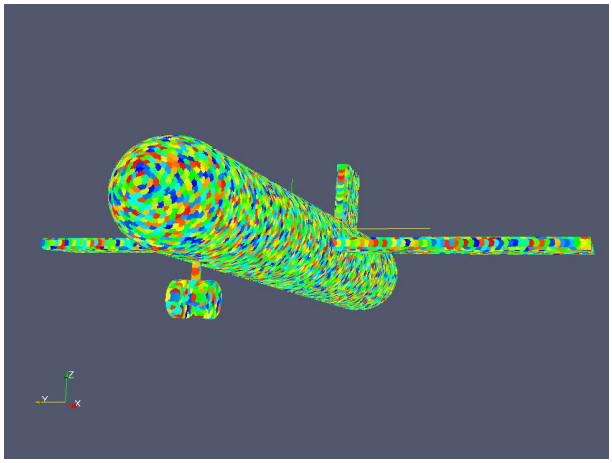

FIGURE 6.1 – Exemple d'utilisation de l'algorithme de pavage sur un modèle simplifié d'avion avec train d'aterrissage ; critère utilisant au plus 3000 éléments par groupe.

- K, le nombre total de groupes voulus,
- $\mathcal{C}_N$, le coût de l'algorithme de pavage.

Proposition 6.1.1 (Coût de l'algorithme)**.** *Sous les hypothèses suivantes :*

1. $n = N^{\alpha}$,*avec* $\alpha \in]0;1[$,
2. $K = \frac{N}{n}$,

le coût $\mathcal{C}_N$ *de l'algorithme de pavage vérifie*

$$\mathcal{C}_N \quad \leq \quad N^{2-\alpha} log(N)$$

Cette borne est très large mais se révèle suffisamment précise pour justifier que cette étape est moins coûteuse que $\mathcal{O}(N^2)$ *pour certaines valeurs du paramètre* α.

Preuve 4. *En effet, l'initialisation revient à déterminer les points formant la direction principale ce qui revient à effectuer un tri de* N *points en* $\mathcal{O}(NlogN)$ *opérations. Les* $N-2$ *points à projeter ajoute* $\mathcal{O}(N)$ *opérations. Dès lors, le nombre d'opérations à chaque itération* $i, i \in \{1, \ldots, K-1\}$ *est principalement dû au tri de* $(N - i.n)$ *points ce qui peut être fait en* $\mathcal{O}((N - i.n)log(N - i.n))$ *opérations. Ainsi, le coût total* $\mathcal{C}_N$ *est donné par*

$$\mathcal{C}_N \simeq \sum_{i=0}^{K-1} (N - i.n)log(N - i.n) + \mathcal{O}(N).$$

On majore alors grossièrement chaque terme de la somme par NlogN ce qui permet d'obtenir l'expression

$$\mathcal{C}_N \leq NKlog(N).$$

D'après les hypothèses précédentes,

$$N = K.n,$$

$$n = N^{\alpha}.$$

La majoration de $\mathcal{C}_N$ *devient*

$$\mathcal{C}_N \leq \mathcal{O}(N^{2-\alpha}logN).$$

6.2 Heuristique pour le choix du nombre d'éléments dans un groupe

Le choix du paramètre α ne doit pas être fait totalement au hasard. Par exemple, pour $\alpha = \frac{1}{4}$, la borne exhibée à la section précédente serait supérieure à N^2 ce qui est inacceptable si l'on désire construire un algorithme en $\mathcal{O}(N^2)$ reposant sur cette découpe ! Plus précisement, en reprenant la borne supérieure déterminée à la section précédente, quelques manipulations fournissent que la complexité est de $\mathcal{O}(N^2)$ pour

$$\alpha = \alpha^* = \frac{log(log(N))}{log(N)}.$$

On choisira alors α de telle sorte qu'il soit compris entre α^* et 1 afin de garder une complexité inférieure à $\mathcal{O}(N^2)$. Outre l'augmentation de la complexité, la réduction du nombre d'éléments par groupe peut aussi déteriorer la compression du bloc par la suite. Pour $N = 1.10^6$ inconnues (ce qui représente un système linéaire plein de taille très grande), $\alpha^* \simeq 0.19006$. On prendra alors soin de choisir une valeur suffisamment éloignée de cette valeur afin de ne pas détériorer la complexité.
On choisira pour nos futures applications

$$\alpha = \frac{1}{2},$$

et le nombres d'éléments n contenus dans un groupe par

$$n = \beta N^{\frac{1}{2}},$$

avec β de l'odre de l'unité.

Chapitre 7

Application de l'algorithme *ACA* à une factorisation de type LU

L'algorithme de pavage développé ci-dessus nous permet de trouver une renumérotation efficace des degrés de liberté assurant de plus que les blocs extra-diagonaux sont de rangs faibles. Dès lors, l'algorithme *ACA* étudié auparavant nous garantit de pouvoir construire en un temps raisonnable une décomposition en somme de produits tensoriels pour chacun des blocs de la matrice renumérotée. Enfin, on dispose d'opérations matricielles adaptées à cette classe de matrices. On termine alors notre étude théorique par la construction d'un algorithme de factorisation de type *LU* exploitant au mieux les propriétés des matrices compressées. On commence par exhiber une factorisation de Crout par blocs bien adaptée à la mise en place d'un code informatique, puis on l'exprime dans le cas de blocs compressés.

7.1 Factorisation de Crout par blocs

Considérons une matrice A symétrique par blocs et non singulière dont les bloc diagonaux sont non singuliers également. On aimerait disposer d'un algorithme de factorisation possédant les caractéristiques suivantes :

1. le stockage sur disque de la factorisation est optimal, *ie* on ne stocke qu'un nombre minimal de coefficients nécessaires à la reconstruction de la matrice,
2. un bloc matriciel peut être lu/écrit sur disque mais ne doit pas pouvoir être modifié une fois écrit (lecture seule),
3. la factorisation doit tenir compte de la symétrie de la matrice.

Commentons ces points en gardant à l'esprit que l'on veut un solveur direct rapide ! Le premier point vise tout simplement à ne pas utiliser plus de mémoire que

nécessaire. Le deuxième point est en réalité le plus imortant. En effet, si l'on prévoit de traiter des blocs compressibles, on ne connait *a priori* pas le rang de ce bloc. Supposons qu'il est initialement écrit dans une zone mémoire bien précise. Considérons une mise à jour impliquant une addition avec un autre bloc de rang faible, le rang de la somme (même approchée) ne sera vraisemblablement pas le même ! Ainsi, dans le cas où il serait supérieur, on devrait écrire dans une zone mémoire qui n'est pas allouée ! Imposer une seule écriture par bloc règle ce problème de dépassement de mémoire. Enfin, le dernier point est naturel en ce sens où l'on ne veut pas répéter des calculs déjà effectués !

Exemple de factorisation en petite dimension

L'écriture suivante de la factorisation de Crout $A = Z^H D Z$ se prête alors à toutes ces exigences. On peut trouver ce développement dans [14] ou [10]. Considérons l'exemple suivant où l'on décompose la matrice A de la manière suivante,

$$\begin{bmatrix} Z_{11} & 0 & 0 \\ Z_{12}^H & Z_{22} & 0 \\ Z_{13}^H & Z_{23}^H & Z_{33} \end{bmatrix} \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{bmatrix} \begin{bmatrix} Z_{11} & Z_{12} & Z_{13} \\ 0 & Z_{22} & Z_{23} \\ 0 & 0 & Z_{33} \end{bmatrix}$$

Cette décomposition existe bien comme nous l'avons mentionné dans les rappels en début de second partie, avec Z triangulaire par blocs supérieure et

$$Z_{ii} = I,$$

I étant la matrice de l'identité. Le produit complet donne par ailleurs

$$\begin{bmatrix} Z_{11}D_{11}Z_{11} & Z_{11}D_{11}Z_{12} & Z_{11}D_{11}Z_{13} \\ Z_{12}^H D_{11}Z_{11} & Z_{12}^H D_{11}Z_{12} + Z_{22}D_{22}Z_{22} & Z_{12}^H D_{22}Z_{23} + Z_{22}D_{22}Z_{23} \\ Z_{13}^H D_{11}Z_{11} & Z_{13}^H D_{11}Z_{12} + Z_{23}^H D_{22}Z_{22} & Z_{13}^H D_{11}Z_{13} + Z_{23}^H D_{22}Z_{23} + Z_{33}D_{33}Z_{33} \end{bmatrix}.$$

L'équation matricielle associée au bloc $(1,1)$ est équivalent à

$$A_{11} = Z_{11}D_{11}Z_{11},$$

soit encore

$$D_{11} = A_{11}.$$

Passons à l'équation fournie par le bloc $(2,1)$,

$$A_{12}^H = Z_{12}^H D_{11} Z_{11},$$

soit compte tenu des résultats déjà obtenus,

$$Z_{12}^H = A_{12}^H D_{11}^{-1}.$$

On procède alors de même pour $j = 2, 3$ avec les blocs $(j,i), j \leq i$. On établit alors la formule générale suivante.

Cas général

Proposition 7.1.1. *Soit A une matrice découpée en $N \times N$ blocs, symétrique par blocs, non singulière et dont les blocs diagonaux sont inversibles. Alors la factorisation $A = Z^H D Z$ définie comme suit satisfait aux exigences 2 et 3.*

Pour $j = 1, \ldots, N$, les blocs diagonaux sont mis à jours par la formule

$$D_{jj} = A_{jj} - \sum_{p=1}^{j-1} Z_{pj}^H D_{pp} Z_{pj}. \tag{7.1}$$

Pour $i = j+1, \ldots, N$, les autres blocs sont mis à jour par la formule

$$Z_{ji}^H = D_{jj}^{-1} \left(A_{ji}^H - \sum_{1}^{j-1} Z_{pi}^H D_{pp} Z_{pj} \right). \tag{7.2}$$

Les blocs diagonaux de Z_{jj} étant des matrices de l'identité, on peut stocker les matrices D_{jj} à la place pour gagner en espace mémoire, satisfaisant ainsi toutes les exigences précédentes.

Résolution du système linéaire La résolution du système linéaire $AJ = V$, ou encore $Z^H D Z J = V$ se fait alors de manière usuelle par substitutions avant et arrière avec les formules respectives pour $j \in \{1, \ldots, N\}$,

$$X_j = V_j - \sum_{p=1}^{j-1} Z_{jp}^H D_{pp}^{-1} X_p, \text{Avant} \tag{7.3}$$

$$J_j = D_{jj}^{-1} \left(X_j - \sum_{p=j-1}^{1} Z_{jp} J_p \right), \text{Arrière} \tag{7.4}$$

7.2 Version par blocs compressés de la factorisation de Crout

On montre finalement dans cette partie comment construire un solveur direct rapide pour un système plein. En effet, ayant à disposition l'algorithme ACA pour compresser les blocs extra-diagonaux, on note le résultat de la compression d'un bloc Z_{ij} de la façon suivante

$$Z_{ij} = (U_Z V_Z^H)_{ij}^H.$$

En remarquant que $Z_{ij}^H = (U_Z V_Z^H)_{ij}^H = (V_Z U_Z^H)_{ij}$, on obtient simplement, par substitution, à partir de (7.1)

$$D_{jj} = A_{jj} - \sum_{p=1}^{j-1} (V_Z U_Z^H)_{pj} D_{pp} (U_Z V_Z^H)_{pj},$$

soit

$$D_{jj} = A_{jj} - \bigoplus_{p=1}^{j-1} \{(V_Z U_Z^H)_{pj} \otimes D_{pp} \otimes (U_Z V_Z^H)_{pj}\},$$

où $\bigoplus$ désigne la somme approchée -en sommant les termes deux à deux- pour les matrices compressées, tandis que $\otimes$ désigne la multiplication favorisant les produit scalaires de matrices compressées. Le bloc D_{jj} étant diagonal, il est de rang maximal et la soustraction est à considérer comme la soustraction usuelle. Ce bloc est évidemment stocké de manière usuelle coefficient par coefficient.

De même on effectuera la mise à jour (7.2) de la façon suivante

$$(V_Z U_Z^H)_{ji} = D_{jj}^{-1} \otimes \left((V_A U_A^H)_{ji} \oplus .(-1) \bigoplus_{1}^{j-1} (V_Z U_Z^H)_{pi} \otimes D_{pp} \otimes (U_Z V_Z^H)_{pj} \right).$$

Enfin, selon les mêmes règles, les relations (7.3) et (7.4) deviennent respectivement

$$X_j = V_j - \bigoplus_{p=1}^{j-1} (V_Z U_Z^H)_{jp} \otimes D_{pp}^{-1} \otimes X_p, \text{Avant} \tag{7.5}$$

$$J_j = D_{jj}^{-1} \left(X_j - \bigoplus_{p=j-1}^{1} (U_Z V_Z^H)_{jp} \otimes J_p \right), \text{Arrière} \tag{7.6}$$

ce qui achève la construction de la factorisation compressée.

Remarque 7.2.1. *On souligne le fait que cette manipulation fournit des résultats très spectaculaires dans [13], néanmoins nous n'avons pas trouvé de démonstration visant à prouver que si A peut être compressée par blocs, alors A^{-1} peut l'être aussi !*

Remarque 7.2.2. *On peut faire varier la précision de la factorisation à travers la tolérance servant à calculer les sommes compressées. Il s'agit d'une façon d'obtenir de manière économique un préconditionneur. En effet, une tolérance grossière mène à ne sélectionner que les plus grandes valeurs sigulières (cf SVD d'une matrice compressée) et ainsi réduire drastiquement le rang des blocs compressés au détriment d'une précision moindre. Cependant, l'utilisation en tant que préconditionneur est très bonne et largement commentée dans [4] et [3].*

Remarque 7.2.3. *On peut faire aussi varier la précision de l'algorithme de compression ACA ce qui va augmenter ou diminuer l'erreur relative commise sur l'approximation des blocs de la matrice. Dans le cas où l'on souhaite résoudre le système linéaire de façon directe, il paraît naturel de choisir les deux tolérances mises en jeu égales.*

Bibliographie

[1] T Abboud and I Terrasse. *Modélisation des phénomènes de propagation d'ondes*. Cours de l'Ecole Polytechnique. Ecole Polytechnique, 2007.

[2] M Bebendorf. Approximation of boundary element matrices. *Numer.Math.*, 2000.

[3] M Bebendorf. *Hierarchical Matrices*. Springer, 2000.

[4] M Bebendorf. Hierarchical lu decomposition-based preconditionners for bem. *Computing*, 2004.

[5] M Bebendorf and S Kunis. Recompression techniques for adaptative cross approximation. *Journal of Integral Equations and Applications*, 2009.

[6] H. Cheng, Z. Gimbutas, P.G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.*, 26, 2005.

[7] G Golub and C van Loan. *Matrix Computations*. Johns Hopkins, third edition edition, 1996.

[8] L. Greengard, D. Gueyffier, P.G. Martinsson, and V. Rokhlin. Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numerica*, 2009.

[9] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 1987.

[10] P Lascaux and R Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingéneur*. Masson, 1986.

[11] Sergej Rjasanow. Adaptative cross approximation of dense matrices. mai 2002. presented at the Int. Association Boundary Element Methods Conf., IABEM 2002, Austin, TX.

[12] V. Rokhlin and P.G. Martinsson. A fast direct solver for scattering problem involving elongated structures. *Journal of Computational Physics*, 2007.

[13] John Shaeffer. Direct solve of electrically large integral equations for problems sizes to 1m unknowns. *IEEE Transactions on Antennas and Propagation*, 56-8, 2008.

[14] John Shaeffer. Direct solve of electrically large integral equations for problems sizes to 1m unknowns. September 2008. NASA/CR-2008-215353,National Aeronautics and Space Administration.

[15] Kezhong Zhao, Marinos N. Vouvakis, and Jin-Fa Lee. The adaptative cross approximation algorithm for accelerated method of moments computations of emc problems. *IEEE Transactions on Electromagnetic Compatibility*, 47-4, 2005.